

ENDTERM 2024 IOT PAPER SOLUTION

SIMPLIFIED

Q1. Attempt all questions

(a) Write the characteristics of IOT system. (3)

- **Connectivity:** IoT devices must connect to a network (like the internet).
 - **Intelligence & Identity:** Devices gather useful data (intelligence) and each has a unique ID.
 - **Scalability:** Systems must handle many devices and large amounts of data.
 - **Dynamic & Self-Adapting:** Devices should adjust to changing situations.
 - **Heterogeneous Architecture:** Supports various devices and technologies from different makers.
 - **Safety & Security:** Must protect data/privacy and ensure physical device safety.
 - **Self-Configuring:** Devices should set themselves up and update easily.
-

(b) Why gateway is important for device management in IOT systems? (3)

- **Connects Different Protocols:** Translates between local device protocols (Zigbee, Bluetooth) and internet protocols (TCP/IP).
 - **Aggregates Data:** Collects data from many local devices and sends it to the cloud through one main connection.
 - **Enhances Security:** Acts as a security checkpoint (firewall, encryption) for vulnerable IoT devices.
 - **Local Processing (Edge):** Can process data locally, reducing cloud load and enabling faster responses for device management tasks.
 - **Facilitates Provisioning:** Helps manage device lifecycle (setup, configuration, updates) by bridging devices and management servers.
-

(c) List the basic difference between transducers, sensors, and actuators. (3)

Feature	Transducer	Sensor	Actuator
Main Job	Converts energy from one form to another.	Detects physical changes in environment.	Creates physical action/movement.
Input	Any energy/signal.	Physical stimulus (light, temp).	Electrical signal (usually).
Output	Different energy/signal.	Electrical signal representing detection.	Physical action (motion, light).

Feature	Transducer	Sensor	Actuator
Role in IoT	Underlies sensor/actuator function.	Gathers data from the physical world.	Interacts with/controls physical world.

(d) Why is an IDE required for prototyping the embedded device platform? (3)

- **Code Writing:** Provides an editor designed for writing code (syntax highlighting, auto-completion).
 - **Compiling:** Converts human-readable code into machine code that the embedded device understands.
 - **Uploading:** Transfers the compiled code onto the embedded device's memory.
 - **Debugging:** Helps find and fix errors in code using tools like simulators or serial monitors.
 - **Library Management:** Simplifies using pre-written code (libraries) for common tasks and hardware.
 - **Integrated Tools:** Combines all necessary development tools (editor, compiler, uploader) in one place.
-

(e) What is a smart sensor, how it is different from sensor node. (3)

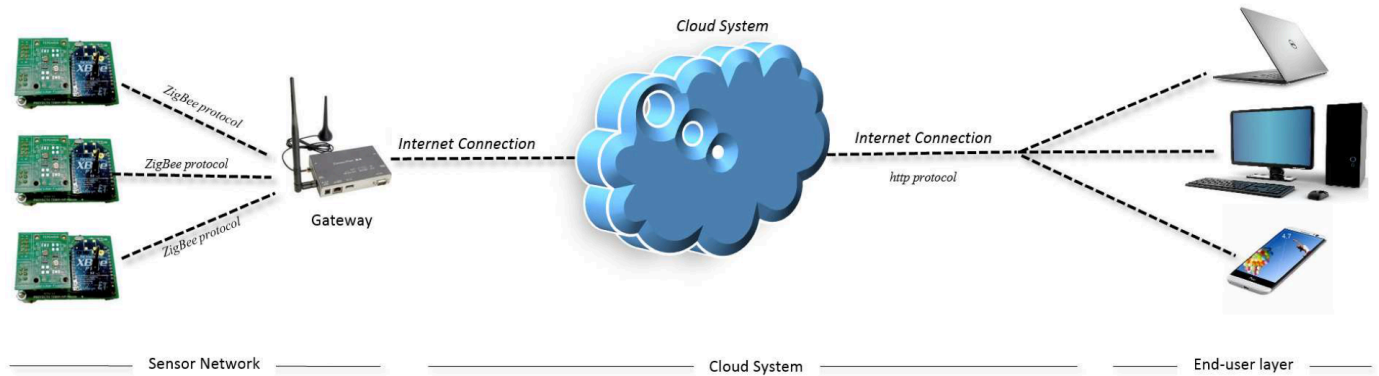
- **Smart Sensor:** A sensor with built-in processing capabilities. It can perform tasks like signal conditioning, data conversion (analog to digital), and some local analysis before sending data.
 - **Sensor Node:** A complete device in a network that *includes* one or more sensors (which can be basic or smart), a microcontroller, a wireless communicator, and a power source. Its main job is to collect and transmit data.
 - **Difference:**
 - A smart sensor is an *advanced component* with onboard intelligence.
 - A sensor node is a *networked device* that *uses* sensors. A sensor node can *contain* a smart sensor.
 - Smart sensors output more refined data; sensor nodes transmit raw or processed data from their sensors.
-

UNIT-I

Q2. (a) Explain the conceptual model and capabilities of an IoT solution with a neat diagram. (10)

Conceptual Model:

An IoT solution connects physical "things" to the internet to make them smart.



