UNIT -1 Introduction to Internet of Things

Introduction

Internet of Things (IoT) comprises things that have unique identities and are connected to the internet. While many existing devices, such as networked computers or 4G-enabled mobile phones, already have some form of unique identities and are also connected to the Internet, that are focus on IoT is in the configuration, control and networking via the Internet of devices or "things" that are traditionally notassociated with the Internet. These include devices such as thermostats, utility meters, a Bluetooth connected headset , irrigation pumps and sensors, or control circuits for an electric car's engine.

Internet of Things is a new revolution in the capabilities of the endpoints that are connected to the Internet, and is being driven by the advancement in capabilities in sensor networks, mobile devices, wireless communications and networking and cloud technologies.

Data: Raw and unprocessed data obtained from IoT device/Systems

Information: Information is inferred from data by filtering, processing, categorizing, condensing and contextualizing data.

Knowledge: knowledge is inferred from information by organizing & structuring information and is put into action to achieve specific objectives.

Inferring information and knowledge in IoT

- The scope of IOT is not limited to just connecting things (devices, appliances, machines) to the internet.
- IOT allows these things to communicate and exchange data (control & information, that could include data associated with the users) while executing meaningful applications towards a common user or machine goal.
- Data itself does not have a meaning until it is contextualized processed into useful information
- Application on IoT networks extract and create information from lower level data by filtering, processing, categorizing, condensing and contextualizing the data.
- This information obtained is then organized and structured to infer knowledge about the system and/or its users, its environment, and its operations and progress towards its objectives, allowing a smarter performance.

Applications of IoT

Home

- Smart lighting
- Smart Appliance
- Intrusions Detection
- Smoke/Gas Detectors

Cities

- Smart Parking
- mart Roads
- Structural Health Monitoring
- Emergency Response

Environment

- Wealth Monitoring
- Air pollution Monitoring
- Noise pollution Monitoring
- Forest fire Detection

Energy

- Smart Grids
- Renewable Energy Systems
- Prognostics

Retail

- Inventory Management
- Smart Payments
- Smart Vending Machines

Logistics

- Route Generation & Scheduling
- Fleet Tracking
- Shipment Monitoring
- Remote Vehicle Diagnostics

Agriculture

- Smart Irrigation
- Green House Control

Industry

- Machine Diagnosis & Prognosis
- Indoor Air Quality Monitoring

Health & Life style

- Health & Fitness Monitoring
- Wearable Electronics

- IoT has several applications such as smart lighting that adapt the lighting to suit the ambient conditions, Smart appliances that can be remotely monitored and controlled, intrusion detection systems, smart smoke detectors, etc.
- For cities, IoT has applications such as smart parking systems that provide status updates on available slots, Smart lighting that helps in saving energy, smart roads thatprovide information on driving conditions and structural health monitoring system.
- For environment, IoT has applications such as weather monitoring, air and noise pollution, forest fire detection and river flood detection systems.
- For energy systems, IoT has applications such as including smart grids, grid integration of renewable energy sources.
- For retail domain, IoT has applications such as inventory management, smart payments and smart vending machines.
- For agriculture domain, IoT has applications such as smart irrigation systems that help in saving water while enhancing productivity and green house control systems.
- Industrial applications of IoT include machine diagnosis and prognosis system that help in predicting faults and determining the cause of faults and indoor air quality systems.
- For health and life style, IoT has applications such as health and fitness monitoring systems and wearable electronics.

Definition & characteristics of IoT

The Internet of Things (IoT) has been defined as

Definition:

A dynamic global network infrastructure with self-configuring capabilities based on standard and interoperable communication protocols where physical and virtual "things" have identities, physical attributes, and virtual personalities and the intelligent interfaces, and are seamlessly integrated into the information network, often communicate data associated with users and their environment's

- **Dynamic and Self-Adapting:** IoT devices and systems may have the capability to dynamically adapt with the changing contexts and take actions based on their operating conditions, user context or sensed environment. For example, thesurveillance cameras can adapt their modes based on whether it is day or night. In this example, the surveillance system is adapting itself based on the context and changing (e.g., dynamic) conditions.
- **Self-Configuring:** IoT devices may have self-configuring capability, allowing a large number of devices to work together to provide certain functionality (such as weather monitoring). These devices have the ability configure themselves (in associate with the IoT Infrastructure), setup the networking, and fetch latest software upgrades with minimal manual or user intervention.
- **Interoperable Communication Protocols:** IoT devices may support a number of interoperable communication protocols and can communicate with other devices and also with the infrastructure.
- **Unique Identity:** Each IoT device has a unique identity and unique identifier (such as an IP address (or) a URI). IoT systems may have intelligent interface which adapt

based on the context, allow communicating with users and environmental contexts. IoT device interfaces allow users to query the devices, monitor their status, andcontrol them remotely in association with the control, configuration and management infrastructure.

- **Integrated into Information Network:** IoT devices are usually integrated into the information network that allows them to communicate and exchange data with other devices and systems. IoT devices can be dynamically discovered in the network, by other devices and/or the network, and have capability to describe themselves to other devices or user applications. For example, weather monitoring node can describe its monitoring capabilities to another connected node so that they can communicate and exchange data. Integration into the information network helps to make IoT systems "smarter" due to the collection intelligence of the individual devices in collaboration with the infrastructure.

PHYSICAL DESIGN OF IoT

Things in IoT

- The "Things" in IoT usually refers to IoT devices which have unique identities and can perform remote sensing, actuating and monitoring capabilities.
- IoT devices can exchange data with other connected devices and applications, (or) collect data from other devices and process the data locally (or) data to centralize severs (or) cloud-based application back-ends for processing the data.

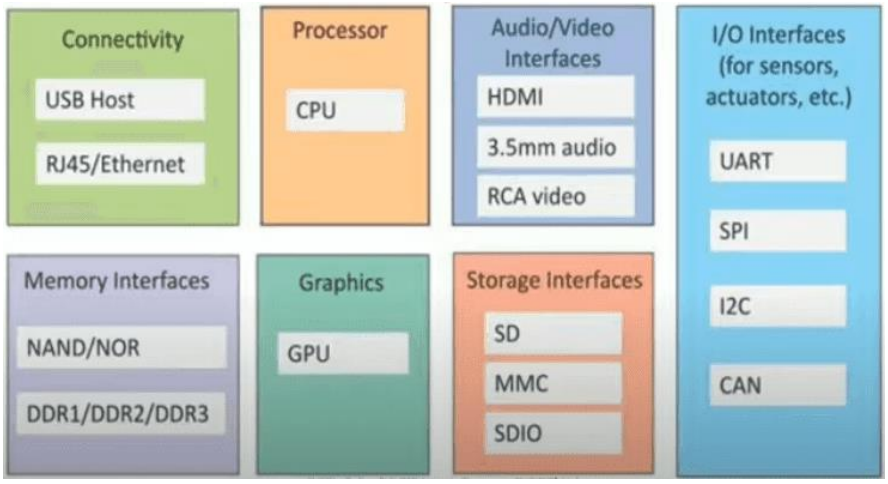


Fig : Generic block diagram of IoT device

- An IoT device may consist of several interfaces for connections to other devices, both wired and wireless.

These include

(i) I/O interfaces for sensors.

(ii) Interfaces for Internet connectivity

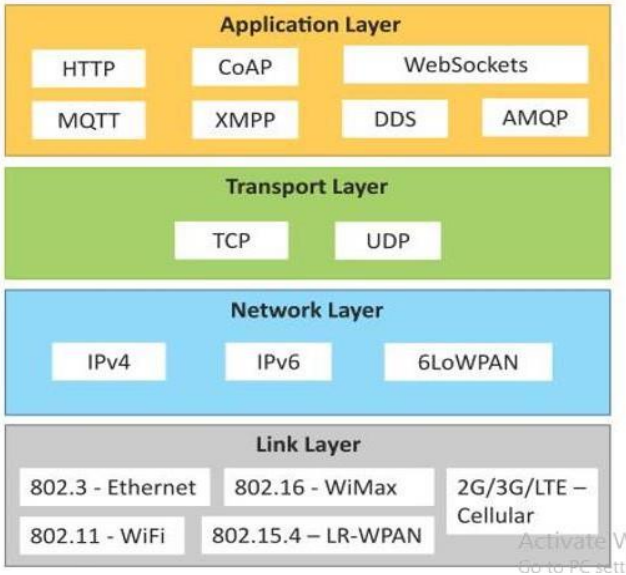
(iii) Memory & Storage interfaces and

(iv) Audio/ Video interfaces

- An IoT device can collect various types of data from the on-board or attached sensors, such as temperatures, humidity, and light intensity.
- The sensed data can be connected to actuators that allow them to interact with other physical entities in the vicinity of the device.
- IoT devices can also be of various types for instances, wearable sensors, smart watches , LED lights, automobile and industrial machines
- For instance, sensor data generated by a soil moisture device in a garden, when processed can help in determining the optimum watering schedules.

Fig : IoT devices

IOT Protocols



Link Layer

- Link Layer protocols determine how the data is physically sent over the network's physical layer or medium (e.g. copper wire, coaxial cable, or a radio wave).
- The scope of the link layer is the local network connection to which host is attached.
- Hosts on the same link exchange data packets over the link layer using link layer protocols.
- Link layer determines how the packets are coded and signalled by hardware devices over the medium to which the host is attached (such as coaxial cable).

Some link layer protocols are:

802.3-Ethernet:

- IEEE 802.3 is a collection of wired Ethernet standards for the link layer.
  For example,
  802.3 is the standard for 10BASE5 Ethernet that uses coaxial cable as a shared medium,802.3.i is the standard for 10BASE-T Ethernet over copper twisted-pair connections 802.3.j is the standard for 10BASE-F Ethernet over fiber optic .
  These standards provides data rates from 10Mb/s to 40G/b and higher.
- The shared medium in the Ethernet can be a coaxial cable, twisted- pair wire (or) an optical fiber. The shared medium (i.e., broadcast medium) carries the communication for all the devices on the network, thus data sent by one device can received by all devices subject to propagation conditions and transceiver capabilities.

802.11-wifi:

- IEEE 802.11 is a collection of wireless local area network (WLAN) communication standards including extensive description of the link layer.
  For example,
  802.11a operates in the 5HZ band, 802.11b and 802.11g operates in the 2.4 GHZ
  These standards provide data rates from 1Mb/s to upto 6.75Gb/s.

802.16-WiMax:

- IEEE 802.16 is a collection of wireless broadband standards including extensive descriptions for the link layer (also called WiMax).
- WiMax standard provide data rates from 1.5 Mb/s to 1Gb/s.
- The recent update of 802.16 is used to provides data rates of 100 Mb/s for mobile stations and 1Gb/s for fixed stations.

802.15.4 LR-WPAN

- IEEE 802.15.4 is a collection of standards for low-rate wireless personal area networks.
- These standards form the basis of specifications for high level communication protocols such as Zigbee.
- LR-WPAN standards provide data rates from 40 kb/s 250 kb/s.
- These standards provide low-cost and low-speed communication for power constrained devices.

2G/3G/4G- Mobile Communication

- There are different generations of mobile communication standards including second generation (2G including GSM and CDMA), third generation (3G including UMTS and CDMA2000) and fourth generation (4G including -LTE).
- IoT devices based on these standards can communication over cellular networks.
- Data rates for these standards range from 9.6 Kb/s (for 2G) to upto 100 Mb/s and are available from the 3GPP websites.

Network/Internet layer

- The network layers are responsible for sending of IP datagrams from source network to the destination network. This layer performs the host addressing and packet routing.
- The datagrams contain the source and destination addresses which one used to route them from the source to destination across multiple networks. Host Identification is done using hierarchical IP addressing schemes such as: IPV4 or IPV6.

IPV4

Internet protocol version4 (IPV4) is the most deployed Internet Protocol that is used to identify the devices on a network using a hierarchical addressing scheme. IPV4 uses a 32-bit address scheme that allows total of 2^{32} (or) 4,294,967,296 addresses. The protocols establish connection on packet networks, but do not guarantee delivery of packets. Guaranteed delivery and data integrity are handled by the upper layer protocols (such as TCP).

IPV6

Internet protocol version6 (IPV6) is the newest version of Internet protocol and successor to IPV4. IPV6 uses 128-bit address scheme that allows total of 2^{128} or 3.4×10^{38} addresses.

6LoWPAN

6LoWPAN (IPv6 over Low-Power Wireless Personal Area Networks) bring IP protocol to the low-power devices which have limited processing capability. 6LoWPAN operates in the 2.4GHz frequency range and provides data transfer rates of 250 Kb/s. 6LoWPAN works with the 802.15.4 link layer protocol and defines compression mechanismsfor IPV6 datagrams over IEEE 802.15.4-based networks.

Transport layer

The transport layer protocols provide end-to-end message transfer capability independent of the underlying network. The message transfer capability can be set up on connections, either using handshakes (as in TCP) or without handshakes/acknowledgements (as in UDP). The transport layer provides functions such as error control, segmentation, flow control and congestion control.

- **TCP :** Transmission Control Protocol (TCP) is the most widely used transport layer protocol, that is used by web browsers (along with HTTP, HTTPS application layer protocols), email programs (SMTP application layer protocol) and file transfer (FTP). TCP is a connection oriented and stateful protocol. While IP protocol deals with sending packets, TCP ensures reliable transmission of packets in-order. TCP also provides error detection capability so that duplicate packets can be discarded and lost

- **UDP:** UDP is connectionless protocol. UDP is useful for time-sensitive applications that have very small data units to exchange and do not want the overhead of connection setup. UDP is a transaction oriented and stateless protocol. UDP does not provide guaranteed delivery, ordering of messages and duplicate elimination. Higher levels of protocols can ensure reliable delivery or ensuring connections created are reliable.

Application Layer

Application layer protocols define how the applications interface with the lower layer protocols to send the data over the network. The application data, typically in files, is encoded by the application layer protocol and encapsulated in the transport layer protocol which provides connection or transaction oriented communication over the network. Port numbers are used for application addressing (for example port 80 for HTTP, port 22 for SSH, etc.). Application layer protocols enable process-to-process connections using ports.

- **HTTP :** Hypertext Transfer Protocol (HTTP) is the application layer protocol that forms the foundation of the World Wide Web (WWW). HTTP includes commands such as GET, PUT, POST, DELETE, HEAD, TRACE, OPTIONS, etc. The protocol follows a request-response model where a client sends requests to a server using the HTTP commands. HTTP is a stateless protocol and each HTTP request is independent of the other requests. An HTTP client can be a browser or an application running on the client (e.g., an application running on an IoT device, a mobile application or other software). HTTP protocol uses Universal Resource Identifiers (URIs) to identify HTTP resources.

- **CoAP :** Constrained Application Protocol (CoAP) is an application layer protocol for machine-to-machine (M2M) applications, meant for constrained environments with constrained devices and constrained networks. CoAP is a web transfer protocol and uses a request-response model, however it runs on top of UDP instead of TCP. CoAP uses a client-server architecture where clients communicate with servers using connectionless datagrams. CoAP is designed to easily interface with HTTP.
- **WebSocket :** WebSocket protocol allows full-duplex communication over a single socket connection for sending messages between client and server. WebSocket is based on TCP and allows streams of messages to be sent back and forth between the client and server while keeping the TCP connection open. The client can be a browser, a mobile application or an IoT device.
- **MQTT :** Message Queue Telemetry Transport (MQTT) is a light-weight messaging protocol based on the publish-subscribe model. MQTT uses a client-server architecture where the client (such as an IoT device) connects to the server (also called MQTT Broker) and publishes messages to topics on the server. The broker forwards the messages to the clients subscribed to topics. MQTT is well suited for constrained environments where the devices have limited processing and memory resources and the network bandwidth is low.

- **XMPP :** Extensible Messaging and Presence Protocol (XMPP) is a protocol for real-time communication and streaming XML data between network entities. XMPP powers wide range of applications including messaging, presence, data syndication, gaming, multi-party chat and voice/video calls. XMPP allows sending small chunks of XML data from one network entity to another in near real-time. XMPP is a decentralized protocol and uses a client-server architecture. XMPP supports both client-to-server and server-to-server communication paths. In the context of IoT, XMPP allows real-time communication between IoT devices.

- **DDS :** Data Distribution Service (DDS) is a data-centric middleware standard for device-to-device or machine-to-machine communication. DDS uses a publish-subscribe model where publishers (e.g. devices that generate data) create topics to which subscribers (e.g., devices that want to consume data) can subscribe. Publisher is an object responsible for data distribution and the subscriber is responsible for receiving published data. DDS provides quality-of-service (QoS) control and configurable reliability. DDS is described in Object Management Group (OMG) DDS specification.
- **AMQP :** Advanced Message Queuing Protocol (AMQP) is an open application layer protocol for business messaging. AMQP supports both point-to-point and publisher/subscriber models, routing and queuing. AMQP brokers receive messages from publishers (e.g., devices or applications that generate data) and route them over connections to consumers (applications that process data). Publishers publish the messages to exchanges which then distribute message copies to queues. Messages are either delivered by the broker to the consumers which have subscribed to the queues or the consumers can pull the messages from the queues.

1.3 Logical Design of IoT

Logical design of an IoT system refers to an abstract representation of the entities and processes without going into the low-level specifics of the implementation.

1.3.1 IoT Functional Blocks

An IoT system comprises of a number of functional blocks that provide the system the capabilities for identification, sensing, actuation, communication, and management

These functional blocks are described as follows:

- **Device :** An IoT system comprises of devices that provide sensing, actuation, monitoring and control functions.

- **Communication :** The communication block handles the communication for the IoT system.

- **Services :** An IoT system uses various types of IoT services such as services for device monitoring, device control services, data publishing services and services for device discovery.
- **Management :** Management functional block provides various functions to govern the IoT system.
- **Security** Security functional block secures the IoT system and by providing functions such as authentication, authorization, message and content integrity, and data security.
- **Application :** IoT applications provide an interface that the users can use to control and monitor various aspects of the IoT system. Applications also allow users to view the system status and view or analyze the processed data.

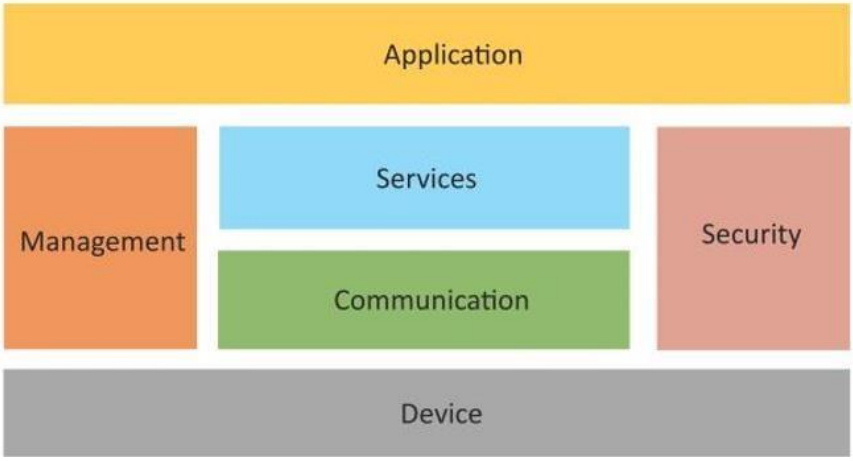


Fig: Functional Blocks Of IoT

1.3.2 IoT Communication Models

- **Request-Response :** Request-Response is a communication model in which the client sends requests to the server and the server responds to the requests. When the server receives a request, it decides how to respond, fetches the data, retrieves

resource representations, prepares the response, and then sends the response to the client. Request-Response model is a stateless communication model and each request-response pair is independent of others.

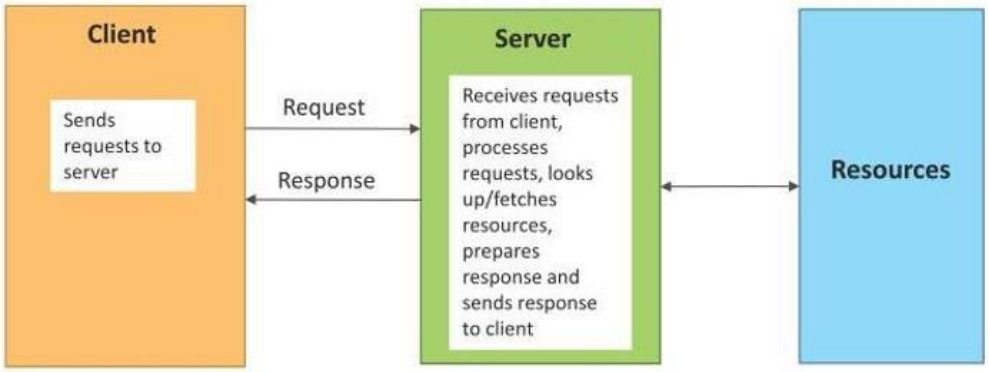


Fig: Request Response Communication Model

- **Publish-Subscribe :** Publish-Subscribe is a communication model that involves publishers, brokers and consumers. Publishers are the source of data. Publishers send the data to the topics which are managed by the broker. Publishers are not aware of the consumers. Consumers subscribe to the topics which are managed by the broker. When the broker receives data for a topic from the publisher, it sends the data to all the subscribed consumers.

