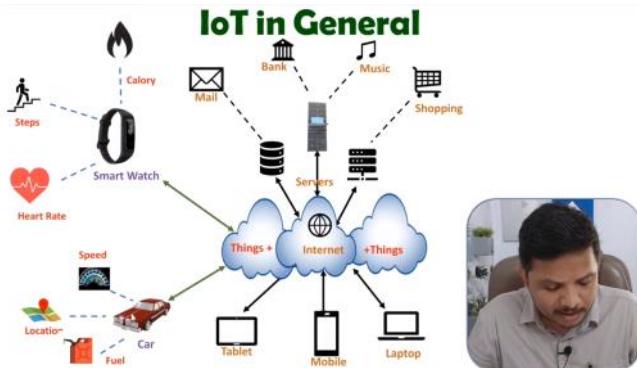


IOT NOTES

12 February 2024 12:45

02:48



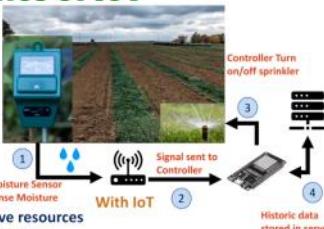
06:18

Importance of IoT



Without IoT

- Uses more resources
- Consumes more time
- Decrease farm yield



With IoT

IOT Definition

▪ The Internet of Things (IoT) is the network of physical objects or "things" embedded with electronics, software, sensors, and connected them with internet, which enables these objects to collect and exchange data.

▪ IoT allows objects to be sensed and controlled remotely across existing network infrastructure, creating opportunities for more direct integration between the physical world and computer-based systems, and resulting in improved efficiency, accuracy and economic benefit.



08:46

▪ "Things," in the IoT sense, can refer to a wide variety of devices such as heart monitoring implants, biochip transponders on farm animals, electric clams in coastal waters, automobiles with built-in sensors, DNA analysis devices for environmental/food/pathogen monitoring or field operation devices that assist fire-fighters in search and rescue operations.



09:24

- "Things," in the IoT sense, can refer to a wide variety of devices such as heart monitoring implants, biochip transponders on farm animals, electric clams in coastal waters, automobiles with built-in sensors, DNA analysis devices for environmental/food/pathogen monitoring or field operation devices that assist fire-fighters in search and rescue operations.

- These devices collect useful data with the help of various existing technologies and then autonomously flows the data between other devices.



05:14

IoT is a simple concept of basically controlling different devices by establishing a connection & communication from mobile app or web browser.

~~✓~~ Taking everyday things, embedding them with electronics, software, sensors & then connecting them to Internet & enabling them to collect & exchange data without human intervention is called as the IoT.

Goal of IoT -

To extend Internet connectivity from

06:03

human intervention is called as the IoT.

~~✓~~ Goal of IoT -

To extend Internet connectivity from standard devices such as computer, mobile phone & electronic devices.

~~✓~~ IoT makes everything smart.

* The term Internet of Things was invented by Kevin Ashton in 1999 while he was working with Procter & Gamble.

08:38

Information for actions

- A new world of smart devices will make human life easier.

~~✓~~ Connectivity is a human need & desire

Roads connected places

Telephone connected people

Internet connected people & communities

~~✓~~ IoT makes entire world (every object) deeply & widely connected.

~~✓~~ IoT makes every object -

Addressable

Accessible

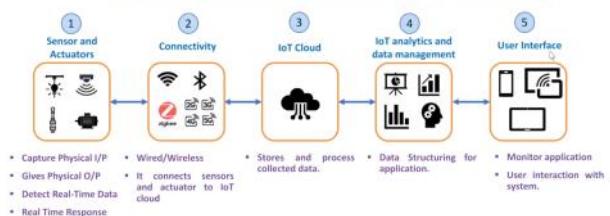
Actuatable

Smart devices → Every day industrial equipment

Consumer objects onto the net

03:47

How does IoT work? Components of IoT (IoT stack)



06:00

1.a - Sensors



- Definition – It measures physical input from the surroundings and converts it into electrical signal (Analog or Digital).
- Integration with microprocessor/microcontroller embedded system.
- E.G Smartphones - (GPS, Figure print, Tilt, camera, etc)
- The size of sensors is very compact due to advancements in technology.
- The choice of sensors is based on accuracy, precision, Level of intelligence, and the needs of the application.

07:17

1.b - Actuators



- While sensors, sense; actuators act
- Cause an action, to control the situation.
- To make something happen based on the trigger, actuators are used.
- Control signal could be analog or digital.
- To turn something on or off Actuators are used.



08:02

2. Connectivity & Gateway

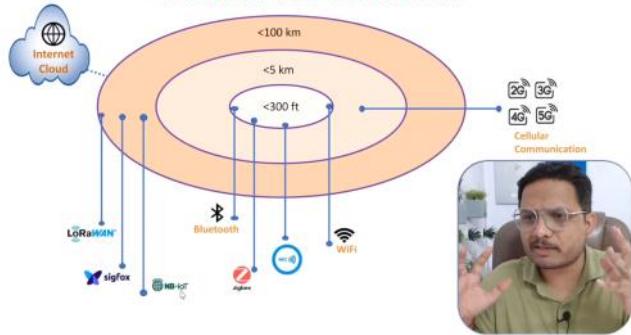
□ Roles of Connectivity Protocols

- It plays a crucial role in connecting sensors and actuators with IoT ecosystem.
- Based on medium and range, various protocols are available.
- E.g Cellular, Bluetooth, Wi-Fi, LoRaWAN, etc.



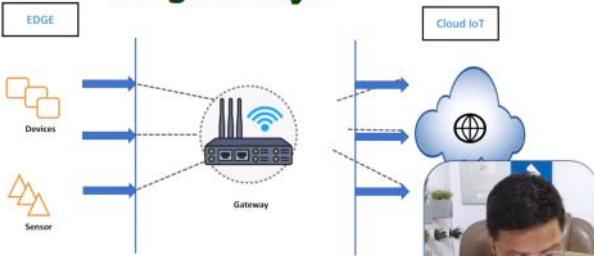
08:46

Various IoT Protocols



09:11

IoT gateways



- It provides an effective communication bridge in between edge devices and IoT Cloud.



10:36

□ Role of a Gateway

- Ensures seamless communication
- Easy management of data traffic.
- Offers security
- Latest encryption practices
- Data Preprocessing
- Minimizing the large data
- Intelligent IoT gateways



12:02

3. IoT Cloud

□ Role of an IoT cloud

- Data needs to be processed
- To make or break a deal. (Based on processed data)
- Latency cannot be compromised. (For The real-time applications)
- IoT Cloud is The brain of the IoT ecosystem
- Cloud is optional
- Edge computing (It is used for local processing)



13:09

4. IoT analytics and data management



14:17

- Data is the fuel for IoT Applications.
- Converting raw data into useful insights
- Data extraction, data aggregation, data classification are major tasks of IoT Analytics and data management.
- Deep learning (It can be for data prediction for some applications)
- Storage power and intelligent computation are key parameters of IoT Analytics and data management.



5. User Interface



Notification



Alerts



Remote Control



Live Trends

- Task is to monitor and control the field device for the user.
- Types of notification (Notification, Email, Alerts, Alarms etc.)
- It can also send back the command to the field devices
- It might provide the user with an actual live feed or show trends etc.



1. **Things:** Things are physical devices or objects embedded with sensors, actuators, and other technologies to connect and communicate data over the internet. Examples include smart thermostats, wearable fitness trackers, industrial machines, etc.
2. **Controllers:** Controllers are devices or systems that manage and coordinate the operation of multiple IoT devices. They often serve as the central hub for collecting data from sensors, making decisions, and sending commands to actuators. Controllers can range from simple microcontrollers to more sophisticated edge computing devices.
3. **Actuators:** Actuators are components that convert digital signals from the controller into physical action or movement. They enable IoT devices to interact with the physical world by controlling motors, valves, switches, etc. Examples include motors for opening/closing doors, valves for regulating fluid flow, and relays for switching electrical circuits.
4. **Sensors:** Sensors are devices that detect and measure physical or environmental parameters such as temperature, humidity, pressure, light, motion, etc. They capture real-world data and convert it into digital signals that can be processed by IoT systems. Sensors come in various types and configurations depending on the specific application and the type of data to be collected.
5. **Communicators:** Communicators are components responsible for establishing and maintaining communication between IoT devices and the wider network, typically the internet or local network infrastructure. They can include wireless communication modules (e.g., Wi-Fi, Bluetooth, Zigbee, LoRa, cellular), wired interfaces (e.g., Ethernet, ↓

specific application and the type of data to be collected.

5. **Communicators:** Communicators are components responsible for establishing and maintaining communication between IoT devices and the wider network, typically the internet or local network infrastructure. They can include wireless communication modules (e.g., Wi-Fi, Bluetooth, Zigbee, LoRa, cellular), wired interfaces (e.g., Ethernet, USB), and protocols for data exchange (e.g., MQTT, CoAP, HTTP). Communicators ensure seamless connectivity and data transmission between IoT devices and the cloud or other backend systems.

In summary, in an IoT ecosystem, things equipped with sensors and actuators are controlled and managed by controllers, while communicators facilitate the exchange of data between these devices and external networks or systems. Each component plays a critical role in enabling the functionality and capabilities of IoT solutions across various domains and applications.



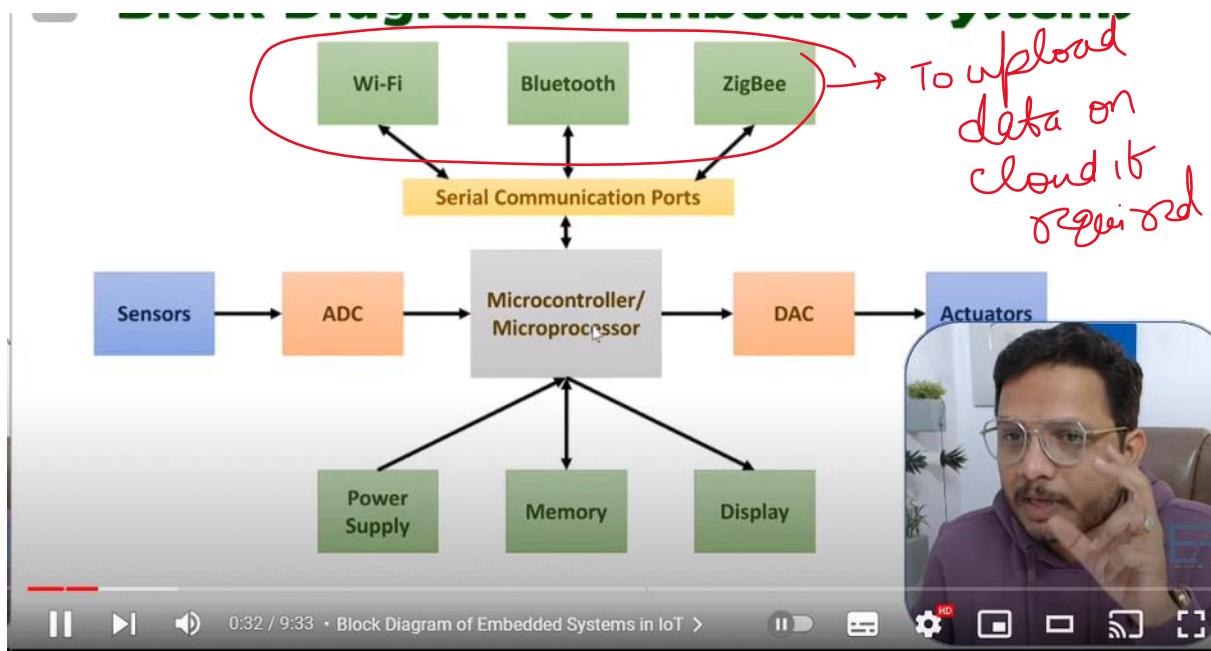
Block Diagram of Embedded Systems

Wi-Fi

Bluetooth

ZigBee

To upload
on



Basics of Embedded Systems

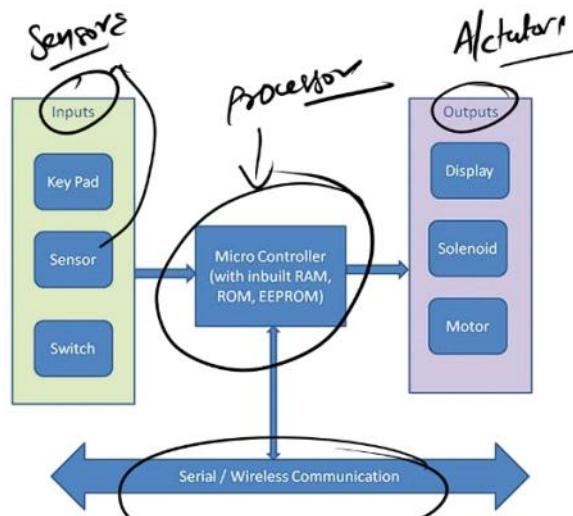
- An Embedded System is a computer system that has computer hardware and software embedded to perform specific tasks.
- In contrast to general-purpose computers or personal computers (PCs) which can perform various types of tasks, embedded systems are designed to perform a specific set of tasks.
- Key components of an embedded system include a microprocessor or microcontroller, memory (RAM, ROM, cache), networking units (Ethernet, Wi-Fi adapters), input/output units (display, keyboard, etc.), and storage (such as flash memory).
- Some embedded systems have specialized processors such as digital signal processors (DSPs), graphics processors, and application-specific processors.
- Embedded systems run embedded operating systems such as real-time operating systems (RTOS).
- Embedded systems range from low-cost miniaturized devices such as digital watches to devices such as digital cameras, point-of-sale terminals, vending machines, appliances (such as washing machines), etc.

8:18 / 9:33 • Basics of Embedded Systems in IoT > [play] [stop] [volume] [HD] [settings] [full screen]

Embedded System/Computer

- "Any sort of device which includes a programmable computer but itself is not intended to be a general-purpose computer" - Wayne Wolf

to
some
applicatn



Embedded systems

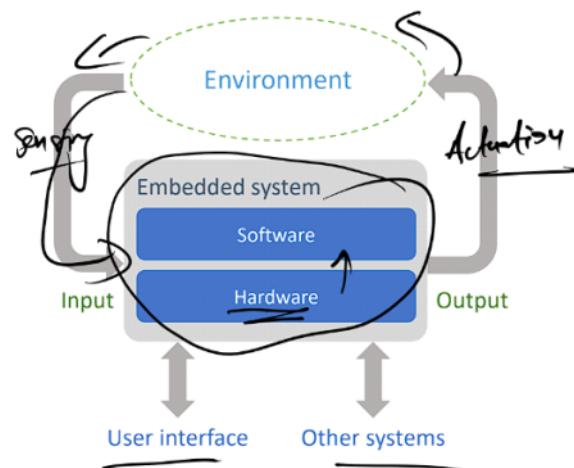
Internet of Things (IoT)

- Application-specific computer system
- Built into a larger system
- Often with real-time computing constraints

Adding embedded systems to larger systems

- Better performance
- More functions and features
- Lower cost, for e.g., through automation
- More dependability

Examples: smartphones, smart watches, printers, gaming consoles, wireless routers



- CPU (Central Processing Unit):**
 - A CPU is the primary component of a computer that carries out instructions of a computer program by performing basic arithmetic, logical, control, and input/output (I/O) operations specified by the instructions.
 - CPUs are typically found in general-purpose computing devices like desktops, laptops, servers, and smartphones.
 - They are designed for high-performance computing tasks and can execute a wide range of software applications.
- MCU (Microcontroller Unit):**
 - An MCU is a compact integrated circuit that includes a microprocessor core, memory, and various peripherals such as input/output ports, timers, analog-to-digital converters, etc., all on a single chip.
 - MCUs are designed for embedded systems, which are systems dedicated to specific functions within larger mechanical or electrical systems.
 - They are commonly used in devices like consumer electronics, automotive systems, industrial control systems, medical devices, etc.
 - MCUs are optimized for low-power operation and real-time processing, making them suitable for applications that require precise control and minimal energy consumption.

consumption.

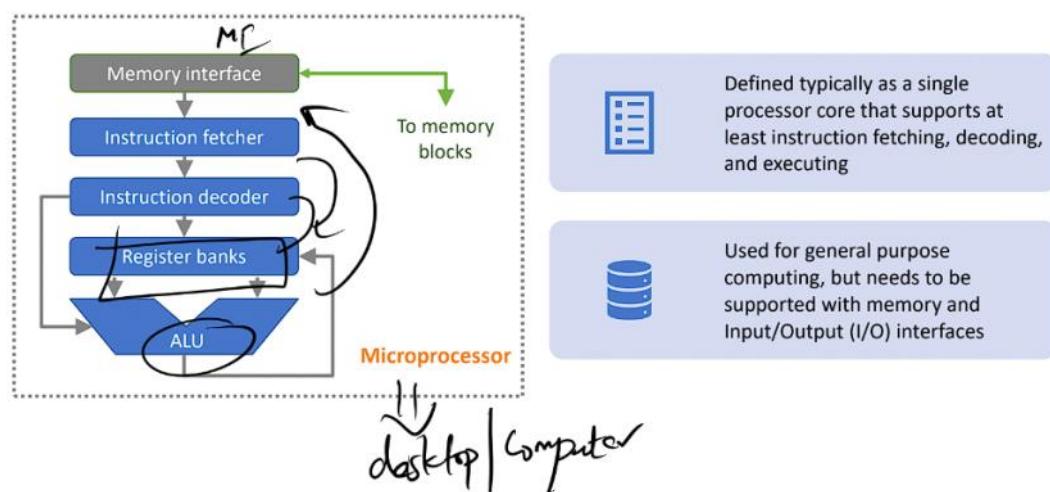
- Embedded Systems:**
 - An embedded system is a specialized computing system designed to perform dedicated functions within a larger mechanical or electrical system.
 - Embedded systems typically consist of an MCU or other specialized processors, along with software tailored to the specific requirements of the application.
 - They are often deployed in devices where reliability, real-time operation, and/or low-power consumption are critical, such as automotive electronics, household appliances, medical devices, industrial automation, etc.
 - Embedded systems can range from simple systems with minimal processing power to complex systems with sophisticated functionalities.

In summary, while CPUs are general-purpose processors found in traditional computing devices, MCUs are specialized integrated circuits designed for embedded systems, which are dedicated computing systems optimized for specific applications and tasks.



CPUs → MCUs → Embedded Systems

Microprocessor or Central Processing unit (CPU)



CPUs → MCUs → Embedded Systems

Embedded system

- Typically implemented using MCUs
- Often integrated into a larger mechanical or electrical system
- Usually has real-time constraints

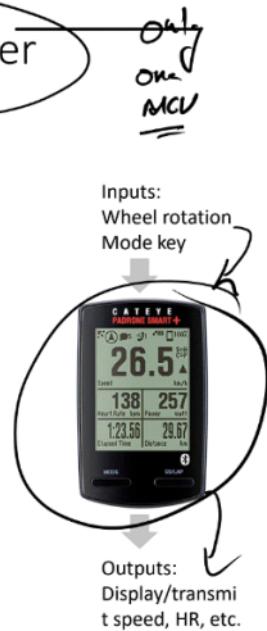


Page 7 / 29

- +

Embedded system example: Bike computer

- Functions
 - Speed, cadence, distance, heart rate (HR) measurements
- Constraints
 - Size, weight, and cost; power and energy
- Inputs (Sensor)
 - Wheel rotation sensor and mode key
- Output
 - Liquid crystal display (LCD), BLE interface to smartphone
- Uses low performance microcontroller



L what are Automotive Embedded Systems ?

Show drafts ▾ 🔍

- 💡 In the realm of automobiles, **Automotive Embedded Systems (AES)** play a crucial role in controlling and managing various functionalities. They are essentially dedicated computer systems designed to perform specific tasks within a vehicle, impacting everything from safety to performance and entertainment.

Here's a deeper dive into their essence:

What they comprise:

- **Hardware:** Microcontrollers (brains), sensors (gather data), actuators (translate instructions into actions), and communication ports (connect to other systems).
- **Software:** Embedded software programs the system to achieve specific goals, like engine control, braking, or infotainment.

Where you find them:

- **Safety features:** Airbags, anti-lock braking systems (ABS), traction control, and lane departure warnings.
- **Performance systems:** Engine control units (ECUs), transmission control, and fuel injection.
- **Comfort and convenience:** Infotainment systems, navigation, climate control, and power windows.
- **Emerging technologies:** Driver-assistance systems, autonomous driving, and

- **Emerging technologies:** Driver-assistance systems, autonomous driving, and connected car features.

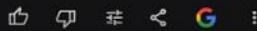
Key benefits:

- **Enhanced safety:** By constantly monitoring and adjusting car functions, AES can prevent accidents and minimize their impact.
- **Improved performance:** Optimized engine control and fuel efficiency lead to better fuel economy and overall performance.
- **Greater comfort and convenience:** User-friendly features and automated systems create a more enjoyable driving experience.
- **Advanced functionalities:** Pave the way for self-driving cars and intelligent transportation systems.

Future outlook:

As technology evolves, Automotive Embedded Systems are expected to become even more sophisticated, integrating with artificial intelligence, advanced sensors, and cloud connectivity. This will further improve safety, performance, and user experience in automobiles.

So, the next time you're behind the wheel, remember that these little systems play a big role in making your ride smoother, safer, and more enjoyable.

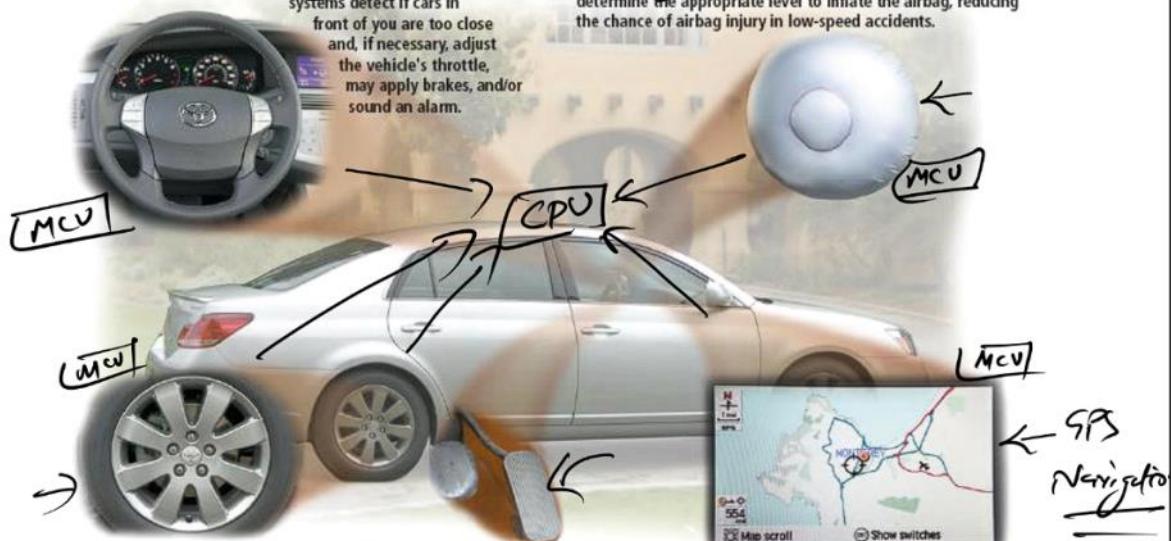


Automotive Embedded Systems

(self-drive car)

Adaptive cruise control systems detect if cars in front of you are too close and, if necessary, adjust the vehicle's throttle, may apply brakes, and/or sound an alarm.

Advanced airbag systems have crash-severity sensors that determine the appropriate level to inflate the airbag, reducing the chance of airbag injury in low-speed accidents.



Page 10 / 29 - +

Automotive Embedded Systems

- Today's high-end automobile may have 100+ microprocessors:

Seat belt; dashboard devices; engine control; ABS; automatic stability control; navigation system; infotainment system; collision avoidance system; tire pressure monitoring; lane warning; adaptive cruise control; climate control; airbag control unit; electric window and central locking; parking aid; automatic wiper control; alarm and immobilizer; power seat; electric power steering; electronic transmission; active suspension

Project

penetration

Power

Steer. li.

Ref. ly.

Zebra Cross

Other car

side mirror

back mirror

A/C

Page 11 / 29 - +

Characteristics of Embedded Systems

- Dedicated functionality ✓
- Real-time operation ✓
- Small size and low weight ✓
- Low power ✓
- Harsh environments ✓
- Safety-critical operation ✓
- Cost sensitive ✓

Embedded Systems:

- **Definition:** Dedicated computer systems designed for specific tasks within a larger device or system.
- **Focus:** Functionality and resource efficiency within limited hardware and software constraints.
- **Examples:** Microcontrollers in appliances, automotive ECUs, industrial control systems.
- **Key attributes:** Small footprint, low power consumption, real-time responsiveness (optional).

Real-Time Systems:

- **Definition:** Systems where the correctness of their operation depends on **timing constraints**.
- **Focus:** Meeting strict deadlines for processing and data responses to ensure system stability and safety.
- **Examples:** Flight control systems, medical equipment, industrial process control, high-frequency trading systems.
- **Key attributes:** Guaranteed response times, deterministic behavior, predictable performance.

Key Differences:

- **Emphasis:** Embedded systems prioritize specific functionalities, while real-time systems prioritize timely responses.
- **Constraints:** Embedded systems have resource limitations, while real-time systems have stricter timing constraints.
- **Complexity:** Embedded systems can be simpler, while real-time systems often require specialized hardware and software for guaranteed timing.

Overlap:

- Many embedded systems also have real-time requirements, like automotive braking systems.
- Real-time systems often use embedded hardware and software platforms.

Summary:

Think of **embedded systems** as dedicated tools with specific functionalities, while **real-time systems** are more like precise instruments focused on guaranteed response times. Both play crucial roles in various applications, and understanding their differences helps in choosing the right system for the right job.



Your statement is almost correct, but with a slight nuance:

All embedded systems are not real-time:

While it's true that many embedded systems have some level of timing sensitivity, **not all of them require strict real-time guarantees**. For example:

- A simple embedded system like a calculator or basic thermostat may not have critical deadlines and may operate comfortably without guaranteed response times.
- Even within complex systems like cars, not all embedded controls are real-time. Infotainment systems might have some responsiveness requirements, but they are less critical compared to real-time systems like engine control units.

Most real-time systems are embedded:

It's true that **most real-time systems are implemented on dedicated embedded hardware and software platforms**. This is because they often require specialized features like:

- Microcontrollers with predictable processing times
- Real-time operating systems optimized for responsiveness
- Communication protocols ensuring timely data delivery

However, there are some exceptions:

hardware and software platforms. This is because they often require specialized features like:

- Microcontrollers with predictable processing times
- Real-time operating systems optimized for responsiveness
- Communication protocols ensuring timely data delivery

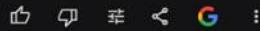
However, there are some exceptions:

- High-performance computing systems can be used for real-time applications like financial trading, but they wouldn't necessarily fit the traditional definition of "embedded."
- High-end industrial control systems may use specialized hardware that isn't considered "embedded" in the strictest sense.

In summary:

- **Not all embedded systems are real-time.**
- **Most real-time systems are embedded.**
- There are some exceptions on both sides, highlighting the nuanced relationship between these two types of systems.

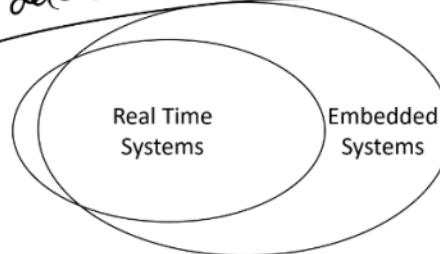
I hope this clarifies the distinction!



Embedded vs. Real Time Systems

- Embedded system: is a computer system that performs a limited set of specific functions; it often interacts with its environment
- RTS: Correctness of the system depends not only on the logical results, but also on the **time** in which the results are produced

*Tire pressure
pedestrian detection - in self drive car - RTS*

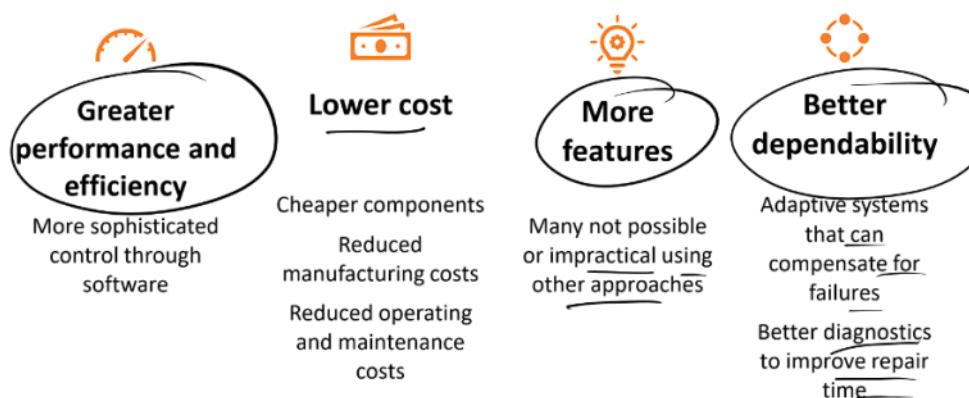


Examples?

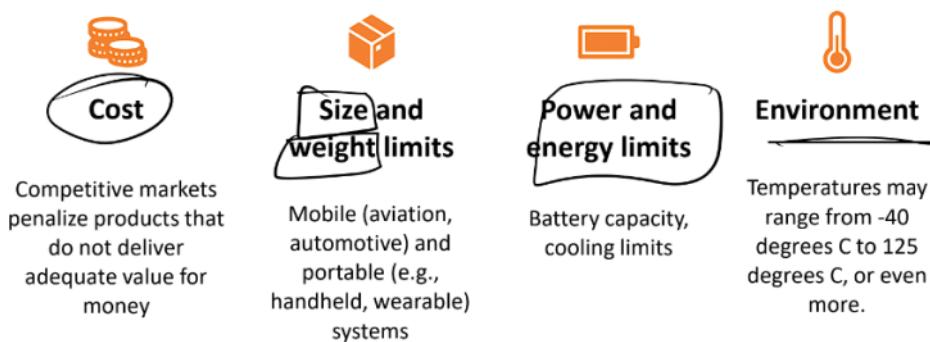
Examples

- Real Time Embedded:
 - Nuclear reactor control ✓
 - Flight control ✓
 - