Dr. N. Penchalaiah, Associate Professor ,

AI&ML, Annamacharya University

UNIT-1
Introduction to Internet of Things

Contents

- Definition and Characteristics of IoT
- Introduction to Internet of Things
- History of IoT
- Physical Design of IoT
- Logical Design of IoT
- IoT Enabling Technologies
- IoT Levels and Deployment Templates
- Applications of IoT

Definition and Characteristics of IoT

What is an IoT (Internet of Things):

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 use intelligent interfaces, and are seamlessly integrated into the information network, often communicate data associated with users and their environments.

Some of the terms from the definition are:

- The interconnection via the Internet of computing devices embedded in everyday objects, enabling them to send and receive data.
- The ***internet of things***, or IoT, is a system of interrelated computing devices, mechanical and digital machines, objects, animals or people that are provided with unique identifiers (UIDs) and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction.

Self configuring

- IoT devices may have self configuring capability allowing a large number of devices to work together to provide certain functionality. These devices have the ability to configure themselves, 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. Ex: **MQTT, CoAP, Websockets, AMQP, REST, etc**

Unique identity:

- each IoT device has a unique identity and a unique identifier

Integrated into information network:

- IoT devices are integrated into the information network that allows them to communicate and exchange data with other devices and systems.

Introduction to Internet of Things:

- Internet of Things comprises things that have unique identities and are connected to the internet (networked computers or 4G enabled mobile phones which have some form of unique identities are connected to the internet (MAC Address))
- The main focus here is to configure, control and network the things that are traditionally not associated with the internet(ex: thermostats, utility meters, bluetooth connected headset, irrigation pumps and sensors).
- Scope of IoT is not limited to just connecting things to the internet. IoT allows these things to communicate and exchange data(control and information that could include data associated with users).
- Applications on IoT networks extract and create information from lower level data by filtering, processing, categorizing, condensing and contextualizing the data.
- For example, (72,45) raw data is generated by the sensors which by themselves have no meaning. To give meaning to the data a context is added, like each tuple in data represents the temperature and humidity measured every minute. Further information is obtained by categorizing, condensing or processing this data like the average temperature and humidity readings for last five minutes is obtained by averaging the last five data tuples.

- Next step is to organize the information and understand the relationships between pieces of information to infer knowledge which can be put into action. Like an alert is raised if the average temperature in last five minutes exceeds 120F.

- Applications of Internet of Things span a wide range of domains including homes, cities, environment, energy systems, retail, logistics, industry, agriculture and health.

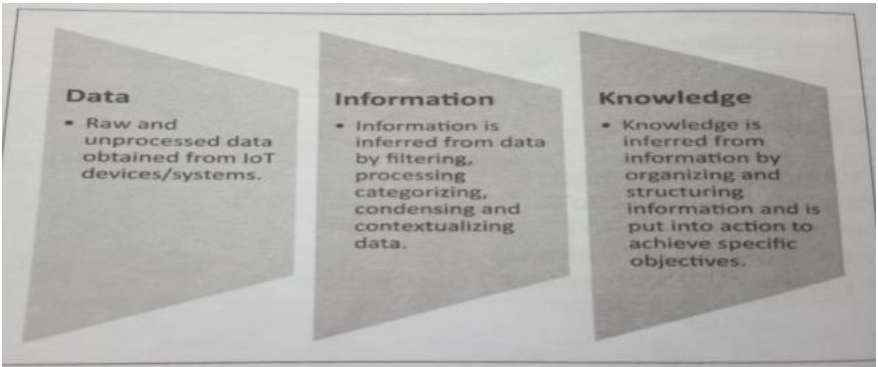


Figure 1.1: Inferring information and knowledge from data

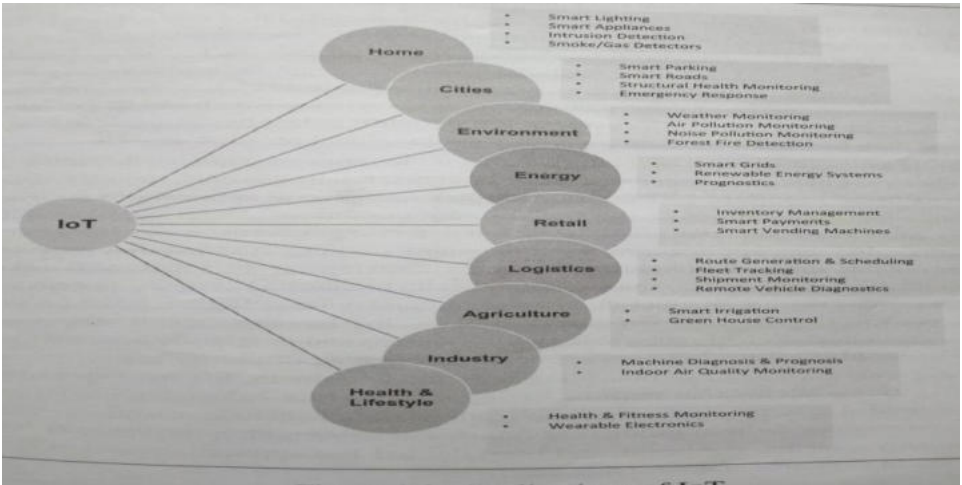
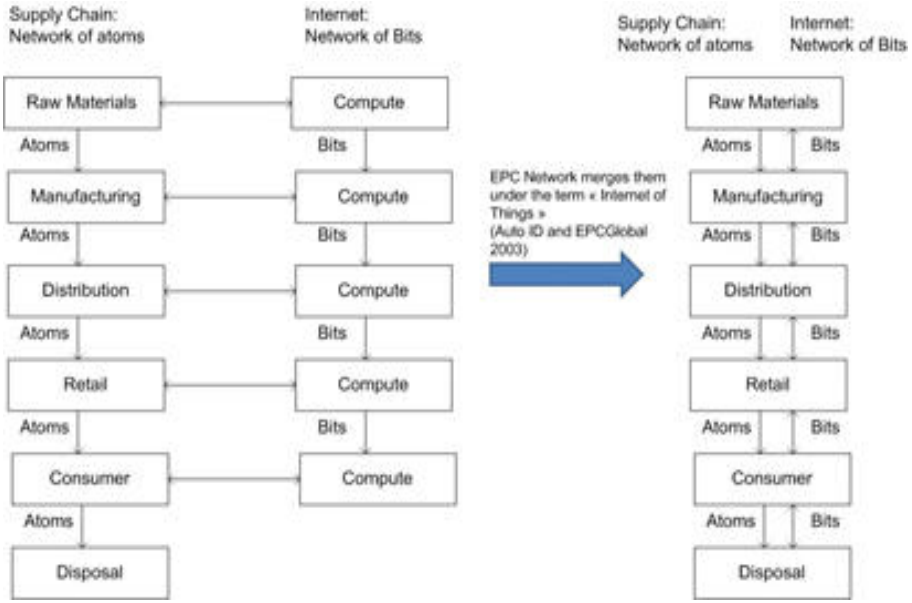


Figure 1.2: Applications of IoT

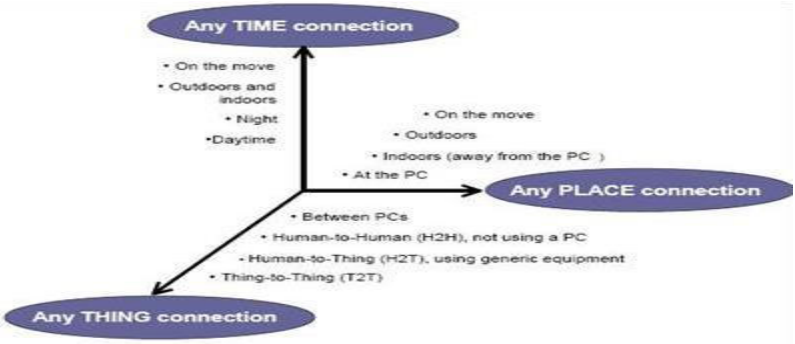
History of IoT:

- The term IoT was first coined by KEVIN ASHTON in 1999.
- The concept of IoT first became popular in the Auto-ID research center at the MIT [Massachusetts Institute], where an important effort was made to uniquely identify products. The result was termed EPC(electronic product code). And it was then commercialized by EPCglobal.
- Radio Frequency Identification was seen as a prerequisite for the IoT at that point, if all objects and people in daily life were equipped with identifiers, computers could manage and inventory them.
- A "thing" or "object" is any possible item in the real world that might join the communication chain.
- The initial main objective of the IoT was to combine the communication capabilities characterized by data transmission.
- This was viewed as the Internet, also known as the network of bits representing the "digital world".
- The process of automation was viewed as connecting the real or physical world, named the "network of atoms"



- In 2005, the ITU (International Telecommunication Unit) showed interest in new telecommunication business possibilities that could be built into services around the new connectivity of environment objects to the network.
- The ITU produced a comprehensive report on the IoT from technical, economical and ethical views.
- It introduced a new axis in the ubiquitous networking path to complete the existing "anywhere" and "anytime" connectivity. It is the "anything" connectivity axes where the thing-to-thing or machine-to-machine interaction is added to complete the existing person-to-person and person-to-machine interaction in the possible connectivity.
- Figure presents the ITU view of ubiquitous networking, adding the "anything connection" to the connectivity anywhere and anytime.

ITU any place, any time and any thing vision



By connecting these new objects will raise many questions such as

- The connecting technology of the so-called object
- Interoperability between objects
- Communication model of these connected objects
- Possible interaction with the existing models(internet)
- Choice of the transport model
- Addressing, identifying and naming
- Security and privacy

Logical Design of IoT

Logical design of an IoT system refers to an abstract representation of the entities and processes.

Here we will be discussing about

- IoT Functional Blocks
- IoT Communication models
- IoT Communication APIs

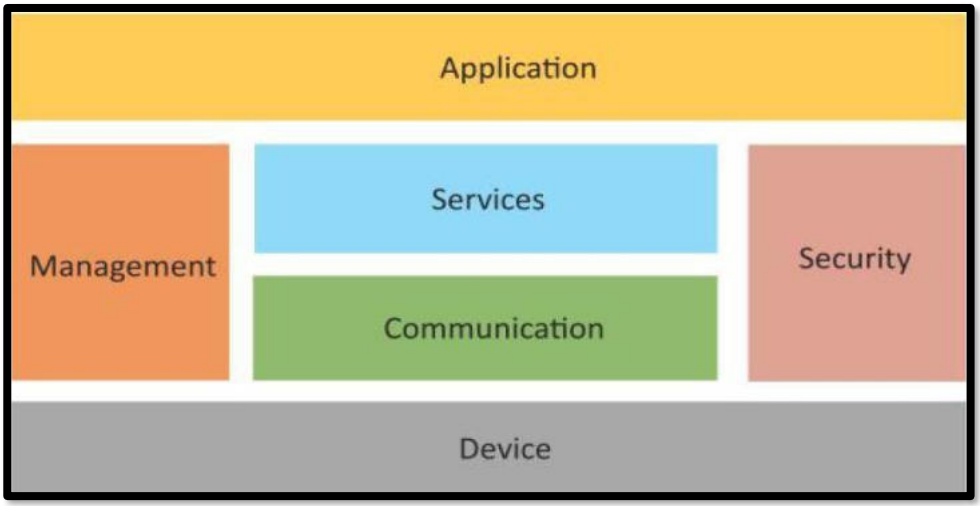
1. IoT Functional Blocks:

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

Functional blocks are

- **Devices**, 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**, 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.

Logical Design of IoT

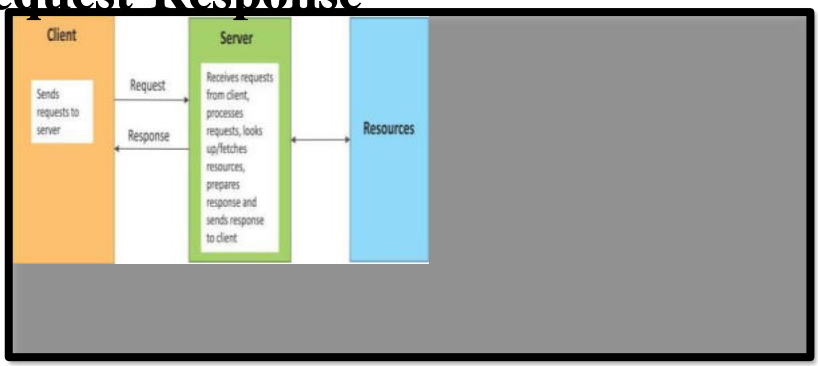


- **Management**, management functional block provides various functions to govern the IoT system.
- **Security,** security functional block secures the IoT system and by providing functions such as authentication, authorization, message and content integrity and data security.
- **Application**, applications provide interface through which users can control and monitor various aspects of IoT system. Applications also allow users to view the system status and view or analyze the processed data.

Logical Design of IoT

2. IoT communication Models:

Request-Response



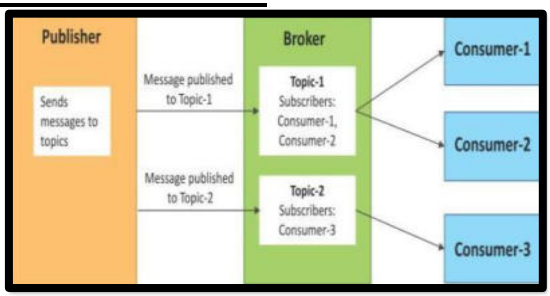
- In this communication model, client sends requests to the server and the server responds to the requests.
- When the server receives a request it decides how to respond, fetches the data, retrieves resource representations, prepares the response and then sends the response to the client.

Logical Design of IoT

IoT communication Models:

Publish-subscribe:

Publish-subscribe:

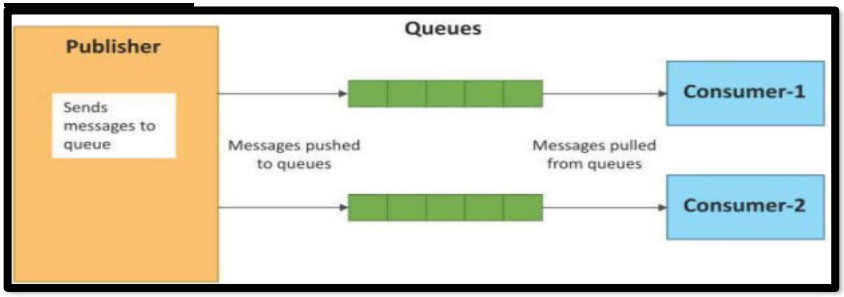


- Publish-subscribe is a communication model that involves publishers brokers and consumers.
- Publishers are the source of data, publishers send the data to the topics which are managed by the broker. Publishers are not aware of the consumers.
- Consumers subscribe to the topics which are managed by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

Logical Design of IoT

IoT communication Models:

Push –Pull:

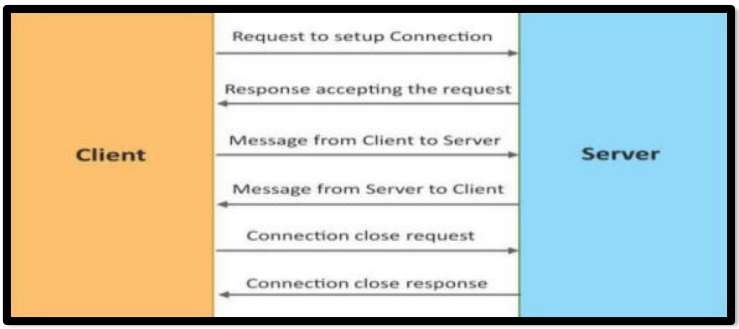


- Push pull is a communication model in which the data producers push the data to queues and the consumers pull the data from the queues. Producers do not need to be aware of the consumers.
- Queues decouples producers and consumers and they act as buffers.

Logical Design of IoT

IoT communication Models:

Exclusive pair:



- This communication model 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.

3. IoT Communication APIs:

REST based Communication APIs:

REST stands for representational state transfer

REST APIs follow the **request-response communication model**

REST architectural constraints are as follows:

- **Client-server**, the reason behind client server constraint is the separation of concerns. Client 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 the concern of the client. And this separation allows client and server to be independently developed and updated.

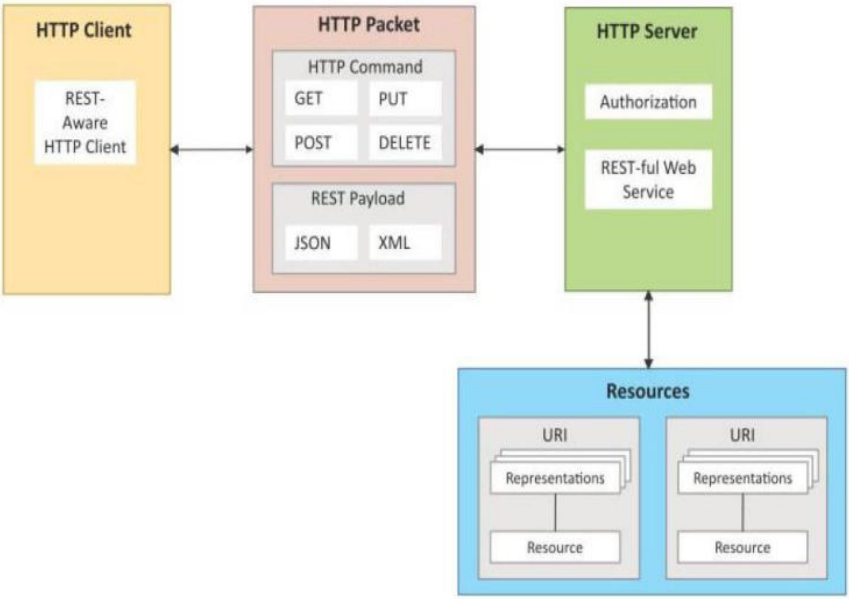
- **Stateless:** each request from client to server must contain all the information necessary to understand the request.

- **Cacheable:** if a response is cacheable then a client cache is given the right to reuse that response data for later equivalent requests. Caching can eliminate some interactions and improve efficiency and scalability.

- **Layered systems:** layered system constraint will make each component cannot see beyond the immediate layer with which they are interacting.

Logical Design of IoT

- **Uniform interface**, uniform interface constraint requires that the method of communication between a client and a server must be uniform.
- **Code on demand**, server can provide executable code or scripts for clients to execute in their context.



Logical Design of IoT

- RESTful web service is a "web API" implemented using HTTP and REST principles.

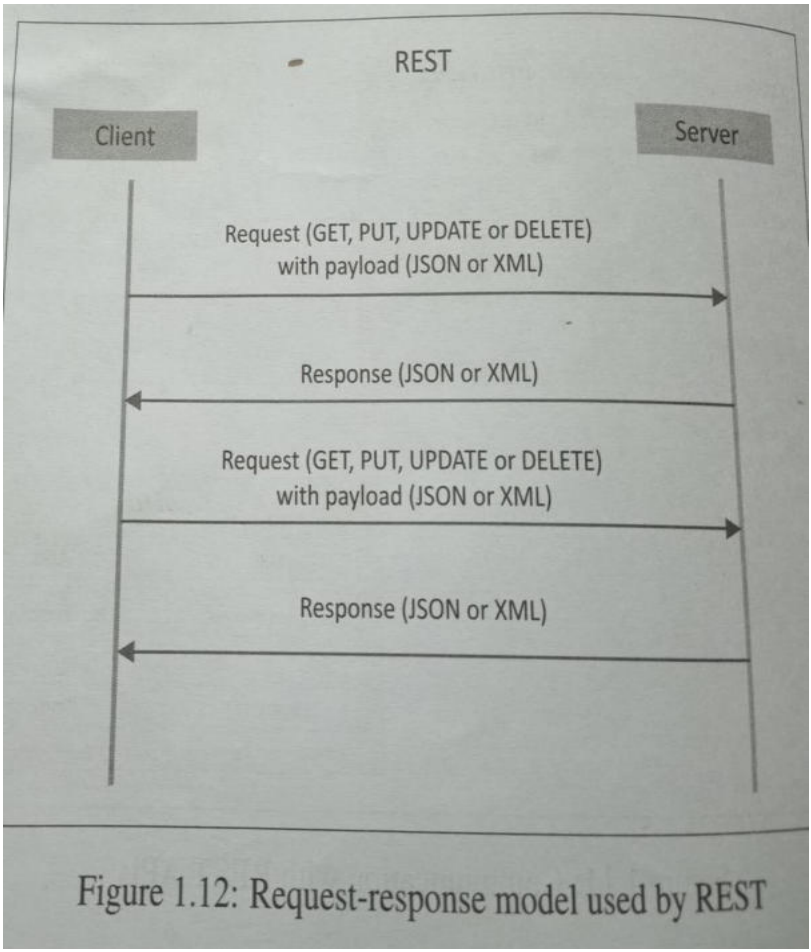


Figure 1.12: Request-response model used by REST

Logical Design of IoT

- web service is a "web API" implemented using HTTP and REST principles.

HTTP Method	Resource Type	Action	Example
GET	Collection URI	List all the resources in a collection	http://example.com/api/tasks/ (list all tasks)
GET	Element URI	Get information about a resource	http://example.com/api/tasks/1/ (get information on task-1)
POST	Collection URI	Create a new resource	http://example.com/api/tasks/ (create a new task from data provided in the request)
POST	Element URI	Generally not used	
PUT	Collection URI	Replace the entire collection with another collection	http://example.com/api/tasks/ (replace entire collection with data provided in the request)
PUT	Element URI	Update a resource	http://example.com/api/tasks/1/ (update task-1 with data provided in the request)
DELETE	Collection URI	Delete the entire collection	http://example.com/api/tasks/ (delete all tasks)
DELETE	Element URI	Delete a resource	http://example.com/api/tasks/1/ (delete task-1)

Table 1.1: HTTP request methods and actions

Logical Design of IoT

Websocket-based communication APIs:

Websocket API's allow bidirectional, full duplex communication between clients and servers.

Websocket API's follow the **exclusive pair communication model**

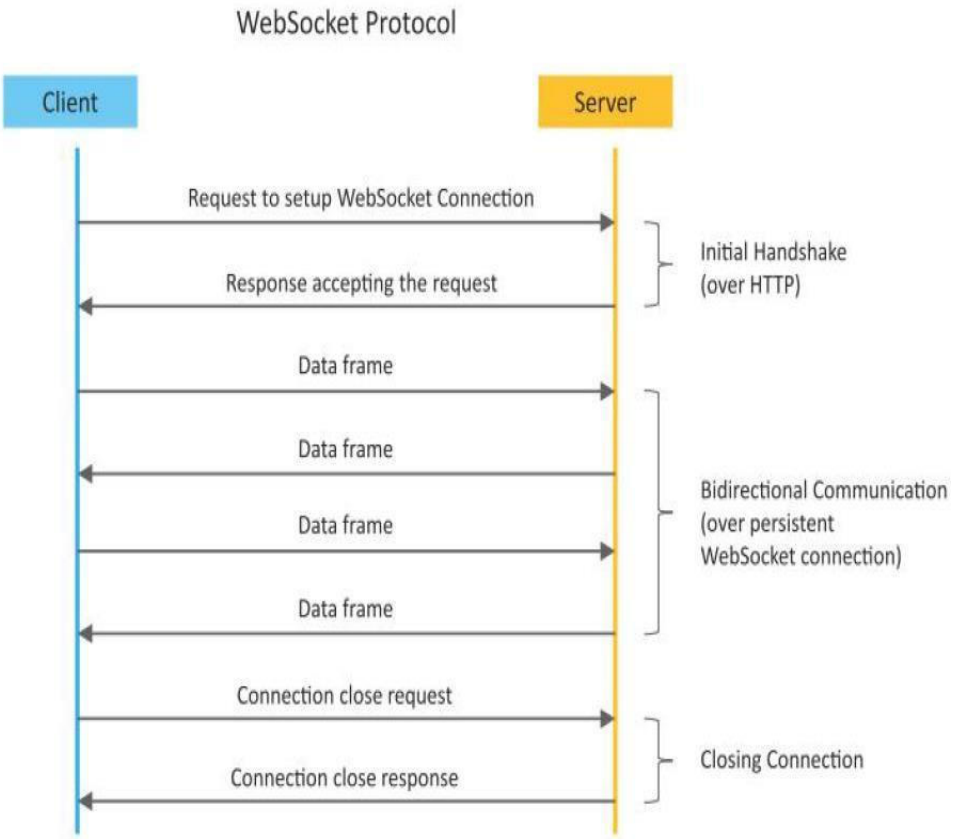
Websocket communication begins with a connection setup request sent by the client to the server. This request is sent over HTTP and the server interprets it as an upgrade request. If the server supports websocket protocol, the server responds to the websocket handshake response. After the connection is setup the client and server can send data/messages to each other in full-duplex mode.

Web socket API's reduce the network traffic and latency as there is no overhead

for connection setup and termination requests for each message.

Websocket API's is suitable for IoT applications that have low latency or high throughput requirements.

Logical Design of IoT



Physical Design of IoT

Things in IoT:

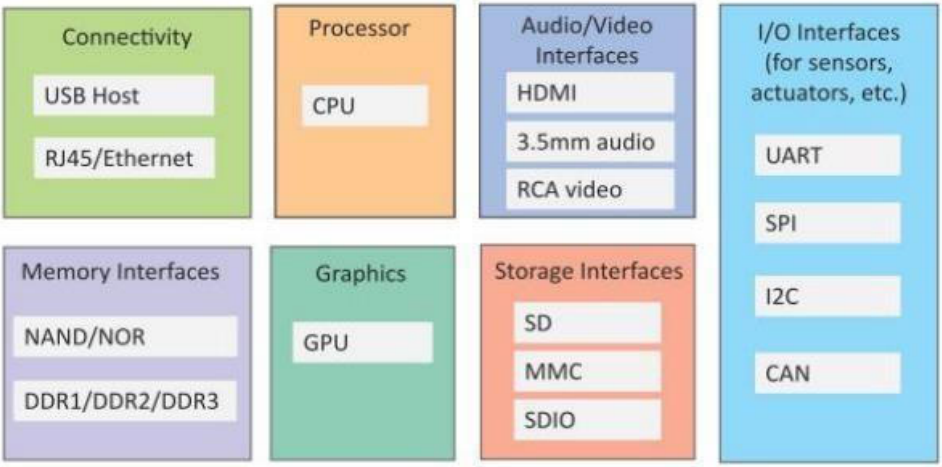
1. The "things" in IoT usually refers to IoT devices which have unique identities and can perform remote sensing, actuating and monitoring capabilities.

