UNIT IV

Interoperability of IoT:

Internet of Things (IoT) is an ever-growing network of physical devices embedded with sensors, actuators, and wire-less connectivity to communicate and share their information among themselves. The application of IoT is in diverse areas such as agriculture, poultry and farming, smart city, and health care, where a sensor node must support heterogeneous sensors/actuators, and varying types of wireless connectivity. **Interoperability** is the ability of two or more devices, systems, platforms or networks to work in conjunction. Interoperability enables communication between heterogeneous devices or system in order to achieve a common goal. However, the current devices and systems are fragmented with respect to the communication technologies, protocols, and data formats. This diversity makes it difficult for devices and systems in the IoT network to communicate and share their data with one another. The utility of IoT network is limited by the lack of interoperability.

Figure 1 shows the various interactions between components of an IoT system, each arrow being an interaction capability or an *interoperability point*. Some interactions can be visible by users, for instance, a smartphone can interact with a smartwatch. Others are internal, for instance, an interaction between a security surveillance application and an IoT home gateway, or an interaction between the home gateway and an intrusion detector.

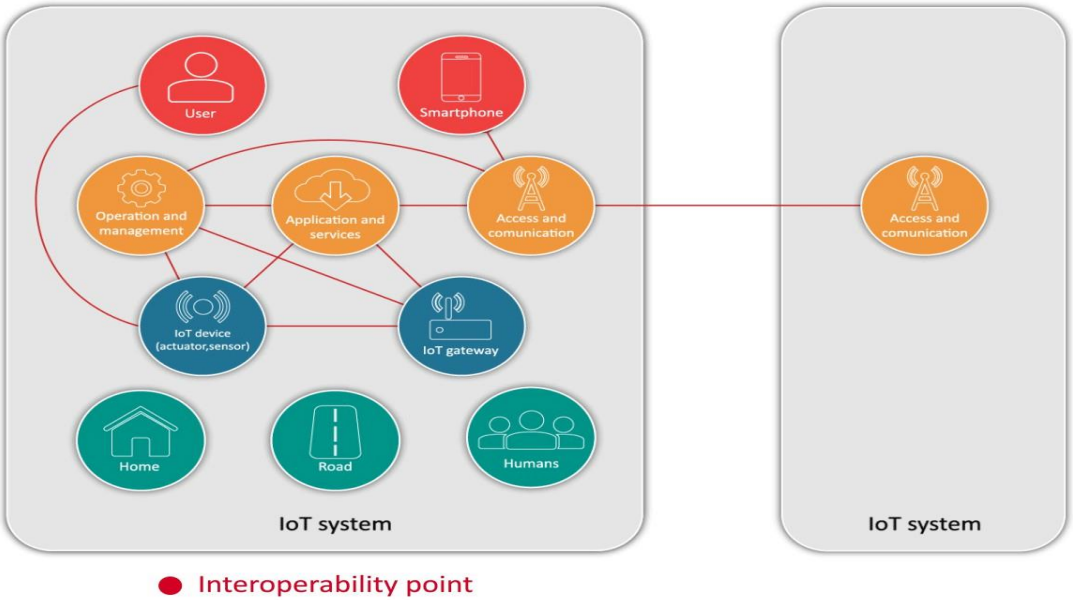
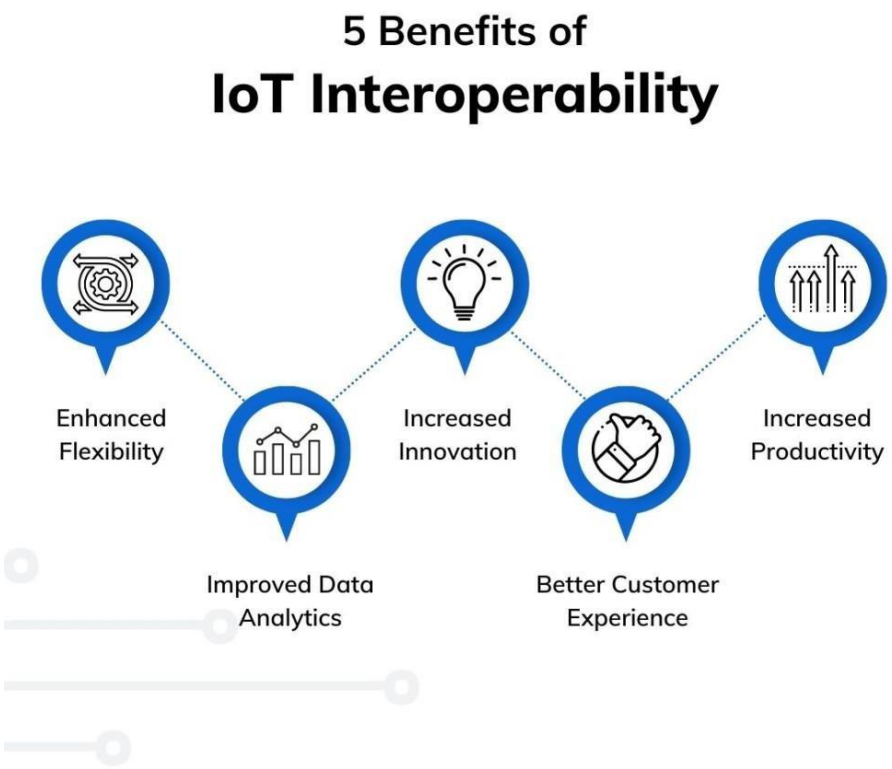


Figure 1. various interactions between components of an IoT system

Benefits of IoT Interoperability



Benefits of IoT Interpretability

As the Internet of Things (IoT) continues to expand, interoperability has become a crucial aspect of its growth. By enabling different devices and systems to communicate and work together seamlessly, IoT interoperability offers a range of benefits for businesses and consumers alike.

Enhanced Flexibility and Scalability: IoT interoperability allows different devices, platforms, and systems to work together seamlessly. This enables businesses to scale up or down their solutions, depending on their changing requirements, without worrying about compatibility issues.

Increased Efficiency and Productivity: When devices and systems can communicate and share data with each other, businesses can automate many of their processes and operations. This reduces the need for manual intervention and increases overall efficiency and productivity.

Improved Data Analytics: Interoperability enables businesses to collect data from different sources and analyze it together. This allows for a more comprehensive understanding of the data and can provide valuable insights that might not be available if the data were analyzed in isolation.

Better Customer Experience: Interoperability can provide a seamless experience for customers, as devices and systems can work together to deliver a more cohesive and personalized experience. For example, a smart home system that can communicate with a home security system can provide a more secure and convenient experience for the homeowner.

Increased Innovation: Interoperability can encourage innovation by enabling businesses to combine different technologies and solutions to create new products and services. This can lead to the development of new markets and business opportunities, as well as the creation of new jobs and industries.

Introduction to arduino programming part I:

Arduino programming in the context of the Internet of Things (IoT) involves using the Arduino IDE to write code for Arduino boards, enabling them to interact with sensors, actuators, and the internet. This allows for building various IoT applications like smart homes, environmental monitoring, and more. The Arduino platform simplifies the process with its user-friendly hardware and software, making it accessible for beginners to learn and implement IoT solutions.

Arduino is an open-source platform for developing electronic projects. It is an easy tool for fast prototyping for people who do not have a programming background. Arduino comprises two parts—software (Arduino IDE) and a circuit board as hardware, called the microcontroller.

Key aspects of Arduino programming for IoT:

- **Arduino IDE:**

  The Integrated Development Environment (IDE) is the software used to write, compile, and upload code to the Arduino board.

- **Programming Language:**

  Arduino code is based on a variant of C++, with specific functions and methods for interacting with the hardware.

