SCHOOL OF COMPUTER SCIENCE & ENGINEERING
CONTINUOUS ASSESSMENT TEST - II
WINTER SEMESTER 2024-2025

Programme Name & Branch : B.Tech Computer Science and Engineering (Common to all branches)
Course Code and Course Name : BCSE310L IoT Architectures and Protocols
Faculty Name(s) : Prof. Abdul Gaffar H, Prof. Ganesh Shamrao Khekare, Prof. Narayanan Prasanth, Prof. Sendhil Kumar K.S, Prof. Senthilnathan
Class Number(s) : VL2024250501540, 45, 48, 49, 50
Date of Examination : 18/03/2025
Exam Duration : **90 minutes** **Maximum Marks: 50**

General instruction(s):

- **Some answers may differ to some extent as per the assumptions made by students.**

Q. No	Answer Key cum Marking Scheme	M	CO	BL
1.	**Evaluate the challenges of integrating SCADA systems with IoT in industrial automation. Propose a layered framework to mitigate these challenges, considering both hardware and software.**	**10**	**2**	**4**
	Challenges of Integrating SCADA Systems with IoT in Industrial Automation (5 Marks) The integration of Supervisory Control and Data Acquisition (SCADA) systems with the Internet of Things (IoT) in industrial automation presents several challenges across hardware, software, data handling, communication, and security domains. Key challenges include: ➢ Legacy System Compatibility: SCADA systems often use proprietary protocols that are not easily compatible with modern IoT standards. Many legacy systems lack APIs or middleware support for direct IoT integration. ➢ Data Volume Management, Velocity & Interoperability: IoT devices generate massive amounts of data, which traditional SCADA architectures may struggle to process. Different vendors use varying data formats and protocols, creating interoperability issues. ➢ Security & Cyber Threats: Expanding SCADA networks with IoT increases attack surfaces, making them vulnerable to cyber threats. Lack of standardized security frameworks in industrial IoT environments. ➢ Scalability & Network Constraints: SCADA systems were not originally designed for the large-scale, decentralized nature of IoT. Bandwidth limitations and network congestion due to real-time IoT data streams. ➢ Latency, Real Time Processing & Reliability Concerns: IoT devices operating on cloud-based architectures introduce latency, which can be critical in industrial automation. Ensuring real-time processing for time sensitive industrial applications. ➢ Regulatory & Compliance Issues: Industrial sectors must comply with strict regulations, which can hinder rapid IoT adoption. Data sovereignty and governance concerns when integrating cloud-based IoT solutions. ➢ Heterogeneous Communication Protocols: SCADA systems often use legacy protocols like Modbus, DNP3, or OPC, while IoT devices rely on modern protocols like MQTT, CoAP, or HTTP. Bridging these protocols requires middleware or protocol translation, which can introduce latency and complexity.			

Proposed Layered Framework for SCADA-IoT Integration (5 Marks)

To address these challenges, a multi-layered framework can be implemented, combining hardware and software solutions. The framework consists of the following layers for probable mitigation:

1. Perception/Device Layer (Hardware & Sensors)
   - Function: Interface and collects real time data from industrial assets (e.g., PLCs, RTUs, sensors) and IoT devices.
   - Components: Industrial IoT sensors and actuators. SCADA-compatible adapters for legacy systems. Secure edge gateways for initial data filtering.
   - Mitigation: Use protocol converters to bridge legacy SCADA and modern IoT. Implement security-hardened firmware for IoT devices.
2. Data Processing Layer (Pre-processing & Analytics)
   - Function: Processes data at the edge to reduce latency and bandwidth usage. Handle data aggregation, filtering, and analysis
   - Components: Cloud / Edge computing units with AI/ML capabilities. Industrial IoT gateways with protocol translation.
   - Mitigation: Deploy edge AI for real-time anomaly detection. Implement local event-driven processing to reduce cloud dependency. Use edge devices to preprocess data and send only relevant information to the SCADA system. Implement data compression and buffering to manage high data volumes.
3. Communication Layer (Network & Protocols)
   - Function: Ensures seamless and secure communication between SCADA, IoT devices, and cloud.
   - Components: Industrial communication protocols (MQTT, OPC UA, Modbus, VPNs, etc.). Secure transmission via TLS/SSL encryption.
   - Mitigation: Implement network segmentation to isolate critical SCADA systems from IoT networks. Use firewalls and intrusion detection systems (IDS) to monitor and secure traffic. Protocol standardization.
4. Middleware Layer (Data Management & Security)
   - Function: Acts as an integration hub between SCADA, IoT, and enterprise applications. Protect the system from cyber threats
   - Components: Industrial Data Lake / Message Bus (e.g. MQTT Broker). Security services (IAM, Zero Trust Architecture). APIs and SDKs for integrating IoT data with SCADA software. End-to-end encryption for data in transit and at rest.
   - Mitigation: Use standardized data models to ensure compatibility. Develop custom adapters if necessary to bridge specific systems. Implement AI-driven anomaly detection for cybersecurity. Implement anomaly detection systems to identify and respond to threats.
5. Application Layer (SCADA, Cloud, & Enterprise Integration)
   - Function: Provides user interface for insights, visualization, and decision-making capabilities.
   - Components: SCADA dashboards, ERP, and cloud-based AI analytics. API-based integration with enterprise IT systems.
   - Mitigation: Use hybrid cloud models for scalable and compliant data processing. Develop custom dashboards for predictive maintenance for industrial assets.

