UNIT-I

Introduction to IoT:

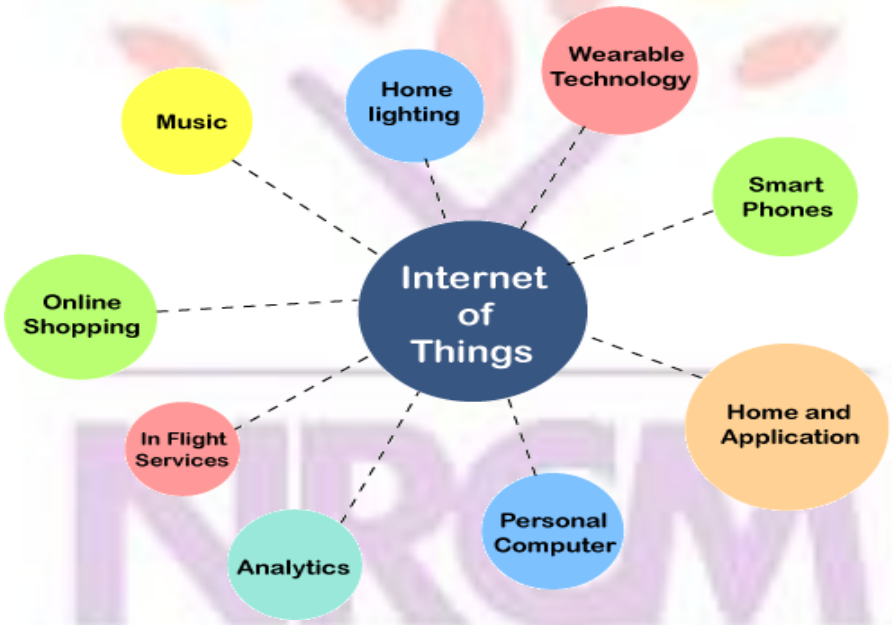
Internet of things (IoT) comprises things that have unique identities and are connected to the internet.

While many existing devices, such as networked computers or 4G enabled mobile phones, already have some form of unique identities and are also

connected to the internet, the focus on IoT is in the configuration, control and networking via the internet of devices or "things" that are traditionally not associated with the internet.

The scope of IoT is not limited to just connecting things to the Internet. IoT also allows these things to communicate and exchange data while executing meaningful applications towards a common user or machine goal.

Applications of IoT:



1) Home
2) Cities
3) Environment
4) Energy
5) Retail
6) Logistics

7) Agriculture

8) Industry

9) Health & Life Style

Definition and characteristics of IOT:

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 information network, often communicate data associated with users and their environments.

Characteristics:

- Dynamic and self-adapting
- Self –configuring
- Interoperable communication protocols
- Unique identity
- Integrated into Information Network

Physical Design of IoT :

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

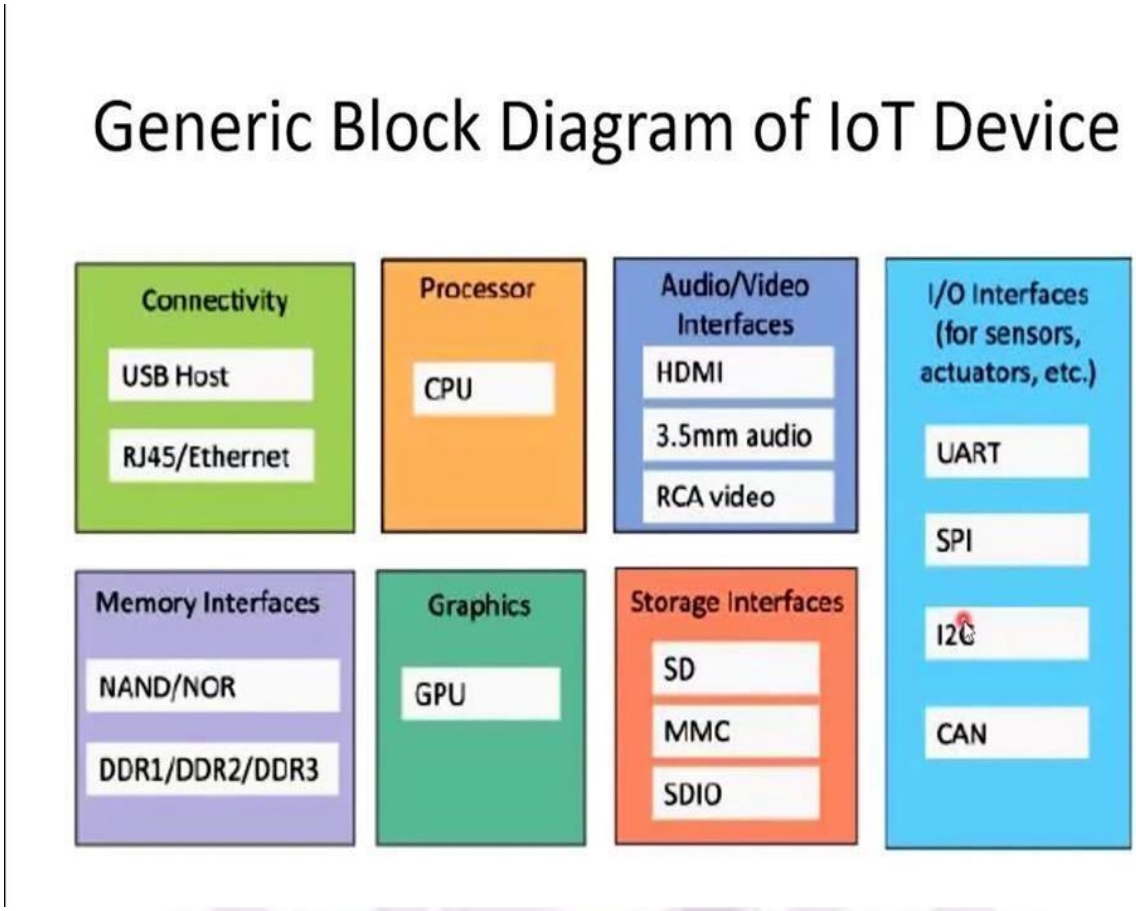
IoT devices can:

Exchange data with other connected devices and applications (directly or indirectly), Collect data from other devices and process the data locally or Send the data to centralized servers or cloud-based application back-ends for processing the data,Perform some tasks locally and other tasks within the IoT infrastructure, based on temporal and space constraints

Generic block diagram of an IoT Device

- An IoT device may consist of several interfaces for connections to other devices, both wired and wireless.
- I/O interfaces for sensors
- Interfaces for Internet connectivity

- Memory and storage interfaces
- Audio/video interfaces.



- HDMI: High-definition multimedia Interface.
- 3.5mm: Audio Jack which headphone adapter.
- RCA: Radio corporation of America.
- UART: Universal Asynchronous Receiver Transmitter.
- SPI: Serial Peripheral Interface.
- I2C: Inter integrated circuit
- CAN: Controller Area Network used for Micro-controllers and devices to communicate.
- SD: Secure digital (memory card)
- MMC: multimedia card
- SDIO: Secure digital Input Output
- GPU: Graphics processing unit.

- DDR: Double data rate

IoT Protocols:

Link Layer : Protocols determine how data is physically sent over the network's physical layer or medium. Local network connect to which host is attached. Hosts on the same link exchange data packets over the link layer using link layer protocols. Link layer determines how packets are coded and signalled by the h/w device over the medium to which the host is attached.

Protocols:

- 802.3-Ethernet: IEEE802.3 is collection of wired Ethernet standards for the link layer. Eg: 802.3 uses co-axial cable; 802.3i uses copper twisted pair connection; 802.3j uses fiber optic connection; 802.3ae uses Ethernet overfiber.
- 802.11-WiFi: IEEE802.11 is a collection of wireless LAN(WLAN) communication standards including extensive description of link layer. Eg: 802.11a operates in 5GHz band, 802.11b and 802.11g operates in 2.4GHz band, 802.11n operates in 2.4/5GHz band, 802.11ac operates in 5GHz band, 802.11ad operates in 60Ghzband.
- 802.16 - WiMax: IEEE802.16 is a collection of wireless broadband standards including exclusive description of link layer. WiMax provide data rates from 1.5 Mb/s to 1Gb/s.
- 802.15.4-LR-WPAN: IEEE802.15.4 is a collection of standards for low rate wireless personal area network(LR-WPAN). Basis for high level communication protocols such as ZigBee. Provides data rate from 40kb/s to250kb/s.
- 2G/3G/4G-Mobile Communication: Data rates from 9.6kb/s(2G) to up to100Mb/s(4G). B)

Network/Internet Layer:

Responsible for sending IP datagrams from source n/w to destination n/w. Performs the host addressing and packet routing. Datagrams contains source and destination address.

Protocols:

- IPv4: Internet Protocol version4 is used to identify the devices on a n/w using a hierarchical addressing scheme. 32 bit address. Allows total of 2*32addresses.
- IPv6: Internet Protocol version6 uses 128 bit address scheme and allows 2*128 addresses.
- 6LOWPAN:(IPv6 over Low power Wireless Personal Area Network) operates in 2.4 GHz frequency range and data transfer 250 kb/s.

Transport Layer:

Provides end-to-end message transfer capability independent of the underlying n/w. Set up on connection with ACK as in TCP and without ACK as in UDP. Provides functions such as error control, segmentation, flow control and congestion control.

Protocols:

TCP: Transmission Control Protocol used by web browsers(along with HTTP and HTTPS), email(along with SMTP, FTP). Connection oriented and

stateless protocol. IP Protocol deals with sending packets, TCP ensures reliable transmission of protocols in
order. Avoids n/w congestion and congestion collapse.

UDP: User Datagram Protocol is connectionless protocol. Useful in time sensitive
applications, very small data units to exchange. Transaction oriented and stateless protocol. Does not provide guaranteed delivery.

C) **Application Layer**: Defines how the applications interface with lower layer protocols to
send data over the n/w. Enables process-to-process communication using ports.

Protocols:

- HTTP: Hyper Text Transfer Protocol that forms foundation of WWW. Follow request response model Stateless protocol.
- CoAP: Constrained Application Protocol for machine-to-machine(M2M) applications with constrained devices, constrained environment and constrained n/w. Uses client-server architecture.
- Web Socket: allows full duplex communication over a single socket connection.
- MQTT: Message Queue Telemetry Transport is light weight messaging protocol based on publish-subscribe model. Uses client server architecture. Well suited for constrained environment.
- XMPP: Extensible Message and Presence Protocol for real time communication and streaming XML data between network entities. Support client-server and server-server communication.
- DDS: Data Distribution Service is data centric middleware standards for device-to-device or machine-to-machine communication. Uses publish-subscribe model.
- AMQP: Advanced Message Queuing Protocol is open application layer protocol for business messaging. Supports both point-to-point and publish-subscribe model.

LOGICAL DESIGN of IoT

Refers to an abstract represent of entities and processes without going into the low level specifies of implementation.

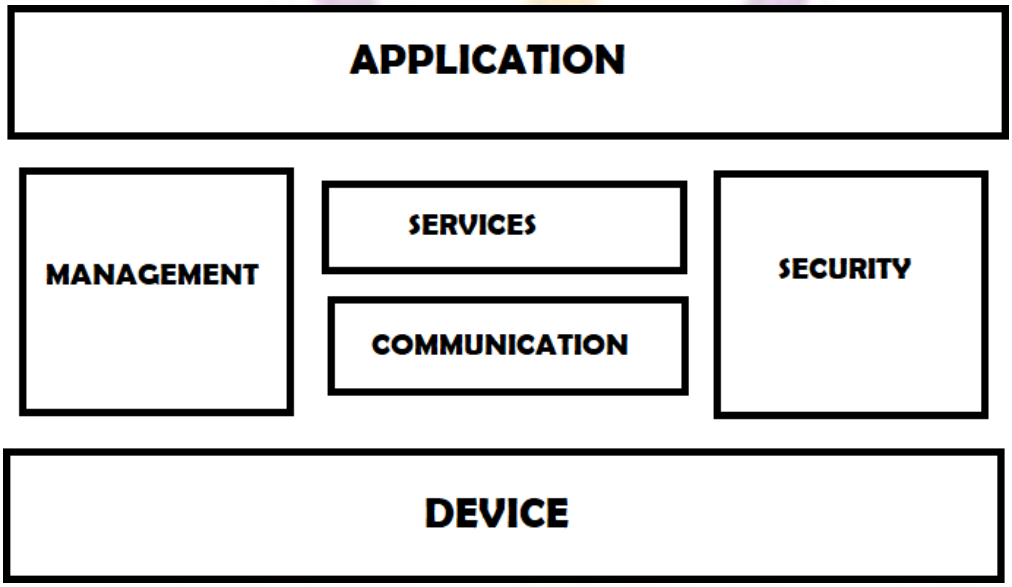
1) IoT Functional Blocks
2) IoT Communication Models
3) IoT Comm. APIs

1) IoT Functional Blocks:

Provide the system the capabilities for identification, sensing, actuation, communication and management

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

- Communication: handles the communication for IoT system.
- Services: for device monitoring, device control services, data publishing services and services for device discovery.
- Management: Provides various functions to govern the IoT system.
- Security: Secures IoT system and priority functions such as authentication, authorization, message and context integrity and data security.
- Application: IoT application provide an interface that the users can use to control and monitor various aspects of IoT system.

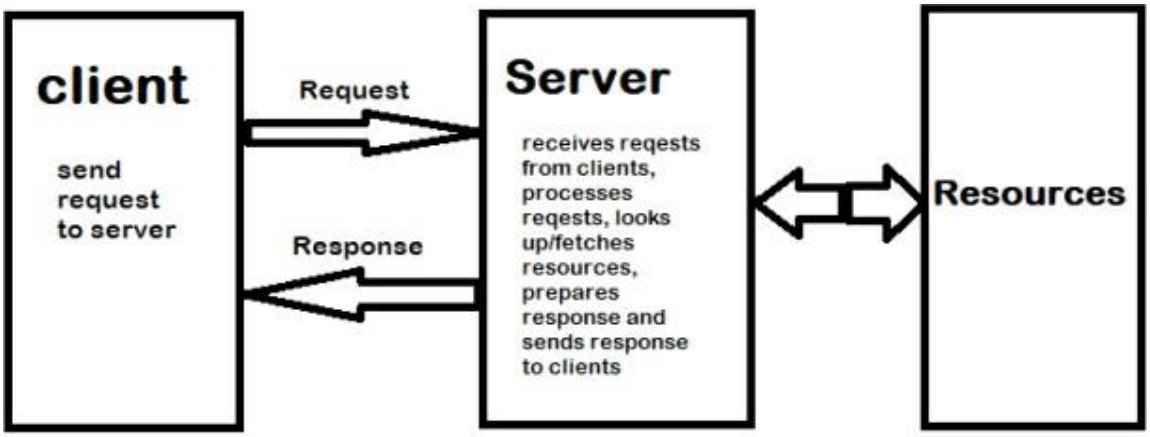


2) IoT Communication Models:

A) Request-Response B) Publish-Subscribe C)Push-Pull D) Exclusive Pair

A) Request-Response

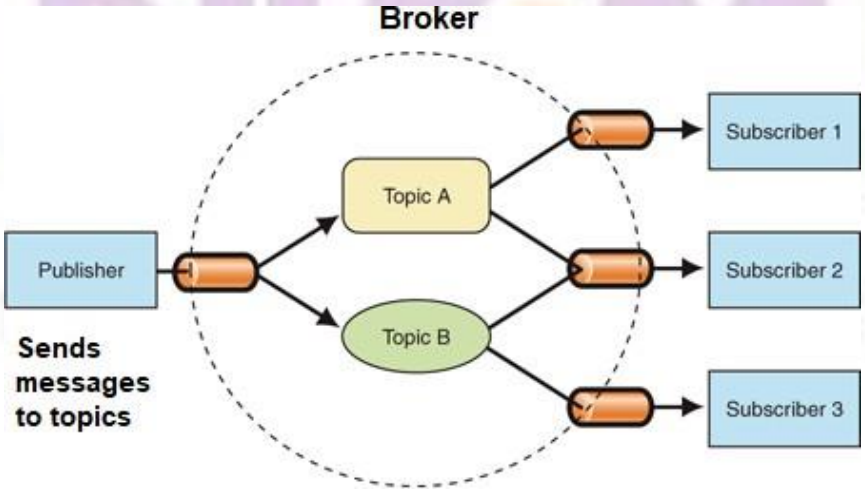
Request-Response is a communication model in which the client sends requests to the server and the server responds to the requests. When the server receives a request, it decides how to respond, fetches the data, retrieves resource representations, prepares the response, and then sends the response to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



Request-Response Communication Model

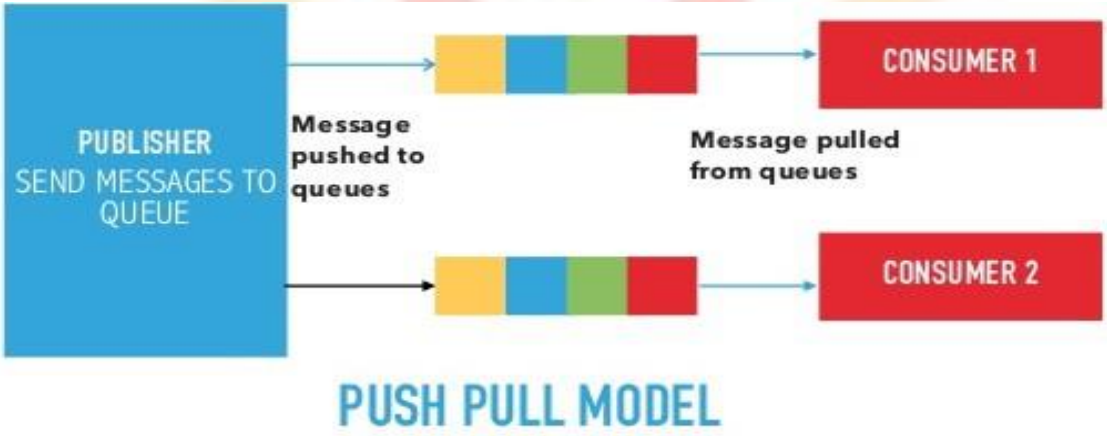
B) Publish-Subscribe communication model:

- 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



C)Push-Pull communication model:

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



PUSH PULL MODEL

D) Exclusive Pair communication model:

- Exclusive Pair is a bidirectional, 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 COMMUNICATION MODEL

4)IoT Communication APIs:

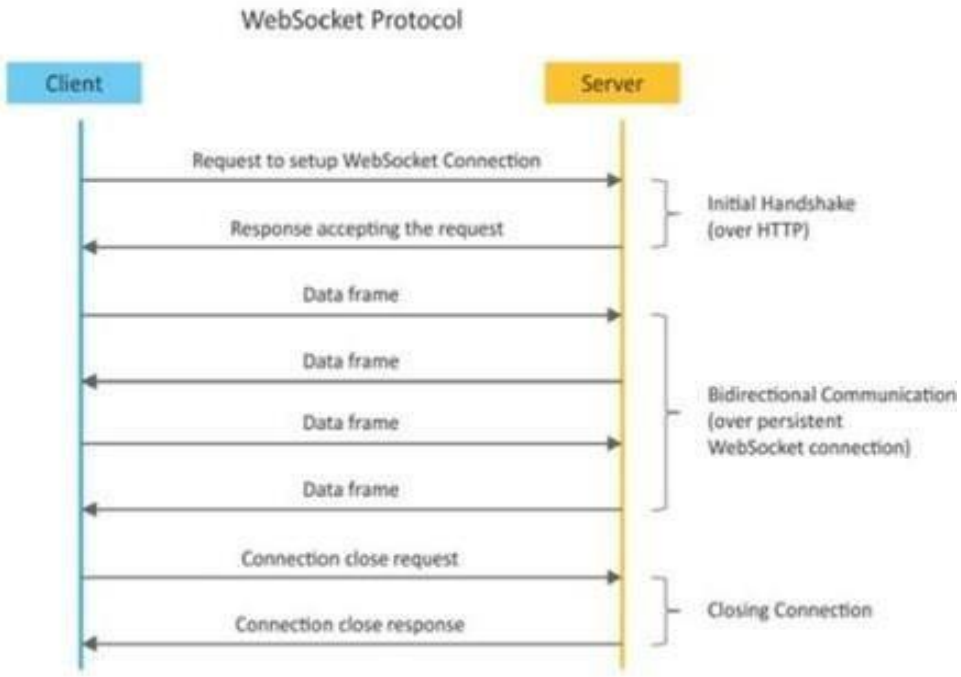
a) REST based communication APIs(Request- Response Based Model)

b) Web Socket based Communication APIs(Exclusive Pair Based Model)

Request-Response model used by REST:

RESTful webservice is a collection of resources which are represented by URLs. RESTful web API has a base URL(e.g:http://example.com/api/tasks/). The clients and requests to these URIs using the methods defined by the HTTP protocol(e.g: GET, PUT, POST or DELETE). A RESTful web service can support various internet media types.

b) WebSocket Based Communication APIs: WebSocket APIs allow bi-directional, full duplex communication between clients and servers. WebSocket APIs follow the exclusive pair communication model.



Sensing:

- Generally speaking, a sensor is a device that is able to detect changes in an environment. By itself, a sensor is useless, but when we use it in an electronic system, it plays a key role. A sensor is able to measure a physical phenomenon (like temperature, pressure, and so on) and transform it into an electric signal. These three features should be at the base of a good sensor:
- It should be sensitive to the phenomenon that it measures
- It should not be sensitive to other physical phenomena
- It should not modify the measured phenomenon during the measurement process
- There is a wide range of sensors we can exploit to measure almost all the physical properties around us. A few common sensors that are widely adopted in everyday life include thermometers, pressure sensors, light sensors, accelerometers, gyroscopes, motion sensors, gas sensors and many more.
- A sensor can be described using several properties, the most important

Being

- **Range**: The maximum and minimum values of the phenomenon that the sensor can measure.
- **Sensitivity**: The minimum change of the measured parameter that causes a detectable change in output signal.
- **Resolution**: The minimum change in the phenomenon that the sensor can detect.

Sensor Classification:

Sensors can be grouped using several criteria:

Passive or Active: Passive sensors do not require an external power source to monitor an environment, while Active sensors require such a source in order to work. A passive sensor is one which just 'listens' to what is happening.

Examples include:

- A light sensor which detects if a light is shining on it.
- An infra-red sensor which detects the temperature of an object.

An active sensor is one which transmits a signal into the environment and then measures the response that comes back.

One example is an ultrasonic system:

- A pulse of ultrasound is emitted.
- If an object is in the way, the pulse is reflected back.
- The sensor detects it.
- The time taken between emission and detection gives an indication of the distance of the object.

Another classification is based on the method used to detect and measure the property (mechanical, chemical, etc.).

Analog and Digital:

Analog sensors produce an analog, or continuous, signal while digital sensors produce a discrete signal.

There are different types of sensors that produce continuous analog output signal and these sensors are analog sensors. This continuous output signal produced by the analog sensors is proportional to the measure and.

Generally, There are various types of analog sensors; practical examples of various types of analog sensors are as follows: accelerometers, pressure sensors, light sensors, sound sensors, temperature sensors, and so on. Unlike analog sensor, Digital Sensor produce discrete values (0 and 1's). Discrete values often called digital or binary signals in digital communication.

Electronic sensors or electrochemical sensors in which data conversion and data transmission take place digitally are digital sensors. These digital sensors are replacing analog sensors as they are capable of overcoming the drawbacks of analog sensors. The digital sensor consists of majorly three components such as senor, cable, and transmitter. But, In digital sensors, the signal measured directly converted into digital signal output inside the digital sensor itself. So, this digital signal transmitted through cable digitally. There are different types of digital sensors that overcome the disadvantages of analog sensors.

Then, scaler sensors basically measure scalar variables which can measure only the changes in the magnitude whereas, the vector senses not only the magnitude, but also the direction. So, scalar sensor example would be temperature sensor is an example of scalar sensor because you know irrespective of which orientation you put, the sensor temperature sensor or in which direction you are taking it, it is going to give you the magnitude value.

Analog sensors produce an analog, or continuous, signal while digital sensors produce a discrete signal.

There are different types of sensors that produce continuous analog output signal and these sensors are analog sensors. This continuous output signal produced by the analog sensors is proportional to the measure and.

Generally, There are various types of analog sensors; practical examples of various types of analog sensors are as follows: accelerometers, pressure sensors, light sensors, sound sensors, temperature sensors, and so on. Unlike analog sensor, Digital Sensor produce discrete values (0 and 1's). Discrete values often called digital or binary signals in digital communication.

Electronic sensors or electrochemical sensors in which data conversion and data transmission take place digitally are digital sensors. These digital sensors are replacing analog sensors as they are capable of overcoming the drawbacks of analog sensors. The digital sensor consists of majorly three components such as senor, cable, and transmitter. But, In digital sensors, the signal measured directly converted into digital signal output inside the digital sensor itself. So, this digital signal transmitted through cable digitally. There are different types of digital sensors that overcome the disadvantages of analog sensors.

Then, scaler sensors basically measure scalar variables which can measure only the changes in the magnitude whereas, the vector senses not only the magnitude, but also the direction. So, scalar sensor example would be temperature sensor is an example of scalar sensor because you know irrespective of which orientation you put, the sensor temperature sensor or in which direction you are taking it, it is going to give you the magnitude value.

Then, scaler sensors basically measure scalar variables which can measure only the changes in the magnitude whereas, the vector senses not only the magnitude, but also the direction. So, scalar sensor example would be temperature sensor is an example of scalar sensor because you know irrespective of which orientation you put, the sensor temperature sensor or in which direction you are taking it, it is going to give you the magnitude value.

Only the changes in the magnitude of the temperature, on the contrary we have the vector sensor. For example, the camera sensor or the accelerometer sensor whose values are dependent on the orientation on the direction and so on direction in which the sensor is being put and the weight is measuring. Scalar sensors measure only the magnitude physical quantities, such as temperature colour, pressure, strain etcetera. These are scalar quantities and measurement of the change of magnitude is sufficient to convey the information.

On the other hand, vector sensors produce output signal of the voltage which is generally proportional to the magnitude as well as the direction and orientation of the quantity that is being measured. So, physical quantities such as the sound, image, velocity, acceleration orientation, these are all vector quantities and their measurement is not just dependent on the magnitude, but also on the direction. So, for example, accelerometer sensor, they give outputs in three dimensions x, y and z coordinate axis.

Some of the types of sensors:

1) Temperature Sensors

- Temperature sensors measure the amount of heat energy in a source, allowing them to detect temperature changes and convert these changes to data. Machinery used in manufacturing often requires environmental and device temperatures to be at specific levels. Similarly, within agriculture, soil temperature is a key factor for crop growth.

LM35 Temperature Sensor

2) Humidity Sensors

- These types of sensors measure the amount of water vapor in the atmosphere of air or other gases. Humidity sensors are commonly found in heating, vents and air conditioning (HVAC) systems in both industrial and residential domains. They can be found in many other areas including hospitals, and meteorology stations to report and predict weather.

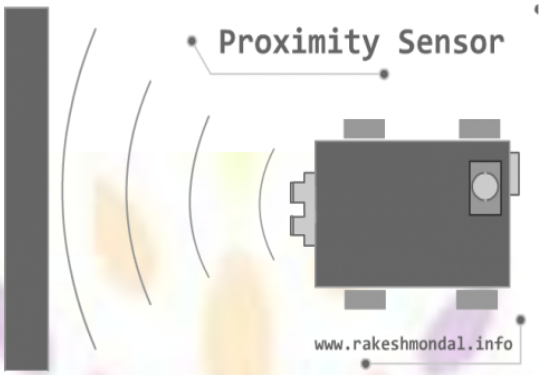
3)Pressure Sensors

1. A pressure sensor sens When the pressure changes, the sensor detects these changes, and communicates them to connected systems. Common use cases include leak testing which can be a result of decay. Pressure sensors are also useful in the manufacturing of water systems as it is easy to detect fluctuations or drops in pressure.

2. Proximity Sensors

- Proximity sensors are used for non-contact detection of objects near the sensor. These types of sensors often emit electromagnetic fields or beams of radiation such as infrared. Proximity sensors have some interesting use cases. In retail, a proximity sensor can detect the motion between a customer and a product in which he or she is interested. The user can be notified of any discounts or special offers of products located near the sensor. Proximity sensors are also used in the parking lots of malls, stadiums and airports to indicate parking availability. They can also be

used on the assembly lines of chemical, food and many other types of industries.



3. Level Sensors

- Level sensors are used to detect the level of substances including liquids, powders and granular materials. Many industries including oil manufacturing, water treatment and beverage and food manufacturing factories use level sensors. Waste management systems provide a common use case as level sensors can detect the level of waste in a garbage can or dumpster.

4. Accelerometers

- Accelerometers detect an object's acceleration i.e. the rate of change of the object's velocity with respect to time. Accelerometers can also detect changes to gravity. Use cases for accelerometers include smart pedometers and monitoring driving fleets. They can also be used as anti-theft protection alerting the system if an object that should be stationary is moved.

5. Gyroscope

- Gyroscope sensors measure the angular rate or velocity, often defined as a measurement of speed and rotation around an axis. Use cases include automotive, such as car navigation and electronic stability control (anti-skid) systems. Additional use cases include motion sensing for video games, and camera-shake detection systems.

6. Gas Sensors

- These types of sensors monitor and detect changes in air quality, including the presence of toxic, combustible or hazardous gasses. Industries using gas sensors include mining, oil and gas, chemical research and manufacturing. A common consumer use case is the familiar carbon dioxide detectors used in many homes**.**

7. Infrared Sensors

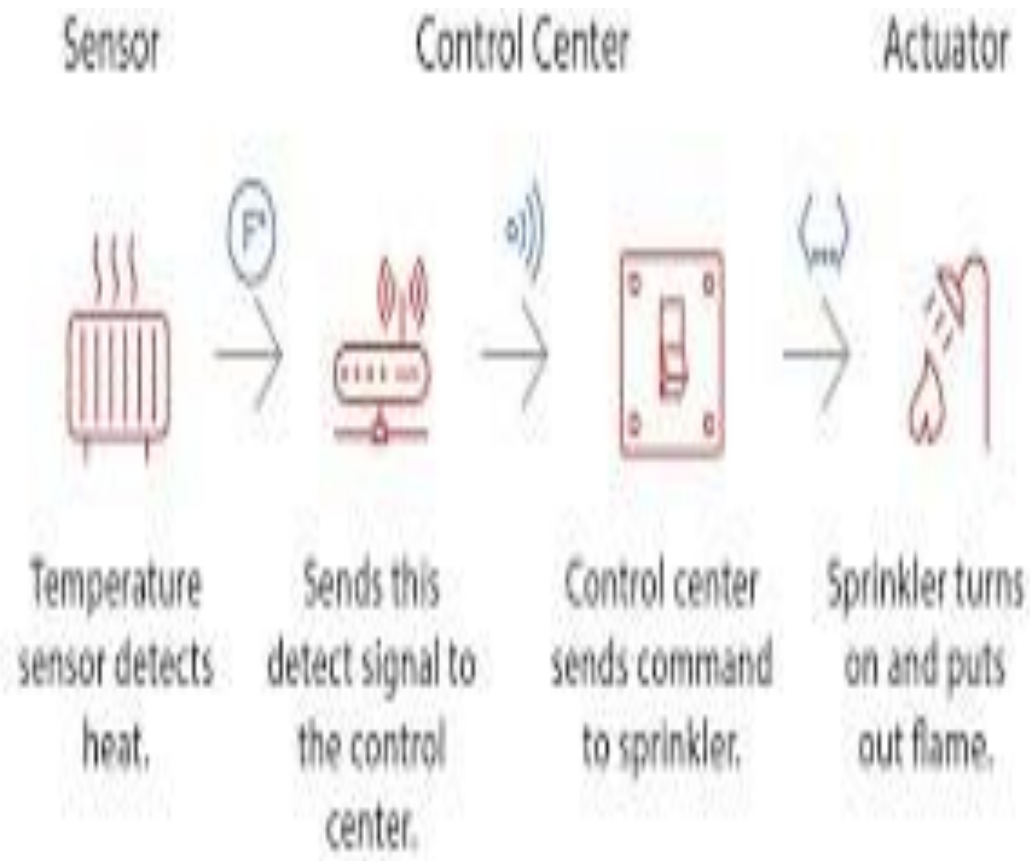
- These types of sensors sense characteristics in their surroundings by either emitting or detecting infrared radiation. They can also measure the heat emitted by objects. Infrared sensors are used in a variety of different IoT projects including healthcare as they simplify the monitoring of blood flow and blood pressure.
- Televisions use infrared sensors to interpret the signals sent from a remote control. Another interesting application is that of art historians using infrared sensors to see hidden layers in paintings to help determine whether a work of art is original or fake or has been altered by a restoration process.

8. Optical Sensors

Optical sensors convert rays of light into electrical signals. There are many applications and use cases for optical sensors. In the auto industry, vehicles use optical sensors to recognize signs, obstacles, and other things that a driver would notice when driving or parking. Optical sensors play a big role in the development of driverless cars. Optical sensors are very common in smart phones. For example, ambient light sensors can extend battery life. Optical sensors are also used in the biomedical field including breath analysis and heart- rate monitors.

9. Actuators:

- An IoT device is made up of a Physical object ("thing") + Controller ("brain") + Sensors + Actuators + Networks (Internet). An actuator is a machine component or system that moves or controls the mechanism or the system. Sensors in the device sense the environment, then control signals are generated for the actuators according to the actions needed to perform.
- A servo motor is an example of an actuator. They are linear or rotatory actuators, can move to a given specified angular or linear position. We can use servo motors for IoT applications and make the motor rotate to 90 degrees, 180 degrees, etc., as per our need.
- The following diagram shows what actuators do; the controller directs the actuator based on the sensor data to do the work.



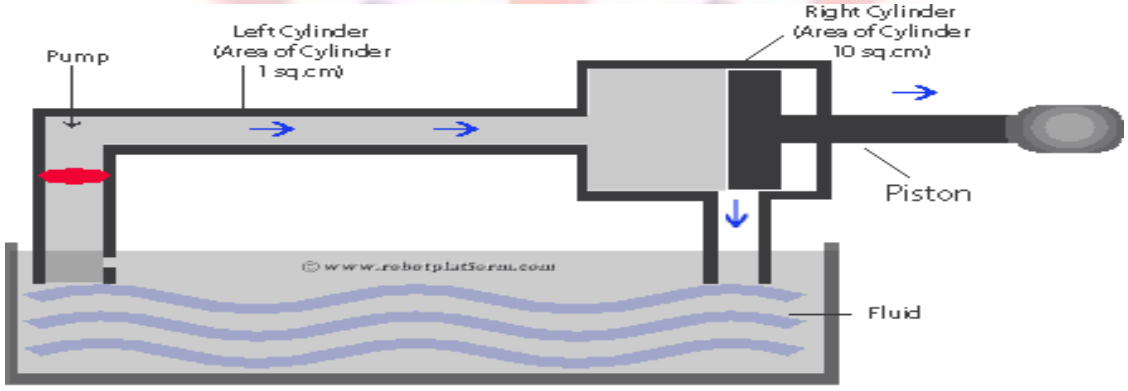
Sensor to Actuator Flow

- The control system acts upon an environment through the actuator. It requires a source of energy and a control signal. When it receives a control signal, it converts the source of energy to a mechanical operation. On this basis, on which form of energy it uses, it has different types given below.

Types of Actuators:

Hydraulic Actuators –

A hydraulic actuator uses hydraulic power to perform a mechanical operation. They are actuated by a cylinder or fluid motor. The mechanical motion is converted to rotary, linear, or oscillatory motion, according to the need of the IoT device. Example- construction equipment uses hydraulic actuators because hydraulic actuators can generate a large amount of force. So, this name suggests, these hydraulic actuators consist of a cylinder or fluid motor that uses hydraulic power to facilitate mechanical operation. The mechanical motion is converted to linear rotary or oscillatory motion. Basically when some fluid passes through, then you know that motion is converted to some linear motion or some oscillatory motion or rotary motion and since liquids are nearly impossible to compress, most of the hydraulic actuators basically exert considerable force which is the reason why liquid based actuators are typically used and these are quite popular because of this particular reason.

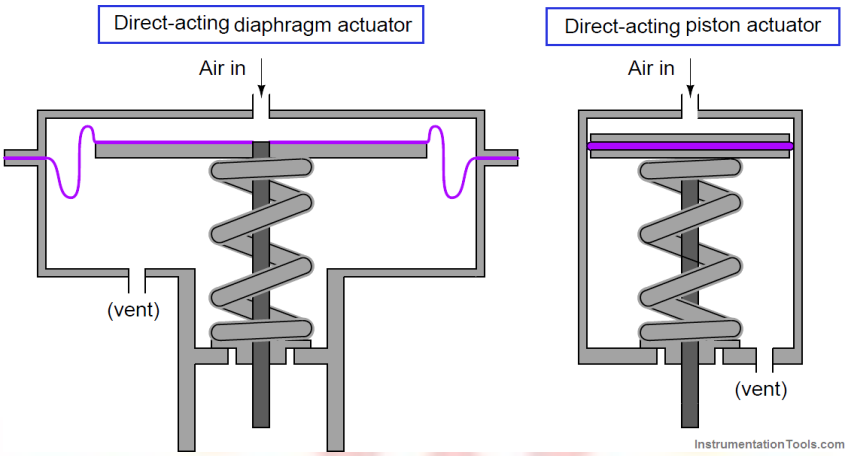


Hydraulic Actuator

Pneumatic Actuators –

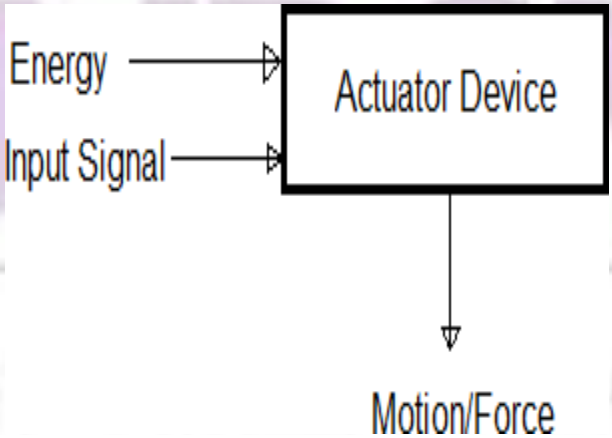
A pneumatic actuator uses energy formed by vacuum or compressed air at high pressure to convert into either linear or rotary motion. Example- Used in robotics, use sensors that work like human fingers by using compressed air. Pneumatic actuator, pneumatic means air based. A pneumatic actuator basically converts the energy formed by vacuum or compressed air at high pressure into either linear or rotatory motion. Pneumatic actuators basically exert a lot of force and for example, the pneumatic brakes can be very responsive to small changes in pressure that are applied by the driver.

Pneumatic brakes are quite common in different devices like trucks etc. They use pneumatic brakes. So, hydraulic brakes are more common in cars, in trucks pneumatic brakes are quite common. The advantage of pneumatic brakes, is that they are very responsive to small changes.



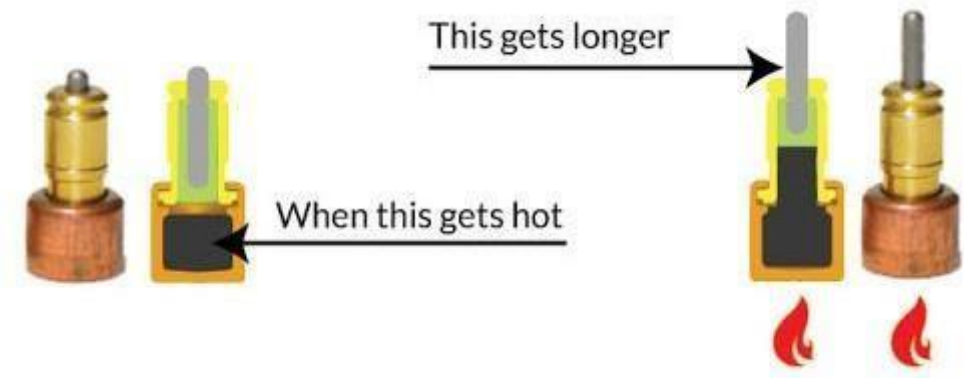
Electrical Actuators –

An electric actuator uses electrical energy, is usually actuated by a motor that converts electrical energy into mechanical torque. An example of an electric actuator is a solenoid based electric bell. An electric actuator is generally powered by a motor that converts electrical energy into mechanical torque. So, this electrical energy is used to actuate the equipment, such as the solenoid valve which control the flow of water in pipes in response to electrical signals.



Thermal /Magnetic Actuators –

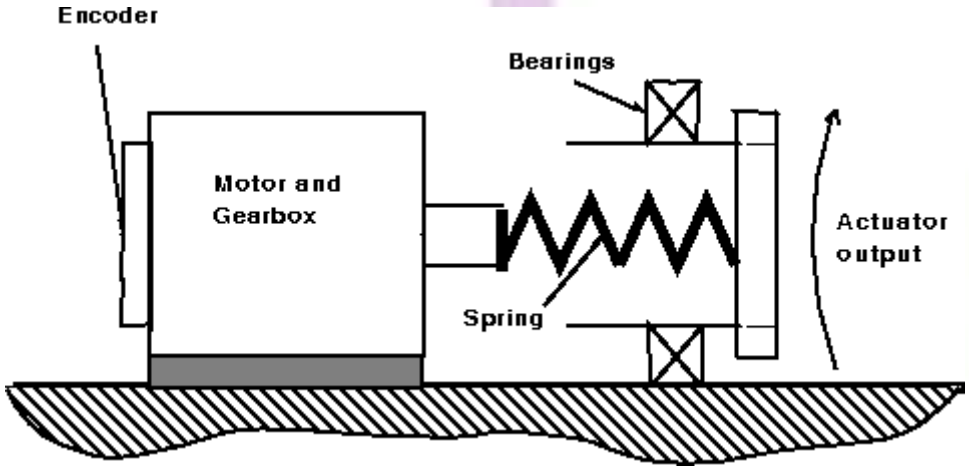
- Actuators are simply devices used to transform energy into motion. A thermal actuator is a type of non-electric motor made of components such as a piston and a thermal sensitive material capable of producing linear motion in response to temperature changes.



- Magnetic Actuators: Magnetic Actuators use magnetic effects to generate forces which impact on the motion of a part in the actuator.

Mechanical Actuators –

- A mechanical actuator executes movement by converting rotary motion into linear motion. It involves pulleys, chains, gears, rails, and other devices to operate.

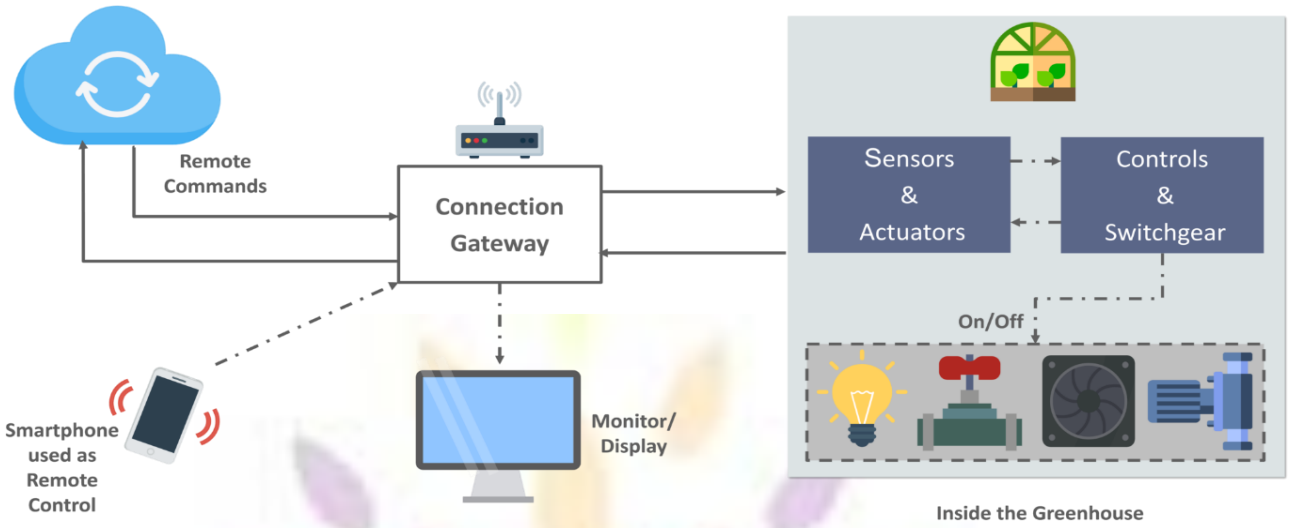


Basics of Networking in IoT:

When we talk about IoT, if we think about IoT, what we have? We have these different things which are fitted. These are basically physical objects are fitted with different sensors and these sensors basically sense different physical phenomena that are occurring around them. These sensors fitted things, sensors actuators and different other emirate devices, these are one component of the IoT, but these become different nodes in the network, these are the individual nodes in the network. These nodes they have to communicate with one another and the information that is sensed by one of these sensors fitted to these nodes, this information from the sensor and the other sensors, these are taken and are sent to the other sensor nodes, the destination nodes.

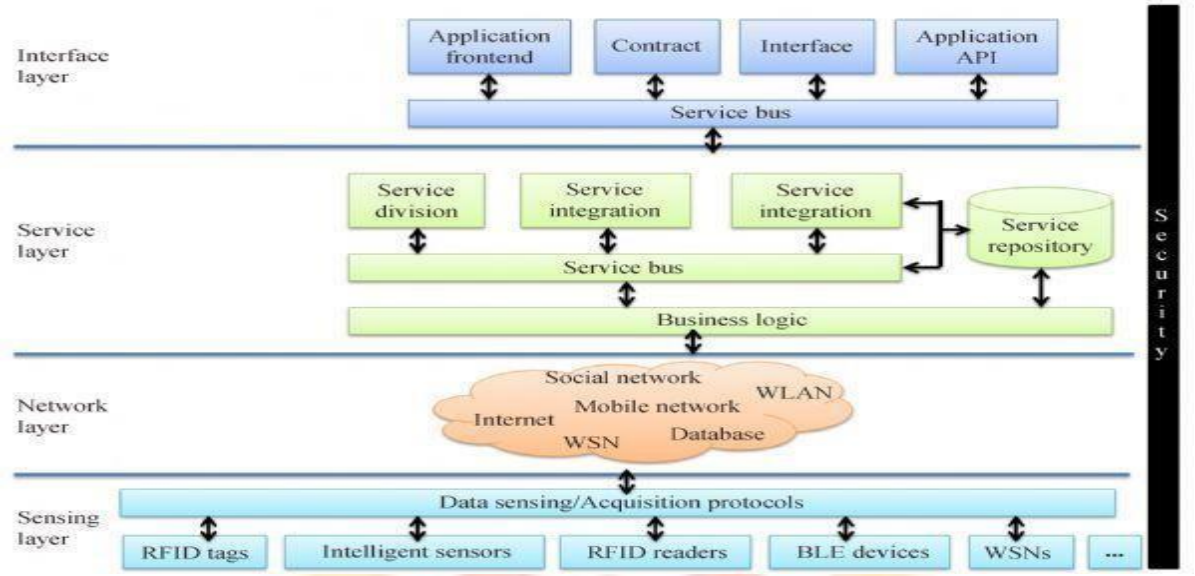
First this information has to flow through the local network and then, if the intended destination is outside this local network, then it is sent through the internet. Typically, if we are talking about an IoT which is basically internet based IoT, then it is going to flow through the internet or some other wide area network and finally, it is going to arrive at the intended destination node and may be there can be some at that point analytic engine which is running on some backend server and from these analytics, they can run on these servers decisions about actuation could be made. So from sensors to actuators through the local area network, the internet involving backend services analytics which includes some high end processing at different servers and different complex algorithms, execution of different algorithms which are based on may be machine learning neural networks and so on and so forth.

Basically an IoT is a very complex system involving sensors, actuators, networks, local area, wide area internet and different servers, different algorithms, machine learning and so on, all executing together to make the system function as one single entity. So, going back we have in this local network as you are saying then we have the internet, we have the backend services and finally, the applications that have been served. So, what we have we are these different physical objects which are fitted with different sensors. These things could be telephones, lightning systems, could be cameras, could be different other scanner, sensors like the temperature sensor and so on and these things are able to communicate with one another with the help of wireless technologies like Zigbee, Bluetooth, WiFi and so on. So, as you can see that this wireless basically helps these different devices to talk to one another and this information from these devices will go through a local network and from a local network, they will go through the internet to the backend services involving different server's processors and so on and so forth. For running different analytics and then based on that, different devices can be actuated you know may be a pump.



So, above is a figure which shows that we have different sensors, processors and Phone. It refitted to each of these devices or the sensor nodes or the sensor motes or the IoT motes as you may want to call them. So, these motes, they talk to one another, but these different sensor nodes, they are basically within the jurisdiction or the domain of the gateway. So, the gateway is basically tasked to assign different locally unique addresses to these different nodes, to these different IoT nodes and the gateway basically takes care of the local addressing within that particular local area network.

Service oriented Architecture:



Now, let us look at the service orientation, the service oriented architecture of IoT. So, in the IoT we have these different layers, the sensing layer, the network layer, the service layer and the interface layer.

So, we have four different layers and as the name suggests, sensing layer basically takes care of sensing through different RFID tags sensors, RFID readers and so forth and then, data are sensed and are acquired and so on are sent to the next layer higher up which is the network layer. The network layer basically serves sensor networks, social networks and different other networks like Internet WLAN, Wireless Sensor networks (WSN) and data bases internet. Then, we have the service layer which deals mostly with the service delivery such as service, division service, integration service, repository service, business logic and so on. So, all these different things involved with the offering of the services to support the business functions. A service repository is a catalogue in which you can see what services are available on a network. Instead of keeping track of services manually, you can register the services in a service repository. To find out what services are available on the network you query the service repository. This can be done via a standardized interface, meaning computerized clients can do this. It does not have to be done by a human. Service Bus allows you to group operations against multiple messaging entities within the scope of a single transaction. A message entity can be a queue, topic, or subscription. Service Integration and Management (SIAM) is an approach to managing multiple suppliers of services (business services as well as information technology services) and integrating them to provide a single business-facing IT organization. It aims at seamlessly integrating interdependent services from various internal and external service providers into end-to-end services in order to meet business requirements.

Then, we have the interface layer, we have the application frontend, we have
The consideration that has to be taken into account while building the IoT systems the complexity of the networks. If the number of nodes in the network increases, then in the solution the system is going to be sustainable or whether it can be scaled up or not, then we have interference among the different devices. This is very much vital in any network
a contract interface and application APIS. The application frontend of a software program or website is everything with which the user interacts. From a user standpoint, the frontend is

synonymous with the user interface. From a developer standpoint, it is the interface design and the programming that makes the interface function.

Complexity of Networks:

- The consideration that has to be taken into account while building the IoT systems the complexity of the networks. If the number of nodes in the network increases, then in the solution the system is going to be sustainable or whether it can be scaled up or not, then we have interference among the different devices. This is very much vital in any network.
- Interference is a crucial issue and particularly IoT networks involve lot of large number of typically densely deployed nodes and these nodes as you know typically wireless power by wifi or bluetooth or zigbee and so on. So, interference between these different communication between these different nodes that at the corresponding radios and so on is possible. So, how do you handle it? Network management, involving computation management, involving communication management, involving service management and infrastructure management and so on.
- Addressing Issues is incredibly high number of nodes each of which will produce content that should be retrievable by any authorized user. This requires effective addressing policies IPv4 protocol may already reached its limit. IPv6 addressing has been proposed for low-power wireless communication nodes within the 6LoWPA context IPv6 addresses are expressed by means of 128 bits which is addresses, enough to identify objects worth to be addressed.

So, network management as a whole then heterogeneity in the networks. Heterogeneity in terms of the device's standards the protocols, the algorithms and so on. So, how do you handle because IoT devices unlike traditional internet, IoT networks come in different, you know come from different vendors, different devices coming from different vendors, different devices using different algorithms, these different protocols being used and so on. So, all these basically invite dealing with the issue of heterogeneity and lot of heterogeneity is involved. How heterogeneity is taken care of and protocol organization and standardization within the network, how the different protocols can be standardized, so that a device running one protocol can talk to another device and so on.

IoT Protocols/Communication protocols:

A) Link Layer:

Protocols determine how data is physically sent over the network's physical layer or medium. Local network connect to which host is attached. Hosts on

the same link exchange data packets over the link layer using link layer protocols. Link layer determines how packets are coded and signalled by the h/w device over the medium to which the host is attached.

Protocols:

- 802.3-Ethernet: IEEE802.3 is collection of wired Ethernet standards for the link layer. Eg: 802.3 uses co-axial cable; 802.3i uses copper twisted pair connection; 802.3j uses fiber optic connection; 802.3ae uses Ethernet over fiber.
- 802.11-WiFi: IEEE802.11 is a collection of wireless LAN (WLAN) communication standards including extensive description of link layer. Eg: 802.11a operates in 5GHz band, 802.11b and 802.11g operates in 2.4GHz band, 802.11n operates in 2.4/5GHz band, 802.11ac operates in 5GHz band, 802.11ad operates in 60Ghzband.
- 802.16 - WiMAX: IEEE802.16 is a collection of wireless broadband standards including exclusive description of link layer. WiMAX provide data rates from 1.5 Mb/s to 1Gb/s.
- 802.15.4-LR-WPAN: IEEE802.15.4 is a collection of standards for low-rate wireless personal area network (LR-WPAN). Basis for high level communication protocols such as ZigBee. Provides data rate from 40kb/s to250kb/s. So, this particular standard is helpful in environments which are noise prone and have lot of interferences and in a presence of noise and interference, this particular standard can help in improving the reliability of the network.
- 2G/3G/4G-Mobile Communication: Data rates from 9.6kb/s(2G) to up to100Mb/s(4G). B)

b) Network/Internet Layer:

Responsible for sending IP datagrams from source n/w to destination n/w. Performs the host addressing and packet routing. Datagrams contains source and destination address.

Protocols:

- IPv4: Internet protocol version 4 (IPV4) is the most deployed Internet protocol that is used to identify the devices on a network using hierarchical addressing scheme. 32bit address. Allows total of 2*32addresses.
- IPv6: Internet Protocol version6 uses 128bit address scheme and allows 2*128addresses.
- 6LOWPAN:(IPv6over Low power Wireless Personal Area Network) operates in 2.4 GHz frequency range and data transfer 250 kb/s. it allows for the smallest devices and each of these devices having limited processing ability to transmit information wirelessly over the internet protocol. So, we have low power small devices limited processing capability as is typical of IoT systems and wireless communication being present. So, it basically helps in establishing connectivity in this kind of networks.

C) **Transport Layer**:

Provides end-to-end message transfer capability independent of the underlying n/w. Set up on connection with ACK as in TCP and without ACK as in UDP. Provides functions such as error control, segmentation, flow control and congestion control.

Protocols:

- TCP: Transmission Control Protocol used by web browsers (along with HTTP and HTTPS), email (along with SMTP, FTP). Connection oriented and stateless protocol. IP Protocol deals with sending packets, TCP ensures reliable transmission of protocols in order. Avoids n/w congestion and congestion collapse.
- UDP: User Datagram Protocol is connectionless protocol. Useful in time sensitive applications, very small data units to exchange. Transaction oriented and stateless protocol. Does not provide guaranteed delivery.

D) **Application Layer**: Defines how the applications interface with lower layer protocols to send data over the n/w. Enables process-to-process communication using ports.

Protocols:

- HTTP: Hyper Text Transfer Protocol that forms foundation of WWW. Follow request response model Stateless protocol. HTTP follows a request – response model where a client sends request to a server using the HTTP commands. An HTTP can be a browser or an application running on the client.
- CoAP: Constrained Application Protocol for machine-to-machine(M2M) applications with constrained devices, constrained environment and constrained network. Uses client-server architecture. This protocol is particularly used for web transfer and by web transfer very similar to the HTTP, but web transfer in the context of constrained networks resource, constrained networks with nodes which are constrained with respect to different resources, such as limited energy or power supply, limited computational resources, limited communication resource, limited bandwidth environment and so on.

- WebSocket: Allows full duplex communication over a single socket connection. They allow bi-directional between clients and servers. WebSocket follow the exclusive pair communicational model. WebSocket communication begins with a connection setup request sent. WebSocket communication presents a suitable protocol for the IoT environment where bundles of data are transmitted continuously within multiple devices. A WebSocket makes server and device communication easy. A server needs a WebSocket library to be installed and we need to have the WebSocket client and web browser installed on the client or device that supports WebSocket.
- MQTT: Message Queue Telemetry Transport is light weight messaging protocol based on publish-subscribe model. Uses client server architecture. Well suited for constrained environment. So, in MQTT there are three concepts that are involved. The first we are going to go through is the concept of a message broker the concept of a message broker that basically serves like a broker which takes control of publishing of the messages and subscription of the messages. So, publish subscribe is basically controlled by the message broker. The data are sent to the clients by the message broker, this data are distributed by the message broker to the clients who have subscribed to the services.

- XMPP: Extensible Message and Presence Protocol for real time communication and streaming XML data between network entities. Support client- server and server-server communication. So, it is a message oriented middleware that is based on XML, whereas XML is particularly used for unstructured data. XMPP is useful for real time exchange of structured data and it is an open standard protocol. XMPP uses a client server architecture, it uses a decentralized model meaning that there is no server that is involved in the message transfer and it provides facilities for discovery of messages which are residing locally or globally across the network and the availability information of these services. some of these highlights of the XMPP protocol, it is based on the concept of decentralization where there is no central server and then, you know everybody can run the XMPP server theoretically and it is based on open standard. So, there is no involvement of royalties or granting permissions to implement the XMPP specifications, different security features that the standard ones, such as authentication, encryption,
- DDS: Data Distribution Service is data centric middleware standards for device-to-device or machine-to-machine communication. Uses publish- subscribe model. Designed and summarized by Object Management Group (OMG), Data Distribution Service (DDS) is a competent IoT protocol for scalable, real-time M2M communication. This protocol leverages multicasting techniques in the transmission of data and high-quality QoS in the small memory footprint devices and to applications. It employs Data- Centric Public-Subscribe (DCPS) layer to directly communicate the information from publishers to subscribers and deliver reliable, scalable performance in embedded systems.
- AMQP: Advanced Message Queuing Protocol is open application layer protocol for business messaging. Supports both point-to-point and publish-subscribe model. So, this standard basically helps define how mists are going to be passed from businesses, business applications or organizations. So, in other words, a particular business is comprised of different systems and different processes, business processes. So, a business can be conceived as a collection of different systems and business processes. So, this particular standard helps in communicating between these systems connecting rather connecting between these different systems and the business processes of that particular business.

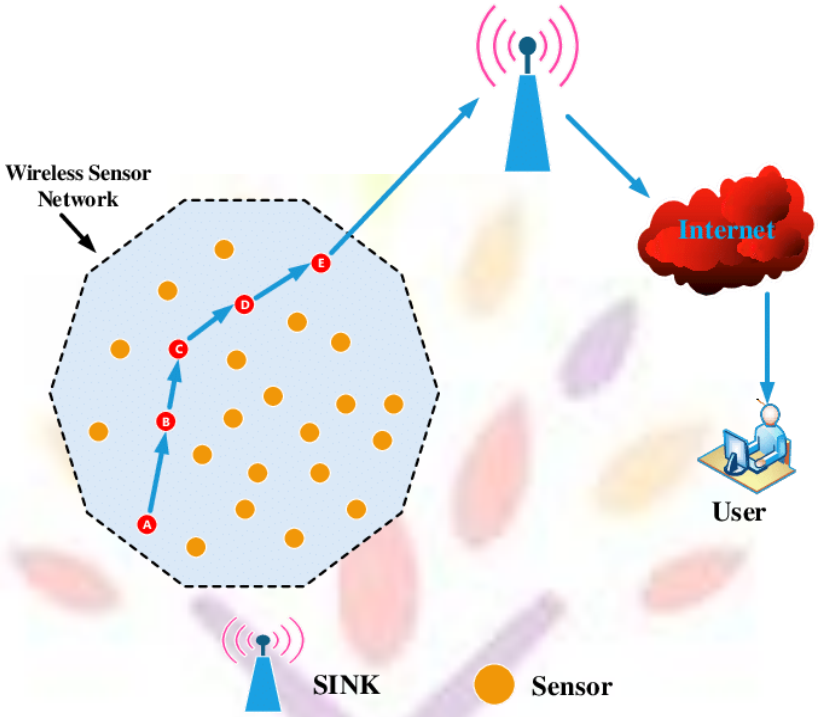
Sensor Network:

Sensor network is a very important technology that is used for building IoT. Sensors, transducers, actuators these are all very important things for realization of IoT systems. But, when we talk about sensors when we talk about actuators, these are the things that we have already gone through in one of the previous topics. So, these are stand-alone devices that we talk about, but if we can have these sensors connect with one an another we can derive important information continuously, in real time remotely, from a larger terrain. So, in sensor networks what we have? In sensor networks we have individual sensors, which are embedded in something known as sensor devices or sensor nodes. So, these nodes or devices they have one of their components which is the sensor, and they have other components as well. So, these components taken together they comprise that particular node or the device which can help them to communicate. And one device communicates with another device, that device communicates

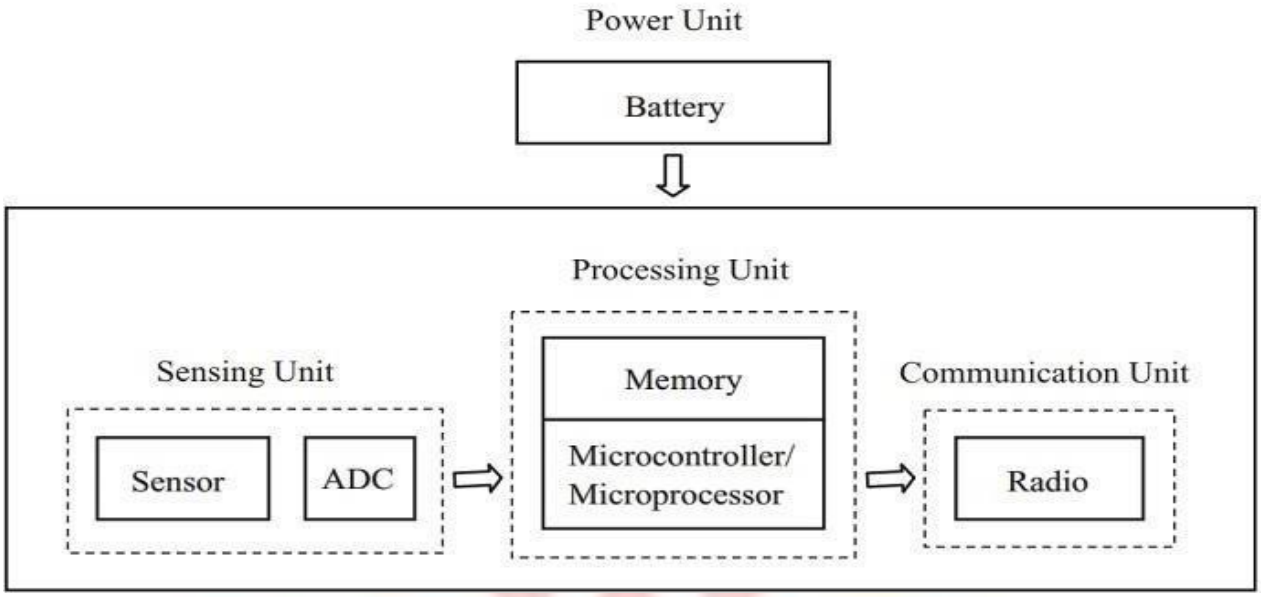
with another device, the third device with a fourth, fourth with the first and so on. We can expand the sensing by having them communicate with one another. So, what we have are different types of topologies. We can have all sorts of topologies that we have already heard of in networks being implemented in the case of sensor networks as well we can use a star topology. We can use a mesh topology we can have a mesh of we can have a mesh of sensor nodes that are all put together.

Let us look at the different basics of sensor networks. So, in a sensor network we have sensor nodes. Every sensor node has a sensing unit. The sensing unit basically senses. Senses what? Senses the particular physical phenomena that it is supposed to sense. A temperature sensor would be sensing the temperature fluctuations. A humidity sensor will be sensing the humidity fluctuations. A camera sensor would be sensing; that means, taking the images of what is around you know what is happening around it. A vibration sensor will be sensing the vibration. A light sensor will be looking at will be sensing the illumination conditions and so on. But each of them is sensing locally and, every other node that is deployed they are all doing their own tasks separately individually. And now, in a sensor network we all have to we have to put them all together; we have to put them all together how it is possible to put them together? We have to just have some kind of radio connectivity between these different devices. These devices mean their sensor nodes and, this is how we build up a sensor network. And what is the main motivation of building a sensor network? To have greater coverage of sensing and, continuously we can monitor in real time we can monitor remotely we can monitor what is going on in a particular terrain, without actually having somebody humanly sitting and monitoring that particular region or space.

This is how the entire sensor network concept functions. So, what we have is multi hop communication. So, let us consider a stationary sensor network. Stationary means what? Stationary means that, the node all these nodes, when they are deployed, they will maintain the position at subsequent instants of time after deployment. So, they will all maintain their own respective positions. And they will not move they are all stationary the nodes are all stationary. So, this is an example of a stationary sensor network. Mobile sensor network on the contrary, have the sensors and the sensor nodes that move around. For example, the sensors that are fitted.



So, if we look at in a sensor network, what we have? In a sensor network we have different units. WSN have different units the first thing that is required is to have some kind of sensing unit. Then we have some kind of a processing unit. Third is we need some kind of communication to take place between these different nodes and communication unit which include transceiver. Then we have different other units such as, the analog digital converter, we have the power unit, we have, power unit will do what? Are in it includes things such as battery and so on, which is going to power these devices. So, these are the different units. And then we have other optional units optional units such as the location finding systems for example, GPS etc.



So, this is how a sensor node looks like they have a processor, a communication unit, analog digital converter, we have a power unit, a sensing unit, a processing unit, it is already given and so on and so forth. This is how a sensor node looks like. So, if you recall in one of the previous topic when we talked about sensors, we also talked about the different types of sensors. We have to develop something like a sensor board. So, this becomes a sensor board or a sensor node or a sensor device. This sensor board is the hardware which has different components.