2. IoT devices can exchange data with other connected devices and applications or collect data from other devices and process the data either locally or send the data to the centralized servers or cloud based application back-ends for processing the data or perform some tasks locally and other tasks within the IoT infrastructure based on temporal and space constraints.

3. Fig shows the block diagram of a typical IoT device, an IoT device may consist of several interfaces for connections to other devices, both wired and wireless, these include 1. I/O interfaces for sensors, 2) interfaces for Internet connectivity, 3) memory and storage interfaces, 4) audio/video interfaces.

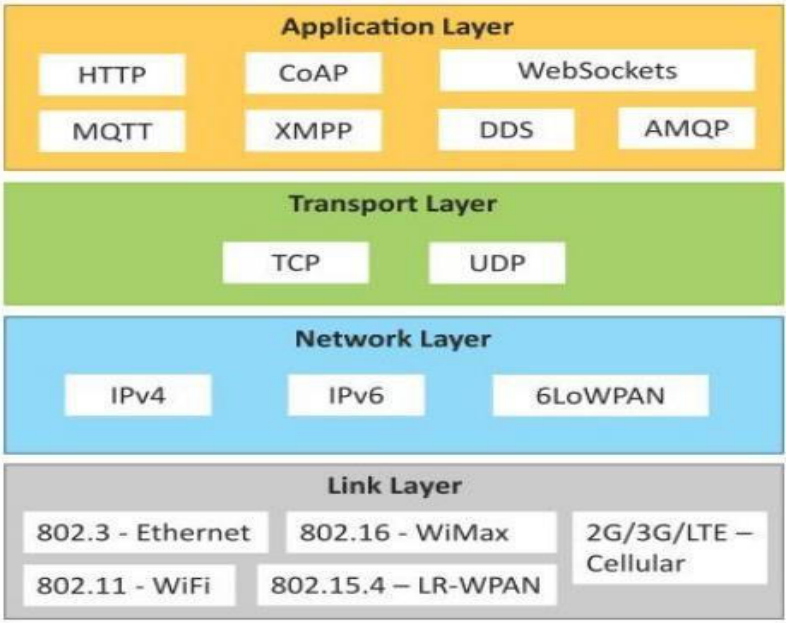
4. An IoT device can collect various types of data and the sensed data can be communicated either to other devices or cloud based storage. IoT devices can be connected to actuators that allow them to interact with other physical entities in the vicinity of the device (ex: relay switch connected to IoT device can turn an appliance on/off based on the commands sent to the IoT Device over the internet.)

Physical Design of IoT



UART- universal asynchronous receiver- transmitter
HDMI-high definition
multimedia interface RCA-root
cause analysis
RJ45-registered jack
SPI-serial peripheral interface
CAN- controller area network
SD- secure digital
MMC-multimedia car
SDIO-secure digital input output
DDR- Double data rate

Physical Design of IoT



Physical Design of IoT

IoT protocols:

1. Link layer protocols
2. Network layer protocols
3. Transport layer protocols
4. Application layer protocols

Application Layer Protocols:

Application layer protocols define how the applications interface with the lower layer protocols to send the data over the network.

Application layer protocols enable process to process connections using ports.

1. **HTTP:** Hypertext Transfer protocol is the application layer protocols that forms the foundation of the world wide web, HTTP includes commands such as GET, PUT, POST, DELETE, HEAD, TRACE and 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. HTTP uses Universal Resource Identifiers to identify HTTP resources. HTTP client can be a browser or an application running on the client.

Physical Design of IoT

2. **CoAP:** Constrained Application protocol is an application layer protocol for **machine to machine** applications, meant for constrained environments with constrained devices and constrained networks.

   CoAP protocol 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 supports methods such as GET,PUT,POST and DELETE.

3. **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.

4. **MQTT: Message Queue Telemetry Transport** is a light weight messaging protocol based on the **publish-subscribe model**. MQTT uses a client-server architecture where the client connects to the server 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.

Physical Design of IoT

5. **XMPP: Extensible Messaging and Presence Protocol** is a protocol for real time communication and streaming XML data between network entities. XMPP powers wide ranges 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. (CLIENT-TO-SERVER AND SERVER-TO-SERVER Communication will be supported here)

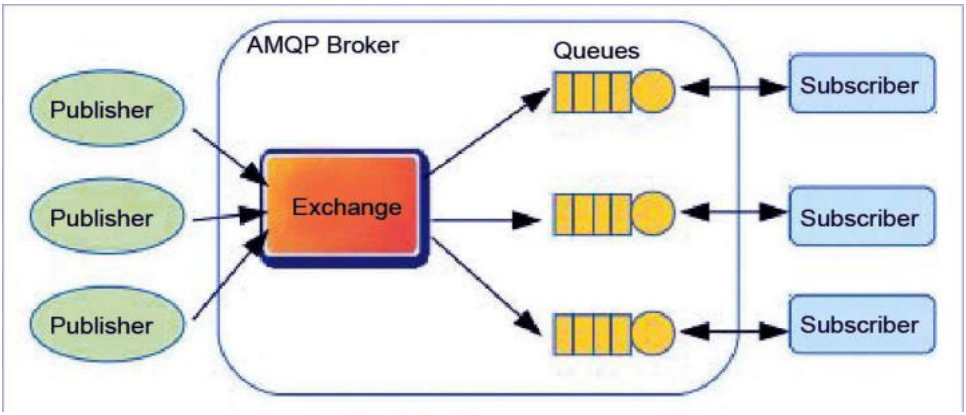
   XMPP allows real time communication between IoT DEVICES.

6. **DDS: Data Distribution service** is a data centric middleware standard for **device to device or machine to machine communication.** DDS uses a **Publish- subscribe model** where publishers create topics to which subscribers can subscribe. Publisher is an object responsible for data distribution and the subscriber is responsible for receiving published data.

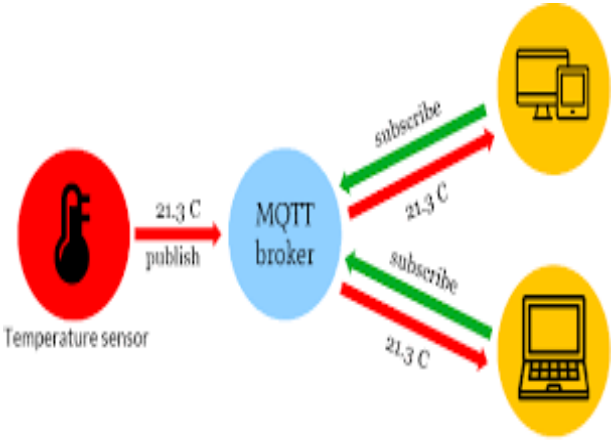
7. **AMQP: Advanced message queuing protocol** is an open application layer protocol for business messaging. AMQP supports **both point-to-point and publisher/subscriber models, routing and queuing**. AMQP brokers receive messages from publishers and route them over connections to consumer.

Physical Design of IoT

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.



Physical Design of IoT



Schematic data flow from sensor (machine) to device (machine)

Physical Design of IoT

Transport Layer:

Transport layer protocols provide **end to-end message** transfer capability independent of the underlying network.

The transport layer is responsible for providing functions such as **error control, segmentation, flow control and congestion control**.

TCP: Transmission control protocol is the most widely used transport layer protocol that is used by **web browsers, email programs and file transfers**.

TCP is a connection oriented and stateful protocol, TCP ensures reliable transmission of packets in-order.

TCP also provides error detection capability so that duplicate packets can be discarded and lost packets are retransmitted.

TCP also provides flow control capability of TCP ensures that rate at which the sender sends the data is no too high for the receiver to process.

The congestion control capability of TCP helps in avoiding network congestion and congestion collapse which can lead to degradation of network performance.

Physical Design of IoT

UDP:

Unlike TCP which requires carrying out an initial setup procedure, UDP is a 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.

Network/Internet Layer:

Network layers are responsible for sending of IP datagrams from the source network to the destination network.

This layer perform the host addressing and packet routing.

The datagram contain the source and destination addresses which are used to route them from the source to destination across multiple networks.

Host Identification is done using IP addressing schemes such as IPv4, IPv6

Physical Design of IoT

IPv4:

Internet Protocol version 4 is used to identify the devices on a network.

IPv4 uses a 32 bit address scheme that allows a total of 2^{32} or 4,294,967,296 addresses.

As more and more devices introduced into the network IPv4 addresses becomes exhausted which is succeeded by IPv6.

IPv6:

Internet Protocol version 6 is the newest version of internet protocol and successor to IPv4.

IPv6 uses 128bit address scheme that allows a total of 2^{128} or $3.4*10^{38}$ addresses.

6LoWPAN: IPv6 over low power wireless personal area networks brings 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 250kb/s.

Physical Design of IoT

LINK LAYER PROTOCOLS:

Link layer protocols determine how the data is physically sent over the networks physical layer or medium.

Protocols:

802.3-Ethernet:

IEEE 802.3 is a collection of wired ethernet

standards for the link layer. Ex

802.3 standard for 10BASE5 which uses COAXIAL CABLE as a shared medium

802.3.i standard for 10BASE-T which uses COPPER

TWISTED PAIR as a shared

medium

802.3.j standard for 10BASE-F which uses FIBER OPTICS as a shared medium 802.3ae is the standard for 10Gbits/s Ethernet over fiber.

These standards provide data rates from 10Mb/s to 40Gb/s.

Physical Design of IoT

802.11-WiFi:

IEEE 802.11 is a collection of wireless local area networks

802.11a operates at 5GHz band

802.11b & 802.11g operate at 2.4GHz band 802.11n operates at 2.4/5GHz bands 802.11ac operates at 5GHz band and 802.11ad operates at 60GHz band

These standards provide data rates from 1 Mb/s to upto 6.75 Gb/s.

802.16- WiMax:

IEEE 802.16 is a collection of wireless broadband standards

These standards provide data rates 1.5Mb/s to 1 Gb/s

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.

These standards provide data rates from 40Kb/s to 250Kb/s

These standards provide low-cost and low-speed communication for power constrained devices.

Physical Design of IoT

2G/3G/4G-Mobile Communication:

There are different generations of mobile communication standards like 2G,3G,4G.

IoT devices based on these standards can communicate over cellular networks Data rates for these standards range from 9.6Kb/s to upto 100Mb/s

IoT levels and Deployment Templates

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
- **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.

IoT levels and Deployment Templates

- **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).
- **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.
- **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.

Comparison between REST and Websocket

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 unidirectional request is always sent by a client and the sever responds to the request.

on the other hand websocket is a bidirectional 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 request and the server responds to the request.

WebSocket on the other hand allow full-duplex communication between client and server i.e. Both client and server can send messages to each other independently.

Comparison between REST and Websocket

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 other. Thus each request carries HTTP headers which is an overhead. due to 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 the client and server exchange messages with minimal frame information. Thus WebSocket is suitable for real time applications.

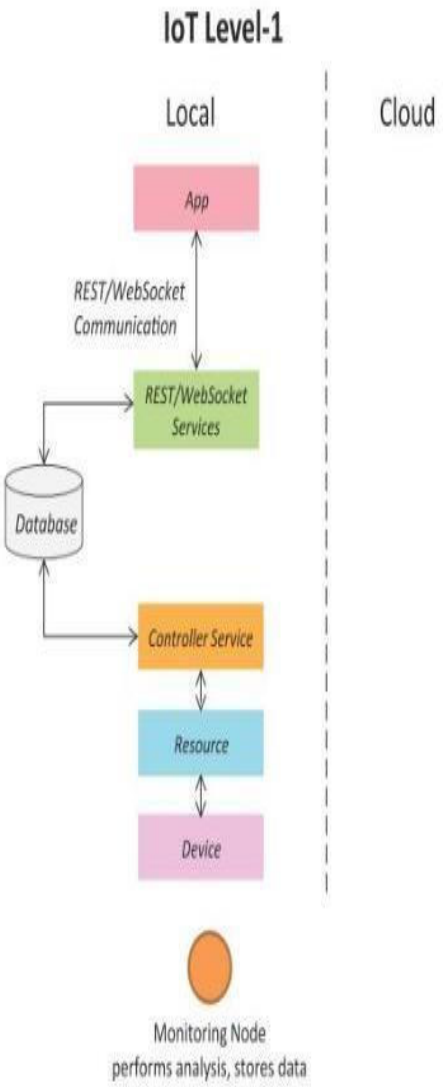
Scalability: scalability is easier in the case of REST services as requests are independent and no state information needs to be maintained by the server thus horizontal and vertical scaling solutions are possible for REST services.

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.

IoT levels and Deployment Templates

IoT LEVEL1:

- A level-1 IoT system has a single node/device that performs sensing and/or actuation, stores data, performs analysis and hosts the application
- 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.



IoT levels and Deployment Templates

Example:

Consider an 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.

The **controller service** continuously monitors the state of each light or appliance and triggers the relay switch accordingly.

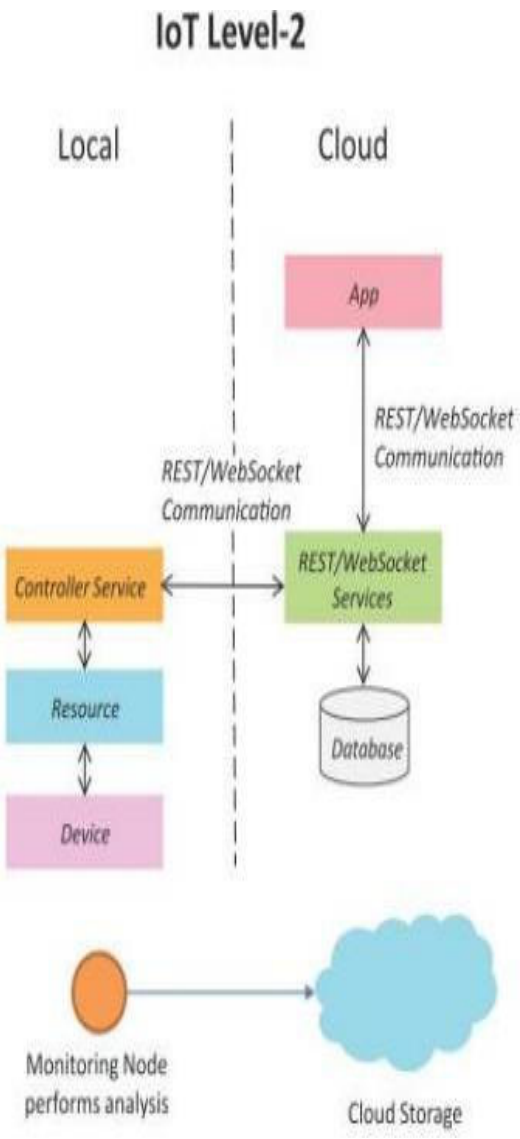
The **application** which is deployed locally has a user interface for controlling the lights or appliances.

Since the device is connected to the internet the application can be accessed remotely.

IoT levels and Deployment Templates

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.



IoT levels and Deployment Templates

Example:

Consider an 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 monito

For controlling the irrigation system **actuators such as solenoid valves** can be used.

The controller also sends the moisture data to the **computing cloud**

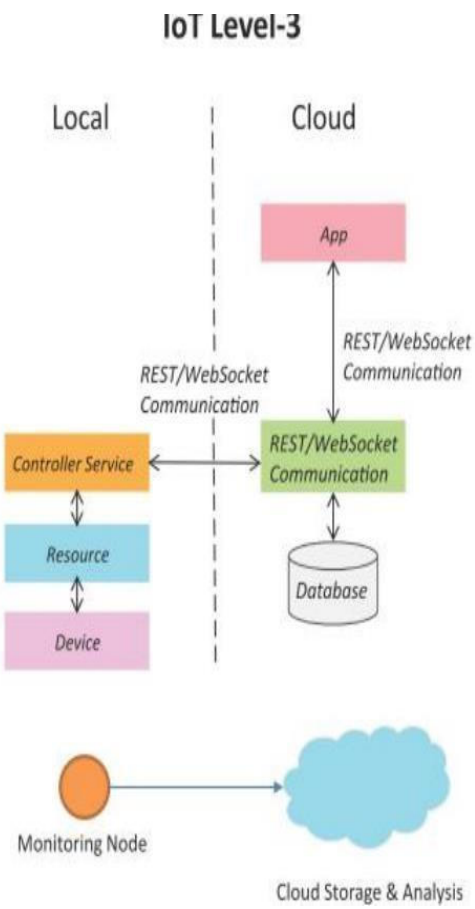
A **cloud based REST web service** is used for storing and retrieving moisture data which is stored in the cloud database

A **cloud based application** is used for visualizing the moisture levels over a period of time.

IoT levels and Deployment Templates

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.



IoT levels and Deployment Templates

Example:

Consider an IoT system **for tracking package handling**

The **system** consists of a **single node** 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.

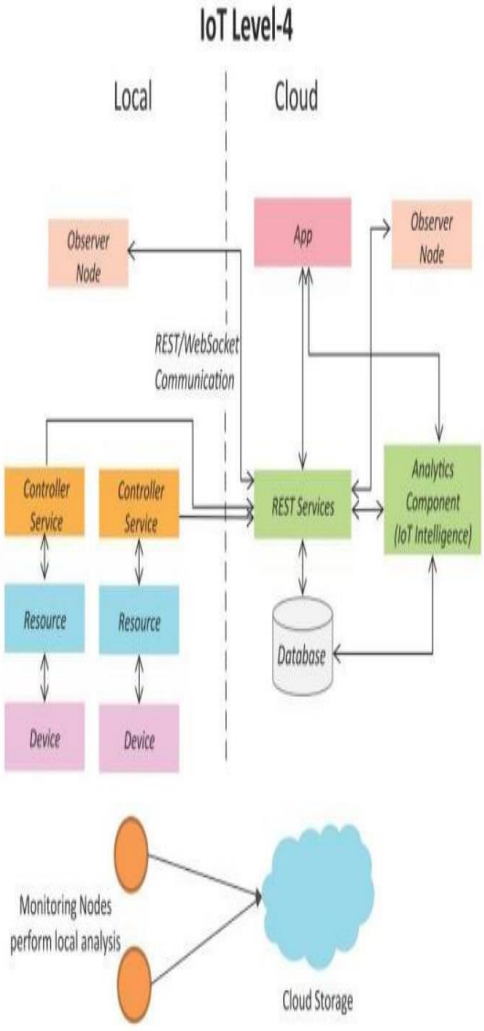
The **analysis componen**ts in the cloud can trigger alerts if the vibration levels become greater than a threshold.

The **cloud based applications** can subscribe to the sensor data feeds for viewing the real time data.

IoT levels and Deployment Templates

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.
- 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.



Example:

Consider **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.

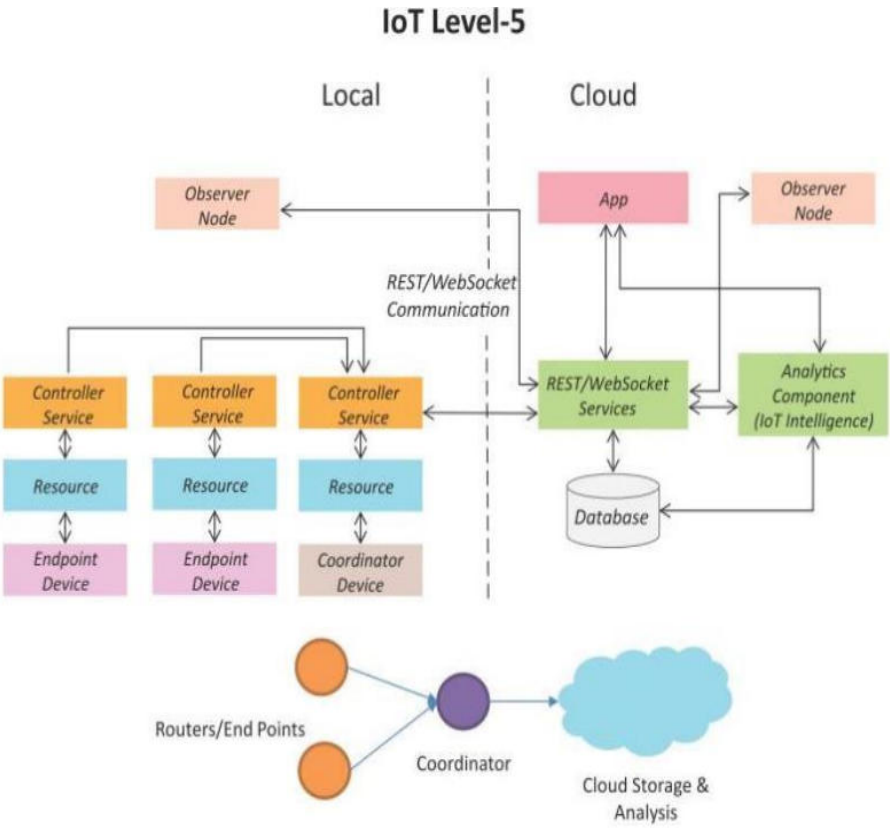
Each node runs its own **controller service** that sends the data to the cloud. The data stored in a cloud database.

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.

IoT levels and Deployment Templates

IoT LEVEL-5



IoT levels and Deployment Templates

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.

IoT levels and Deployment Templates

Example:

Consider **forest fire detection**:

System consists of **multiple nodes** placed in different locations for monitoring temperature, humidity and carbon dioxide levels in a forest.

The **end nodes** in this example are equipped with various sensors

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.

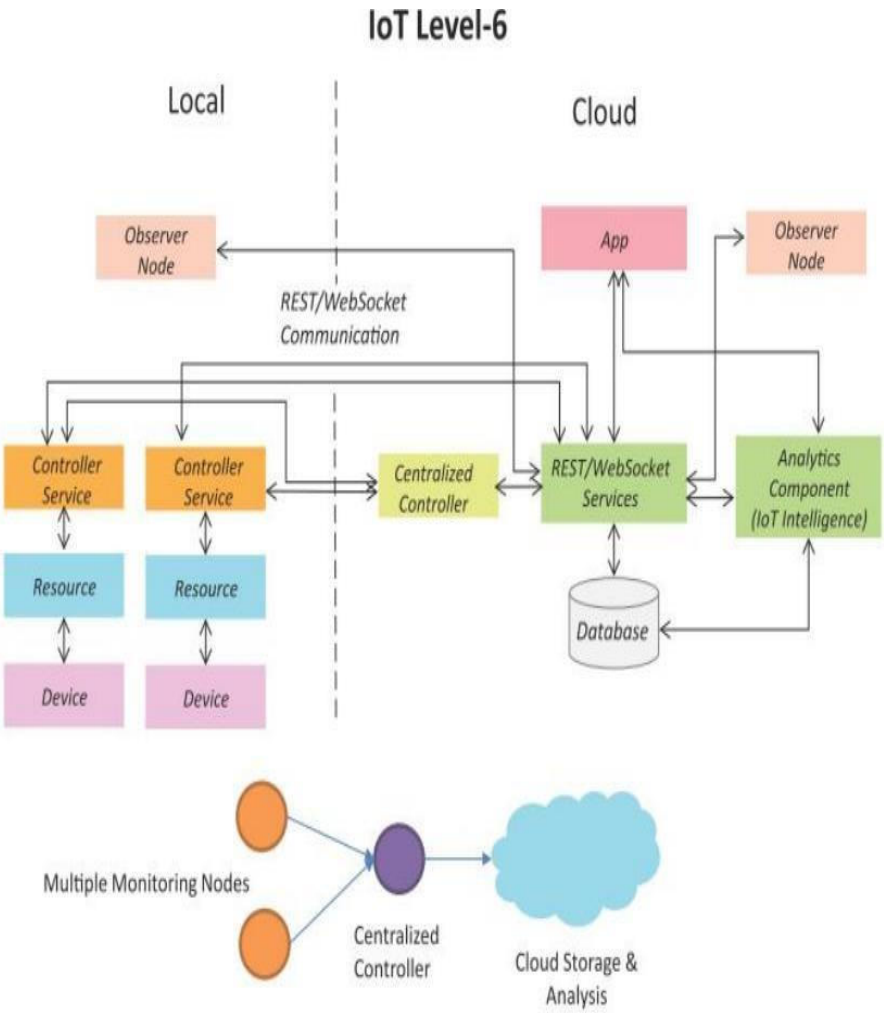
The data is stored in a **cloud database**.

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.

IoT levels and Deployment Templates

IoT LEVEL-6



IoT levels and Deployment Templates

IoT LEVEL-6

- A level-6 IoT system has multiple independent end nodes that perform sensing and/or actuation and send data to the cloud.
- Data is stored in the cloud and application is cloud-based.
- The analytics component analyzes the data and stores the results in the cloud database.
- The results are visualized with the cloud-based application.
- The centralized controller is aware of the status of all the end nodes and sends control commands to the nodes.

IoT levels and Deployment Templates

IoT LEVEL-6

Let us consider an example of **weather monitoring system**

The **system nodes** consists of **multiple** placed in different locations for monitoring temperature, humidity and pressure in an area.

The **end nodes** are equipped with various **sensors(such as**

The **end nodes** send the data to the cloud in the real time using a **Websocket service.**

The data is stored in a **cloud database**.

The **analysis of data** is done in the cloud to aggregate the data and make predictions.

A **cloud based application** is used for visualizing the data.

IoT Enabling Technologies

IoT is enabled by several technologies including wireless sensor networks, cloud computing, big data analytics, embedded systems, security protocols and architectures, communication protocols, web services, mobile internet and semantic search engines,

Wireless Sensor Networks:

a wireless sensor network comprises of distributed devices with sensors which are used to monitor the environmental and physical conditions.

A wireless sensor networks will be consists of a number of end nodes and routers and a coordinators.

End nodes have several sensors attached to them, end nodes can act as routers where routers are responsible for routing the data packets from end nodes to the coordinator.

The coordinator collects the data from all the nodes.

Coordinator also acts as a gateway that connects the wireless sensor networks to the internet.

IoT Enabling Technologies

Example:

Weather monitoring system uses WSNs in which the nodes collect temperature, humidity and other data, which is aggregated and analyzed.