	6. Management Layer • Objective: Monitor and manage the entire system for performance, scalability, and reliability. • Components: Centralized management platforms for SCADA and IoT devices. Automated provisioning and configuration tools. • Mitigation: Use AI-driven tools for predictive maintenance and system optimization. Implement redundancy and failover mechanisms to ensure system availability. Integrating SCADA systems with IoT in industrial automation is complex but achievable with a well-designed layered framework. By addressing challenges related to communication, data processing, interoperability, and security, this framework ensures a robust and scalable integration. The key is to balance the real-time requirements of SCADA with the scalability and flexibility of IoT, while maintaining stringent security standards to protect critical infrastructure.			
2.	**Big Data Analytics plays a crucial role in IoT. How would the absence of Big Data processing impact decision making in a smart city IoT deployment? Propose an alternative solution to compensate for this absence.**	**10**	**2**	**3**
	Impact of the Absence of Big Data Processing in Smart City IoT Deployments (5 Marks) Big Data Analytics (BDA) is essential in smart city IoT deployments, as it enables real-time decision-making, predictive analytics, and data-driven policy formulation. Without BDA, the following issues arise: 1. Delayed Decision-Making: Lack of real-time insights prevents proactive responses in areas like traffic management, emergency services, and pollution control. 2. Inefficient Resource Allocation/Utilization: Utilities like smart grids, water distribution, traffic management, and waste management would be unable to optimize resources based on demand patterns. Without Big Data processing, these systems would operate sub-optimally, leading to inefficiencies 3. Reduced Predictive Capabilities: Predictive maintenance for infrastructure (e.g., roads, bridges, public transport) would be compromised, leading to higher operational costs and failures. 4. Limited Personalization & Automation: Services like smart parking, smart lighting, and public transport scheduling would lack AI-driven optimization, affecting citizen experience. 5. Increased Security Risks: Without Big Data-driven anomaly detection, smart city security systems (CCTV, IoT sensors) would be unable to identify threats in real-time, increasing vulnerability to cyber and physical threats. 6. Increased Operational Costs: The inability to analyze large datasets would result in manual or semi-automated decision-making, increasing labor costs and reducing operational efficiency. **Alternative Solution: Edge Computing for Distributed Analytics (5 Marks)** To compensate for the absence of Big Data processing, a distributed analytics approach leveraging edge computing can be implemented. This solution shifts some of the data			