In our agricultural field we have sensor nodes that are deployed and these are all solar powered. The power unit basically has a solar energy through the solar panel and powers the rest of the components of the node. The sensor nodes are multifunctional. So, they can be used for different purposes. In fact, the sensor board can change different sensors and you can use the sensor board with the different components of it, typically you know it will work for serving different types of applications. So, the number of sensor nodes that are used depends on the application type. The sensor nodes they have short communication range, may be powered by zigbee. A zigbee has very short communication range. So, this short communication range means that the sensed information by a particular node can be sent only up to couple of meters or tens of meters. And after that what? Finally, it has to be sent to the sink node. So, after that what? After that within that communication range of let us say thirty meters if it is zigbee up to above thirty meters then we need to have another node which again has to relay that information that has been received from this particular node. This is how we have this multi hop communication taking place between these different nodes in the network. Multi hop, multi hop means what? A particular node has to send something to a remote destination, but it is not within the direct communication range of it.

Constraints on Sensor Networks:

- Small size maybe less than cubic cm.
- Must consume extremely low power.
- Operate in an unattended manner in highly dense area.
- Should have low production cost and be dispensable.
- Be autonomous.
- Be adaptive to the environment.

Applications:

- Temperature measurement
- Humidity measurement
- Lightening condition
- Soil makeup
- Air pressure
- Noise level
- Vibration

Unit II

M2M Communication

Machine-to-machine communication, or M2M, is exactly as it sounds: two machines "communicating," or exchanging data, without human interfacing or interaction. This includes serial connection, powerline connection (PLC), or wireless communications in the industrial Internet of Things (IoT). Switching over to wireless has made M2M communication much easier and enabled more applications to be connected. In general, when someone says M2M communication, they often are referring to cellular communication for embedded devices.

Examples of M2M communication in this case would be vending machines sending out inventory information or ATM machines getting authorization to dispense cash. As businesses have realized the value of M2M, it has taken on a new name: The Internet of Things (IoT). IoT and M2M have similar promises: to fundamentally change the way the world operates. Just like IoT, M2M allows virtually any sensor to communicate, which opens up the possibility of systems monitoring themselves and automatically responding to changes in the environment, with a much reduced need for human involvement. M2M and IoT are almost synonymous—the exception is IoT (the newer term) typically refers to wireless communications, whereas M2M can refer to any two machines— wired or wireless—communicating with one another.

Traditionally, M2M focused on "industrial telematics," which is a fancy way of explaining data transfer for some commercial benefit. But many original uses of M2M still stand today, like smart meters. Wireless M2M has been dominated by cellular since it came out in the mid-2000's with 2G cell networks. Because of this, the cellular market has tried to brand M2M as an inherently cellular thing by offering M2M data plans. But cellular M2M is only one subsection of the market, and it shouldn't be thought of as a cellular-only area.

How M2M Works

As previously stated, machine-to-machine communication makes the Internet of Things possible. According to Forbes, M2M is among the fastest- growing types of connected device technologies in the market right now, largely because M2M technologies can connect millions of devices within a single network. The range of connected devices includes anything from vending machines to medical equipment to vehicles to buildings. Virtually anything that houses sensor or control technology can be connected to some sort of wireless network.

This sounds complex, but the driving thought behind the idea is quite simple. Essentially, M2M networks are very similar to LAN or WAN networks, but are exclusively used to allow machines, sensors, and controls, to communicate. These devices feed information they collect back to other devices in the network. This process allows a human (or an intelligent control unit) to assess what is going on across the whole network and issue appropriate instructions to member devices.

M2M Applications

The possibilities in the realm of M2M can be seen in four major use cases, which we've detailed below:

1. MANUFACTURING

Every manufacturing environment—whether it's food processing or general product manufacturing—relies on technology to ensure costs are managed properly and processes are executed efficiently. Automating manufacturing processes within such a fast-paced environment is expected to improve processes even more. In the manufacturing world, this could involve highly automated equipment maintenance and safety procedures.

For example, M2M tools allow business owners to be alerted on their smartphones when an important piece of equipment needs servicing, so they can address issues as quickly as they arise. Sophisticated networks of sensors connected to the Internet could even order replacement parts automatically.

2. HOME APPLIANCES

IoT already affects home appliance connectivity through platforms like Nest. However, M2M is expected to take home-based IoT to the next level. Manufacturers like LG and Samsung are already slowly unveiling smart home appliances to help ensure a higher quality of life for occupants.

For example, an M2M-capable washing machine could send alerts to the owners' smart devices once it finishes washing or drying, and a smart refrigerator could automatically order groceries from Amazon once its inventory is depleted. There are many more examples of home automation that can potentially improve quality of life for residents, including systems that allow members of the household to remotely control HVAC systems using their mobile devices. In situations where a homeowner decides to leave work early, he or she could contact the home heating system before leaving work to make sure the temperature at home will be comfortable upon arrival.

3. HEALTHCARE DEVICE MANAGEMENT

One of the biggest opportunities for M2M technology is in the realm of health care. With M2M technology, hospitals can automate processes to ensure the highest levels of treatment. Using devices that can react faster than a human healthcare professional in an emergency situation make this possible. For instance, when a patient's vital signs drop below normal, an M2M-connected life support device could automatically administer oxygen and additional care until a healthcare professional arrives on the scene. M2M also allows patients to be monitored in their own homes instead of in hospitals or care centers.

For example, devices that track a frail or elderly person's normal movement can detect when he or she has had a fall and alert a healthcare worker to the situation.

4. SMART UTILITY MANAGEMENT

In the new age of energy efficiency, automation will quickly become the new normal. As energy companies look for new ways to automate the metering process, M2M comes to the rescue, helping energy companies automatically gather energy consumption data, so they can accurately bill customers. Smart meters can track how much energy a household or business uses and automatically alert the energy company, which supplants sending out an employee to read the meter or requiring the customer to provide a reading. This is even more important as utilities move toward more dynamic pricing models, charging consumers more for energy usage during peak times. A few key analysts predict that soon, every object or device will need to be able to connect to the cloud. This is a bold but seemingly accurate statement. As more consumers, users, and business owners demand deeper connectivity, technology will need to be continually equipped to meet the needs and challenges of tomorrow. This will empower a wide range of highly automated processes, from equipment repairs and firmware upgrades to system diagnostics, data retrieval, and analysis. Information will be delivered to users, engineers, data scientists, and key decision-makers in real time, and it will eliminate the need for guesswork.

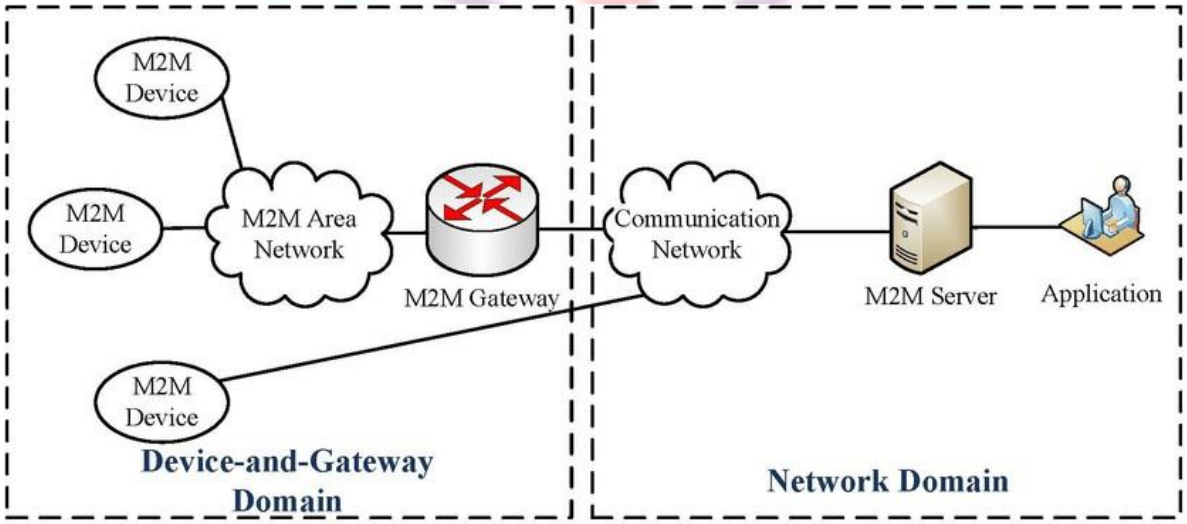
There are different M2M applications, environment monitoring, civil protection and public safety, supply chain management, energy and utility distribution as in smart grid, smart grid separately common. we have intelligent transportation systems, healthcare, automation of buildings, military applications, agriculture, home networks all these are different applications of M2M.

M2M features:

- Large number of nodes or devices
- Low cost
- Energy efficient
- Small traffic per device/machine
- M2M communication free from human intervention

General Architecture of M2M Systems:

- M2M device connects to the network domain via direct connectivity or M2M gateway. In the first case, the M2M device connects to the network domain via the access network, which performs the procedures such as registration, authentication, authorization, management, and provisioning with the network domain. In the second case, the M2M device connects to the M2M gateway using the M2M area network.
- M2M area network provides connectivity between M2M devices and M2M gateways.
- M2M gateway acts as a proxy between M2M devices and the network domain. As an example, an M2M gateway can run an application that collects and treats various information (e.g., contextual parameters) from sensors and meters.
- M2M communication network provides connection between the M2M gateways/devices and the M2M servers. Usually it contains two parts: the access network and the Internet.
- M2M server works as a middleware layer to pass data through various application services.



Difference Between IoT and M2M:

M2M, or machine-to-machine, is a direct communication between devices using wired or wireless communication channels. M2M refers to the interaction of two or more devices/machines that are connected to each other. These devices capture data and share with other connected devices,creating an intelligent network of things or systems. Devices could be sensors, actuators, embedded systems or other connected elements.

M2M technology could be present in our homes, offices, shopping malls and other places. Controlling electrical appliances like bulbs and fans using RF or Bluetooth from your smartphone is a simple example of M2M applications at

home. Here, the electrical appliance and your smartphone are the two machines interacting with each other.

The Internet of Things (IoT) is the network of physical devices embedded with sensors, software and electronics, enabling these devices to communicate with each other and exchange data over a computer network. The things in the IoT refer to hardware devices uniquely identifiable through a network platform within the Internet infrastructure.

M2M versus the IoT

M2M	IoT
M2M is about direct communication between machines.	The IoT is about sensors automation and Internet platform.
It supports point-to-point communication.	It supports cloud communication.
Devices do not necessarily rely on an Internet connection.	Devices rely on an Internet connection.
M2M is mostly hardware-based technology.	The IoT is both hardware- and software-based technology.
Machines normally communicate with a single machine at a time.	Many users can access at one time over the Internet.
A device can be connected through mobile or other network.	Data delivery depends on the Internet protocol (IP) network.

Some more differences like:

Communication Protocols:

• M2M and IoT can differ in how the communication between the machines ordevices happens.

• M2M uses either proprietary or non-IP based communication protocols for communication within the M2M area networks. IoT uses IP bases communication protococls.

Machines in M2M vs Things in IoT:

• The "Things" in IoT refers to physical objects that have unique identifiers and can sense and communicate with their external environment (and user applications) or their internal physical states.

• M2M systems, in contrast to IoT, typically have homogeneous machine types within an M2M area network.

Hardware vs Software Emphasis:

• While the emphasis of M2M is more on hardware with embedded modules, the emphasis of IoT is more on software.

Data Collection & Analysis:

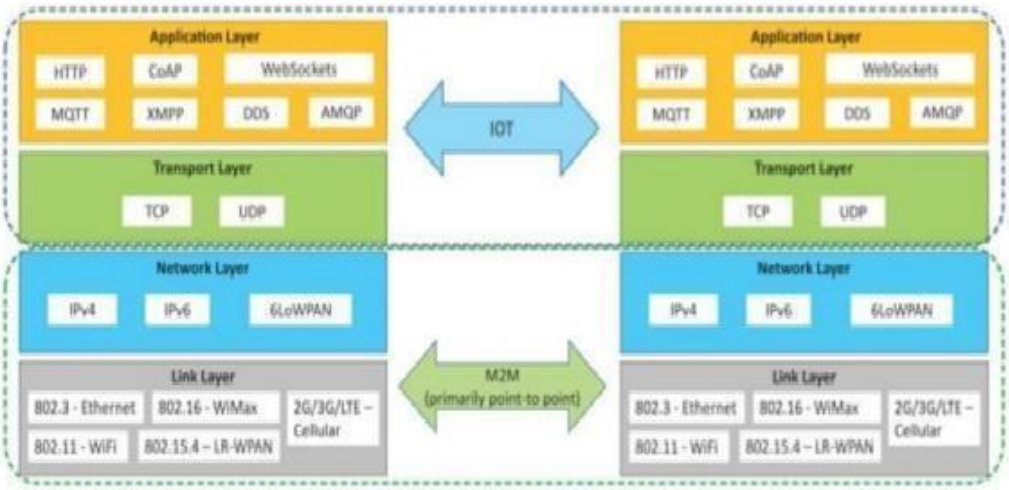
• M2M data is collected in point solutions and often in on-premises storage infrastructure.
• In contrast to M2M, the data in IoT is collected in the cloud (can be public, private or hybrid cloud).

Applications:

• M2M data is collected in point solutions and can be accessed by on premises applications such as diagnosis applications, service management applications, and on- premises enterprise applications.

• IoT data is collected in the cloud and can be accessed by cloud applications such as analytics applications, enterprise applications, remote diagnosis and management applications, etc.

- Internet of things uses different types of devices.
- These devices are made by different vendors following different specifications; there is no one standard for IOT.
- So, consequently what happens is for different things the different IoT devices are made by different vendors following different specifications.
- Again these different devices by different vendors they follow different protocols not necessarily that they all follow the same protocol.
- Even the kind of users their user profiles these can also be different.
- So, there is so much of diversity that is inherent to these systems IoT systems and that is why it is very important to address this particular issue.



Interoperability in Internet of Things:

Internet of things uses different types of devices. These devices are made by different vendors following different specifications; there is no one standard for IOT. So, consequently what happens is for different things the different IoT devices are made by different vendors following different specifications. Again these different devices by different vendors they follow different protocols not necessarily that they all follow the same protocol. Even the kind of users their user profiles these can also be different. So, there is so much of diversity that is inherent to these systems IoT systems and that is why it is very important to address this particular issue.

In internet of things one of the core problems or issues that has been studied quite extensively is heterogeneity of devices, protocols, user groups and many other heterogeneity aspects in from different other angles. So, this has been studied quite extensively. And one of the requirements to handle this heterogeneity issue is basically to have some kind of Interoperability between these different heterogeneous aspects. Interoperability means that one particular device is following a particular protocol; another device follows another protocol. So, how do they talk to each other, this is one aspect. Similarly, at different physical levels, different specifications, different devices how do they talk to each other, they all have been made in different ways because there is no one standard that has been followed in developing these systems. So, when you want to build a singular IoT system comprising of all these different heterogeneous objects, devices, protocols, standards etc you need to have some kind of handshaking. And that handshaking is where these protocols have been devised which can be some kind of a middleman a middleware rather which can help these two different diverse groups to be able to talk to each other.

So, when we talk about IoT we are talking about large scale networks. Large scale networks requiring the use of large number of different devices millions and billions of devices, these devices are distributed all across in the internet or all across in the world. And what is required is to have some kind of cooperation between the different devices, some kind of coordination mechanism to be enforced between these different devices to be able to talk to each other. So, this is one issue. Second issue is that the devices they all have been designed with different specifications heterogeneous in all respects in the physical device level, in the protocol level, user level, so in all different aspects. So, heterogeneous IoT devices and their subnets is a challenge that has to be worked on in the context of internet of things. Another very typical concern that is specific of IoT is the device configuration. Typically these IoT devices their configuration is unknown all across. The different configuration modes for IoT devices come from unknown owners and inherently that brings in lot of complexity and that has to be handled. Another very interesting complexity is how do you handle differences in semantics. So, different processing logics are applied to the same IoT network devices or applications by different developers, different user groups and so on. So, how do you handle these differences this conflict in the semantics.

Interoperability is a characteristic of a product or system whose interfaces are completely understood to work with other products or systems present or future in either implementation or access without any restrictions. So, that there is exchange of data, exchange of services and to the user it should the interoperability has to be handled in such a manner that to the user, the user should feel that he or she is getting access to the services of the IoT system in a seamless manner. The user should not have to get into how these are implemented what is the translation that is going so on and so forth. Heterogeneity - different communication protocols ZigBee following IEEE802.15.4. Bluetooth following 802.15.1, GPRS, 6LowPAN, Wi-Fi which follows 802.11 standard. So, all these different types of standards all different types of communication protocols handshaking with each other, communicating with each other. Different wired communication protocols such as 802.3 and 802.1 talking to each other that is required because otherwise we cannot have this seamless you know anytime, anywhere, any device connectivity that is not going to be possible. So, so much of heterogeneity is going to be there all across. Then different programming languages are used in different computing systems for example, JavaScript in one, Java in another, C, C++, Visual Basic, PHP, python, so many different programming languages platforms of different kinds are used. Again there are different hardware platforms as well not just programming platforms in the hardware platforms such as crossbow based products talking to national instrument products talking to Cisco products and so on. So, all these hardware platforms can also be varying they you know different

types of heterogeneous hardware platforms. So, interoperability is very much required in this kind of backdrop.

Syntactic Interoperability:

Syntactic interoperability for device interaction the interoperability between devices and device user in terms of message formats is what is the concern of this type of interoperability. The message format from a device to a user is understandable for the user's computer. On the other hand the message format from the user to the device is executable by the device. middleware technology is sort of like a software middleware bridge, which dynamically maps the physical devices with the different domains and based on the map the devices can be discovered and controlled remotely. Then we have the cross context syntactic interoperability, which concerns collaborative concept exchange and using XML syntax.

Semantic interoperability

Semantic interoperability for device interaction; and here we are talking about the semantics and the exchange of the semantics. So, the messages that are sent between these different devices whether they are understood by the respective party if not there has to be some middleware in between which has to make it happen. So, the device can understand the meaning of users instructions that is sent from the user to the device, similarly the user can understand the meaning of devices response sent from the device.

Let us say that even before we start with the device interoperability, let us say that we have you know two devices: device A and device B. And in our example let us assume that A and B as such do not follow the same specific same protocol, they did not have the same specifications at the physical layer or you know the other layers. So, no common protocol is available across all these different layers. So, how do they communicate how do they communicate. So, this is the problem of device interoperability. So, how do they communicate. So, let us assume initially that we have some kind of a middleware some kind of a middleware, which will understand the language of A and the language of B. So, it will understand the language of A as well as the language of B. So, this will help this one to be able to translate what A is saying A wants to send, he wants to communicate, and similarly what B is say. So, this sort of approach not only can be used for device interoperability, but also other forms of interoperability as well. So, this one basically becomes a translation device translation unit. So, if we are talking about two different protocols we can call it as a protocol translation unit - PTU. So, this protocol translation unit will translate the protocols separate protocols or the languages that are followed by both, ok. So, this is an approach that can be adopted and can be extended for similar other situations as well.

So, we talked about in device interoperability we talked about some kind of a universal middleware bridge (UMB) which solves seamless interoperability problems caused by heterogeneity of several kinds of home network middleware. So, this bridge is basically it is a middleware that creates a virtual map among the physical devices of all middleware home networks. And it creates a compatibility among these middleware home networks. So, it is basically some middleware- based solution this middleware will act as an agent for this kind of translation or handshaking between two different heterogeneous devices

Introduction to Arduino:

We have seen that there are different types of sensors, different types of sensing possible, different types of actuators that principles behind different types of sensing, different types of actuation. We have seen that there are different types of networks that are possible for use for adoption for use in IoT, different types of communication devices standards can also be used for communicating in internet of things. So, having understood those how can we use these concepts for building a real internet of things may be in a smart phone scenario at home to improve some of the you know daily tasks that we do at home. Or in a smart homes smart cities scenario like in a smart hospital smart you know smart transportation connected vehicles and so on. So, of are all these we need to take help of different IoT devices. And one of the very popular once is Arduino.

Arduino is a simple and robust development board. It's one of the simplest options available for making the electronics world programmable, and it's extremely reliable as well. First, we'll show you how to install the Arduino development environment (often called IDE, or integrated development environment) on your computer. After that, you'll plug in a USB cable and upload your first program (called a sketch in Arduino parlance). There's only one program you install on the Arduino—the sketch that you're running. Aside from that, there's nothing else to maintain because, unlike with Raspberry Pi, Arduino has no operating system.

First of all the main reasons is it is an open source programmable board with a built in microcontroller and the software IDE. And this software IDE will help you change the behaviour of the microcontroller according to your needs. So, it accepts Analog as well as digital signals which can be given as inputs and it will give outputs which are mainly digital. So, no extra hardware is required to load a program into the controller board. So, for the people who have work with 8051 series microcontrollers, 8085 microprocessors, they must have remembered that you needed an extra programmer to actually program the processor board and they are well lots of interfacing ICs and all those things are not required with the Arduino base systems.

Arduino UNO

One of the most popular Arduino boards out there is the Arduino Uno. While it was not actually the first board to be released, it remains to be the most actively used and most widely documented on the market.

Types of Arduino boards:

- Arduino boards based on ATMEGA328 microcontroller.
- Arduino boards based on ATMEGA32u4 microcontroller.
- Arduino boards based on ATMEGA2560 microcontroller.
- Arduino boards based on AT91SAM3X8E microcontroller.

Features of Arduino UNO:

- The operating voltage is 5V.
- The recommended input voltage will range from 7v to 12V.
- The input voltage ranges from 6v to 20V.
- Digital input/output pins are 14.
- Analog i/p pins are 6.
- DC Current for each input/output pin is 40 mA.
- Flash Memory is 32 KB.

Power Supply

The Arduino Uno power supply can be done with the help of a USB cable or an external power supply. The external power supplies mainly include AC to DC adapter otherwise a battery. The adapter can be connected to the Arduino Uno by plugging into the power jack of the Arduino board. Similarly, the battery leads can be connected to the Vin pin and the GND pin of the POWER connector. The suggested voltage range will be 7 volts to 12 volts.

Input & Output

The 14 digital pins on the Arduino Uno can be used as input & output with the help of the functions like pinMode(), digitalWrite(), & Digital Read().

Pin 2 & Pin 3 (External Interrupts):

External pins can be connected to activate an interrupt over a low value, change in value.

Pins 3, 5, 6, 9, 10, & 11 (PWM):

This pin gives 8-bit PWM o/p by the function of analog Write().

SPI Pins (Pin-10 (SS), Pin-11 (MOSI), Pin-12 (MISO), Pin-13 (SCK):

These pins maintain SPI-communication, even though offered by the fundamental hardware, is not presently included within the Arduino language.

Pin-13(LED):

The inbuilt LED can be connected to pin-13 (digital pin). As the HIGH-value pin, the light emitting diode is activated, whenever the pin is LOW.

Pin-4 (SDA) & Pin-5 (SCL) (I2C):

It supports TWI-communication with the help of the Wire library.

TX and RX LEDs

On your board, you will find two labels: TX (transmit) and RX (receive). They appear in two places on the Arduino UNO board. First, at the digital pins 0 and 1, to indicate the pins responsible for serial communication. Second, the TX and RX led (13). The TX led flashes with different speed while sending the serial data. The speed of flashing depends on the baud rate used by the board. RX flashes during the receiving process.

Digital I/O

The Arduino UNO board has 14 digital I/O pins (15) (of which 6 provide PWM (Pulse Width Modulation) output. These pins can be configured to work as input digital pins to read logic values (0 or 1) or as digital output pins to drive different modules like LEDs, relays, etc. The pins labelled "~" can be used to generate PWM.

AREF

AREF stands for Analog Reference. It is sometimes, used to set an external reference voltage (between 0 and 5 Volts) as the upper limit for the analog input pins.

Reset Pin: This pin is used for reset (RST) the microcontroller.

Memory:

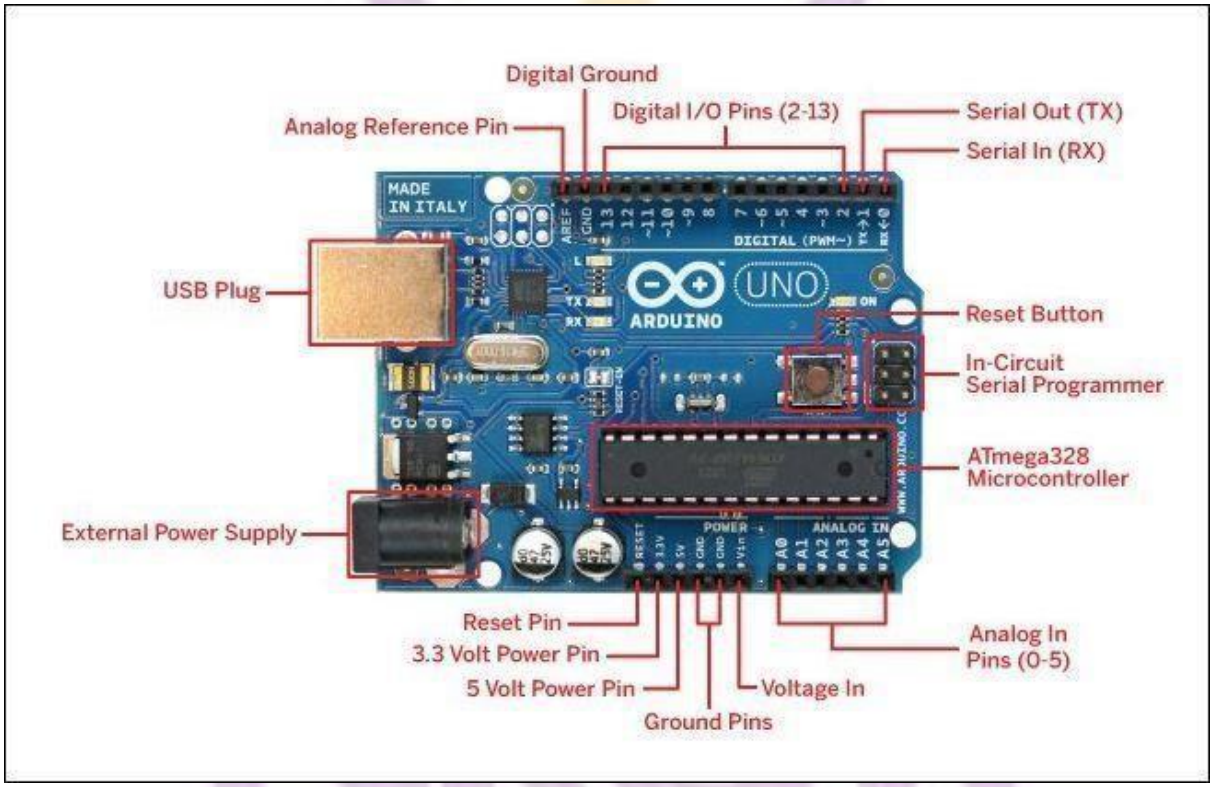
The memory of this Atmega328 Arduino microcontroller includes flash memory-32 KB for storing code, SRAM-2 KB EEPROM-1 KB.

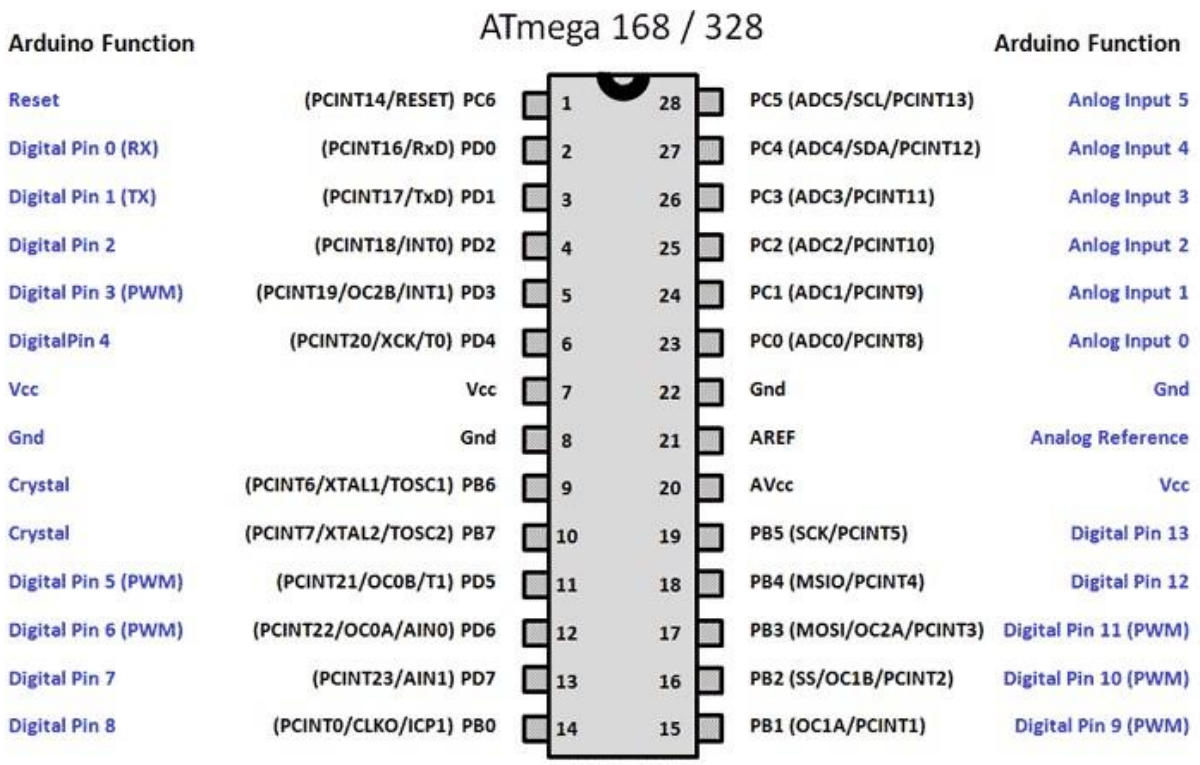
Communication:

The Arduino Uno ATmega328 offers UART TTL-serial communication, and it is accessible on digital pins like TX (1) and RX (0). The software of an Arduino has a serial monitor that permits easy data. There are two LEDs on the board like RX & TX which will blink whenever data is being broadcasted through the USB.

Arduino Reset

You can reset your Arduino board, i.e., start your program from the beginning. You can reset the UNO board in two ways. First, by using the reset button on the board. Second, you can connect an external reset button to the Arduino pin labelled RESET.





Digital Pins 11, 12 & 13 are used by the ICSP header for MISO, MOSI, SCK connections (ATmega pins 17, 18 & 19). Avoid low impedance loads on these pin, when using the ICSP header.

How To Program Arduino

Once the circuit has been created on the breadboard, you'll need to upload the program (known as a sketch) to the Arduino. The sketch is a set of instructions that tells the board what functions it needs to perform. An Arduino board can only hold and perform one sketch at a time. The software used to create Arduino sketches is called the IDE which stands for Integrated Development Environment.

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.

Operators:

Logical operators evaluate either one or two relational or logical statements. There are 3 logical operators in Arduino IDE:
AND, OR,NOT

Arduino control statements include:

- If statement
- Else statement
- Else if statement
- For statement
- While statement
- Do while statement
- Switch case
- Continue

If Statement:

IF statement is basically the simplest form conditional control statements, it is a conditional statement. An "if statement" code evaluates a unique condition, and executes a series of instructions or just an instruction if the condition is true.

Else Statement

Most time, an IF statement is immediately followed by an ELSE statement, the ELSE statement tells the alternate instruction that should be executed when the IF statement is false.

Else If Statement

"Else if statement" is used when we want to check for three different conditions. It includes an IF statement, ELSE IF statement and ELSE statement all in same.

For statement:

For statement is also a conditional statement for Arduino control structure used for repetitive operation. As the name implies, it is used to carry out a repetitive operation for a true condition.

While Statement

A while statement is just like an "if statement" except it continues to repeat block of code (a block of code is what is within the curly braces) as long as the condition is true.

Do While Statement

A do while statement is like the else if statement but works in the same manner as the while loop, except that the condition is tested at the end of the loop, hence, the do statement will always run at least once.

Switch Case Statement

There comes a time in a design, when we wish to have an action taking with respect to a specific result, in a wide range of results. Take for example, let's say you are trying to monitor the level of water in a tank using an ultrasonic sensor,

you wish to turn on an LED various levels of the water in the tank. Let's say we are looking at 10 levels. In our arduino code, we would have a variable that records the distance of the water from the ultrasonic sensor, with this distance; we can pick the levels we want.

Loop Statements:

while loop

while loops will loop continuously, and infinitely, until the expression inside the parenthesis, () becomes false. Something must change the tested variable, or the while loop will never exit.

do…while loop

The do…while loop is similar to the while loop. In the while loop, the loop-continuation condition is tested at the beginning of the loop before performed the body of the loop.

for loop

A for loop executes statements a predetermined number of times. The control expression for the loop is initialized, tested and manipulated entirely within the for loop parentheses.

Nested Loop

This allows you to use one loop inside another loop. The following example illustrates the concept.

Infinite loop

It is the loop having no terminating condition, so the loop becomes infinite.

Arduino Program

An Arduino program starts by executing the code inside the setup() function once. After that, the code inside loop() is repeated forever (or until you disconnect the power).