Indoor air quality monitoring systems use WSNs to collect data on the indoor air quality and concentration of various gases

Soil moisture monitoring systems use WSNs to monitor soil moisture at various locations

Surveillance systems use WSNs for

collecting surveillance data Smart

grids use WSNs for monitoring the

grid at various points

Structural health monitoring systems use WSNs to monitor the health of structures by collecting vibration data from sensor nodes deployed at various points in the structure.

WSNs are enabled by wireless communication protocol such as IEEE 802.15.4. ZigBee is one of the most popular Wireless technologies used by WSNs.

ZigBee operates at 2.4GHz frequency and offers data rate upto 250kb/s and range from 10-100 meters depending on the power output and environmental conditions.

IoT Enabling Technologies

Cloud Computing:

Cloud Computing involves provisioning of computing, networking and storage resources on demand and providing these resources as metered services to the users, in a **pay as you go model.**

Cloud computing services are

1. **INFRASTRUCTURE AS A SERVICE** provides the users the ability to provision computing and storage resources.
2. **PLATFORM AS A SERVICE** provides the users the ability to develop and deploy application in the cloud using the development tools, application programming interfaces, software libraries and services provided by the cloud service provider.
3. **SOFTWARE AS A SERVICE,** provides the users a complete software application or the user interface to the application itself.

Big Data Analytics:

Big data is defined as collections of data sets whose volume, velocity or variety is so large that its difficult to store, manage, process and analyze the data using traditional databases and data processing tools.

Example:

Sensors data generated by IoT systems such as weather monitoring stations Machine sensor data collected from sensors embedded in industrial and energy systems for monitoring their health and detecting failures.

Health and fitness data generated by IoT devices such as wearable fitness bands Data generated by IoT systems for location and tracking of vehicles

Data generated by retail inventory monitoring systems

Volume: big data is used for massive scale data that is difficult to store, manage and process using traditional databases and data processing architectures.

Velocity: velocity is another important characteristic of big data which refers to how fast the data is generated and how frequently it varies.

Variety: variety refers to the forms of the data. Big data comes in different forms such as structured or unstructured data, including text data, image, audio, video and sensor data.

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.

Embedded systems:

An embedded system is a computer system that has computer hardware and software embedded to perform specific tasks.

Key components of an embedded syst

Embedded systems run embedded operatin

Applications

Home Automation

1. Smart Lighting

Smart Lighting solutions for home achieve energy savings by sensing the human movements and their environments and controlling the lights accordingly.

Wireless enabled and internet connected lights can be controlled remotely from IoT applications such as a mobile or web application

Smart lights with sensors for occupancy, temperature, lux level etc can be configured to adapt the lighting based on the ambient conditions sensed in order to provide a good ambiance.

2. Smart Appliances:

Smart refrigerators can keep track of the items stored and send updates to the users when an item is low on stock.

Smart TVs allows users to search and stream videos and movies from the internet on a local storage drive, search TV channel schedules and fetch news and other content from the internet.

Applications

Home Automation

3. Intrusion Detection:

Home Intrusion Detection systems use security cameras and sensors to detect intrusions and raise alerts. Alerts can be in the form of an SMS or an email sent to the user.

4. Smoke/Gas Detectors:

Smoke detectors are installed in homes and buildings to detect smoke that is typically an early sign of fire. Smoke detectors use optical detection, ionization or air sampling techniques to detect smoke.

Alert raised by smoke detectors can be in the form of signals to a fire alarm system.

Gas detectors can detect the presence of harmful gases such as carbon monoxide, liquid petroleum gas etc.

Applications

CITIES

1. Smart Parking:

smart parking make the search for parking space easier and convenient for drivers.

Smart parking's are powered by IoT SYSTEMS that detect the number of empty parking slots and send the information over the internet to smart parking application back-ends.

These applications can be accessed by the drivers from smart-phones,tablets and in-car navigation systems.

In smart parking sensors are used for each parking slot, to detect the whether the slot is empty or occupied. This information is aggregated by a local controller and then sent over the internet to the database.

2. Smart lighting:

smart lighting allows lighting to be dynamically controlled and also adaptive to the ambient conditions. Smart lights connected to the internet can be controlled remotely to configure lighting schedules and lighting intensity.

Smart lights equipped with sensors can communicate with other lights and exchange information on the sensed ambient conditions to adapt the lighting.

Applications

CITIES

3. Smart roads:

Smart roads equipped with sensors can provide information on **driving conditions, travel time estimates and alerts in case of poor driving conditions, traffic congestions and accidents,**

Such information can help in making the roads safer and help in reducing traffic jams

Information sensed from the roads can be communicated via internet to cloud based applications and social media and disseminated to the drivers who subscribe to such applications.

4. Structural Health Monitoring:

This system uses a network of sensors to monitor the vibration levels in the structures such as bridges and buildings.

The data collected from those sensors is analyzed to assess the health of the structures.

By analyzing the data its possible to detect cracks and mechanical breakdowns, locate the damages to a structure and also calculate the remaining life of the structure,

Using such systems, advance warnings can be given in the case of imminent failures of the structure.

Applications

CITIES

5. Surveillance

Surveillance of infrastructure, public transport and event in cities is required to ensure safety and security.

City wide surveillance infrastructure comprising of large number of distributed and internet connected video surveillance cameras can be created.

The video feeds from surveillance cameras can be aggregated in cloud based scalable storage solutions.

Cloud based video analytics applications can be

6. Emergency Response

IoT systems can be used for monitoring the critical infrastructure in cities such as buildings, gas and water pipelines, public transport and power substations.

IoT systems for fire detection, gas and water leakage detection can help in generating alerts and minimizing their effects on the infrastructure.

Applications

Environment:

1. Weather Monitoring

IoT based weather Monitoring systems can collect data from a number of sensors attached(such as temp, humidity & pressure etc) and send the data to cloud based applications and storage back ends.

The data collected in the cloud can then be analyzed and visualized by cloud based applications.

Weather alerts can be sent to the subscribed users from such applications.

2. Air Pollution Monitoring:

IoT based air pollution monitoring systems can monitor emission of harmful gases by factories and automobiles using gaseous and meteorological sensors.

The collected data can be analyzed to make informed decision on pollutions control approaches.

3. Noise Pollution Monitoring:

IoT based noise pollution monitoring systems use a number of noise monitoring stations that are deployed at different places in a city.

The data on noise levels from the stations is collected on servers or in the cloud. The collected data is then aggregated to generate noise maps.

Applications

Environment:

4. Forest fire detection:

Early detection of forest fires can help in minimizing the damage. IoT based forest fire detection systems use a number of monitoring nodes deployed at different locations in a forest.

Each monitoring node collects measurements on ambient conditions including temperature, humidity, light levels etc.

5. River Floods Detection:

River Floods can cause extensive damage to the natural and human resources and human life.

Early warning of floods can be given by monitoring the water level and flow rate.

IoT based river flood monitoring system use a number of sensor nodes that monitor the water level and flow rate

Data from a number of such sensor nodes is aggregated in a server or in the cloud.

Monitoring applications raise alerts when rapid increase in water level and flow rate is detected.

Applications

ENERGY:

SMART GRIDS, One of the definitions for the smart grid is that the smart grid is a communication network on top of the electricity grid to gather and analyze data from different components of a power grid to predict power supply and demand which can be used for power management

1. In electricity generation, IoT can be used to monitor electricity generation of different kinds of power plants (such as coal, wind, solar, biomass), gas emissions, energy storage, energy consumption, and predict necessary power to supply consumers.

2. IoT can be used to acquire electricity consumption, dispatch, monitor and protect transmission lines, substations, and towers, manage and control equipment.

3. IoT can be used in customer side in smart meters to measure different types of parameters, intelligent power consumption, interoperability between different networks, charging and discharging of electric vehicles, manage energy efficiency and power demand.

Applications

2. RENEWABLE ENERGY SYSTEMS

Due to the variability in the output from renewable energy sources integrating them into the grid can cause grid stability and reliability problems. Variable output produces local voltage swings that can impact power quality.

3. PROGNOSTICS

Energy systems have a large number of critical components that must function correctly so that the systems can perform their operations correctly.

Applications

Retail:

Inventory Management,

Over stocking of products can result in additional storage expenses and risk, under-stocking can lead to loss of revenue. IoT systems using RFID tags can help in inventory management and maintaining the right inventory levels.

RFID tags attached to the products allow them to be tracked in real time so that the inventory levels can be determined accurately and products which are low on stock can be replenished.

Tracking can be done using RFID readers attached to the retail store shelves or in the warehouse. IoT systems enable remote monitoring of inventory using the data collected by the RFID readers.

Smart Payments: Customers can store the credit card information in their NFC enabled smart phones and make payments by bringing the smart phones near the point of sale terminals.

Smart Vending Machines

Smart vending machines connected to the internet allow remote monitoring of inventory levels, elastic pricing of products, promotions and contact less payments using NFC.

Applications

Smart phone applications that communicate with smart vending machines allow user preferences to be remembered and learned with time.

Applications

Logistics:

Route Generation and scheduling

It generates end-to-end routes using com

Fleet tracking, this system uses GPS technology to track the locations of the vehicles in real time.

Alerts can be generated in case of deviations in planned routes.

The vehicle locations and routes data can be aggregated and analyzed for detecting bottlenecks in the supply chain such as traffic congestions on routes, assignments and generation of alternative routes and supply chain optimization.

Shipment monitoring, this system allow monitoring the conditions inside containers, for ex: containers carrying fresh food produce can be monitored to prevent spoilage of food.

IoT based shipment monitoring systems use sensors such as temperature, pressure, humidity for instance to monitor the conditions inside the containers and send the data to the cloud, where it can be analyzed to detect food spoilage.

The analysis and interpretation of data on the environmental conditions in the container and food truck positioning can enable more effective routing decisions in real time.

Applications

Remote vehicle diagnostics:

Remote vehicle diagnostic systems can detect faults in the vehicles or warn of impending faults.

These diagnostic systems use on-board IoT devices for collecting data on vehicle operation and status of various vehicle sub-systems.

Such data can be captured by integrating on board diagnostic systems with IoT devices using protocols such as CAN bus.

Applications

Agriculture:

1. Smart irrigation:

Smart irrigation systems can improve crop yields while saving water. Smart irrigation systems use IoT Devices with soil moisture sensors to determine the amount of moisture in the soil and releases the flow of water through the irrigation pipes only when the moisture levels go below a predefined threshold.

Smart irrigation systems also collect moisture level measurements on a server or in the cloud where the collected data can be analyzed to plan watering schedules.

2. Green house control:

Green houses are structures with glass or plastic roofs that provide conducive environment for growth of plants.

The climatological conditions inside a green house can be monitored and controlled to provide the best conditions for growth of plants.

The humidity, temperature, soil moisture, light and carbon dioxide levels are monitored using sensors and the climatological conditions are controlled automatically using actuation devices.

Applications

Industry:

1. Machine Diagnosis and Prognosis:

Machine prognosis refers to predicting the performance of a machine by analyzing the data on the current operating conditions and how much deviations exists from the normal operating conditions.

Machine diagnosis refers to determining the cause of a machine fault.

IoT plays a major role in both prognosis and diagnosis of industrial machines. Industrial machines have a large number of components that must function correctly for the machine to perform its operations.

Sensors in machines can monitor the operating conditions such as temperature, vibration levels etc.

IoT based systems integrated with cloud based storage and analytics back ends can help in storage, Collection and analysis of such massive scale machine sensor data.

Applications

2. Indoor Air Quality Monitoring:

Monitoring indoor air quality in factories is important for health and safety of the workers.

Harmful and toxic gases such as carbon monoxide, nitrogen monoxide, nitrogen dioxide etc can cause serious health problems.

IoT based gas monitoring systems can help in monitoring the indoor air quality using various gas sensors.

Applications

HEALTH AND LIFESTYLE:

1. Health and Fitness monitoring:

Wearable devices from a type of wireless sensor networks called body area networks in which the measurements from a number of wearable devices are continuous sent to a master node(such as smart phone) which then sends the data to a server or a cloud based back end for analysis and archiving .

Health care providers can analyze the collected health care data to determine any health conditions or anomalies.

Commonly used sensors include: body temperature, heart rate, pulse oximeter oxygen saturation, blood pressure, electrocardiogram, movement and electroencephalogram.

2. Wearable electronics:

Wearable electronics such as wearable gadgets provide various functions and features to assist us in our daily activities and making us lead healthy lifestyles.

Dr. N. Penchalaiah, Associate Professor, Department of AI&ML, Annamacharya University

UNIT 2 Prototyping IoT Objects using Microprocessor/Microcontroller

What Is Prototyping in IoT?

IoT prototyping is the action of experimenting and implementing design ideas into preliminary versions of a finished product.

Essentially, it involves trying out and testing different ways to bring something from the planning phase to reality.

In the world of IoT, a prototype could be:

- **A user interface (UI)**
- **A hardware device**
- **Backend software**
- **Connectivity of a system**

Working principles of sensors and actuators :

SENSOR: Sensors are such devices which are used to convert physical quantities, events or characteristics into the electrical signals for the purpose of monitoring and controlling. So sensor takes input from environment and converts into electrical form then fed to the system or controller. Sensor works as an input device.

Example- **Thermocouple, photo cell, RTD, LVDT, strain gauge, Load cell etc.**

Figure- Block Diagram of a Sensor

ACTUATOR : Actuators are such devices which deliver physical quantity (like force or motion) to the environment by converting source energy according to control signal received that can be in electrical form. Here source energy can be pneumatic, hydraulic or electric type and motion produced (by actuator) can be either linear or rotary. Actuator acts as output device. For examples- different types of electric motor actuator, heaters, electro pneumatic actuator, electro-hydraulic actuator, magnetic actuator etc.

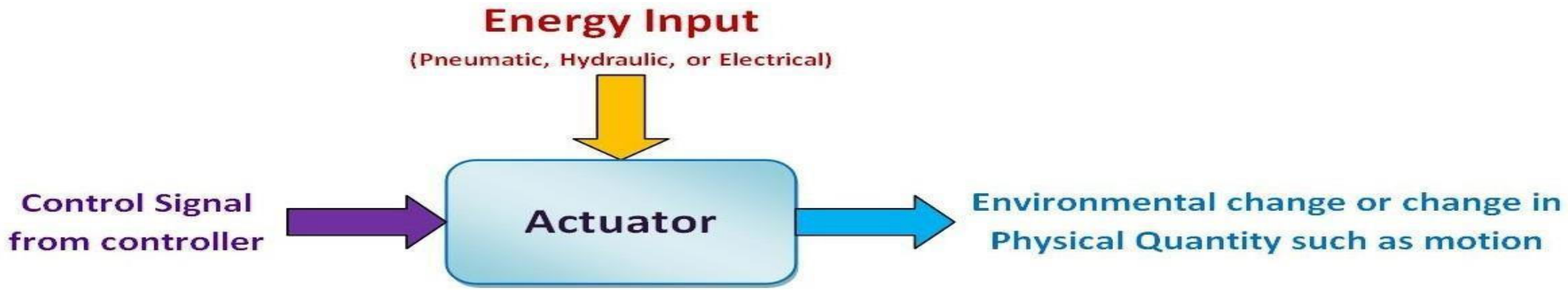


Figure- Block diagram of an actuator

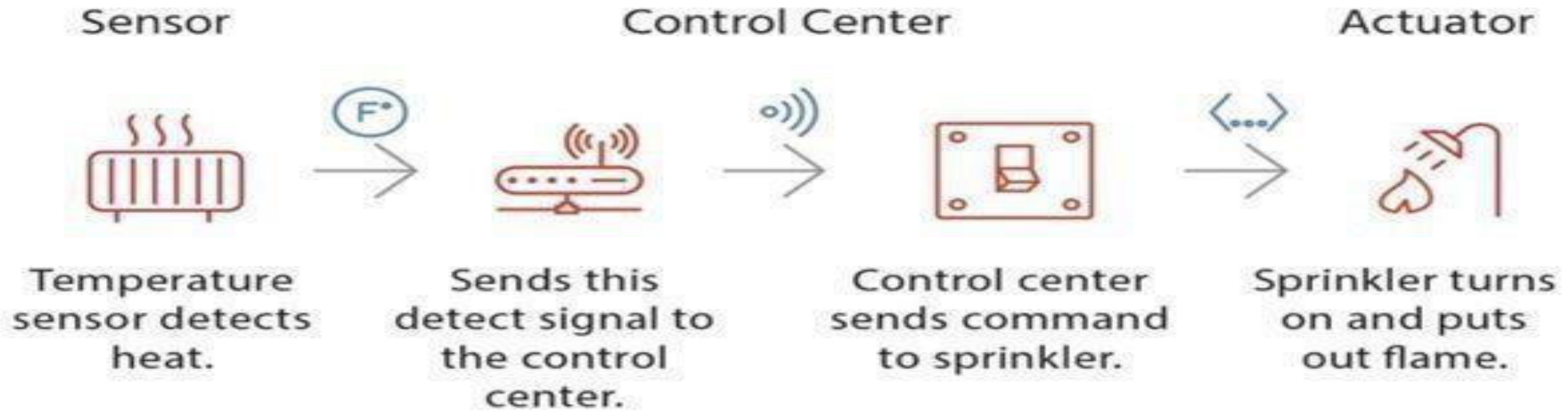
	SENSOR	ACTUATOR
1	Sensor converts physical quantities and characteristics into electrical signals.	Actuator converts electrical signals into physical action such as force and motion.
2	It acts as an input device in any control system and placed in input port	It acts as an output device in a control system and placed in output port
3	Sensor takes input from environment and senses surroundings condition.	Actuator takes input from output signal conditioning unit of system.
4	Sensor gives output to input signal conditioning unit of system to convert into electrical form.	It gives output to environment and makes impact on load to control parameters.
5	It gives information to the system about environment condition to monitor and control.	It accepts command from system to deliver physical action.
6	Sensors are often used to measure process pressure, temperature, fluid levels, flow, vibration, speed etc.	Actuators are often used to operate control valves, dampers, guide vanes, and to move objects from one place to another, to move conveyor belts in robotic arms movement etc..
7	Sensor examples- Thermocouple, photo cell, RTD, LVDT, strain gauge, Load cell, hall sensors, differential flow meters, speed probes, PH meter etc	Actuator examples- motor actuator, servo motor, stepper motor, heaters, electro pneumatic actuator, electro- hydraulic actuator, magnetic actuator etc

SENSOR AND ACTUATORS IN IOT : The Internet of Things is defined as a paradigm in which objects equipped with sensor, actuator and processor communicate with each other to serve a meaningful purpose. IoT can also be seen simply as an interaction between the physical and digital worlds. Once a stand-alone device and application now has the ability to connect to the network via sensors, actuators, processors and transceivers.

An IoT device is basically made up of a Physical object (things) + Controller (brain) + Networks (Internet) along with sensor and actuator. Sensor and Actuators are devices that enable interaction with the physical world in IoT technology. The following diagram shows how a sensor and actuators works together.

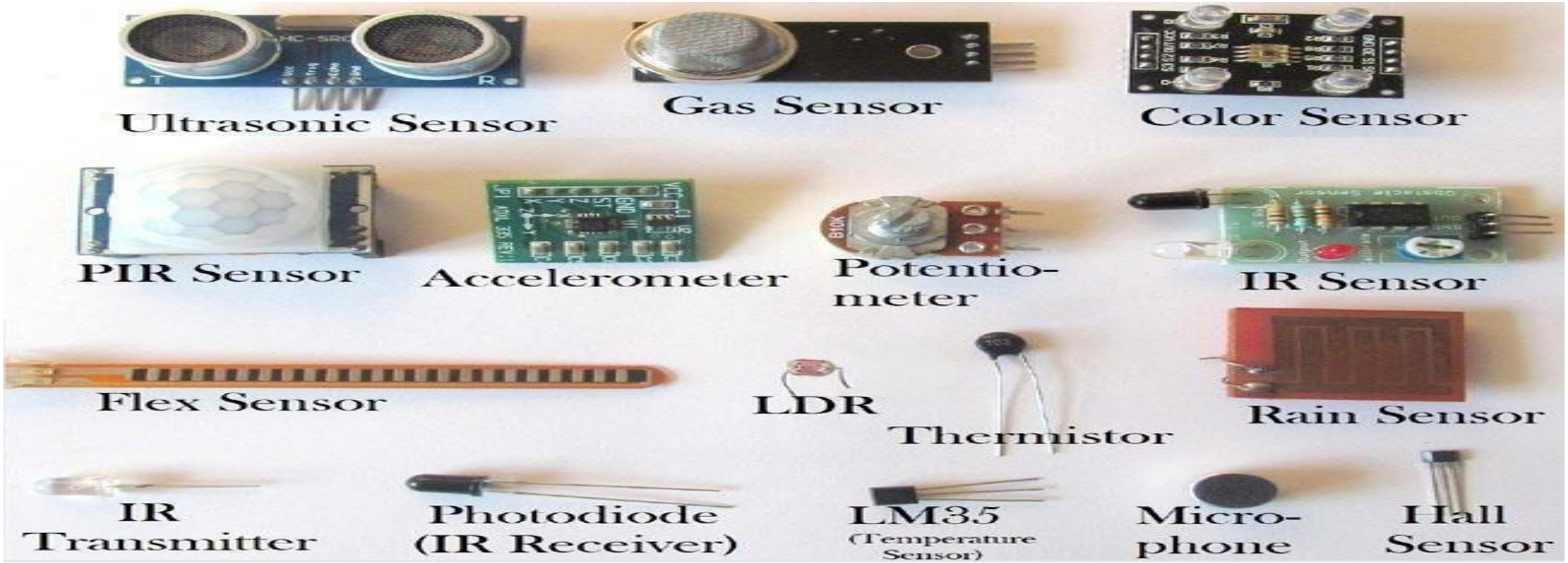
Figure- Sensor and actuator in a system

Here in basic form, sensors in the device sense the environment and fed to the controller, then based on set value, control signal is generated for the actuator to perform actions required to maintain set value. Such control signal is sent to actuator that moves or controls the mechanism or the system. It is to be noted that an actuator needs external energy to perform action.

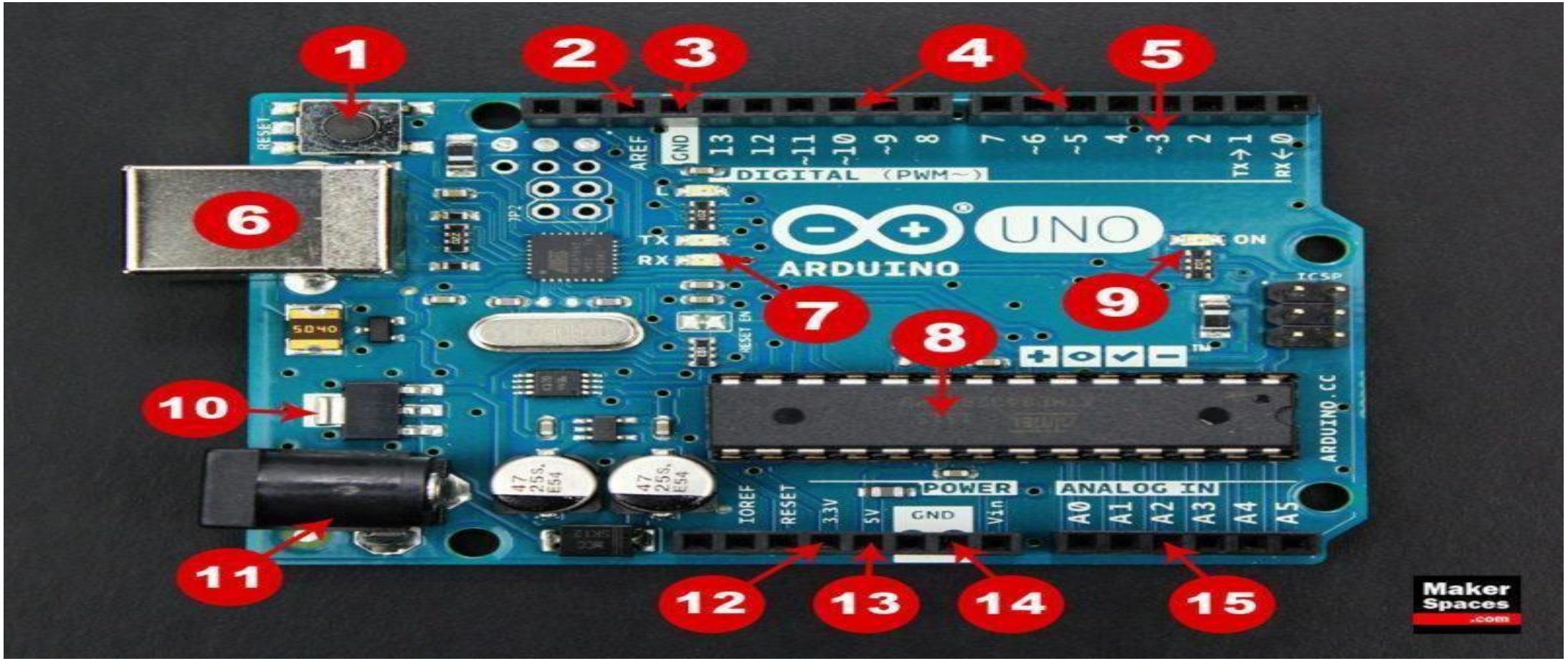


From Sensor to Actuator in IoT

A diagram for Sensor to actuator flow in IoT is shown. Sensors, actuators, computer servers, and the communication network form the core infrastructure of an IoT Framework. Data collection, handling, communication, and processing of the data are done under IoT technology. The IoT device collects a high amount of information from various sensors, and it is being decided through decision making which data is relevant for their condition and which places it is to be processed or stored, as well as which is desired communication level, while actuators enable the automation of the system for relevant information processing.



SETTING UP THE BOARD : we will learn about the different components on the Arduino board. We will study the Arduino UNO board because it is the most popular board in the Arduino board family. In addition, it is the best board to get started with electronics and coding. Some boards look a bit different from the one given below, but most Arduinos have majority of these components in common.

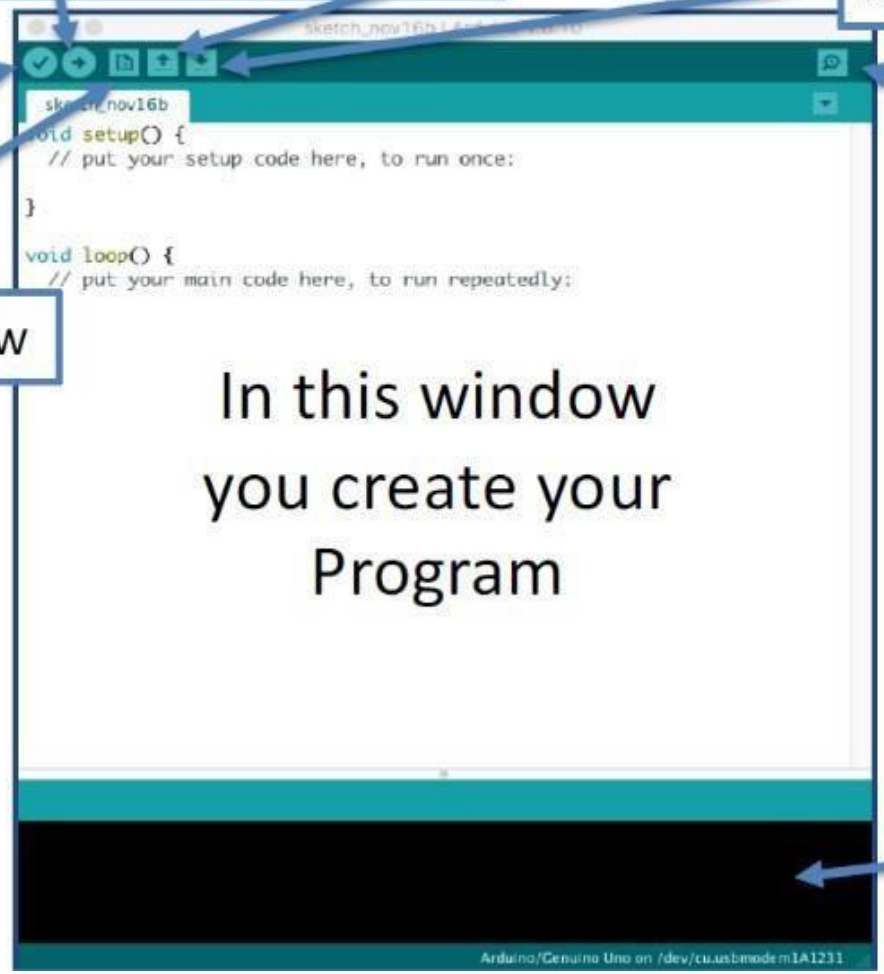


Here are the components that make up an Arduino board and what each of their functions are.

- **Reset Button** – This will restart any code that is loaded to the Arduino board
- **AREF** – Stands for "Analog Reference" and is used to set an external reference voltage
- **Ground Pin** – There are a few ground pins on the Arduino and they all work the same
- **Digital Input/Output** – Pins 0-13 can be used for digital input or output
- **PWM** – The pins marked with the (~) symbol can simulate analog output
- **USB Connection** – Used for powering up your Arduino and uploading sketches
- **TX/RX** – Transmit and receive data indication LEDs
- **ATmega Microcontroller** – This is the brains and is where the programs are stored
- **Power LED Indicator** – This LED lights up anytime the board is plugged in a power source
- **Voltage Regulator** – This controls the amount of voltage going into the Arduino board
- **DC Power Barrel Jack** – This is used for powering your Arduino with a power supply
- **3.3V Pin** – This pin supplies 3.3 volts of power to your projects
- **5V Pin** – This pin supplies 5 volts of power to your projects
- **Ground Pins** – There are a few ground pins on the Arduino and they all work the same
- **Analog Pins** – These pins can read the signal from an analog sensor and convert it to digital

Step 1: First you must have your Arduino board (you can choose your favorite board) and a USB cable.
Step 2: Download Arduino IDE Software. You can get different versions of Arduino IDE from the Download page on the Arduino Official website. You must select your software, which is compatible with your operating system (Windows, IOS, or Linux). After your file download is complete, unzip the file.
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. If you are using an Arduino Diecimila, you have to make sure that the board is configured to draw power from the USB connection. 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.
Step 4: Launch Arduino IDE.
After your 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.
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.

Arduino Software



Upload Code to Arduino Board

Open existing Code

Save

Compile and Check
if Code is OK

Open Serial Monitor

Creates a New Code Window

Error Messages
can be seen here

An be downloaded for free:

www.arduino.cc

Programming for IoT :

Every Arduino sketch has two main parts to the program:

void setup() – Sets things up that have to be done once and then don't happen again.

void loop() – Contains the instructions that get repeated over and over until the board is turned off.

Arduino Programs

All Arduino programs must follow the following main structure:

```
// Initialization, define variables, etc.

void setup()
{
    // Initialization
    ...
}

void loop()
{
    //Main Program
    ...
}
```

```
void setup ( )
{
pinMode (pin-number, OUTPUT); // set the 'pin-number' as output pinMode (pin-
number, INPUT); // set the 'pin-number' as output
}
```

After the setup ( ) function is executed, the execution block runs next. The execution block hosts statements like reading inputs, triggering outputs, checking conditions etc..

In the above example loop ( ) function is a part of execution block. As the name suggests, the loop( ) function executes the set of statements (enclosed in curly braces) repeatedly.

```
Void loop ( )
{
digitalWrite (pin-number,HIGH); // turns ON the component connected to 'pin-number'
delay (1000); // wait for 1 sec
digitalWrite (pin-number, LOW); // turns OFF the component connected to 'pin-number'
delay (1000); //wait for 1sec
}
```

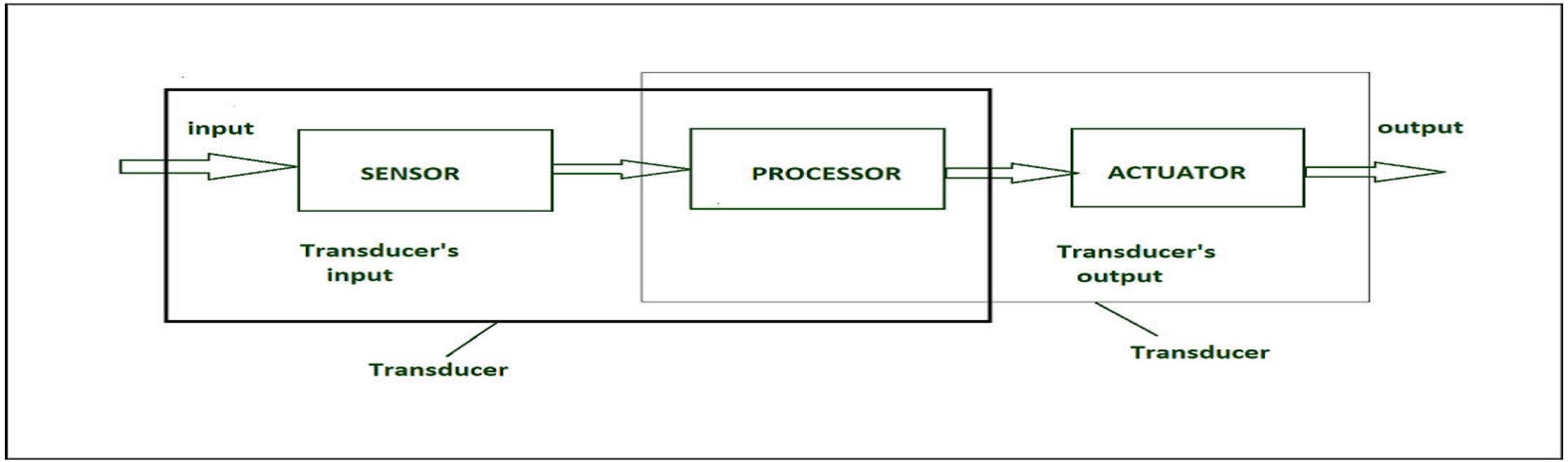
Note: Arduino always measures the time duration in millisecond. Therefore, whenever you mention the delay, keep it in milli seconds. Now, let's take a giant leap and do some experiments with Arduino

- Blinking the LED
- Fade-in and fade-out the LED

Reading from Sensors :

Sensors are used for sensing things and devices etc.

A device that provides a usable output in response to a specified measurement. The sensor attains a physical parameter and converts it into a signal suitable for processing (e.g. electrical, mechanical, optical) the characteristics of any device or material to detect the presence of a particular physical quantity. The output of the sensor is a signal which is converted to a human-readable form like changes in characteristics, changes in resistance, capacitance, impedance, etc.



A transducer converts a signal from one physical structure to another.
It converts one type of energy into another type.
It might be used as actuator in various systems.

Sensors characteristics :

Static

Dynamic

Static characteristics :

It is about how the output of a sensor changes in response to an input change after steady state condition.

Accuracy: Accuracy is the capability of measuring instruments to give a result close to the true value of the measured quantity. It measures errors. It is measured by absolute and relative errors. Express the correctness of the output compared to a higher prior system. Absolute error = Measured value – True value

Relative error = Measured value/True value

Range: Gives the highest and the lowest value of the physical quantity within which the sensor can actually sense. Beyond these values, there is no sense or no kind of response.

e.g. RTD for measurement of temperature has a range of -200`c to 800`c.

Resolution: Resolution is an important specification for selection of sensors. The higher the resolution, better the precision. When the accretion is zero to, it is called the threshold.

Provide the smallest changes in the input that a sensor is able to sense.

Precision: It is the capacity of a measuring instrument to give the same reading when repetitively measuring the same quantity under the same prescribed conditions.

- It implies agreement between successive readings, NOT closeness to the true value.
- It is related to the variance of a set of measurements.
- It is a necessary but not sufficient condition for accuracy.

Sensitivity: Sensitivity indicates the ratio of incremental change in the response of the system with respect to incremental change in input parameters. It can be found from the slope of the output characteristics curve of a sensor. It is the smallest amount of difference in quantity that will change the instrument's reading.

Linearity: The deviation of the sensor value curve from a particularly straight line. Linearity is determined by the calibration curve. The static calibration curve plots the output amplitude versus the input amplitude under static conditions.

A curve's slope resemblance to a straight line describes linearity.

Drift: The difference in the measurement of the sensor from a specific reading when kept at that value for a long period of time.

Repeatability: The deviation between measurements in a sequence under the same conditions. The measurements have to be made under a short enough time duration so as not to allow significant long-term drift.

Dynamic Characteristics :

Properties of the systems

Zero-order system: The output shows a response to the input signal with no delay. It does not include energy-storing elements.

Ex. potentiometer measure, linear and rotary displacements.

First-order system: When the output approaches its final value gradually. Consists of an energy storage and dissipation element.

Second-order system: Complex output response. The output response of the sensor oscillates before steady state.

Communication through Bluetooth, Wi-Fi :

Initial Configuration of the HC05 modules

This is the additional step required to connect two HC05 modules together. We need to change some settings inside the HC05 Bluetooth Module, to do this, we have to go into the HC05 module's **AT Command Mode** and send commands to it through the Arduino IDE's serial monitor. To do this, we need to write an Arduino code to send commands through the serial monitor to the HC05.

The **code to configure the HC05 module** can be found at the bottom of this page, the explanation of the code is as follows

Add the Software Serial library to this code. **#include <SoftwareSerial.h>**

Define the transmit (Tx) and Receive (Rx) pin numbers. I'm using pin 2 for Tx and pin 3 for Rx

#define tx 2

#define rx 3

Give the Bluetooth connection some name (here I am using configBt), then tell the SoftwareSerial library which pin is Tx and which pin is Rx.
The syntax is **bluetoothName(Rx, Tx);**
SoftwareSerial configBt(rx, tx); // RX, TX
In order to configure the Bluetooth module, the Arduino needs to send commands to it at a baud rate of 38400 baud. Similarly, we set the baud rate of the Bluetooth connection as well to 38400 baud. Set the Transmit (Tx) to the output pin and Receive (Rx) to the input pin

```
void setup()
{
Serial.begin(38400);
configBt.begin(38400);
pinMode(tx, OUTPUT);
pinMode(rx, INPUT);
}
```

Inside the forever loop, we have the main chunk of the code. The idea here is to send whatever is typed in the textbox in the serial monitor to the HC05 through the Arduino's Tx pin. Then display whatever is output by the HC05 in the serial monitor.

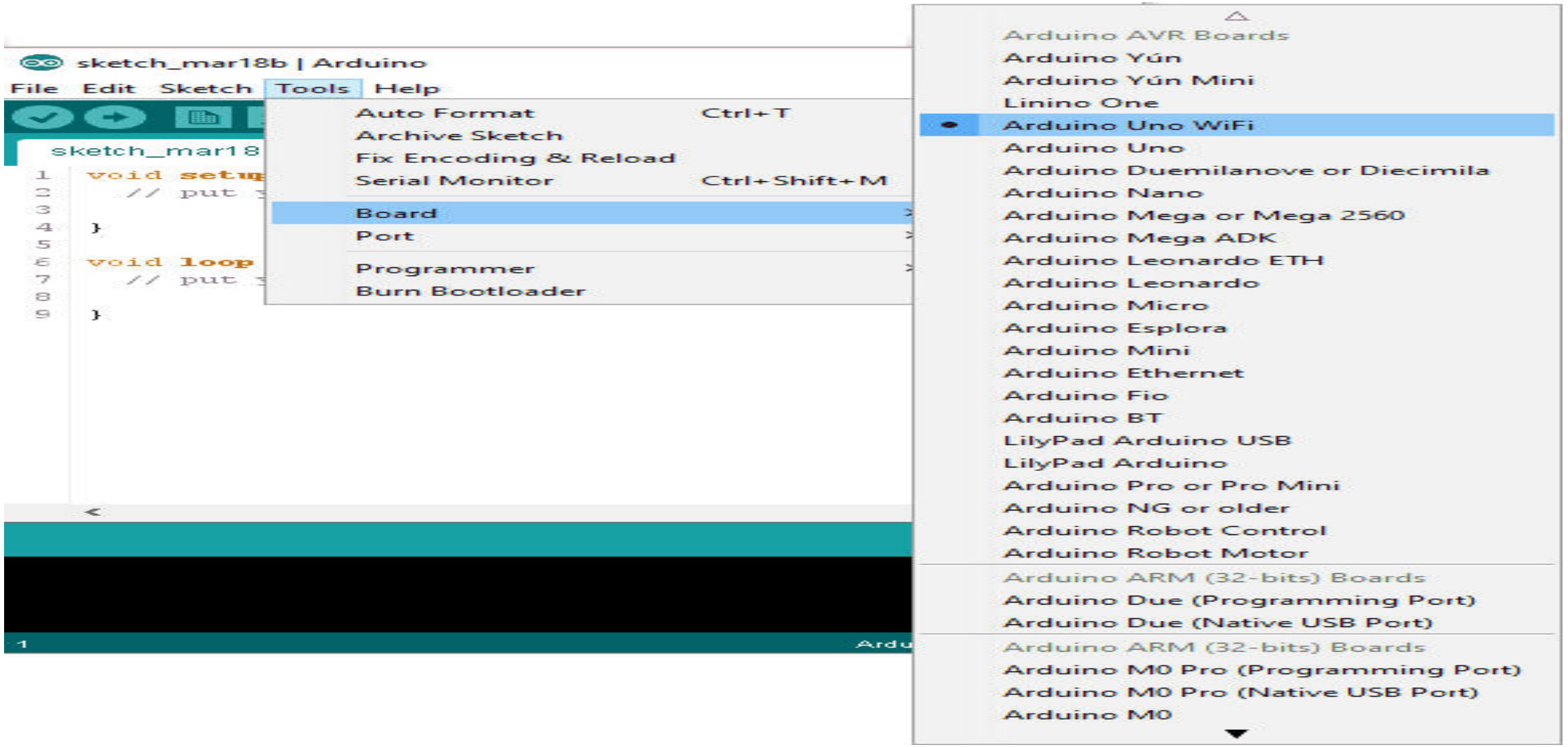
```
void loop()
{
if(configBt.available()) // if the HC05 is sending something…
{
Serial.print(configBt.readString()); // print in serial monitor
}
if(Serial.available()) // if serial monitor is outputting something…
{
configBt.write(Serial.read()); // write to Arduino's Tx pin
}}
```

Upload this code into the Arduino connected to the master HC05 module first. After uploading the code, plug out the Arduino power cable. **Press and hold** the button on the HC05. Now plug in the Arduino power cable while still holding the button on the HC05. Alright, now you can release the button on the HC05. This is how you go into the AT mode of the HC05. To check if you have done this right, make sure the **red light on the HC05 is blinking approximately every one second** (slow blinking!). Normally before the HC05 is connected to any Bluetooth device, it's red light blinks at a very high frequency (fast blinking!).

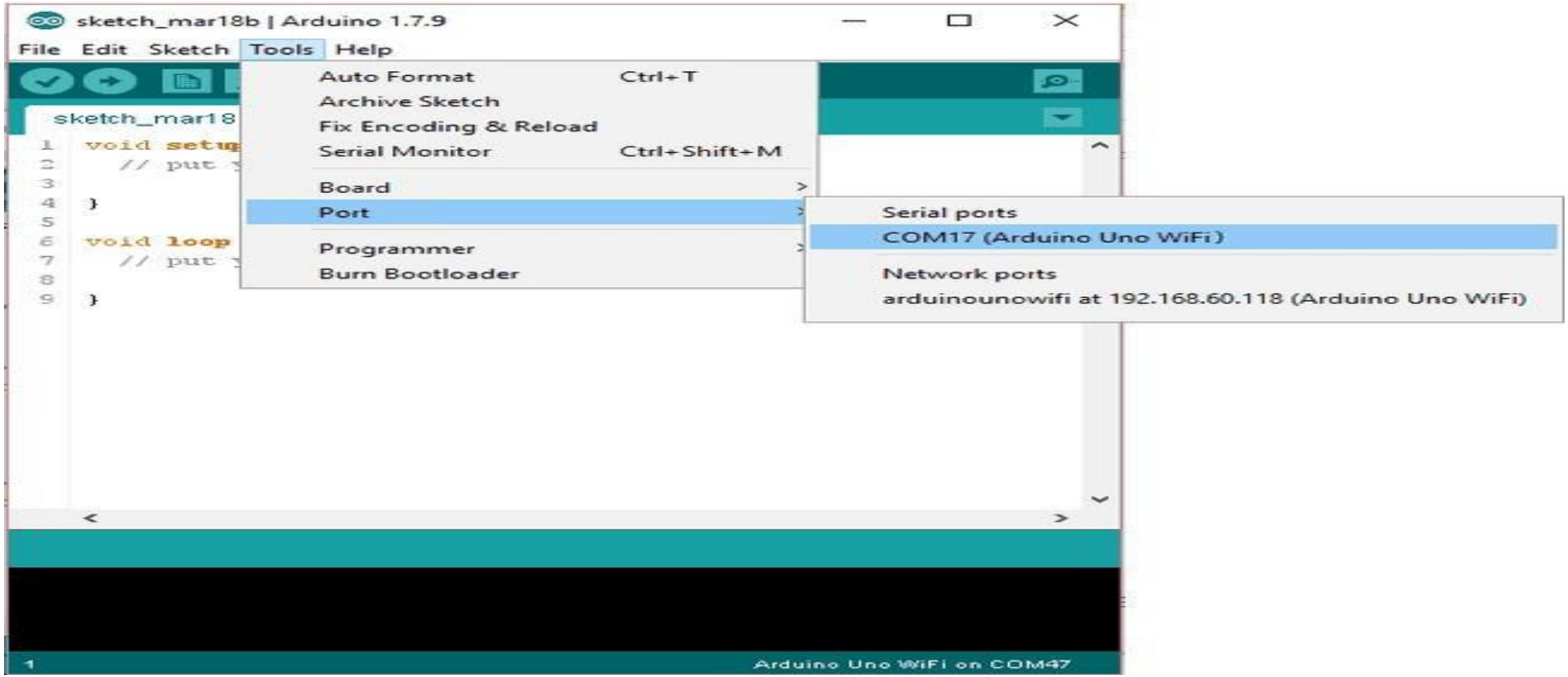
Next, open the serial monitor (the serial monitor button is at the top right of the Arduino IDE). At the bottom right corner of the Serial monitor window, if you haven't already done so, make sure that you set the line ending setting to **"Both NL and CL"** and baud rate to **38400**. Now, type in AT in the serial monitor, if all goes well, you'll get an "OK" from the HC05 displayed in the serial monitor window. Congratulations! You are have successfully logged into the HC05 module's AT command mode.

WIFI: Select your board type and port

You'll need to select the entry in the **Tools > Board** menu that corresponds to your Arduino Uno WiFi board.

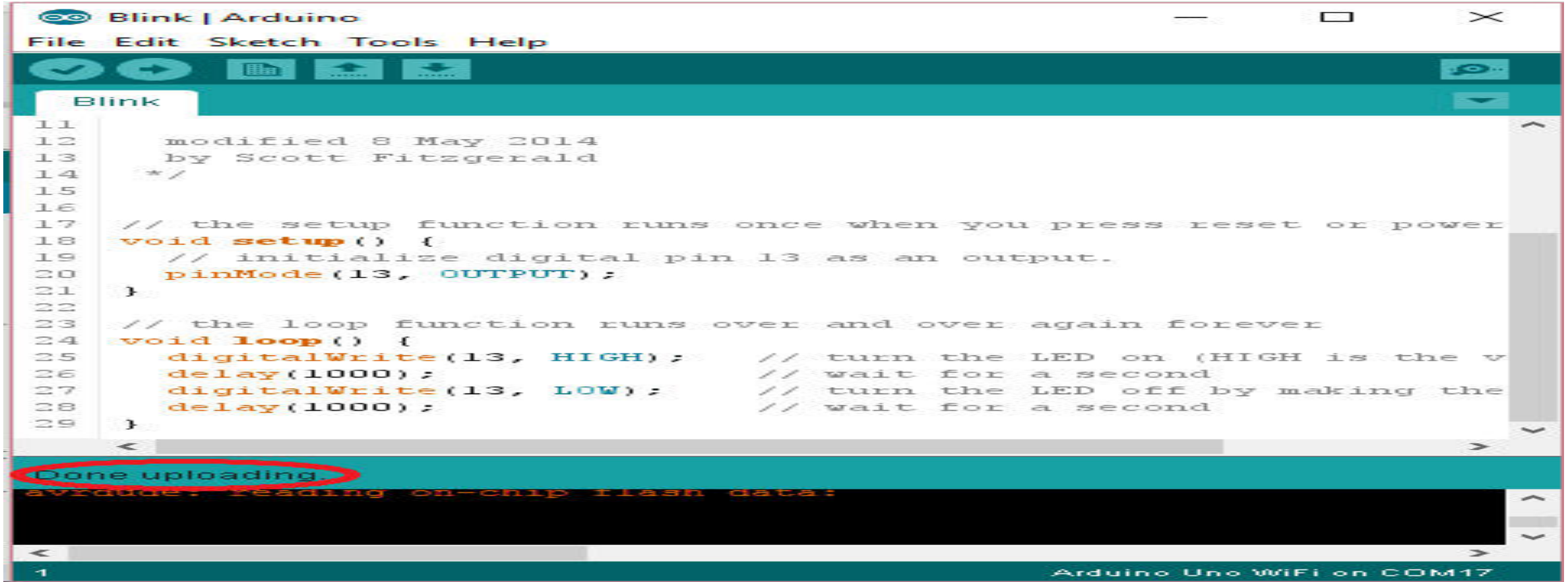


Select the serial device of the board from the Tools | Serial Port menu. This is likely to be **COM3** or higher (**COM1** and **COM2** are usually reserved for hardware serial ports). To find out, you can disconnect your board and re-open the menu; the entry that disappears should be the Arduino Uno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



Upload the program
Now, simply click the "Upload" button in the environment.

Wait a few seconds - you should 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.



A few seconds after the upload finishes, you should see the on-board LED start to blink. If it does, congratulations! You've gotten your Uno WiFi board up-and-running for the USB programming. If you have problems, please see the troubleshooting suggestions.

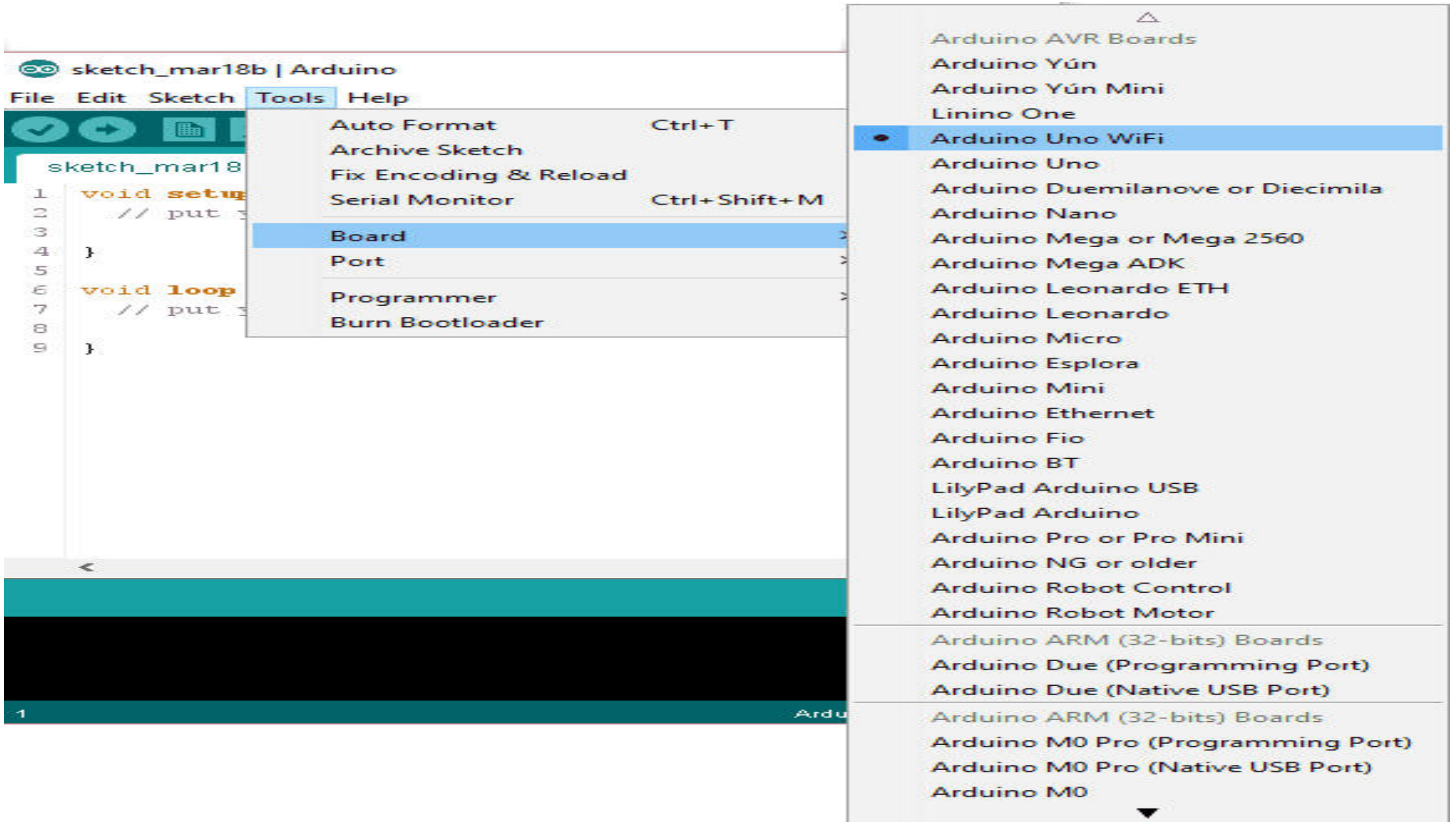
Programming via OTA

This board allows you to upload your sketches over the air (OTA) using the WiFi connection. To get this method working, you need that your board is already connected to the same WiFi network to which your PC is connected. Please refer to the *First Configuration* chapter below to configure and connect the Arduino Uno WiFi to your WiFi network.