- **Hardware:**

  Arduino boards, like the Arduino Uno, are microcontroller-based and can be powered via USB or a barrel jack.

- **Input and Output:**

  Arduinos can read data from sensors (light, temperature, etc.) and control actuators (motors, LEDs) based on the programmed logic.

- **IoT Applications:**
  Arduino's simplicity and hardware flexibility make it a popular choice for prototyping and building various IoT projects, such as smart homes, environmental monitoring, and automation systems.

Basic Structure of an Arduino Program (Sketch):

- void setup():

  This function runs once at the beginning when the Arduino is powered on or reset. It's used to initialize the board, configure pins, and set up serial communication.

- void loop():
  This function runs repeatedly after the setup() function completes. It contains the main logic of the program, reading inputs, processing data, and controlling outputs.
  Example:

A simple sketch to blink an LED connected to pin 13 would include:

1. setup(): Setting pin 13 as an output: pinMode(13, OUTPUT);
2. loop(): Toggling the LED on and off with a delay:

Code:

```
digitalWrite(13,HIGH);
delay(1000);
digitalWrite(13,LOW);
delay(1000);
```

Introduction to arduino programming part II:

Arduino programming part 2 for IoT builds on the basics, delving into more advanced concepts and their application in Internet of Things projects. This part typically covers topics like operators, control statements, loops, arrays, strings, the math library, random number generation, and interrupts. It also often includes practical examples of how to connect and control hardware components like LEDs and sensors, and how to use these elements in simple IoT applications, such as a traffic control system.

Here's a more detailed breakdown:

Core Programming Concepts:

- **Operators:**

  Understanding arithmetic, relational, logical, and bitwise operators is crucial for writing conditional logic and manipulating data.

- **Control Statements:**

  if, else, switch statements allow for decision-making based on conditions, enabling the Arduino to respond differently to various inputs.

- **Loops:**

  for, while, and do-while loops enable repetitive tasks, essential for tasks like reading sensor data or controlling actuators over time.

- **Arrays:**

  Arrays allow storing and accessing multiple values of the same data type using an index, which is useful for managing collections of data, like sensor readings.

- **Strings:**

  Working with strings (sequences of characters) is important for displaying messages, processing user input, and communicating with other devices.

- **Math Library:**

  The Arduino's math library provides functions for mathematical operations, such as calculating square roots, trigonometric functions, and more.

- **Random Numbers:**

  Generating random numbers is useful for simulations, games, and other applications where unpredictable behavior is needed.

- **Interrupts:**
  Interrupts allow the Arduino to respond to external events (like a button press) without constantly polling for changes, enabling more efficient code execution.
  IoT Specifics:

- **Sensor Interfacing:**

  Learning how to connect and read data from various sensors (temperature, light, motion, etc.) is fundamental to IoT applications.

- **Actuator Control:**

  Controlling actuators (like motors, LEDs, relays) is essential for interacting with the physical world.

- **Communication:**

  Understanding how to send and receive data over various communication protocols (like serial, WiFi, Bluetooth) is crucial for connecting the Arduino to other devices and the internet.

- **MQTT:**

  A common protocol for IoT communication, MQTT allows for lightweight publish/subscribe messaging.

- **Shields:**
  Arduino shields are add-on circuit boards that provide additional functionality, such as WiFi connectivity, motor control, or display interfaces.
  Example:

  A common example in Arduino IoT projects is a traffic control system. This involves using multiple LEDs to simulate traffic lights, and using the concepts mentioned above to control their sequence and timing.

  Resources:

- **Arduino IDE:**

  The Integrated Development Environment (IDE) is used to write, compile, and upload code to the Arduino board.

- **Online Tutorials and Examples:**

  Numerous online resources, including websites, videos, and documentation, offer guidance and examples for Arduino programming and IoT projects.

- **Arduino Forums:**
  The Arduino community provides a platform for asking questions, sharing knowledge, and troubleshooting issues.