Blinking an LED

The LED blinking sketch is the first program that you should run to test whether your Arduino board is working and is configured correctly. It is also usually the very first programming exercise someone does when learning to program a microcontroller. A light-emitting diode (LED) is a small electronic component that's a bit like a light bulb, but is more efficient and requires lower voltages to operate

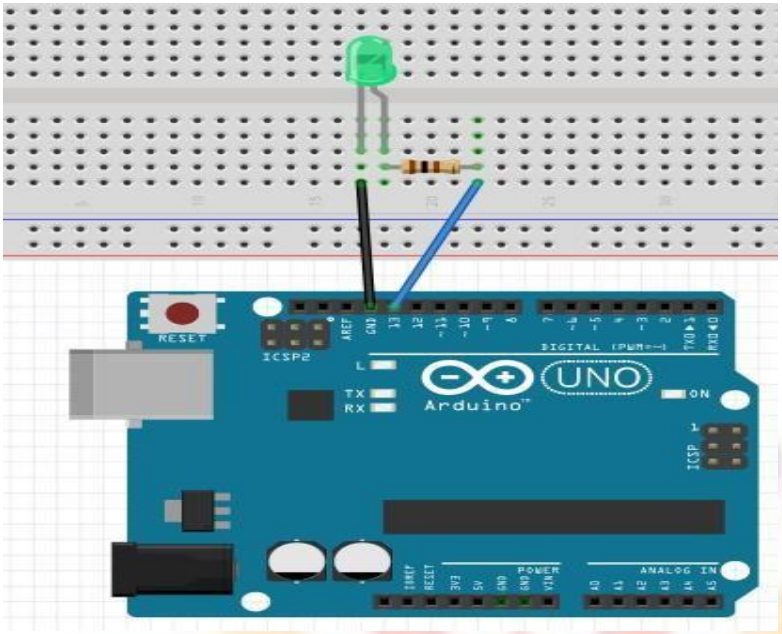
Example of led blinking:

```
const int LED = 13; // LED connected to digital pin 13 void
setup()
```

```
{
 pinMode(LED, OUTPUT); // sets the digital pin as output
}
void loop()
{
 digitalWrite(LED, HIGH); // turns the LED on
 delay(1000); // waits for a second digitalWrite(LED,
 LOW); // turns the LED off delay(1000); // waits for
 a second
}
```

Program Explanation:

- Turns pin 13 into an output (just once at the beginning)
- Enters a loop
- Switches on the LED connected to pin 13
- Waits for a second
- Switches off the LED connected to pin 13
- Waits for a second
- Goes back to beginning of the loop.



Example : Turn on LED while the button is pressed

```
const int LED = 13; // the pin for the LED

const int BUTTON = 7; // the input pin where the pushbutton is connected int
val = 0; // val will be used to store the state
 // of the input pin
void setup() {
 pinMode(LED, OUTPUT); // tell Arduino LED is an output
 pinMode(BUTTON, INPUT); // and BUTTON is an input
}

void loop(){

 val = digitalRead(BUTTON); // read input value and store it

 // check whether the input is HIGH (button pressed) if
 (val == HIGH)

 {

 digitalWrite(LED, HIGH); // turn LED ON

 } else

 { digitalWrite(LED,
LOW);
 }

}
```

Now go test this code. You'll find that the light changes so rapidly that you can't reliably set it on or off with a button press. Let's look at the interesting parts of the code: state is a variable that stores either 0 or 1 to remember whether the LED is on or off. After the button is released, we initialise it to 0 (LED off).

Later, we read the current state of the button, and if it's pressed (val == HIGH), we change state from 0 to 1, or vice versa. We do this using a small trick, as state can be only either 1 or 0. The trick I use involves a small mathematical expression based on the idea that 1 – 0 is 1 and 1 – 1 is 0:state = 1 – state. The line may not make much sense in mathematics, but it does in programming. The symbol = means "assign the result of what's after me to the variable name before me"—in this case, the new value of state is assigned the value of 1 minus the old value of state.

Example : Fade an LED in and out like on a sleeping Apple computer

```
const int LED = 13; // the pin for the LED

int i = 0; // We'll use this to count up and down
void setup( )
{

 pinMode(LED, OUTPUT); // tell Arduino LED is an output

}

void loop ( )

{

 for (i = 0; i < 255; i++)

{                              // loop from 0 to 254 (fade in)
 analogWrite(LED, i); // set the LED brightness
 delay(10);           // Wait 10ms because analogWrite is instantaneous and we
                      would not see any change

 }
 for (i = 255; i > 0; i--)

 {                                    // loop from 255 to 1 (fade out)
 analogWrite(LED, i); // set the LED brightness
 delay(10); // Wait 10ms

 }
```

Use a Light Sensor Instead of the Pushbutton:

Arduino is able to detect whether there is a voltage applied to one of its pins and report it through the digital Read() function. This kind of either/or response is fine in a lot of applications, but the light sensor that we just used is able to tell us not just whether there is light, but also how much light there is. This is the difference between an on/off sensor (which tells us whether something is there) and an analogue sensor, whose value continuously changes. In order to read this type of sensor, we need a different type of pin. In the lower-right part of the Arduino board, you'll see six pins marked "Analog In"; these are special pins that can tell us not only whether there is a voltage applied to them, but if so, also its value. By using the analog Read() function, we can read the voltage applied to one of the pins. This function returns a number between 0 and 1023, which represents voltages between 0 and 5 volts. For example, if there is a voltage of 2.5 V applied to pin number 0, analog Read(0) returns 512.

Blink LED at a rate specified by the // value of the analogue input

```
int ldrpin =A0// light sensor used is LDR int
LED = 13; // the pin for the LED
int val = 0; // variable used to store the value coming from the sensor void
setup ( )
  {
 pinMode(LED, OUTPUT); // LED is as an OUTPUT
 // Note: Analogue pins are  automatically set as inputs
}
void loop( )
{
 val = analogRead(0); // read the value from the sensor
 digitalWrite(LED, HIGH); // turn the LED on
 delay(val); // stop the program for some time
 digitalWrite(LED, LOW); // turn the LED off
 delay(val); // stop the program for some time
}
```

Traffic lights code:

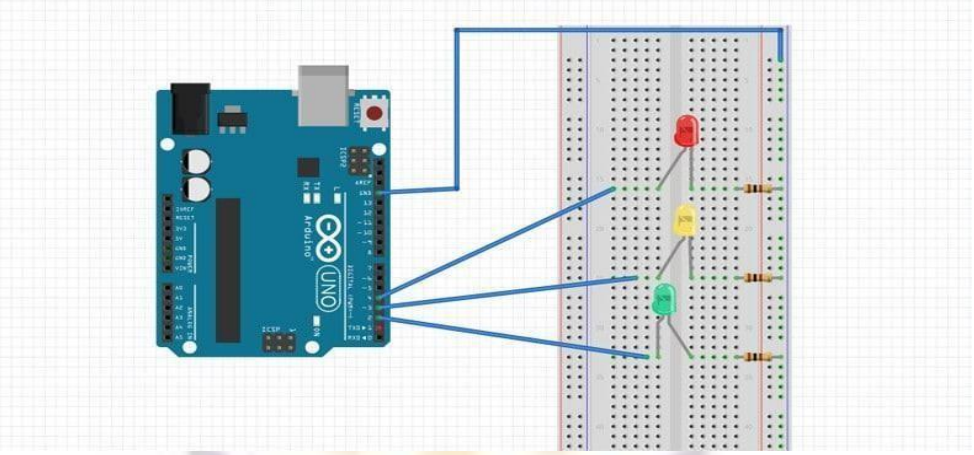
Start by defining variables so that you can address the lights by name rather than a number.

```
// variables

 int GREEN = 2;
 int YELLOW =3;
 int RED = 4;

// basic functions
```

let's add the setup function, where you'll configure the red, yellow and green LEDs to be outputs. Since you have created variables to represent the pin numbers, you can now refer to the pins by name instead:

```
void setup()

{

  pinMode(GREEN,OUTPUT);
  pinMode(YELLOW,OUTPUT);
  pinMode(RED, OUTPUT);
}

void loop( )

{

  green_light();
  delay(5000);
  yellow_light();
  delay(2000);
  red_light();
  delay(5000);

  }
```

```
void green_light()
{
  digitalWrite(GREEN,  HIGH);
  digitalWrite(YELLOW,
  LOW);        digitalWrite(RED,
  LOW);
}

void yellow_light()
{
  digitalWrite(GREEN,                LOW);
  digitalWrite(YELLOW,               HIGH);
  digitalWrite(RED, LOW);
}

void red_light()
{
  digitalWrite(GREEN,                LOW);
  digitalWrite(YELLOW,               LOW);
  digitalWrite(RED, HIGH);
}
```



Integration of Sensors and Actuators with Arduino

So, first of all as we have already learned by now; sensors are basic electronic elements they convert physical quantities or measurements into electrical signals and more or less sensors can be classified into either analog or digital sensors.

Types of Sensors

Some commonly used sensors:

- Temperature
- Humidity
- Compass
- Light
- Sound
- Accelerometer

Sensor Interface with Arduino:

Digital Humidity and Temperature Sensor (DHT)

- PIN 1-3.3V-5V Power supply
- PIN 2- Data
- PIN 3-Null
- PIN 4- Ground

DHT Sensor Library

Arduino supports a special library for the DHT11 and DHT22 sensors.

Provides function to read the temperature and humidity values from the data pin dht.readHumidity() dht.readTemperature()

TMP36 Temperature Sensor Example:

We'll use analog input 0 to read Temperature Data const
int temperature Pin = 3;
void setup()

{

Serial.begin(9600);

}

void loop()

{

float voltage, degreesC, degreesF; voltage

= getVoltage(temperaturePin);

// Now we'll convert the voltage to degrees Celsius.

// This formula comes from the temperature sensor datasheet: degreesC = (voltage - 0.5) * 100.0;

// Send data from the Arduino to the serial monitor window

```
Serial.print("voltage:          ");
Serial.print(voltage);
Serial.print("   deg   C:   ");
Serial.println(degreesC);
delay(1000); // repeat once per second (change as you wish!)

}
```

Light Sensor:

In this example we will use a light sensor to measure the light intensity of the room. If it's dark, we will turn on the light (LED). If it's bright, we'll turn off the light (LED). A light sensor / photocell is a sensor used to detect light. The resistance decreases with increasing light intensity (stronger light).

Program:

```
int photocellPin = 2;
int ledPin = 13;
int photocellReading;
const float limit = 100;
void setup(void)
{

Serial.begin(9600); //baud rate of data is 9600bps
pinMode(ledPin, OUTPUT);
}

void loop( )

{

photocellReading = analogRead(photocellPin);
Serial.print("Analog reading = ");
Serial.println(photocellReading);
if (photocellReading < limit)

{

digitalWrite(ledPin, HIGH);

}
```

```
else
{
digitalWrite(ledPin, LOW);
}
Delay(1000)
}
```

Actuators:

So we will learn how to integrate this motors and make it perform according to our requirements. So, actuators are basically mechanical or electro mechanical devices. They convert energy or signals into motion. And mainly use to provide controlled motion to other components of various mechanical structures or devices. So, basic working principle is, in the servo motor you have various mechanical structures like gears and screws and ball bearings, which are interfaced with a small motor over here, and this produces very control motion, but is able to perform much more efficiently then this motor alone would have been able to. So, like generally for servers the top requirements are high as compared to normal dc motors.

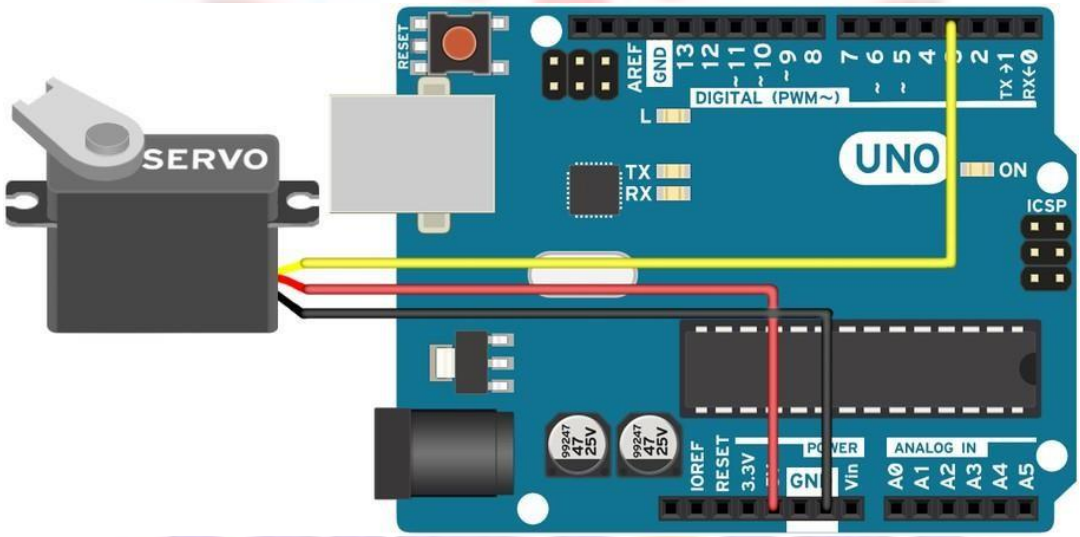
Servo motors are great devices that can turn to a specified position. Usually, we have a servo arm that can turn 180 degrees. Using the Arduino, we can tell a servo to go to a specified position and it will go there. A servo motor has everything built in: a motor, a feedback circuit, and most important, a motor driver. It just needs one power line, one ground, and one control pin.

Program for servo motors:

```
//Include the Servo library
#include <Servo.h>
// Declare the Servo pin int
servoPin = 3;
// Create a servo object
Servo Servo1;
void setup() {
  // We need to attach the servo to the used pin number  Servo1.attach(servoPin);
}
void loop(){
```

```
    // Make servo go to 0 degrees
    Servo1.write(0);
    delay(1000);

    // Make servo go to 90 degrees Servo1.write(90);
    delay(1000);
    // Make servo go to 180 degrees
    Servo1.write(180);
    delay(1000);

}
```

Circuit diagram:



How to Use an Arduino with Linear Actuators:

Arduino is a specific open-source community\company\project in one, which specializes in microcontrollers, specifically the building and programming of them. Arduino also offers simple kits made for easy assembly. Arduino controllers are small controllers of microchips and boards which allow for remote control of certain pieces of equipment. These microcontrollers are both digital and analog, meaning they can be used for a wide variety of equipment, regardless of whether or not the equipment is digital or analog itself. These microcontrollers can be used with linear actuators, as a means of controlling them.

As for coding your Arduino microcontroller, we have included a simple sweep program that shows how to extend and retract a linear actuator at full speed.

```
//Define pin numbers for Single Board
int ENABLE1 = 8;

int  FWD1  =  11;
int REV1 = 3;
int     Speed;
void setup()
{

 // initialize the digital pins as an output.
  pinMode(ENABLE1,OUTPUT);
  pinMode(FWD1,OUTPUT);
  pinMode(REV1, OUTPUT);
}

void  loop()

 {

  Speed  =  255;  //set  a  speed  between  0-255
  Forward();
  delay(5000);  //5  second  delay
  Stop();
  delay(1000);
  Reverse();
  delay(5000);
  Stop();
  delay(1000);
}

void Forward()

{

  digitalWrite(ENABLE1,HIGH);
  analogWrite(REV,                    0);
  analogWrite(FWD, Speed);
}
```

```
void Reverse()
{
  digitalWrite(ENABLE1,HIGH);
  analogWrite(FWD,0);
  analogWrite(REV, Speed);
}
void Stop()
{
  digitalWrite(ENABLE1,LOW);
  analogWrite(FWD1, 0);
  analogWrite(REV1, 0);
}
```

Unit -III

Introduction to Python programming:

What is Python?

Python is a high-level scripting language which can be used for a wide variety of text processing, system administration and internet-related tasks. Unlike many similar languages, it's core language is very small and easy to master, while allowing the addition of modules to perform a virtually limitless variety of tasks. Python is a true object-oriented language, and is available on a wide variety of platforms. There's even a python interpreter written entirely in Java, further enhancing python's position as an excellent solution for internet-based problems. Python was developed in the early 1990's by Guido van Rossum, then at CWI in Amsterdam, and currently at CNRI in Virginia. In some ways, python grew out of a project to design a computer language which would be easy for beginners to learn, yet would be powerful enough for even advanced users. This heritage is reflected in python's small, clean syntax and the thoroughness of the implementation of ideas like object-oriented programming, without eliminating the ability to program in a more traditional style. So python is an excellent choice as a first programming language without sacrificing the power and advanced capabilities that users will eventually need.

The very Basics of Python

There are a few features of python which are different than other programming languages, and which should be mentioned early on so that subsequent examples don't seem confusing. Python statements do not need to end with a special character – the python interpreter knows that you are done with an individual statement by the presence of a newline, which will be generated when you press the "Return" key of your keyboard. If a statement spans more than one line, the safest course of action is to use a backslash (\) at the end of the line to let python know that you are going to continue the statement on the next line; you can continue using backslashes on additional continuation lines. Python provides you with a certain level of freedom when composing a program, but there are some rules which must always be obeyed. One of these rules, which some people find very surprising, is that python uses indentation (that is, the amount of white space before the statement itself) to indicate the presence of loops, instead of using delimiters like curly braces ({}) or keywords (like "begin" and "end") as in many other languages. The amount of indentation you use is not important, but it must be consistent within a given depth of a loop, and statements which are not indented must begin in the first column.

Python Features:

1. **Easy-to-learn** − Python has few keywords, simple structure, and a clearly defined syntax. This allows a student to pick up the language quickly.

2. **Easy-to-read** − Python code is more clearly defined and visible to the eyes.

3. **Easy-to-maintain** − Python's source code is fairly easy-to-maintain.

4. **A broad standard library** − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.

5. **Interactive Mode** − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.

6. **Portable** − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.

7. **Extendable** − You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.

8. **Databases** − Python provides interfaces to all major commercial databases.

9. **GUI Programming** − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

10. **Scalable** − Python provides a better structure and support for large programs than shell scripting

Basic Principles of Python

Python has many features that usually are found only in languages which are much more complex to learn and use. These features were designed into python from its very first beginnings, rather than being accumulated into an end result, as is the case with many other scripting languages. If you're new to programming, even the basic descriptions which follow may seem intimidating. But don't worry – all of these ideas will be made clearer in the chapters which follow. The idea of presenting these concepts now is to make you aware of how python works, and the general philosophy behind python programming. If some of the concepts that are introduced here seem abstract or overly complex, just try to get a general feel for the idea, and the details will be fleshed out later

1) **Basic Core Language**

Python is designed so that there really isn't that much to learn in the basic language. For example, there is only one basic structure for conditional programming (if/else/elif), two looping commands (while and for), and a consistent method of handling errors (try/except) which apply to all python programs. This doesn't mean that the language is not flexible and powerful, however. It simply means that you're not confronted with an overwhelming choice of options at every turn, which can make programming a much simpler task.

2) Modules

Python relies on modules, that is, self-contained programs which define a variety of functions and data types, that you can call in order to do tasks beyond the scope of the basic core language by using the import command. For example, the core distribution of python contains modules for processing files, accessing your computer's operating system and the internet, writing CGI(Common Gateway Interface) scripts which handle communicating with pages displayed in web browsers, string handling and many other tasks.

3) Object Oriented Programming

Python is a true object-oriented language. The term "object oriented" has become quite a popular buzzword; such high profile languages as C++ and Java are both object oriented by design. Many other languages add some object-oriented capabilities, but were not designed to be object oriented from the ground up as python was. Why is this feature important? Object oriented program allows you to focus on the data you're interested in, whether it's employee information, the results of a scientific experiment or survey, setlists for your favorite band, the contents of your CD collection, information entered by an internet user into a search form or shopping cart, and to develop methods to deal efficiently with your data. A basic concept of object oriented programming is encapsulation, the ability to define an object that contains your data and all the information a program needs to operate on that data. In this way, when you call a function (known as a method in object-oriented lingo), you don't need to specify a lot of details about your data, because your data object "knows" all about itself. In addition, objects can inherit from other objects, so if you or someone else has designed an object that's very close to one you're interested in, you only have to construct those methods which differ from the existing object, allowing you to save a lot of work. Another nice feature of object oriented programs is operator overloading. What this means is that the same operator can have different meanings when used with different types of data. For example, in python, when you're dealing with numbers, the plus sign (+) has its usual obvious meaning of addition. But when you're dealing with strings, the plus sign means to join the two strings together. In addition to being able to use overloading for built-in types (like numbers and strings), python also allows you to define what operators mean for the data types you create yourself.

Python uses the object model abstraction for data storage. Any construct that contains any type of value is an object. Although Python is classified as an "object-oriented programming (OOP) language," OOP is not required to create perfectly working Python applications. You can certainly write a useful Python script without the use of classes and instances. However, Python's object syntax and architecture encourage or "provoke" this type of behavior. Let us now take a closer look at what a Python object is.

All Python objects have the following three characteristics: an identity, a type, and a value.

IDENTITY--Unique identifier that differentiates an object from all others. Any object's identifier can be obtained using the id() built-in function (BIF). This value

is as close as you will get to a "memory address" in Python (probably much to the relief of some of you). Even better is that you rarely, if ever, access this value, much less care what it is at all.

TYPE--An object's type indicates what kind of values an object can hold, what operations can be applied to such objects, and what behavioral rules these objects are subject to. You can use the type() BIF to reveal the type of a Python object. Since types are also objects in Python (did we mention that Python was object-oriented?), type() actually returns an object to you rather than a simple literal.

VALUE--Data item that is represented by an object.

4) Exception Handling

Regardless how carefully you write your programs, when you start using them in a variety of situations, errors are bound to occur. Python provides a consistent method of handling errors, a topic often referred to as exception handling. When you're performing an operation that might result in an error, you can surround it with a try loop, and provide an except clause to tell python what to do when a particular error arises. While this is a fairly advanced concept, usually found in more complex languages, you can start using it in even your earliest python programs.

As a simple example, consider dividing two numbers. If the divisor is zero, most programs (python included) will stop running, leaving the user back at a system shell prompt, or with nothing at all. Here's a little python program that illustrates this concept; assume we've saved it to a file called

div.py:

```
#!/usr/local/bin/python

x = 7

y = 0
print x/y
print "Now we're done!"
```

When we run this program, we don't get to the line which prints the message, because the division by zero is a "fatal" error:

```
% div.py

Traceback (innermost last):

File "div.py", line 5, in ?
print x/y
Zero Division Error: integer division or modulo
```

While the message may look a little complicated, the main point to notice is that the last line of the message tells us the name of the exception that occured. This

allows us to construct an except clause to handle the problem:

```
x = 7
y = 0
try:
print x/y

except Zero Division Error:

print "Oops - I can't divide by zero, sorry!"
print "Now we're done!"
```

Now when we run the program, it behaves a little more nicely:

```
% div.py

Oops - I can't divide by zero, sorry!
Now we're done!
```

Since each exception in python has a name, it's very easy to modify your program to handle errors whenever they're discovered. And of course, if you can think ahead, you can construct try/except clauses to catch errors before they happen.

Data types in python:

Standard Types

- Numbers (separate subtypes; three are integer types)
  - o Integer
    - Boolean
    - Long integer
  - o Floating point real number
  - o Complex number
- String
- List
- Tuple
- Dictionary

- **Types of Numeric Data**

Python supports four types of numeric objects: integers, long integers, floating point numbers, and complex numbers. In general, python will not automatically convert numbers from one type to another, although it provides a complete set of functions to allow you to explicitly do these conversions.

To enter numeric data inside an interactive python session or in a script, simply set the value of a variable to the desired number. To specify a floating point number,

either include a decimal point somewhere in the number, or use exponential notation, for example 1e3 or 1E3 to represent 1000 (1 times 10 to the third power). Note that when using exponential notation, numbers are always stored as floating point, even if there is no decimal point. Long integers can be entered by following an ordinary integer with the letter "L", either lowercase (e.g. 27l) or uppercase (e.g. 27L). (Since a lowercase "l" looks so much like the number "1", you may want to get in the habit of using uppercase "L"s in this context.) In python, long integers are actually what are sometimes called "arbitrary precision" integers, since they can have as many digits as you have the patience to type into the computer. On most computers, ordinary integers have a range from about -2 billion to +2 billion. Trying to use an integer larger than this value results in an Overflow Error.

```
>>> x = 2000000000

>>> x = x + x / 2
Traceback (innermost last):
File "<stdin>", line 1, in ?
Overflow Error: integer addition
```

You'll never see such an error when using a long integer:

```
>>> x = 2000000000L

>>> x = x + x / 2

>>>                x

3000000000L
```

Working with numbers in python: #integer

```
>>>a=5

>>>type(a)

<type 'int'>


#floating point

>>>b=2.5

>>>type(b)

<type 'float'>
#Long
```

```
>>>x=988848989897L

>>>type(x)

<type 'long'>
#complex
>>>y=2+5j

>>>y
(2+5j)
>>>type(y)

<type 'complex'>

>>>y.real
2
>>>y.imag
5
#addition
>>>c=a+b

>>>c
7.5
>>>type(b)

<type 'float'>


#subtraction

>>>d=a-b

>>>d
17.5
>>>type(d)

<type 'float'>
```

- **String Constants**

Strings are a collection of characters which are stored together to represent arbitrary text inside a python program. You can create a string constant inside a python program by surrounding text with either single quotes (’), double quotes ("), or a collection of three of either types of quotes (’’’ or """). In the first two cases, the opening and closing quotes must appear on the same line in your program; when you use triple quotes, your text can span as many lines as you

like. The choice of which quote symbol to use is up to you – both single and double quotes have the same meaning in python. Here are a few examples of how to create a string constant and assign its value to a variable:

```
name = 'Phil'
value = "$7.00"
helptext = """You can create long strings of text
spanning several lines by using triple quotes at the
beginning and end of the text"""
```

When the variable help text is printed it would display as three lines, with the line breaks at the same points as in the triple-quoted text. Using a single backslash as a continuation character is an alternative to using triple quoted strings when you are constructing a string constant. Thus, the following two expressions are equivalent, but most programmers prefer the convenience of not having to use backslashes which is offered by triple quotes.

```
Three lines = 'First\
Second\
Third'

Three lines = '''First
Second
Third'''
```

Working with string :

#create string

```
>>>s= "Hello World!"

>>>type(s)

<type 'str'>
```

 # string concatenation

```
>>>t= "this is sample program."

>>>r= s+t

>>>r

'Hello World! This is sample program.'
```

#get length of string

```
>>>len (s)
12
```

#print string

```
>>>print      s
Hello World!
```

List Data

Lists provide a general mechanism for storing a collection of objects indexed by a number in python. The elements of the list are arbitrary — they can be numbers, strings, functions, user-defined objects or even other lists, making complex data structures very simple to express in python. You can input a list to the python interpreter by surrounding a comma separated list of the objects you want in the list with square brackets ([ ]) Thus, a simple list of numbers can be created as follows:

```
>>> mylist = [1,7,9, 13, 22, 31]
```

Python ignores spaces between the entries of a list. If you need to span multiple lines with a list entry, you can simply hit the return key after any comma in the list:

```
>>> newlist = [7, 9, 12, 15,
... 17,19,103]
```

Note that the python interpreter changes its prompt when it recognizes a continuation line, and that no indentation is necessary to continue a line like this. Inside a script, your input can also be broken up after commas in a similar fashion. To create an empty list, use square brackets with no elements in between them ([]).The elements of a list need not be of the same type. The following list contains numeric, string and list data, along with a function:

```
>>> mixlist = [7,'dog','tree',[1,5,2,7],abs].
```

Working with lists:

```
>>>fruits=['apple', 'orange' , 'banana', 'mango']
>>>type(fruits)
<type 'list'>
>>>len(fruits) 4
>>>fruits[1]
'orange'
>>>fruits [1:]
['orange', 'banana' ,'mango']
```

#appending an item to list

```
>>>fruits.append('pear')
>>>fruits
['apple', 'orange' , 'banana', 'mango', 'pear']
```

#Removing an item from list

>>>fruits.remove(‘mango’)

>>>fruits

#appending an item to list

>>>fruits.append(‘pear’)

>>>fruits

[‘apple’, ‘orange’ , ‘banana’, ‘mango’, ‘pear’]

['apple', 'orange' , 'banana', 'pear']

#Inserting an item to list

>>>fruits.insert (1,'mango')

>>>fruits

['apple', 'mango', 'orange', 'banana', 'pear']

#combining lists

>>>vegetables=['potato', 'carrot', 'onion']

>>>vegetables

>>>eatables=fruits+vegatables

>>>['apple', 'mango', 'orange', 'banana', 'pear', ' potato', 'carrot', 'onion']

Tuple Objects

Tuples are very much like lists, except for one important difference. While lists are mutable, tuples, like strings, are not. This means that, once a tuple is created, its elements can't be modified in place. Knowing that a tuple is immutable, python can be more efficient in manipulating tuples than lists, whose contents can change at any time, so when you know you won't need to change the elements within a sequence, it may be more efficient to use a tuple instead of a list. In addition, there are a number of situations (argument passing and string formatting for example) where tuples are required. Tuples are created in a similar fashion to lists, except that there is no need for square brackets surrounding the value. When the python interpreter displays a tuple, it always surrounds it with parentheses; you can use parentheses when inputting a tuple, but it's not necessary unless the tuple is part of an expression. This creates a slight syntactic problem when creating a tuple with either zero or one element; python will not know you're creating a tuple. For an empty (zero-element) tuple, a pair of empty parentheseis (()) can be used. But surrounding the value with parentheses is not enough in the case of a tuple with exactly one element, since parentheses are used for grouping in arithmetic expression. To specify a tuple with only one element in an assignment statement, simply follow the element with a comma, arithmetic expressions, you need to surround it with parentheses, and follow the element with a comma before the closing parenthesis.

Working with Tuple:

>>>fruits=('apple', 'orange' , 'banana', 'mango')

>>>type(fruits)

<type 'tuple'>

>>>len(fruits) 4

>>>fruits[0]
'apple'
>>>fruits [:3]

('apple', 'orange', 'banana')

Dictionaries

Dictionaries (sometimes refered to as associative arrays or hashes) are very similar to lists in that they can contain arbitrary objects and can be nested to any desired depth, but, instead of being indexed by integers, they can be indexed by any immutable object, such as strings or tuples. Since humans can more easily associate information with strings than with arbitrary numbers, dictionaries are an especially convenient way to keep track of information within a program. As a simple example of a dictionary, consider a phonebook. We could store phone numbers as tuples inside a list, with the first tuple element being the name of the person and the second tuple element being the phone number:

>>> phonelist = [('Fred','555-1231'),('Andy','555-1195'),('Sue','555-2193')]

However, to find, say, Sue's phone number, we'd have to search each element of the list to find the tuple with Sue as the first element in order to find the number we wanted. With a dictionary, we can use the person's name as the index to the array. In this case, the index is usually refered to as a key. This makes it very easy to find the information we're looking for:

>>> phonedict = {'Fred':'555-1231','Andy':'555-1195','Sue':'555-2193'}

>>> phonedict['Sue']
'555-2193'
As the above example illustrates, we can initialize a dictionary with a comma-separated list of key/value pairs, separated by colons, and surrounded by curly braces. An empty dictionary can be expressed by a set of empty curly braces ({}).

Working with dictionaries:

>>>student={'name':'Mary', 'id': '4033', 'year': '3'}

>>>student

>>>type (student)

<type 'dict'>

#get length of dictionary

```
>>>len(student) 3

# Get value of key in dictionary

>>>student        ['name']
'Mary'
# Get all key in dictionary

>>>student.keys()     [
'name',  'id',  'year']
```

Python if...else Statement

Every value in Python has a datatype. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes. Decision making is required when we want to execute a code only if a certain condition is satisfied.

The ifelse statement is used in Python for decision making.

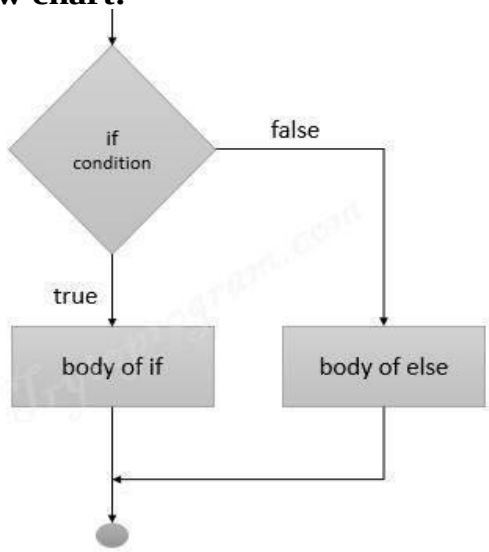
Syntax

```
if test expression
Body of ifelse:
Body of else
```

The if statement evaluates test expression and will execute body of if only when test condition is True.

If the condition is False, body of else is executed.

Python if...else Statement flow chart:



Example of if...else

Program checks if the number is positive or negative

And displays an appropriate message
num = 3
# Try these two variations as well. #
num = -5
# num = 0
if num >= 0:
print("Positive or Zero")
else:
print("Negative number")

In the above example, when num is equal to 3, the test expression is true and body of if is executed and body of else is skipped.

If num is equal to -5, the test expression is false and body of else is executed and body of if is skipped.

If num is equal to 0, the test expression is true and body of if is executed and body of else is skipped

Python if Statement
Syntax
if test expression:

statement(s)

Here, the program evaluates the test expression and will execute statement(s) only if the text expression is True. If the text expression is False, the statement(s) is not executed. In Python, the body of the if statement is indicated by the indentation. Body starts with an indentation and the first unindented line marks the end. Python interprets non-zero values as True. None and 0 are interpreted as False.

Python if Statement Flowchart:

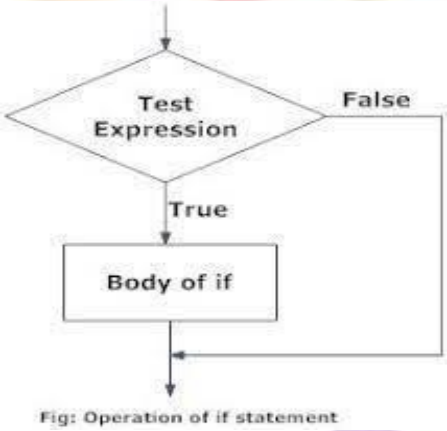


Fig: Operation of if statement

Example: Python if Statement

```
# If the number is positive, we print an appropriate message
num = 3
if num > 0:

print(num, "is a positive number.")
print("This is always printed.")

num = -1

if num >0:

print(num, "is a positive number.") print("This
is also always printed.")
```

When you run the program, the output will be:

```
3 is a positive number
This is always printed
```

This is also always printed.