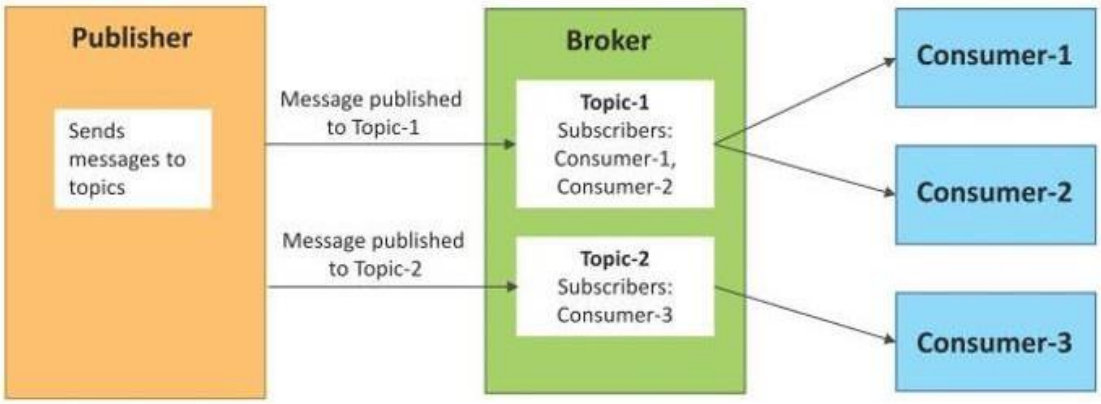


Fig: Publish-Subscribe Communication Model

- **Push-Pull :** Push-Pull is a communication model in which the data producers push the data to queues and the consumers pull the data from the queues. Producers do not need to be aware of the consumers. Queues help in decoupling the messaging between the producers and consumers. Queues also act as a buffer which helps in situations when there is a mismatch between the rate at which the producers push data and the rate rate at which the consumers pull data.

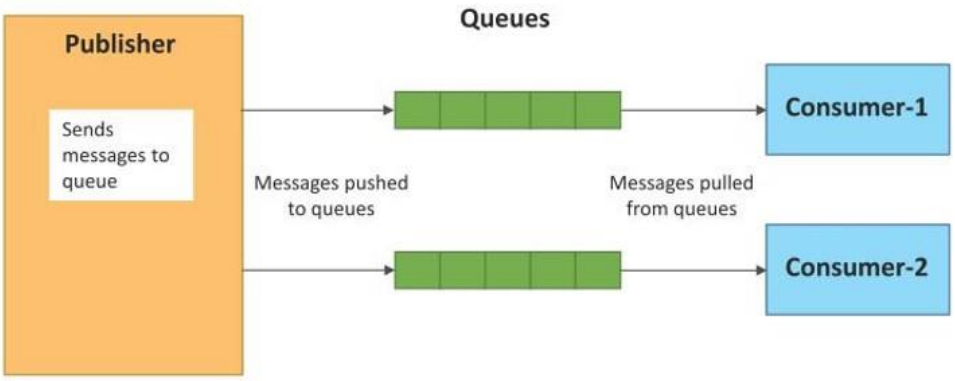


Fig: Push-Pull communication model

- **Exclusive Pair :** Exclusive Pair is a bi-directional, fully duplex communication model that uses a persistent connection between the client and server. Once the connection is setup it remains open until the client sends a request to close the connection. Client and server can send messages to each other after connection setup. Exclusive pair is a stateful communication model and the server is aware of all the open connections.

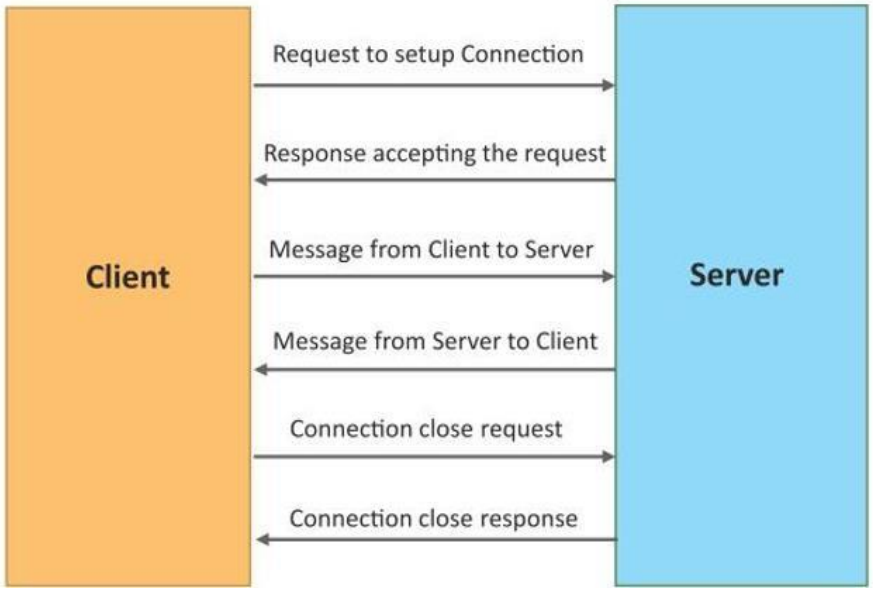


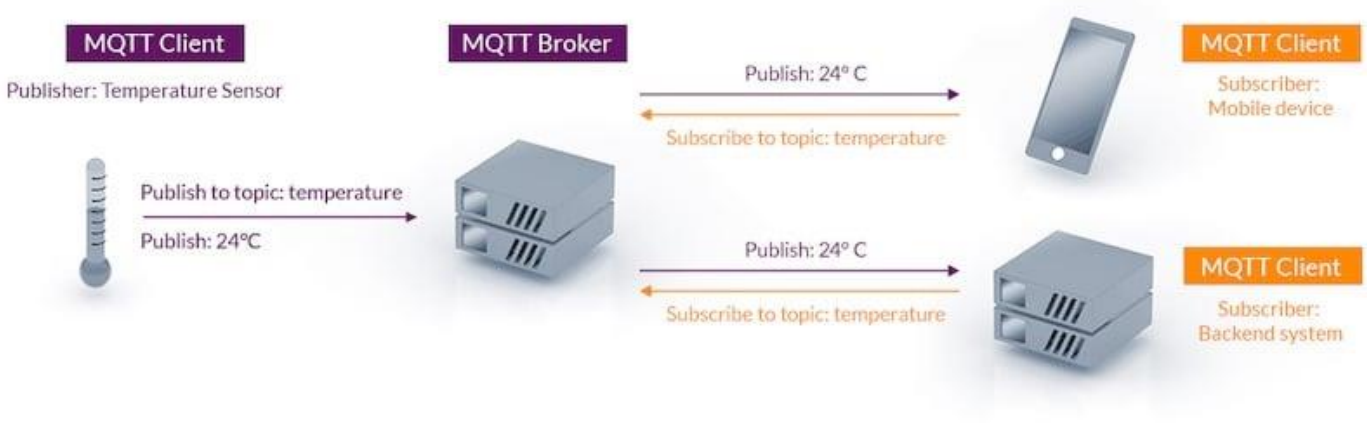
Fig: Exclusive-Pair Communication Model

Iot communication protocols:

Message Queue Telemetry Transport (MQTT)

Designed to be lightweight, so it can work in very low bandwidth networks, MQTT allows communication between nodes in both reliable and unreliable networks. MQTT follows a publish/subscribe architecture, meaning that there are nodes (brokers) that make the information available, while others (clients) can read the available information after subscribing by accessing the corresponding URL.

A use case of MQTT is in a smart factory where there are temperature sensors installed along with the production plant. The installed sensors will connect to the MQTT broker and will publish the data within sensor topics, as follows:



sensors/temperature/assemblyLineInit

Afterward, the MQTT clients, which can be of several types and quantities, will subscribe to the same topic in order to read the temperature data. An example of an MQTT architecture can be seen in Figure 1.

Figure 1. MQTT's publish/subscribe architecture. Image used courtesy of MQTT

in addition, MQTT defines three levels of quality of service, depending upon the reliability, from lowest to highest:

- Level 0: there is no guarantee of the message delivery.
- Level 1: the delivery is guaranteed, but it is possible to receive duplicate messages.
- Level 2: the delivery is guaranteed and there will be no duplicates.

HyperText Transfer Protocol (HTTP)

This protocol has been the origin of data communication for the World Wide Web (WWW), so logically it is being used in the IoT world. However, it is not optimized for it because of the following:

- The HTTP is made for two systems communicating to each other at a time, not more, so it is time and energy-consuming to connect several sensors to get information.
- The HTTP is unidirectional, made for one system (client) to be sending one message to another one (server). This makes it quite hard to escalate an IoT solution.
- Power consumption: HTTP relies on Transmission Control Protocol (TCP), which requires a lot of computing resources, so it is not suitable for battery-powered applications.

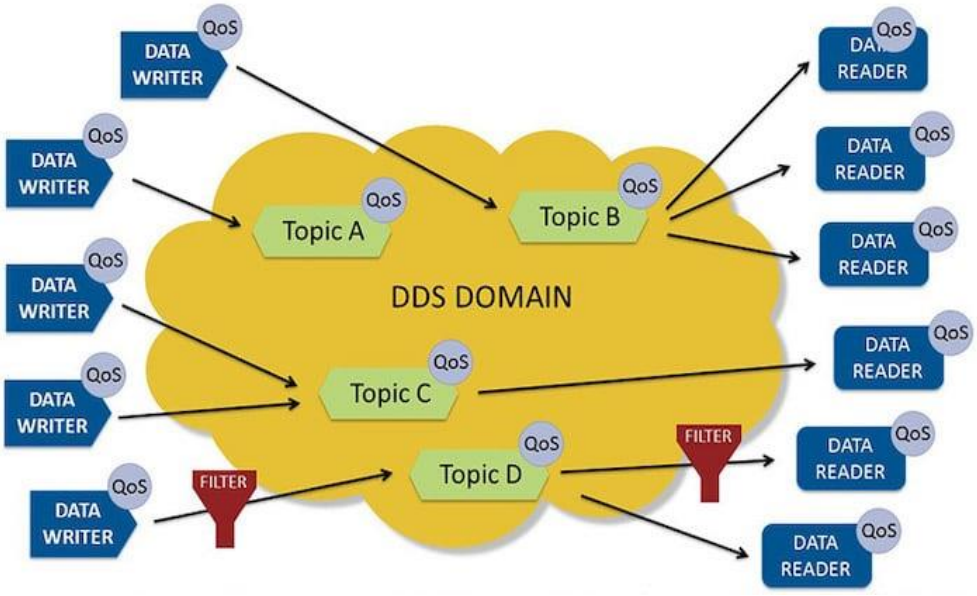
Constrained Application Protocol (CoAP)

CoAP is a web transfer protocol to be used with limited networks with low bandwidth and low availability. It follows a client/server architecture and is built similarly to HTTP, supporting the REST model: servers make resources available with an URL, and clients can make requests of types GET, POST, PUT and DELETE.

The CoAP communication links are 1:1 and UDP-based, so the delivery is not guaranteed. CoAP is made to work in highly congested networks, where nodes do not have a lot of intelligence and are not always working.

Data Distribution Service (DDS)

Similar to MQTT, DDS follows a publish-subscribe methodology, with the main difference being that there are no brokers. It means that all publishers (i.e., temperature sensors) and subscribers (i.e., mobile phones) are all connected to the same network. This network is known as Global Data Space (GDS) and it interconnects each node with all the other ones to avoid bottlenecks. An example of the DDS GDS can be seen in Figure 2.



WebSocket

Linked to the HTTP protocol, the WebSocket technology establishes a TCP connection between a browser and a server, and then both of them exchange information until the connection is closed. Figure 3 shows a high-level comparison between HTTP and WebSocket.

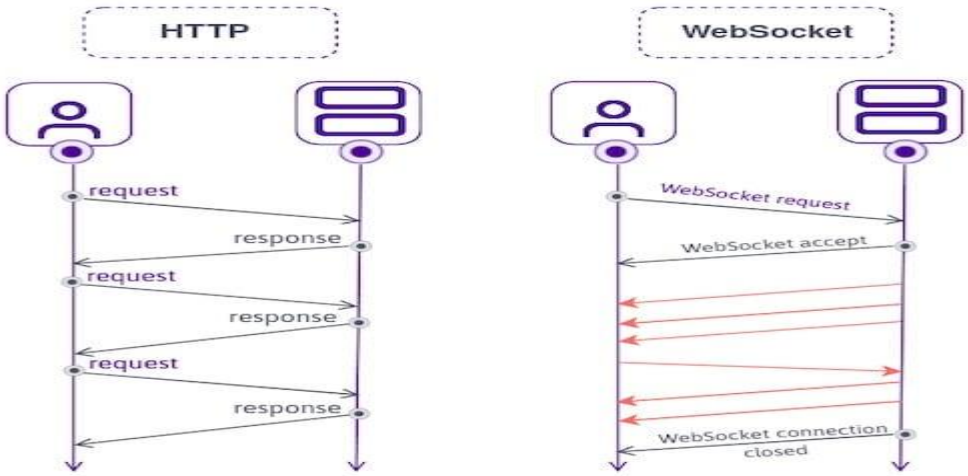


Figure 3. Comparison between HTTP and WebSocket. Image used courtesy of Scaleway

Although this protocol can be seen as an improvement of the HTTP connection, the WebSocket is still very overloaded and heavy for IoT applications.

Advanced Message Queue Protocol (AMQP)

In the beginning, AMQP was not initially created for IoT applications, but for banking environments. AMQP accepts publish/subscribe architectures, as well as request/response types. It is TCP-based, so delivery is guaranteed, as well as acknowledgment, which makes this protocol reliable, with the consequent overhead message reliability.

Compared to MQTT, AMQP offers two Quality of Service levels:

- At most once: the sender does not wait until having an acknowledgment from the receiver to delete a message.
- At least once: for each message, the sender will receive an acknowledgment from the receiver before deleting the message. In a case where the acknowledgment is lost, the message is re-sent.
- Exactly once: the messages are sent only once. It requires special coordination between the sender and the receiver.

Extensible Messaging and Presence Protocol (XMPP)

It is based on Extensible Markup Language (XML) and in the past, it was known as Jabber. It is an open-source, decentralized, secure protocol to exchange XML messages.

A characteristic factor of XMPP is its addressing method and how nodes are identified. It uses a Jabber ID with the format jabberID@domain.com, which allows two nodes to interchange information regardless of the distance between them.

1.3.3 IoT Communication API's

Iot communication API's is about varoius communication models.They are Two specific communication API's

REST-based Communication API's

- Representational State Transfer (REST) is a set of architectural principles to design web services and web API's that focus on a system's resources and resources states are addressed and transferred.
- REST API's follow the request response model.
- The REST architectural constraints apply to the components, connectors, and data elements, with in a distributed hypermedia system. The REST architectural constraints as follows

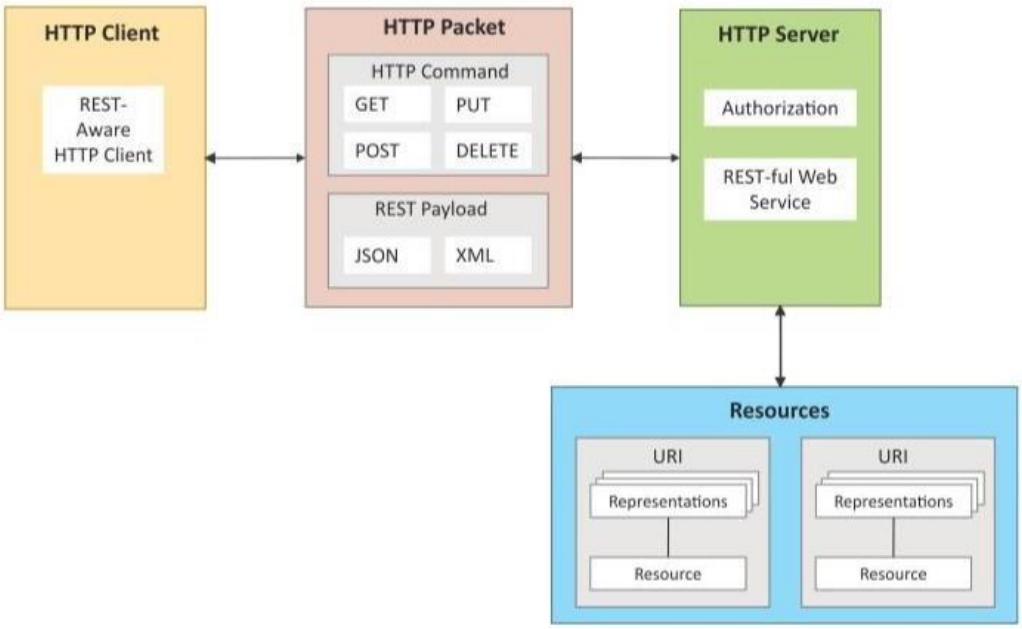
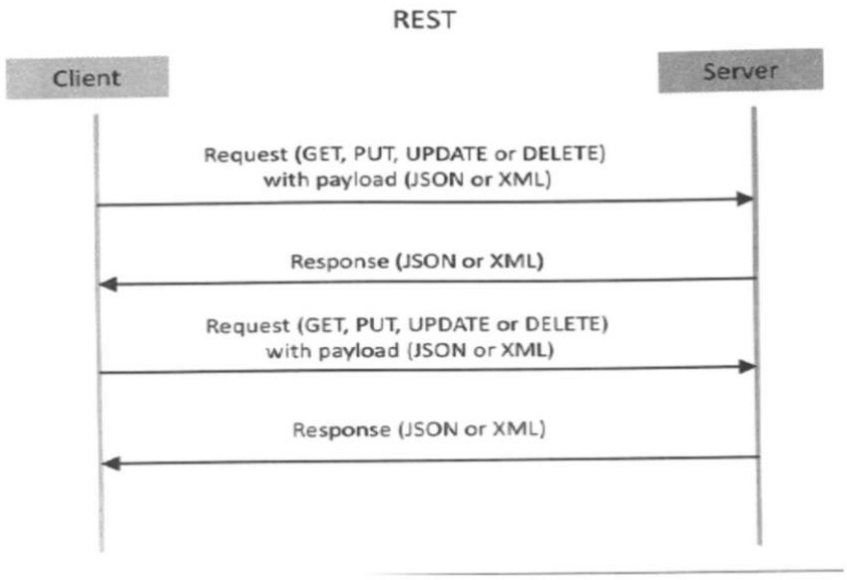


Fig: Communication with REST API's

- **Client-Server**: The principle behind the client-server constraint is the separation of concerns. For example, clients should not be concerned with the storage of data which is a concern of the server. Similarly, the server should not be concerned about the user interface, which is a concern of the client. Separation allows client and server to be independently developed and updated.
- **Stateless**: Each request from client to server must contain all the information necessary to understand the request, and cannot take advantage of any stored context on the server. The session state is kept entirely on the client.
- **Cache-able**: Cache constraint requires that the data within a response to a request be implicitly or explicitly labeled as cache-able or non-cache-able. If a response is cache-able, then a client cache is given the right to reuse that response data for later, equivalent requests. Caching can partially or completely eliminate some interactions and improve efficiency and scalability.



Request-response model used by REST

- **Layered System**: Layered system constraint, constrains the behavior of components such that each component cannot see beyond the immediate layer with which they are interacting. For example, a client cannot tell whether it is connected directly to the end server, or to an intermediary along the way. System scalability can be improved by allowing intermediaries to respond to requests instead of the end server, without the client having to do anything different.
- **Uniform Interface**: Uniform Interface constraint requires that the method of communication between a client and a server must be uniform. Resources are identified in the requests (by URIs in web based systems) and are themselves separate from the representations of the resources that are returned to the client. When a client holds a representation of a resource it has all the information required to update or delete the resource (provided the client has required permissions). Each message includes enough information to describe how to process the message.
- **Code on demand**: Servers can provide executable code or scripts for clients to execute in their context.

A RESTful web services is a "web API" implementing using HTTP and REST principles. RESTful web service is a collection of resources which are represented by URIs. RESTful web API has a base URI (eg. http://example.com/api/tasks). The client send requests to these URIs defined by the HTTP Protocol (eg., GET,PUT,POST or DELETE). IP for smart objects alliances (IPSO alliances) an application framework that defines a RESTful design for use in IP smart object system.

HTTP request methods and actions

HTTP Method	Resource Type	Action
GET	Collection URI	List all the resources in a collection
GET	Element URI	Get information about a resource
POST	Collection URI	Create a new resource
POST	Element URI	Generally not used
PUT	Collection URI	Replace the entire collection with another collection
PUT	Element URI	Update a resource
DELETE	Collection URI	Delete the entire collection
DELETE	Element URI	Delete a resource

WebSocket – Based Communication API's

WebSocket API's allow bi-directional, full duplex communication between client &servers. WebSocket APIs allow full duplex communication and do not require a new connection to be setup for each message to be sent. WebSocket communication begins with a connection setup request sent by the client to the server. This request (called a WebSocket handshake) is sent over HTTP and the server interprets it as an upgrade request. If the server supports WebSocket protocol, the server responds to the WebSocket handshake response. After the connection is setup, the client and server can send data/messages to each other in full-duplex mode. WebSocket APIs reduce the network traffic and latency as there is no overhead for connection setup and termination requests for each message. WebSocket is suitable for IoT applications that have low latency or high throughput requirements.

.