Aim : write a program to turn on LED for 1 second after every two seconds

Program:

```
void setup ()

{

pinMode ( 13, OUTPUT);  // to set the OUTPUT mode of pin number 13.

pinMode ( 7, OUTPUT);  // to set the OUTPUT mode of pin number 7.

}

void loop ()

{

digitalWrite (13, HIGH);
```

```
digitalWrite (7, LOW);
delay(1500);  // 1.5 second = 1.5 x 1000 milliseconds
digitalWrite (13, LOW);
digitalWrite (7, HIGH);
delay(1000);  // 1 second = 1 x 1000 milliseconds
  }
```

Write a program for Controlling the Light Emitting Diode (LED) with a push button.
Program:

```
/*****************Pressing Button LED*****/
    const int buttonPin = 7; // choose the pin for the pushbutton
    const int ledPin = 11;  // choose the pin for a LED
    int buttonState = 0;   // variable for reading the pushbutton pin status
    void setup()
    {
     pinMode(ledPin, OUTPUT);  // declare LED as output
     pinMode(buttonPin, INPUT);    // declare pushbutton as input
    }
    void loop()
    {
     buttonState = digitalRead(button Pin);  // read input value
     if (buttonState == HIGH)
     {        // check if the input is HIGH (button pressed)
     digitalWrite(ledPin, HIGH);  // turn LED ON

     }
    else
     {
      digitalWrite(ledPin, LOW);  // turn LED OFF}}
```

Integrating sensors and actuators with Arduino:

Integrating sensors and actuators with Arduino for IoT applications involves several key steps:

1. Understanding Sensors and Actuators:

- **Sensors:**

  Devices that detect and measure physical parameters from the environment (e.g., temperature, humidity, light, motion) and convert them into electrical signals for the Arduino. Examples include DHT11 (temperature/humidity), PIR (motion), LDR (light), ultrasonic sensors (distance).

- **Actuators:**
  Devices that receive signals from the Arduino and perform a physical action in response (e.g., turning on/off lights, controlling motors, opening/closing valves). Examples include LEDs, servo motors, relays, DC motors.
  2. Hardware Connection:

- **Power:**

  Connect the VCC/power pin of the sensor/actuator to the appropriate power pin on the Arduino (e.g., 5V or 3.3V).

- **Ground:**

  Connect the GND/ground pin of the sensor/actuator to the GND pin on the Arduino.

- **Data/Signal:**
  Connect the data or signal pin of the sensor/actuator to a suitable digital or analog pin on the Arduino, depending on the device's requirements. Some actuators, like high-power motors, may require external power sources and driver circuits.
  3. Software Development (Arduino Sketch):

- **Include Libraries:**

  For many sensors and actuators, specific libraries are available to simplify interaction. Install these libraries through the Arduino IDE's Library Manager (Sketch > Include Library > Manage Libraries).

- **Declare Pins and Variables:**

  Define the Arduino pins connected to the sensor/actuator and declare any necessary variables for storing data or controlling states.

- Setup Function (setup()):
- Initialize serial communication for debugging and monitoring (e.g., Serial.begin(9600);).
- Set pin modes for connected components (e.g., pinMode(sensorPin, INPUT); for sensors, pinMode(actuatorPin, OUTPUT); for actuators).
- Attach servo motors if using the Servo library (e.g., myservo.attach(servoPin);).
- Loop Function (loop()):

- **Sensor Reading:** Read data from the sensor using appropriate functions provided by its library or direct pin reads (e.g., digitalRead(), analogRead()).
- **Data Processing:** Process the sensor data as needed (e.g., converting raw readings to meaningful units, applying logic based on thresholds).
- **Actuator Control:** Control the actuator based on the processed sensor data or other logic (e.g., digitalWrite(), analogWrite(), myservo.write()).
- **Upload Code:**
  Connect the Arduino to your computer, select the correct board and port in the Arduino IDE, and upload the sketch.
  4. Testing and Iteration:
- Monitor sensor readings through the Serial Monitor.
- Observe actuator responses to verify correct functionality.
- Adjust code or hardware connections as needed to optimize performance and achieve desired behavior.

Introduction to Raspberry pi:

Raspberry pi is the name of the "credit card-sized computer board" developed by the Raspberry pi foundation, based in the U.K. It gets plugged in a TV or monitor and provides a fully functional computer capability. It is aimed at imparting knowledge about computing to even younger students at the cheapest possible price. Although it is aimed at teaching computing to kids, but can be used by everyone willing to learn programming, the basics of computing, and building different projects by utilizing its versatility.

Used:
It also provides a set of general purpose input/output pins allowing you to control electronic components for physical computing and explore the Internet of Things (IOT).

Raspberry Pi, a small, affordable computer, is widely used in Internet of Things (IoT) applications due to its versatility and processing power. Its General Purpose Input/Output (GPIO) pins allow it to interface with various sensors and actuators, while its processing capabilities enable data analysis and control of connected devices. It can act as a central hub for collecting, processing, and transmitting data, making it suitable for diverse IoT projects.

Key Features of Raspberry Pi for IoT:

- **Small and Affordable:**

  Its compact size and low cost make it ideal for resource-constrained environments and prototyping.

- **GPIO Pins:**

  These pins allow for easy connection to sensors, actuators, and other hardware components, enabling real-time data acquisition and control.

- **Processing Power:**

  The Raspberry Pi's processor enables on-device data processing, reducing the need to transmit all raw data to the cloud.

- **Networking Capabilities:**

  It supports Wi-Fi, Bluetooth, and Ethernet, allowing for seamless connectivity to networks and cloud platforms.

- **Software Flexibility:**
  It supports various operating systems like Raspberry Pi OS (Debian-based) and Python, offering flexibility in development.
  Common IoT Applications of Raspberry Pi:

- **Smart Homes:**

  Controlling lighting, temperature, security systems, and other appliances.

- **Environmental Monitoring:**

  Building weather stations, tracking air quality, and monitoring plant health.

- **Industrial Automation:**

  Controlling and monitoring industrial processes, collecting data from sensors, and optimizing operations.

- **Security Systems:**

  Implementing surveillance systems with cameras, motion detectors, and alarm systems.

- **Agricultural Technology:**
  Developing automated irrigation systems, monitoring crop health, and optimizing farming practices.
  How Raspberry Pi Functions as an IoT Device:

  1. Sensing:

  Sensors connected to the GPIO pins collect data from the physical environment.

  2. Processing:

  The Raspberry Pi processes this data, performing tasks like filtering, analysis, and decision-making.

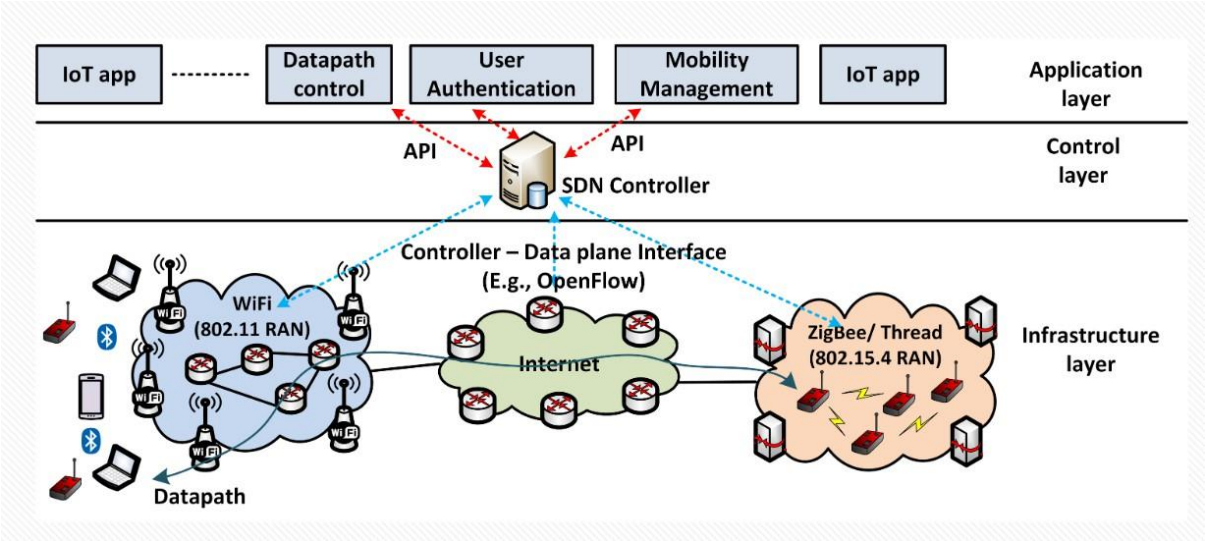
  3. Actuation:

  Based on the processed data, the Raspberry Pi can control actuators, such as motors, lights, or valves.

4. Communication:
The Raspberry Pi transmits data to other devices or cloud platforms using its networking capabilities.

Software Defined Networking for IoT

Internet of things (IoT) poses challenges that are different from traditional Internet in different aspects — heterogeneous communication technologies, application-specific QoS requirements, massive influx of data, and unpredictable network conditions. On the other hand, software-defined networking (SDN) is a promising approach to control the network in a unified manner using rule-based management. The abstractions provided by SDN enable holistic control of the network using high-level policies, without being concerned about low-level configuration issues. Hence, it is advantageous to address the heterogeneity and application-specific requirements of IoT.



Software-Defined Networking (SDN) is an approach to networking that uses software-based controllers or application programming interfaces (APIs) to communicate with underlying hardware infrastructure and direct traffic on a network.

Why Software-Defined Networking is important?

SDN represents a substantial step forward from traditional networking, in that it enables the following:

- **Increased control with greater speed and flexibility:** Instead of manually programming multiple vendor-specific hardware devices, developers can control the

flow of traffic over a network simply by programming an open standard software-based controller. Networking administrators also have more flexibility in choosing networking equipment, since they can choose a single protocol to communicate with any number of hardware devices through a central controller.

- **Customizable network infrastructure:** With a software-defined network, administrators can configure network services and allocate virtual resources to change the network infrastructure in real time through one centralized location. This allows network administrators to optimize the flow of data through the network and prioritize applications that require more availability.
- **Robust security:** A software-defined network delivers visibility into the entire network, providing a more holistic view of security threats. With the proliferation of smart devices that connect to the internet, SDN offers clear advantages over traditional networking. Operators can create separate zones for devices that require different levels of security, or immediately quarantine compromised devices so that they cannot infect the rest of the network.

The key difference between SDN and traditional networking is infrastructure:

SDN is software-based, while traditional networking is hardware-based. Because the control plane is software-based, SDN is much more flexible than traditional networking. It allows administrators to control the network, change configuration settings, provision resources, and increase network capacity — all from a centralized user interface, without the need for more hardware.

There are also security differences between SDN and traditional networking. Thanks to greater visibility and the ability to define secure pathways, SDN offers better security in many ways. However, because software-defined networks use a centralized controller, securing the controller is crucial to maintaining a secure network.

How does Software-Defined Networking (SDN) work?

Here are the SDN basics: In SDN (like anything virtualized), the software is decoupled from the hardware. SDN moves the control plane that determines where to send traffic to software, and leaves the data plane that actually forwards the traffic in the hardware. This allows network

administrators who use software-defined networking to program and control the entire network via a single pane of glass instead of on a device by device basis.

There are three parts to a typical SDN architecture, which may be located in different physical locations:

Applications, which communicate resource requests or information about the network as a whole

Controllers, which use the information from applications to decide how to route a data packet

Networking devices, which receive information from the controller about where to move the data.