In the above example, num > 0 is the test expression. The body of if is executed only if this evaluates to True. When variable num is equal to 3, test expression is true and body inside body of if is executed. If variable num is equal to
-1, test expression is false and body inside body of if is skipped. The print() statement falls outside of the if block (unindented). Hence, it is executed regardless of the test expression.

Python if...elif...else Statement Syntax

if test expression:

Body of if

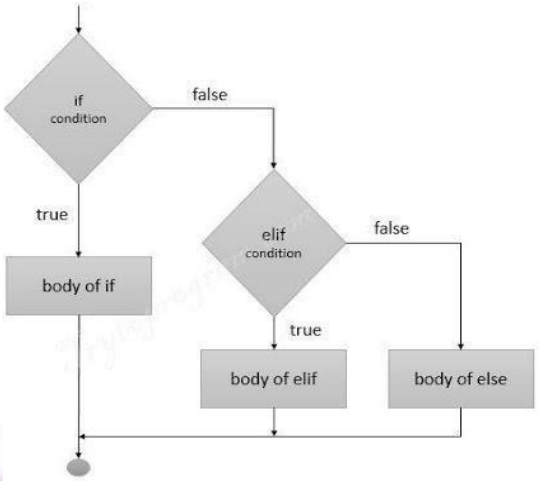
elif test expression:

Body of elif
else:
Body of else

The elif is short for else if. It allows us to check for multiple expressions. If the condition

for if is False, it checks the condition of the next elif block and so on. If all the conditions are False, body of else is executed. Only one block among the several if...elif...else blocks is executed according to the condition. The if block can have only one else block. But it can have multiple elif blocks.

Flowchart of if...elif...else



Example of if...elif...else

```
# In this program,

# we check if the number is positive or #
negative or zero and
# display an appropriate message

num = 3.4                    # Try these two variations as well: #
num = 0
# num = -4.5
if num > 0:
print("Positive number") elif
num == 0:
print("Zero")
else:
print("Negative number")
```

When variable num is positive, Positive number is printed. If num is equal to 0, Zero is printed.
If num is negative, Negative number is printed.

Python Nested if statements

We can have a if...elif...else statement inside another if...elif...else statement. This is called nesting in computer programming. Any number of these statements can be nested inside one another. Indentation is the only way to figure out the level of nesting. This can get confusing, so must be avoided if we can.

Python Nested if Example

```
# In this program, we input a number

# check if the number is positive or negative or zero and display an appropriate message. This time we use nested if
 num = float(input("Enter a number: ")) if
num >= 0:
if num == 0:

print("Zero")
else:
print("Positive number") else:

print("Negative number")
```

Output 1

```
Enter a number: 5
Positive number
```

Output 2

```
Enter a number: -1
Negative number
```

Output 3

```
Enter a number: 0

Zero
```

Python for Loop

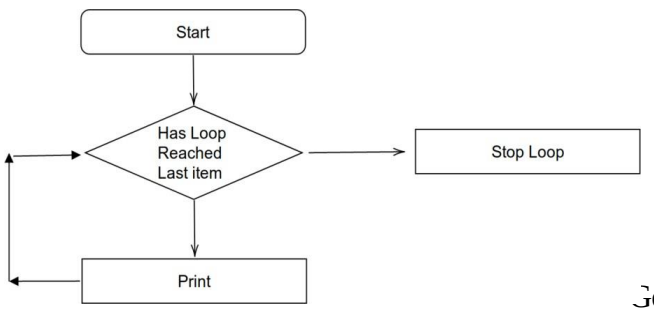
The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects. Iterating over a sequence is called traversal.

Syntax of for Loop

```
for val in sequence:

Body of for
```

Here, val is the variable that takes the value of the item inside the sequence on each iteration. Loop continues until we reach the last item in the sequence. The body of for loop is separated from the rest of the code using indentation.

Flowchart of for Loop



Syntax

```
# Program to find the sum of all numbers stored in a list #

List of numbers

numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]

# variable to store the sum
sum = 0
# iterate over the list for
val in numbers: sum =
sum+val
# Output: The sum is 48
print("The sum is", sum)
```

when you run the program, the output will be:

```
The sum is 48
```

Introduction to raspberry pi

The Raspberry Pi is a remarkable device: a fully functional computer in a tiny and low-cost package. Whether you're looking for a device you can use to browse the web or play games, are interested in learning how to write your own programs, or are looking to create your own circuits and physical devices, the Raspberry Pi – and its amazing community – will support you every step of the way. The Raspberry Pi is known as a single-board computer, which means exactly what it sounds like: it's a computer, just like a desktop, laptop, or smartphone, but built on a single printed circuit board. Like most single-board computers, the Raspberry Pi is small – roughly the same footprint as a credit card – but that doesn't mean it's not powerful: a Raspberry Pi can do anything a bigger and more power-hungry computer can do, though not necessarily as quickly. The Raspberry Pi family was born from a desire to encourage more hands-on computer education around the world. Its creators, who joined together to form the non-profit Raspberry Pi Foundation, had little idea that it would prove so popular: the few thousand built in 2012 to test the waters were immediately sold out, and millions have been shipped all over the world in the years since. These boards have found their ways into homes, classrooms, offices, data centres, factories, and even self-piloting boats and spacefaring balloons. Various models of Raspberry Pi have been released since the original Model B, each bringing either improved specifications or features specific to a particular use-case. The Raspberry Pi Zero family, for example, is a tiny version of the full- size Raspberry Pi which drops a few features – in particular the multiple USB ports and wired network port – in favour of a significantly smaller layout and lowered power needs.

Figure: raspberry pi

While it may look like there's a lot packed into the tiny board, the Raspberry Pi is very simple to understand – starting with its components, the inner workings that make the device tick. This includes the central processing unit (CPU), commonly thought of as the 'brain' of a computer, and the graphics processing unit (GPU), which handles the visual side of things. A brain is no good without memory, however, and on the underside of the Raspberry Pi you'll find exactly that: another chip, which looks like a small, black, plastic square. This is the Pi's random access memory (RAM). When you're working on the Pi, it's the RAM that holds what you're doing; only when you save your work will it be written to the microSD card. Together, these components form the Pi's volatile and non- volatile memories: the volatile RAM loses its contents whenever the Pi is powered off, while the non-volatile microSD card keeps its contents. Turning the board over again you'll find another metal lid to the upper-right, this one featuring an etched Raspberry Pi logo (Figure, overleaf). This covers the radio, the component which gives the Raspberry Pi the ability to communicate with devices wirelessly. The radio itself acts as two main components, in fact: a WiFi radio, for connecting to computer networks; and a Bluetooth radio, for connecting to peripherals like mice and for sending data to or receiving data from nearby smart devices like sensors or smartphones. Another black, plastic-covered chip can be seen to the bottom edge of the board, just behind the middle set of USB ports. This is the network and USB controller, and is responsible for running the Ethernet port and the four USB ports. A final black chip, much smaller than the rest, can be found a little bit above the micro USB power connector to the upper-left of the board (Figure 1); this is known as a power management integrated circuit (PMIC), and handles turning the power that comes in from the micro USB port into the power the Pi needs to run.

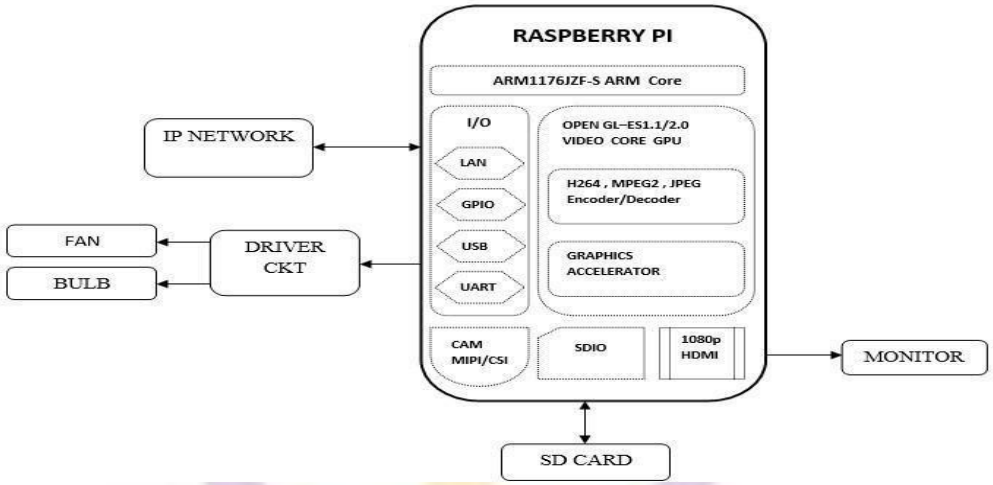


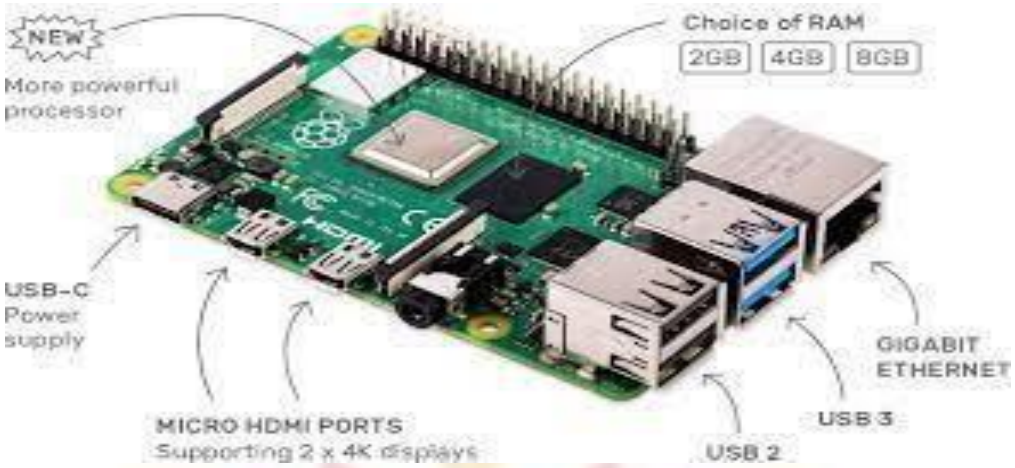
Figure: raspberry pi -block diagram

Features of Raspberry Pi:

- Raspberry Pi is a low-cost mini-computer with the physical size of a credit card.
- Raspberry Pi runs various flavors of Linux and can perform almost all tasks that a normal desktop computer can do.
- Raspberry Pi also allows interfacing sensors and actuators through the general purpose I/O pins.
- Since Raspberry Pi runs Linux operating system, it supports Python "out of the box".

The Raspberry Pi's ports

The Raspberry Pi has a range of ports, starting with four Universal Serial Bus (USB) ports (Figure 2) to the middle and right-hand side of the bottom edge. These ports let you connect any USB-compatible peripheral, from keyboards and mice to digital cameras and flash drives, to the Pi. Speaking technically, these are known as USB 2.0 ports, which means they are based on version two of the Universal Serial Bus standard.



To the left of the USB ports is an Ethernet port, also known as a network port (Figure above).You can use this port to connect the Raspberry Pi to a wired computer network using a cable with what is known as an RJ45 connector on its end. If you look closely at the Ethernet port, you'll see two light-emitting diodes (LEDs) at the bottom; these are status LEDs, and let you know that the connection is working. Just above the Ethernet port, on the left-hand edge of the Raspberry Pi, is a 3.5 mm audio-visual (AV) jack (Figure 2). This is also known as the headphone jack, and it can be used for that exact purpose – though you'll get better sound connecting it to amplified speakers rather than headphones. It has a hidden, extra feature, though: as well as audio, the 3.5 mm AV jack carries a video signal which can be connected to TVs, projectors, and other displays.

High-Definition Multimedia Interface (HDMI) port:

It is the same type of connector you'll find on a games console, set-top box, and TV. The multimedia part of its name tells you that it carries both audio and video signals, while high-definition tells you that you can expect excellent quality. You'll use this to connect the Raspberry Pi to your display device, whether that's a computer monitor, TV, or projector.

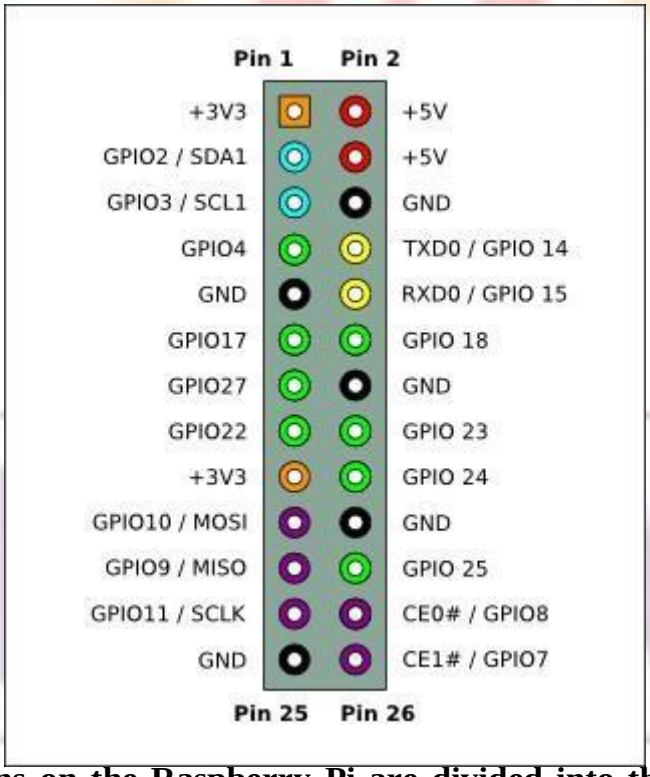
Micro USB power port : which you'll use to connect the Raspberry Pi to a power source. The micro USB port is a common sight on smartphones, tablets, and other portable devices. So you could use a standard mobile charger to power the Pi, but for best results you should use the official Raspberry Pi USB Power Supply.

Raspberry Pi's peripherals

A Raspberry Pi by itself can't do very much, just the same as a desktop computer on its own is little more than a door-stop. To work, the Raspberry Pi needs peripherals: at the minimum, you'll need a microSD card for storage; a monitor or TV so you can see what you're doing; a keyboard and mouse to tell the Pi what to do; and a 5 volt (5 V) micro USB power supply rated at 2.5 amps (2.5 A) or better. With those, you've got yourself a fully functional computer.

USB power supply: A power supply rated at 2.5 amps (2.5A) or 12.5 watts (12.5W) and with a micro USB connector. The Official Raspberry Pi Power Supply is the recommended choice, as it can cope with the quickly switching power demands of the Raspberry Pi.

Raspberry Pi General purpose Input/Output(GPIO)



The GPIO pins on the Raspberry Pi are divided into the following groups:

Power: Pins that are labeled 5.0v supply 5 volts of power and those labeled 3V3 supply 3.3 volts of power. There are two 5V pins and two 3V3 pins.

GND: These are the ground pins. There are eight ground pins.

Input/Output pins: These are the pins labelled with the # sign, for example, #17, #27, #22, etc. These pins can be used for input or output.

UART: The Universal Asynchronous Receiver/Transmitter allows your Raspberry Pi to be connected to serial peripherals. The UART pins are labelled TXD and RXD.

SPI: The Serial Peripheral Interface is a synchronous serial communication interface specification used for short distance communication, primarily in embedded systems. The SPI pins are labeled MOSI, MISO, SCLK, CE0, and CE1.

- MISO(Master In Slave Out): Master line for sending data to the master.
- MOSI(Master Out Slave In): Slave line for sending data to the Peripherals.
- SCK(Serial Clock): Clock generated by master to synchronize data transmission.
- CE0(chip Enable 0): to enable or disable devices.
- CE1(chip Enable 1): to enable or disable devices.

ID EEPROM: Electrically Erasable Programmable Read-Only Memory is a user-modifiable read-only memory that can be erased and written to repeatedly through the application of higher than normal electrical voltage. The two EEPROM pins on the Raspberry Pi (EED and EEC) are also secondary I2C ports that primarily facilitate the identification of Pi Plates (e.g., Raspberry Pi Shields/Add-On Boards) that are directly attached to the Raspberry Pi.SPI(Serial peripheral interface)

I2C (Inter Integrated chip)

The I2C interface pins on raspberry pi allow you to connect hardware modules. I2C interface allow synchronous data transfer with two pins -SDA(data line) and SCL(clock line).

Serial

The serial interface on Raspberry Pi has receive (Rx) and Transmit (Tx) pins for communication with serial periperals.

Interfacing Raspberry Pi with basic peripherals:

In this section you will learn how to get started with developing python programs on Raspberry Pi. Raspberry Pi runs Linux and supports Python out of box. Therefore, you can run any python program which runs on normal computer. However, it is the general purpose input/output

pins(GPIO) on Raspberry Pi that makes it useful for IoT. We can interface Raspberry Pi with sensors and actuators with GPIO pins and SPI, I2C and serial interfaces.

Controlling LED with Raspberry Pi

Let us start with basic example of controlling an LED from Raspberry Pi. In this example the LED is connected to GPIO pin 18. We can connect LED to other GPIO pin as well.

The program uses the RPi.GPIO module to control the GPIO on Raspberry Pi. In this program we set pin 18 direction to output and then True/False alternatively after a delay of one second.

To begin, we import the GPIO package that we will need so that we can communicate with the GPIO pins. We also import the time package, so we're able to put the script to sleep for when we need to. We then set the GPIO mode to GPIO.BOARD/GPIO.BCM, and this means all the numbering we use in this script will refer to the physical numbering of the pins.

Program:

```
import RPi.GPIO as GPIO    # Import Raspberry Pi GPIO library
from time import sleep     # Import the sleep function from the time module
GPIO.setmode(GPIO.BCM/BOARD)    # Use physical pin numbering
GPIO.setup(18, GPIO.OUT)    # Set pin 18 to be an output pin
while True:                                    # Run forever
 GPIO.output(18, GPIO.HIGH)                    # Turn on
 sleep(1)                                      # Sleep for 1 second
 GPIO.output(18,  GPIO.LOW)                    # Turn off
 sleep(1)                                      # Sleep for 1 second
```

Interfacing an LED and Switch with Raspberry Pi

In this example the LED is connected to GPIO pin 18 and the switch is connected to pin 13. In the infinite while loop the value of pin 13 is checked and the state of LED is toggled if the switch is pressed. This example shows how to get input from GPIO pins and process the input and take some actions.

Program

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setup(13,GPIO.IN)                #button
GPIO.setup(18,GPIO.OUT)               #led

while True:

        if (GPIO.input(13)):

                print("on")
GPIO.output(18, GPIO.HIGH)
while False:

        if (GPIO.input(13)):

                print("off")

                GPIO.output(18, GPIO.LOW)
```

Interfacing Light Dependent Resistor (LDR) in Raspberry Pi:

This is code for interfacing LDR in Raspberry Pi. First of all, we have to import the Light Sensor code for LDR from the GPIOZERO library. Assign GPIO pin to variable LDR and also pass the GPIO pin number as an argument to the Light Sensor method. While loop print's value of LDR.

Program:

```
from gpiozero import LightSensor

LDR = LightSensor (4)      # light sensor given to pin 4
while True:
        print(ldr.value)
```

Python program for switching LED based on LDR readings:

Since we only have one input/output pin, we only need to set one variable. Set this variable to the number of the pin you have acting as the input/output pin. Next, we have a function called rc_time that requires one parameter, which is the pin number to the circuit. In this function, we initialize a variable called count, and we will return this value once the pin goes to high. We then set our pin to act as an output and then set it to low. Next, we have the script sleep for 10ms.After this, we then set the pin to become an input, and then we enter a while loop. We stay in this loop until the pin goes to high, this is when the capacitor charges to about 3/4.Once the pin goes high, we return the count value

to the main function. You can use this value to turn on and off an LED, activate something else, or log the data and keep statistics on any variance in light.

Program:

```
import RPi.GPIO as GPIO
import time import sleep def
rc_time (pin_to_circuit):
    count = 0

      #Output on the pin for
    GPIO.setup(pin_to_circuit, GPIO.OUT)
    GPIO.output(pin_to_circuit, GPIO.LOW)
    sleep(1)
    #Change the pin back to input
    GPIO.setup(pin_to_circuit, GPIO.IN)
      #Count until the pin goes high

    while (GPIO.input(pin_to_circuit) == GPIO.LOW): count
        += 1
    return count
```

Implementation of IoT with Raspberry Pi
Why it is important for IoT:

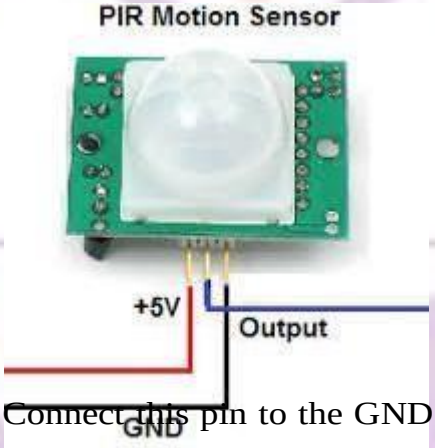
The Internet of Things (IoT) is a scenario in which objects, animals or people are provided with single identifiers and the capability to automatically transfer and the capability to automatically transfer data more to a network without requiring human-to-human or human-to-computer communication. IoT has evolved from the meeting of wireless technologies, micro- electromechanical systems (MEMS) and the internet. The Raspberry Pi is a popular choice when developing IoT products. It offers a complete Linux server with a tiny platform at an incredibly low price. Actually, the Raspberry Pi is so well-known to IoT that the company has featured several Internet of Things projects on their site. Here you will find projects and community support for a range of IoT activities. Take for example, the World's First Cloud Texting Enabled Espresso Machine – powered by Raspberry Pi.Partnered with the Zipwhip cloud texting application, the Raspberry Pi connects to an espresso machine and allows people to text a message to it that automatically turns it on and starts brewing beverages. Raspberry Pi can be plugged into a TV, computer monitor, and it uses a standard keyboard and mouse. It is user-friendly as it can be handled by all the age groups. It does everything you would expect a desktop computer to do like word-processing, browsing the internet spreadsheets, playing games to playing high definition videos. It is used in many

applications like in a wide array of digital maker projects, music machines, parent detectors to the weather station and tweeting birdhouses with infrared cameras. All models feature on a Broadcom system on a chip (SOC), which includes chip graphics processing unit GPU(a Video Core IV), an ARM-compatible and CPU. The CPU speed ranges from 700 MHz to 1.2 GHz for the Pi 3 and onboard memory range from 256 MB to 1 GB RAM. An operating system is stored in the secured digital SD cards and program memory in either the Micro SDHC or SDHC sizes. Most boards have one to four USB slots, composite video output, HDMI and a 3.5 mm phone jack for audio. Some models have WiFi and Bluetooth.

Python program for using PIR sensor:

Connecting to a Sensor to Detect Motion

To demonstrate how to use the GPIO to connect to an external sensor, we'll now use a PIR motion sensor to detect motion. For this, I used the Parallax PIR Motion Sensor (see fig). The PIR Sensor detects motion by measuring changes in the infrared (heat) levels emitted by surrounding objects of up to three meters.The Parallax Motion sensor has three pins (see Figure):



GND: The Ground pin. Connect this pin to the GND on the GPIO.

VCC: The voltage pin. Connect this pin to one of the 5V pins on the GPIO.

OUT: The output pin. Connect this to one of the Input/Output pins on the GPIO.

```
import RPi.GPIO as GPIO                    #1
import time                                #2
pirsensor = 4                              #3
GPIO.setmode(GPIO.BCM)                     #4

GPIO.setup(pirsensor, GPIO.IN)             #5
```

```
previous_state=False                                    #6
current_state = False

while          True:
    time.sleep(0.1)                                     #7
    previous_state = current_state                      #8
                                                        #9
    current_state = GPIO.input(pirsensor)               #10
    if current_state != previous_state:                 #11
    if current_state:                                   #12
   print("Motion not Detected!")              #13
```

#1: The latest version of Raspbian includes the RPI.GPIO Python library pre-installed, so you can simply import that into your Python code. The RPI.GPIO is a library that allows your Python application to easily access the GPIO pins on your Raspberry Pi. The as keyword in Python allows you to refer to the RPI.GPIO library using the shorter name of GPIO.

#2: The application is going to insert some delays in the execution, so you need to import the time module.

#3: You declare a variable named pirsensor to indicate the pin number for which the Output pin on the PIR sensor is connected to the GPIO pin. In this example, it's GPIO pin #4.

#4: There are two ways to refer to the pins on the GPIO: either by physical pin numbers (starting from pin 1 to 40 on the Raspberry Pi 2/3), or Broadcom GPIO numbers (BCM). Using BCM is very useful with a ribbon cable (such as the Adafruit T-Cobbler Plus) to connect the Raspberry Pi to the breadboard. The BCM numbers refer to the labels printed on the T- Cobbler Plus (see Figure 8). For this example, we're using the BCM numbering scheme. That means that when we say we're getting the input from pin 4, we're referring to the pin printed as #4 on the T-Cobbler Plus.

#5: Initialize the pin represented by the variable pin sensor as an input pin. Also, we use a pull-down resistor (GPIO.PUD_DOWN) for this pin.

#6: There are two variables to keep track of the state of the sensor. #7: We use an infinite loop to check the state of the sensor repeatedly. #8: Inserts a slight delay of 1 second to the execution of the program #9: Save the current state of the sensor.
#10: The GPIO.input() function reads the value of the GPIO pin (#4 in this

case). When motion is detected, it returns a value of true.

#11: Compare the previous state and the current state to see if the motion sensor has a change in state. If there's a change, it means that either the sensor has just detected motion (when the state changes from false to true), or that the sensor is resetting itself (when the state changes from true to false) a few seconds after motion has been detected.

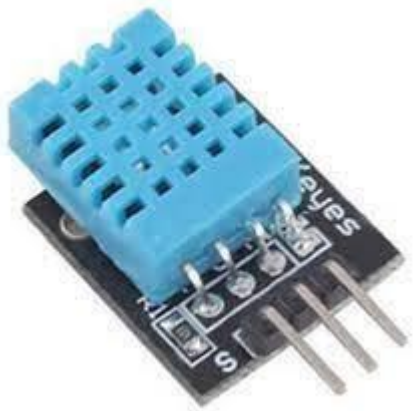
#12: If the current state is true, it means that motion has been detected. #13:
Print out the string "Motion Detected!"

DHT11 Temperature and Humidity Sensor and the Raspberry Pi

The DHT11 requires a specific protocol to be applied to the data pin. In order to save time trying to implement this yourself it's far easier to use the Adafruit DHT library. The library deals with the data that needs to be exchanged with the sensor but it is sensitive to timing issues. The Pi's operating system may get in the way while performing other tasks so to compensate for this the library requests a number of readings from the device until it gets one that is valid.

Program:

```
import RPi.GPIO as GPIO
import Adafruit_DHT
# Set sensor type : Options are DHT11,DHT22 or AM2302
sensor=Adafruit_DHT.DHT11
gpio=17

# Use read_retry method. This will retry up to 15 times to get a sensor reading (waiting 2 seconds between each retry).

humidity, temperature = Adafruit_DHT.read_retry(sensor, gpio)

# Reading the DHT11 is very sensitive to timings and occasionally the Pi might fail to get a valid reading. So check if readings are valid.

if humidity is not None and temperature is not None:

  print('Temp={0:0.1f}*C Humidity={1:0.1f}%'.format(temperature, humidity)) else:
  print('Failed to get reading. Try again!')
```



Motors programming:

```
Import RPi.GPIO as GPIO
from gpiozero import Motor
motor1 = Motor(4, 14)          # to make it easier to see which pin is which, you can use Motor(forward=4, backward=14) .
motor2 = Motor(17, 27)         # forward=17, backward =27
motor1.forward()
motor2.forward()
motor1.backward()
motor2.backward()
while True: # The Motor class also allows you to reverse the motor's direction.
    sleep(5)
    motor1.reverse()
    motor2.reverse()
motor1.stop()                  # Now stop the motors
motor2.stop()
```

Buzzer program:

```
from gpiozero import Button, Lights, buzzer. buzzer
= Buzzer(15)
while True:
    lights.on()
    buzzer.on()
    button.wait_for_press()
    lights.off()
    buzzer.off()
    button.wait_for_release()
```

Traffic lights program:

```
from gpiozero import, Button, TrafficLights
from time import sleep
```

```
while True:
    lights.green.on()
    sleep(1)
    lights.orange.on()
    sleep(1)
    lights.red.on()
    sleep(1) lights.off()
```

Add a wait_for_press so that pressing the button initiates the sequence:

Try adding the button for a pedestrian crossing. The button should move the lights to red (not immediately), and give the pedestrians time to cross before moving back to green until the button is pressed again.

Program:

```
while True:
    button.wait_for_press()
    lights.green.on() sleep(1)
    lights.amber.on() sleep(1)
    lights.red.on()
    sleep(1)
    lights.off()
```

Unit-IV

Implementation of IoT using Raspberry pi

Smart_Health:

The Internet of things is the inter-connection of devices, apps, sensors and network connectivity that enhances these entities to gather and exchange data. The distinguishing characteristic of Internet of Things in the healthcare system is the constant monitoring a patient through checking various parameters and also infers a good result from the history of such constant monitoring. Many such devices equipped with medical sensors are present in the ICUs now-a-days. There could be instances where the doctor couldn't be alerted in time when there is an emergency, despite of 24 hours of monitoring. Also there might be hurdles in sharing the data and information with the specialist doctors and the concerned family members and relatives.

The most tremendous use of IoT is in healthcare management which provides health and environment condition tracking facilities. IoT is nothing but linking computers to the internet utilizing sensors and networks [9, 10]. These connected components can be used on devices for health monitoring. The used sensors then forward the information to distant locations like M2M, which are machinery for computers, machines for people, handheld devices, or smartphones. It is a simple, energy-efficient, much smarter, scalable, and interoperable way of tracking and optimizing care to any health problem. Nowadays, modern systems are providing a flexible interface, assistant devices, and mental health management to lead a smart life for the human being. The major hardware components which are used here are pulse sensor, temperature sensor, BP sensor, ECG sensor, and raspberry pi. The data were collected from sensors and sent it to raspberry pi for processing and again transmitted it to IoT network.

Smart Home automation

IOT is transforming the way devices operate at home. Using IOT several home devices are automated and they can interact among themselves and with humans. Sensors are often used for automation in home appliances. Some home automation IOT projects for engineering students are automation using hand gestures, smart garage door, facial recognition door, smart alarm clock and automated blinds. Home automation lets you manage your household—e.g. lights, heating and cooling, security systems, locks, etc.—via connected devices, eliminating the need for manual controls. Tasks are performed in real time based on sensory data, pre-set schedules or a prompt from your device.

For homeowners, it means less worry, and a safe and convenient lifestyle. As a basic example, your lights could turn on whenever someone enters the room (sensory data) or at a specific time of day (pre-set schedule). In either case, you

didn't need to flip the light switch. The system intelligently knew what to do based on settings you configured, and executed accordingly. System triggers and settings for home automation are typically managed within an online portal that's accessible via your desktop or mobile devices. This gives you remote control of your home, at anytime and from anywhere. All that's required is an Internet connection, and the infrastructure—sensors, cameras, connected devices, etc.—in place to perform.

Smart car Navigation:

As with all stages of technology, motor vehicles take up an important place in our lives and but also in smart car internet of things too. Motor cars, for example, and their effects on our lives. Generally, almost most fossil fuels were used and humans were more involved in evolution in terms of savings and terms of the ecoregion, which depending on the level of technology. New level cars reduce accident risks to zero and provide comfort in terms of safe driving are among the priority targets for car users.

What will be the benefits of smart car technology? For example, vehicles connected with the IoT system will report to the user the condition of the car, such as the amount of gasoline, general maintenance, road map, etc. This will result in significant savings and efficiency in terms of time and many more itself before the user leaves the house, and will also be able to set a special tracking distance for the vehicle user in constant communication with other vehicles to reduce the risk of accidents while cruising. Some of the most successful companies using this technology today for beneficial cruising such as Mercedes, Google, Toyota, and BMW, and they are making revolutionary progress in history. Google, in particular, has developed driverless vehicles with special technology.

With the vehicle tracking system, the location information received from GPS satellites, which is usually via units or units put into the vehicles, is transmitted to the main centre via GSM/GPRS via sensors on the vehicle and providing the users a continuous exchange of information over the internet. The information that comes to the centre is compiled with the software on the device and processed and recorded in the database. Car owners can monitor their vehicles from any point with internet access by creating a special user name and password for them and have access to detailed information about their vehicles and their fleets.

Industrial IoT

Automation has been extensively used in all the industries to reduce the manual errors in the production process. Using IOT devices, industries will be able to control and monitor various equipment, machinery, processes and other applications with less or no human intervention. Automation helps in reducing the errors and improving the efficiency of production. Thus, many industries are trying to adapt IOT system into their operations. Some industries also offer IOT

training to their employees to get knowledge on how to handle IOT systems and devices. Some of the industrial automation IOT projects for final year students are vehicle simulation, smart parking system, smart building project, biometrics and smart security model. IOT can help to perform any desired action like controlling a device or monitoring from a remote location. This technology makes equipment and machinery more digitized and connected. This technology is preferred even by the government to achieve better energy efficiency, a cleaner city and higher productivity.

Smart city IoT applications:

Aim to ensure that citizens live in maximum comfort and resource consumption is made wisely. It aims to reduce and ultimately eliminate traffic density, air pollution, polluted water resources, garbage and waste problems, population agglomeration, and crime rates.

In short, the goal of smart city IoT applications is basically to put an end to all problems that endanger human safety, health and well-being. Smart cities that solve the traffic problem with smart traffic lights or end the dirty water problem with clean water projects get very efficient results.

Smart Farming IoT Applications:

To understand smart farming IoT applications, first, let's define the concept of smart farming. Combining many advanced technologies and using them in agriculture is called smart farming. Smart farming and smart agriculture use modern informatics methods in agriculture and aim to increase productivity. Smart agriculture makes the life of both producers and farmers is much easier. IoT applications in agriculture, control of agricultural areas can be done remotely. This saves time for everyone working in the agricultural sector. The simultaneous operation of agricultural machinery is one of the factors that save time and speed.

The use of IoT based applications in agriculture is also an action aimed at protecting the environment. With the spread of smart farming practices all over the world, it is aimed to prevent problems such as water scarcity and drought in time. It is aimed to reduce the chemical products used in agriculture and thus to produce healthier products. Thus, the cost of such chemical products will be eliminated and savings will be provided. IoT based applications in agriculture, which ensure that each natural resource is used only in the required amount, aim to avoid waste.

Smart Grids IoT Applications:

It is aimed to establish mutual electronic communication between the supplier and the consumer through smart grids. Smart grids IoT and its applications work intertwined with each other. Smart grids IoT applications are encountered in many fields especially in the energy systems. It is aimed to monitor a more reliable, quality and safe process.

Wearables IoT Applications:

Wearable IoT applications are mainly used in the health and fitness sectors. The wearable devices manufactured with IoT technology, it is possible to make measurements of people's body, disease follow-up and many other measurements at any time. The number of wearables IoT applications that are causing serious and positive changes especially in the health sector is increasing day by day.

Software Defined Networking for IoT

Now let us understand some basic networking devices and their working :

Hub: Forwards the packet to every device that is connected to its network. It does not have an intelligence purpose. It is part of the physical layer (layer 1) of the OSI model.

Switch: Used to receive the data and transfer them to the destination. The switch has an intelligence purpose. It forwards the packets based on the destination address. It is a part of data link layer (layer 2).

Router: Forwards the packet between the same or different network. It is a network layer device (layer 3). A router uses IP address.

Now let us see the various networking planes present in general :

Data Plane: Data plane is used to transfer the data between the devices. It deals with the entire processes involved in forwarding the packets from source to destination. It is also referred to as forwarding plane , user plane and carrier plane .

Control Plane: As the name suggests, it controls the movement of data. It deals with functions and processes that help determine the best path to transfer the data. Routing protocols are used in this plane to discover the devices on the network and understand network topology.

Management Plane: Its prime function is to control, monitor and manage the devices, and carry the administrative traffic. Protocols such as SNMP can be used for such operations.

What Is Software Defined Network (SDN Technology)?

SDN work mainly on the Control and data plane (it separates both the planes). It employs the virtualization techniques in the networking space. SDN technology uses a smart controller to manage and monitor traffic through various Routers and switch combinations and further make the use of the dynamic routing of traffic efficiently. SDN architecture is built using a combination of software and hardware that separates the SDN control plane and the SDN data plane of the network.

1. Abstract the Hardware: No dependence on physical infrastructure. Software API. Use network resource without worrying about where it is physically located, how much it is, how it is organized.

2. Programmable: Shift away from static manual operation to fully configurable and dynamic

3. Centralized Control of Policies.

4. Should be able to control and manage thousands of devices with one command.

5. Should be able to change behaviour on the fly.

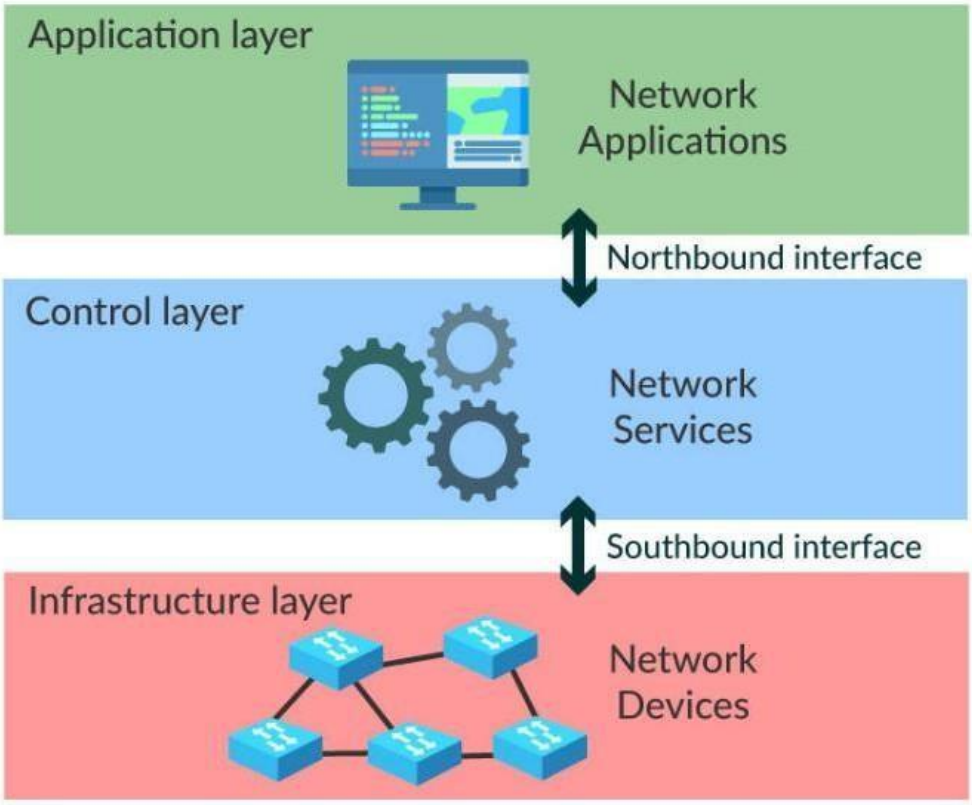
6. Automation: To lower OpEx minimize manual involvement
- Troubleshooting
- Reduce downtime
- Policy enforcement
- Provisioning/Re-provisioning/Segmentation of resources Add
- new workloads, sites, devices, and resources.

7. Visibility: Monitor resources, connectivity

8. Performance: Optimize network device utilization
- Traffic engineering/Bandwidth management Capacity
- optimization
- Load balancing
- High utilization
- Fast failure handling.

What is the Software Defined Network (SDN) Architecture ?

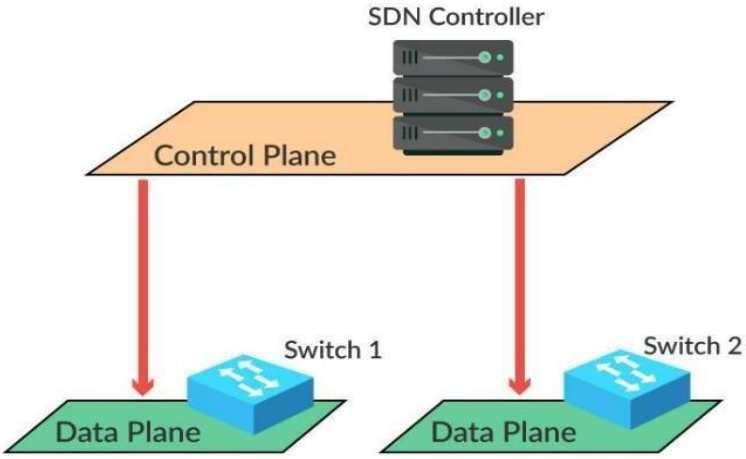
As stated earlier, SDN basically work to make the control and data plane separate and well segregated in true sense. Let us now look at its various layers:



Infrastructure Layer: It consists of network devices. This layer will be the physical layer.

Control Layer: Network Intelligence resides in the control layer. Control plane has the control logic for managing the network. In SDN, Controller is the brain of the network. It is the separate hardware that is hidden by Hardware abstraction layer (HAL). Flow entries of multiple devices will be manipulated by the controller.

Let us take one example, consider a bus carrying passengers in it. Here the passenger in the bus works like physical/data plane. Moreover, the controller who controls the bus (driver of the bus) works like the control plane or SDN controller.



In SDN, we have a central controller for the control plane. In the above diagram, you can see the SDN controller, which is responsible for the control plane. In SDN, switches have a data plane and no control plane. The SDN controller feeds the switches' data plane with information from its control plane. SDN basically employs a centralized controller, a single device does the configuration of the entire network. This controller has the full access and observation of everything that is happening in our network.

Northbound interface (NBI)
Southbound interface (SBI)

Southbound Interface:

To program the data plane, the SDN controller has to communicate with our network devices. This action is done through the southbound interface, which is a software interface, often an Application Programming Interface (API).

Northbound Interface:

The northbound interface is utilized to access the SDN controller. It allows the administrator to access the SDN to configure it or to retrieve information from it.

SDN focuses on the following key areas:

- Separation of data plane from the control plane.
- Centralization of the control plane.
- Standardized interfaces between the device and controller.
- Programmability of control plane by external applications.

SDN allows the user to treat all the devices equally by hiding the vendor or device specifics of the data layer, thus representing the entire data plane as a virtual abstract layer. SDN provides the flexibility to view the entire data plane infrastructure as a virtual resource that can be configured and controlled by an upper layer control plane. In an SDN architecture, the network appears as one logical device to the applications.

The control plane defines the controls and intelligence required by the data plane. The data layer hardware devices are now free from their individual control layers

and can act as a collective resource. Centralizing the control plane allows to inspect the state of the data layer and make adjustments dynamically to respond to new demands and changing conditions. The control layer provides a global view of all the network-wide resources, representing all the network devices as one virtual logical network. Control and Data layers are generally referred to as North and Southbound interfaces. As you know, Southbound interfaces from the controller communicate with lower-level hardware infrastructure and northbound interfaces communicate with business applications. Programming the control plane allows different parts of the network to communicate absolutely and gives a network flexible adoption control.

What Is OpenFlow Concept In SDN Network ? (Protocols Used In Software Defined Network Technology)

OpenFlow is the protocol that allows the SDN controllers to communicate with the forwarding plane of network devices. It is considered one of the first software-defined networking (SDN) standards. An SDN Controller is the "brain" of the SDN network.

Any device that wants to communicate to an SDN Controller must support OpenFlow protocol. Through the OpenFlow, the SDN Controller pushes down changes to the switch or router flow-table allowing network administrators to segment traffic, control flows for optimal performance, and start testing new configurations and applications.

Only one protocol is available for rule placement that is Open Flow protocol. The match fields the protocol uses is :

- Source IP
- Destination IP
- Source port
- Priority

The amount of time that the flow rule has to be kept is given by : Hard Timeout:All the rules are deleted during this time in the switch

- This can be used to reset the switch

Soft Timeout:

- If no flow is received associated with a rule for particular time then the rule is deleted.
- This is used to empty the rule space by deleting an unused rule.

Benefits of OpenFlow Protocol in any SDN Network Programmability:

- Programmability enables innovation/differentiation.
- Accelerate new features and services introduction.

Centralized Intelligence:

- Simplify provisioning.

- Optimize performance.

Abstraction:

- Decoupling of Hardware and Software,
- Control plane and forwarding
- Physical and logical configuration.

Controller Placement:

- Controllers specify the flow rule based on the application specific requirements.
- The controllers must be able to handle all incoming requests from the switches.
- Rule should be placed without incurring much delay.
- Typically, a controller can handle 200 request in a second.
- The controllers are logically connected to the switches in one Hop - distance.
- If we have a very small number of controllers for a large network, the network might be congested with control packets.

So, typically a controller can handle roughly about 2 hundred requests in a second and that is applicable for only the single threaded applications thing.
So, the controllers which are implementing similar thread, but currently multi threaded applications in controllers are also possible. The controllers are logically connected to the switches in a one-hop distance it is just a logical connection. So, the controller from the switch the switch thinks that the controller is away just a hop distance from it just one hop from it, but actually it is not so; actually you know physically when we are talking about the controller and the switch can be multiple hops away and typically they are multiple hops away.

Control mechanism:

1. Distributed
2. Centralized

In the distributed the control divisions can be taken in a distributed manner. For example, each sub network is controlled by a different controller. And in the centralized mechanism the control decisions are taken in a centralized manner. For example, a network is controlled by a single controller. If the controller is down so, in such a case it is a centralized control and dividing into different sub networks are having a controller corresponding to it will be distributed solution there is also a concept of backup controller. So, if the primary controller is down then the backup controller takes over. So, backup controller has a replica of the main controller and if the main controller is down the backup controller controls the network to have uninterrupted network management another very important thing is true in SDN one can have enhanced level of security in the networking.

SDN for IoT:

SDN is useful to manage and control IoT network

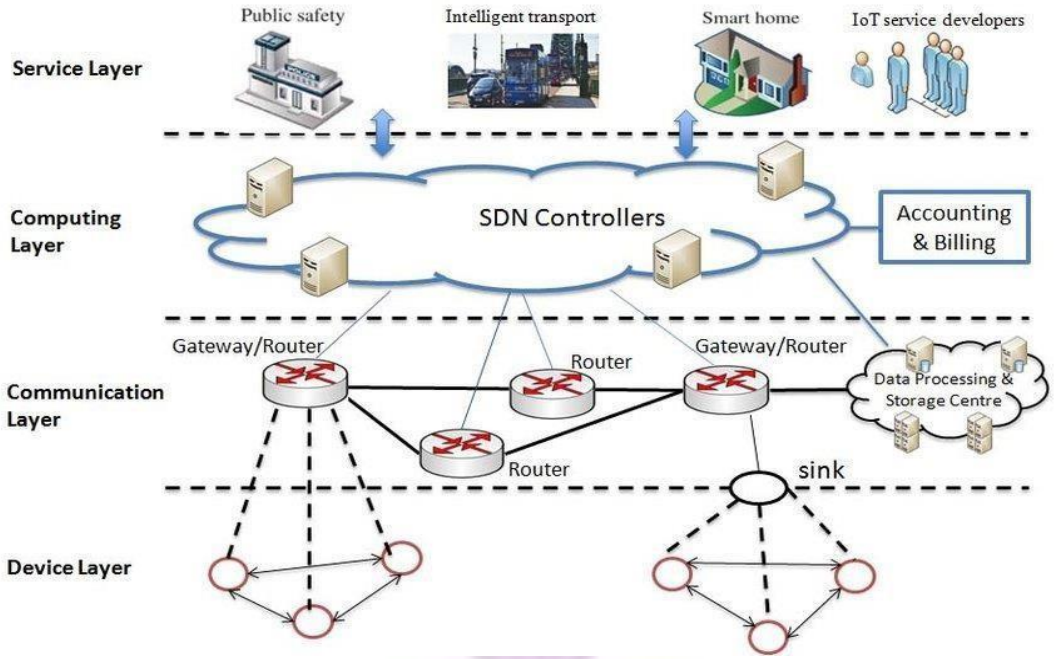
* Wireless sensor nodes (WSN) and network can be controlled using SDN-based applications.

* Network performance can be improved significantly using SDN-based approaches over the traditional approaches.

Benefits of Integrating SDN :

- Intelligence routing decisions can be deployed using SDN
- Simplification of information collection, analysis and decision making
- Visibility of network resources - network management is simplified based on user, device and application - specific requirements
- Intelligent traffic pattern analysis and coordinated decision

SDN for IoT Architecture:



Challenges

- Real-time programming of sensor nodes
- Vendor-specific architecture
- Resource constrained - heavy computation cannot be performed
- Limited memory - cannot insert too many control programs

Opportunities

- Can we program the sensor nodes in real-time?
- Can we change the forwarding path in real-time?
- Can we integrate different sensor nodes in a WSN?

Sensor OpenFlow

→ Value-centric data forwarding

- Forward the sensed data if exceeds a certain value.

→ **ID-centric data forwarding**

- Forward the sensed data based on the ID of the source node.

Sensor management

- Multiple sensors can be implemented in a single sensor board.
- Sensors can be used depending on application-specific requirements.

→ **Delay management**

- Delay for sensing can be changed dynamically in real-time.

→ **Active-Sleep Management**

- States of active and sleep mode can be changed dynamically.

Soft-WSN

Topology Management

- **Node-specific management** - forwarding logic of a particular sensor can be modified

- **Network-specific management**

- Forward all traffic of a node in the network.
- Drop all traffic of a node in the network.

Rule Placement challenges:

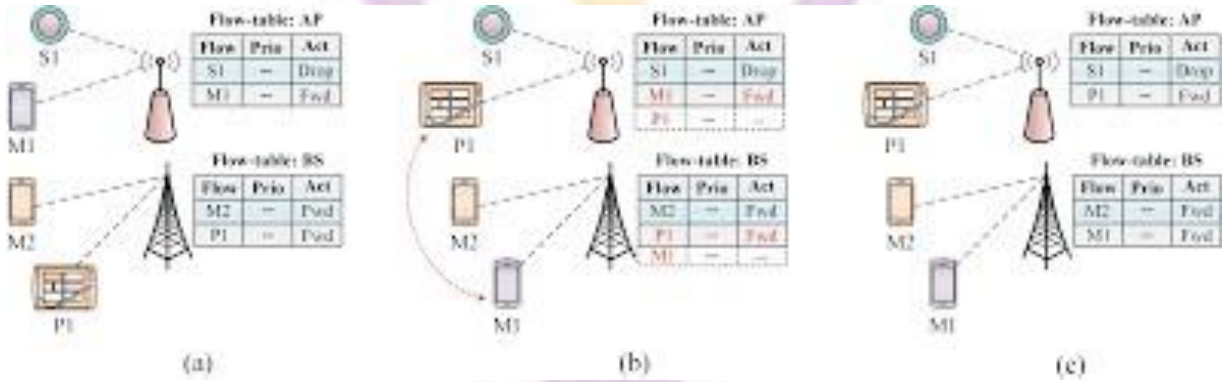
- Typical users are mobile users with the dynamic network.
- It requires continues in rule placement is required.
- How to support such heterogenous devices in such a single platform.

SDN for Mobile Networks for IoT(SDIoT):

So, basically you know what happened at that last point, what it says the last challenge what it says is different types of heterogeneous devices having different configurations supporting different you know vendor specific solutions and protocols and so on exist in a realistic mobile based IoT network. So, how to support such heterogeneous devices in a single platform? So, these are the challenges that have to be worked upon in order to deal with the rule placement issue. So, there are different approaches or different solutions that have been proposed for it.

So, Mobi-Flow basically gives a mobility aware flow, flow rule placement for mobile based IoT solutions, mobile place based IoT systems. So, let us look at these three scenarios over here. So, here we have different nodes S1 M1 connected to this particular access point. M2 and P1 connecting to another access point and their corresponding flow tables are shown here, the corresponding flows tables corresponding to this particular

access point and this particular access point of the base station, the corresponding flow table is given over here. It is very easy to understand this part. Now, let us say that the nodes M1 and P1 have interchanged their positions. Let us assume a very simple scenario, where M1 and P1 have interchanged their positions and the current situation looks like this. So, essentially you know what should happen is, we will have something like this. The another flow rule has to be added in the flow table corresponding to this particular access point and the base station in this particular manner, but at the same time we also know that flow table spaces are very limited. So, we cannot simply have these flow tables grow with the addition of these flow rules. So, you know what essentially has to be done is the scenario like this. So, how can we do it, this is what Mobi-Flow talks about.



So, Mobi-Flow gives a proactive rule placement scheme depending on the users movement in the network and when we talk about such a scheme, you know what is very important to have some kind of prediction of the location of the end users at the next time instant give you the data of how the user was moving till a particular time instant. And then, placing the flow rules at the access points which can be associated to the users based on the predicted locations, location prediction, this is done in Mobi-Flow using something known as the Order-K Markov Predictor which basically takes the last k^{th} location instances to predict the next location and the flow rule placement which basically is a linear programming- based solution that is used to select which access point is going to be the optimal one. So, what we have is with respect to message overhead and energy consumption, Mobi- Flow performs better compared to the conventional network, conventional solution and this is quite evident from these two plots. So, control message overhead and energy consumption can be minimized significantly using Mobi-Flow compared to the conventional rule placement schemes.

So, next we have to take care of how to perform rule placement at the backbone network. So, the existing rule placement schemes for wired networks can be used over here because most of the backbone networks are have a topology and the structure. The overall architecture is similar to that which exists for wired networks. So, we can use the existing rule placement

schemes for wired networks over here as well for the backbone network of IoT and the load balancing is an important issue due to the dynamic nature of the IoT network. So, dynamic resource allocation can also be integrated.

Software-Defined Networking for Data Handling:

So, another very important thing that we have to remember in this particular context is the Data Center Networking. So, there are two types of flows that occur in Data Center Networks. So, Data Center Networks basically you know here we are talking about implementation of data center networks with the help of SDN. So, typically two different solutions of adoption of SDN are in data center networks. So, here we have typically two types of flows that are observed. One is the Mice-Flow which are basically you know small flows and the other one is Elephant-Flow which is basically you know large scale flows where big volumes of data are coming, big sized data are coming and so on and small flows, Mice- Flows where small sized data are coming. So, here what is suggested is for the Mice-Flow wildcard rules can be placed to deal with these flows, the wildcard rules and for the Elephant-Flows what is required is to have exact match of the rules.

So, we need to classify the flows before inserting the flow rules at the switches to adequately forward them in the network. So, if it is an elephant flow, you know we have to have exact match. If it is a Mice-Flow, we will go for a wildcard match. Anomaly detection can also be performed in IoT network using SDN or open flow. So, here you know it is required to monitor the network through open flow to detect any anomaly in the network which can be done by monitoring each flow in the network. It is also possible to collect different port statistics at the different switches and thereby from these statistics, you know anomaly detection techniques can be implemented and different anomalies can be found out.

What is Data?

The quantities, characters, or symbols on which operations are performed by a computer, which may be stored and transmitted in the form of electrical signals and recorded on magnetic, optical, or mechanical recording media.

What is Big Data?

Big Data is a collection of data that is huge in volume, yet growing exponentially with time. It is a data with so large size and complexity that none of traditional data management tools can store it or process it efficiently. Big data is also a data but with huge size.

Types Of Big Data

Following are the types of Big Data:

Structured
Unstructured
Semi-structured

Structured:

Any data that can be stored, accessed and processed in the form of fixed format is termed as a 'structured' data. Over the period of time, talent in computer science has achieved greater success in developing techniques for working with such kind of data (where the format is well known in advance) and also deriving value out of it. However, nowadays, we are foreseeing issues when a size of such data grows to a huge extent, typical sizes are being in the rage of multiple zettabytes. 10^{21} bytes equal to 1 zettabyte or one billion terabytes forms a zettabyte.

Examples Of Structured Data

An 'Employee' table in a database is an example of Structured Data.

Unstructured:

Any data with unknown form or the structure is classified as unstructured data. In addition to the size being huge, un-structured data poses multiple challenges in terms of its processing for deriving value out of it. A typical example of unstructured data is a heterogeneous data source containing a combination of simple text files, images, videos etc. Now day organizations have wealth of data available with them but unfortunately, they don't know how to derive value out of it since this data is in its raw form or unstructured format.

Examples Of Un-structured Data

The output returned by 'Google Search'.

Semi-structured

Semi-structured data can contain both the forms of data. We can see semi- structured data as a structured in form but it is actually not defined with e.g. a table definition in relational DBMS. Example of semi-structured data is a data represented in an XML file.

Examples Of Semi-structured Data Personal data stored in an XML file- **Characteristics Of Big Data:**

Big data can be described by the following characteristics:

Volume
Variety
Velocity
Variability

(i) Volume – The name Big Data itself is related to a size which is enormous. Size of data plays a very crucial role in determining value out of data. Also, whether a particular data can actually be considered as a Big Data or not, is dependent upon the volume of data. Hence, 'Volume' is one characteristic which needs to be considered while dealing with Big Data.

(ii) Variety – The next aspect of Big Data is its variety.

Variety refers to heterogeneous sources and the nature of data, both structured and unstructured. During earlier days, spreadsheets and databases were the only sources of data considered by most of the applications. Nowadays, data in the form of emails, photos, videos, monitoring devices, PDFs, audio, etc. are also being considered in the analysis applications. This variety of unstructured data poses certain issues for storage, mining and analysing data.

(iii) Velocity – The term 'velocity' refers to the speed of generation of data. How fast the data is generated and processed to meet the demands, determines real potential in the data.

Big Data Velocity deals with the speed at which data flows in from sources like business processes, application logs, networks, and social media sites, sensors, Mobile devices, etc. The flow of data is massive and continuous.

(iv) Variability – This refers to the inconsistency which can be shown by the data at times, thus hampering the process of being able to handle and manage the data effectively.

Data handling at Data centres:

- Storing, organizing and managing data.
- Estimates and provides necessary processing capacity.
- Provides sufficient network infrastructure
- Effectively manages power consumption.
- Replicates data to keep backup.
- Helps business personal to analyze data.
- Discovers problems in business operations.

Data storage strategies

The data can be stored on the premises, somewhere on the cloud, or based on a strategy that's a hybrid of these two. The important considerations in deciding the right data-storage strategy are volume of data, the connectivity to the network, and power availability.

For example, aircrafts have intermittent connectivity to the base stations, so they need on-premise storage mechanisms for critical data. Similarly, a device that's prone to power outages will require on-premise non-volatile storage to recover the critical data when power is back or the device is recovered manually (like flight recorders). On the other hand, devices like temperature sensors having continuous

connectivity and power may relay everything to the cloud and need no on-premise storage at all.

Another thing to consider is that data is destined for different purposes. Data that is intended for archival purposes rather than real-time analytics can be stored using different approaches that complement the analytics approach that will be adopted. Data access needs to be fast and support querying for discrete real-time data analytics.

There's also an emerging trend of "edge computing" to offload pre-processing of data on-premise before actually pushing it to the cloud.

Data storage technologies

Data storage technologies that you use for data stored temporarily at the edge while it is being processed on devices and gateways will need to be physically robust (to withstand the often harsh operating environments in which devices are installed), but also fast and reliable so that pre-processing of data can be performed quickly and accurately before data is sent upstream.

For real-time analytics like applications, storage technologies adopted should support concurrent reads and writes, be highly available, and include indexes that can be configured to optimize data access and query performance.

High volume archival data can be stored on cloud storage that may be slower to access but it has the advantage of being lower cost and elastic so it will scale as the volume of data increases. Access to large file format data like video is usually sequential, so this data might be stored using object storage (such as OpenStack Object Storage known as Swift), or written directly to a distributed file system like Hadoop HDFS or IBM GPFS. Data warehouse tools like Apache Hive can assist with managing reading, writing, and accessing data that is stored on distributed storage.

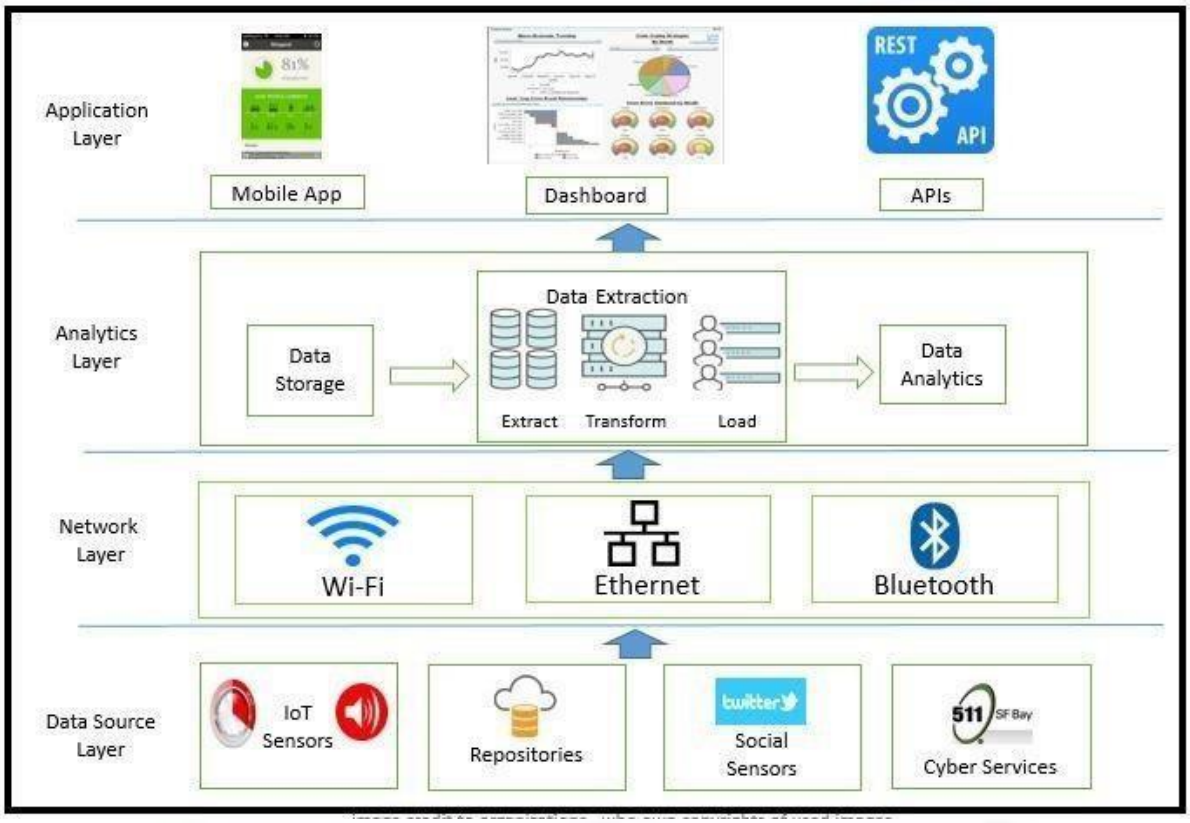
Data storage technologies that are often adopted for IoT event data include NoSQL databases and time series databases.

NoSQL databases are a popular choice for IoT data used for analytics because they support high throughput and low latency, and their schema- less approach is flexible, allowing new types of data to be added dynamically. Open source NoSQL databases used for IoT data include Couchbase, Apache Cassandra, Apache CouchDB, MongoDB and Apache HBase, which is the NoSQL database for Hadoop. Hosted NoSQL cloud storage solutions include IBM's Cloudant database and AWS DynamoDB.

Time series databases can be relational or based on a NoSQL approach. They are designed specifically for indexing and querying time-based data, which is ideal for most IoT sensor data, which is temporal in nature. Time series databases used for IoT data include InfluxDB, OpenTSB, Riak, Prometheus and Graphite.

WHAT IS DATA ANALYTICS?

Data Analytics is the science (and art!) of applying statistical techniques to large data sets to obtain actionable insights for making smart decisions. It is the process to uncover hidden patterns, unknown correlations, trends and any other useful business information.



Analysing data

IoT data needs to be analysed in order to make it useful, but manually processing the flood of data produced by IoT devices is not practical. So, most IoT solutions rely on automated analytics. Analytics tools are applied to the telemetry data to generate descriptive reports, to present data through dashboards and data visualizations, and to trigger alerts and actions.

There are many different open-source analytics frameworks or IoT platforms that can be used to provide IoT data processing and analytics to your IoT solutions. Analytics can be performed in real-time as the data is received or through batch processing of historical data. Analytics approaches include distributed analytics, real-time analytics, edge analytics and machine learning.

Distributed analytics

Distributed analytics is necessary in IoT systems to analyse data at scale, particularly when dealing with historical data that is too vast to be stored or processed by a single node. Data can be spread across multiple databases; for example, device data might be bucketed into databases for each device per time

period, such as hourly, daily, or monthly, like the IBM Watson IoT Historian Service that connects to Cloudant NoSQL database that stores the IoT data. Analytics may involve aggregating results which are distributed across multiple geographical locations. You'll want to adopt a storage driver or analytics framework that bridges distributed storage and compute infrastructure to allow seamless querying across distributed databases. Of particular note for processing of distributed data are the ecosystem of frameworks arising from the Hadoop community. Apache Hadoop is a batch processing framework that uses a MapReduce engine to process distributed data. Hadoop is very mature and was one of the first open-source frameworks to take off for big data analytics. There's also Apache Spark, which was started later with an intention to improve on some of the weak points of Hadoop. Hadoop and Spark are ideal for historical IoT data analytics for batch-processing where time sensitivity is not an issue, such as performing analysis over a complete set of data and producing a result at a later time.

Real-time analytics

Analytics for high-volume IoT data streams is often performed in real-time, particularly if the stream includes time-sensitive data, where batch processing of data would produce results too late to be useful or any other application where latency is a concern.

Real-time analytics are also ideal for time series data, because unlike batch processing, real-time analytics tools usually support controlling the window of time analysis, and calculating rolling metrics, for example, to track hourly averages over time rather than calculating a single average across an entire dataset.

Frameworks that are designed for real-time stream analytics include Apache Storm and Apache Samza (usually used with Kafka and Hadoop YARN). Hybrid engines that can be used for either stream or batch analytics include Apache Apex, Apache Spark, and Apache Flink. Apache Kafka acts as an ingestion layer that can sit over the top of an engine like Spark, Storm or Hadoop. For a guide to selecting between these open source frameworks, read Choosing the right platform for high- performance, cost-effective stream processing applications.

Edge analytics

IoT analytics is not usually applied to raw device data. The data is pre-processed to filter out duplicates or to re-order, aggregate or normalize the data prior to analysis. This processing typically occurs at the point of acquisition, on the IoT devices themselves or on gateway devices that aggregate the data, to determine which data needs to be sent upstream.

Analytics applied at the edges of the network, as close as possible to the devices generating the data is known as edge analytics. Linux Foundation's Edge X Foundry, an open source IoT edge computing framework, also supports edge analytics.

Edge analytics is low-latency and reduces bandwidth requirements because

not as much data needs to be transmitted from the device. However, constrained devices have limited processing capacity, so most IoT solutions use a hybrid approach involving edge analytics and upstream analytics.

Machine learning

Using traditional mathematical statistical models for analytics provides value as they can be used to track goals, create reports and insights, predict trends, and create simulations that are used to predict and optimize for specific outcomes. For example, you can predict the outcome of applying a specific action, predict the time to failure for a given piece of equipment, or optimize the configuration of an IoT system in terms of cost or performance.

However, the value of statistical analytics models diminishes when applied to dynamic data that contains many variables that change over time, when you don't know what factors to look for, or what variables to change to achieve a desired outcome like reducing cost or improving efficiency. In these cases, instead of using a statistical model, machine learning algorithms that learn from the data can be applied.

Machine learning can be applied to historic or real-time data. Machine learning techniques can be used to identify patterns, identify key variables and relationships between them to automatically create and refine analytics models, and then use those model for simulations or to produce decisions. Machine learning approaches have the advantage over static statistical analytics models that as new data comes in, the models can be improved over time, which leads to improved results. The state of the art Machine learning techniques mostly come in the domain of Deep learning using neural networks (Convolutional Neural Networks, Long Short Term Memory networks). Emerging and promising areas of research include Active learning, Multi modal and Multi-Task learning, and Transformer- based language models. That being said, the traditional machine learning methods like Regression, Support Vector Machines, and Decision trees can still prove to be effective in a lot of applications.

Unit -V

Cloud Computing

Introduction to Cloud Computing

Cloud Computing is the delivery of computing services such as servers, storage, databases, networking, software, analytics, intelligence, and more, over the Cloud (Internet). Cloud Computing provides an alternative to the on-premises datacentre. With an on-premises datacentre, we have to manage everything, such as purchasing and installing hardware, virtualization, installing the operating system, and any other required applications, setting up the network, configuring the firewall, and setting up storage for data. After doing all the set-up, we become responsible for maintaining it through its entire lifecycle.

But if we choose Cloud Computing, a cloud vendor is responsible for the hardware purchase and maintenance. They also provide a wide variety of software and platform as a service. We can take any required services on rent. The cloud computing services will be charged based on usage. The cloud environment provides an easily accessible online portal that makes handy for the user to manage the compute, storage, network, and application resources.

Cloud computing means that instead of all the computer hardware and software you're using sitting on your desktop, or somewhere inside your company's network, it's provided for you as a service by another company and accessed over the Internet, usually in a completely seamless way. Exactly where the hardware and software is located and how it all works doesn't matter to you, the user—it's just somewhere up in the nebulous "cloud" that the Internet represents. Cloud computing is named as such because the information being accessed is found remotely in the cloud or a virtual space. Companies that provide cloud services enable users to store files and applications on remote servers and then access all the data via the Internet. This means the user is not required to be in a specific place to gain access to it, allowing the user to work remotely.

Types of Cloud Services

Regardless of the kind of service, cloud computing services provide users with a series of functions including:

- Email
- Storage, backup, and data retrieval
- Creating and testing apps
- Analyzing data
- Audio and video streaming
- Delivering software on demand

Cloud computing is still a fairly new service but is being used by a number of different organizations from big corporations to small businesses, non- profits to government agencies, and even individual consumers.

Advantages of cloud computing

Cost: It reduces the huge capital costs of buying hardware and software.
Speed: Resources can be accessed in minutes, typically within a few clicks.
Scalability: We can increase or decrease the requirement of resources according to the business requirements.

Productivity: While using cloud computing, we put less operational effort. We do not need to apply patching, as well as no need to maintain hardware and software. So, in this way, the IT team can be more productive and focus on achieving business goals.

Reliability: Backup and recovery of data are less expensive and very fast for business continuity.

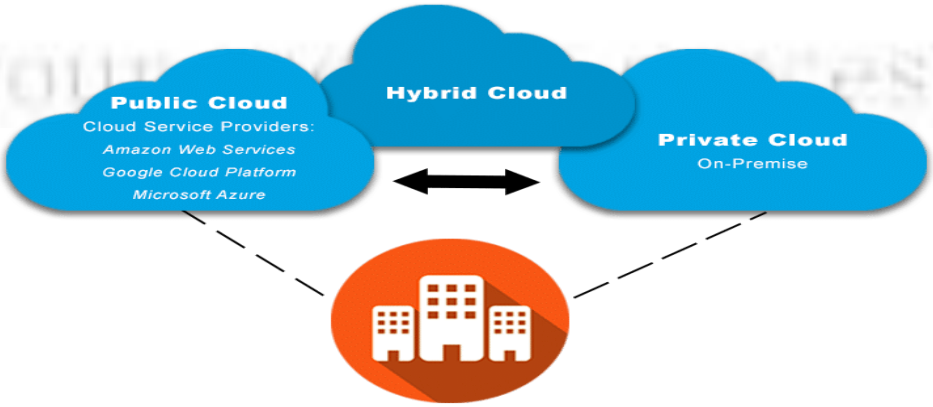
Security: Many cloud vendors offer a broad set of policies, technologies, and controls that strengthen our data security.

Types of Cloud Computing:

Public Cloud: The cloud resources that are owned and operated by a third- party cloud service provider are termed as public clouds. It delivers computing resources such as servers, software, and storage over the internet

Private Cloud: The cloud computing resources that are exclusively used inside a single business or organization are termed as a private cloud. A private cloud may physically be located on the company's on-site datacentre or hosted by a third-party service provider.

Hybrid Cloud: It is the combination of public and private clouds, which is bounded together by technology that allows data applications to be shared between them. Hybrid cloud provides flexibility and more deployment options to the business.

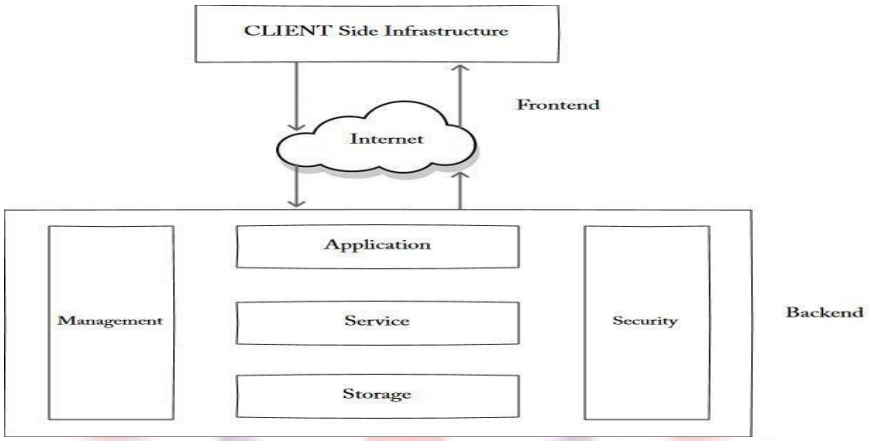


Cloud Computing Architecture:

Cloud computing architecture is divided into the following two parts -

- Front End
- Back End

The below diagram shows the architecture of cloud computing –



Front-end corresponds to the client-side of cloud computing. This component is all about interfaces, applications and network those allow accessibility for a cloud system. The thing that has to be clear in this aspect is that not all the entire computing systems will work as a single interface.

Whereas back-end corresponds to the resources utilized by cloud computing servers. This component mainly deals with servers, security scenarios, virtualizing, data storage and many others. Also, back-end holds the responsibility to reduce traffic congestion mechanisms, and protocols that establish communication. Here, the operating system is termed as the bare metal server which is prominent with the name "hypervisor" where it utilizes well-defined protocols that allow for concurrent operation of numerous guest virtual servers. Hypervisor acts as a communication interface between its containers and for the connected world.

Apart from these, cloud-based delivery and cloud services network are also considered as cloud computing architecture. Delivery of cloud services can be done either publicly or privately through the internet. In a few cases, enterprises make use of both scenarios to deliver their services.

Components of Cloud Computing Architecture

There are the following components of cloud computing architecture "

1. Client Infrastructure:

Client Infrastructure is a Front-end component. It provides GUI (Graphical User Interface) to interact with the cloud.

2. Application:

The application may be any software or platform that a client wants to

access.

3. Service:

A Cloud Services manages that which type of service you access according to the client's requirement.

4. Runtime Cloud

Runtime Cloud provides the execution and runtime environment to the virtual machines.

5. Storage:

Storage is one of the most important components of cloud computing. It provides a huge amount of storage capacity in the cloud to store and manage data.

6. Infrastructure:

It provides services on the host level, application level, and network level. Cloud infrastructure includes hardware and software components such as servers, storage, network devices, virtualization software, and other storage resources that are needed to support the cloud computing model.

7. Management:

Management is used to manage components such as application, service, runtime cloud, storage, infrastructure, and other security issues in the backend and establish coordination between them.

8. Security:

Security is an in-built back end component of cloud computing. It implements a security mechanism in the back end.

9. Internet:

The Internet is medium through which front end and back end can interact and communicate with each other.

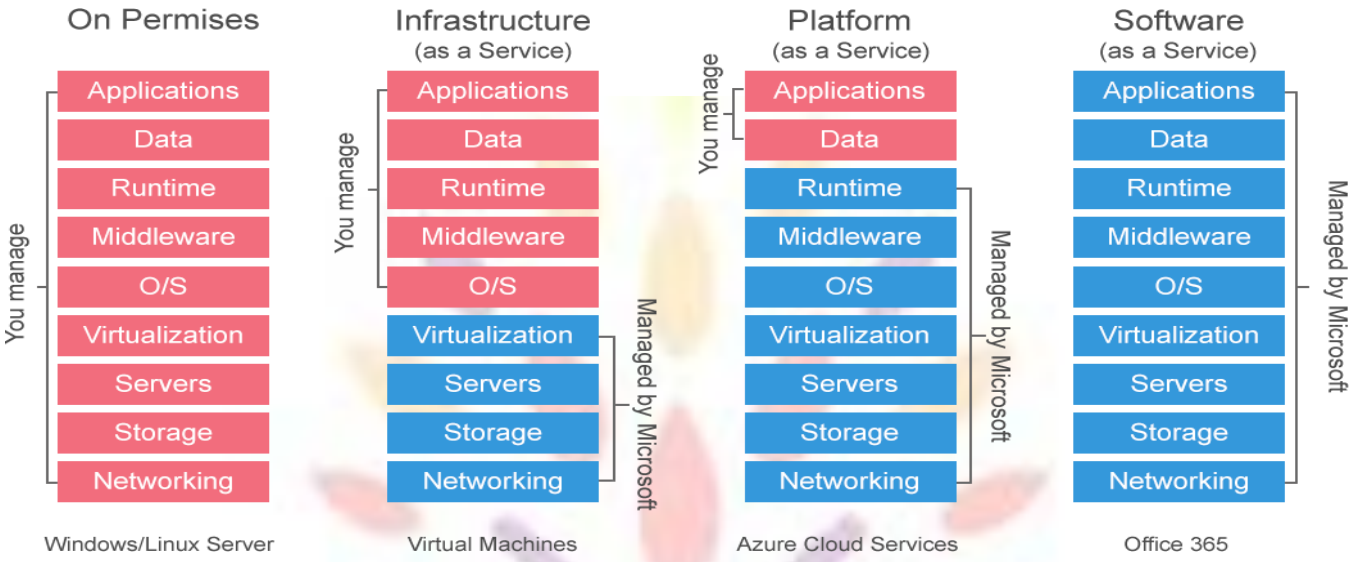
Types of Cloud Services:

Infrastructure as a Service (IaaS): In IaaS, we can rent IT infrastructures like servers and virtual machines (VMs), storage, networks, operating systems from a cloud service vendor. We can create VM running Windows or Linux and install anything we want on it. Using IaaS, we don't need to care about the hardware or virtualization software, but other than that, we do have to manage everything else. Using IaaS, we get maximum flexibility, but still, we need to put more effort into maintenance.

Platform as a Service (PaaS): This service provides an on-demand environment for developing, testing, delivering, and managing software applications. The developer is responsible for the application, and the PaaS vendor provides the ability to deploy and run it. Using PaaS, the flexibility gets reduce, but the management of the environment is taken care of by the cloud vendors.

Software as a Service (SaaS): It provides a centrally hosted and managed software services to the end-users. It delivers software over the internet, on-

demand, and typically on a subscription basis. E.g., Microsoft One Drive, Dropbox, WordPress, Office 365, and Amazon Kindle. SaaS is used to minimize the operational cost to the maximum extent.



How are IoT and Cloud Computing Related?

The IoT and Cloud Computing complement one another, often being branded together when discussing technical services and working together to provide an overall better IoT service. However, there are crucial differences between them, making each of them an effective technical solution separately and together.

Cloud Computing in IoT works as part of a collaboration and is used to store IoT data. The Cloud is a centralised server containing computer resources that can be accessed whenever required. Cloud Computing is an easy travel method for the large data packages generated by the IoT through the Internet. Big Data can also help in this process. Combined, IoT and Cloud Computing allow systems to be automated in a cost-effective way that supports real-time control and data monitoring.

The Benefits of Using Big Data, IoT and the Cloud:

So why are Big Data, IoT and the Cloud such a good partnership? Well, there are numerous benefits for utilising both of these services by combining them, with a few of the main benefits outlined below:

Scalability for device data:

Cloud-based solutions can be scaled vertically and horizontally to meet the needs of Big Data hosting and analytics. For example, you can increase a server's capacity with more applications, or expand your hardware resources when necessary. The Cloud enables the expansion of Big Data and data analytics.

Scalable infrastructure capacity:

Big Data and Cloud Data can be used in conjunction to store large amounts of data and provides scalable processing and improved real-time analysis of data. The lack of physical infrastructure needed to get Big Data, IoT and the Cloud up and running together reduces costs and means you can focus on the improved analytical capacity rather than worry about maintenance and support.

Increased efficiency in daily tasks:

IoT and Big Data generate a large amount of data, which the Cloud provides the pathway for the data to travel.

Quicker use and distribution of Apps worldwide:

You can access Big Data remotely and easily from anywhere in the world to still carry out actions on devices when using the Cloud, allowing for better collaboration.

Sensor Cloud:

The advancement and application of wireless sensor networks become an invincible trend into the various industrial, environmental, and commercial fields. A typical sensor network may consist of a number of sensor nodes acting upon together to monitor a region and fetch data about the surroundings. A WSN contains spatially distributed self-regulated sensors that can cooperatively monitor the environmental conditions, like sound, temperature, pressure, motion, vibration, pollution, and so forth. Each node in a sensor network is loaded with a radio transceiver or some other wireless communication device, a small microcontroller, and an energy source most often cells/battery. The nodes of sensor network have cooperative capabilities, which are usually deployed in a random manner. These sensor nodes basically consist of three parts: sensing, processing, and communicating. Some of the most common sensor devices deployed in sensor network as sensor nodes are camera sensor, accelerometer sensor, thermal sensor, microphone sensor, and so forth.

Currently, WSNs are being utilized in several areas like healthcare, defence such as military target tracking and surveillance, government and environmental services like natural disaster relief, hazardous environment exploration, and seismic sensing, and so forth. These sensors may provide various useful data when they are closely attached to each of their respective applications and services directly. However, sensor networks have to face many issues and challenges regarding their communications (like short communication range, security and privacy, reliability, mobility, etc.) and resources (like power considerations, storage capacity, processing capabilities, bandwidth availability, etc.). Besides, WSN has its own resource and design constraints. Design constraints are application specific and dependent on monitored environment. Based on the monitored environment, network size in WSN varies. For monitoring a small area, fewer nodes are required to form a network whereas the coverage of a very large area requires a huge number of sensor nodes. For monitoring large environment, there is limited communication between nodes due to obstructions into the environment, which in turn affects the overall

network topology (or connectivity). All these limitations on sensor networks would probably impede the service performance and quality. In the midst of these issues, the emergence of cloud computing is seen as a remedy.

A Sensor-Cloud collects and processes information from several sensor networks, enables information sharing on big scale, and collaborates with the applications on cloud among users. It integrates several networks with a number of sensing applications and cloud computing platform by allowing applications to be cross-disciplinary that may span over multiple organizations. Sensor-Cloud enables users to easily gather, access, process, visualize, analyse, store, share, and search for a large number of sensor data from several types of applications and by using the computational IT and storage resources of the cloud.

In a sensor network, the sensors are utilized by their specific application for a special purpose, and this application handles both the sensor data and the sensor itself such that other applications cannot use this. This makes wastage of valuable sensor resources that may be effectively utilized when integrating with other application's infrastructure. To realize this scenario, Sensor-Cloud infrastructure is used that enables the sensors to be utilized on an IT infrastructure by virtualizing the physical sensor on a cloud computing platform. These virtualized sensors on a cloud computing platform are dynamic in nature and hence facilitate automatic provisioning of its services as and when required by users. Furthermore, users need not to worry about the physical locations of multiple physical sensors and the gapping between physical sensors; instead, they can supervise these virtual sensors using some standard functions.

Within the Sensor-Cloud infrastructure, to obtain QoS, the virtual sensors are monitored regularly so users can destroy their virtual sensors when they becomes meaningless. A user interface is provisioned by this Sensor-Cloud infrastructure for administering, that is, for controlling or monitoring the virtual sensors, provisioning and destroying virtual sensors,registering and deleting of physical sensors, and for admitting the deleting users. For example, in a health monitoring environment, a patient may use a wearable computing system (that may include wearable accelerometer sensors, proximity sensors, temperature sensors, etc.) like Life Shirt and Smart Shirt or may use a handheld device loaded with sensors, and consequently the data captured by the sensors may be made accessible to the doctors. But out of these computing systems, active continuous monitoring is most demanding, and it involves the patient wearing monitoring devices to obtain pervasive coverage without being inputted or intervened.

Sensor Network Architecture:

Sensor Network Architecture is used in Wireless Sensor Network (WSN). It can be used in various places like schools, hospitals, buildings, roads, etc for various applications like disaster management, security management, crisis management, etc.

There are 2 types of architecture used in WSN: Layered Network Architecture, and Clustered Architecture. These are explained as following below.

1. Layered Network Architecture:

Layered Network Architecture makes use of a few hundred sensor nodes and a single powerful base station. Network nodes are organized into concentric Layers.

It consists of 5 layers and three cross layers.
The 5 layers are:
1. Application Layer
2. Transport Layer
3. Network Layer
4. Data Link Layer
5. Physical Layer

The advantage of using Layered Network Architecture is that each node participates only in short-distance, low power transmissions to nodes of the neighbouring nodes because of which power consumption is less as compared to other Sensor Network Architecture. It is scalable and has a higher fault tolerance.

2. Clustered Network Architecture:

In Clustered Network Architecture, Sensor Nodes autonomously clubs into groups called clusters. It is based on the Leach Protocol which makes use of clusters. Leach Protocol stands for "Low Energy Adaptive Clustering Hierarchy".

Properties of Leach Protocol:

- It is a 2-tier hierarchy clustering architecture.
- It is a distributed algorithm for organizing the sensor nodes into groups called clusters.
- The cluster head nodes in each of the autonomously formed clusters create the Time-division multiple access (TDMA) schedules.
- It makes use of the concept called Data Fusion which makes it energy efficient.

Advantages of Sensor-Cloud

Cloud computing is very encouraging solution for Sensor-Cloud infrastructure due to several reasons like the agility, reliability, portability, real-time, flexibility, and so forth. Structural health and environment-based monitoring contains highly sensitive data and applications of these types cannot be handled by normal data tools available in terms of data scalability, performance, programmability, or accessibility. So a better infrastructure is needed that may contain tools to cope with these highly sensitive applications in real time. In the following, we describe the several advantages and benefits of Sensor-Cloud infrastructure that may be the cause of its glory, and these are as follows.

- Analysis: The integration of huge accumulated sensor data from several sensor networks and the cloud computing model make it attractive for various kinds of analyses required by users through provisioning of the scalable processing power.
- Scalability: Sensor-Cloud enables the earlier sensor networks to scale on

very large size because of the large routing architecture of cloud. It means that as the need for resources increases, organizations can scale or add the extra services from cloud computing vendors without having to invest heavily for these additional hardware resources.

- Collaboration: Sensor-Cloud enables the huge sensor data to be shared by different groups of consumers through collaboration of various physical sensor networks. It eases the collaboration among several users and applications for huge data sharing on the cloud.
- Free Provisioning of Increased Data storage and Processing Power: It provides free data storage and organizations may put their data rather than putting onto private computer systems without hassle. It provides enormous processing facility and storage resources to handle data of large- scale applications.
- Dynamic Provisioning of Services: Users of Sensor-Cloud can access their relevant information from wherever they want and whenever they need rather than being stick to their desks.
- Automation: Automation played a vital role in provisioning of Sensor- Cloud computing services. Automation of services improved the delivery time to a great extent.
- Flexibility: Sensor-Cloud provides more flexibility to its users than the past computing methods. It provides flexibility to use random applications in any number of times and allows sharing of sensor resources under flexible usage environment.
- Agility of Services: Sensor-Cloud provides agile services and the users can provision the expensive technological infrastructure resources with less cost. The integration of wireless sensor networks with cloud allows the high- speed processing of data using immense processing capability of cloud.

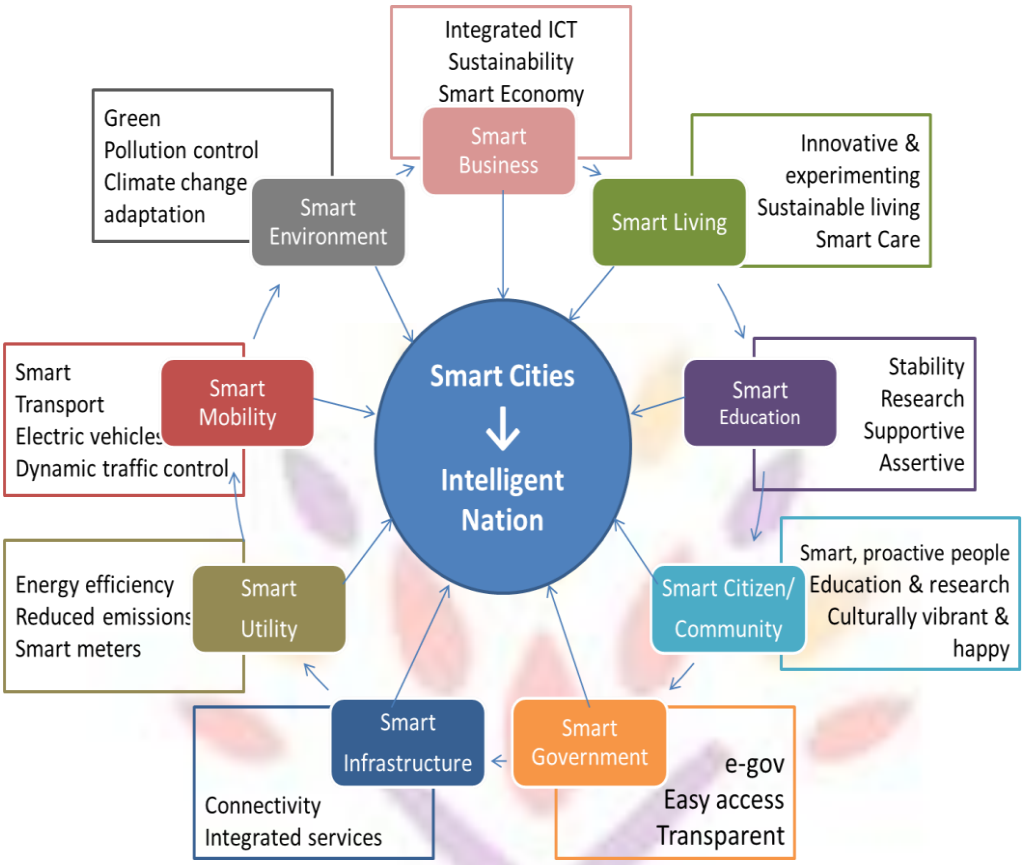
- Resource Optimization: Sensor-Cloud infrastructure enables the resource optimization by allowing the sharing of resources for several number of applications. The integration of sensors with cloud enables gradual reduction of resource cost and achieves higher gains of services. With Sensor-Cloud, both the small and midsized organizations can benefit from an enormous resource infrastructure without having to involve and administer it directly.

Smart Cities:

The urbanization process has greatly improved people's standard of living, providing water supplies and sewerage systems, residential and office buildings, education and health services and convenient transportation. The concentration of educated people in cities helps to improve the industrial structure and promote production efficiency. However, urbanization also creates new challenges and problems. As a representative developing country, the economic advantages of Indian cities are being offset by the perennial urban curses of overcrowding, air and water pollution, environmental degradation, contagious diseases and crime; the urban issues of reducing air pollution and providing clean water, safe neighborhoods and efficient infrastructure desperately need to be addressed.

All these challenges and problems force citizens, governments and

stakeholders to pay attention to the environment and sustainable development of cities, and to try to find a set of technical solutions to reduce these urban problems. The Information and Communication Technology (ICT) revolution has offered people the opportunity to reduce the scale of and/or solve urbanization issues. During the past 10 years, city systems have become more digital and information-based, and there has been a fundamental change in the living environment of citizens and the governing mode of cities. The economy, culture, transport, entertainment and all other aspects of cities have become closely combined with ICT, and the Internet has become a major part of citizens' daily lives. The abundant accomplishments of digitizing a city's information not only introduce daily convenience to the population, but also establish an infrastructure and conglomeration of data as a basis for further evolution of modern cities. Over the last 10 years, innovative information technologies such as cloud computing, 'big data', data vitalization, the 'Internet of Things' and mobile computing have become widely adopted in a variety of different areas. Cloud computing enables, developers to provide internet services without the need for a large capital outlay on hardware for deployment or the staff to operate it. The amount of information published and processed both on- and offline has given rise to an information explosion, and a new field dedicated to dealing with it— big data—which has spawned the need for new, more scalable, techniques to derive answers from huge sets of data. The emergence of the Internet of Things makes it possible to access remote sensor data and to control the physical world from a distance, meaning that cities can effectively sense and manage essential elements such as the water supply, building operations, and road and transport networks. Data vitalization proposes a new paradigm for large-scale dataset analysis and offers ubiquitous data support for top-level applications for smart cities. With the help of mobile computing, users can access and process information anywhere, and anytime, on all aspects of life. The urbanization, growth and associated problems of modern cities, coupled with the rapid development of new ICT, has enabled us to first envisage the 'smart cities' concept, and now to begin to build smart cities, which is seen as the future form for cities. Figure 1 shows how a smart city is formed. Smart city includes smart business, smart living, smart education, smart community, smart government, smart infrastructure, smart utility, smart mobility and smart environment.



The new Internet of Things (IoT) applications are enabling Smart City initiatives worldwide. It provides the ability to remotely monitor, manage and control devices, and to create new insights and actionable information from massive streams of real-time data. The main features of a smart city include a high degree of information technology integration and a comprehensive application of information resources. The essential components of urban development for a smart city should include smart technology, smart industry, smart services, smart management and smart life. The Internet of Things is about installing sensors (RFID, IR, GPS, laser scanners, etc.) for everything, and connecting them to the internet through specific protocols for information exchange and communications, in order to achieve intelligent recognition, location, tracking, monitoring and management. With the technical support from IoT, smart city need to have three features of being instrumented, interconnected and intelligent. Only then a Smart City can be formed by integrating all these intelligent features at its advanced stage of IOT development. The explosive growth of Smart City and Internet of Things applications creates many scientific and engineering challenges that call for ingenious research efforts from both academia and industry, especially for the development of efficient, scalable, and reliable Smart City based on IoT. New protocols, architectures, and services are in dire needs to respond for these challenges. The goal of the special issue is to bring together scholars, professors, researchers, engineers and administrators

resorting to the state-of-the-art technologies and ideas to significantly improve the field of Smart City based on IoT.

So, when we talked about smart cities; what is it. So, in addition to the regular infrastructure that is there in any city for example, the urban infrastructure consisting of office buildings residential areas hospitals schools transportation police and so on you also need something in addition to make the cities smart. So, what is this in addition let us talk about. So, smart means what smart means that it is in terms of the services that are given to the respective stake holders of these cities. So, citizens are able to do things in a better manner in an improved manner then usual and how is that made possible that is made possible with the help of nothing, but the ICT technologies information and communication technologies which also includes electronics embedded electronics different other advanced topologies in electrical in a electrical sciences and so on. So, computers electronics put together can make these cities smart. So, definitely will have to take help of sensors ,sensor networks sensor networks then actuators then the different other communication technologies RFID, NFC, ZWAVE and so and so forth.

Modern urban spaces are hotbeds of new ideas and world-shaking innovations. As for urban adoption of connected tech: all things considered, it really makes practical sense. Densely populated areas stand to gain the most from improved surroundings, and depending on the city, they might already come equipped with the fundamental IT infrastructures, which makes the further adaption easier. Meanwhile, the IoT might also offer some solutions to ease the huge burden that the urban explosion has meant for the existing infrastructures.

Connected City

A common definition for a smart city is using ICT to make a city

(administration, education, transportation, etc.) more intelligent and efficient. The definitions and concepts of smart cities are still emerging, and there is currently no clear and consistent definition of a smart city among the different stakeholders. In order to implement and assess smart cities in practice, a deeper understanding of the 'smart city' still needs to be defined. Many countries and cities have launched their own smart city projects to resolve urbanization issues and challenges. The USA was one of the first countries to launch a smart city project with a high compliment of smarter planet notions from President Barack Obama. In particular, for developing countries, the speed of urbanization is considerably faster and, as a consequence, the infrastructure problems faced are much greater. In 2014, India declared an intention to build more than 100 smart cities, with high- technology communication capabilities, throughout the country. ICT plays an important role in smart city construction. Top-level architecture research plays a considerable role in guiding technology development in every domain of a smart city and improving research into resource configuration. Now let's extrapolate the potential use cases to an entire city in which we have many objects that are capable of capturing information and interacting with other objects. The street lamp can now not only communicate with the devices that are closest but with other objects that are connected to the Internet and process this information to make decisions, for example, about the intensity of the light that is the most appropriate. The objects can also send information about what is happening in their environment or process different information. If the information from the street lamp is processed alongside with information from a nearby traffic light, we can start talking about the IoT use cases in the smart city environment.

When it comes to smart cities and the management of public space, the scope of possibilities, that IoT offers, is infinite. In other words, the IoT comes with considerable possibilities and room for manoeuvre within the field of smart cities. It is one of the aspects that we will touch in the Master's in Global Smart City Manager. IoT is a technology that is already there, that has been developed for a long time, but whose implantation in the public space will prevail in the years to come.

And depending on the way we approach our smart city project or the implementation of this technology in public space, smart city projects will be developed in one way, or another, they will be able to achieve common objectives in one way or another.