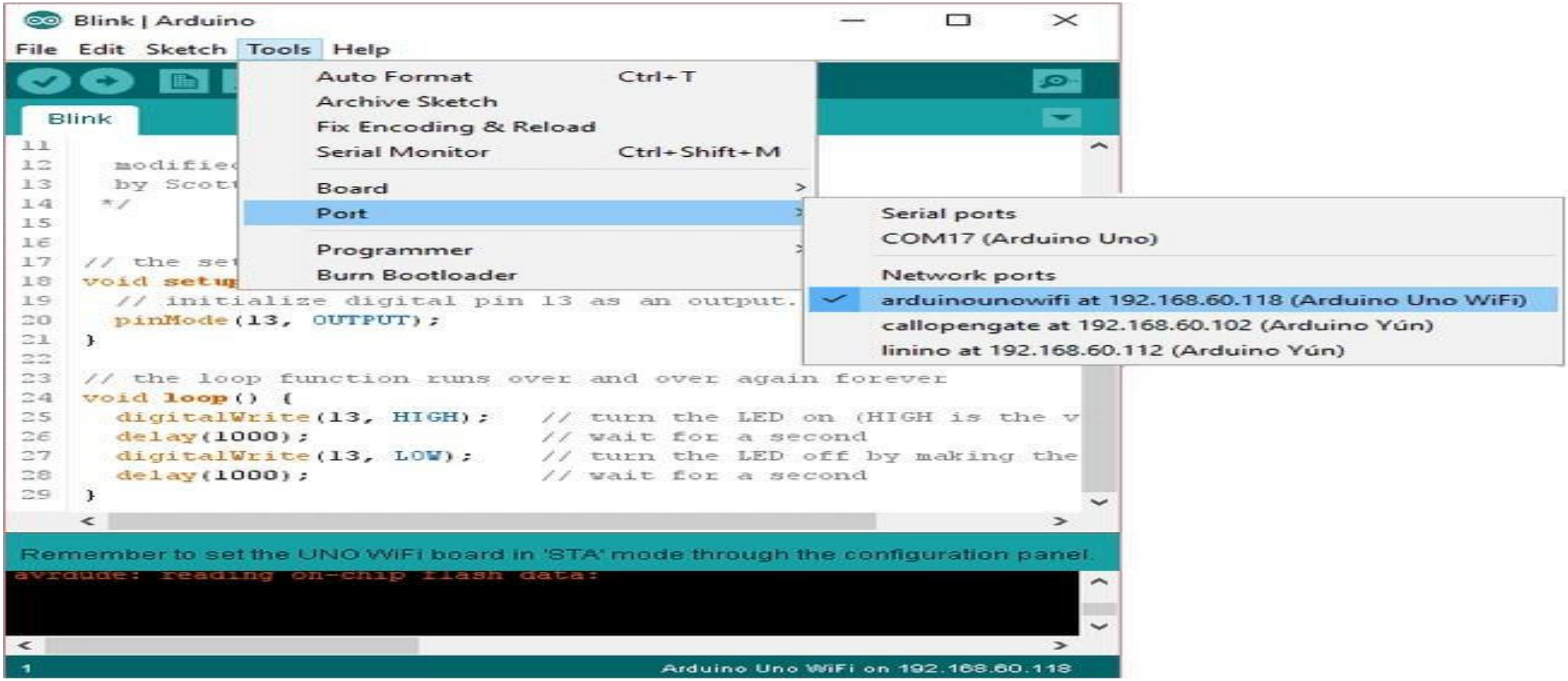
Power the board using the USB cable and a 5V USB power supply or use an external power supply connected to the power connector.. Now the procedure to program the board via OTA is the same of that shown above but it differs only when you select the port. Here are all the steps..

Select your board type and port

You'll need to select the entry in the **Tools > Board** menu that corresponds to your Arduino Uno WiFi board.


sketch_mar18b | Arduino
File Edit Sketch Tools Help
sketch_mar18
1 void setup
2 // put
3
4 }
5
6 void loop
7 // put
8
9 }
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Board
Port
Programmer
Burn Bootloader
Arduino AVR Boards
Arduino Yún
Arduino Yún Mini
Linino One
Arduino Uno WiFi
Arduino Uno
Arduino Duemilanove or Diecimila
Arduino Nano
Arduino Mega or Mega 2560
Arduino Mega ADK
Arduino Leonardo ETH
Arduino Leonardo
Arduino Micro
Arduino Esplora
Arduino Mini
Arduino Ethernet
Arduino Fio
Arduino BT
LilyPad Arduino USB
LilyPad Arduino
Arduino Pro or Pro Mini
Arduino NG or older
Arduino Robot Control
Arduino Robot Motor
Arduino ARM (32-bits) Boards
Arduino Due (Programming Port)
Arduino Due (Native USB Port)
Arduino ARM (32-bits) Boards
Arduino M0 Pro (Programming Port)
Arduino M0 Pro (Native USB Port)
Arduino M0
1

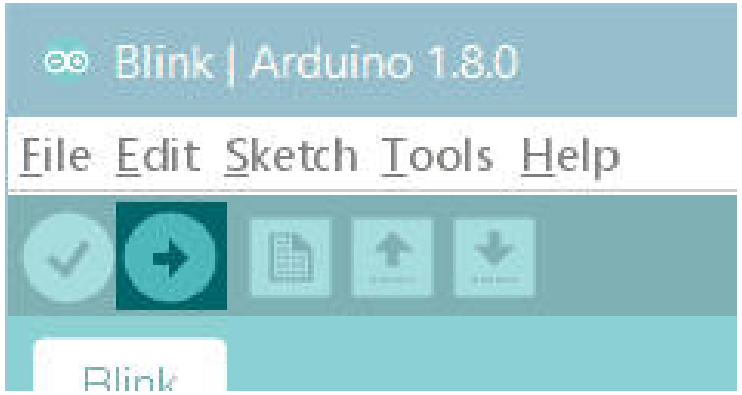
Select the board from **Tool>Port>Network** ports menu it will appear a device as shown in the below image:



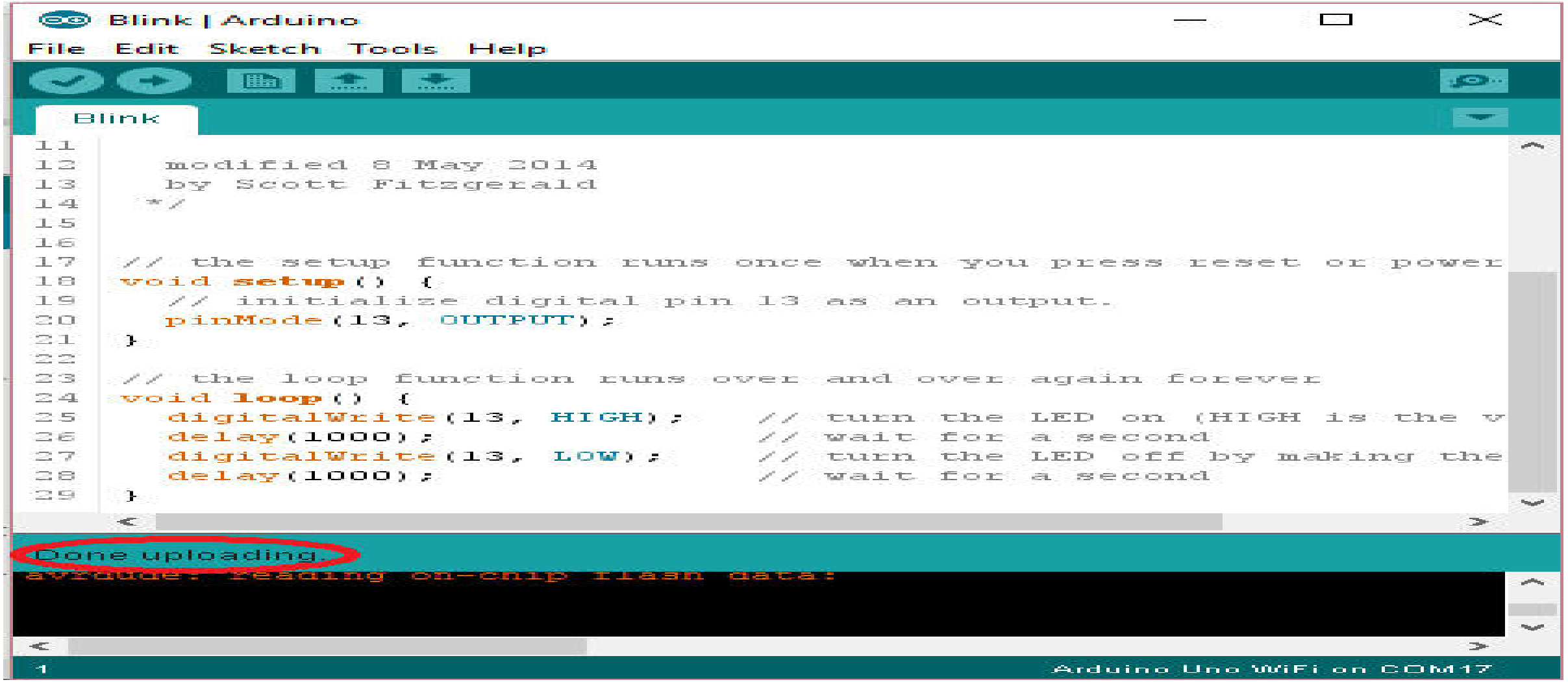
Note: Be sure that the PC and the board are connected to the same network and that the board is in STA MODE, for more information see *First Configuration* below.

Upload the program

Now, simply click the "Upload" button in the environment.



Wait a few seconds - you should 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.



A few seconds after the upload finishes, you should see the on-board LED start to blink. If it does, congratulations! You've gotten your Uno WiFi board up-and-running for the USB programming. If you have problems, please see the troubleshooting suggestions.

UNIT III : IOT ARCHITECTURE and PROTOCOLS
Architecture Reference Model-Introduction, Reference Model and Architecture, IoT reference model - Domain model - Information model - Functional model - Communication model – Protocols- 6LowPAN, RPL, CoAP, MQTT, IoT Frameworks- Thing Speak.

Architecture Reference Model:

Introduction:

The Internet of Things (IoT) has seen an increasing interest in adaptive frameworks and architectural designs to promote the correlation between IoT devices and IoT systems. This is because IoT systems are designed to be categorized across diverse application domains and geographical locations. It, therefore, creates extensive dependencies across domains, platforms and services. Considering this interdependency between IoT devices and IoT systems, an intelligent, connection-aware framework has become a necessity, this is where IoT architecture comes into play.

In essence, an IoT architecture is the system of numerous elements that range from sensors, protocols, actuators, to cloud services, and layers. Besides, devices and sensors the Internet of Things (IoT) architecture layers are distinguished to track the consistency of a system through protocols and gateways. Different architectures have been proposed by researchers and we can all agree that there is no single consensus on architecture for IoT.

State of the art

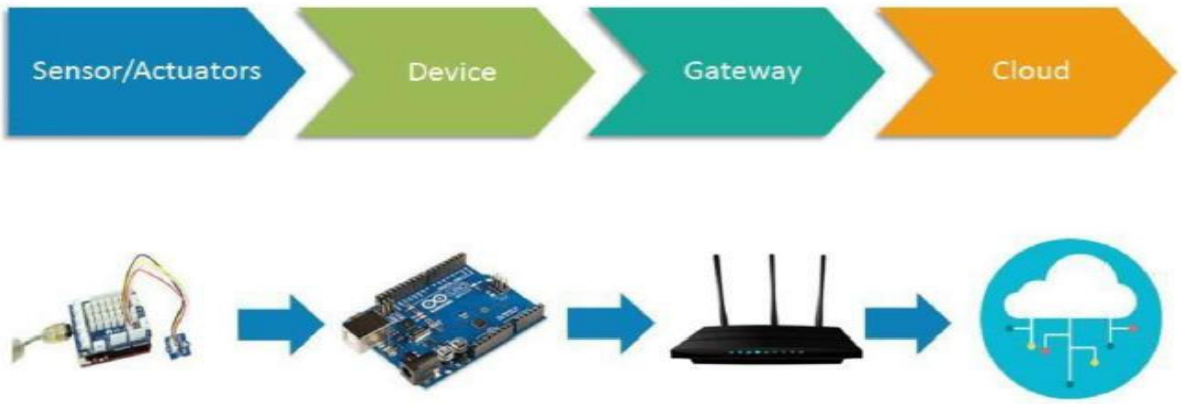
IoT architecture varies from solution to solution, based on the type of solution which we intend to build. IoT as a technology majorly consists of four main components, over which an architecture is framed.

1) Sensors

2) Devices

3) Gateway

4) Cloud



Stages of IoT Architecture:

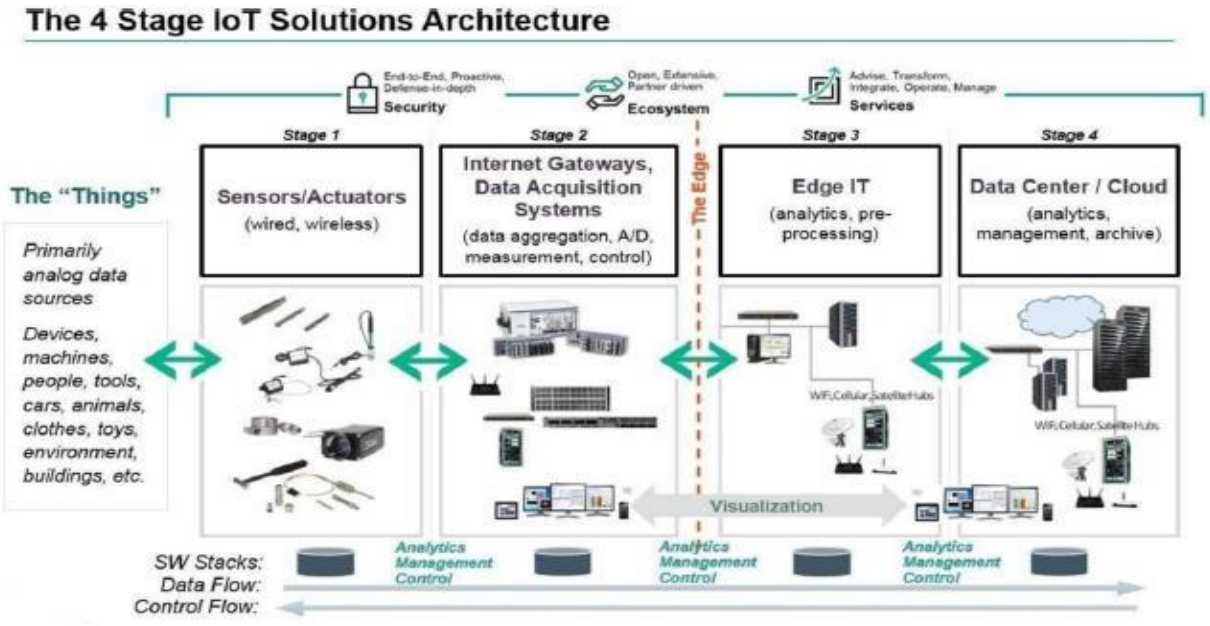


Figure 3.1: Stages of IoT Architecture

1. Sensors/actuators

Sensors collect data from the environment or object under measurement and turn it into useful data. Think of the specialized structures in your cell phone that detect the directional pull of gravity and the phone's relative position to the ―thing¦ we call the earth and convert it into data that your phone can use to orient the device.

Actuators can also intervene to change the physical conditions that generate the data. An actuator might, for example, shut off a power supply, adjust an air flow valve, or move a robotic gripper in an assembly process.

The sensing/actuating stage covers everything from legacy industrial devices to robotic camera systems, water level detectors, air quality sensors, accelerometers, and heart rate monitors. And the scope of the IoT is expanding rapidly, thanks in part to low-power wireless sensor network technologies and Power over Ethernet, which enable devices on a wired LAN to operate without the need for an A/C power source.

2. The Internet gateways

The data from the sensors starts in analog form. That data needs to be aggregated and converted into digital streams for further processing downstream. Data acquisition systems (DAS) perform these data aggregation and conversion functions. The DAS connects to the sensor network, aggregates outputs, and performs the analog-to-digital conversion. The Internet gateway receives the aggregated and digitized data and routes it over Wi-Fi, wired LANs, or the Internet, to Stage 3 systems for further processing. Stage 2 systems often sit in close proximity to the sensors and actuators.

For example, a pump might contain a half-dozen sensors and actuators that feed data into a data aggregation device that also digitizes the data. This device might be physically attached to the pump. An adjacent gateway device or server would then process the data and forward it to the

Stage 3 or Stage 4 systems. Intelligent gateways can build on additional, basic gateway functionality by adding such capabilities as analytics, malware protection, and data management services. These systems enable the analysis of data streams in real time.

3. Edge IT

Once IoT data has been digitized and aggregated, it's ready to cross into the realm of IT. However, the data may require further processing before it enters the data centre. This is where edge IT systems, which perform more analysis, come into play. Edge IT processing systems may be located in remote offices or other edge locations, but generally these sit in the facility or location where the sensors reside closer to the sensors, such as in a wiring closet. Because IoT data can easily eat up network bandwidth and swamp your data centre resources, it's best to have systems at the edge capable of performing analytics as a way to lessen the burden on core IT infrastructure. You'd also face security concerns, storage issues, and delays processing the data. With a staged approach, you can pre-process the data, generate meaningful results, and pass only those on. For example, rather than passing on raw vibration data for the pumps, you could aggregate and convert the data, analyse it, and send only projections as to when each device will fail or need service.

4. The data centre and cloud

Data that needs more in-depth processing, and where feedback doesn't have to be immediate, gets forwarded to physical data centre or cloud-based systems, where more powerful IT systems can analyse, manage, and securely store the data. It takes longer to get results when you wait until data reaches Stage 4, but you can execute a more in-depth analysis, as well as combine your sensor data with data from other sources for deeper insights. Stage 4 processing may take place on-premises, in the cloud, or in a hybrid cloud system, but the type of processing executed in this stage remains the same, regardless of the platform.

IoT Reference architecture

- Architecture Reference Model (ARM) consists of two main parts:
  1. a Reference model
  2. a Reference Architecture.
- The foundation of an IoT Reference Architecture description is an IoT reference model.
- A System Architecture is a communication tool for different stakeholders of the system.
- Developers, component and system managers, partners, suppliers, and customers have different views of a single system based on their requirements and their specific interactions with the system.
- The high-level abstraction is called Reference Architecture as it serves as a reference for generating concrete architectures and actual systems, as shown in the Figure 3.2.

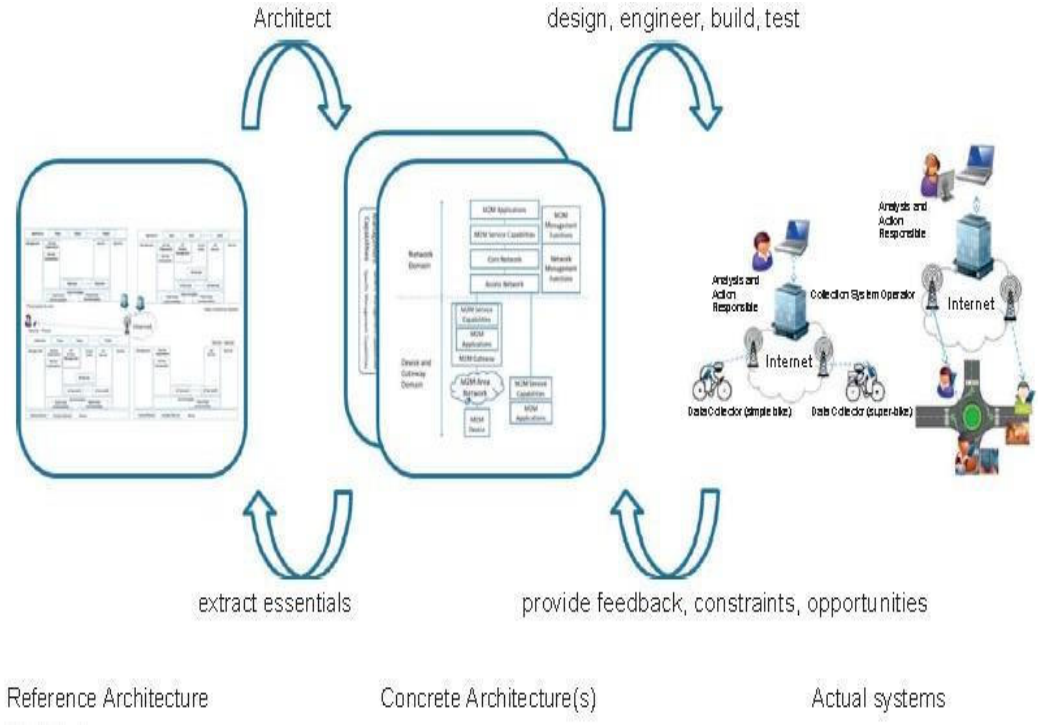


Figure 3.2: Reference to Concrete Architectures and Actual Systems

- Concrete architectures are instantiations of rather abstract and high-level Reference Architectures.
- A Reference Architecture captures the essential parts of an architecture, such as design principles, guidelines, and required parts (such as entities), to monitor and interact with the physical world for the case of an IoT Reference Architecture.
- A concrete architecture can be further elaborated and mapped into real world components by designing, building, engineering, and testing the different components of the actual system.
- The general essentials out of multiple concrete architectures can then are aggregated, and contribute to the evolution of the Reference Architecture.

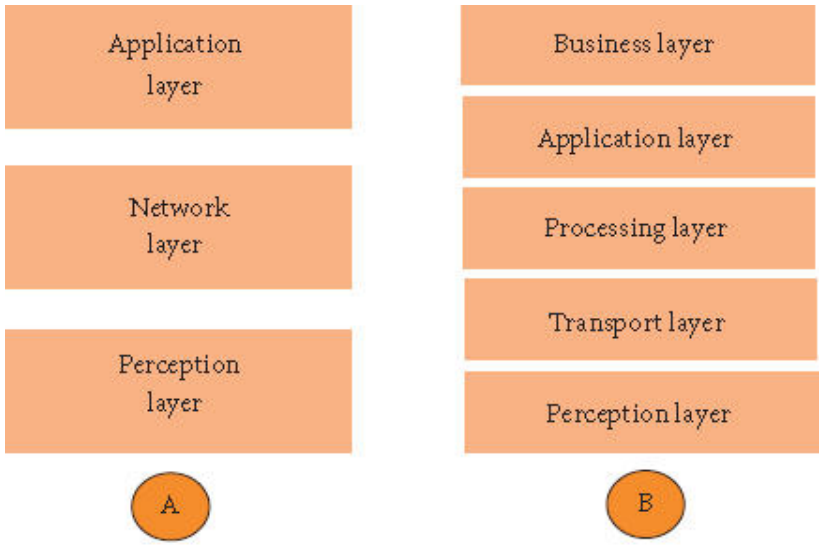


Figure 3.3: Architecture of IoT a). Three layers b). Five layers

- It has two types of Architecture:
  Three Layer Architectures
  Five-Layer Architectures

Three Layer Architectures

- It has three layers, namely, the perception, network, and application layers.

  (i) The *perception layer* is the physical layer, which has sensors for sensing and gathering information about the environment. It senses some physical parameters or identifies other smart objects in the environment.
  (ii) The *network layer* is responsible for connecting to other smart things, network devices, and servers. Its features are also used for transmitting and processing sensor data.
  (iii) The *application layer* is responsible for delivering application specific services to the user. It defines various applications in which the Internet of Things can be deployed, for example, smart homes, smart cities, and smart health.

- The three-layer architecture defines the main idea of the Internet of Things, but it is not sufficient for research on IoT because research often focuses on finer aspects of the Internet of Things.

Five Layer Architectures

- The five layers are perception, transport, processing, application, and business layers (seeThe role of the perception and application layers is the same as the architecture with threelayers. We outline the function of the remaining three layers.

(i) The *transport layer* transfers the sensor data from the perception layer to the processing layer and vice versa through networks such as wireless, 3G, LAN, Bluetooth, RFID, and NFC.

(ii) The *processing layer* is also known as the middleware layer. It stores, analyzes, and processes huge amounts of data that comes from the transport layer. It can manage and provide a diverse set of services to the lower layers. It employs many technologies such as databases, cloud computing, and big data processing modules.
(iii) The *business layer* manages the whole IoT system, including applications, business and profit models, and users' privacy.

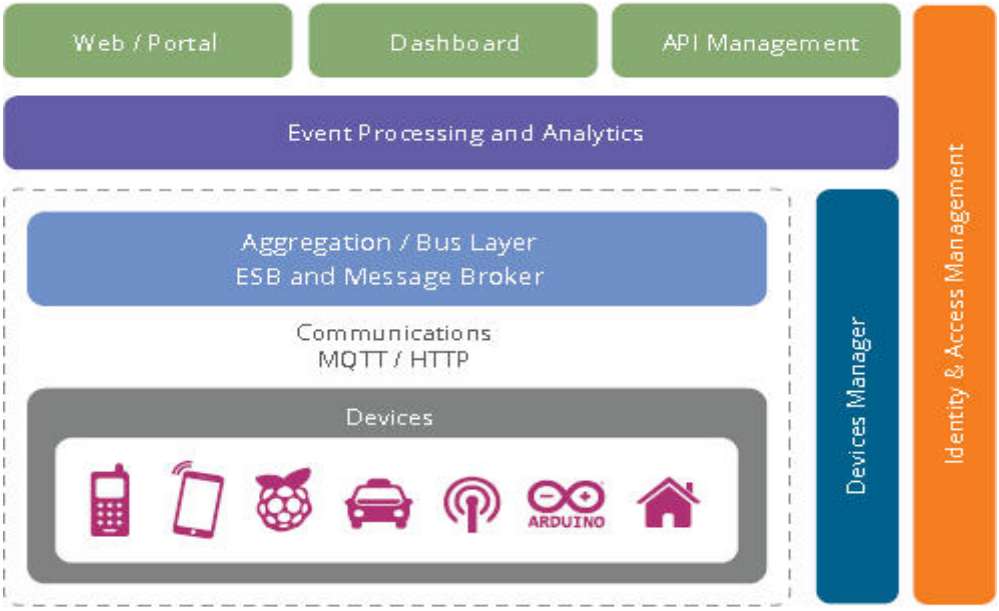


Figure 3.4: Reference Architecture for IoT

- 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 be considered 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 or Wi-Fi Examples of indirectly connected device 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 a Raspberry Pi

- Each device typically needs an identity.
- The identity may be one of the following:
  • A unique identifier (UUID) burnt into the device
  • A UUID provided by the radio subsystem (e.g. Bluetooth identifier, Wi-Fi MAC address)
  • An OAuth2 Refresh/Bearer Token
  • An identifier stored in nonvolatile memory such as EEPROM

The Communications Layer

• The communication layer supports the connectivity of the devices.
• There are multiple potential protocols for communication 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

• Constrained application protocol (CoAP)

- HTTP supports many libraries. Because it is a simple textbased protocol, many small devices such as 8-bit controllers can only partially support the .
- The larger 32-bit based devices can utilize full HTTP client libraries that properly implement the whole protocol.
- MQTT solve issues in embedded systems and SCADA.
- MQTT is a publish-subscribe messaging system based on a broker model. The protocol has a very small overhead.
- It is designed to support lossy and intermittently connected networks.
- MQTT was designed to flow over TCP.
- In addition there is an associated specification designed for ZigBee-style networks called MQTT-SN (Sensor Networks).
- CoAP is a protocol from the IETF that is designed to provide a RESTful application protocol modeled on HTTP semantics. CoAP is a more traditional client-server approach
- CoAP is designed to be used over UDP.

The Aggregation/Bus Layer

- This layer 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 that are mediated into an MQTT message going to the device.

- The bus layer may also provide some simple correlation and mapping from different correlation models (e.g. mapping a device ID into an owner's ID or vice-versa).
- 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 policy enforcement point (PEP) for policy-based access.

The 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 core capability here is the requirement to store the data into a database.
- It has 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

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 front-ends and portals that interact with devices and with the event-processing layer.
  - Secondly, we need the ability to create dashboards that offer views into analytics and event processing.
  - Finally, we need to be able to interact with systems outside this network using machine-to-machine communications (APIs).
- The API management layer provides three main functions:
  • The first is that it provides a developer-focused portal where developers can find, explore, and subscribe to APIs from the system. There is also 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 (for external requests) as well as throttling usage based on policies. It also performs routing and load-balancing;
  • The final aspect is that the gateway publishes data into the analytics layer where it is stored as well as processed to provide insights into how the APIs are used.

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.
- It also remotely manages software and applications deployed on the device.
- It can lock and/or wipe the device if necessary.
- The device manager works in conjunction with the device management agents.
- There are multiple different 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 to manage access controls over devices.
- There are three levels of device: non-managed, semi-managed and fully managed (NM, SM, FM).
- 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's location if available

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 validationOther identity services including SAML2 SSO and OpenID Connect support for

  identifying inbound requests from the Web layer

  - XACML PDP

  - Directory of users (e.g. LDAP)

  - Policy management for access control (policy control point)

IOT Reference Model

- The IoT Reference Model aims at establishing a common grounding and a common language for IoT architectures and IoT systems.
- A reference model describes the domain using a number of sub-models (Figure 7.1).
- The domain model of an architecture model captures the main concepts or entities in the domain, the domain model adds descriptions about the relationship between the concepts.
- These concepts and relationships serve the basis for the development of an information model because a working system needs to capture and process information about its main entities and their interactions.
- A working system that captures and operates on the domain and information model contains concepts and entities of its own, and these need to be described in a separate model, the functional model.
- An M2M and IoT system contain communicating entities, and therefore the corresponding communication model needs to capture the communication interactions of these entities.

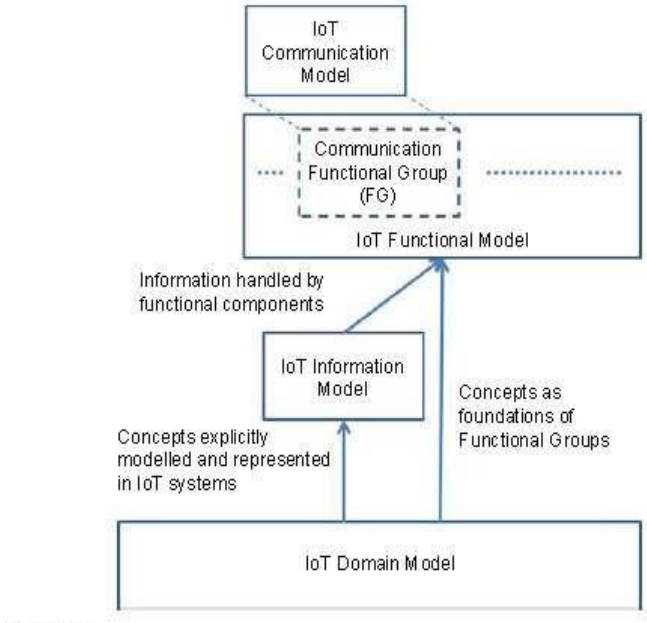


Figure 3.5: IoT Reference Model

- The foundation of the IoT Reference Model is the IoT Domain Model, which introduces the main concepts of the Internet of Things like Devices, IoT Services and *Virtual Entities* (VE), and it also introduces relations between these concepts.
- Based on the IoT Domain Model, the IoT Information Model has been developed. It defines the structure (e.g. relations, attributes) of IoT related information in an IoT system on a conceptual level without discussing how it would be represented.

- The information pertaining to those concepts of the IoT Domain Model is modelled, which is explicitly gathered, stored and processed in an IoT system, e.g. information about Devices, IoT Services and Virtual Entities.
- The IoT Functional Model identifies groups of functionalities, of which most are grounded in key concepts of the IoT Domain Model.
- A number of these *Functionality Groups* (FG) build on each other, following the relations identified in the IoT Domain Model.
- The Functionality Groups provide the functionalities for interacting with the instances of these concepts or managing the information related to the concepts, e.g. information about Virtual Entities or descriptions of IoT Services.

- The functionalities of the FGs that manage information use the IoT Information Model as the basis for structuring their information.
- A key functionality in any distributed computer system is the communication between the different components.
- The IoT Communication Model introduces concepts for handling the complexity of communication in heterogeneous IoT environments. Communication also constitutes one FG in the IoT Functional Model.

IoT domain model

- Domain model as a description of concepts belonging to a particular area of interest.
- The domain model also defines basic attributes of these concepts, such as name and identifier.
- The domain model defines relationships between concepts, for instance <*Services* expose *Resources*=.
- Domain models also help to facilitate the exchange of data between domains.
- The main purpose of a domain model is to generate a common understanding of the target domain in question.
- The domain model is an important part of any reference model since it includes a definition of the main abstract concepts (abstractions), their responsibilities, and their relationships.
- The domain model captures the basic attributes of the main concepts and the relationship between these concepts.

Model notation and semantics

- Class diagrams in order to present the relationships between the main concepts of the IoT domain model.
- The Class diagrams consist of boxes that represent the different classes of the model connected with each other through typically continuous lines or arrows, which represent relationships between the respective classes.
- Each class is a descriptor of a set of objects that have similar structure, behavior, and relationships.
- A class contains a name and a set of attributes and operations.
- Notation-wise this is represented as a box with two compartments, one containing the class name and the other containing the attributes.
- The Generalization/Specialization relationship is represented by an arrow with a solid line and a hollow triangle head.
- Depending on the starting point of the arrow, the relationship can be viewed as a generalization or specialization.For example, in Figure 7.4, Class A is a general case of Class B or Class B is special caseor specialization of Class A.
- Generalization is also called an <is-a= relationship. For example, in Figure 7.4 Class B

<is-a= Class A. A specialized class/subclass/child class inherits the attributes and the operations from the general/super/parent class, respectively, and also contains its own attributes and operations.

- The Aggregation relationship is represented by a line with a hollow diamond in one end and represents a whole-part relationship or a containment relationship and is often called a <has-a= relationship.

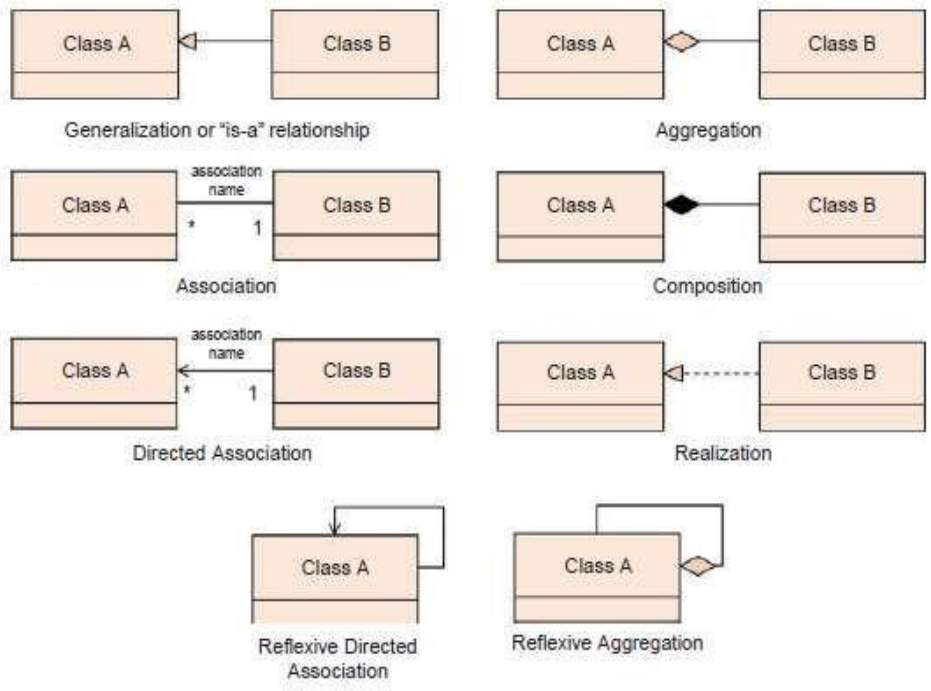


Figure 3.6: UML Class Diagram main modeling concepts

- The class that touches the hollow diamond is the whole class while the other class is the part class.
- For example, in Figure 3.6, class B represents a part of the whole Class A, or in other words, an object of Class A <contains= or <has-a= object of Class B.
- When the line with the hollow diamond starts and ends in the same class, then this relationship of one class to itself is called Reflexive Aggregation, and it denotes that objects of a class (e.g. Class A in Figure 3.6) contain objects of the same class.
- The Composition relationship is represented by a line with a solid black diamond in one end, and also represents a whole-part relationship or a containment relationship.
- The class that touches the solid black diamond is the whole class while the other class is the part class. For example, in Figure 3.6, Class B is part of Class A. Composition and Aggregation are very similar, with the difference being the coincident lifetime to the objects of classes related with composition.
- In other words, if an object of Class B is part of an object of Class A (composition), when the object of Class A disappears, the object of Class B also disappears.
- A plain line without arrowheads or diamonds represents the Association relationship.
- Directed Association that is represented with a line with a normal arrowhead.
- An Association (Directed or not) contains an explicit association name. The Directed Association implies navigability from a Class B to a Class A in Figure 3.6.
- Navigability means that objects of Class B have the necessary attributes to know that they relate to objects of Class A while the reverse is not true: objects of Class A can exist

without having references to objects of Class B.

- When the arrow starts and ends at the same class, then the class is associated to itself with a Reflexive Directed Association, which means that an object of this class is associated with objects of the same class with the specific named association.

- An arrow with a hollow triangle head and a dashed line represents the Realization relationship. This relationship represents a association between the class that specifies the functionality and the class that realizes the functionality.
- For example, Class A in Figure 3.6 specifies the functionality while Class B realizes it. Contain multiplicity information such as numbers (e.g <1=), ranges (e.g. <0_1=, open ranges <1. . ._=), etc. in one or the other end of the relationship line/arrow.
- These multiplicities denote the potential number of class objects that are related to the other class object.
- For example, in Figure 3.6, a plain association called <association name,= relates one (1) object of Class B with zero (0) or more objects from Class A. An asterisk <_= denotes zero (0) or more

Main concepts

- The IoT is a support infrastructure for enabling objects and places in the physical world to have a corresponding representation in the digital world.

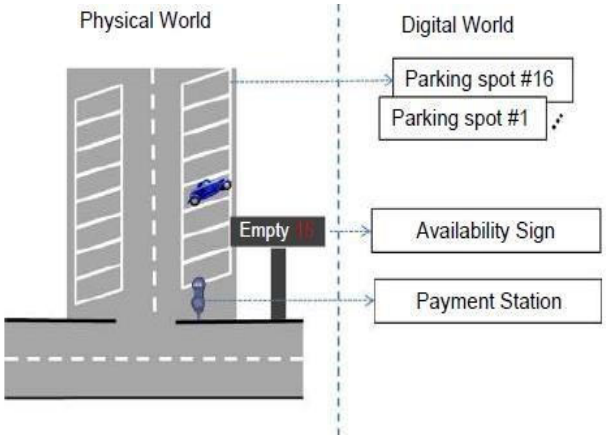


Figure 3.7: Physical Vs Virtual World

- Monitoring a parking lot with 16 parking spots.
- The parking lot includes a payment station for drivers to pay for the parking spot after they park their cars.The parking lot also includes an electronic road sign on the side of the street that shows in real-time the number of empty spots.
- Frequent customers also download a smart phone application that informs them about the availability of a parking spot before they even drive on the street where the parking lot is located.
- The relevant physical objects as well as their properties need to be captured and translated to digital objects such as variables, counters, or database objects so that software can operate on these objects and achieve the desired effect.
- In the digital world, a parking spot is a variable with a binary value (<available= or <occupied=).
- The parking lot payment station also needs to be represented in the digital world in order to check if a recently parked car owner actually paid the parking fee.
- Internet serves a rather virtual world of content and services (although these services are hosted on real physical machines)
- IoT is all about interaction through the Internet with physical Things.

- As interaction with the physical world is the key for the IoT; it needs to be captured in the domain model.
- A User and a Physical Entity are two concepts that belong to the domain model.
- A User can be a Human User, and the interaction can be physical (e.g. parking the car in the parking lot). The physical interaction is the result of the intention of the human to achieve a certain goal (e.g. park the car).
- The objects, places, and things represented as Physical Entities are the same as Assets.
- A Physical Entity is represented in the digital world as a Virtual Entity.
- A Virtual Entity can be a database entry, a geographical model (mainly for places), an image or avatar, or any other Digital Artifact.
- Each Virtual Entity also has a unique identifier for making it addressable among other Digital Artifacts.
- A Virtual Entity representation contains several attributes that correspond to the Physical Entity current state (e.g. the parking spot availability).
- The Virtual Entity representation and the Physical Entity actual state should be synchronized.
- For example, a remotely controlled light (Physical Entity) represented by a memory location (Virtual Entity) in an application could be switched on/off by the User by changing the Virtual Entity representation, or in other words writing a value in the corresponding memory location.
- A Digital Artifact is an artifact of the digital world, and can be passive (e.g. a database entry) or active (e.g. application software).
- The model captures human-to-machine, application (active digital artifact)-to-machine, and M2M interaction when a digital artifact, and thus a User, interacts with a Device that is a Physical Entity.
- Physical Entities or their surrounding environment needs to be instrumented with certain kinds of Devices, or certain Devices need to be embedded/attached to the environment.
- IoT Domain Model, three kinds of Device types are the most important:

1. **Sensors:**
   - These are simple or complex Devices that typically involve a transducer that converts physical properties such as temperature into electrical signals.
   - These Devices include the necessary conversion of analog electrical signals into digital signals, e.g. A video camera can be example of a complex sensor that could detect and recognize people.
2. **Actuators:**
   - These are also simple or complex Devices that involve a transducer that converts electrical signals to a change in a physical property (e.g. turn on a switch or move a motor).
   - These Devices also include potential communication capabilities, storage of intermediate commands, processing, and conversion of digital signals to analog electrical signals.
3. **Tags:**
   - Tags in general identify the Physical Entity that they are attached to.
   - In reality, tags can be Devices or Physical Entities but not both, as the domain model shows.

- An example of a Tag as a Device is a Radio Frequency Identification (RFID) tag, while a tag as a Physical Entity is a paper-printed immutable barcode or Quick Response (QR) code.

- Either electronic Devices or a paper-printed entity tag contains a unique identification that can be read by optical means (bar codes or QR codes) or radio signals (RFID tags).
- The reader Device operating on a tag is typically a sensor, and sometimes a sensor and an actuator combined in the case of writable RFID tags.

- Any type of IoT Device needs to
  (a) have energy reserves (e.g. a battery)
  (b) be connected to the power grid
  (c) perform energy scavenging (e.g. converting solar radiation to energy).
- The Device communication, processing and storage, and energy reserve capabilities determine several design decisions such as if the resources should be on-Device or not.
- Resources are software components that provide data for, or are endpoints for, controlling Physical Entities.
- Resources can be of two types, on-Device resources and Network Resources.
- An on-Device Resource is typically hosted on the Device itself and provides information, or is the control point for the Physical Entities that the Device itself is attached to.
- An example is a temperature sensor on a temperature node deployed in a room that hosts a software component that responds to queries about the temperature of the room.
- The Network Resources are software components hosted somewhere in the network or cloud.A Virtual Entity is associated with potentially several Resources that provide informationor control of the Physical Entity represented by this Virtual Entity.
- Resources can be of several types: sensor resources that provide sensor data, actuator resources that provide actuation capabilities or actuator state (e.g. <on=/<off=), processing resources that get sensor data as input and provide processed data as output, storage resources that store data related to Physical Entities, and tag resources that provide identification data of Physical Entities.
- IoT Services can be classified into three main classes according to their level of abstraction:

  1. Resource-Level Services typically expose the functionality of a Device by exposing the on-Device Resources. In addition, these services typically handle quality aspects such as security, availability, and performance issues. An example of such a Network Resource is a historical database of measurements of a specific resource on a specific Device.

  2. Virtual Entity-Level Services provide information or interaction capabilities about Virtual Entities, and as a result the Service interfaces typically include an identity of the Virtual Entity.

  3. Integrated Services are the compositions of Resource-Level and Virtual Entity-Level services, or any combination of both service classes.

Further considerations

- Identification of Physical Entities is important in an IoT system in order for any User to interact with the physical world though the digital world.
- Two ways to describe :
  (a) primary identification that uses natural features of a Physical Entity, and
  (b) secondary identification when using tags or labels attached to the Physical Entity.
- Both types of identification are modeled in the IoT Domain Model.
- Extracting natural features can be performed by a camera Device (Sensor) and relevant Resources that produce a set of features for specific Physical Entities.
- In physical spaces, a GPS Device or another type of location Device (e.g. an indoor location Device) can also be used to record the GPS coordinates of the space occupied by the Physical Entity.
- With respect to secondary identification, tags or labels attached to Physical Entities are modeled in the IoT Domain model, and there are relevant RFID or barcode technologies to realize such identification mechanisms.
- Apart from identification, location and time information are important for the annotation of the information collected for specific Physical Entities and represented in Virtual Entities.
- Information without one or the other (i.e. location or time) is practically useless apart from the case of Body Area Networks (BAN, networks of sensors attached to a human body for live capture of vital signals, e.g. heart rate); that location is basically fixed and associated with the identification of the Human User.
- Nevertheless, in such cases, sometimes the location of the whole BAN or Human User is important for correlation purposes (e.g. upon moving outdoors, the Human User heart rate increases in order to compensate for the lower temperature than indoors).
- Therefore, the location, and often the timestamp of location, for the Virtual Entity can be modeled as an attribute of the Virtual Entity that could be obtained by location sensing resources (e.g. GPS or indoor location systems).

Communication model

- The communication model for an IoT Reference Model consists of the identification of the endpoints of interactions, traffic patterns (e.g. unicast vs. multicast), and general properties of the underlying technologies used for enabling such interactions.
- It is used to identification of the endpoints of the communication paths.
- The potential communicating endpoints or entities are the Users, Resources, and Devices from the IoT Domain Model.
- Users include Human Users and Active Digital Artifacts (Services, internal system components, external applications).
- Devices with a Human_Machine Interface mediate the interactions between a Human User and the physical world (e.g. keyboards, mice, pens, touch screens, buttons, microphones, cameras, eye tracking, and brain wave interfaces, etc.), and therefore the Human User is not a communication model endpoint.
- The User (Active Digital Artifact, Service)-to-Service interactions include the User-to-Service and Service-to-Service interactions as well as the Service_Resource_Device

interactions.

- The User-to-Service and Service-to-Service communication is typically based on Internet protocols and one or both Services are hosted in Service-to-Service interactions on constrained/low-end Devices such as embedded systems.
- The communication model for these interactions includes several types of gateways (e.g. network, application layer gateways) to bridge between two or more disparate communication technologies.
- The Devices may be so constrained that they cannot host the Services, while the Resources could be hosted or not depending on the Device capabilities.
- This inability of the Device to host Resources or Services results in moving the corresponding Resources and/or Services out of the Device and into more powerful Devices or machines in the cloud.
- Then the Resource-to-Device or the Service-to-Resource communication needs to involve multiple types of communication stacks.

Functional model

- The IoT Functional Model aims at describing mainly the Functional Groups (FG) and their interaction with the ARM.
- Functional View of a Reference Architecture describes the functional components of an FG, interfaces, and interactions between the components.
- The Functional View is typically derived from the Functional Model in conjunction with high level requirements.

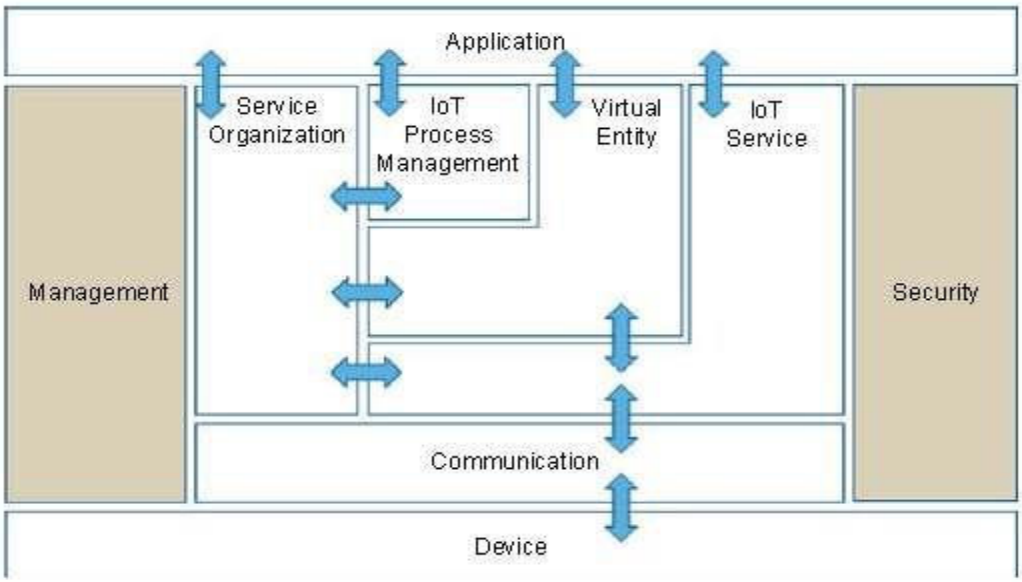


Figure 3.8: IoT-A Functional Model

- The Application, Virtual Entity, IoT Service, and Device FGs are generated by starting from the User, Virtual Entity, Resource, Service, and Device classes from the IoT Domain Model.
- The need to compose simple IoT services in order to create more complex ones, as well

as the need to integrate IoT services (simple or complex) with existing Information and Communications Technology (ICT) infrastructure, is the main driver behind the introduction of the Service Organization and IoT Process Management FGs respectively.

- All the above-mentioned FGs need to be supported by management and security functionality captured by the corresponding FGs.

Device functional group

- The Device FG contains all the possible functionality hosted by the physical Devices that are used for instrumenting the Physical Entities.
- This Device functionality includes sensing, actuation, processing, storage, and identification components, the sophistication of which depends on the Device capabilities.

Communication functional group

- The Communication FG abstracts all the possible communication mechanisms used by the relevant Devices in an actual system in order to transfer information to the digital world components or other Devices.
- Examples of such functions include wired bus or wireless mesh technologies through which sensor Devices are connected to Internet Gateway Devices.
- Communication technologies used between Applications and other functions such as functions from the IoT Service FG are out of scope because they are the typical Internet technologies.