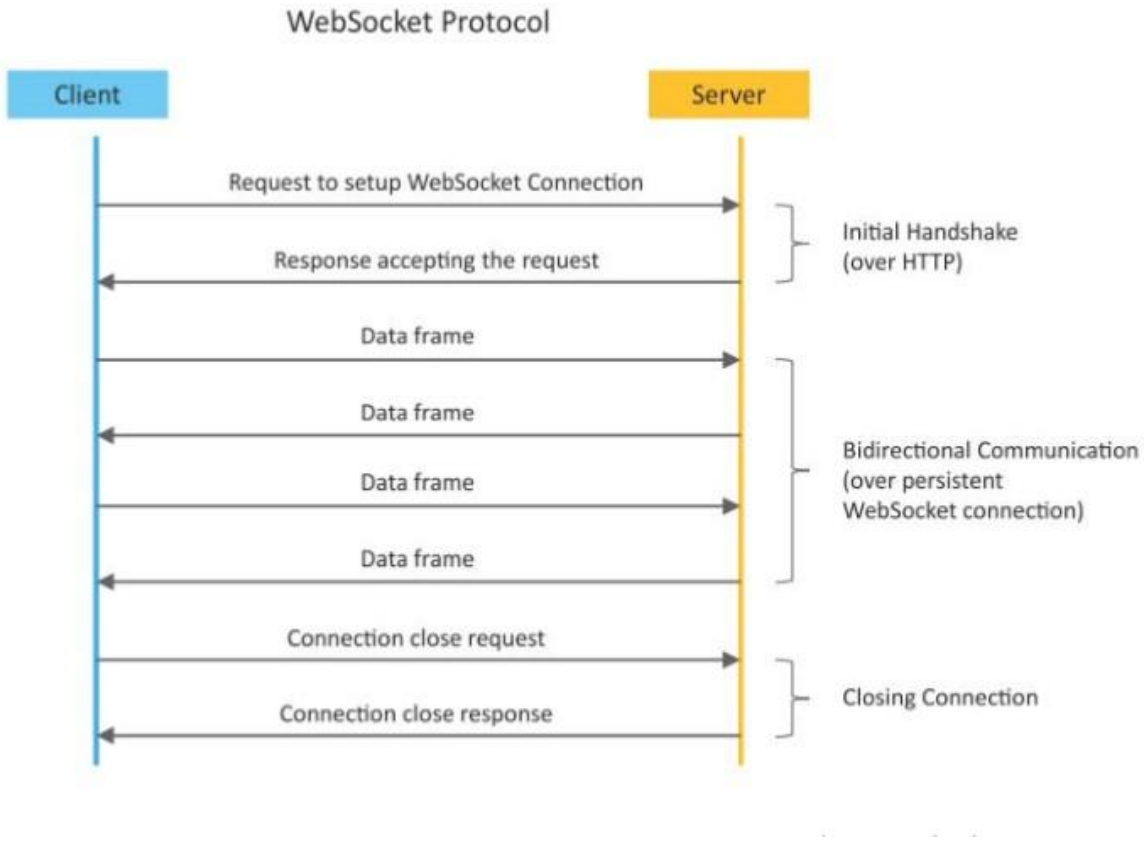


Fig: Exclusive Pair model used by WebSocket API's

1.4.4 Communication Protocols

Communication protocols form the backbone of IoT systems and enable network connectivity and coupling to applications. Communication protocols allow devices to exchange data over the network. The communication protocols are link, network, transport and application layer protocols. These protocols define the data exchange formats, data encoding, addressing schemes for devices and routing of packets from source to destination. Other functions of the protocols include sequence control (that helps in ordering packets determining lost packets), flow control (that helps in controlling the rate at which the sender is sending the data so that the receiver or the network is not overwhelmed) and retransmission of lost packets.

1.4.5 Embedded Systems

An Embedded System is a computer system that has computer hardware and software embedded to perform specific tasks. In contrast to general purpose computers or personal computers (PCs) which can perform various types of tasks, embedded systems are designed to perform a specific set of tasks. Key components of an embedded system include, microprocessor or microcontroller, memory (RAM, ROM, cache), networking units (Ethernet, WiFi adapters), input/output units (display, keyboard, etc.) and storage (such as flash memory).

Embedded means something that is attached to another thing. An embedded system can be thought of as a computer hardware system having software embedded in it. An embedded system can be an independent system or it can be a part of a large system. An embedded system is a microcontroller or microprocessor based system which is designed to perform a specific task. For example, a fire alarm is an embedded system; it will sense only smoke.

An embedded system has three components –

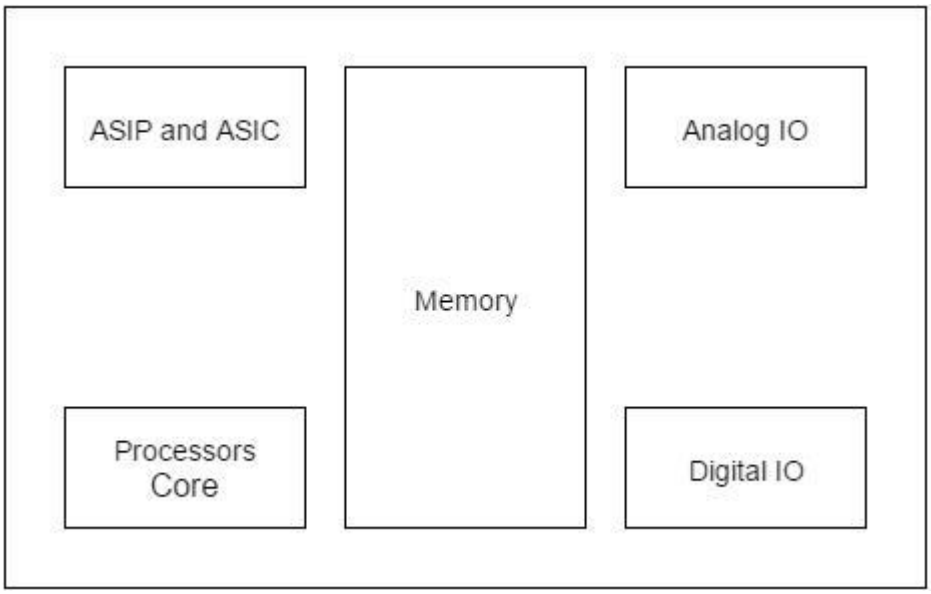
- It has hardware.
- It has application software.
- It has Real Time Operating system (RTOS) that supervises the application software and provide mechanism to let the processor run a process as per scheduling by following a plan to control the latencies. RTOS defines the way the system works. It sets the rules during the execution of application program. A small scale embedded system may not have RTOS.

So we can define an embedded system as a Microcontroller based, software driven, reliable, real-time control system.

Characteristics of an Embedded System

- **Single-functioned** – An embedded system usually performs a specialized operation and does the same repeatedly. For example: A pager always functions as a pager.
- **Tightly constrained** – All computing systems have constraints on design metrics, but those on an embedded system can be especially tight. Design metrics is a measure of an implementation's features such as its cost, size, power, and performance. It must be of a size to fit on a single chip, must perform fast enough to process data in real time and consume minimum power to extend battery life.

- **Reactive and Real time** – Many embedded systems must continually react to changes in the system's environment and must compute certain results in real time without any delay. Consider an example of a car cruise controller; it continually monitors and reacts to speed and brake sensors. It must compute acceleration or de-accelerations repeatedly within a limited time; a delayed computation can result in failure to control of the car.
- **Microprocessors based** – It must be microprocessor or microcontroller based.
- **Memory** – It must have a memory, as its software usually embeds in ROM. It does not need any secondary memories in the computer.
- **Connected** – It must have connected peripherals to connect input and output devices.
- **HW-SW systems** – Software is used for more features and flexibility. Hardware is used for performance and security.



Advantages

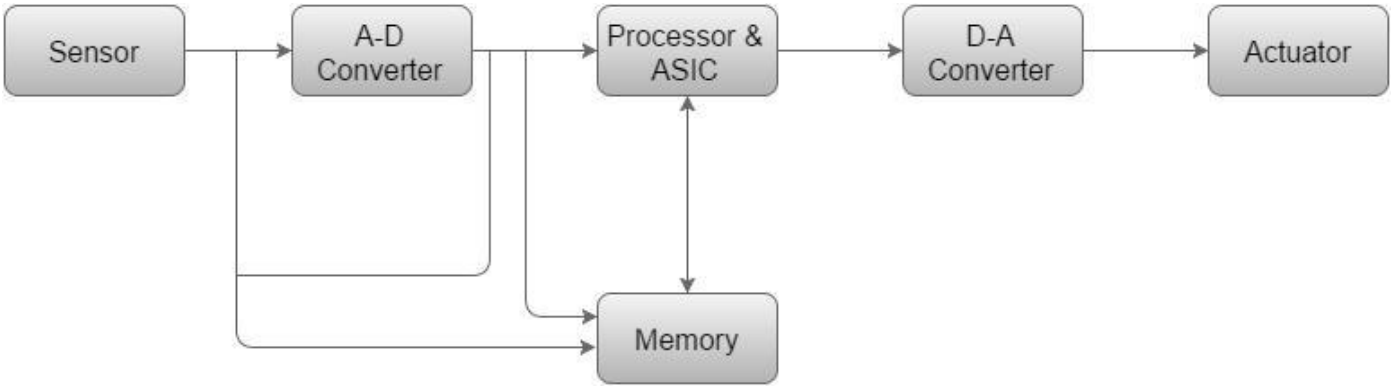
- Easily Customizable
- Low power consumption
- Low cost
- Enhanced performance

Disadvantages

- High development effort
- Larger time to market

Basic Structure of an Embedded System

The following illustration shows the basic structure of an embedded system –



- **Sensor** – It measures the physical quantity and converts it to an electrical signal which can be read by an observer or by any electronic instrument like an A2D converter. A sensor stores the measured quantity to the memory.
- **A-D Converter** – An analog-to-digital converter converts the analog signal sent by the sensor into a digital signal.
- **Processor & ASICs** – Processors process the data to measure the output and store it to the memory.
- **D-A Converter** – A digital-to-analog converter converts the digital data fed by the processor to analog data
- **Actuator** – An actuator compares the output given by the D-A Converter to the actual (expected) output stored in it and stores the approved output

1.5 IoT Levels & Deployment Templates

In this section we define various levels of IoT systems with increasing completely. An IoT system comprises of the following components:

- **Device:** An IoT device allows identification, remote sensing, actuating and remote monitoring capabilities. You learned about various examples of IoT devices in section 1.2.1.
- **Resource:** Resources are software components on the IoT device for accessing processing, and storing sensor information, or controlling actuators connected to the device. Resources also include the software components that enable network access for the device.
- **Controller Service:** Controller service is a native service that runs on the device and interacts with the web services. Controller service sends data from the device to the web service and receives commands from the application (via web services) for controlling the device.

- **Database:** Database can be either local or in the cloud and stores the data generated by the IoT device.
- **Web Service:** Web services serve as a link between the IoT device, application, database and analysis components. Web service can be either implemented using HTTP and REST principles (REST service) or using WebSocket protocol (WebSocket service). A comparison of REST and WebSocket is provided below:
  - **Stateless/Stateful:** REST services are stateless in nature. Each request contains all the information needed to process it. Requests are independent of each other. WebSocket on the other hand is stateful in nature where the server maintains the state and is aware of all the open connections.
  - **Uni-directional/Bi-directional:** REST services operate over HTTP and are uni-directional. Request is always sent by a client and the server responds to the requests. On the other hand, WebSocket is a bi-directional protocol and allows both client and server to send messages to each other.
  - **Request-Response/Full Duplex:** REST services follow a request-response communication model where the client sends requests and the server responds to the requests. WebSocket on the other hand allow full-duplex communication between the client and server, i.e., both client and server can send messages to each other independently.
  - **TCP Connections:** For REST services, each HTTP request involves setting up a new TCP connection. WebSocket on the other hand involves a single TCP connection over which the client and server communicate in a full-duplex mode.
  - **Header Overhead:** REST services operate over HTTP, and each request is independent of others. Thus each request carries HTTP headers which is an overhead. Due the overhead of HTTP headers, REST is not suitable for real-time applications. WebSocket on the other hand does not involve overhead of headers.

After the initial handshake (that happens over HTTP), the client and server exchange messages with minimal frame information. Thus WebSocket is suitable for real-time applications.

- **Scalability:** Scalability is easier in the case of REST services as requests are independent and no state information needs to be maintained by the server. Thus both horizontal (scaling-out) and vertical scaling (scaling-up) solutions are possible for REST services. For WebSockets, horizontal scaling can be cumbersome due to the stateful nature of the communication. Since the server maintains the state of a connection, vertical scaling is easier for WebSockets than horizontal scaling.

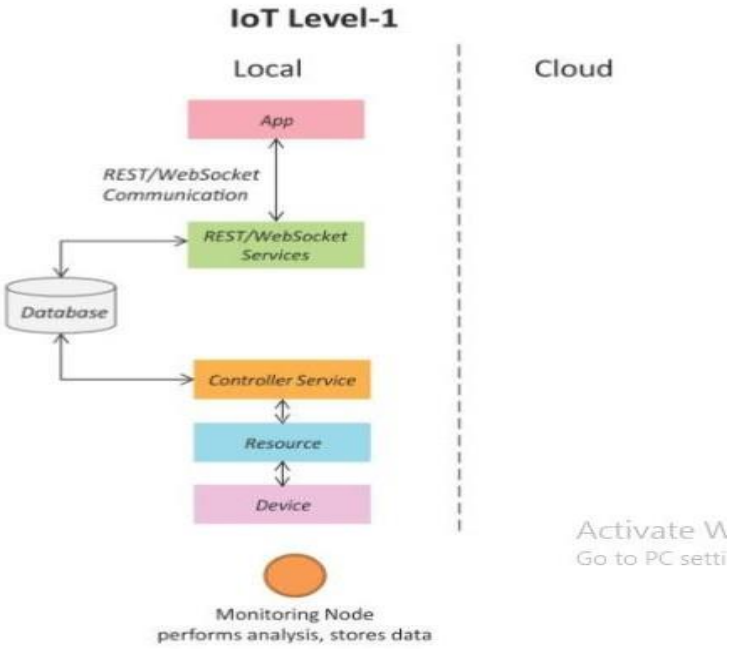
- **Analysis Component:** The Analysis Component is responsible for analyzing the IoT data and generate results in a form which are easy for the user to understand. Analysis of IoT data can be performed either locally or in the cloud. Analyzed results are stored in the local or cloud databases.

- **Application:** IoT applications provide an interface that the users can use to control and monitor various aspects of the IoT system. Applications also allow users to view the system status and view the processed data.

1.5.1 IoT Level-1

A level-1 IoT system has a single node/device that performs sensing and/or actuation, stores data, performs analysis and hosts the application. The Level-1 IoT systems are suitable for modeling low-cost and low-complexity solutions where the data involved is not big and the analysis requirements are not computationally intensive.

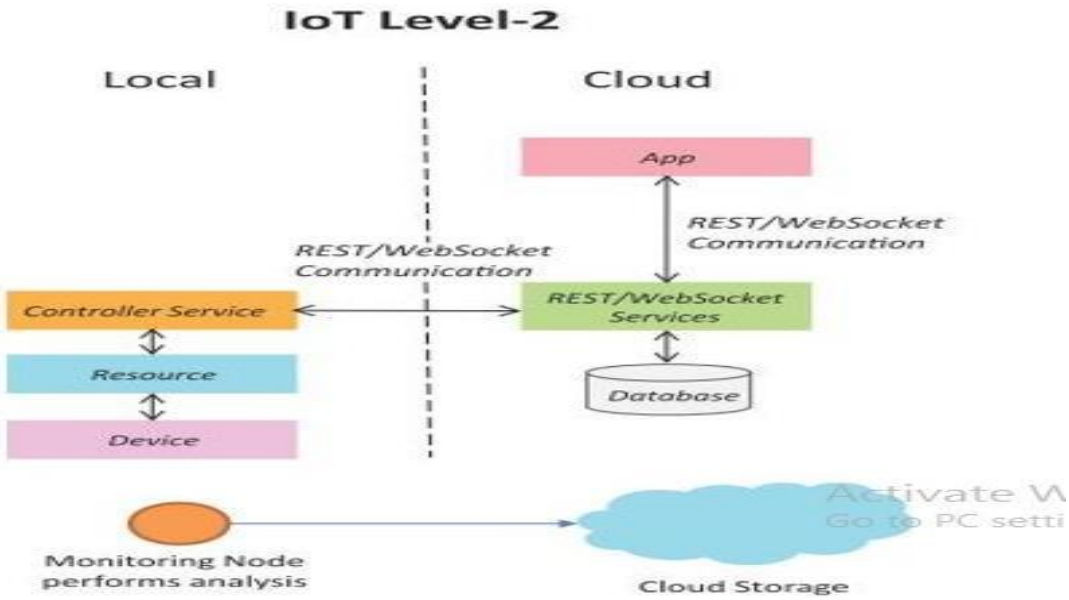
consider an example of a level-1 IoT system for home automation. The system consists of a single node that allows controlling the lights and appliances in a home remotely. The device used in this system interfaces with the lights and appliances using electronic relay switches. The status information of each light or appliance is maintained in a local database. REST services deployed locally allow retrieving and updating the state of each light or appliance in the status database. The controller service continuously monitors the state of each light or appliance (by retrieving state from the database) and triggers the relay switches accordingly. The application which is deployed locally has a user interface for controlling the lights or appliances. Since the device is connected to the Internet, the application can be accessed remotely as well.



1.5.2 IoT Level-2

A level-2 IoT system has a single node that performs sensing and/or actuation and local analysis Data is stored in the cloud and application is usually cloud-based. Level-2 IoT systems are suitable for solutions where the data involved is big, however, the primary analysis requirement is not computationally intensive and can be done locally itself.

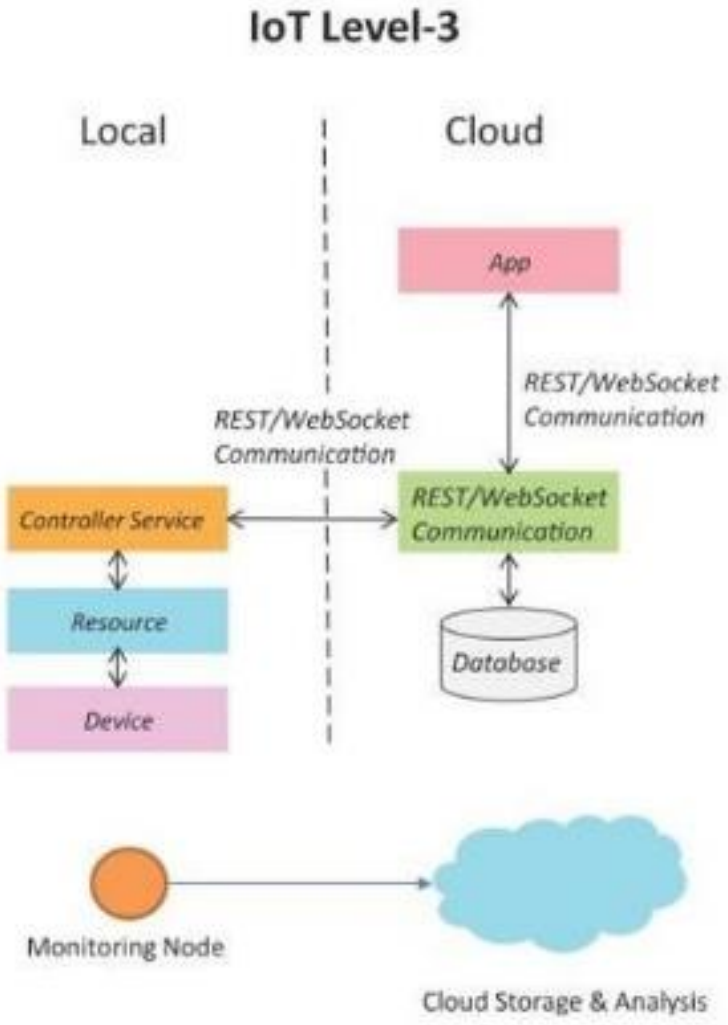
Let us consider an example of a level-2 IoT system for smart irrigation. The system consists of a single node that monitors the soil moisture level and controls the irrigation system. The device used in this system collects soil moisture data from sensors. The controller service continuously monitors the moisture levels. If the moisture level drops below a threshold, the irrigation system is turned on. For controlling the irrigation system actuators such as solenoid valves can be used. The controller also sends the moisture data to the computing cloud. A cloud-based REST web service is used for storing and retrieving moisture data which is stored in the cloud database. A cloud-based application is used for visualizing the moisture levels over a period of time, which can help in making decisions about irrigation schedules.



1.5.3 IoT Level-3

A level-3 IoT system has a single node. Data is stored and analyzed in the cloud and application is cloud-based Level-3 IoT systems are suitable for solutions where the data involved is big and the analysis requirements are computationally intensive.

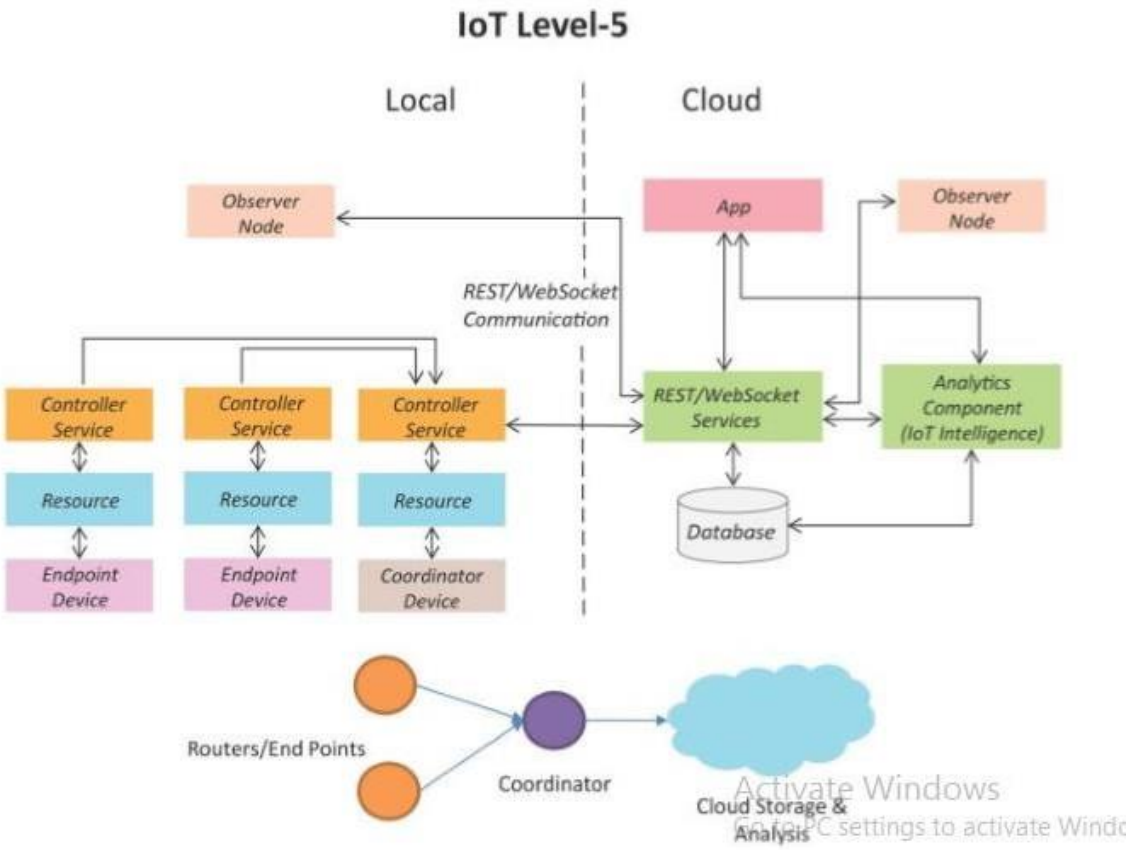
Let us consider an example of a level-2 IoT system for tracking package handling. The system consists of a single node (for a package) that monitors the vibration levels for a package being shipped. The device in this system uses accelerometer and gyroscope sensors for monitoring vibration levels. The controller service sends the sensor data to the cloud in real-time using a WebSocket service. The data is stored in the cloud and also visualized using a cloud-based application. The analysis components in the cloud can trigger alerts if the vibration levels become greater than a threshold. The benefit of using WebSocket service instead of REST service in this example is that, the sensor data can be sent in real time to the cloud. Moreover, cloud based applications can subscribe to the sensor data feeds for viewing the real-time data.

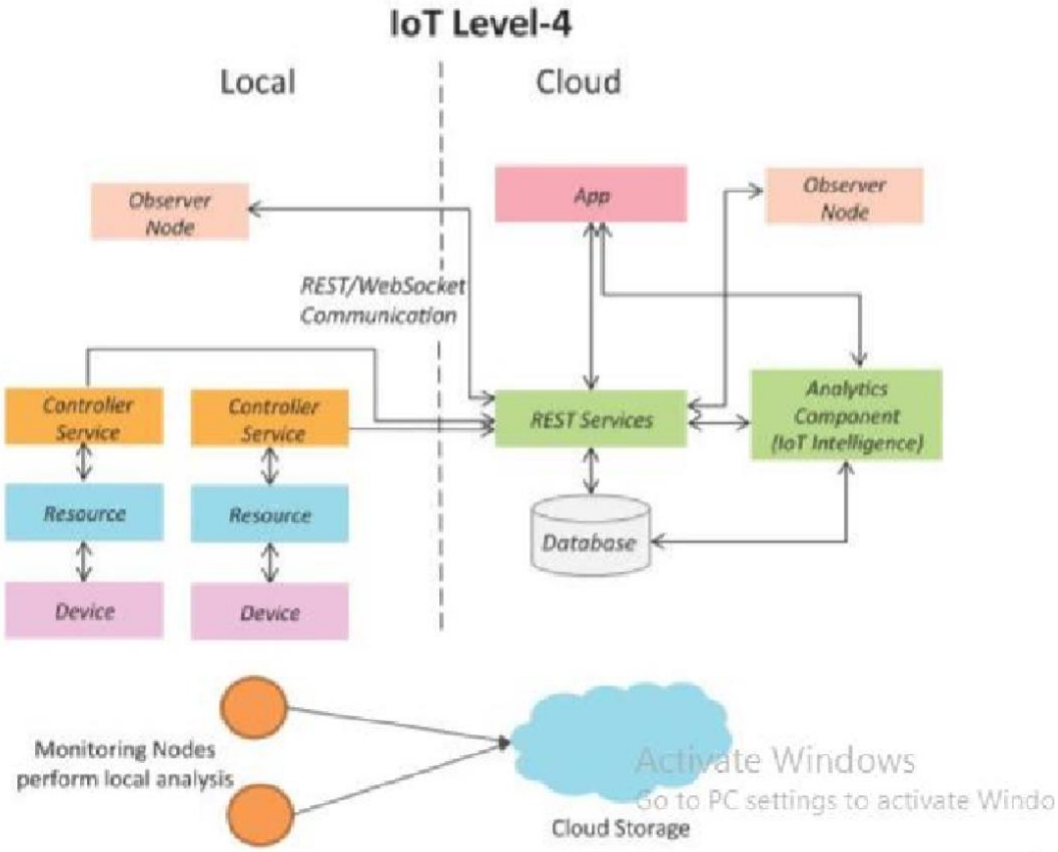


1.5.4 IoT Level-4

A level-4 IoT system has multiple nodes that perform local analysis. Data is stored in the cloud and application is cloud-based Level-4 contains local and cloud-based observer nodes which can subscribe to and receive information collected in the cloud from IoT devices. Observer nodes can process information and use it for various applications, however, observer nodes do not perform any control functions. Level-4 IoT systems are suitable for solutions where multiple nodes are required, the data involved is big and the analysis requirements are computationally intensive.

Let us consider an example of a level-4 IoT system for noise monitoring. The system consists of multiple nodes placed in different locations for monitoring noise levels in an area. The nodes in this example are equipped with sound sensors. Nodes are independent of each other. Each node runs its own controller service that sends the data to the cloud. The data is stored in a cloud database. The analysis of data collected from a number of nodes is done in the cloud. A cloud-based application is used for visualizing the aggregated data.





1.5.5 IoT Level-5

A level-5 IoT system has multiple end nodes and one coordinator node . The end nodes that perform sensing and/or actuation. Coordinator node collects data from the end nodes and sends to the cloud. Data is stored and analyzed in the cloud and application is cloud-based. Level-5 IoT systems are suitable for solutions based on wireless sensor networks, in which the data involved is big and the analysis requirements are computationally intensive.

Let us consider an example of a level-5 IoT system for forest fire detection. The system consists of multiple nodes placed in different locations for monitoring temperature, humidity and carbon dioxide (CO_2) levels in a forest. The end nodes in this example are equipped with various sensors (such as temperature, humidity and CO_2). The coordinator node collects the data from the end nodes and acts as a gateway that provides Internet connectivity to the IoT system. The controller service on the coordinator device sends the collected data to the cloud. The data is stored in a cloud database. The analysis of data is done in the computing cloud to aggregate the data and make predictions. A cloud-based application is used for visualizing the data.

UNIT 2

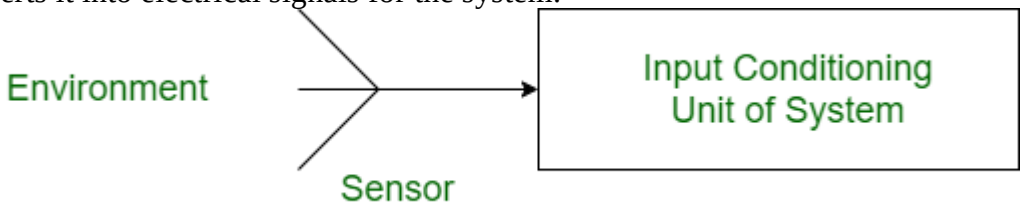
PROTOTYPING IOT OBJECTS USING MICRO PROCESSOR/MICRO CONTROLLER

Sensors and ***actuators*** are critical components of ***embedded systems***. Sensors and actuators differ primarily in their purpose; the sensor is utilized to track environmental changes using measurands, whereas the actuator is utilized when monitoring is combined with control, such as controlling physical changes.

SENSORS:

- A ***sensor*** is a device that detects changes and events in a physical environment. It may convert physical parameters like ***humidity, pressure, temperature, heat, motion, etc.***, into electrical signals.
- This signal can be converted into a human-readable display and sent across a network for additional processing.

A sensor works by sensing a quantity by utilizing a particular detecting device. Each sensor operates on a distinct principle, such as an electromagnetic sensor, a resistive sensor, a capacitor sensor, etc. In general, they sense the matching attribute in the environment and convert it into a proportional magnitude electrical signal. **1. Sensor:** Sensor is a device used for the conversion of physical events or characteristics into the electrical signals. This is a hardware device that takes the input from environment and gives to the system by converting it.
For example, a thermometer takes the temperature as physical characteristic and then converts it into electrical signals for the system.



TYPES OF SENSORS:

1. Active Sensor
2. Passive Sensor

- ***Active sensors*** necessitate a power supply, whereas ***passive sensors*** don't require a power supply.

  Eg: There are various types of sensors available, including temperature, ultrasonic, pressure, and location sensors, among others. They are utilized for detecting and measuring the relevant quantities.

Some important ***sensors*** are as follows:

1. Biosensors

These biosensors utilize electrochemical technology. These sensors are used in medical, food, and water testing devices. These biosensors also aid in analyzing proteins, cells, nucleic acid, etc.

2. Accelerometers

These sensors utilize the Micro Electro Mechanical Sensor Technology. These sensors utilize in patient monitoring, vehicle systems, etc.

3. Image Sensors

These sensors utilize the Complementary Metal Oxide Sensor mechanism. They detect and transfer data that is utilized to make an image. These image sensors are very useful in consumer surveillance and electronic systems.

3. **Chemical Sensors**

These sensors use ultrasonic, microwave, and radar technology, and they are used in security systems, video games, and other applications.

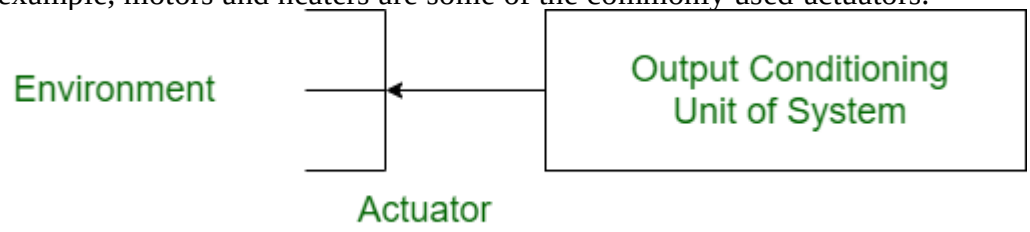
Features of Sensors

There are various features of ***Sensors***. Some main features of Sensors are as follows:

1. A sensor could be either active or passive. Active sensors necessitate a power source, but passive doesn't necessitate a power source.
2. It is a device that monitors and measures changes in the environment.
3. It is responsible for converting physical quantities into electrical signals.
4. It is connected to a system's input.
5. It generates an electrical signal as its output.

2. Actuator:

Actuator is a device that converts the electrical signals into the physical events or characteristics. It takes the input from the system and gives output to the environment. For example, motors and heaters are some of the commonly used actuators.



- A device that changes electrical signals into mechanical work is known as an ***actuator***. It is used to cause movement or a change in the surroundings.

- ***Actuators*** are connected to a system's output. It receives an electrical signal as input and produces mechanical movement as output. It receives input or instruction from a system or a signal conditioning device and outputs it to the environment.
- The ***actuator*** is dependent on the sensor data. The sensor sends data to a signal condition unit, which analyzes the data or information and transmits commands to the actuator depending on that data. A ***"temperature control system"*** is an instance of an actuator system in which a temperature sensor manages the temperature. If the temperature surpasses a specific limit, the device instructs the fan to increase its speed and decrease the temperature.

Types of Actuators

There are various types of ***actuators***. Some of these are as follows:

1. Manual Actuator

This type of actuator is manually operated via gears, levers, and wheels, among other things. They do not need a power source because they are powered by human action.

2. Spring Actuator

It has a loaded spring that is triggered and released to generate mechanical work. It may be triggered in several ways.

3. Hydraulic Actuator

Hydraulic actuators generate pressure by compressing fluid in a cylinder, allowing mechanical movement.

4. Electric Actuators

These actuators require power to function. It utilizes an electric motor to produce movement. They are quick and effective.

Features of Actuators

There are various features of ***Actuators***. Some main features of Actuators are as follows:

1. The actuator assists in managing the environment based on sensor readings.
2. A device that converts electrical signals into mechanical movement is known as an actuator.
3. It requires an additional power source to function.
4. It receives an electrical signal as input.

5. It is connected to a system's output.

It produces mechanical work. **Difference between Sensor and Actuator :**

SENSOR	ACTUATOR
It converts physical characteristics into electrical signals.	It converts electrical signals into physical characteristics.
It takes input from environment.	It takes input from output conditioning unit of system.
It gives output to input conditioning unit of system.	It gives output to environment.
Sensor generated electrical signals.	Actuator generates heat or motion.
It is placed at input port of the system.	It is placed at output port of the system.
It is used to measure the physical quantity.	It is used to measure the continuous and discrete process parameters.
It gives information to the system about environment.	It accepts command to perform a function.
Example: Photo-voltaic cell which converts light energy into electrical energy.	Example: Stepper motor where electrical energy drives the motor.

Setting up the board-programming for iot:

Setting up the board programming for IoT involves several steps, depending on the specific board you are using and the programming language you want to use. Here are some general steps you can follow:

1. Choose your board: There are many IoT boards available, such as Arduino, Raspberry Pi, ESP8266, etc. Choose a board that fits your project requirements.
2. Install the necessary software: You will need to install the necessary software on your computer to program the board. For example, if you are using an Arduino board, you will need to install the Arduino IDE.
3. Connect your board: Connect your board to your computer using a USB cable or another suitable method.
4. Write your code: Write your code in a programming language supported by your board. For example, Arduino boards use a version of C++, while Raspberry Pi boards can use Python or other languages.
5. Upload your code: Upload your code to the board using the software you installed earlier. This will program the board to execute your code.

6. Test your code: Test your code by connecting sensors or other devices to the board and seeing if it works as intended.
7. Deploy your project: Once your code is working correctly, deploy your project to the intended location, such as a remote server or IoT device.

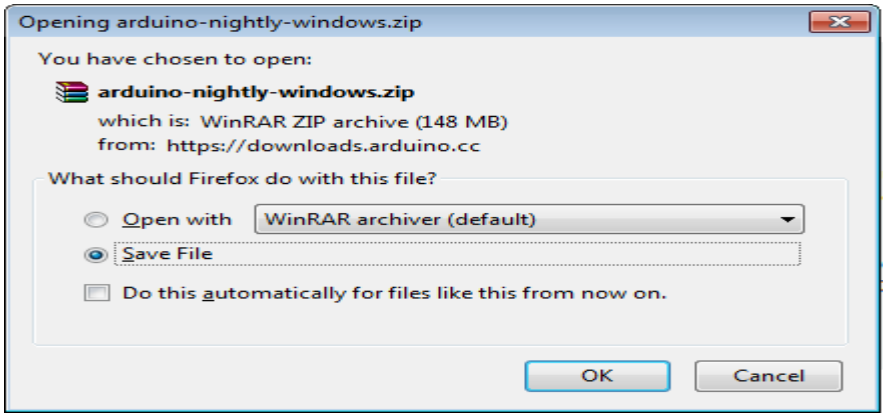
These are the general steps involved in setting up the board programming for IoT. However, the specifics of each step will vary depending on your board, programming language, and project requirements.

Installation of Arduino UNO:

After learning about the main parts of the Arduino UNO board, we go for how to set up the Arduino IDE. we will be ready to upload our program on the Arduino board.our computer and prepare the board to receive the program via USB cable.

Step 1 − First you must have your Arduino board and a USB cable. In case you use Arduino UNO, Arduino Duemilanove, Nano, Arduino Mega 2560, or Diecimila, you will need a standard USB cable (A plug to B plug), the kind you would connect to a USB printer as shown in the following image.

In case you use Arduino Nano, you will need an A to Mini-B cable instead as shown in the following image.



Step 3 − Power up your board.

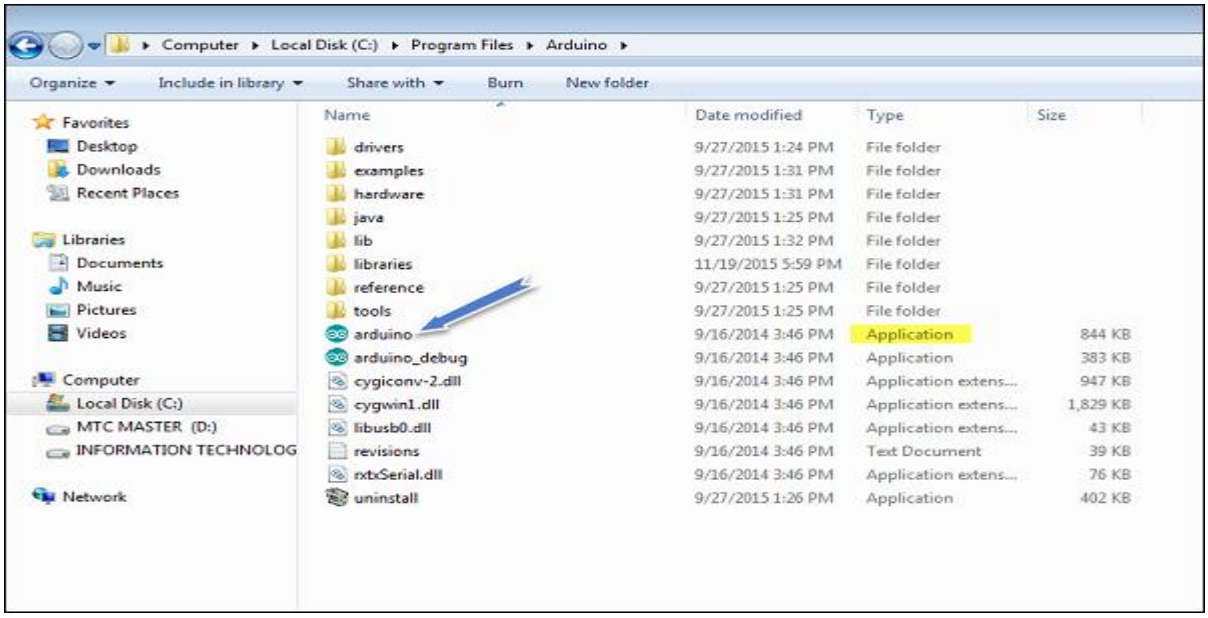
The Arduino Uno, Mega, Duemilanove and Arduino Nano automatically draw power from either, the USB connection to the computer or an external power supply. The power source is

selected with a jumper, a small piece of plastic that fits onto two of the three pins between the USB and power jacks. Check that it is on the two pins closest to the USB port.

Connect the Arduino board to your computer using the USB cable. The green power LED should glow.

Step 4 − Launch Arduino IDE.

After Arduino IDE software is downloaded, you need to unzip the folder. Inside the folder, you can find the application icon with an infinity label (application.exe). Double-click the icon to start the IDE.

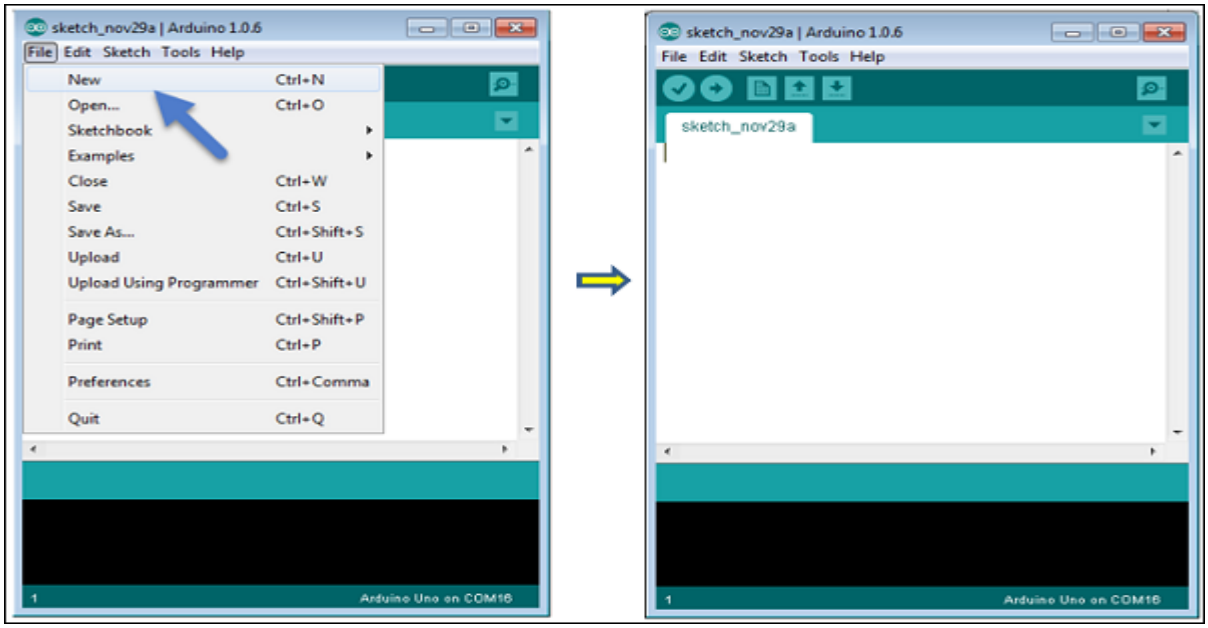


Step 5 − Open your first project.

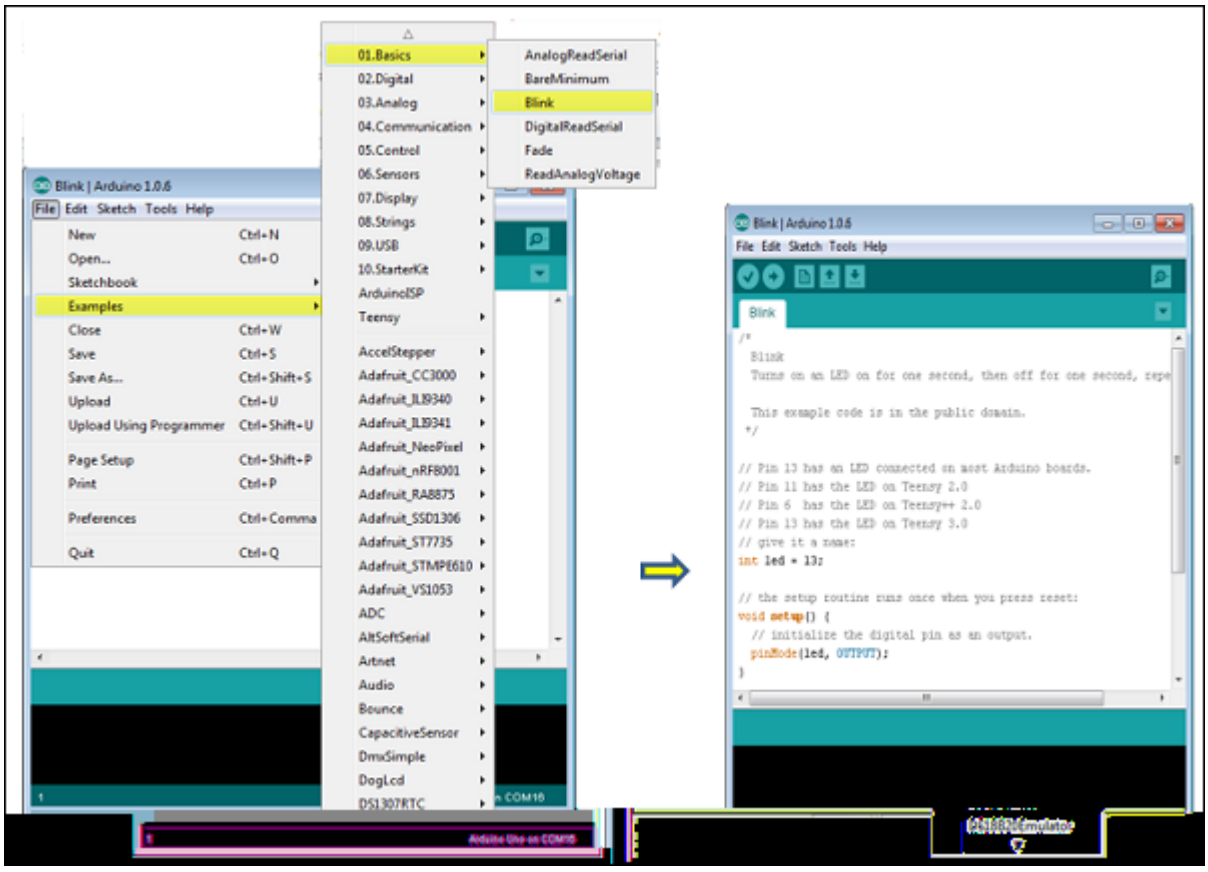
Once the software starts, you have two options −

- Create a new project.
- Open an existing project example.

To create a new project, select File → **New**.



To open an existing project example, select File → Example → Basics → Blink.

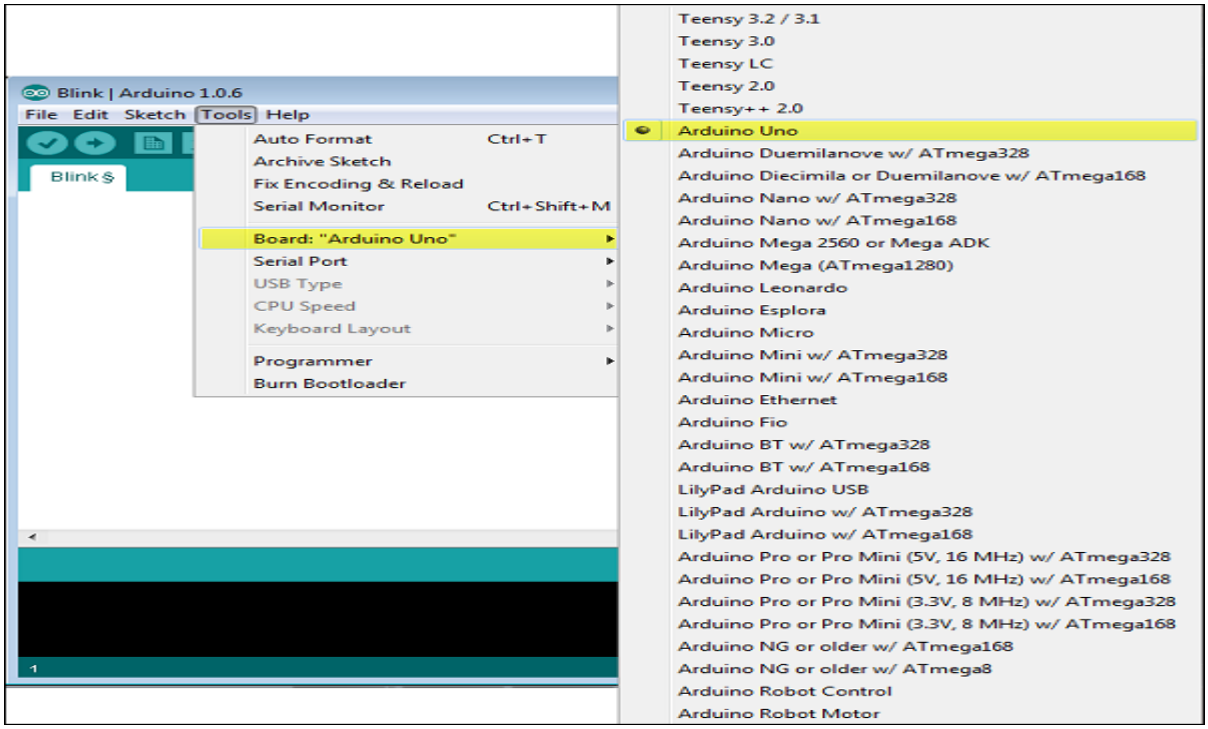


Here, we are selecting just one of the examples with the name **Blink**. It turns the LED on and off with some time delay. You can select any other example from the list.

Step 6 − Select your Arduino board.

To avoid any error while uploading your program to the board, you must select the correct Arduino board name, which matches with the board connected to your computer.

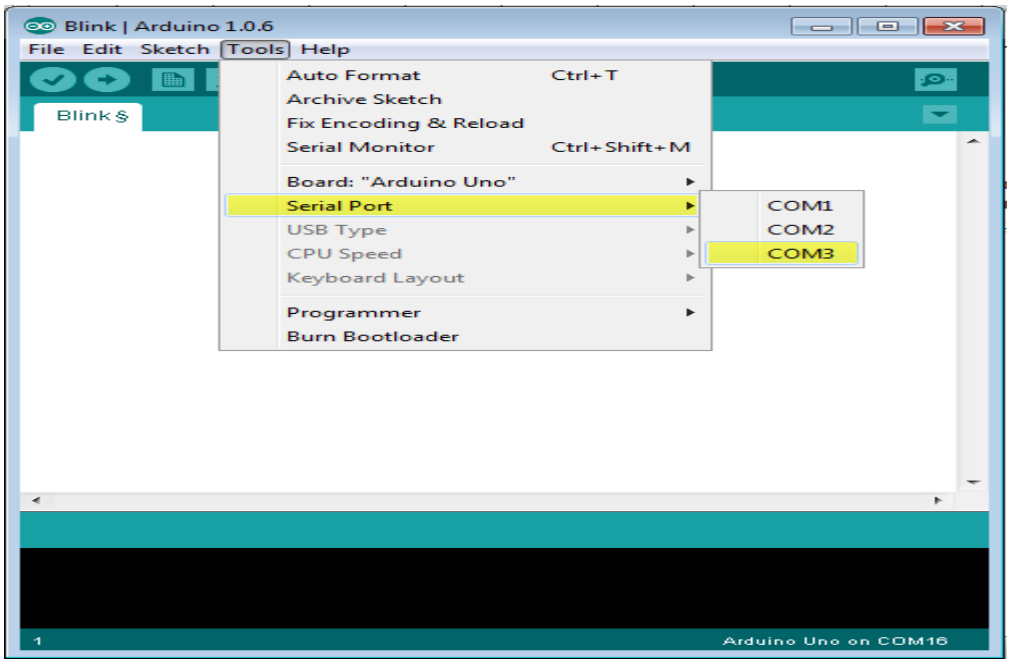
Go to Tools → Board and select your board.



Here, we have selected Arduino Uno board, but you must select the name matching the board that you are using.

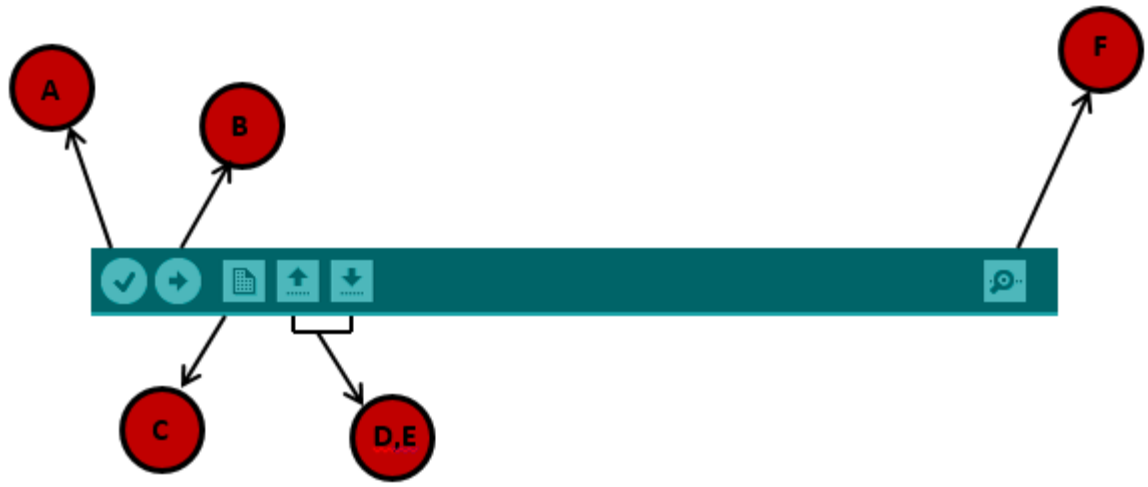
Step 7 − Select your serial port.

Select the serial device of the Arduino board. Go to **Tools → Serial Port** menu. This is likely to be COM3 or higher (COM1 and COM2 are usually reserved for hardware serial ports). To find out, you can disconnect your Arduino board and re-open the menu, the entry that disappears should be of the Arduino board. Reconnect the board and select that serial port.



Step 8 − Upload the program to your board.

Before explaining how we can upload our program to the board, we must demonstrate the function of each symbol appearing in the Arduino IDE toolbar.



A − Used to check if there is any compilation error.

B − Used to upload a program to the Arduino board.

C − Shortcut used to create a new sketch.

D − Used to directly open one of the example sketch.

E − Used to save your sketch.

F − Serial monitor used to receive serial data from the board and send the serial data to the board.

Now, simply click the "Upload" button in the environment. Wait a few seconds; you will see the RX and TX LEDs on the board, flashing. If the upload is successful, the message "Done uploading" will appear in the status bar.

Reading data from sensors:

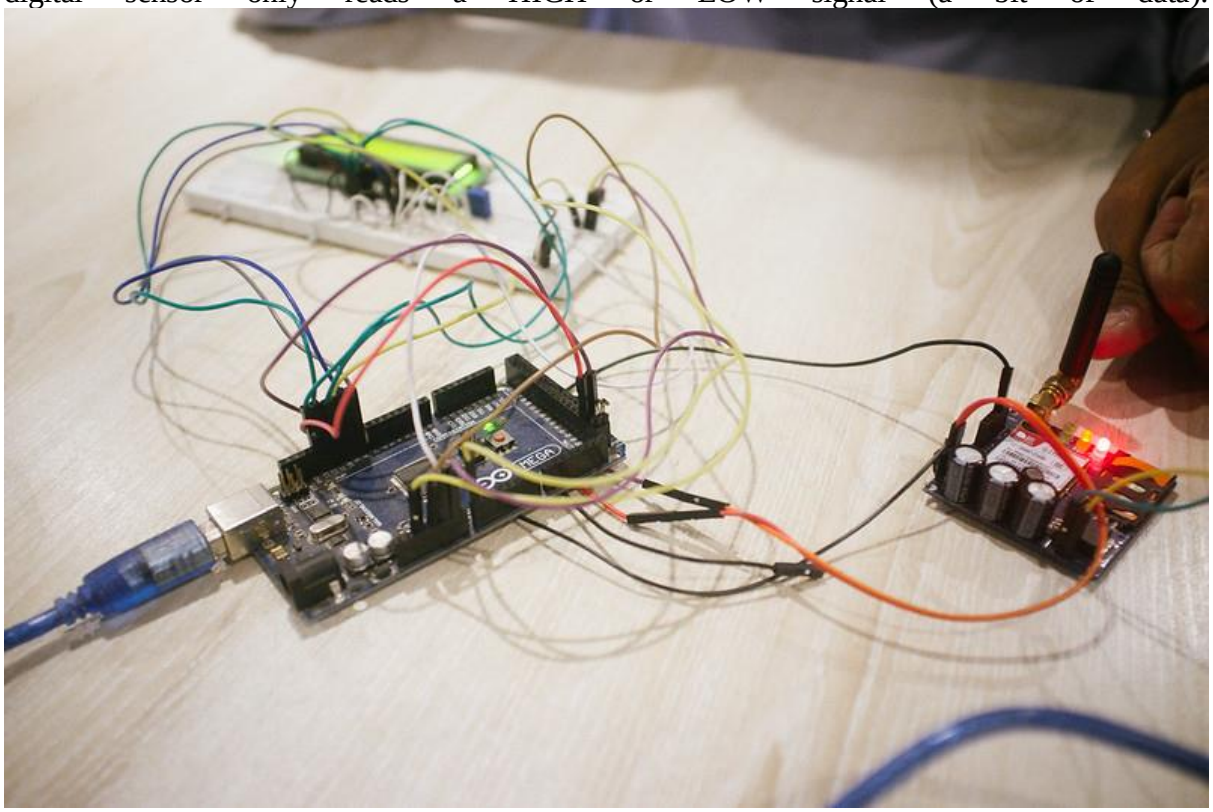
Reading from a sensor typically involves the following steps:

1. Connect the sensor: The first step is to connect the sensor to the device that will be reading its output. This could be a microcontroller, a single-board computer, or any other device capable of reading analog or digital signals.
2. Configure the device: Once the sensor is connected, the device must be configured to read the sensor's output. This may involve setting up communication protocols, such as I2C or SPI, and configuring the input pins or channels on the device to receive the sensor's output.
3. Read the sensor: Once the device is configured, it can begin reading the sensor's output. Depending on the type of sensor and the device being used, this may involve polling the sensor for data or receiving a continuous stream of data from the sensor.
4. Process the data: Once the data has been read from the sensor, it may need to be processed or analyzed in some way. For example, if the sensor is measuring temperature, the raw data may need to be converted into a meaningful temperature reading using a formula or lookup table.

5. Use the data: Finally, once the data has been processed, it can be used for whatever purpose it was intended. This could be as simple as displaying the data on a screen or as complex as using the data to control a system or make decisions in real-time.

How to read Analog Sensors using Arduino

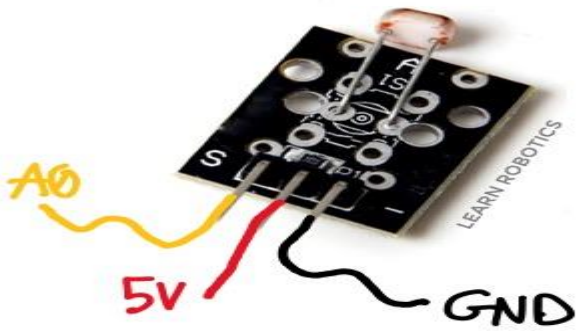
The Arduino has built-in analog and digital input and output (I/O) pins The difference between analog and digital sensors is that an analog sensor collects readings over a range of values, and a digital sensor only reads a HIGH or LOW signal (a bit of data).



The Arduino has a 10-bit Analog-to-Digital-Converter (ADC), which maps sensor readings between 0 and the operating voltage (5V or 3.3V) into integer values between 0 and 1023.

Step 1. Wire Analog Sensors to Arduino

The majority of analog sensors for Arduino are wired and programmed the same. So, once you learn how to wire and read data from one analog sensor, you'll be able to wire and program thousands of additional sensors to collect a whole bunch of data. For this example, I'll walk you through wiring and programming a light-dependent resistor (LDR) also known as a photoresistor.



The first step is to connect the analog sensor to the Arduino. Analog sensors for Arduino have three wires (Ground, Signal, Power). Then, connect the ground wire to GND on the Arduino. Next, attach the Signal wire to an analog pin on the Arduino. Lastly, connect the power wire to the 5V on the Arduino

Step 2. Setup your Arduino Sketch

The next step is to set up the Arduino Sketch. First, configure a global variable for the analog sensor.

```
int ldr = A0;
```

Then, in the setup() method, initialize the sensor as an input and start the Serial monitor.

```
void setup(){
   pinMode(ldr, INPUT); //initialize ldr sensor as INPUT
   Serial.begin(9600); //begin the serial monitor at 9600 baud
}
```

The Arduino Uno has a baud rate of 9600. We can use the begin method to start the Serial Monitor. Now, we're ready to write the Arduino code to collect readings from our analog sensor.

Step 3. Write code to collect readings from Analog Sensors

Next, collect a sensor reading using the analogRead(ldr) method, and store it in an integer variable. I called this variable "data."

We will use a few print statements to show the readings in the Serial Monitor. Serial.print() will print data horizontally across the screen. Serial.println() will print data vertically down the screen. I used both to label the data while making it easy to read.

```
void loop(){
   int data=analogRead(ldr);
   Serial.print("ldr reading=");
   Serial.println(data);
   delay(1000);
}
```

Finally, add a delay. This prevents the Arduino from taking readings faster than we can see them. Feel free to adjust this delay to whatever interval makes sense for your application. Once you have the test code written, save the sketch and upload it to the Arduino. Open up the serial monitor and you should see values from 0-1023 depending on how bright or dark the area is.

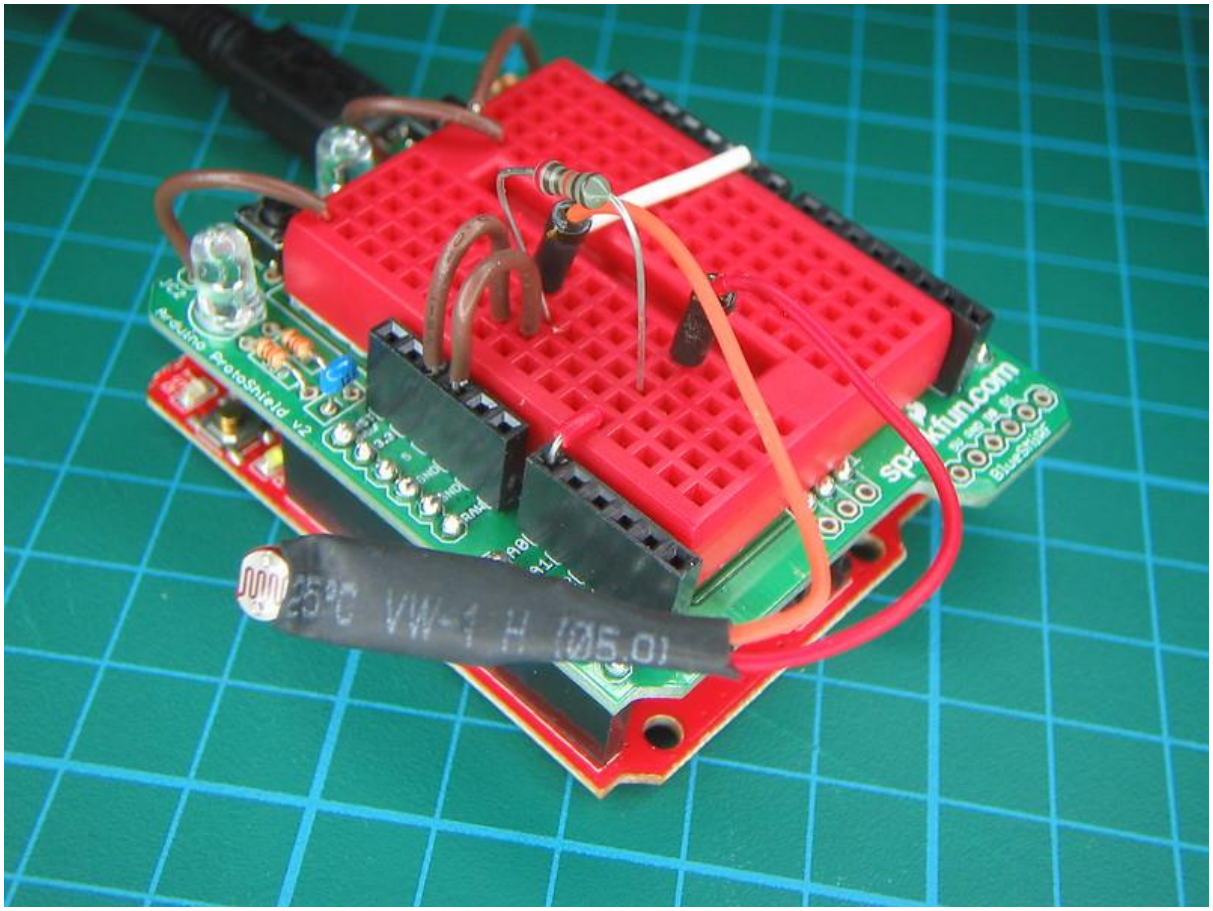
Step 4. Analyze and Convert Sensor Readings as needed

Some sensors require a unit conversion. For example, it's easier to understand what temperature it is when the units are in Celcius or Fahrenheit. Furthermore, with our LDR we could convert the unit readings into a brightness percentage. That way when we analyze the data, we can check for conditions based on 25% bright, 100% bright, or a unit that makes more sense for the application.

Most datasheets specify formulas that you can use to make these conversions. You don't always have to "make up" a unit for your sensor.

Step 5. Use sensor data to make decisions

Once you have an understanding of how data is collected from your analog sensor, you can use the readings to make decisions.



We'll use conditional statements to check to see if a condition is TRUE or FALSE. Then, based on that condition, we'll react accordingly.

Here's an example using the LDR. If the condition is LIGHT, then let's write the word "daylight" to the Serial Monitor. Otherwise, write the word "nighttime" to the Serial Monitor.

First, add a method called lightCheck() to the previous Arduino sketch.

```
//globals to store data
int reading;
int threshold = 900; //range of 0-1023 / higher value = brighter

void lightCheck(){
   reading = analogRead(ldr);
   if(reading >= threshold){ //this condition means the readings are light
      Serial.println("daylight");
   }
   else{
      Serial.println("nighttime");
   }
   delay(1000);
}
```

Then, we'll call lightCheck() in our loop() method. Here's how it should work.

Communication through Bluetooth iot:

Bluetooth is a widely used wireless communication technology that is frequently used in Internet of Things (IoT) devices to enable communication between them. Bluetooth enables low-power, short-range wireless communication between devices, making it an ideal choice for many IoT applications.

Bluetooth is often used in IoT devices for various purposes such as transferring data, controlling devices, and enabling communication between devices. For example, a Bluetooth-enabled sensor in a smart home can communicate with a smartphone app to send data on temperature, humidity, and other environmental factors.

One of the main advantages of using Bluetooth in IoT is that it is a low-power technology, which means that it does not consume a lot of battery power. This makes it an ideal choice for IoT devices that are designed to be powered by a battery or other low-power sources. Additionally, Bluetooth is relatively easy to implement and has a high level of compatibility with many devices.

In summary, Bluetooth is an important technology for enabling communication in IoT devices, and its low-power and compatibility features make it a popular choice for many IoT applications

Bluetooth is a short-range wireless communication network over a radio frequency. Bluetooth is mostly integrated into smartphones and mobile devices. The Bluetooth communication network works within 2.4 ISM band frequencies with data rate up to 3Mbps.

There are three categories of Bluetooth technology:

1. Bluetooth Classic
2. Bluetooth Low Energy
3. Bluetooth SmartReady

The features of Bluetooth 5.0 version is introduced as Bluetooth 5 which have been developed entirely for the Internet of Things.

Play Video

Properties of Bluetooth network

- **Standard:** Bluetooth 4.2

- **Frequency:** 2.4GHz
- **Range:** 50-150m
- **Data transfer rates:** 3Mbps

Advantages of Bluetooth network

- It is wireless.
- It is cheap.
- It is easy to install.
- It is free to use if the device is installed with it.

Disadvantages of Bluetooth network

- It is a short-range communication network.
- It connects only two devices at a time.
- Communication through WiFi is a type of wireless communication that enables devices to communicate with each other using radio waves over a local area network (LAN). WiFi (Wireless Fidelity) is a popular wireless networking technology that uses radio waves to provide wireless high-speed Internet and network connections.

- To communicate through WiFi, devices must have a wireless network adapter that can transmit and receive signals over the air. The wireless adapter converts the data into radio waves and sends it to the wireless router or access point, which then sends the data to other devices connected to the same network.

- WiFi communication is commonly used in homes, offices, public places, and other locations where there is a need for wireless connectivity. Some examples of devices that use WiFi communication include smartphones, laptops, tablets, gaming consoles, and smart home devices.

- One of the advantages of WiFi communication is its convenience and flexibility, as it allows devices to connect to the Internet and network without the need for cables or physical connections. However, WiFi communication can be affected by factors such as distance, interference, and network congestion, which can impact its speed and reliability.

communication through wifi:

IoT (Internet of Things) devices often use Wi-Fi to communicate with each other and with the internet. Wi-Fi is a wireless networking technology that uses radio waves to transmit data between devices that are within range of a Wi-Fi network.

To communicate through Wi-Fi in IoT, devices need to have Wi-Fi radios and be connected to a Wi-Fi network. The devices can then send and receive data over the network using various communication protocols such as HTTP, MQTT, CoAP, and others.

One important consideration when using Wi-Fi for IoT communication is security. Wi-Fi networks can be vulnerable to hacking and other security threats, so it's important to implement strong security measures such as encryption, secure authentication, and access controls to protect IoT devices and the data they transmit.

Overall, Wi-Fi is a popular choice for IoT communication due to its widespread availability, relatively low cost, and ease of use. However, other wireless communication technologies such as Bluetooth, Zigbee, and LoRaWAN may also be suitable for specific IoT use cases depending on factors such as range, power consumption, and data transfer speed.

Applications of Wi-Fi :

Wi-Fi has many applications, it is used in all the sectors where a computer or any digital media is used, also for entertaining Wi-Fi is used. Some of the applications are mentioned below –

- Accessing Internet: Using Wi-Fi we can access the internet in any Wi-Fi-capable device wirelessly.
- We can stream or cast audio or video wirelessly on any device using Wi-Fi for our entertainment.
- We can share files, data, etc between two or more computers or mobile phones using Wi-Fi, and the speed of the data transfer rate is also very high. Also, we can print any document using a Wi-Fi printer, this is very much used nowadays.
- We can use Wi-Fi as **HOTSPOTS** also, it points Wireless Internet access for a particular range of area. Using Hotspot the owner of the main network connection can offer temporary network access to Wi-Fi-capable devices so that the users can

use the network without knowing anything about the main network connection. Wi-Fi adapters are mainly spreading radio signals using the owner network connection to provide a hotspot.

- Using Wi-Fi or WLAN we can construct simple wireless connections from one point to another, known as Point to point networks. This can be useful to connect two locations that are difficult to reach by wire, such as two buildings of corporate business.
- One more important application is **VoWi-Fi**, which is known as **voice-over Wi-Fi**. Some years ago telecom companies are introduced VoLTE (Voice over Long-Term Evolution). Nowadays they are introduced to VoWi-Fi, by which we can call anyone by using our home Wi-Fi network, only one thing is that the mobile needs to connect with the Wi-Fi. Then the voice is transferred using the Wi-Fi network instead of using the mobile SIM network, so the call quality is very good. Many mobile phones are already getting the support of VoWi-Fi.
- Wi-Fi in offices: In an office, all the computers are interconnected using Wi-Fi. For Wi-Fi, there are no wiring complexities. Also, the speed of the network is good. For Wi-Fi, a project can be presented to all the members at a time in the form of an excel sheet, ppt, etc. For Wi-Fi, there is no network loss as in cable due to cable break.
- Also using W-Fi a whole city can provide network connectivity by deploying routers at a specific area to access the internet. Already schools, colleges, and universities are providing networks using Wi-Fi because of its flexibility.
- Wi-Fi is used as a *positioning system* also, by which we can detect the positions of Wi-Fi hotspots to identify a device location.

Types of Wi-Fi:

Wi-Fi has several types of standards, which are discussed earlier, here just the name of the standards are defined,

Standards	Year of Release	Description
Wi-Fi-1 (802.11b)	**1999**	**This version has a link speed from 2Mb/s to 11 Mb/s over a 2.4 GHz frequency band**
Wi-Fi-2 (802.11a)	**1999**	**After a month of release previous version, 802.11a was released and it provide up to 54 Mb/s link speed over 5 Ghz band**
Wi-Fi-3 (802.11g)	**2003**	**In this version the speed was increased up to 54 to 108 Mb/s over 2.4 GHz**
802.11i	**2004**	**This is the same as 802.11g but only the security mechanism was increased in this version**
802.11e	**2004**	**This is also the same as 802.11g, only Voice over Wireless LAN and multimedia streaming are involved**

Wi-Fi-4 (802.11n)	**2009**	**This version supports both 2.4 GHz and 5 GHz radio frequency and it offers up to 72 to 600 Mb/s speed**
Wi-Fi-5 (802.11ac)	**2014**	**It supports a speed of 1733 Mb/s in the 5 GHz band**

Comparison between WiFi and Bluetooth

The following table highlights the major differences between WiFi and Bluetooth.

Key	WiFi	Bluetooth
Definition	WiFi stands for Wireless Fidelity. Wi-Fi is a technology that enables devices to connect to the Internet wirelessly.	Bluetooth is a wireless technology that is u to connect devices in short range.
Component	WiFi requires wireless adaptor on all devices and Wireless Router for connectivity.	Bluetooth requires an Bluetooth adaptor on devices for connectivity.
Power Consumption	WiFi consumes high power.	Bluetooth is easier to use and consumes l power than Wi-Fi because it only requires adapter on each connecting device.
Security	WiFi is more secure than Bluetooth.	Bluetooth is less secure than other wirel technologies such as WiFi.
Number of Users	Wi-Fi allows more devices and users to communicate at the same time.	Bluetooth restricts the number of devices t can connect at any given moment.
Bandwidth	WiFi needs high bandwidth.	Bluetooth has a low bandwidth.
Coverage	WiFi coverage area is up to 32 meters.	Bluetooth coverage area is about 10 meters.

Unit 3- IoT Architecture and Protocols

Topics:

- Architecture Reference Model- Introduction
- Refernece Model and Architecture
- IoT reference Model
- Protocols -6LowPAN, RPL, CoAP, MQTT
- IoT frameworks- Thing Speak

IoT Architecture

Introduction

The Internet of Things (IoT) has seen an increasing interest in adaptive frameworks and architecturaldesigns to promote the correlation between IoT devices and IoT systems. IoT systems aredesigned to be categorized across diverse application domains and geographical locations. It, therefore, createsextensive dependencies across domains, platforms, and services. Considering this interdependency between IoTdevices and IoT systems, an intelligent, connection-aware framework has become a necessity, this is where IoTarchitecture comes into play! Imagine a variety of smart IoT systems from sensors and actuators to internetgetaways and Data Acquisition Systems all under the centralized control of one "brain"! The brain here can bereferred to as the IoT architecture, whose effectiveness and applicability directly correlate with the quality of itsbuilding blocks. The way a system interacts and the different functions an IoT device performs are variousapproaches to IoT architecture. Since we can call the architecture the brain, it's also possible to say that the keycauses of poor integration in IoT systems are the shortage of intelligent, connection-aware architecture tosupport interaction in IoT systems.An IoT architecture is a system of numerous elements that range from sensors, protocols, and actuators, tocloud services, and layers. Besides, devices and sensors the Internet of Things (IoT) architecture layers aredistinguished to track the consistency of a system through protocols and gateways. Different architectures havebeen proposed by researchers and we can all agree that there is no single consensus on architecture for IoT. Themost basic architecture is a three-layer architecture.

Architecture Reference Model Introduction

A reference model is a division of functionality together with data flow between the pieces. A referencemodel is a standard decomposition of a known problem into parts that cooperatively solve the problem.Arising from experience, reference models are a characteristic of mature domains. Can you name thestandard parts of a compiler or a database management system? Can you explain in broad terms how the parts
work together to accomplish their collective purpose? If so, it is because you have been taught a reference modelof these applications.A reference architecture is a reference model mapped onto software elements (that cooperatively implement thefunctionality defined in the reference model) and the data flows between them. Whereas a reference modeldivides the functionality, a reference architecture is the mapping of that functionality onto a system
decomposition. The mapping may be, but by no means necessarily is, one to one. A software element mayimplement part of a function or several functions. Reference models, architectural

patterns, and referencearchitectures are not architectures; they are useful concepts that capture elements of an architecture. Each is theoutcome of early design decisions. The relationship among these design elements is shown in Figure 1

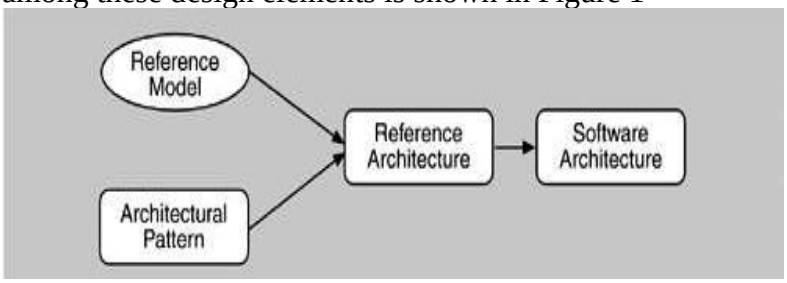


Fig 2.2 The relationships of reference models, architectural patterns, reference architectures, andsoftware architectures.

IoT Reference Architecture

The reference architecture consists of a set of components. Layers can be realized by means of specifictechnologies, and we will discuss options for realizing each component. There are also some crosscutting/vertical layers such as access/identity management.

Fig

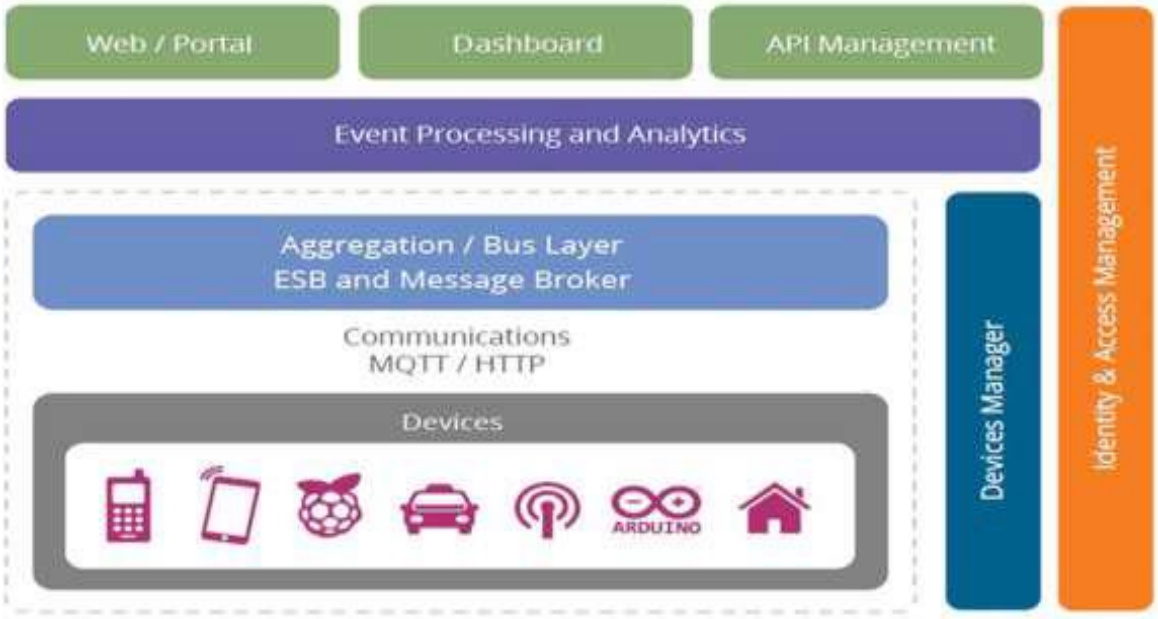


Fig 2.3 IoT Reference Architecture

The layers are

• Client/external communications - Web/Portal, Dashboard, APIs
• Event processing and analytics (including data storage)
• Aggregation/bus layer – ESB and message broker
• Relevant transports - MQTT/HTTP/XMPP/CoAP/AMQP, etc.
• Devices: The cross-cutting layers are
• Device manager
• Identity and access management

THE DEVICE LAYER

The bottom layer of the architecture is the device layer. Devices can be of various types, but in order to beconsidered as IoT devices, they must have some communications that either indirectly or directly attaches to the
Internet. Examples of direct connections are
• Arduino with Arduino Ethernet connection
• Arduino Yun with a Wi-Fi connection
• Raspberry Pi connected via Ethernet or Wi-Fi
• Intel Galileo connected via Ethernet orWi-Fi Examples of indirectly connected devices include
• ZigBee devices connected via a ZigBee gateway
• Bluetooth or Bluetooth Low Energy devices connecting via a mobile phone
• Devices communicating via low power radios to aRaspberry Pi There are many more such examples of each type.Each device typically needs an identity. The identity may be one of the following:
• A unique identifier (UUID) burnt into the device (typically part of the System-on Chip, or provided
by a secondary chip)
•A UUID provided by the radio subsystem (e.g. Bluetooth identifier, Wi-Fi MAC address)
• An OAuth2 Refresh/Bearer Token (this may be in addition to one of the above)
• An identifier stored in nonvolatile memory such as EEPROMFor the reference architecture we recommend that every device has a UUID (preferably an unchangeable ID
provided by the core hardware) as well as an OAuth2 Refresh and Bearer token stored in EEPROM. Thespecification is based on HTTP; however, (as we will discuss in the communications section) the referencearchitecture also supports these flows over MQTT.

COMMUNICATIONS LAYER

The communication layer supports the connectivity of the devices. There are multiple potential protocols forcommunication between the devices and the cloud.
The most well-known three potential protocols are
• HTTP/HTTPS (and RESTful approaches on those)
• MQTT 3.1/3.1.1(Message Queuing Telemetry Transport)
• Constrained application protocol (CoAP)
HTTP is well known, and there are many libraries that support it. Because it is a simple text-based protocol,many small devices such as 8-bit controllers can only partially support the protocol – for example enough codeto POST or GET a resource. The larger 32-bit based devices can utilize full HTTP client libraries that properlyimplement the whole protocol.
There are several protocols optimized for IoT use. The two best known are
MQTT6 and CoAP7. MQTT was invented in 1999 to solve issues in embedded systems and SCADA. It hasbeen through some iterations and the current version (3.1.1) is undergoing standardization in the OASIS MQTTTechnical Committee8. MQTT is a publish-subscribe messaging system based on a broker model. The protocolhas a very small overhead (as little as 2 bytes per message), and was designed to support lossy and
intermittently connected networks. MQTT was designed to flow over TCP. In addition, there is an associatedspecification designed for ZigBee-style networks called MQTT-SN (Sensor Networks). CoAP is a protocolfrom the IETF that is designed to provide a RESTful application protocol modeled on HTTP semantics, butwith a much smaller footprint and a binary rather than a text- based approach. CoAP is a more traditional clientserver
approach rather than a brokered approach. CoAP is designed to be used over UDP. For the referencearchitecture we have opted to select MQTT as the preferred device communication protocol, with HTTP as analternative option.

The reasons to select MQTT and not CoAP at this stage are
• Better adoption and wider library support for MQTT;
• Simplified bridging into existing event collection and event processing systems; and
• Simpler connectivity over firewalls and NAT networks
However, both protocols have specific strengths (and weaknesses) and so there will be some situations whereCoAP may be preferable and could be swapped in. In order to support MQTT we need to have an MQTTbroker in the architecture as well as device libraries. We will discuss this with regard to security and scalabilitylater. One important aspect with IoT devices is not just for the device to send data to the cloud/ server, but alsothe reverse. This is one of the benefits of the MQTT specification: because it is a brokered model, clients connect an outbound connection to the broker, whether or not the device is acting as a publisher or subscriber.