Possible IoT Use Cases for Smart Cities

- **Smart parking**

An IoT solution will permit monitoring the availability of parking spots in the city. With the GPS data from drivers' smartphones (or road-surface sensors in the ground), smart parking solutions let the user know when the closest parking spot becomes free to find a parking spot faster and easier instead of blindly driving around.

- **Smart roads and smart traffic congestion management**

Different IoT solutions will permit to monitor vehicle and pedestrian levels to optimize driving and walking routes. The use of different types of sensors, as well as GPS data from drivers' smartphones will help to determine the number, location and the speed of vehicles. Thanks to a cloud management platform which connects various traffic lights, the city will be able to monitor green light timings and automatically alter the lights based on the current traffic situation to prevent congestion. Better control of traffic congestion will also help to improve air quality.

- **Smart public transport**

With the help of IoT sensors, we can obtain data to learn about the patterns of how citizens use public transport. Smart public transport solutions can combine multiple sources, such as ticket sales and traffic information. The users could also use an app to contact the authorities in case they spot incidents or suspicious activities.

- **Smart street lighting**

IoT-based smart cities allow better maintenance and control of street lamps. Equipping streetlights with sensors and connecting them to a cloud management solution makes them more straightforward and cost-effective. With this system, the city can adapt the lighting schedule to the lighting zone and weather conditions.

- **Smart waste management**

Waste-collecting is another service that could be optimized with an IoT- enabled solution by tracking waste levels, as well as providing route optimization and operational analytics.

Advantages of a Smart City

Smart cities can be described as cities capable of using information technology to create efficiencies and create sustainability, and improve the quality of life of it's residents. A smart city is basically a living entity, capable of extraordinary adaptations that we once thought were not possible. This post will be discussing smart cities, including what makes a smart city, it's benefits, it's effects on the environment, and what negative effects, if any it might have on it's citizens and the world as a whole.

The benefits of smart cities

- **Efficient distribution of resources**

Smart cities have an overall better organization and infrastructure. All the sectors are involved in a complex interplay that simplifies everyday life for people who live and work in the city. The cameras at the bus stops can identify how many people are waiting to board; the sensors on the approaching bus know how many people ride the bus at any given point in time, and how many people are currently

on the bus. The combination of the information from the bus stop and the bus then leads to the city's response. There can then be redistribution of people and buses if it appears that the current course of events will not be efficient.

- **Seamless communication**

Communications between the various systems and sensors in a smart city is very important. In fact, without them the smart city cannot efficiently redistribute resources and make citizens' lives better. However, smart cities bring about a different, equally efficient communication–the communication between the citizens and the government of the particular city.

In prior times, policies and programs were made based on what the government perceived to be needed by the city. This often led to massive oversights and the omission of key policies altogether. In a smart city, the policy makers have all they could ever need to make informed decisions. The information gathered all across the city provide an invaluable line of communication between the needs of the city, and the people who can address those needs.

- **Speed of implementation**

Still on governments and policies, every country with a democracy can testify to the fact that it takes quite a while for policies, or any sort of new development to get implemented. This is partly due to bureaucracy and the multiple levels of government, and also partly due to the human factor. Smart cities overcome these problems very easily. Because the points that need improvements have already been identified, the implementation becomes easier. All the automation, analytics, and sensors contribute to making it easier for most of the changes to be implemented remotely, creating a seamless flow of change from conception to execution.

Smart Home:

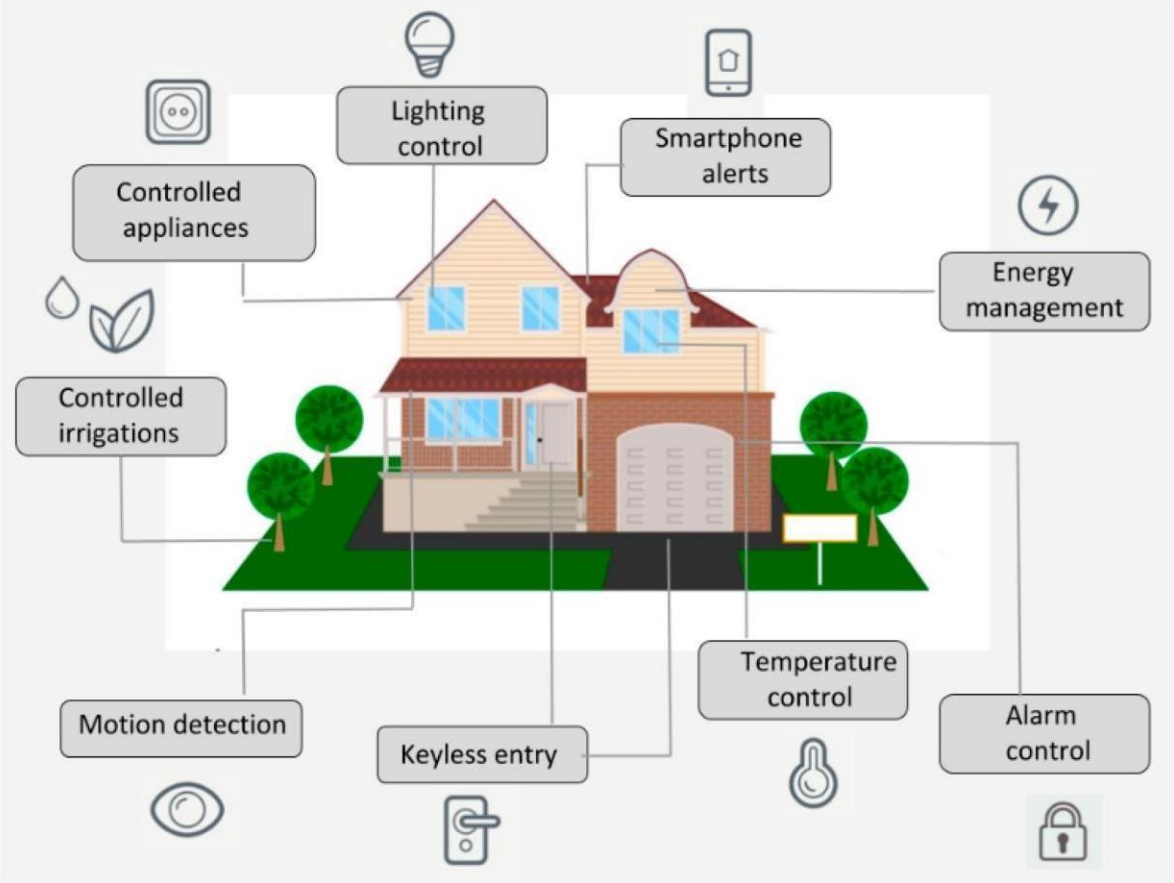
The Internet of Things (IoT) is a system that allows devices to be connected and remotely monitored across the Internet. In the last years, the IoT concept has had a strong evolution, being currently used in various domains such as smart homes, telemedicine, industrial environments, etc. Wireless sensor network technologies integrated into the IoT enable a global interconnection of smart devices with advanced functionalities. A wireless home automation network, composed of sensors and actuators that share resources and are interconnected to each other, is the key technology to making intelligent homes.

A "smart home" is a part of the IoT paradigm and aims to integrate home automation. Allowing objects and devices in a home to be connected to the Internet enables users to remotely monitor and control them. These include light switches that can be turned on and off by using a smartphone or by voice command, thermostats that will adjust the indoor temperatures and generate reports about energy usage, or smart irrigation systems that will start at a specific time of a day, on a custom monthly schedule, and thus will control water waste. Smart

home solutions have become very popular in the last years. Figure 1 shows an example of a smart home that uses different IoT-connected utilities.

Cloud computing and its contribution to IoT and smart home: Cloud computing is a shared pool of computing resources ready to provide a variety of computing services in different levels, from basic infrastructure to most sophisticated application services, easily allocated and released with minimal efforts or service provider interaction. In practice, it manages computing, storage, and communication resources that are shared by multiple users in a virtualized and isolated environment. IoT and smart home can benefit from the wide resources and functionalities of cloud to compensate its limitation in storage, processing, communication, support in pick demand, backup and recovery. For example, cloud can support IoT service management and fulfillment and execute complementary applications using the data produced by it. Smart home can be condensed and focus just on the basic and critical functions and so minimize the local home resources and rely on the cloud capabilities and resources. Smart home and IoT will focus on data collection, basic processing, and transmission to the cloud for further processing. To cope with security challenges, cloud may be private for highly secured data and public for the rest.

NRCM
your tools to success...



One of the greatest advantages of home automation systems is their easy management and control using different devices, including smartphones, laptops and desktops, tablets, smart watches, or voice assistants. Home automation systems offer a series of benefits; they add safety through appliance and lighting control, secure the home through automated door locks, increase awareness through security cameras, increase convenience through temperature adjustment, save precious time, give control, and save money.

Several home automation systems involved with IoT have been proposed by academic researchers in the literature in the last decade. In wireless-based home automation systems, different technologies have been used, each of them with their pros and cons. For example, Bluetooth-based automation is low cost, fast, and easy to be installed, but it is limited to short distances. GSM and ZigBee are widely used wireless technologies as well. GSM provides long-range communication at the cost of a mobile plan of the service provider that operates in the area. Zigbee is a wireless mesh network standard that is designed to be low-cost and with low power consumption, targeted at battery-powered devices in wireless control and monitoring applications. However, it has a low data speed, low transmission, as well as low network stability, and has a high maintenance cost. The advantages of Wi-Fi technology over ZigBee or Z-Wave are related to price, complexity (meaning simplicity), and accessibility. First, Wi-Fi-enabled smart devices are usually cheap. In addition, it is easier to find do-it-yourself devices that use Wi-Fi, resulting a less expensive option. Second, Wi-Fi is already a necessity and it is in most homes, so it is easier to buy

devices that are already Wi-Fi-enabled. Finally, Wi-Fi is characterized by simplicity, meaning that a user must connect only a minimal number of devices for a home automation setup. Since it is very common, the investment on extra hardware is avoided; a user only needs the basic setup for a home automation system. However, Wi-Fi is not designed to create mesh networks, it consumes ten times more energy than similar devices using ZigBee, Z-Wave, or Bluetooth for example, and many Wi-Fi routers can only allow up to thirty devices connected at once. As compared to Ethernet, Wi-Fi brings several advantages, including the easy connection and access of multiple devices, the expandability (adding new devices without the hassle of additional wiring), lower cost, or single access point requirement.

The basic architecture enables measuring home conditions, process instrumented data, utilizing microcontroller-enabled sensors for measuring home conditions and actuators for monitoring home embedded devices.The popularity and penetration of the smart home concept is growing in a good pace, as it became part of the modernization and reduction of cost trends. This is achieved by embedding the capability to maintain a centralized event log, execute machine learning processes to provide main cost elements, saving recommendations and other useful reports.

Smart home services

- **Measuring home conditions**

A typical smart home is equipped with a set of sensors for measuring home conditions, such as: temperature, humidity, light and proximity. Each sensor is dedicated to capture one or more measurement. Temperature and humidity may be measured by one sensor, other sensors calculate the light ratio for a given area and the distance from it to each object exposed to it. All sensors allow storing the data and visualizing it so that the user can view it anywhere and anytime. To do so, it includes a signal processer, a communication interface and a host on a cloud infrastructure.

- **Managing home appliances**

Creates the cloud service for managing home appliances which will be hosted on a cloud infrastructure. The managing service allows the user, controlling the outputs of smart actuators associated with home appliances, such as such as lamps and fans. Smart actuators are devices, such as valves and switches, which perform actions such as turning things on or off or adjusting an operational system. Actuators provides a variety of functionalities, such as on/off valve service, positioning to percentage open, modulating to control changes on flow conditions, emergency shutdown (ESD). To activate an actuator, a digital write command is issued to the actuator.

- **Controlling home access**

Home access technologies are commonly used for public access doors. A common system uses a database with the identification attributes of authorized people. When a person is approaching the access control system, the person's identification

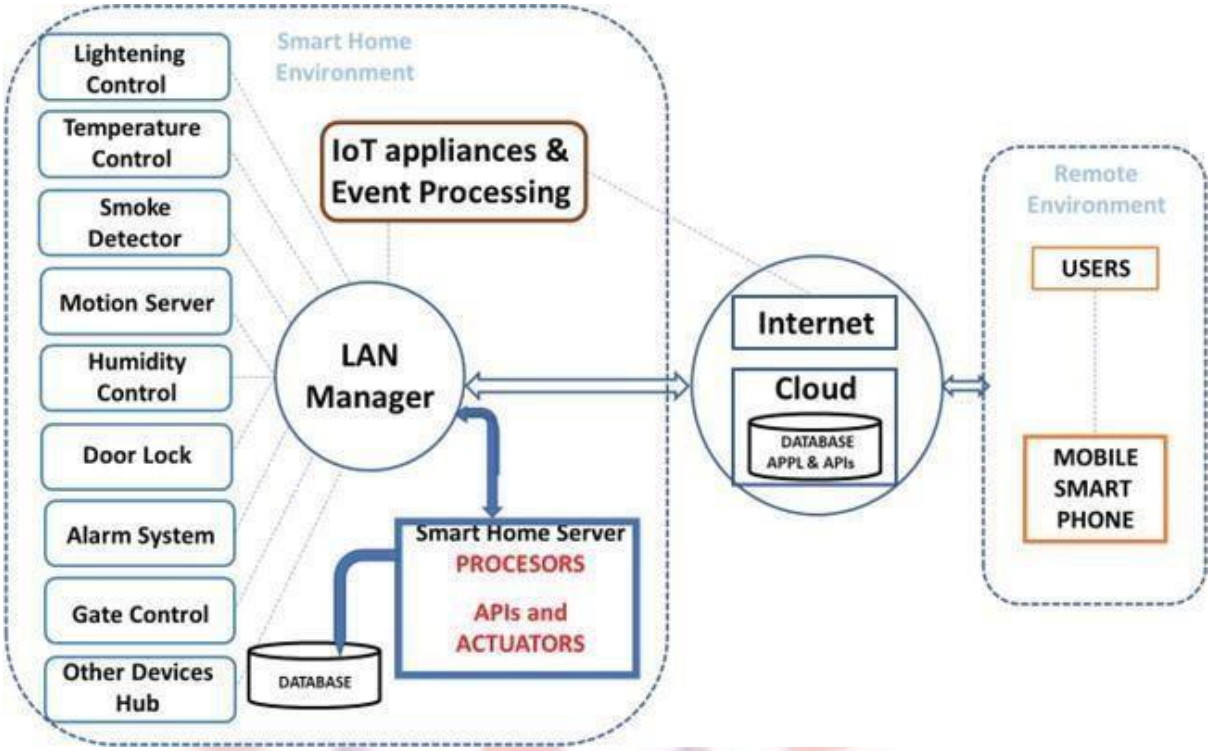
attributes are collected instantly and compared to the database. If it matches the database data, the access is allowed, otherwise, the access is denied. For a wide distributed institute, we may employ cloud services for centrally collecting persons' data and processing it. Some use magnetic or proximity identification cards, other use face recognition systems, finger print and RFID.

In an example implementation, an RFID card and an RFID reader have been used. Every authorized person has an RFID card. The person scanned the card via RFID reader located near the door. The scanned ID has been sent via the internet to the cloud system. The system posted the ID to the controlling service which compares the scanned ID against the authorized IDs in the database.

The main components

To enable all of the above described activities and data management, the system is composed of the following components, as described in Figure 1.

- Sensors to collect internal and external home data and measure home conditions. These sensors are connected to the home itself and to the attached-to-home devices. These sensors are not internet of things sensors, which are attached to home appliances. The sensors' data is collected and continually transferred via the local network, to the smart home server.
- Processors for performing local and integrated actions. It may also be connected to the cloud for applications requiring extended resources. The sensors' data is then processed by the local server processes.
- A collection of software components wrapped as APIs, allowing external applications execute it, given it follows the pre-defined parameters format. Such an API can process sensors data or manage necessary actions.
- Actuators to provision and execute commands in the server or other control devices. It translates the required activity to the command syntax; the device can execute. During processing the received sensors' data, the task checks if any rule became true. In such case the system may launch a command to the proper device processor.
- Database to store the processed data collected from the sensors [and cloud services]. It will also be used for data analysis, data presentation and visualization. The processed data is saved in the attached database for future use.



- Cloud computing and its contribution to IoT and smart home: Cloud computing is a shared pool of computing resources ready to provide a variety of computing services in different levels, from basic infrastructure to most sophisticated application services, easily allocated and released with minimal efforts or service provider interaction. In practice, it manages computing, storage, and communication resources that are shared by multiple users in a virtualized and isolated environment. IoT and smart home can benefit from the wide resources and functionalities of cloud to compensate its limitation in storage, processing, communication, support in pick demand, backup and recovery. For example, cloud can support IoT service management and fulfillment and execute complementary applications using the data produced by it. Smart home can be condensed and focus just on the basic and critical functions and so minimize the local home resources and rely on the cloud capabilities and resources. Smart home and IoT will focus on data collection, basic processing, and transmission to the cloud for further processing. To cope with security challenges, cloud may be private for highly secured data and public for the rest.

IoT challenges for Smart City and Smart Home:

Infrastructure: Smart Cities utilize sensor technology to gather and analyze information in an effort to improve the quality of life for residents. Sensors collect data on everything from rush hour stats to crime rates to overall air quality. Complicated and costly infrastructure is involved in installing and maintaining these sensors. How will they be powered? Will it involve hard- wiring, solar energy, or battery operation? Or, in case of

power failure, perhaps a combination of all three? Funding for new infrastructure projects is limited and approval processes can take years. Installing new sensors and other improvements cause temporary – though still frustrating – problems for people living in these cities.

- **Security and Hackers**

As IoT and sensor technology use expands, so does the threat level to security. This begs the question…is technology really considered "smart" if hackers can break into it and shut down an entire city? Recent discussion involving cyber-terror threats to vulnerable and outdated power grids has everyone a bit more concerned and skeptical about technology and security. Smart Cities are investing more money and resources into security, while tech companies are creating solutions with new built-in mechanisms to protect against hacking and cyber-crimes.

- **Privacy Concerns**

In any major city, there's a balance between quality of life and invasion of privacy. While everyone wants to enjoy a more convenient, peaceful, and healthy environment, nobody wants to feel like they are constantly being monitored by "Big Brother."

Cameras installed on every street corner may help deter crime, but they can also install fear and paranoia in law-abiding citizens. Another valid concern is the amount of data being collected from all the smart sensors residents come into contact with each day.

- **Educating & Engaging the Community**

For a Smart City to truly exist and thrive, it needs "smart" citizens who are engaged and actively taking advantage of new technologies. With any new city-wide tech project, part of the implementation process must involve educating the community on its benefits. This can be done through a series of in-person town hall-style meetings and email campaigns with voter registration, as well as an online education platform that keeps citizens engaged and up-to-date. When a community feels like it's playing a part in the overall decisions that affect daily life, and is being communicated to in a clear and thoughtful manner, it's more apt to use the technology and encourage others to use it as well. This is key to a Smart City's success.

Connected vehicles:

Connected vehicle technology can change our transportation system as we know it by enabling safe, interoperable networked wireless communications among vehicles, the infrastructure, and passengers' personal communications devices. Connected vehicle technology will enable cars, trucks, buses, and other vehicles to "talk" to each other with in-vehicle or aftermarket devices that continuously share important safety and mobility information. Connected vehicles can also use wireless communication to "talk" to traffic signals, work zones, toll booths, school zones, and other types of infrastructure.

Different communications technologies (satellite, cellular, dedicated short range communications) may be utilized depending on the performance requirements of the

connected vehicle applications. Cars, trucks, buses, and other vehicles can "talk" to each other with in-vehicle or aftermarket devices that continuously share important safety and mobility information. Connected vehicles can also use wireless communication to "talk" to traffic signals, work zones, toll booths, school zones, and other types of infrastructure. The vehicle information communicated does not identify the driver or vehicle, and technical controls have been put in place to help prevent vehicle tracking and tampering with the system. The vision for connected vehicle technologies is to transform surface transportation systems to create a future where:

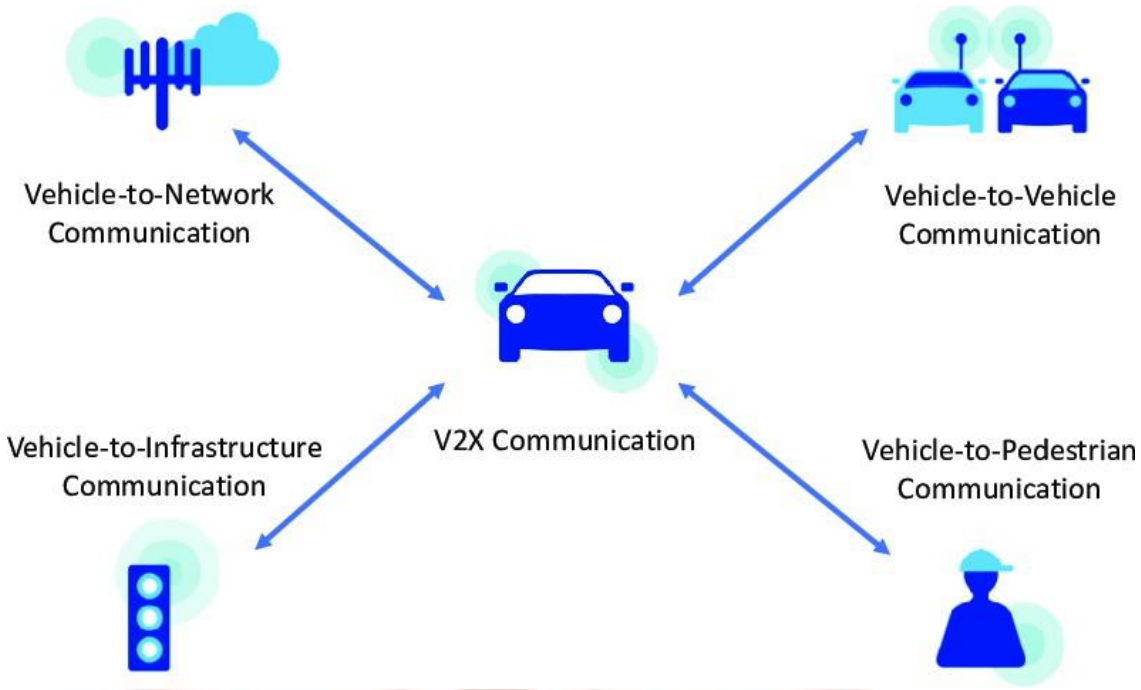
- Highway crashes and their tragic consequences are significantly reduced
- Traffic managers have data to accurately assess transportation system performance and actively manage the system in real time, for optimal performance
- Travelers have continual access to accurate travel time information about mode choice and route options, and the potential environmental impacts of their choices
- Vehicles can talk to traffic signals to eliminate unnecessary stops and help drivers operate vehicles for optimal fuel efficiency.

Challenges:

- Security
- Privacy
- Scalability
- Reliability
- Quality of service
- Lack of Global Standards

What is Vehicle to Everything (V2X)?

Vehicle to Everything (V2X) is a vehicular communication system that supports the transfer of information from a vehicle to moving parts of the traffic system that may affect the vehicle. The main purpose of V2X technology is to improve road safety, energy savings, and traffic efficiency on the roads.



How Vehicle to Everything (V2X) Works

In a V2X communication system, the information travels from the vehicle sensors and other sources through high-bandwidth, high-reliability links, allowing it to communicate with other cars, infrastructure such as parking spaces and traffic lights, and smartphone-tossing pedestrians. By sharing information, such as speed, with other entities around the vehicle, the technology improves the driver's awareness of potential dangers and helps reduce the severity of injuries, road accident fatalities, and collision with other vehicles. The technology also enhances traffic efficiency by warning drivers of upcoming traffic, suggesting alternative routes to avoid traffic and identifying available parking spaces.

Components of V2X Technology

The key components of V2X technology include V2V (vehicle-to-vehicle) and V2I (vehicle-to-infrastructure). V2V allows vehicles to communicate with other vehicles on the road, while V2I allows vehicles to communicate with external entities, such as traffic lights, parking spaces, cyclists, and pedestrians. The technologies help improve road safety, reduce fuel consumption, and enhance the experience between drivers and other road users, such as cyclists and pedestrians. When V2X systems are integrated into traditional vehicles, drivers can receive important information about the weather patterns, nearby accidents, road conditions, road works warning, emergency vehicle approaching, and activities of other drivers using the same road. Autonomous vehicles equipped with V2X systems may provide more information to the existing navigation system of the vehicle. The systems also make it possible for autonomous vehicles to scan the surrounding environment and make immediate decisions based on the information received.

Smart Grid:

What is the Smart Grid?

The "grid" is the electrical network serving every resident, business and infrastructure service in a city. The "smart grid" is the next generation of those energy systems, which have been updated with communications technology and connectivity to drive smarter resource use.

The technologies that make today's IoT-enabled energy grid "smart" include wireless devices such as sensors, radio modules, gateways and routers. These devices provide the sophisticated connectivity and communications that empower consumers to make better energy usage decisions, allow cities to save electricity and expense, and enables power authorities to more quickly restore power after a blackout.

The Smart Grid is critical to building a secure, clean, and more efficient future, according to the International Energy Agency (IEA). The Smart Grid is part of an IoT framework, which can be used to remotely monitor and manage everything from lighting, traffic signs, traffic congestion, parking spaces, road warnings, and early detection of things like power influxes as the result of earthquakes and extreme weather. The Smart Grid does this through a network of transmission lines, smart meters, distribution automation, substations, transformers, sensors, software and more that are distributed to businesses and homes across the city.

Smart Grid technologies all contribute to efficient IoT energy management solutions that are currently lacking in the existing framework. What makes the IoT Smart Grid better is two-way communication between connected devices and hardware that can sense and respond to user demands. These technologies mean that a Smart Grid is more resilient and less costly than the current power infrastructure.

The main advantages identified in this document are:

- Energy savings through reducing consumption

One of the advantages of smart grids is that they can tell us the consumption at an energy meter at any time, so users are better informed of their real consumption. Moreover, with better consumption monitoring, contracted power can be adjusted to meet the real need of each consumer. These two factors result in users reducing their consumption and tailoring their contracted power to their real needs.

- Better customer service and more accurate bills

Another key advantage offered by tele-management systems is that bills are more accurate. They always reflect the real consumption of each month instead of estimates, reducing the cost of the old system of manual energy meter readings. In addition to being able to access information about the installation remotely, problems become easier to diagnose and solutions can therefore be implemented faster, improving customer service. Now a days customers have to notify

companies for them to take action. But with remote management the system itself automatically reports all incidents to the electric company so it can respond faster to users.

- Reduced balancing cost

Smart Grids can collect much more data than the manual energy meter reading system. This permits the use of data analysis techniques and the preparation of highly realistic consumption forecasts as many more variables are taken into account. Utilities can then better tailor their production to consumption (balances) and reduce energy surpluses.

- Reduction of carbon emissions

All the benefits above involve reducing consumption, which entails a reduction in CO2 emissions. We can thus say that Smart Grids lead to a more sustainable future. All this will directly contribute to the future integration of electric vehicle charging systems on the mains. The deployment of renewable energy systems is also made easier as utilities gain greater control of their grids.

- Smart Grid Enables Renewable Energy Generation

Traditional energy grids are designed to transmit electricity from a large, centralized power station to a wide network of homes and businesses in the area. At this stage, the electric grid is not designed to accept inputs from homes and businesses that are generating power via solar panels or windmills. A smart grid is designed to accept power from renewable resources. Crucially, the smart grid in conjunction with wirelessly enabled smart meters can keep track of how much energy a net-positive establishment is generating and reimburse them accordingly. The smart grid also allows for monitoring of solar panels and equipment as well. We mentioned earlier that a smart grid can mitigate the effects of a disaster such as a terrorist attack or natural disaster on a power station, a feat that's possible due to decentralized energy generation. Under the traditional model, a small number of power plants powered a city. This left these services vulnerable to threats that would result in widespread blackouts and energy shortages. With a decentralized model, even if the centralized power plant is taken offline, multiple alternative sources, including wind and solar, can supplant the resources in the grid. This decentralized system is much harder to take offline and can provide a robustness that's not possible when one plant is powering an entire city.

Smart Grid for the Future

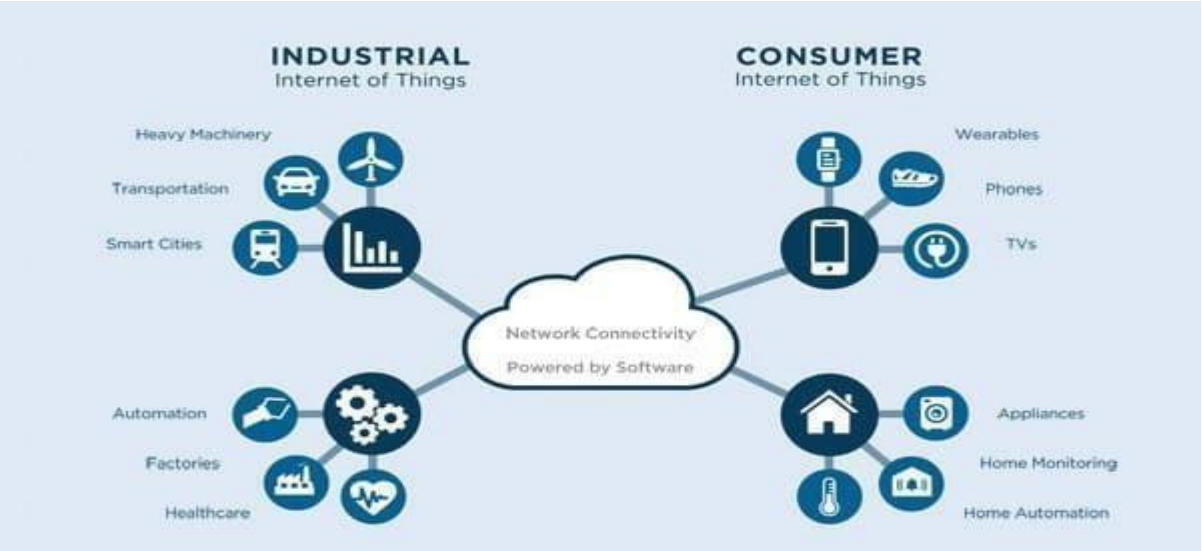
Smart grid technology can be expressed in a single sentence: a new electric grid with two-way communication. For the first time, businesses and consumers can get real time billing information while utility companies can better meet the needs of their customers as they react to demand spikes and fix or manage blackouts and other challenges. Smart grid is resilient, efficient and green which is good for the consumer, the utility company and the environment. Wireless technology will replace thousands of miles of cable that would have been needed to advance the smart grid to where it is today.

Challenges:

- High Investment
- Cyber attacks

Industrial IoT:

The Industrial Internet of Things (IIoT), which is considered as the main future IoT-application area, is defined by the Industrial Internet Consortium as machines, computers and people enabling intelligent industrial operations using advanced data analytics for transformational business outcomes" ("Industrial Internet Consortium," 2017). Generally, IIoT is one basis of Industry 4.0 and the digital transformation. The IIoT is the connection between IT (information technology) and OT (operational technology). IIoT is the most important segment in IoT, much more than consumer applications. The Industrial Internet of Things is related to the Industry 4.0: all IoT applications in Industry 4.0 are forms of IIoT but not all IIoT use cases are about the industries which are categorized as Industry 4.0. Typical use cases of the Industrial Internet of Things include intelligent machine applications, industrial control applications, factory floor use cases, condition monitoring, use cases in agriculture or smart grid applications. It is important to know that the IIoT is not just about saving costs and optimizing efficiency though. Companies also have the possibility to realize important transformations and can find new opportunities, e.g., entirely new business models in Industry



According to TechTarget, IIoT can be formally defined as "the use of smart sensors and actuators to enhance manufacturing and industrial processes. Also known as the industrial internet or Industry 4.0, IIoT leverages the power of smart machines and real-time analytics to take advantage of the data that dumb machines have produced in industrial settings for years."

Industrial IoT capabilities require widespread digitization of manufacturing operations. Organizations must include four primary pillars to be considered a fully IIoT-enabled operation:

- Smart machines equipped with sensors and software that can track and log data.
- Robust cloud computer systems that can store and process the data.
- Advanced data analytics systems that make sense of and leverage data collected from systems, informing manufacturing improvements and operations.
- Valued employees, who put these insights to work and ensure proper manufacturing function.

Benefits of IIoT: These are 5 of the biggest benefits of adopting a fully connected IIoT manufacturing operation.

Increase efficiency

The biggest benefit of IIoT is that it gives manufacturers the ability to automate, and therefore optimize their operating efficiency. Robotics and automated machinery can work more efficiently and accurately, boosting productivity and helping manufacturers streamline their functions.

Additionally, physical machinery can be connected to software via sensors that monitor performance on a constant basis. This enables manufacturers to have better insights into the operational performance of individual pieces of equipment as well as entire fleets.

IIoT-enabled data systems empower manufacturers to improve operating efficiencies by:

- Bypassing manual tasks and functions and implementing automated, digital ones
- Making data-driven decisions regarding all manufacturing functions
- Monitoring performance from anywhere – on the manufacturing floor or from thousands of miles away

Reduce Errors

Industrial IoT empowers manufacturers to digitize nearly every part of their business. By reducing manual process and entries, manufacturers are able to reduce the biggest risk associated with manual labor – human error.

This goes beyond just operational and manufacturing errors. IIoT solutions also can reduce the risk of cyber and data breaches caused by human error. A Cyber Security Trend report cited people as the biggest cause of cyber security breaches, with human error being the culprit 37% of the time. AI and machine learning-enabled programs and machinery can do much of the required computing themselves, eliminating the potential for someone to make a simple mistake, and put the manufacturer's data at risk.

Predictive Maintenance

Nothing negatively impacts a manufacturing operation more than machine downtime. When maintenance in the manufacturing world is reactive rather than proactive, manufacturers are stuck trying to identify what the issue is, how it can be repaired, and what it will cost. With predictive maintenance powered by industrial IoT solutions, all of those issues are alleviated.

When machinery performance and function is monitored consistently, manufacturers can create a baseline. This baseline and the corresponding data empowers companies with the information they need to see any issue before it occurs. They can then schedule maintenance prior to downtime, which benefits them in that they:

- Have the parts required for the job

- Know the cost of the project beforehand, and can budget for it
- Move production to another area of the facility, so the product quotas are unaffected
- Ensure that machinery is operating at maximum efficiency

Improve Safety

All of the data and sensors required of a fully functioning IIoT manufacturing operation are also helping to bolster workplace safety. "Smart manufacturing" is turning into "smart security" when all of the IIoT sensors work together to monitor workplace and employee safety.

Integrated safety systems are protecting workers on the floor, on the line, and in distribution. If an accident occurs, everyone in the facility can be alerted, operations can cease, and company leadership can intervene and make sure the accident and incident is resolved. This incident can also generate valuable data that can help prevent a repeat occurrence in the future.

A newer option some manufacturers are utilizing is the use of wearable technology among their employees. Wearables have been part of IoT since its infancy, and it are just now being utilized in industrial IoT operations.

Wearables help leadership keep tabs on things like employee posture and the surrounding noise levels, and they can then improve work conditions and potentially improve performance. They can also alert employees when they aren't following proper workplace safety procedures, so they can correct their actions and stay safe on the job.

Reduce Costs

Knowledge is power, and the knowledge provided to manufacturers via IIoT solutions is giving them the tools they need to reduce costs and generate more revenue. Data-driven insights into operations, production, marketing, sales, and more can steer businesses in a profitable direction.

All of the aforementioned benefits of IIoT – predictive maintenance, fewer errors, improved quality control, and maximized efficiencies – will all boost profits for a manufacturer. Industrial IoT also offers arguably the most valuable tool for leaders of a manufacturing company – insights from anywhere, anytime.

Remote monitoring of manufacturing operations is now possible 365 days a year, 24/7, from anywhere in the world. This 360-degree view into the entire manufacturing process, and the follow-up service provided to customers in their buying journey, is an invaluable asset.