- **1. Things/Devices (Sensors & Actuators):**

- **Sensors:** Collect data from the environment (e.g., temperature, motion).
- **Actuators:** Perform actions based on data/commands (e.g., turn on a light).

- **2. Connectivity (Gateways & Networks):**

- Connects devices to the internet/cloud, often via a gateway.
- Uses technologies like Wi-Fi, Bluetooth, Cellular, LoRaWAN.

- **3. Data Processing (Cloud/Edge):**

- Raw data is sent here to be stored, processed, and analyzed.
- Cloud platforms (AWS, Azure) provide these services. Edge computing processes data locally.

- **4. Application & User Interface:**

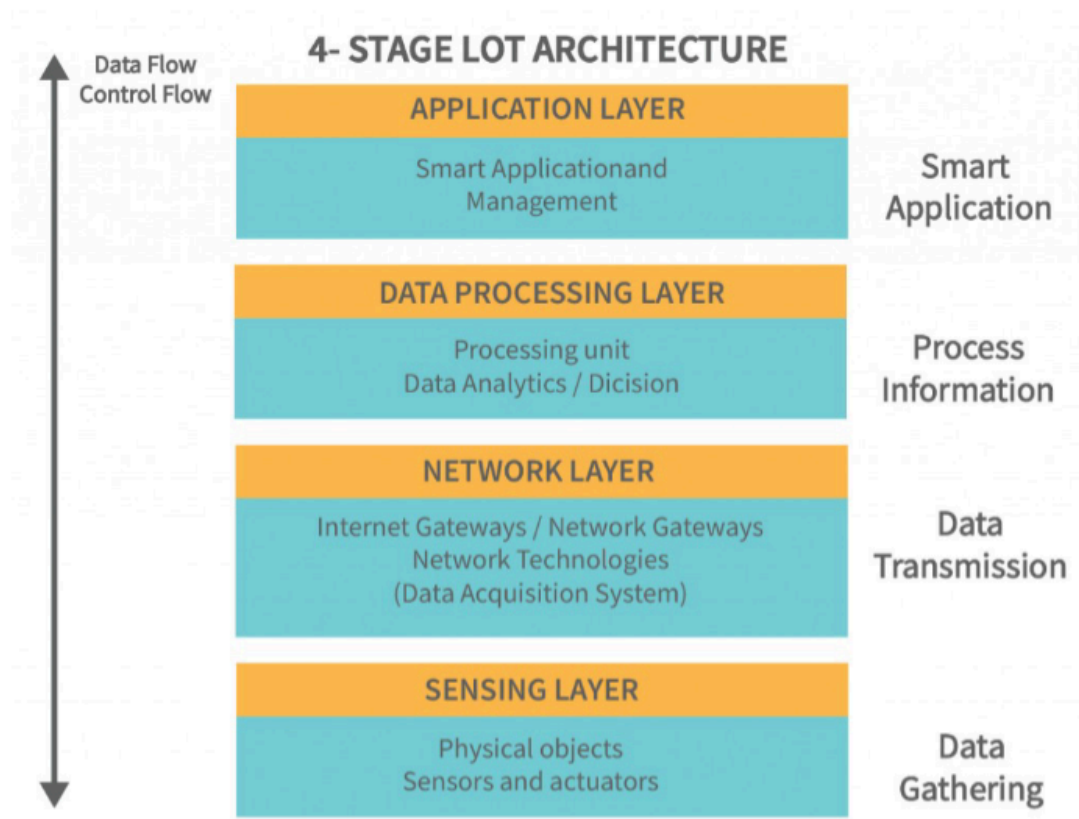
- Presents information to users (e.g., mobile app, dashboard).
- Allows users to control devices and view insights.

Capabilities of an IoT Solution:

- **Sensing & Data Collection:** Gathers real-time data.
- **Connectivity:** Enables communication between all parts.
- **Remote Monitoring & Control:** Manage devices from anywhere.
- **Data Analytics:** Turns raw data into useful insights and predictions.
- **Automation:** Automates tasks and processes based on data.
- **Optimization:** Improves efficiency and resource use.
- **Enhanced Decision Making:** Provides data for better decisions.

(b) Explain the role of four-layers in a smart city architectural framework. (5)

A smart city uses a layered architecture to manage its complex systems:



- **1. Perception/Sensing Layer:**
 - **Role:** Collects data from the city environment.
 - **Examples:** Traffic sensors, air quality monitors, smart meters, CCTV cameras.
- **2. Network Layer:**
 - **Role:** Transmits data from sensors to processing centers and commands to actuators.
 - **Examples:** Wi-Fi, 5G, LoRaWAN, fiber optic networks, gateways.
- **3. Processing/Data Management Layer:**
 - **Role:** Stores, processes, and analyzes city data to create insights.
 - **Examples:** Cloud platforms, big data analytics, AI algorithms (for traffic prediction, energy optimization).
- **4. Application/Service Layer:**
 - **Role:** Delivers smart city services to citizens and administrators.
 - **Examples:** Smart traffic control apps, smart parking, waste management systems, public safety alerts.

Q3. (a) Specify functions of COAP, RESTful HTTP, MQTT and XMPP in IoT applications. (7.5)

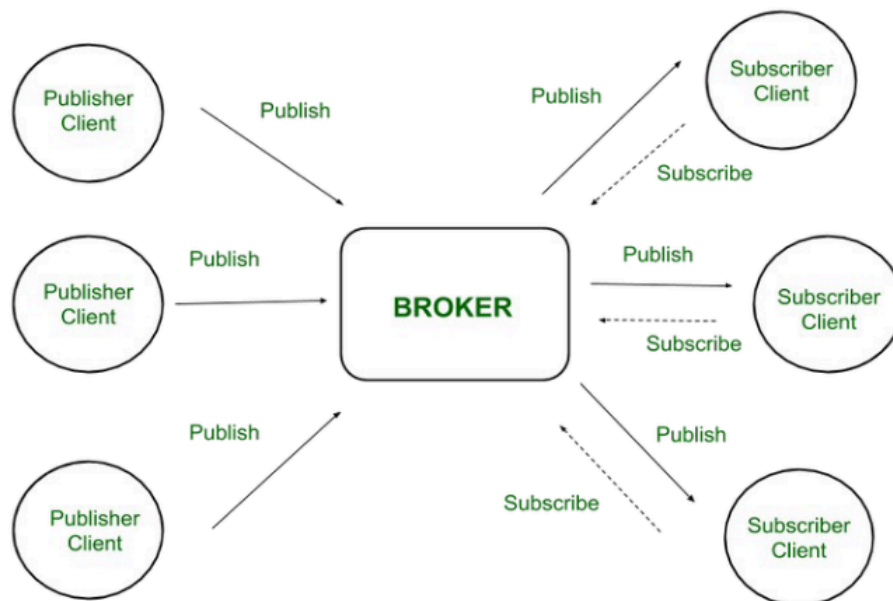
- **CoAP (Constrained Application Protocol):**
 - **Function:** Lightweight request/response for constrained (low-power, limited memory) devices. Runs on UDP. Good for simple sensor reading and actuator control. Supports "observe" for updates.

- **RESTful HTTP:**

- **Function:** Standard web protocol (using GET, POST, PUT, DELETE) for devices/gateways to communicate with cloud services and web APIs. Uses TCP. Good for robust, feature-rich communication with backend systems.

- **MQTT (Message Queuing Telemetry Transport):**

- **Function:** Lightweight publish/subscribe messaging for unreliable or low-bandwidth networks. Devices publish data to "topics," and other devices/apps subscribe to those topics. Uses a central broker. Good for telemetry and real-time updates.

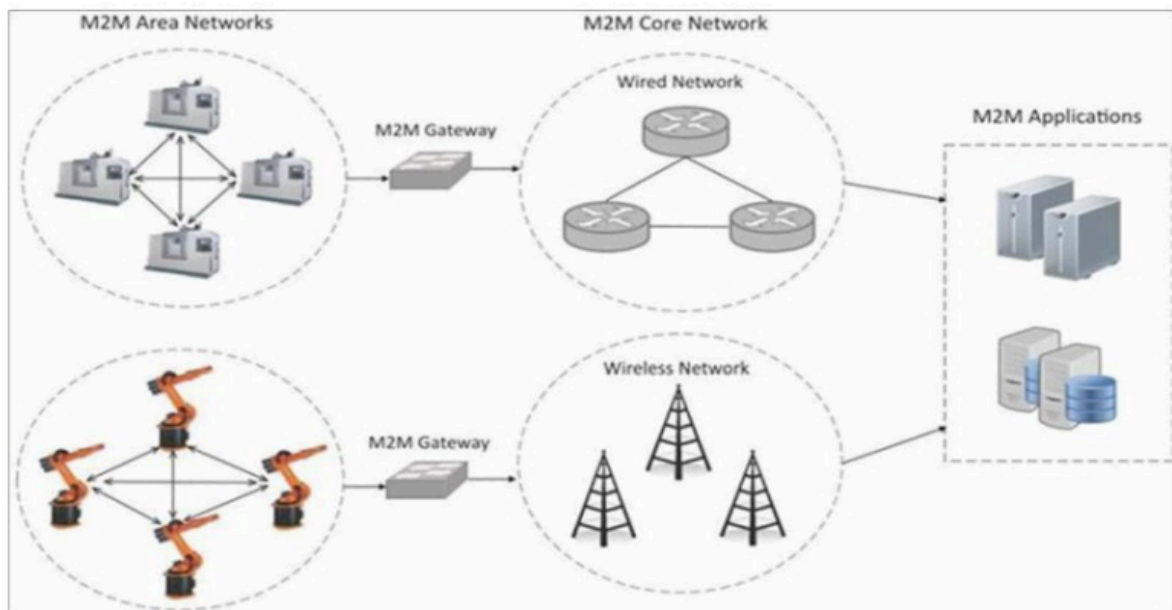


- **XMPP (Extensible Messaging and Presence Protocol):**

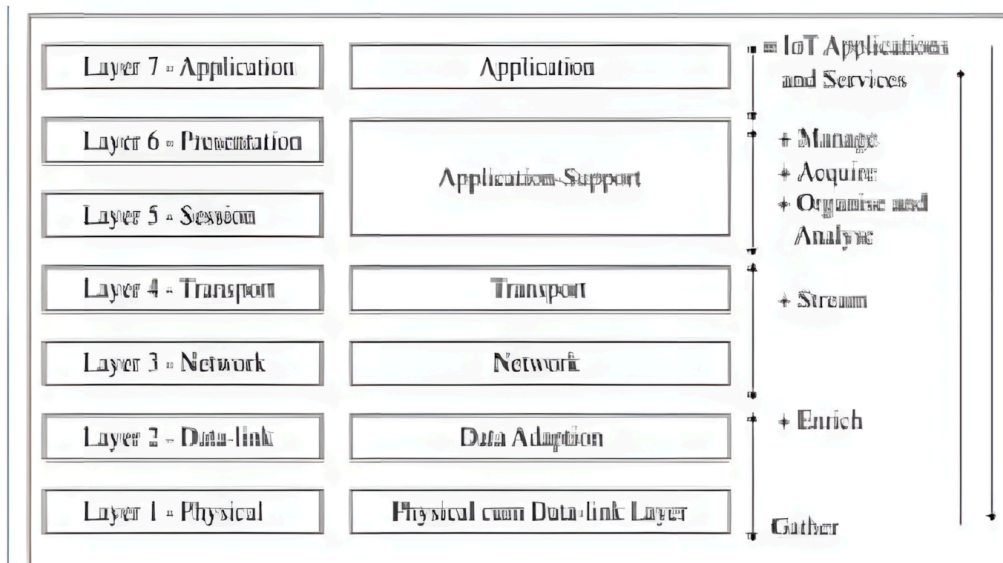
- **Function:** Real-time communication, presence (online/offline status), and exchange of structured XML data. Decentralized. Good for instant messaging-like interactions with devices or real-time status updates.

(b) Correlate M2M architectural domains with IoT architecture levels. (7.5)

M2M (Machine-to-Machine) is a foundation for IoT. ETSI M2M domains can be mapped to common IoT layers:



(ETSI M2M Domains)

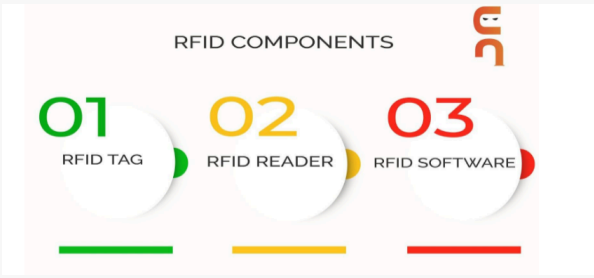


(IETF 6-Layer Model)

- **M2M Devices & Area Network:** Correspond to IoT **Perception Layer** (sensors/actuators) and the local part of the **Network Layer**. (Maps to IETF L1/L2).
- **M2M Gateway:** Part of IoT **Network Layer** (edge connectivity) and potentially **Processing Layer** (edge computing). (Maps to IETF L2).
- **M2M Access & Core Network:** Correspond to the wider **Network Layer** in IoT (WAN connectivity). (Maps to IETF L3/L4).
- **M2M Service Capabilities Layer (SCL):** Corresponds to IoT **Processing/Middleware Layer** (data management, device management). (Maps to IETF L5).
- **M2M Applications:** Correspond to IoT **Application Layer** (end-user services). (Maps to IETF L6).

UNIT-II

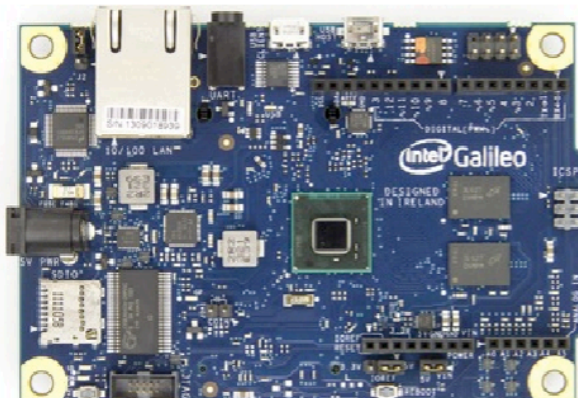
Q4. (a) Compare NFC and RFID protocols which can be used for device communication in IoT. (7.5)

Feature	NFC (Near Field Communication)	RFID (Radio Frequency Identification)
Range	Very Short (<10-20 cm)	Variable (cm to many meters)
Frequency	13.56 MHz (Standardized HF)	LF, HF (includes 13.56MHz), UHF, Microwave
Communication	Two-way (peer-to-peer, read/write, card emu.)	Mostly one-way (reader to tag); some writeable
Interaction	User-initiated "tap"	Can be automated (portal reading)
Multiple Reads	Typically one-to-one	Can read many tags at once (esp. UHF)
Use Cases	Payments, access, smart posters, pairing	Logistics, inventory, asset tracking, tolls
Security	Good due to short range; supports encryption	Varies; passive tags can be less secure
 <p>RFID COMPONENTS</p> <p>01 RFID TAG 02 RFID READER 03 RFID SOFTWARE</p> <p>(RFID Components)</p>		

(b) Describe usages of Intel Galileo, Raspberry and BeagleBone boards for IoT applications. (7.5)

- **Intel Galileo:**

- **Usage:** IoT prototyping needing x86 compatibility, Linux OS, and Arduino shield use. Good for more complex tasks than basic Arduino, like simple gateways or networked data loggers.



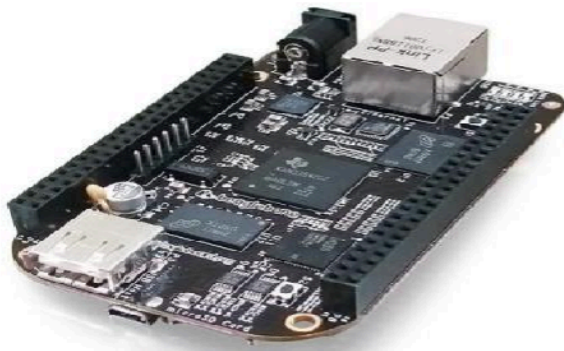
- **Raspberry Pi:**

- **Usage:** Versatile for IoT gateways/hubs, home automation controllers, local data servers, media centers with IoT control, robotics, and educational projects. Runs full Linux, strong processing & connectivity (Wi-Fi, Ethernet, Bluetooth).



- **BeagleBone:**

- **Usage:** Excellent for robotics, industrial control, and projects needing many I/O pins or real-time control (due to PRUs). Good for custom IoT gateways and data acquisition systems with specific hardware interfacing needs.



Q5. (a) What are the data-link, network, security and application layer protocols used in the WSNs? (10)

- **Data-Link Layer:**

- **IEEE 802.15.4:** Foundation for Zigbee, 6LoWPAN. Uses CSMA/CA.
- **WSN-specific MACs:** S-MAC, T-MAC, B-MAC (focus on energy efficiency, sleep cycles).

- **Network Layer:**

- **RPL:** Standard for routing in low-power, lossy networks (LLNs) over IPv6.
- **Adapted Ad-hoc:** AODV, DSR.
- **Clustering:** LEACH.
- **Geographic Routing.**

- **Security Protocols:**

- **Link Layer:** AES in IEEE 802.15.4.
- **Network/Transport:** SPINS, TinySec, (DTLS if using CoAP over UDP).

- **Key Management:** Critical challenge.
 - **Application Layer:**
 - **CoAP:** Lightweight RESTful protocol for constrained devices over UDP.
 - **MQTT/MQTT-SN:** Lightweight publish/subscribe for telemetry over TCP (MQTT) or UDP/non-IP (MQTT-SN).
 - **Custom Protocols:** Often used for specific, very constrained applications.
-

(b) Explain various node behaviours in WSN? (5)

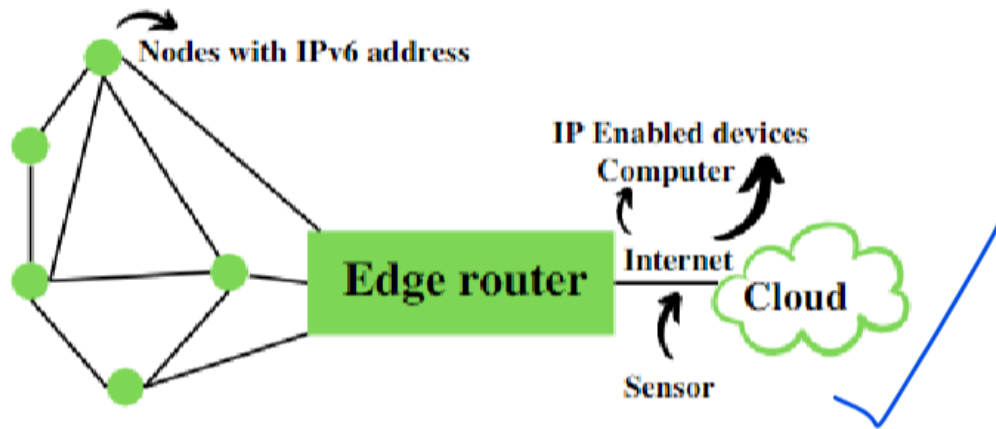
- **Sensing:** Collecting data from the environment using sensors.
 - **Processing:** Performing local calculations, filtering, or aggregation on sensed data.
 - **Communicating:** Transmitting/receiving data to/from other nodes or a sink.
 - **Routing/Relaying:** Forwarding data packets for other nodes in multi-hop networks.
 - **Sleeping/Active (Duty Cycling):** Alternating between low-power sleep and active states to save energy.
 - **Coordinating (Cluster Head/Sink):** Managing network formation, synchronization, data aggregation.
 - **Node Discovery:** Finding and joining the network or discovering neighbors.
-

UNIT-III

Q6. (a) Explain, why IoT device nodes use RPL in place of IPv6 and IPv4, why a CoAP client in place of HTTP client and 6LoWPAN at the adaptation layer in place of MAC. (10)

- **Why RPL instead of standard IPv4/IPv6 routing:**
 - Standard IP routing protocols (OSPF, RIP) are too heavy (large tables, frequent messages, high computation) for constrained IoT devices.
 - RPL is designed for Low-Power and Lossy Networks (LLNs): low memory/processing, builds efficient upward routes (DODAGs), uses Trickle timer for fewer control messages.
- **Why CoAP client instead of HTTP client:**
 - HTTP is too verbose (large text headers) and uses TCP (connection-oriented, high overhead) for constrained devices.
 - CoAP is lightweight (binary header), uses UDP (lower overhead), provides RESTful interaction, supports observe for updates, and is designed for constrained environments.
- **Why 6LoWPAN at adaptation layer instead of just MAC:**
 - MAC layers (like IEEE 802.15.4) provide local link communication but not internetworking (IP).
 - IPv6 packets are too large for small MAC frames of LLNs.
 - 6LoWPAN adapts IPv6 to LLNs by:

- **Header Compression:** Shrinks IPv6/UDP headers.
- **Fragmentation/Reassembly:** Splits large IP packets into smaller MAC-compatible frames.
- Enables end-to-end IP connectivity for constrained devices.



(6LoWPAN conceptual diagram)

(b) What are the header fields in 6LoWPAN? (5)

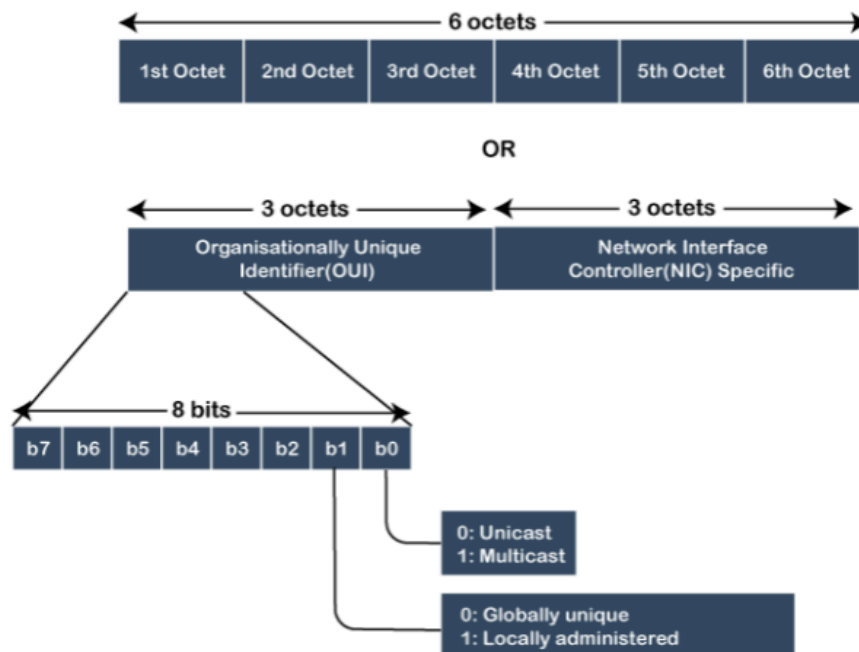
6LoWPAN uses a system of **Dispatch Headers** to indicate how IPv6 packets are compressed and encapsulated. Key header fields/information conveyed include:

- **Dispatch Value:** Identifies the type of header that follows (e.g., IPHC, Fragmentation, Mesh).
- **IPHC (IP Header Compression) fields:**
 - Flags for compressing/eliding IPv6 fields like Traffic Class, Flow Label, Hop Limit.
 - Modes for compressing Source and Destination IPv6 Addresses (e.g., stateless, stateful, short addresses).
- **UDP Header Compression fields (if UDP is used):**
 - Flags for compressing Source and Destination Ports.
 - Checksum elision.
- **Fragmentation Headers (FRAG1, FRAGN):**
 - `datagram_size`: Total size of the original unfragmented IPv6 packet.
 - `datagram_tag`: Unique identifier for all fragments of a single packet.
 - `datagram_offset` (in FRAGN): Offset of the current fragment.
- **Mesh Addressing Headers:**
 - Originator and Final Destination link-layer addresses.
 - Hop Limit for mesh routing.

Q7. (a) What is MAC address? How does a MAC address assign to an IoT node? How does address resolved to enable packets in Ip network reach the node. (10)

- What is MAC Address:

- A unique 48-bit hardware identification number assigned to a Network Interface Controller (NIC).
- Physical address, operates at Data Link Layer (Layer 2).
- Format: 12 hex digits (e.g., `00:1A:2B:3C:4D:5E`). First 24 bits = OUI (manufacturer), last 24 = device ID.



- **How MAC address is assigned to an IoT node:**
 - **Burned-In:** Hard-coded into the NIC hardware by the manufacturer.
 - Each network interface (Wi-Fi, Ethernet) on an IoT node has its own unique MAC.
- **How address is resolved for IP packets to reach the node (on local network):**
 - **ARP (Address Resolution Protocol) for IPv4:**
 1. Device A (sender) knows IP of Device B (IoT node) but not its MAC.
 2. Device A broadcasts ARP Request: "Who has IP_B? Tell MAC_A."
 3. Device B replies with ARP Reply: "IP_B is at MAC_B."
 4. Device A caches this IP-to-MAC mapping (ARP table).
 - **NDP (Neighbor Discovery Protocol) for IPv6:** Uses ICMPv6 messages (Neighbor Solicitation/Advertisement) similarly.
 - **Packet Delivery:** Sender creates Layer 2 frame with destination MAC address (of IoT node or gateway) to carry the IP packet. Switches use MAC addresses to forward frames locally.

(b) A subnet mask is `11111111 11111111 11111111 10010000`. IP address is `198.136.56.2`. How do these figures provide subnet and host addresses? (5)

1. **Subnet Mask (Binary to Decimal):**

`11111111.11111111.11111111.10010000` = `255.255.255.144`

2. **IP Address (Decimal to Binary):**

`198.136.56.2` = `11000110.10001000.00111000.00000010`

3. Network/Subnet Address (IP AND Mask):

11000110.10001000.00111000.00000010 (IP)

11111111.11111111.11111111.10010000 (Mask)

11000110.10001000.00111000.00000000 (Network Address)

Network/Subnet Address = 198.136.56.0

4. Host Address Identification:

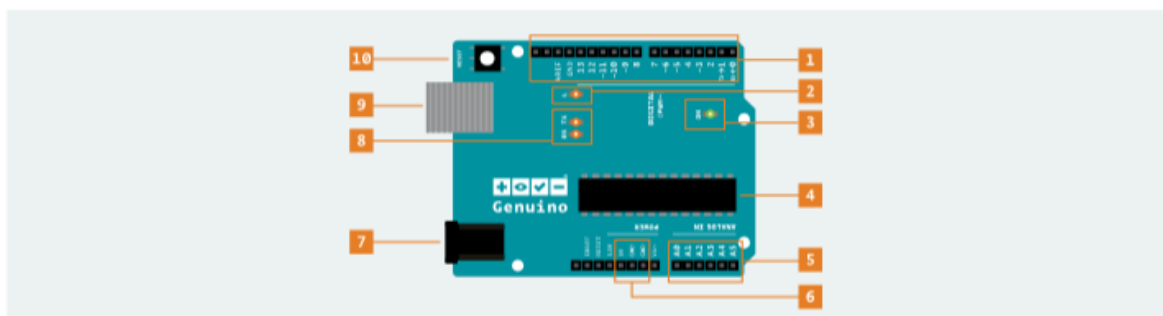
- The subnet mask **255.255.255.144** means the first 26 bits define the network/subnet portion (assuming a standard interpretation where the 1s in the mask are contiguous from the left, which **10010000** in the last octet isn't. If taken literally, the '1' bits in the mask determine the network portion).
- Given the literal mask, the network address is **198.136.56.0**.
- The host portion is what remains. For IP **198.136.56.2**, the host identifier within this subnet is **.2**. The specific host bits in the last octet are **000010**.
- **Bits for Host:** The mask **10010000** has four '0's. This implies 4 host bits in the last octet if we consider it a bitmask for host part directly, which is unconventional. Standard interpretation: **255.255.255.144** indicates where the 1s end and 0s begin to define network vs host. The mask **10010000** (144) has 2 ones, then two zeros, then one, then three zeros. This is not a standard CIDR mask.
- **Standard CIDR approach for 255.255.255.144:** This mask is not standard. A standard mask would have contiguous ones. For instance, **255.255.255.128** (/25) or **255.255.255.192** (/26).
- **If we strictly use the provided binary for the mask:** The network address is **198.136.56.0**. The host part of the IP **198.136.56.2** is the value **2**. The bits in the IP address that correspond to the **0**s in the mask (**01101111** in the last octet) would be **00000010**.

UNIT-IV

Q8. What is Arduino UNO? Explain its components with Pin Structure. List its features. (15)

• What is Arduino UNO:

- Popular open-source microcontroller board based on ATmega328P.
- Easy to use for beginners and prototyping IoT projects.



- **Components & Pin Structure:**

Architecture of Arduino board :

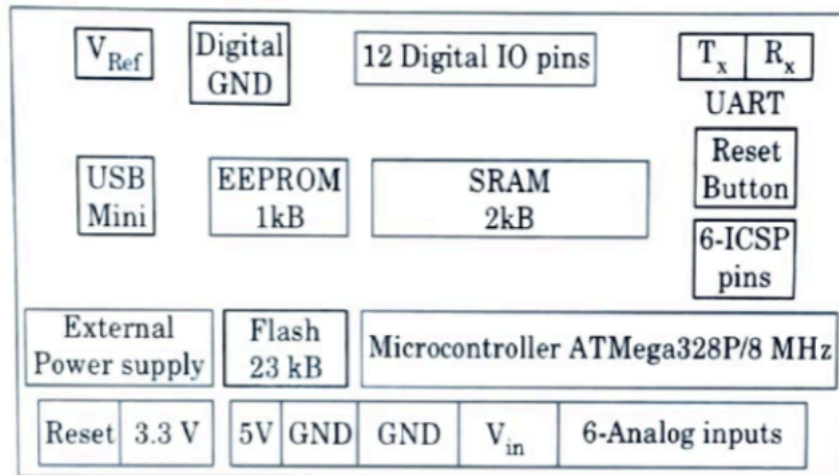


Fig. 2.17.1.

1. **ATmega328P Microcontroller:** The "brain."
2. **Digital I/O Pins (0-13):** For input/output; some have PWM (~). Pin 0 (RX), Pin 1 (TX) for Serial.
3. **Analog Input Pins (A0-A5):** Read analog sensor values (0-5V, converted to 0-1023).
4. **Power Pins:** 5V, 3.3V, GND, V_{in} (external power), IOREF.
5. **USB Port:** For programming, power, and serial communication.
6. **Barrel Jack:** For external power supply (7-12V recommended).
7. **Reset Button:** Restarts the program.
8. **Crystal Oscillator:** 16 MHz clock for the microcontroller.
9. **ICSP Header:** For direct microcontroller programming.
10. **LEDs:** Power (ON), Pin 13 (L), TX, RX.

- **Features:**

- Open-source, ATmega328P MCU, 14 Digital I/O (6 PWM), 6 Analog In, USB, Serial/I2C/SPI, Arduino IDE, large community, extensive libraries, shields for expansion, affordable.

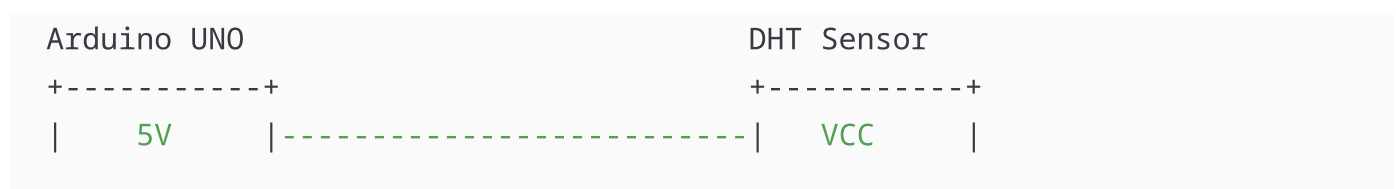
Q9. Draw a circuit diagram of connecting DHT sensor with Arduino. Explain its pin structure.

Write a program to connect DHT sensor with Arduino to read the Temperature and Humidity. (15)

1. DHT Sensor Pin Structure (e.g., DHT11/DHT22):

- * **VCC:** Power (3.3V - 5V)
- * **DATA:** Digital data output (one-wire protocol)
- * **NC (Often):** Not Connected
- * **GND:** Ground

2. Circuit Diagram:



	GND	-----	GND	
	Digital 2	----[10kΩ Pull-up to 5V*]--	DATA	
+-----+			+-----+	

*Pull-up resistor needed if using bare sensor, often included on modules.

(Visual: Arduino connected to DHT11/22 on a breadboard. 5V to VCC, GND to GND, Pin 2 to DATA with a 10kΩ resistor pulling DATA up to 5V.)

 DHT Sensor with Arduino (Placeholder for a drawn circuit)

3. Arduino Program (using Adafruit DHT library):

```
#include "DHT.h"           // Include DHT library

#define DHTPIN 2           // Data pin of DHT connected to Arduino pin 2
#define DHTTYPE DHT11      // Change to DHT22 or DHT21 if using those

DHT dht(DHTPIN, DHTTYPE); // Initialize DHT sensor object

void setup() {
  Serial.begin(9600); // Start serial communication
  Serial.println("DHT Sensor Test");
  dht.begin();        // Initialize the sensor
}

void loop() {
  delay(2000); // Wait 2 seconds between measurements

  float humidity = dht.readHumidity();
  float temperatureC = dht.readTemperature(); // Read temperature in Celsius

  if (isnan(humidity) || isnan(temperatureC)) { // Check for read failure
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  Serial.print("Humidity: ");
  Serial.print(humidity);
  Serial.print("%  Temperature: ");
  Serial.print(temperatureC);
  Serial.println(" °C");
}
```

Explanation:

- Includes DHT library.

- Defines sensor type and data pin.
 - Initializes sensor and serial communication in `setup()`.
 - In `loop()`, reads humidity and temperature every 2 seconds.
 - Prints values to Serial Monitor, includes error check for failed readings.
-