	processing and analysis closer to the source of data generation (IoT devices), reducing reliance on centralized Big Data systems. **Key Components of Solution:** 1. Edge Computing Nodes: Deploy edge devices (e.g., edge servers, gateways) at strategic locations within the smart city infrastructure. These devices preprocess and analyze data locally before sending only relevant information to central systems. 2. Distributed Analytics: Implement lightweight analytics algorithms on edge devices to perform real-time data processing and decision-making. Use machine learning models optimized for edge devices to enable predictive capabilities. 3. Data Filtering and Aggregation: Filter out redundant or low-priority data at the edge to reduce the volume of data sent to central systems. Aggregate data to provide summarized insights rather than raw data. 4. Event-Driven Processing IoT devices react to specific triggers without needing real-time cloud analytics. Example: Emergency response systems where fire sensors immediately notify local firefighting units without waiting for centralized analysis. 5. Local Decision-Making: Enable edge devices to make autonomous decisions for time-sensitive applications (e.g., adjusting traffic signals based on real-time traffic flow). 6. Hybrid Cloud-Edge Architecture: Use a combination of edge computing for real-time processing and cloud computing for long-term storage and advanced analytics. This hybrid approach ensures scalability and flexibility. 7. Inter-Edge Communication: Enable edge devices to communicate with each other to share insights and coordinate actions (e.g., traffic management across multiple intersections). While the absence of Big Data Analytics poses significant challenges to decision-making in smart city IoT deployments, edge computing and distributed analytics offer a viable alternative. By processing data locally and enabling real-time decision-making, this approach compensates for the lack of centralized Big Data processing while providing additional benefits such as reduced latency, improved reliability, and cost efficiency. This solution ensures that smart cities can still achieve their goals of efficiency, sustainability, and improved quality of life.			
3.	**Write a program for real world Smart Agriculture IoT application using Arduino, where sensor data is used for decision making. Justify your choice of microcontroller, communication protocol, and software framework.**	**10**	3	6
	Here is a Smart Agriculture IoT program using Arduino. This system monitors soil moisture, temperature, and humidity to automate irrigation, sending data to the cloud for remote monitoring. **Choices (3 Marks)** 1. Microcontroller: **ESP32** (for more advanced IoT Application) / Arduino Nano o Justification: ▪ Built-in Wi-Fi and Bluetooth for IoT connectivity.			

- Higher processing power than Arduino Uno (240 MHz vs. 16 MHz).
- Low power consumption, ideal for field applications.
- More GPIO pins for multiple sensor connectivity.

2. Sensors Used:
   - DHT11/DHT22 (Temperature & Humidity Sensor)
   - Soil Moisture Sensor
   - Water Pump (Relay Module for Actuation)
3. Communication Protocol: MQTT (Message Queuing Telemetry Transport)
   - Justification:
     - Lightweight and efficient for IoT applications.
     - Works well with low-power devices over Wi-Fi.
     - Ensures low latency real time data transmission/monitoring.
4. Software Framework: Arduino IDE with PubSubClient Library (for MQTT)
   - Justification:
     - Arduino IDE is widely supported and easy to use.
     - PubSubClient simplifies MQTT communication.
     - Supports rapid prototyping.
     - Easily integrates with IoT platforms like AWS, Google Cloud, or Node-RED.

Program for Smart Agriculture IoT Application (7 Marks)

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <DHT.h>

// WiFi and MQTT Server Credentials
const char* ssid = "Your_WiFi_SSID";
const char* password = "Your_WiFi_Password";
const char* mqtt_server = "broker.hivemq.com";  // Public MQTT Broker

// GPIO Pin Definitions
#define DHTPIN 4
#define DHTTYPE DHT22
#define SOIL_MOISTURE_PIN 34
#define RELAY_PIN 5

DHT dht(DHTPIN, DHTTYPE);
WiFiClient espClient;
PubSubClient client(espClient);

// Function to connect to WiFi
void setup_wifi() {
  delay(10);
  Serial.println("Connecting to WiFi...");
  WiFi.begin(ssid, password);
```

```
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
  }
  Serial.println("\nConnected to WiFi!");
}

// Function to reconnect to MQTT Broker
void reconnect() {
  while (!client.connected()) {
    Serial.println("Attempting MQTT connection...");
    if (client.connect("ESP32_SmartAgri")) {
      Serial.println("Connected to MQTT Broker!");
    } else {
      Serial.print("Failed, rc=");
      Serial.print(client.state());
      Serial.println(" Retrying in 5 seconds...");
      delay(5000);
    }
  }
}

void setup()
{
  Serial.begin(115200);
  dht.begin();
  pinMode(RELAY_PIN, OUTPUT);
  digitalWrite(RELAY_PIN, HIGH); // Initially turn off the pump

  setup_wifi();
  client.setServer(mqtt_server, 1883);
}

void loop()
{
  if (!client.connected()) {
    reconnect();
  }
  client.loop();

  float temp = dht.readTemperature();
  float humidity = dht.readHumidity();
  int soil_moisture = analogRead(SOIL_MOISTURE_PIN);
  int moisture_level = map(soil_moisture, 0, 4095, 100, 0); // Convert to percentage

  Serial.print("Temperature: ");
```

```
    Serial.print(temp);
    Serial.print("°C, Humidity: ");
    Serial.print(humidity);
    Serial.print("%, Soil Moisture: ");
    Serial.print(moisture_level);
    Serial.println("%");

    // Send data to MQTT Broker
    char tempString[8], humidityString[8], moistureString[8];
    dtostrf(temp, 6, 2, tempString);
    dtostrf(humidity, 6, 2, humidityString);
    dtostrf(moisture_level, 6, 2, moistureString);

    client.publish("smart_agri/temperature", tempString);
    client.publish("smart_agri/humidity", humidityString);
    client.publish("smart_agri/soil_moisture", moistureString);

    // Automated Irrigation Logic
    if (moisture_level < 30) {
        digitalWrite(RELAY_PIN, LOW);  // Turn ON Pump
        client.publish("smart_agri/irrigation", "ON");
        Serial.println("Pump Activated: Soil Moisture Low!");
    } else {
        digitalWrite(RELAY_PIN, HIGH);  // Turn OFF Pump
        client.publish("smart_agri/irrigation", "OFF");
        Serial.println("Pump Deactivated: Soil Moisture Sufficient.");
    }

    delay(5000);  // Send data every 5 seconds
}
```

4.	**a) Bluetooth, WiFi, and USB each have distinct advantages and limitations in IoT communication. If you were developing a remote patient monitoring system, which communication method would you prioritize, and why?**	**4**	3	6

Best Choice: **Wi-Fi** (Primary) + Bluetooth (Secondary for Wearables)

Why Prioritize Wi-Fi:
Direct Cloud Connectivity – Enables real-time remote monitoring without needing an intermediate device.
Higher Data Bandwidth – Suitable for transmitting continuous ECG, SpO_2, and HD video (if required).
Stronger Security – Supports WPA3 encryption, reducing the risk of cyber threats.
Broad Network Availability – Hospitals, clinics, and homes typically have Wi-Fi infrastructure.

	Why Use Bluetooth as a Secondary Option? Ideal for Wearable Devices – Smartwatches, fitness bands, and ECG patches use Bluetooth Low Energy (BLE) to connect to a central hub (e.g., smartphone or Wi-Fi gateway). Low Power Consumption – Ensures long battery life for patient-worn devices. Seamless Patient Mobility – Bluetooth sensors allow continuous monitoring even when moving around at home or in a hospital. For a remote patient monitoring system, **WiFi** should be prioritized due to its high data throughput, long range, real time remote access, and direct cloud connectivity. However, BLE should be used for wearable sensors to ensure low power consumption and seamless integration with the hub device. This hybrid approach balances the strengths of both communication methods, ensuring a reliable, scalable, and efficient RPM system.			
	b) Develop an Arduino program that: • **Uses Bluetooth to communicate with an Android app.** • **Sends sensor data (e.g., temperature) to the app.** • **Allows the app to send commands to control an LED or a relay on the microcontroller.**	**6**	3	6

	Below is an Arduino program that uses Bluetooth to communicate with an Android app. The program sends sensor data (e.g., temperature from a DHT11 sensor) to the app and allows the app to send commands to control an LED or relay connected to the microcontroller.
	
	Hardware Setup
	1. Microcontroller: Arduino Uno or Nano.
	2. Bluetooth Module: HC-05 or HC-06.
	3. Temperature Sensor: DHT11.
	4. LED: Connected to a digital pin (e.g., pin 13).
	5. Relay: Connected to another digital pin (e.g., pin 12).
	
	Arduino Program
	#include <SoftwareSerial.h>
	#include <DHT.h>
	
	// Define pins
	#define DHTPIN 2 // DHT11 data pin
	#define LED_PIN 13 // LED pin
	#define RELAY_PIN 12 // Relay pin
	#define BLUETOOTH_RX 10 // Bluetooth RX pin
	#define BLUETOOTH_TX 11 // Bluetooth TX pin
	
	// Initialize DHT sensor
	#define DHT_TYPE DHT11
	DHT dht(DHTPIN, DHT_TYPE);
	
	// Initialize Bluetooth communication
	SoftwareSerial bluetooth(BLUETOOTH_RX, BLUETOOTH_TX);			

```
void setup() {
  // Initialize serial communication
  Serial.begin(9600);
  bluetooth.begin(9600);

  // Initialize sensors and actuators
  pinMode(LED_PIN, OUTPUT);
  pinMode(RELAY_PIN, OUTPUT);
  digitalWrite(LED_PIN, LOW);
  digitalWrite(RELAY_PIN, LOW);
  dht.begin();

  Serial.println("System started. Waiting for Bluetooth connection...");
}

void loop() {
  // Read temperature from DHT11 sensor
  float temperature = dht.readTemperature();

