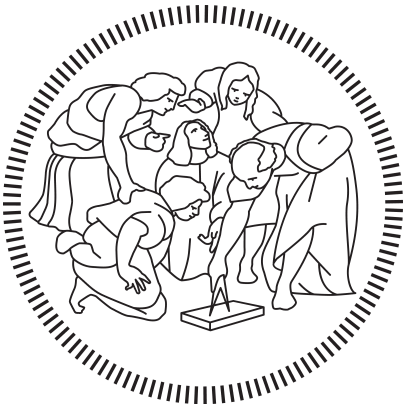
AY 2023/2024



Politecnico di Milano

Internet of Things Homework

Rishabh Tiwari - rishabh.tiwari@mail.polimi.it - 10987397

Professor
Redondi Alessandro Enrico Cesare

Version 1.1
May 14, 2024

1 IoT system design

You are required to design an IoT system to monitor the status of the production process in a small indoor bacterial cellulose factory. The factory is operated in a small university lab (100 sqm) and has about 20 bacterial cellulose growing basins, which must be monitored continuously to ensure the growing process is successful. The main parameters to be monitored are luminosity (2 bytes), content of sugar (2 bytes), and pH of the growing solution (1 byte). Growing bacterial cellulose is a slow process, with growing cycles of about 14 days. Monitoring cycles of 1 hour are needed to allow for changing the environmental parameters for an optimal process.

1. **Propose an overall design for the system, mainly focusing on the communication technology to be used. Motivate your choice**

   Solution:

   - **System Architecture:** The proposed system consists of three layers: Edge Layer, Network Layer, and Cloud Layer.
     - **Edge Layer:**
       20 sensor nodes, each consisting of:
       * A microcontroller (e.g., Arduino or ESP32) to collect and process data from sensors.
       * Three sensors:
         · Luminosity sensor (e.g., BH1750FVI) to measure light intensity.
         · Sugar content sensor (e.g., YSI 2900D) to measure sugar concentration.
         · pH sensor (e.g., pH-450C) to measure the pH level of the growing solution.
       * Power supply (e.g., battery or USB connection).
       * Optional: local data processing capabilities for real-time threshold checks or anomaly detection.
     - **Network Layer:**
       * Wireless communication technology: IEEE 802.15.4 (Zigbee) protocol.
       * 1-2 routers (depending on the lab layout) to extend the network coverage and ensure reliable communication.
       * 1 coordinator node (connected to the Cloud Layer) to manage the network and forward data to the cloud.
       * Network topology: star topology as the default communication method, with mesh capabilities as a fallback or enhancement for reliability and range.
       * CoAP (Constrained Application Protocol) server will be used on the coordinator node to facilitate communication between the sensor nodes and the cloud. CoAP is a lightweight, RESTful protocol suitable for constrained networks and devices.
     - **Cloud Layer:**
       * Cloud-based IoT platform (e.g., AWS IoT, Microsoft Azure IoT, or Google Cloud IoT Core) to store, process, and analyze data.
       * Data analytics and visualization tools (e.g., dashboards, graphs, and alerts) to provide insights into the production process.
       * Data backup and redundancy strategies:
         · Data replication across multiple geographical regions to prevent data loss.
         · Regular backups to ensure data integrity.
       * HTTP server will be used to provide a web-based interface for users to interact with the system, view real-time data, and receive alerts.
   - **Communication Technology:**
     The proposed communication technology is IEEE 802.15.4 (Zigbee) protocol, which is a low-power, low-data-rate wireless communication standard suitable for IoT applications.

- **Communication Stack:**

  The communication stack will consist of the following layers:

  - **Physical Layer:**
    - The IEEE 802.15.4 standard defines the physical layer for Zigbee, offering a maximum data rate of 250 kbps, which is more than sufficient for the data requirements of this application.
    - Operating in the 2.4 GHz ISM band, it provides a global standard for low-power, low-cost, and reliable communication.
  - **Data Link Layer:**
    - The MAC (Media Access Control) sublayer handles the organization of data frames and medium access, ensuring reliable transmission with mechanisms for addressing, channel access, and error checking.
    - Uses CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) to minimize collisions and ensure efficient use of the communication channel.
  - **Network Layer:**
    - Zigbee's network layer supports star, tree, and mesh topologies, with this design primarily using a star topology.
    - The coordinator node manages the network, routes data, and ensures devices can join and leave the network seamlessly.
    - Mesh networking capabilities can be employed as needed for enhanced reliability and coverage.
  - **Transport Layer:**
    - CoAP (Constrained Application Protocol) will be used at the transport layer to provide lightweight, reliable communication between the sensor nodes and the cloud.
    - CoAP operates over UDP, making it suitable for constrained environments while supporting RESTful interactions, which are easy to implement and integrate with web services.
  - **Application Layer:**
    - The application layer will utilize CoAP to facilitate communication with the cloud-based IoT platform.
    - It will handle tasks such as sensor data collection, command and control messaging, and firmware updates.
    - CoAP's simplicity and efficiency make it well-suited for resource-constrained devices and networks.

- **Reasons for Choosing Zigbee:**
  - **Low power consumption:** Zigbee devices can operate for years on a single battery, making them suitable for battery-powered sensor nodes.
  - **Low cost:** Zigbee modules are relatively inexpensive compared to other wireless communication technologies.
  - **Reliability:** Zigbee is a mesh networking technology, which means that each device can act as a router, increasing the network's reliability and coverage.
  - **Low data rate:** The required data rate for this application is relatively low (2 bytes for luminosity, 2 bytes for sugar content, and 1 byte for pH), making Zigbee a suitable choice.
  - **Short-range communication:** Zigbee is designed for short-range communication (up to 100 meters), which is sufficient for a small lab environment.
  - **Scalability:** Zigbee networks can easily scale by adding more nodes without significant changes to the network infrastructure.
  - **Interoperability:** Zigbee's adherence to the IEEE 802.15.4 standard ensures compatibility with a wide range of devices and manufacturers.

- **Data Transmission:**
  - Each sensor node will transmit data to the coordinator node every 1 hour.
  - The coordinator node will forward the data to the Cloud Layer.
  - Data transmission will be done using a star topology, with mesh capabilities as a fallback or enhancement for reliability and range.

- **Data Format:**
  Each data packet will contain the following information:
  - Node ID (1 byte)
  - Luminosity value (2 bytes)
  - Sugar content value (2 bytes)
  - pH value (1 byte)
  - Timestamp (4 bytes)
- **Security:**
  - Data encryption will be implemented using AES-128 to ensure data confidentiality and integrity.
  - Authentication will be implemented using a secure authentication protocol (e.g., TLS or DTLS) to ensure that only authorized devices can join the network and transmit data.
- **Power Management:**
  - Sensor nodes will be powered using batteries, which will be replaced or recharged as needed.
  - Power management techniques (e.g., sleep modes and duty cycling) will be implemented to minimize power consumption and extend the battery life.
- **Environmental Considerations:**
  - The hardware will be protected from potential corrosive or humid conditions typical in a bacterial culture lab by using:
    * Casing for sensors to prevent damage from environmental factors.
    * Industrially rated components to ensure durability and reliability.
- **User Interaction and Alerts:**
  - Users will be able to interact with the system through a user-friendly interface, including:
    * Setting parameters for alerts and notifications.
    * Managing devices remotely.
    * Viewing real-time data and analytics.
  - Alerts will be triggered based on predefined thresholds and anomalies and will be sent to users via email, SMS, or mobile app notifications.
- **Cost Analysis:**
  - Initial setup costs: hardware and software costs for sensor nodes, routers, coordinator node, and cloud infrastructure.
  - Ongoing maintenance costs: battery replacements, software updates, and cloud infrastructure costs.
  - Potential savings: optimized growing conditions can lead to increased yields, reduced energy consumption, and improved product quality.
- **System Maintenance and Updates:**
  - Firmware and software updates will be handled remotely, using secure protocols to prevent security vulnerabilities.
  - Regular system checks and maintenance will be performed to ensure system performance and reliability.
- **Regulatory and Compliance Issues:**
  The system will comply with relevant regulations and standards, including:
  - Wireless communication regulations (e.g., FCC, CE, or TELEC).
  - Data privacy regulations (e.g., GDPR, HIPAA).
  - Industry-specific standards (e.g., ISO, IEC).

2. **Write the pseudo code of the firmware that should be run by the monitoring device installed on each basin**

```
import time
import zigbee
import coap

# Initialize sensors and variables
luminosity_sensor = BH1750FVI()
sugar_content_sensor = YSI2900D()
pH_sensor = pH450C()
node_id = unique_id()
battery_level = 100

# Initialize Zigbee communication module
zigbee_module = zigbee.ZigbeeModule()
zigbee_module.init(node_id)

# Ensure connection to the Zigbee network
def ensure_network_connection():
    if not zigbee_module.ensure_connection():
        handle_network_failure()

ensure_network_connection()

# Initialize CoAP client for communications over Zigbee
coap_client = coap.CoAPClient()
coap_client.init(zigbee_module)

# Security initialization
encryption_key = load_encryption_key()  # Load AES-128 encryption key

# Data collection thresholds
thresholds = {
    'luminosity': 500,
    'sugar_content': 20,
    'pH': 7.0
}

def read_sensors():
    try:
        luminosity = luminosity_sensor.read()
        sugar_content = sugar_content_sensor.read()
        pH_level = pH_sensor.read()
        return luminosity, sugar_content, pH_level
    except SensorError as e:
        log_error("Sensor read error: " + str(e))
        return None, None, None

def send_alert(message, node_id):
    alert_data = {
        'node_id': node_id,
        'alert': message
    }
    alert_data_encrypted = encrypt_data(alert_data, encryption_key)
    coap_client.post('alerts', alert_data_encrypted)

def log_error(message):
    # Log the error to local storage or a secure cloud-based logging service
    print("ERROR: " + message)  # Replace with actual logging implementation

def calculate_power_consumption():
    return 0.05  # Calculate and return power consumption

def current_timestamp():
    return time.time()

def encrypt_data(data, key):
    # Encrypt data using AES-128 with the provided key
    return aes_encrypt(data, key)
```

```
68
69 def handle_network_failure():
70     # Attempt to reconnect to the network
71     while not zigbee_module.ensure_connection():
72         log_error("Network connection failed; attempting to reconnect...")
73         time.sleep(10)  # Wait for 10 seconds before retrying
74     log_error("Network reconnected successfully")
75
76 while True:
77     luminosity, sugar_content, pH_level = read_sensors()
78     if luminosity is None or sugar_content is None or pH_level is None:
79         continue
80
81     # Check and alert if values are below thresholds
82     if luminosity < thresholds['luminosity']:
83         send_alert("Low luminosity", node_id)
84     if sugar_content < thresholds['sugar_content']:
85         send_alert("Low sugar content", node_id)
86     if pH_level < thresholds['pH']:
87         send_alert("Low pH", node_id)
88
89     # Prepare data packet for transmission
90     data_packet = {
91         'node_id': node_id,
92         'luminosity': luminosity,
93         'sugar_content': sugar_content,
94         'pH': pH_level,
95         'timestamp': current_timestamp()
96     }
97
98     # Encrypt data packet before transmission
99     data_packet_encrypted = encrypt_data(data_packet, encryption_key)
100
101    # Send data to coordinator node using CoAP
102    coap_client.post('data', data_packet_encrypted)
103
104    # Manage power usage based on battery status
105    battery_level -= calculate_power_consumption()
106    if battery_level < 20:
107        send_alert("Low battery", node_id)
108
109    # Sleep to save energy
110    time.sleep(3600)  # Sleep for 1 hour
```

3. **As an add-on, you are required to install a VGA camera (640x480 pixels, 8 bits per pixel) to monitor the status of the growing process. Is the solution proposed at the previous points still valid? If not, propose an alternative solution.**

Solution:

- **Why the Previous Solution is Not Valid:**
  - **Bandwidth Limitations of Zigbee:**
    - Zigbee has a maximum data rate of 250 kbps.
    - A VGA camera at 640x480 pixels with 8 bits per pixel produces a significant amount of data per frame, making it impractical to transmit using Zigbee due to its bandwidth constraints.
  - **Power Consumption:**
    - Zigbee is designed for low power consumption, typically around 1 mW in idle mode and up to 100 mW in transmission mode.
    - Continuous high data rate transmission (like image data) would significantly increase power consumption, reducing the battery life of sensor nodes.
  - **System Complexity and Efficiency:**
    - Combining sensor data and image data on Zigbee would lead to network congestion and increased packet loss.

    - Prioritizing critical sensor data over large image data would be challenging, reducing the efficiency of the monitoring system.

- **Alternative Solution and Its Validity:**
  - **High Bandwidth Capability:**
    - Wi-Fi (IEEE 802.11n) supports high data rates, far exceeding the requirements for transmitting VGA camera data.
    - Transmitting large image files is feasible with Wi-Fi, enabling real-time monitoring.
  - **Separation of Data Channels:**
    - Using Wi-Fi for image data and Zigbee for sensor data allows efficient management of data flows.
    - Sensor data continues to be transmitted with low latency and low power via Zigbee, while high-bandwidth image data is handled by Wi-Fi.
  - **MQTT for Efficient Communication:**
    - MQTT is a lightweight messaging protocol with low overhead, making it suitable for IoT applications.
    - MQTT can handle thousands of messages per second, supporting the scalability of the system as more devices or data are added.
  - **Improved Power Management:**
    - While Wi-Fi consumes more power (typically around 1 W during transmission), using it selectively for image data balances overall power consumption.
    - Sensor nodes using Zigbee can remain low-power, extending their battery life.
  - **Scalability and Flexibility:**
    - The dual-communication approach (Wi-Fi for images, Zigbee for sensors) supports scalability as system needs grow.
    - Wi-Fi infrastructure can accommodate additional high-bandwidth peripherals without affecting the performance of the sensor network.
  - **Enhanced Security:**
    - Separate networks allow for tailored security measures. Wi-Fi can use WPA3 encryption for image data, ensuring robust security.
    - Zigbee can continue to use AES-128 encryption for sensor data, maintaining a balance between security and computational overhead.

```
import time
import zigbee
import wifi
import mqtt
import camera

# Initialize sensors and variables
luminosity_sensor = BH1750FVI()
sugar_content_sensor = YSI2900D()
pH_sensor = pH450C()
node_id = unique_id()
battery_level = 100

# Initialize Zigbee communication module for sensor data
zigbee_module = zigbee.ZigbeeModule()
zigbee_module.init(node_id)

# Ensure Zigbee connection
def ensure_zigbee_connection():
    if not zigbee_module.ensure_connection():
        handle_network_failure()

ensure_zigbee_connection()

# Initialize WiFi for camera data transmission
wifi_module = wifi.WiFiModule()
wifi_module.connect('SSID', 'password')

# Initialize MQTT client for image data transmission
```

```
mqtt_client = mqtt.MQTTClient()
mqtt_client.init(wifi_module)

# Initialize VGA camera
camera_module = camera.VGACamera()
camera_module.init()

# Security initialization
encryption_key = load_encryption_key()  # Load AES-128 encryption key

# Data collection thresholds
thresholds = {
    'luminosity': 500,
    'sugar_content': 20,
    'pH': 7.0
}

def read_sensors():
    try:
        luminosity = luminosity_sensor.read()
        sugar_content = sugar_content_sensor.read()
        pH_level = pH_sensor.read()
        return luminosity, sugar_content, pH_level
    except SensorError as e:
        log_error("Sensor read error: " + str(e))
        return None, None, None

def capture_image():
    try:
        image_data = camera_module.capture_image()
        return image_data
    except CameraError as e:
        log_error("Camera error: " + str(e))
        return None

def send_alert(message, node_id):
    alert_data = {
        'node_id': node_id,
        'alert': message
    }
    alert_data_encrypted = encrypt_data(alert_data, encryption_key)
    mqtt_client.publish('alerts', alert_data_encrypted)

def log_error(message):
    # Log the error to local storage or a secure cloud-based logging service
    print("ERROR: " + message)  # Replace with actual logging implementation

def calculate_power_consumption():
    return 0.05  # Calculate and return power consumption

def current_timestamp():
    return time.time()

def encrypt_data(data, key):
    # Encrypt data using AES-128 with the provided key
    return aes_encrypt(data, key)

def handle_network_failure():
    # Attempt to reconnect to the network
    while not zigbee_module.ensure_connection():
        log_error("Network connection failed; attempting to reconnect...")
        time.sleep(10)  # Wait for 10 seconds before retrying
    log_error("Network reconnected successfully")

while True:
    luminosity, sugar_content, pH_level = read_sensors()
    if luminosity is None or sugar_content is None or pH_level is None:
        continue

    # Check and alert if values are below thresholds
    if luminosity < thresholds['luminosity']:
```

```
        send_alert("Low luminosity", node_id)
    if sugar_content < thresholds['sugar_content']:
        send_alert("Low sugar content", node_id)
    if pH_level < thresholds['pH']:
        send_alert("Low pH", node_id)

    # Capture image from VGA camera
    image_data = capture_image()
    if image_data is not None:
        # Encrypt image data
        image_data_encrypted = encrypt_data(image_data, encryption_key)
        # Send image data to cloud-based MQTT broker
        mqtt_client.publish('images', image_data_encrypted)

    # Prepare data packet for transmission
    data_packet = {
        'node_id': node_id,
        'luminosity': luminosity,
        'sugar_content': sugar_content,
        'pH': pH_level,
        'timestamp': current_timestamp()
    }

    # Encrypt data packet before transmission
    data_packet_encrypted = encrypt_data(data_packet, encryption_key)

    # Send data to coordinator node using Zigbee
    zigbee_module.send(data_packet_encrypted)

    # Manage power usage based on battery status
    battery_level -= calculate_power_consumption()
    if battery_level < 20:
        send_alert("Low battery", node_id)

    # Sleep to save energy
    time.sleep(3600)  # Sleep for 1 hour
```

2 Short-range connectivity

A personal area network (PAN) works in IEEE 802.15.4 beacon-enabled mode with CFP only, and with a nominal data rate of 250 [kb/s]. Motes in the network have uplink only traffic towards the PAN with the following distribution: P(r=0 [bit/s])=0.1, P(r=10 [kb/s])=0.3, P(r=20 [kb/s])=0.6. Motes use packets of b = 128 bytes for communication, and each packet fits exactly one slot in the CFP.

1. **What is the beacon interval (BI) in ms?**

   Solution : 102.4

2. **What is the slot time (Ts) in ms?**

   Solution : 4.096

3. **Assuming the maximum duty cycle allowed is 30%, what is the active part of the superframe (Tactive) in ms?**

   Solution : 30.72

4. **How many active slots are there in the CFP Beacon Interval ?**

   Solution : 7

5. **How many inactive slots are there in the CFP Beacon Interval ?**

   Solution : 18

6. **How many motes can join the network?**

   Solution : 3

3 Long-range connectivity

You have setup a weather monitoring station on your balcony and would like to transmit the acquired data over a web service (e.g., ThingSpeak). You have no Wi-Fi connectivity at home, therefore you plan to use a long-range IoT communication technology. After careful consideration, you need to choose between LoRa and NB-IoT

1. **What are the main factors you would look at to make your final choice?**

   Solution :

   (a) **Coverage and Range:**
   - LoRa: Operates on unlicensed frequency bands, providing a range of up to 15 km (9.3 miles) in urban areas and up to 40 km (24.9 miles) in rural areas.
   - NB-IoT: Operates on licensed frequency bands, offering a range of up to 10 km (6.2 miles) in urban areas and up to 20 km (12.4 miles) in rural areas.
   - Consideration: If covering a larger area is necessary, LoRa might be a better choice.

   (b) **Power Consumption:**
   - LoRa: Known for its low power consumption, with devices typically operating for 5-10 years on a single battery.
   - NB-IoT: Also designed for low power consumption, but slightly higher than LoRa, with devices typically operating for 2-5 years on a single battery.
   - Consideration: If power efficiency is crucial, LoRa might be a better fit.

   (c) **Cost:**
   - LoRa: Typically less expensive than NB-IoT, with lower module costs and no need for a cellular subscription.
   - NB-IoT: Requires a cellular subscription, which can increase costs, but offers more reliable connectivity and better QoS (Quality of Service).
   - Consideration: If budget is a constraint, LoRa might be more suitable.

   (d) **Data Rate and Capacity:**
   - LoRa: Offers a lower data rate (up to 50 kbps) and is designed for low-bandwidth applications.
   - NB-IoT: Provides a higher data rate (up to 250 kbps) and is suitable for applications requiring more frequent data transmission.
   - Consideration: If larger amounts of data need to be transmitted or higher data rates are required, NB-IoT might be a better choice.