This usually avoids firewall problems because this approach works even behind firewalls or via NAT. In thecase where the main communication is based on HTTP, the traditional approach for sending data to thedevice would be to use HTTP Polling. This is very inefficient and costly, both in terms of network traffic aswell as power requirements. The modern replacement for this is the WebSocket protocol9 that allows anHTTP connection to be upgraded into a full two-way connection. This then acts as a socket channel (similar toa pure TCP channel) between the server and client. Once that has been established, it is up to the system tochoose an ongoing protocol to tunnel over the connection. For the reference architecture we once againrecommend using MQTT as a protocol with Web Sockets. In some cases, MQTT over Web Sockets will bethe only protocol. This is because it is even more firewall-friendly than the base MQTT specification as well assupporting pure browser/JavaScript clients using the same protocol. Note that while there is some support for Web Sockets on small controllers, such as Arduino, the combination of network code, HTTP and Web Socketswould utilize most of the available code space on a typical Arduino 8-bit device. Therefore, it is recommendedthe use of Web Sockets on the larger 32-bit devices.

AGGREGATION/BUS LAYER

An important layer of the architecture is the layer that aggregates and brokers communications. This is an
important layer for three reasons:
1. The ability to support an HTTP server and/or an MQTT broker to talk to the devices
2. The ability to aggregate and combine communications from different devices and to route communications to a specific device (possibly via a gateway)
3. The ability to bridge and transform between different protocols, e.g. to offer HTTP based APIs thatare mediated into an MQTT message going to the device. The aggregation/bus layer provides these capabilitiesas well as adapting into legacy protocols. The bus layer may also provide some simple correlation and mappingfrom different correlation models (e.g. mapping a device ID into an owner‘s ID or vice-versa).

Finally, theaggregation/bus layer needs to perform two key security roles. It must be able to act as an OAuth2 Resource

Server (validating Bearer Tokens and associated resource access scopes). It must also be able to act as a policyenforcement point (PEP) for policy-based access. In this model, the bus makes requests to the identity andaccess management layer to validate access requests. The identity and access management layer acts as a policydecision point (PDP) in this process. The bus layer then implements the results of these calls to the PDP to
either allow or disallow resource access.

EVENT PROCESSING AND ANALYTICS LAYER
This layer takes the events from the bus and provides the ability to process and act upon these events. A corecapability here is the requirement to store the data into a database. This may happen in three forms. Thetraditional model here would be to write a server-side application, e.g. this could be a JAX-RS application backed by a database. However, there are many approaches where wecan support more agile approaches. The first of these is to use a big data analytics platform. This is acloudscalable platform that supports technologies such as Apache Hadoop to provide highly scalable mapreduce analytics on the data coming from the devices. The second approach is to support complex eventprocessing to initiate near real- time activities and actions based on data from the devices and from the rest ofthe system.

Our recommended approach in this space is to use the following approaches:
• Highly scalable, column-based data storage for storing events
• Map-reduce for long-running batch-oriented processing of data
• Complex event processing for fast in-memory processing and near real-time reaction and autonomic
actions based on the data and activity of devices and other systems
• In addition, this layer may support traditional application processing platforms, such as Java Beans,JAX-RS logic, message-driven beans, or alternatives, such as node.js, PHP, Ruby or Python.

CLIENT/EXTERNAL COMMUNICATIONS LAYER
The reference architecture needs to provide a way for these devices to communicate outside of the device-oriented system. This includes three main approaches. Firstly, we need the ability to create web-based frontendsand portals that interact with devices and with the event-processing layer. Secondly, we need the ability tocreate dashboards that offer views into analytics and event processing. Finally, we need to be able to interactwith systems outside this network using machine-to-machine communications (APIs). These APIs need to bemanaged and controlled and this happens in an API management system. The recommended approach tobuilding the web front end is to utilize a modular front-end architecture, such as a portal, which allowssimple fast composition of useful UIs. Of course, the architecture also supports existing Web server-sidetechnology, such as Java Servlets/ JSP, PHP, Python, Ruby, etc. Our recommended approach is based on theJava framework and the most popular Java-based web server, Apache Tomcat. The dashboard is a re-usable system focused on creating graphs and other visualizations of data coming from the devices and the eventprocessing layer.

The API management layer provides three main functions:
• The first is that it provides a developer-focused portal (as opposed to the user focused portal previously mentioned), where developers can find, explore, and subscribe to APIs from the system. There isalso support for publishers to create, version, and manage the available and published APIs;
• The second is a gateway that manages access to the APIs, performing access control checks (forexternal requests) as well as throttling usage based on policies. It also performs routing and load- balancing;

DEVICE MANAGEMENT
Device management (DM) is handled by two components. A server-side system (the device manager)communicates with devices via various protocols and provides both individual and

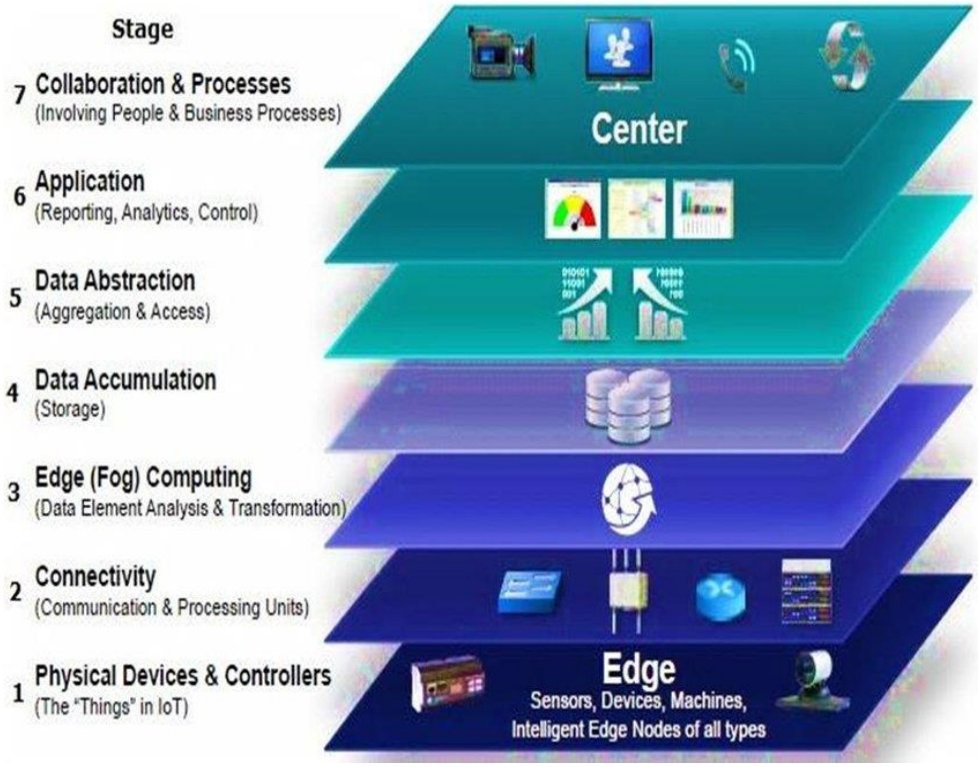
bulk control of devices. Italso remotely manages software and applications deployed on the device. It can lock and/or wipe the device ifnecessary. The device manager works in conjunction with the device management agents. There are multipledifferent agents for different platforms and device types. The device manager also needs to maintain the list of device identities and map these into owners. It must also work with the identity and access management layer tomanage access controls over devices (e.g. who else can manage the device apart from the owner, how muchcontrol does the owner have vs. the administrator, etc.) There are three levels of device: non-managed, semimanagedand fully managed (NM, SM, FM). Fully managed devices are those that run a full DM agent.
A full DM agent supports:
• Managing the software on the device
• Enabling/disabling features of the device (e.g. camera, hardware, etc.)
• Management of security controls and identifiers
• Monitoring the availability of the device • Maintaining a record of the device location if available
• Locking or wiping the device remotely if the device is compromised, etc.
Non-managed devices can communicate with the rest of the network, but have no agent involved. These may
include 8-bit devices where the constraints are too small to support the agent. The device manager may still
maintain information on the availability and location of the device if this is available. Semi-managed devices
are those that implement some parts of the DM (e.g. feature control, but not software management).

IDENTITY AND ACCESS MANAGEMENT
The final layer is the identity and access management layer. This layer needs to provide the following services:
• OAuth2 token issuing and validation
• Other identity services including SAML2 SSO and OpenID Connect support for identifying inboundrequests from the Web layer
• XACML PDP
• Directory of users (e.g. LDAP)
• Policy management for access control (policy control point)
The identity layer may of course have other requirements specific to the other identity and access managementfor a given instantiation of the reference architecture. In this section, we have outlined the major components ofthe reference architecture as well as specific decisions we have taken around technologies. These decisions aremotivated by the specific requirements of IoT architectures as well as best practices for building agile,evolvable, scalable Internet architectures.

IoT Reference Model

In an IoT system, data is generated by multiple kinds of devices, processed in different ways, transmitted todifferent locations, and acted upon by applications. The proposed IoT reference model is comprised of sevenlevels. Each level is defined with terminology that can be standardized to create a globally accepted frame ofreference. The IoT Reference Model does not restrict the scope or locality of its components. For example,from a physical perspective, every element could reside in a single rack of equipment or it could be distributedacross the world. The IoT Reference Model also allows the processing occurring at each level to range fromtrivial to complex, depending on the situation. The model describes how tasks at each

level should be handledto maintain simplicity, allow high scalability, and ensure supportability. Finally, the model defines thefunctions required for an IoT system to be complete. Figure illustrates the IoT Reference model and itslevels. It is important to note that in the IoT, data flows in both directions. In a control pattern, controlinformation flows from the top of the model (level 7) to the bottom (level 1). In a monitoring pattern, theflow of information is the reverse. In most systems, the flow will be bidirectional.



Level 1: Physical Devices and Controllers

The IoT Reference Model starts with Level 1: physical devices and controllers that might control multipledevices. These are the "things" in the IoT, and they include a wide range of endpoint devices that send andreceive information. Today, the list of devices is already extensive. It will become almost unlimited as moreequipment is added to the IoT over time. Devices are diverse, and there are no rules about size, location, formfactor, or origin. Some devices will be the size of a silicon chip. Some will be as large as vehicles. The IoTmust support the entire range. Dozens or hundreds of equipment manufacturers will produce IoT devices. Tosimplify compatibility and support manufacturability, the IoT Reference Model generally describes the level ofprocessing needed from Level 1 devices.

Level 2: Connectivity

Communications and connectivity are concentrated in one level—Level 2.
The most important function of Level 2 is reliable, timely information transmission. This includestransmissions:

- Between devices (Level 1) and the network
- Across networks (east-west)
- Between thenetwork (Level 2) and low-level information processing occurring at Level 3

Traditional data communication networks have multiple functions, as evidenced by the InternationalOrganization for Standardization (ISO) 7-layer reference model. However, a complete IoT system containsmany levels in addition to the communications network. One objective of the IoT Reference Model is forcommunications and processing to be executed by existing networks. The IoT Reference Model does notrequire or indicate creation of a different network—it relies on existing networks. However, some legacydevices aren't IP-enabled, which will require introducing communication gateways. Other devices will require proprietary controllers to serve the communication function. However, over time, standardization will increase.As Level 1 devices proliferate, the ways in which they interact with Level 2 connectivity equipment maychange. Regardless of the details, Level 1 devices communicate through the IoT system by interacting withLevel 2 connectivity equipment

Level 3: Edge (Fog) Computing

The functions of Level 3 are driven by the need to convert network data flows into information that is suitablefor storage and higher-level processing at Level 4 (data accumulation). This means that Level 3 activities focuson high-volume data analysis and transformation. For example, a Level 1 sensor device might generate datasamples multiple times per second, 24 hours a day, 365 days a year. A basic tenet of the IoT Reference Model is that the most intelligent system initiates information processing as early and as close to the edge of thenetwork as possible. This is sometimes referred to as fog computing. Level 3 is where this occurs. Given thatdata is usually submitted to the connectivity level (Level 2) networking equipment by devices in small units,Level 3 processing is performed on a packet-by-packet basis. This processing is limited, because there is onlyawareness of data units—not "sessions" or "transactions." Level 3 processing can encompass many examples, such as:

● Evaluation: Evaluating data for criteria as to whether it should be processed at a higher level
● Formatting: Reformatting data for consistent higher-level processing
● Expanding/decoding: Handling cryptic data with additional context (such as the origin)
● Distillation/reduction: Reducing and/or summarizing data to minimize the impact of data and traffic on the network and higher-level processing systems
● Assessment: Determining whether data represents a threshold or alert; this could include redirecting data to additional destinations

Level 4: Data Accumulation

Networking systems are built to reliably move data. The data is "in motion." Prior to Level 4, data is movingthrough the network at the rate and organization determined by the devices generating the data. The model isevent driven. As defined earlier, Level 1 devices do not include computing capabilities themselves. However,some computational activities could occur at Level 2, such as protocol translation or application of networksecurity policy. Additional compute tasks can be performed at Level 3, such as packet inspection. Driving computational tasks as close to the edge of the IoT as possible, with heterogeneous systems distributed acrossmultiple management domains represents an example of fog computing. Fog computing and fog services will be a distinguishing characteristic of the IoT. Most applications cannot, ordo not need to, process data at network wire speed. Applications typically assume that data is "at rest"—orunchanging—in memory or on disk. At Level 4, Data Accumulation, data in motion is converted to data at rest.

Level 4 determines:

● If data is of interest to higher levels: If so, Level 4 processing is the first level that is configured toserve the specific needs of a higher level.
● If data must be persisted: Should data be kept on disk in a non-volatile state or accumulated inmemory for short-term use?
● The type of storage needed: Does persistency require a file system, big data system, or relationaldatabase?
● If data is organized properly: Is the data appropriately organized for the required storage system?
● If data must be recombined or recomputed: Data might be combined, recomputed, or aggregatedwith previously stored information, some of which may have come from non-IoT sources.

As Level 4 captures data and puts it at rest, it is now usable by applications on a non-real-time basis.
Applications access the data when necessary. In short, Level 4 converts event- based data to query-based
processing. This is a crucial step in bridging the differences between the real-time networking world and
the non-real-time application world.

Level 5: Data Abstraction
IoT systems will need to scale to a corporate—or even global—level and will require multiple storage systemsto accommodate IoT device data and data from traditional enterprise ERP, HRMS, CRM, and other systems.The data abstraction functions of Level 5 are focused on rendering data and its storage in ways that enabledeveloping simpler, performance-enhanced applications.With multiple devices generating data, there are many reasons why this data may not land in the same datastorage:
● There might be too much data to put in one place.
● Moving data into a database might consume too much processing power, so that retrieving it must
be separated from the data generation process. This is done today with online transaction processing (OLTP)
databases and data warehouses.
● Devices might be geographically separated, and processing is optimized locally.
● Levels 3 and 4 might separate “continuous streams of raw data” from “data that represents anevent.” Data storage for streaming data may be a big data system, such as Hadoop. Storage for event data maybe a relational database management system (RDBMS) with faster query times.
● Different kinds of data processing might be required.For example, in-store processing will focus on different things than across-all-stores summary processing. Forthese reasons, the data abstraction level must process many different things.
These include:

- Reconciling multiple data formats from different sources
- Assuring consistent semantics of data across sources
- Confirming that data is complete to the higher-level application

● Consolidating data into one place (with ETL, ELT, or data replication) or providing access tomultiple data stores through data virtualization
● Protecting data with appropriate authentication and authorization
● Normalizing or denormalizing and indexing data to provide fast application access Level

Application Level 6
It is the application level, where information interpretation occurs. Software at this level interacts with Level 5and data at rest, so it does not have to operate at network speeds. The IoT Reference Model does not strictlydefine an application. Applications vary based on vertical markets, the nature of device data, and businessneeds. For example, some applications will focus on monitoring device data. Some will focus on controlling devices. Some will combine device and non-device data. Monitoring and control applications represent manydifferent application models, programming patterns, and software stacks, leading to discussions of operatingsystems, mobility, application servers, hypervisors, multi-threading, multi-tenancy, etc. These topics arebeyond the scope of the IoT Reference Model discussion. Suffice it to say that application complexity will varywidely.
Examples include:
● Mission-critical business applications, such as generalized ERP or specialized industry solutions
● Mobile applications that handle simple interactions
● Business intelligence reports, where the application is the BI server
● Analytic applications that interpret data for business decisions
● System management/control center applications that control the IoT system itself and don't act on
the data produced by it
If Levels 1-5 are architected properly, the amount of work required by Level 6 will be reduced. If Level 6 isdesigned properly, users will be able to do their jobs better.
Level 7: Collaboration and Processes
One of the main distinctions between the Internet of Things (IoT) and IoT is that IoT includes people andprocesses. This difference becomes particularly clear at Level 7: Collaboration and Processes. The IoTsystem, and the information it creates, is of little value unless it yields action, which often requires people andprocesses. Applications execute business logic to empower people. People use applications and associated datafor their specific needs. Often, multiple people use the same application for a range of different purposes. So,the objective is not the application—it is to empower people to do their work better. Applications (Level 6) givebusiness people the right data, at the right time, so they can do the right thing. But frequently, the action neededrequires more than one person. People must be able to communicate and collaborate, sometimes using thetraditional Internet, to make the IoT useful. Communication and collaboration often require multiple steps. Andit usually transcends multiple applications. This is why Level 7, represents a higher level than a single application.

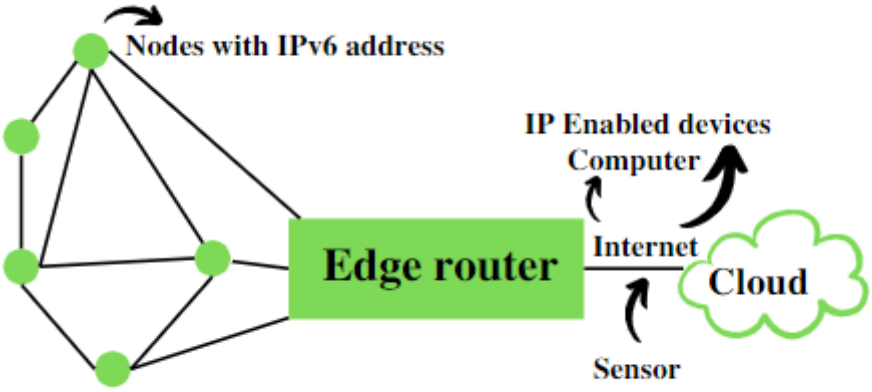
Protocols:

6LowPAN

6LoWPAN is an IPv6 protocol, and It's extended from is IPv6 over Low Power Personal Area Network. As the name itself explains the meaning of this protocol is that this protocol works on Wireless Personal Area Network i.e., WPAN.
WPAN is a Personal Area Network (PAN) where the interconnected devices are centered around a person's workspace and connected through a wireless medium. You can read more about WPAN at WPAN. 6LoWPAN allows communication using the

IPv6 protocol. IPv6 is Internet Protocol Version 6 is a network layer protocol that allows communication to take place over the network. It is faster and more reliable and provides a large number of addresses.
6LoWPAN initially came into existence to overcome the conventional methodologies that were adapted to transmit information. But still, it is not so efficient as it only allows for the smaller devices with very limited processing ability to establish communication using one of the Internet Protocols, i.e., IPv6. It has very low cost, short-range, low memory usage, and low bit rate.

It comprises an Edge Router and Sensor Nodes. Even the smallest of the IoT devices can now be part of the network, and the information can be transmitted to the outside world as well. For example, LED Streetlights.



- It is a technology that makes the individual nodes IP enabled.
- 6LoWPAN can interact with 802.15.4 devices and also other types of devices on an IP Network. For example, Wi-Fi.
- It uses AES 128 link layer security, which AES is a block cipher having key size of 128/192/256 bits and encrypts data in blocks of 128 bits each. This is defined in IEEE 802.15.4 and provides link authentication and encryption.

Basic Requirements of 6LoWPAN:

1. The device should be having sleep mode in order to support the battery saving.
2. Minimal memory requirement.
3. Routing overhead should be lowered.

Features of 6LoWPAN:

1. It is used with IEEE 802.15,.4 in the 2.4 GHz band.
2. Outdoor range: ~200 m (maximum)
3. Data rate: 200kbps (maximum)
4. Maximum number of nodes: ~100

Advantages of 6LoWPAN:

1. 6LoWPAN is a mesh network that is robust, scalable, and can heal on its own.
2. It delivers low-cost and secure communication in IoT devices.
3. It uses IPv6 protocol and so it can be directly routed to cloud platforms.
4. It offers one-to-many and many-to-one routing.
5. In the network, leaf nodes can be in sleep mode for a longer duration of time.

Disadvantages of 6LoWPAN:

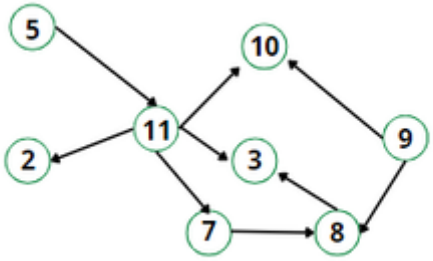
1. It is comparatively less secure than Zigbee.
2. It has lesser immunity to interference than that Wi-Fi and Bluetooth.
3. Without the mesh topology, it supports a short range.

Applications of 6LoWPAN:

1. It is a wireless sensor network.
2. It is used in home-automation,
3. It is used in smart agricultural techniques, and industrial monitoring.