  // Check if the reading is valid
  if (isnan(temperature)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  // Send temperature data to the Android app via Bluetooth
  bluetooth.print("Temperature: ");
  bluetooth.println(temperature);

  // Check for incoming commands from the Android app
  if (bluetooth.available()) {
    char command = bluetooth.read();
    Serial.println(command);  // Print the command to the Serial Monitor for debugging

    // Process the command
    switch (command) {
      case '1':  // Turn on LED
        digitalWrite(LED_PIN, HIGH);
        bluetooth.println("LED turned ON");
        break;
      case '0':  // Turn off LED
        digitalWrite(LED_PIN, LOW);
        bluetooth.println("LED turned OFF");
        break;
      case '3':  // Turn on relay
```

```
      digitalWrite(RELAY_PIN, HIGH);
      bluetooth.println("Relay turned ON");
      break;
    case '2':  // Turn off relay
      digitalWrite(RELAY_PIN, LOW);
      bluetooth.println("Relay turned OFF");
      break;
    default:
      bluetooth.println("Invalid command");
      break;
  }
 }

 // Delay before next reading
 delay(2000);  // Send data every 2 seconds
}
```

5.	**Scalability is a major challenge in IoT networks. If millions of devices suddenly join an IoT network, what kind of network reconfiguration issues might arise? Propose an approach to handle such scalability problems.**	**10**	4	4

Scalability is a critical challenge in IoT networks, especially when millions of devices suddenly join the network. This can lead to several network reconfiguration **issues (5 Marks)** as discussed below,

1. Addressing & Identification Issues: IPv4 has limited address space, making it difficult to assign unique IPs. DNS and device discovery may become overwhelmed.
2. Network Congestion & Latency: Increased data traffic leads to packet collisions, delays, and bottlenecks. High latency affects real-time IoT applications.
3. Bandwidth & Resource Allocation: Limited spectrum availability in WiFi, Zigbee, and Bluetooth networks. Backhaul networks (cellular, fiber) may become overloaded.
4. Security & Authentication Overhead: A sudden increase in devices requires scalable authentication mechanisms. Risk of DDoS attacks if unauthorized devices flood the network.
5. Edge & Cloud Processing Overload: Centralized cloud architectures may struggle with data storage and analytics. Edge devices might not handle increased processing loads effectively.
6. Interoperability Issues: Devices from different manufacturers may use incompatible protocols or standards. Integration and communication between devices become challenging.
7. Energy Efficiency and Dynamic Topology issue

Proposed Approach to Handle IoT Scalability (5 Marks)

To handle these challenges, an Adaptive Multi-Layered IoT Architecture can be implemented:

1. IPv6 & Hierarchical Addressing: Transition from IPv4 to IPv6, which supports 340

	undecillion unique addresses. Use hierarchical addressing (geographic, functional, or application-based segmentation) to optimize routing. 2. Edge Computing & Hierarchical Network Design: Move processing closer to the devices to reduce cloud dependency. Implement edge gateways to locally process & filter data, sending only relevant insights to the cloud. Hierarchical Network Design reduces the distance data must travel and minimizes congestion. 3. Software-Defined Networking (SDN) & Network Slicing: SDN enables dynamic traffic management, adapting to congestion in real time. Network slicing (5G technology) allows IoT devices to use separate virtualized networks based on priority (e.g., low latency vs. best-effort traffic). 4. AI-Driven Load Balancing & Traffic Optimization: AI algorithms predict traffic patterns and distribute requests dynamically. Use content delivery networks (CDNs) and edge caching to reduce cloud strain. 5. Protocols: Protocols like NETCONF and OVSDB help automate and manage configurations at scale. Replace traditional HTTP with lightweight IoT protocols like MQTT, CoAP, or LPWAN (LoRaWAN, NB-IoT). Reduce overhead and energy consumption for large-scale deployments. 6. Scalable Authentication & Security: Decentralized blockchain-based identity management ensures secure device authentication. Smart contracts automate access control and prevent DDoS attacks. 7. Efficient Routing and Modular Design: Energy efficient protocols like LOADng and CTP optimize data transmission, reducing energy consumption and improving network performance. Modular protocols and architectures allow for easy updates and scalability. Scalability challenges in IoT require a combination of IPv6, edge computing, AI-driven traffic management, light weight protocols, SDN, and security. By adopting a multi-layered adaptive architecture, IoT networks can efficiently handle millions of devices without performance degradation.