   (e) **Network Availability and Reliability:**
   - LoRa: Requires a private network infrastructure, which can be set up and managed by the user.
   - NB-IoT: Leverages existing cellular networks, offering better reliability and wider coverage.
   - Consideration: If a more reliable and widely available network is needed, NB-IoT might be a better option.

   (f) **Security:**
   - LoRa: Offers end-to-end encryption and secure authentication mechanisms.
   - NB-IoT: Inherits the security features of cellular networks, including encryption and authentication.
   - Consideration: Both options provide adequate security, but NB-IoT's cellular network security might be more robust.

   (g) **Device Complexity and Integration:**
   - LoRa: Requires more complex device integration and setup, as it operates on a private network.
   - NB-IoT: Offers simpler device integration, as it leverages existing cellular infrastructure.
   - Consideration: If a more straightforward device integration process is preferred, NB-IoT might be a better choice.

   (h) **Penetration and Obstacle Handling:**

- LoRa: Has excellent penetration in urban settings due to its use of sub-gigahertz frequencies, which can help in reaching remote sensors in challenging locations.
- NB-IoT: Benefits from the established infrastructure of cellular networks, which can lead to more consistent coverage within its range, even in urban environments.
- Consideration: If ensuring reliable connectivity in urban or challenging environments is needed, LoRa might be a better fit.

Based on the comparison between LoRa and NB-IoT for long-range IoT communication, LoRa is chosen for the following reasons:

(a) **Coverage and Range:**
- LoRa's excellent coverage and range, including its ability to penetrate urban settings and reach remote sensors, make it well-suited for the weather monitoring station.

(b) **Power Consumption:**
- LoRa's low power consumption is ideal for the station's need for infrequent and low-volume data transmission, which will help extend the battery life of the devices.

(c) **Cost Effectiveness:**
- LoRa's lower module costs and the absence of a need for a cellular subscription make it a more cost-effective option for this application.

(d) **Trade-Offs:**
- While NB-IoT offers better reliability and Quality of Service (QoS), the benefits LoRa provides in terms of coverage, power consumption, and cost make it the preferable choice for this specific application.

2. **You opt to use LoRa, using an open-source gateway close by (e.g., provided by the Things Network). However, your transmission are not successfull. What are the possible causes, and what kind of solutions could be adopted?**

Solution :

Possible Causes

(a) **Insufficient Gateway Coverage**: The open-source gateway might not be close enough or have sufficient coverage to receive the LoRa transmissions effectively.

(b) **Interference from Other Devices and Environment**: Other devices and environmental factors such as trees and weather might be causing interference with the LoRa signals, reducing their strength or corrupting the data.

(c) **Incorrect Configuration**: The LoRa device or the gateway might not be configured correctly, leading to transmission failures.

(d) **Obstacles or Physical Barriers**: Physical obstacles, such as buildings, hills, or dense foliage, might be blocking the LoRa signals, reducing their range or strength.

(e) **Device or Hardware Issues**: The LoRa device or the gateway might have hardware issues, such as faulty antennas or malfunctioning radios.

(f) **Network Congestion**: High traffic on the same LoRa channels can cause delays or blockages in message delivery.

Solutions

(a) **Move the Gateway Closer**: Relocate the open-source gateway to a location that provides better coverage and is closer to the LoRa device.

(b) **Use a Repeater or Range Extender**: Implement a LoRa repeater or range extender to amplify and retransmit the LoRa signals, increasing their range and strength.

(c) **Adjust Device Configuration**: Verify and adjust the configuration of the LoRa device and the gateway to ensure they are set up correctly and compatible with each other.

(d) **Optimize Antenna Placement**: Optimize the placement and orientation of the LoRa device's antenna to minimize the impact of obstacles and physical barriers.

(e) **Replace or Repair Faulty Hardware**: Replace or repair any faulty hardware, such as antennas or radios, to ensure the LoRa device and the gateway are functioning correctly.

(f) **Use a Different Frequency or Spreading Factor**: Experiment with different LoRa frequencies or spreading factors to find a configuration that minimizes interference and improves transmission success, while ensuring compliance with local regulations.

(g) **Implement Error Correction and Retries**: Implement error correction mechanisms, such as forward error correction (FEC), and retries to improve the reliability of the LoRa transmissions.