RPL:

RPL stands for **Routing Protocol for Low Power and Lossy Networks** for heterogeneous traffic networks. It is a routing protocol for Wireless Networks. This protocol is based on the same standard as by Zigbee and 6 Lowpan is IEEE 802.15.4 It holds both many-to-one and one-to-one communication.
It is a **Distance Vector Routing Protocol** that creates a tree-like routing topology called the *Destination Oriented Directed Acyclic Graph (DODAG)*, rooted towards one or more nodes called the root node or sink node.
The *Directed Acyclic Graphs* (DAGs) are created based on user-specified specific Objective Function (OF). The OF defines the method to find the best-optimized route among the number of sensor devices.



Directed Acyclic Graph

The IETF chartered the ROLL (Routing Over Low power and Lossy networks) working group to evaluate all three routing protocols and determine the needs and requirements for developing a routing solution for IP smart objects. After the study of various use cases and a survey of existing protocols, the consensus was that a new routing protocol should be developed for IP smart objects, given the characteristics

and requirements of the constrained network. This new Distance Vector Routing Protocol was named the IPv6 Routing Protocol for Low power and Lossy networks(RPL). The RPL specification was published as RFC 6550 by the ROLL working group.

In an RPL Network, each node acts as a router and becomes part of a mesh network. Routing is performed at the IP Layer. Each node examines every received IPv6 packet and determines the next-hop destination based on the information contained in the IPv6 header. No information from the MAC layer header is needed to perform the next determination.

Modes of RPL:

This protocol defines two modes:
1. Storing mode: All modes contain the entire routing table of the RPL domain. Every node knows how to reach every other node directly.
2. Non-Storing mode: Only the border router(s) of the RPL domain contain(s) the full routing table. All other nodes in the domain maintain their list of parents only and use this as a list of default routes towards the border router. The abbreviated routing table saves memory space and CPU. When communicating in non-storing mode, a node always forwards its packet to the border router, which knows how to ultimately reach the final destination.
RPL is based on the concept of a Directed Acyclic Graph (DAG). A DAG is Directed Graph where no cycle exists. This means that from any vertex or point in the graph, we cannot follow an edge or a line back to this same point. All of the edges are arranged in a path oriented toward and terminating at one or more root nodes.

A basic RPL process involves building a Destination Oriented Directed Acyclic Graph (DODAG). A DODAG is a DAG rooted in one destination. In RPL this destination occurs at a border router known as the DODAG root. In a DODAG, three parents maximum are maintained by each node that provides a path to the root. Typically one of these parents is the preferred parent, which means it is the preferred next hop for upward roots towards the root. The routing graph created by the set of DODAG parents across all nodes defines the full set of upwards roots. RPL protocol information should ensure that routes are loop-free by disallowing nodes from selected DODAG parents positioned further away from a border router.

Implementation of RPL Protocol:

The RPL protocol is implemented using the Contiki Operating system. This Operating System majorly focuses on IoT devices, more specifically Low Power Wireless IoT devices. It is an Open source Model and was first bought into the picture by Adam Dunkels.

The RPL protocol mostly occurs in wireless sensors and networks. Other similar Operating Systems include T-Kernel, EyeOS, LiteOS, etc.

COAP:

This page covers CoAP protocol architecture used in IoT (Internet of Things). It mentions CoAP architecture,CoAP message format and CoAP message exchanges between CoAP client and CoAP server. CoAP is the short form of Constrained Application Protocol.

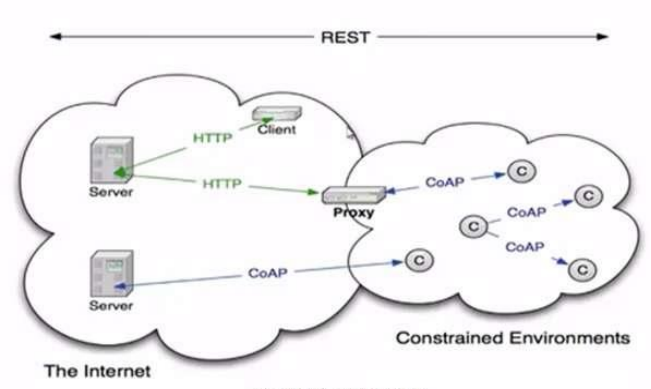
The CoAP protocol is specified in RFC 7252. It is a web transfer protocol which is used in constrained nodes or networks such as WSN, IoT, M2M etc. Hence the name Constrained Application Protocol. The protocol is targetted for Internet of Things (IoT) devices having less memory and less power specifications.As it is designed for web applications it is also known as "The Web of Things Protocol". It can be used to transport data from few bytes to 1000s of bytes over web applications. It exists between UDP layer and Application layer.

Following are the features of CoAP Protocol:
• It is very efficient RESTful protocol.
• Easy to proxy to/from HTTP.
• It is open IETF standard
• It is Embedded web transfer protocol (coap://)
• It uses asynchronous transaction model.
• UDP is binding with reliability and multicast support.
• GET, POST, PUT and DELETE methods are used.
• URI is supported.
• It uses small and simple 4 byte header.
• Supports binding to UDP, SMS and TCP.
• DTLS based PSK, RPK and certificate security is used.

• uses subset of MIME types and HTTP response codes.

• Uses built in discovery mechanism.

CoAP Architecture

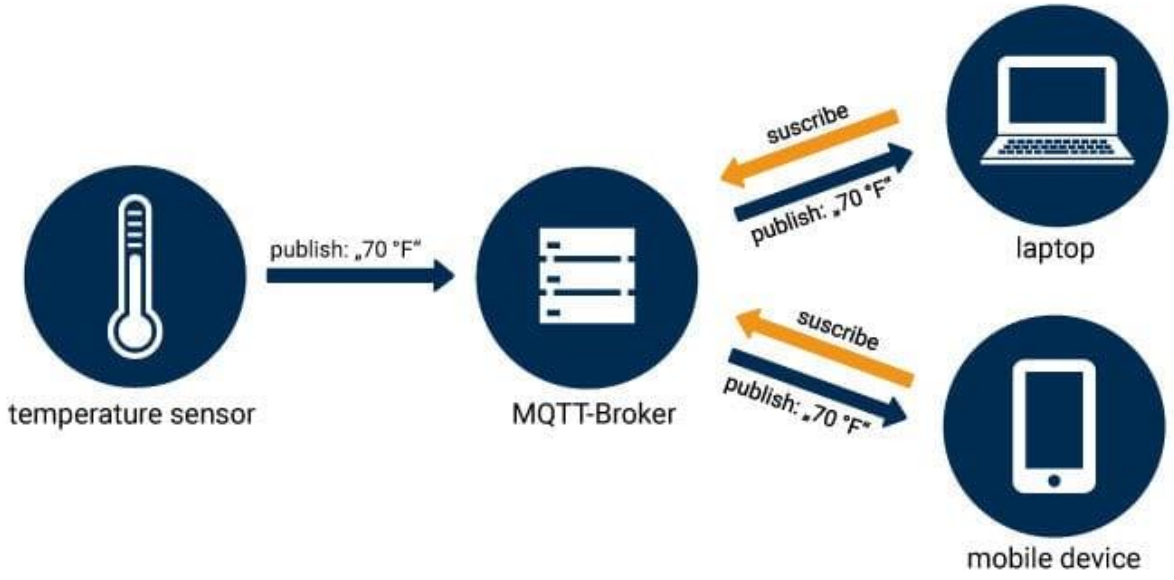


CoAP Architecture

The figure-1 depicts **CoAP Architecture**. As shown it extends normal HTTP clients to clients having resource constraints. These clients are known as CoAP clients. Proxy device bridges gap between constained environment and typical internet environment based on HTTP protocols. Same server takes care of both HTTP and CoAP protocol messages.

MQTT:

What is a MQTT broker?



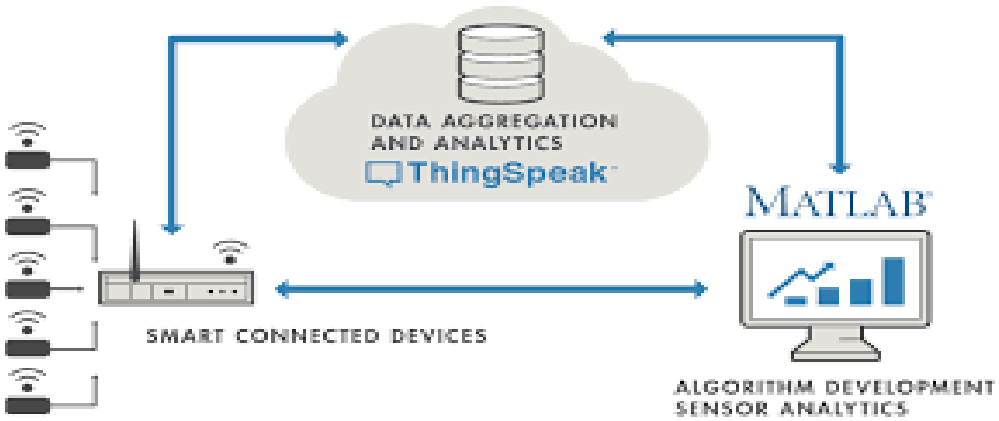
The MQTT broker is the center of every Publish / Subscribe protocol. Depending on the implementation, a broker can manage up to thousands of simultaneously connected MQTT clients. The broker is responsible for receiving all messages, filtering the messages, determining who subscribed to each message and sending the message to those subscribed clients. The Broker also holds the sessions of all persistent clients, including subscriptions and missed messages. Another task of the Broker is the authentication and authorization of clients. Usually the broker is extensible, which facilitates custom authentification, authorization and integration with backend systems. Integration is especially important, because the Broker is often the component directly exposed on the Internet, serves many clients and has to forward messages to downstream analysis and processing systems. In short, the Broker is the central hub through which every message must be routed. It is therefore important that your broker is highly scalable, can be integrated into back-end systems, is easy to monitor and, of course, is fail-safe.

IoT framework: Things Speak

ThingSpeak is a Web Service (REST API) that lets you collect and store sensor data in the cloud and develop Internet of Things applications.

• It works with Arduino, Raspberry Pi and MATLAB (premade libraries and APIs exists).

• But it should work with all kind of Programming Languages, since it uses a REST API and HTTP.



ThingSpeak is an IoT analytics platform service that lets you collect and store sensor data in the cloud and develop Internet of Things applications.

• The ThingSpeak service also lets you perform online analysis and act on your data. Sensor data can be sent to ThingSpeak from any hardware that can communicate using a REST API

• ThingSpeak is a Web Service (REST API) that lets you collect and store sensor data in the cloud and develop Internet of Things applications.

Device discovery and Cloud Services for IOT:

Device discovery capabilities:

Device discovery is a critical capability in the realm of Internet of Things (IoT) that allows IoT devices to identify and communicate with each other on a network. Here are some common device discovery capabilities in IoT:

Network Scanning: IoT devices can scan the network they are connected to in order to discover other devices that are also connected to the same network. This can be done using techniques such as IP scanning, MAC address scanning, or using protocols like Simple Service Discovery Protocol (SSDP) or Address Resolution Protocol (ARP).

1. Service Discovery Protocols: Service discovery protocols are specifically designed for device discovery in IoT networks. Examples include mDNS (Multicast DNS), DNS-SD (DNS Service Discovery), and UPnP (Universal Plug and Play) which allow devices to announce their presence and services on the network, and other devices to discover and communicate with them.
2. Beaconing: IoT devices can periodically broadcast a beacon or a signal that identifies themselves and their services. Other devices in the vicinity can then detect these beacons and initiate communication with the broadcasting devices. Bluetooth Low Energy (BLE) beacons are a common example of this approach, often used in applications such as indoor positioning or asset tracking.
3. Discovery via Gateway or Broker: In some IoT architectures, devices communicate with a central gateway or broker, which acts as a mediator for device discovery. Devices connect to the gateway or broker, and the gateway or broker maintains a registry of devices connected to it, allowing other devices to discover them through the gateway or broker.
4. Discovery via Cloud Services: IoT devices can also leverage cloud-based services for device discovery. Devices can register with a cloud-based service upon connection, and other devices or applications can query the cloud service to discover registered devices.
5. Proximity-based Discovery: Devices can use proximity-based techniques such as Bluetooth or NFC (Near Field Communication) to discover other devices in close physical proximity. This approach is often used in applications such as smart home devices, where devices can discover and communicate with each other when they are physically near each other.

These are some of the common device discovery capabilities in IoT. The specific approach or combination of approaches used for device discovery may vary depending on the IoT architecture, network topology, and application requirements.

Registering a device:-

Registering a device in an Internet of Things (IoT) system typically involves the process of adding the device to the IoT system's registry or database, so that it can be recognized and managed by the system. The exact steps and procedures for registering a device in an IoT system may vary depending on the specific IoT platform or framework being used, but generally, the following steps are typically involved:

1. Device Identification: Each IoT device needs to be uniquely identified within the IoT system. This can be done using methods such as assigning a unique device ID, MAC address, or serial number to the device. The device identification information is used to uniquely identify and differentiate the device from other devices in the system.
2. Device Authentication: Security is an important consideration in IoT systems, and device authentication is typically used to ensure that only authorized devices are allowed to register and participate in the system. Device authentication can involve methods such as providing a secure token or certificate, verifying a shared secret, or using public key infrastructure (PKI) mechanisms to authenticate the device.
3. Registration Process: The registration process typically involves providing information about the device, such as its unique identifier, metadata, capabilities, and configuration details, to the IoT system. This information is stored in the IoT system's registry or database, which can be used for device discovery, management, and communication.
4. Provisioning: Provisioning is the process of configuring and setting up the device for communication with the IoT system. This may involve tasks such as setting up network connectivity, configuring communication protocols, and registering the device with appropriate services or APIs.
5. Onboarding: Onboarding is the process of integrating the device into the IoT system's workflows and processes. This may include tasks such as associating the device with specific users or accounts, configuring access control, defining device behavior or rules, and integrating the device with other system components.
6. Validation and Confirmation: Once the device is registered, the IoT system may validate and confirm the registration to ensure that the device has been successfully added to the system. This may involve sending a confirmation message or notification to the device or the user, and verifying that the device is able to communicate with the system.

The specific steps and procedures for registering a device in an IoT system may vary depending on the platform, protocol, or framework being used, and may require customization based on the requirements of the specific IoT application or use case. Proper device registration is important for managing and securing IoT devices within an IoT system, and it serves as the foundation for device discovery, communication, and management in an IoT ecosystem.

Deregister a device:-
The process of deregistering a device in an Internet of Things (IoT) system or platform may vary depending on the specific IoT solution or platform you are using. However, here are some general steps that can be followed:

1. Identify the IoT platform: Determine the IoT platform or system that the device is registered with. This could be a cloud-based platform, a gateway, or a backend system used to manage and monitor IoT devices.
2. Access the platform: Log in to the IoT platform or system using the appropriate credentials. This may involve accessing a web-based dashboard, a mobile app, or an API depending on the platform.
3. Locate the device: Identify the specific device that you want to deregister from the IoT platform. This may involve navigating through a list of registered devices, searching for the device by name, or using other filtering or sorting options provided by the platform.

4. Initiate deregistration: Once you have located the device, look for an option or feature to deregister or unregister the device. This could be a button, a menu option, or a command provided by the platform.
5. Follow the instructions: Follow the prompts or instructions provided by the platform to initiate the deregistration process. This may involve confirming your decision to deregister, providing additional information such as device identification or confirmation codes, or following other specific steps as required by the platform.
6. Confirm deregistration: Once the deregistration process is initiated, verify that the device has been successfully deregistered from the IoT platform. This may involve checking the device status on the platform, confirming that the device is no longer listed as registered, or checking for any other notifications or messages related to the deregistration process.
7. Update device settings (if needed): If the device is still operational and connected to the IoT system, you may need to update its settings to reflect the deregistration. For example, you may need to configure the device to stop sending data or communicating with the IoT platform.

It's important to note that the specific steps and procedures for deregistering a device in an IoT system may vary depending on the platform or solution being used. Always refer to the documentation, guides, or support resources provided by the IoT platform or system for accurate and up-to-date instructions on how to properly deregister a device. Additionally, consider any implications or consequences of deregistering a device, such as loss of access to data, features, or services, and plan accordingly.

Introduction to cloud storage models and communication APIs web-server:

Cloud Storage Models in IoT: Cloud storage is a popular approach for managing and storing data generated by IoT devices. There are three common cloud storage models used in IoT:

1. Public Cloud: In this model, data generated by IoT devices is stored in a cloud infrastructure managed by a third-party cloud service provider. The data is stored in a shared environment, and the cloud provider is responsible for managing and maintaining the infrastructure, including storage, security, and scalability. Public cloud storage offers cost-effective and scalable storage solutions for IoT data, but may raise concerns about data privacy and security.
2. Private Cloud: In this model, data generated by IoT devices is stored in a cloud infrastructure that is owned and operated by an organization for its internal use. The organization has full control over the cloud infrastructure, and the data is stored in a dedicated environment. Private cloud storage offers greater control over data privacy and security, but may require higher upfront costs for setting up and maintaining the infrastructure.
3. Hybrid Cloud: This model combines elements of both public and private clouds, where data generated by IoT devices is stored in a combination of public and private cloud environments. Organizations can choose to store sensitive data in a private cloud for enhanced security, while utilizing the cost-effective scalability of a public cloud for less sensitive data.

Communication APIs in Web Servers for IoT: Web servers play a crucial role in enabling communication between IoT devices and cloud storage. Communication APIs (Application Programming Interfaces) are interfaces that allow devices and systems to interact with each other, exchange data, and perform actions. In the context of web servers in IoT, communication APIs

enable IoT devices to send data to the cloud for storage and retrieval, as well as receive instructions or commands from the cloud for controlling the devices. Some common communication APIs used in web servers for IoT include:

1. RESTful APIs: Representational State Transfer (REST) is an architectural style used for designing networked applications. RESTful APIs use standard HTTP methods (such as GET, POST, PUT, DELETE) to perform operations on resources identified by URLs. RESTful APIs are widely used in IoT for their simplicity and scalability, making them suitable for communication between IoT devices and cloud storage.
2. MQTT (Message Queuing Telemetry Transport): MQTT is a lightweight messaging protocol that is widely used in IoT for efficient and low-bandwidth communication. It follows a publish-subscribe model, where devices can publish messages to topics and subscribe to topics to receive messages. MQTT is commonly used for real-time, event-driven communication in IoT scenarios where low-latency and efficient data transmission are critical.
3. WebSocket: WebSocket is a protocol that provides bi-directional, full-duplex communication between web servers and clients over a single, long-lived connection. WebSocket enables real-time communication between IoT devices and cloud storage, allowing for efficient and low-latency data exchange. WebSocket is commonly used in IoT applications that require real-time updates and interactive communication between devices and the cloud.

In summary, cloud storage models and communication APIs are important components of IoT systems, allowing IoT devices to store and retrieve data from the cloud, and enabling communication between devices and cloud storage for real-time data exchange and control. Understanding these concepts is crucial for building scalable, efficient, and secure IoT applications.

Web server for IOT:

A web server for IoT (Internet of Things) is a specialized server that enables communication and data exchange between IoT devices and other components of an IoT system, such as cloud storage, databases, and applications. A web server in the context of IoT typically serves as a central hub or gateway that manages the flow of data between IoT devices and other parts of the system.

Here are some key features of a web server for IoT:

1. Data ingestion: The web server should be capable of receiving data from various IoT devices in different formats, such as sensor readings, telemetry data, and control commands. It should have the ability to ingest and process data from multiple devices concurrently, handling data streams efficiently.
2. Data processing: The web server should be able to process and analyze incoming data in real-time or near real-time. This may involve data validation, aggregation, transformation, and enrichment, depending on the requirements of the IoT system. Data processing capabilities are critical for deriving insights from IoT data and triggering appropriate actions or responses.
3. Data storage and retrieval: The web server should be able to store and retrieve data from cloud storage, databases, or other data repositories. This may involve storing raw data, aggregated data, or processed data, depending on the needs of the IoT system. Data storage and retrieval capabilities

are crucial for managing historical data, supporting analytics, and enabling data-driven decision making.

4. Communication and integration: The web server should be capable of communicating with various IoT devices using different communication protocols, such as RESTful APIs, MQTT, WebSocket, or other custom protocols. It should also be able to integrate with other components of the IoT system, such as cloud services, databases, and applications, for seamless data exchange and system integration.
5. Security and authentication: The web server should have robust security measures in place to protect the integrity, confidentiality, and authenticity of IoT data. This may involve authentication and authorization mechanisms, encryption, access control, and other security measures to ensure that only authorized devices and users can access and interact with the IoT system.
6. Scalability and performance: The web server should be designed to handle a large number of concurrent connections and data streams from multiple IoT devices. It should be able to scale horizontally or vertically to accommodate growing data volumes and increasing demands of the IoT system. High performance and low latency are critical for real-time or near real-time data processing in IoT applications.
7. Monitoring and management: The web server should provide monitoring and management capabilities to monitor the health, performance, and status of the IoT system. This may involve logging, monitoring, alerting, and management interfaces for troubleshooting, debugging, and performance optimization.

In summary, a web server for IoT is a specialized server that enables communication, data processing, and integration between IoT devices and other components of an IoT system. It plays a critical role in managing the flow of data and enabling real-time or near real-time communication and data processing in IoT applications.