IoT Service functional group

- The IoT Service FG corresponds mainly to the Service class from the IoT Domain Model, and contains single IoT Services exposed by Resources hosted on Devices or in the Network (e.g. processing or storage Resources).
- Support functions such as directory services, which allow discovery of Services and resolution to Resources, are also part of this FG.

Virtual Entity functional group

- The Virtual Entity FG corresponds to the Virtual Entity class in the IoT Domain Model, and contains the necessary functionality to manage associations between Virtual Entities with themselves as well as associations between Virtual Entities and related IoT Services, i.e. the Association objects for the IoT Information Model.
- Associations between Virtual Entities can be static or dynamic depending on the mobility of the Physical Entities related to the corresponding Virtual Entities.
- An example of a static association between Virtual Entities is the hierarchical inclusion relationship of a building, floor, room/corridor/open space, i.e. a building contains multiple floors that contain rooms, corridors, and open spaces.
- An example of a dynamic association between Virtual Entities is a car moving from one block of a city to another (the car is one Virtual Entity while the city block is another).
- A major difference between IoT Services and Virtual Entity Services is the semantics of the requests and responses to/from these services.
- The parking lot example, the Parking Sensor Service provides as a response only a number <0= or <1= given the identifier of a Loop Sensor (e.g. #11).
- The Virtual Entity Parking Spot #01 responds to a request about its occupancy status as <free.= The IoT Service provides data or information associated to specific Devices or Resources, including limited semantic information (e.g. Parking sensor #11, value5<0=, units 5 none); the Virtual IoT Service provides information with richer semantics (<Parking spot #01 is free=), and is closer to being human-readable and understandable.

IoT Service Organization functional group

- The purpose of the IoT Service Organization FG is to host all functional components that support the composition and orchestration of IoT and Virtual Entity services.
- This FG acts as a service hub between several other functional groups such as the IoT Process Management FG when, for example, service requests from Applications or the IoT Process Management are directed to the Resources implementing the necessary Services.
- Therefore, the Service Organization FG supports the association of Virtual Entities with the related IoT Services, and contains functions for discovery, composition, and choreography of services.
- Simple IoT or Virtual Entity Services can be composed to create more complex services, e.g. a control loop with one Sensor Service and one Actuator service with the objective to control the temperature in a building.
- Choreography is the brokerage of Services so that Services can subscribe to other services in a system.

IoT Process Management functional group

- The IoT Process Management FG is a collection of functionalities that allows smooth integration of IoT-related services (IoT Services, Virtual Entity Services, Composed Services) with the Enterprise (Business) Processes.

Management functional group

- The Management FG includes the necessary functions for enabling fault and performance monitoring of the system, configuration for enabling the system to be flexible to changing User demands, and accounting for enabling subsequent billing for the usage of the system.
- Support functions such as management of ownership, administrative domain, rules and rights of functional components, and information stores are also included in the Management FG.

Security functional group

- The Security FG contains the functional components that ensure the secure operation of the system as well as the management of privacy.
- The Security FG contains components for Authentication of Users (Applications, Humans), Authorization of access to Services by Users, secure communication (ensuring integrity and confidentiality of messages) between entities of the system such as Devices, Services, Applications, assurance of privacy of sensitive information relating to Human Users.
- These include privacy mechanisms such as anonymization of collected data, anonymization of resource and Service accesses (Services cannot deduce which Human User accessed the data), and un-linkability (an outside observer cannot deduce the Human User of a service by observing multiple service requests by the same User).

Application functional group

- The Application FG is just a placeholder that represents all the needed logic for creating an IoT application.
- The applications typically contain custom logic tailored to a specific domain such as a Smart Grid.
- An application can also be a part of a bigger ICT system that employs IoT services such

as a supply chain system that uses RFID readers to track the movement of goods within a factory in order to update the Enterprise Resource Planning (ERP) system.

Modular IoT functions

- It is important to note that not all the FGs are needed for a complete actual IoT system.
- The Functional Model, as well as the Functional View of the Reference Architecture, contains a complete map of the potential functionalities for a system realization.
- The functionalities that will eventually be used in an actual system are dependent on the actual system requirements.
- FGs are organized in such a way that more complex functionalities can be built based on simpler ones, thus making the model modular.

Information model

- Information is defined as the enrichment of data (raw values without relevant or usable context) with the right context, so that queries about who, what, where, and when can be answered.
- IoT information model captures the details of a Virtual Entity centric model.

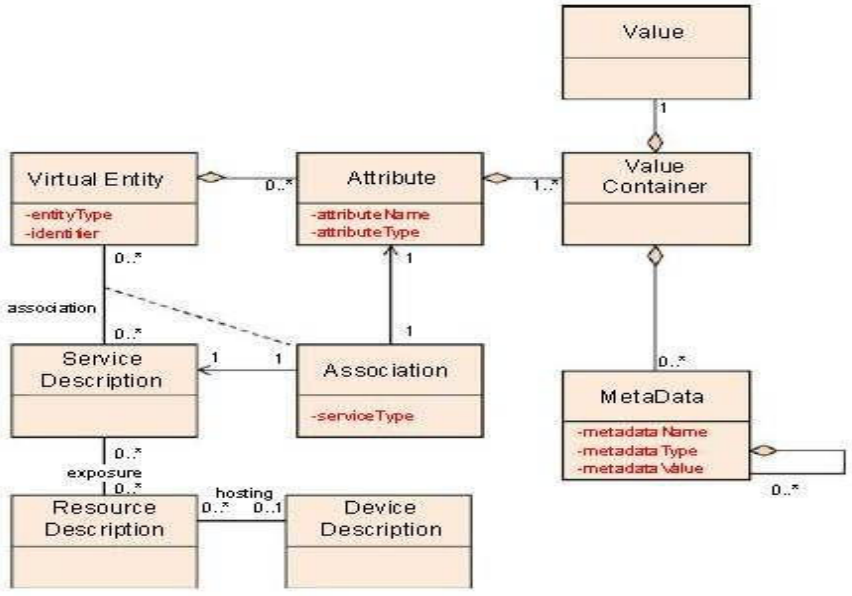


Figure 3.9: High-Level IoT Information Model

- Association class in Figure 3.9 contains information about the specific association

between a Virtual Entity and a related Service.

- On a high-level, the IoT Information Model maintains the necessary information about Virtual Entities and their properties or attributes.
- These properties/attributes can be static or dynamic and enter into the system in various forms, e.g. by manual data entry or reading a sensor attached to the Virtual Entity.
- Virtual Entity attributes can also be digital synchronized copies of the state of an actuator.

- In the presentation of the high-level IoT information model, we omit the attributes that are not updated by an IoT Device (sensor, tag) or the attributes that do not affect any IoT Device (actuator, tag), with the exception of essential attributes such as names andidentifiers.
- Examples of omitted attributes that could exist in a real implementation are room names and floor numbers, in general, context information that is not directly related to IoT Devices, but that is nevertheless important for an actual system.
- The IoT Information Model describes Virtual Entities and their attributes that have one or more values annotated with meta-information or metadata.
- The attribute values are updated as a result of the associated services to a Virtual Entity.
- The associated services are related to Resources and Devices as seen from the IoT Domain Model.
- A Virtual Entity object contains simple attributes/properties:
  (a) entityType to denote the type of entity, such as a human, car, or room (the entity type can be a reference to concepts of a domain ontology, e.g. a car ontology);
  (b) a unique identifier; and
  (c) zero or more complex attributes of the class Attributes.
- The class Attributes should not be confused with the simple attributes of each class.
- This class Attributes is used as a grouping mechanism for complex attributes of the Virtual Entity.
- Objects of the class Attributes, in turn, contain the simple attributes with the self descriptive names attributeName and attributeType.
- The attribute type is the semantic type of the value (e.g. that the value is a temperature value), and can refer to an ontology such as the NASA quantities and units SWEET ontology (NASA JPL 2011).
- The Attribute class also contains a complex attribute ValueContainer that is a container of the multiple values that an attribute can take.
- The container includes complex attributes of the class Value and the class MetaData.
- The container contains exactly one value and meta-information (modeled as the class MetaData), such as a timestamp, describing this single value.
- Objects of the MetaData class can contain MetaData objects as complex attributes, as well as the simple attributes with the self-descriptive namesmetadataName, metadataType, and metadataValue.
- a Virtual Entity is associated with Resources that expose Services about the specific Virtual Entity.
- This association between a Virtual Entity and its Services is captured in the Information Model with the explicit class called Association.
- Objects of this class capture the relationship between objects of the complex Attribute

class (associated with a Virtual Entity) and objects of the Service Description class.

- The class Association describes the relationship between a Virtual Entity and Service Description through the Attribute class, there is a dashed line between Association class and the line between the Virtual Entity and Service Description classes.
- The attribute serviceType can take two values:

  (a) <INFORMATION,= if the associated service is a sensor service (i.e. allows reading of the sensor), or

  (b) <ACTUATION,= if the associated serviceis an actuation service (i.e. allows an action executed on an actuator).
- In both cases, the eventual value of the attribute will be a result of either reading a sensor or controlling an actuator.

Example

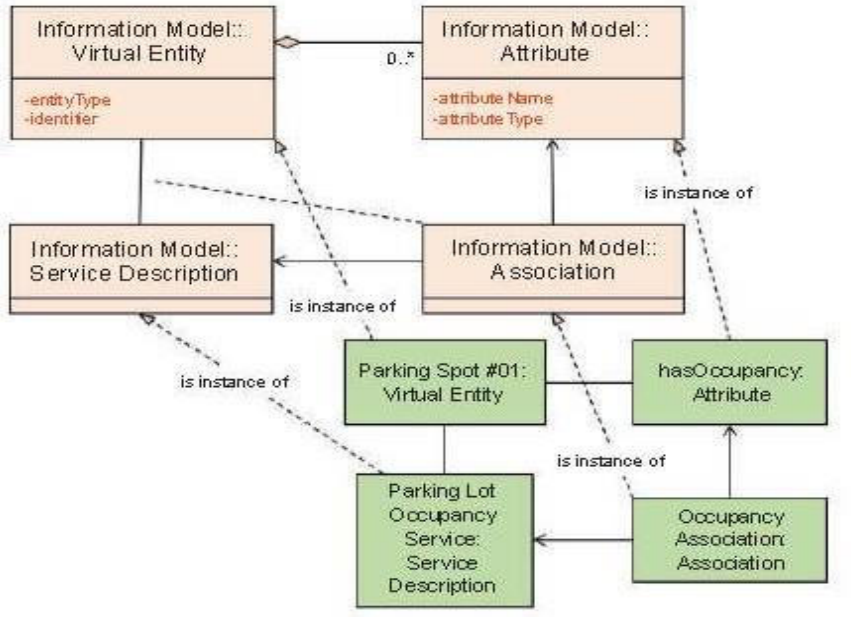


Figure 3.10: IoT information Model example

- Not show all the possible Virtual Entities, but only one corresponding to one parking spot.
- This Virtual Entity is described with one Attribute (among others) called hasOccupancy.
- This Attribute is associated with the Parking Lot Occupancy Service Description through the Occupancy Association.
- The Occupancy Association is the explicit expression of the association (line) between the Parking Spot #1 Virtual Entity and the Parking Lot Occupancy Service.
- The dashed arrows with hollow arrowheads represent the relationship <is instance of= for the information model, as opposed to the Realization relationship for the IoT Domain Model.

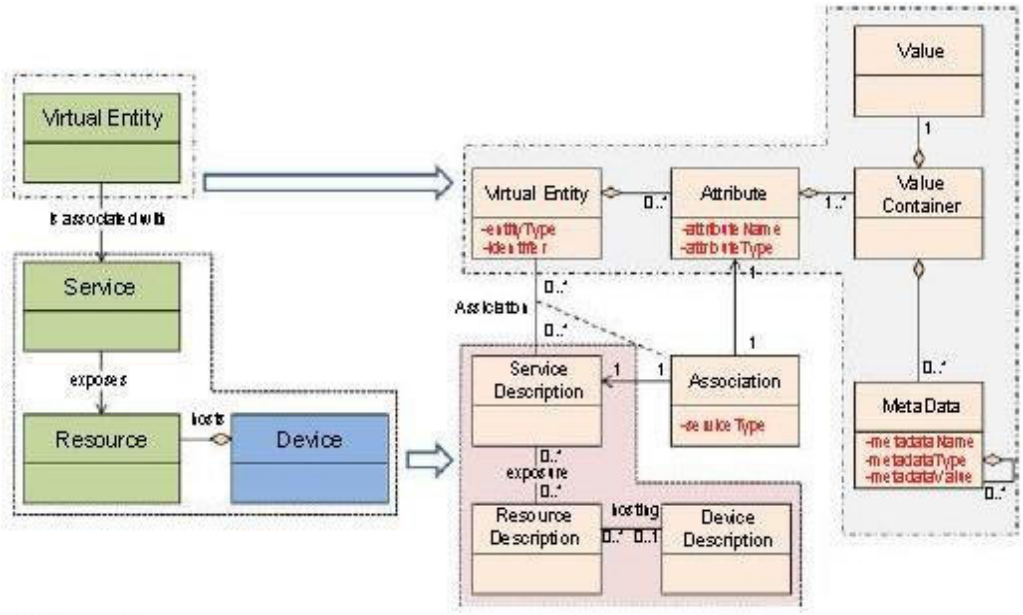


Figure 3.11: Relationship between core concepts of IoT domain Model and IoT Information Model

- Figure 3.11 presents the relationship between the core concepts of the IoT Domain Model and the IoT Information Model.
- The Information Model captures the Virtual Entity in the Domain Model being the <Thing= in the Internet of Things as several associated classes (Virtual Entity, Attribute, Value, MetaData, Value Container) that basically capture the description of a Virtual Entity and its context.
- The Device, Resource, and Service in the IoT Domain Model are also captured in the IoT Information Model because they are used as representations of the instruments and the digital interfaces for interaction with the Physical Entity associated with the Virtual Entity.
- The Information Model is a very high-level model, and omits certain details that could potentially be required in a concrete architecture and an actual system.
- These details could be derived by specific requirements from specific use cases describing the target actual system.
- Several other attributes or properties that could exist in a Virtual Entity description:

  1. Location and its temporal information are important because Physical Entities represented by Virtual Entities exist in space and time. These properties are extremely important when the interested Physical Entities are mobile (e.g. a moving car).A mobile Physical Entity affects the associations between Attributes and related Services, e.g. a person moving close to a camera (sensor) is associated with the Device, Resource, and Services offered by the camera for as long as she stays within the field of view of the camera. In such cases, the temporal availability of the associations between Attributes and Services need to be captured, as availability denotes also temporal observability of the Virtual Entity.

  2. Even non-moving Virtual Entities contain properties that are dynamic with time, and therefore their temporal variations need to be modeled and captured by an information model.

3. Information such as ownership is also important in commercial settings because it may determine access control rules or liability issues. It is important to note that the Attribute class is general enough to capture all the interested properties of a Physical Entity, and thus provides an extensible model whose details can only be specified by the specific actual system in mind.

- The Services in the IoT Domain Model are mapped to the Service Description in the IoT Information Model.
- The Service Description contains the following :

1. Service type, which denotes the type of service, such as Big Web Service or RESTful Web Service. The interfaces of a service are described based on the description language for each service type, for example, Web Application Description Language (WADL) for RESTful Web Services, Web Services Description Language (WSDL) for Big Web Services, Universal Service Description Language (USDL). The interface description includes, among other information, the invocation contact information, e.g. a Uniform Resource Locator (URL).

2. Service area and Service schedule are properties of Services used for specifying the geographical area of interest for a Service and the potential temporal availability of a Service, respectively. For sensing services, the area of interest is equivalent to the observation area, whereas for actuation services the area of interest is the area of operation or impact.

3. Associated resources that the Service exposes.

4. Metadata or semantic information used mainly for service composition. This is information such as the indicator of which resource property is exposed as input or output, whether the execution of the service needs any conditions satisfied before invocation.

- The IoT Information Model also contains Resource descriptions because Resources are associated with Services and Devices in the IoT Domain model.
- **A Resource description contains the following information**:

1. Resource name and identifier for facilitating resource discovery.
2. Resource type, which specifies if the resource is

(a) a sensor resource, which provides sensor readings;

(b) an actuator resource, which provides actuation capabilities (to affect the physical world) and actuator state;

(c) a processor resource, which provides processing of sensor data and output of processed data; a storage resource, which provides storage of data about a Physical Entity;

(d) a tag resource, which provides identification data for Physical Entities.

3. Free text attributes or tags used for capturing typical manual input such as <fire alarm, ceiling.=
4. Indicator of whether the resource is an on-Device resour ce or network resource.
5. Location information about the Device that hosts this resource in case of an on-Device resource.
6. Associated Service information.
7. Associated Device description information.

- A Device is a Physical Entity that could have a sensor, actuator, or tag instantiation.

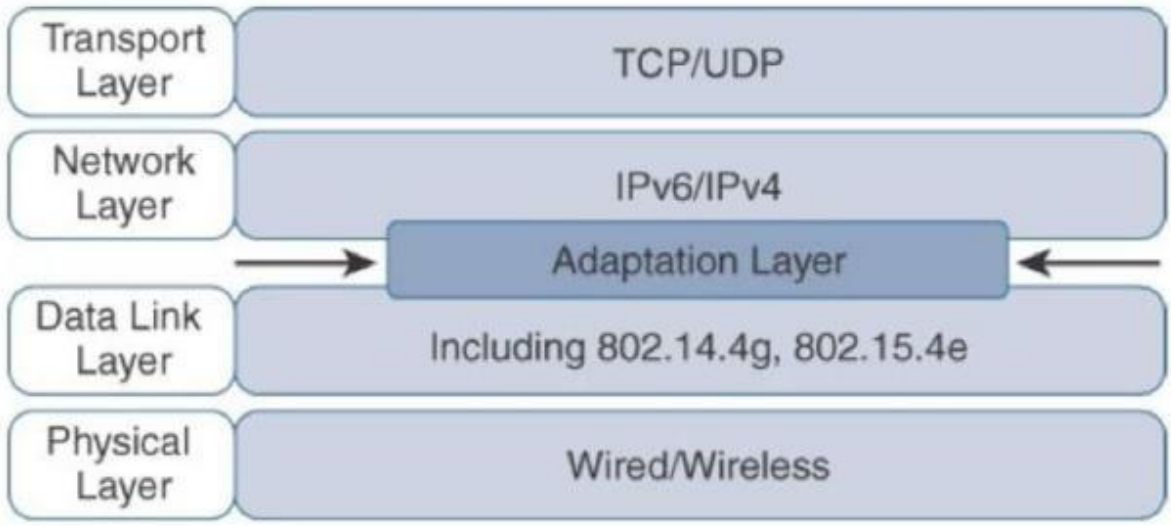
- # Protocols:

  Internet protocol (IP) is a set of rules that dictates how data gets sent to the internet. IoT protocols ensure that information from one device or sensor gets read and understood by another device, a gateway, a service. Protocols they are :

  1. 6LowPAN
  2. RPL
  3. CoAP
  4. MQTT

- **6LoWPAN:(Internet Protocol version 6 (IPv6) over low-power wireless networks):**

  While the Internet Protocol is key for a successful Internet of Things, constrained nodes and constrained networks mandate optimization at various layers and on multiple protocols of the IP architecture. Some optimizations are already available from the market or under development by the IETF. Figure below highlights the TCP/IP layers where optimization is applied.



3.12: **Optimizing IP for IoT Using an Adaptation Layer**

In the IP architecture, the transport of IP packets over any given Layer 1 (PHY) and Layer 2 (MAC) protocol must be defined and documented. The model for packaging IP into lower- layer protocols is often referred to as an adaptation layer.

Unless the technology is proprietary, IP adaptation layers are typically defined by an IETF working group and released as a Request for Comments (RFC). An RFC is a publication from the IETF that officially documents Internet standards, specifications, protocols, procedures, and events. For example, RFC 864 describes how an IPv4 packet gets encapsulated over an Ethernet frame, and RFC 2464 describes how the same function is performed for an IPv6 packet.

IoT-related protocols follow a similar process. The main difference is that an adaptation layer designed for IoT may include some optimizations to deal with constrained nodes and networks. The main examples of adaptation layers optimized for constrained nodes or “things” are the ones under the 6LoWPAN working group and its successor, the 6Lo working group.

The initial focus of the 6LoWPAN working group was to optimize the transmission of IPv6 packets over constrained networks such as IEEE 802.15.4. Figure below shows an example of an IoT protocol stack using the 6LoWPAN adaptation layer beside the well-known IP protocol stack for reference.

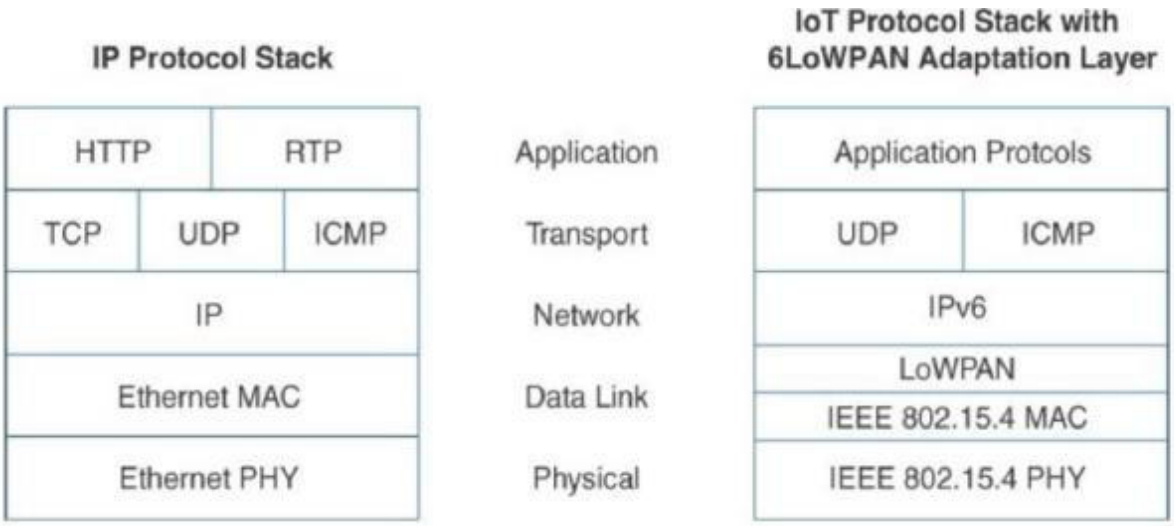


Figure 3.13: Comparison of an IoT Protocol Stack Utilizing 6LoWPAN and an IP Protocol Stack

The 6LoWPAN working group published several RFCs, but RFC 4994 is foundational because it defines frame headers for the capabilities of header compression, fragmentation, and mesh addressing. These headers can be stacked in the adaptation layer to keep these concepts separate while enforcing a structured method for expressing each capability. Depending on the implementation, all, none, or any combination of these capabilities and their corresponding headers can be enabled. Figure below shows some examples of typical 6LoWPAN header stacks.

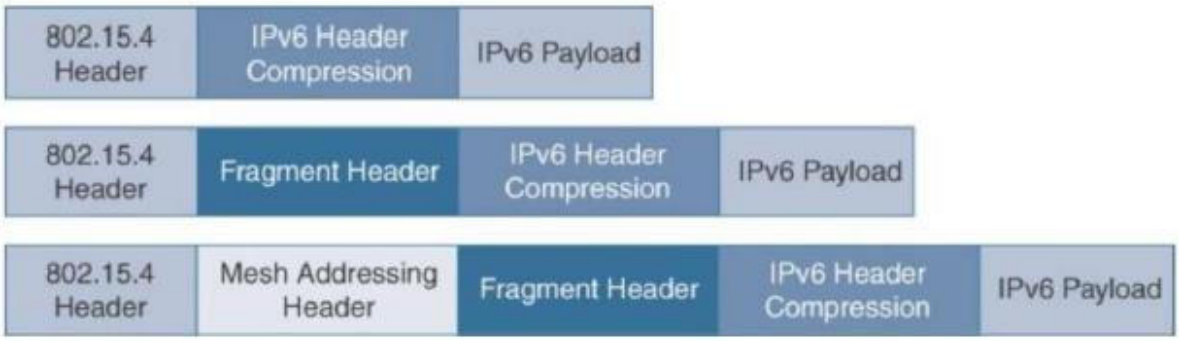


Figure 3.14: 6LoWPAN Header Stack

Header Compression

IPv6 header compression for 6LoWPAN was defined initially in RFC 4944 and subsequently updated by RFC 6282. This capability shrinks the size of IPv6's 40-byte headers and User Datagram Protocol's (UDP's) 8-byte headers down as low as 6 bytes combined in some cases. Note that header compression for 6LoWPAN is only defined for an IPv6 header and not IPv4.

The 6LoWPAN protocol does not support IPv4, and, in fact, there is no standardized IPv4 adaptation layer for IEEE 802.15.4. 6LoWPAN header compression is stateless, and conceptually it is not too complicated. However, a number of factors affect the amount of compression, such as implementation of RFC 4944 versus RFC 6922, whether UDP is included, and various IPv6 addressing scenarios.

At a high level, 6LoWPAN works by taking advantage of shared information known by all nodes from their participation in the local network. In addition, it omits some standard header fields by assuming commonly used values. Figure below highlights an example that shows the amount of reduction that is possible with 6LoWPAN header compression.

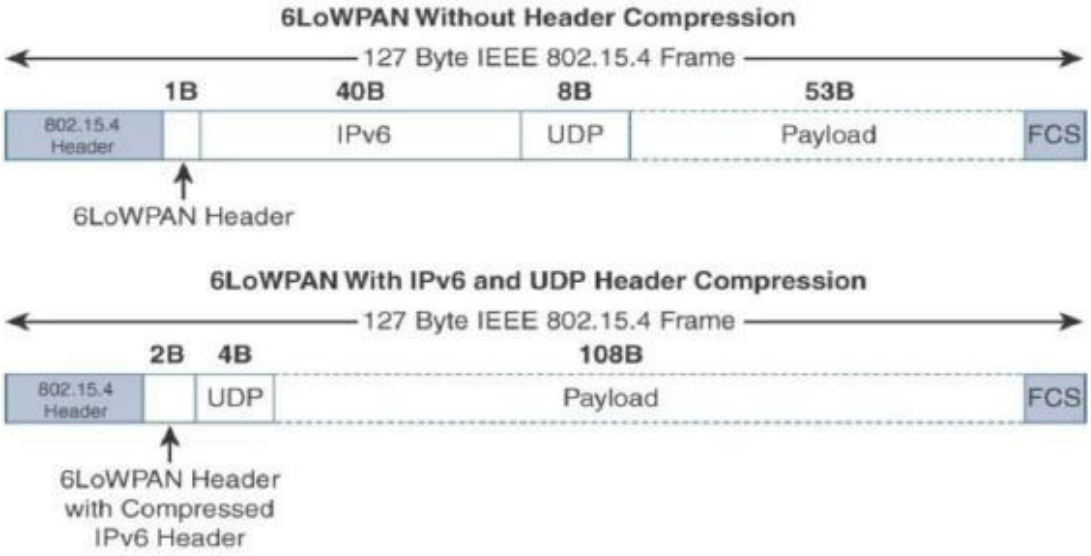


Figure 3.15: 6LoWPAN Header Compression

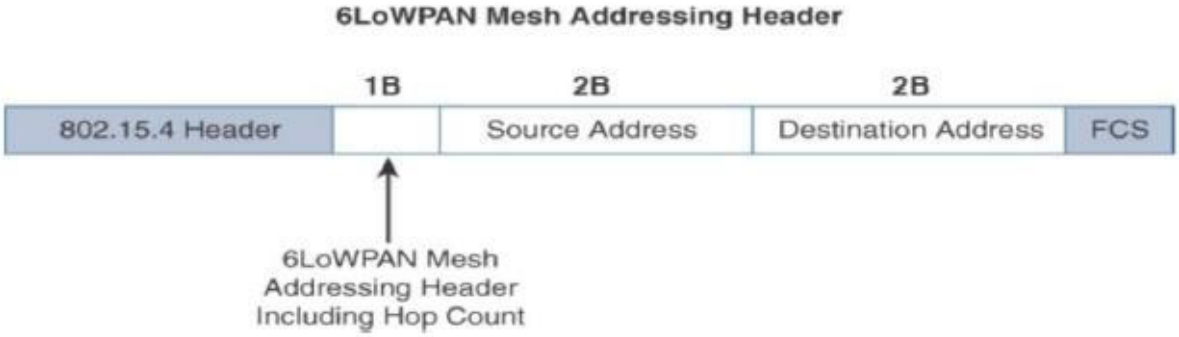
At the top of Figure above, you see a 6LoWPAN frame without any header compression enabled: The full 40- byte IPv6 header and 8-byte UDP header are visible. The 6LoWPAN header is only a single byte in this case. Notice that uncompressed IPv6 and UDP headers leave only 53 bytes of data payload out of the 127- byte maximum frame size in the case of IEEE 802.15.4.

The bottom half of Figure above shows a frame where header compression has been enabled for a best-case scenario. The 6LoWPAN header increases to 2 bytes to accommodate the compressed IPv6 header, and UDP has been reduced in half, to 4 bytes from 8. Most importantly, the header compression has allowed the payload to more than double, from 53 bytes to 108 bytes, which is obviously much more efficient. Note that the 2-byte header compression applies to intra-cell communications, while communications external to the cell may require some field of the header to not be compressed.

Mesh Addressing

The purpose of the 6LoWPAN mesh addressing function is to forward packets over multiple hops. Three fields are defined for this header: Hop Limit, Source Address, and Destination Address. Analogous to the IPv6 hop limit field, the hop limit for mesh addressing also provides an upper limit on how many times the frame can be forwarded. Each hop decrements this value by 1 as it is forwarded. Once the value hits 0, it is dropped and no longer forwarded.

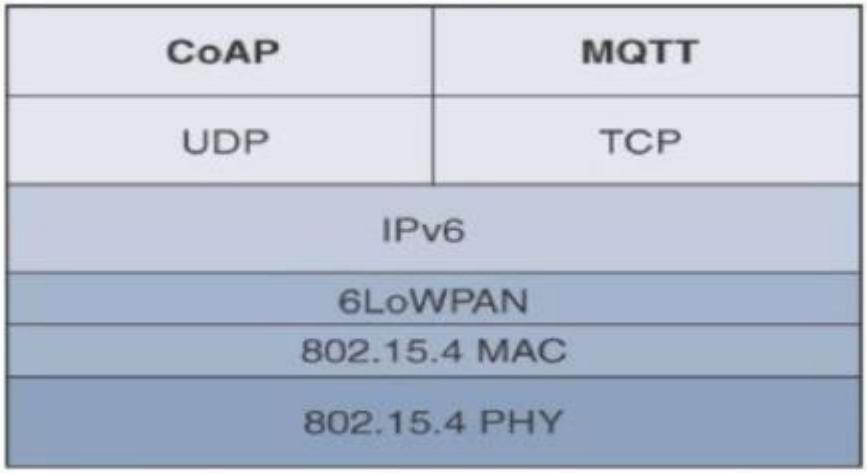
The Source Address and Destination Address fields for mesh addressing are IEEE 802.15.4 addresses indicating the endpoints of an IP hop. Figure below details the 6LoWPAN mesh addressing header fields.



Note that the mesh addressing header is used in a single IP subnet and is a Layer 2 type of routing known as mesh-under. RFC 4944 only provisions the function in this case as the definition of Layer 2 mesh routing specifications was outside the scope of the 6LoWPAN working group, and the IETF doesn't define "Layer 2 routing." An implementation performing Layer 3 IP routing does not need to implement a mesh addressing header unless required by a given technology profile.

IoT Application Layer Protocols (COAP ANS MQTT)

When considering constrained networks and/or a large-scale deployment of constrained nodes, verbose web-based and data model protocols, may be too heavy for IoT applications. To address this problem, the IoT industry is working on new lightweight protocols that are better suited to large numbers of constrained nodes and networks. Two of the most popular protocols are CoAP and MQTT. Figure below highlights their position in a common IoT protocol stack.

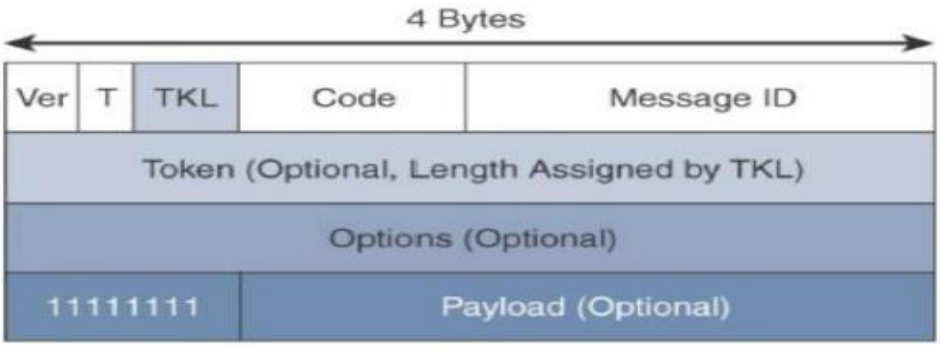


1. **CoAP (Constrained Application Protocol (CoAP)):**

Constrained Application Protocol (CoAP) resulted from the IETF Constrained RESTful Environments (CoRE) working group's efforts to develop a generic framework for resource-oriented applications targeting constrained nodes and networks. The CoAP framework defines simple and flexible ways to manipulate sensors and actuators for data or device management.

The CoAP messaging model is primarily designed to facilitate the exchange of messages over UDP between endpoints, including the secure transport protocol Datagram Transport Layer Security (DTLS).

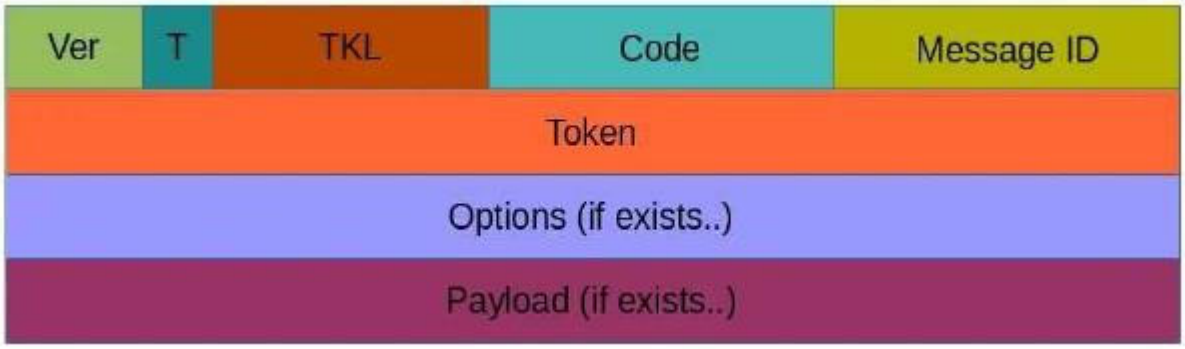
From a formatting perspective, a CoAP message is composed of a short fixed- length Header field (4 bytes), a variable-length but mandatory Token field (0–8 bytes), Options fields if necessary, and the Payload field. Figure below details the CoAP message format, which delivers low overhead while decreasing parsing complexity.



CoAP Message Format

The CoAP message format is relatively simple and flexible. It allows CoAP to deliver low overhead, which is critical for constrained networks, while also being easy to parse and process for constrained devices.

CoAP Message Fields

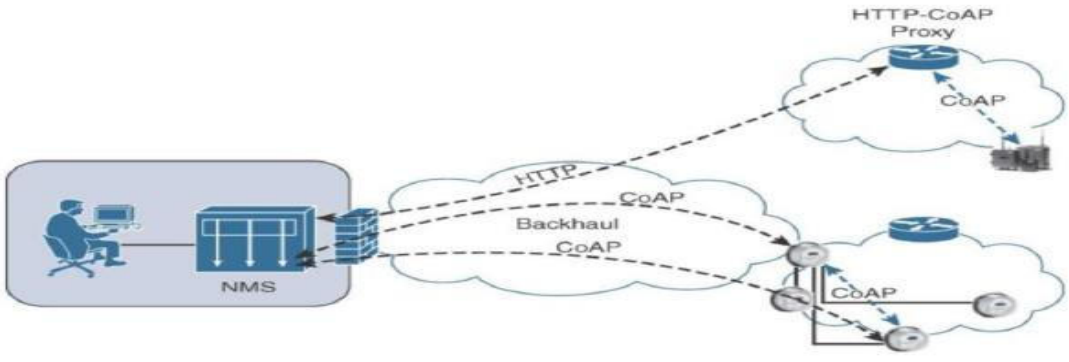


- **Ver**: It is a 2 bit unsigned integer indicating the version
- **T**: it is a 2 bit unsigned integer indicating the message type: 0 confirmable, 1 non-confirmable
- **TKL**: Token Length is the token 4 bit length
- **Code**: It is the code response (8 bit length)
- **Message ID**: It is the message ID expressed with 16 bit
- **Token** :With a length specified by TKL, correlates request and responses
- **Option** : specifies the option number, length and option value.
- **Payload** :carries the COAP application data.

CoAP can run over IPv4 or IPv6. However, it is recommended that the message fit within a single IP packet and UDP payload to avoid fragmentation. For IPv6, with the default MTU size being 1280 bytes and allowing for no fragmentation across nodes, the maximum CoAP message size could be up to 1152 bytes, including 1024 bytes for the payload. In the

case of IPv4, as IP fragmentation may exist across the network, implementations should limit themselves to more conservative values and set the IPv4 Don't Fragment (DF) bit.

CoAP communications across an IoT infrastructure can take various paths. Connections can be between devices located on the same or different constrained networks or between devices and generic Internet or cloud servers, all operating over IP. Proxy mechanisms are also defined, and RFC 7252 details a basic HTTP mapping for CoAP. As both HTTP and CoAP are IP-based protocols, the proxy function can be located practically anywhere in the network, not necessarily at the border between constrained and non-constrained networks.

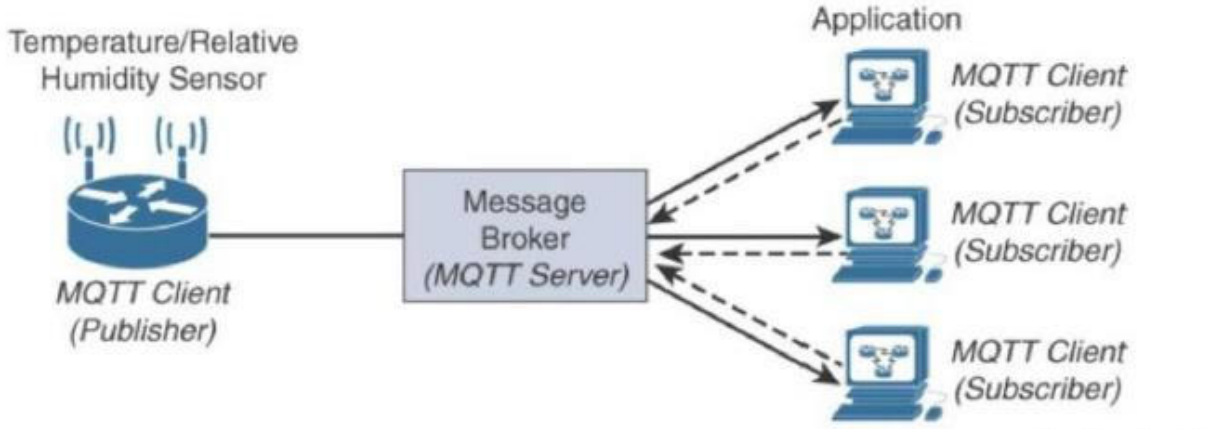


Just like HTTP, CoAP is based on the REST architecture, but with a "thing" acting as both the client and the server. Through the exchange of asynchronous messages, a client requests an action via a method code on a server resource. A uniform resource identifier (URI) localized on the server identifies this resource. The server responds with a response code that may include a resource representation. The CoAP request/response semantics include the methods GET, POST, PUT, and DELETE.

Message Queuing Telemetry Transport (MQTT):

At the end of the 1990s, engineers from IBM and Arcom (acquired in 2006 by Eurotech) were looking for a reliable, lightweight, and cost-effective protocol to monitor and control a large number of sensors and their data from a central server location, as typically used by the oil and gas industries. Their research resulted in the development and implementation of the Message Queuing Telemetry Transport (MQTT) protocol that is now standardized by the Organization for the Advancement of Structured Information Standards (OASIS).

The selection of a client/server and publish/subscribe framework based on the TCP/IP architecture, as shown in Figure below.

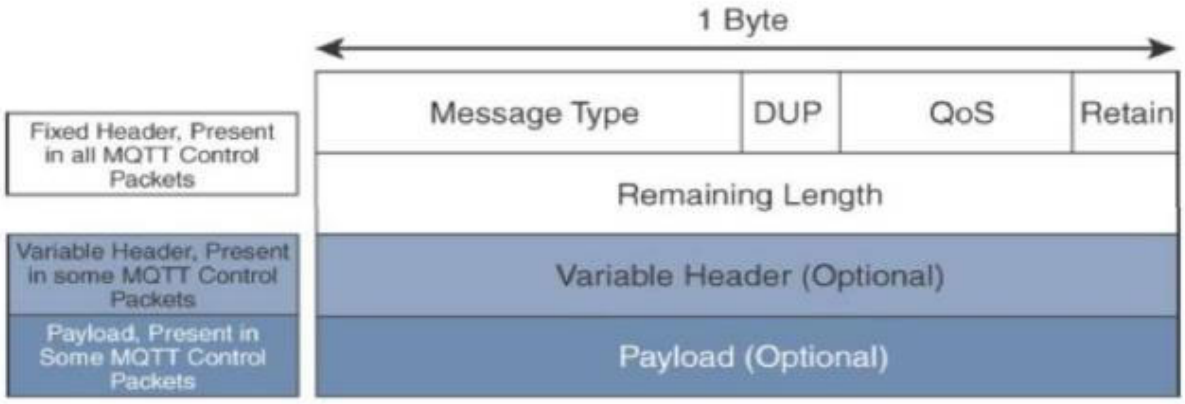


An MQTT client can act as a publisher to send data (or resource information) to an MQTT server acting as an MQTT message broker. In the example illustrated in Figure 2.22, the MQTT client on the left side is a temperature (Temp) and relative humidity (RH) sensor that publishes its Temp/RH data. The MQTT server (or message broker) accepts the network connection along with application messages, such as Temp/RH data, from the publishers. It also handles the subscription and un-subscription process and pushes the application data to MQTT clients acting as subscribers.

The application on the right side of Figure above is an MQTT client that is a subscriber to the Temp/RH data being generated by the publisher or sensor on the left. This model, where subscribers express a desire to receive information from publishers, is well known. A great example is the collaboration and social networking application Twitter.

With MQTT, clients can subscribe to all data (using a wildcard character) or specific data from the information tree of a publisher. In addition, the presence of a message broker in MQTT decouples the data transmission between clients acting as publishers and subscribers. In fact, publishers and subscribers do not even know (or need to know) about each other. A benefit of having this decoupling is that the MQTT message broker ensures that information can be buffered and cached in case of network failures. This also means that publishers and subscribers do not have to be online at the same time. MQTT control packets run over a TCP transport using port 1883. TCP ensures an ordered, lossless stream of bytes between the MQTT client and the MQTT server. Optionally, MQTT can be secured using TLS on port 8883, and WebSocket (defined in RFC 6455) can also be used.

MQTT is a lightweight protocol because each control packet consists of a 2-byte fixed header with optional variable header fields and optional payload. You should note that a control packet can contain a payload up to 256 MB. Figure 2.23 provides an overview of the MQTT message format.



MQTT Message Format

Compared to the CoAP message format, MQTT contains a smaller header of 2 bytes compared to 4 bytes for CoAP. The first MQTT field in the header is Message Type, which identifies the kind of MQTT packet within a message. Fourteen different types of control packets are specified in MQTT version 3.1.1. Each of them has a unique value that is coded into the Message Type field. Note that values 0 and 15 are reserved. MQTT message types are summarized in Table.

Message Type	**Value**	**Flow**	**Description**
CONNECT	1	Client to server	Request to connect
CONNACK	2	Server to client	Connect acknowledgement
PUBLISH	3	Client to server / Server to client	Publish message
PUBACK	4	Client to server / Server to client	Publish acknowledgement
PUBREC	5	Client to server / Server to client	Publish received
PUBREL	6	Client to server / Server to client	Publish release
PUBCOMP	7	Client to server / Server to client	Publish complete
SUBSCRIBE	8	Client to server	Subscribe request
SUBACK	9	Server to client	Subscribe acknowledgement
UNSUBSCRIBE	10	Client to server	Unsubscribe request
UNSUBACK	11	Server to client	Unsubscribe acknowledgement
PINGREQ	12	Client to server	Ping request
PINGRESP	13	Server to client	Ping response
DISCONNECT	14	Client to server	Client disconnecting

The next field in the MQTT header is DUP (Duplication Flag). This flag, when set, allows the client to notate that the packet has been sent previously, but an acknowledgement was not received. The QoS header field allows for the selection of three different QoS levels. The next field is the Retain flag. Only found in a PUBLISH message, the Retain flag notifies the server to hold onto the message data. This allows new subscribers to instantly receive the last known value without having to wait for the next update from the publisher.

The last mandatory field in the MQTT message header is Remaining Length. This field specifies the number of bytes in the MQTT packet following this field.

MQTT sessions between each client and server consist of four phases: session establishment, authentication, data exchange, and session termination. Each client connecting to a server has a unique client ID, which allows the identification of the MQTT session between both parties. When the server is delivering an application message to more than one client, each client is treated independently.

Subscriptions to resources generate SUBSCRIBE/SUBACK control packets, while un- subscription is performed through the exchange of UNSUBSCRIBE/UNSUBACK control packets. Graceful termination of a connection is done through a DISCONNECT control packet, which also offers the capability for a client to reconnect by re-sending its client ID to resume the operations.

A message broker uses a topic string or topic name to filter messages for its subscribers. When subscribing to a resource, the subscriber indicates the one or more topic levels that are used to structure the topic name. The forward slash (/) in an MQTT topic name is used to separate each level within the topic tree and provide a hierarchical structure to the topic names.

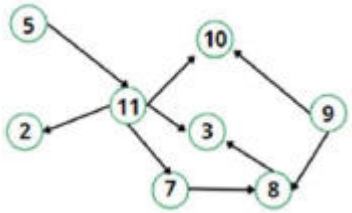
Comparison of CoAP and MQTT

Factor	**CoAP**	**MQTT**
Main transport protocol	UDP	TCP
Typical messaging	Request/response	Publish/subscribe
Effectiveness in LLNs	Excellent	Low/fair (Implementations pairing UDP with MQTT are better for LLNs.)
Security	DTLS	SSL/TLS
Communication model	One-to-one	Many-to-many
Strengths	Lightweight and fast, with low overhead, and suitable for constrained networks; uses a RESTful model that is easy to code to; easy to parse and process for constrained devices; support for multicasting; asynchronous and synchronous messages	TCP and multiple QoS options provide robust communications; simple management and scalability using a broker architecture
Weaknesses	Not as reliable as TCP-based MQTT, so the application must ensure reliability.	Higher overhead for constrained devices and networks; TCP connections can drain low-power devices; no multicasting support

RPL:(RPL (IPv6 Routing protocol):

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.



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:

Storing mode: All modes contain the entire routing table of the RPL domain. Every node knows how to reach every other node directly.

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.

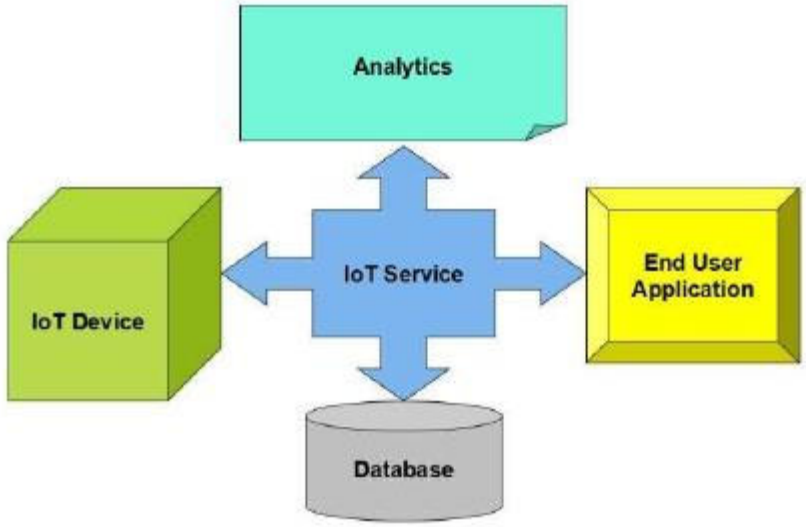
Implementation of RPL Protocol:

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 Dunkel's.

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

IOT FRAMEWORK-THING SPEAK:

The Internet of Things(IoT) is a system of 'connected things'. The things generally comprise of an embedded operating system and an ability to communicate with the internet or with the neighbouring things. One of the key elements of a generic IoT system that bridges the various 'things' is an IoT service. An interesting implication from the 'things' comprising the IoT systems is that the things by themselves cannot do anything. At a bare minimum, they should have an ability to connect to other 'things'. But the real power of IoT is harnessed when the things connect to a 'service' either directly or via other 'things'. In such systems, the service plays the role of an invisible manager by providing capabilities ranging from simple data collection and monitoring to complex data analytics. The below diagram illustrates where an IoT service fits in an IoT ecosystem:



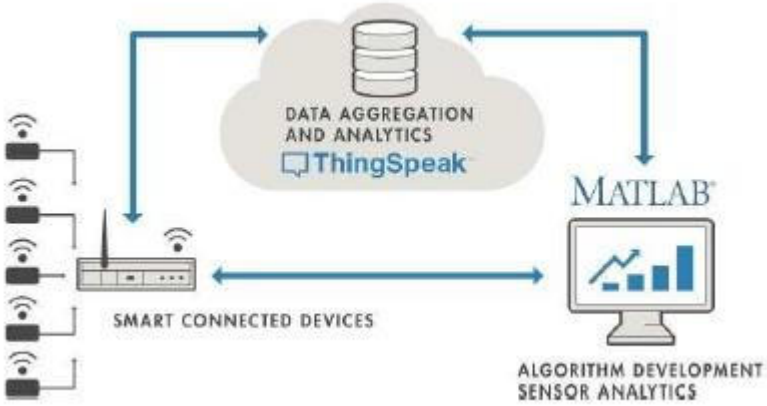
What is Thing Speak:

Thing Speak is a platform providing various services exclusively targeted for building IoT applications. It offers the capabilities of real-time data collection, visualizing the collected data in the form of charts, ability to create plugins and apps for collaborating with web services, social network and other APIs. We will consider each of these features in detail below.

The core element of Thing Speak is a 'Thing Speak Channel'. A channel stores the data that we send to Thing Speak and comprises of the below elements:

- 8 fields for storing data of any type - These can be used to store the data from a sensor or from an embedded device**.**
- 3 location fields - Can be used to store the latitude, longitude and the elevation. These are very useful for tracking a moving device.
- 1 status field - A short message to describe the data stored in the channel.

To use Thing Speak, we need to signup and create a channel. Once we have a channel, we can send the data, allow Thing Speak to process it and also retrieve the same. Let us start exploring Thing Speak by signing up and setting up a channel.



Thing Speak allows for IoT analytics with its cloud supportive features that make it easier for you to analyse the live data. It supports MATLAB code that a developer can write and perform actions on the live data streams. It includes different functions like data visualization, pre- processing, analysis, and more.

The functions included in Thing Speak are:

- Location Tracing
- Information distribution through public channels and gathering through a private channel
- Includes cloud support
- Online analytics of data to identify patterns and relations
- device executions supported through command schedule
- Social sharing support through Twilio and Twitter
- Alerts for every reaction

It allows one to prototype an IoT system in advance before they start the development. The analytics and data generated through Thing Speak are incredibly reliable as the tool enables performing the best operations and delivers excellent results to make your IoT system full proof.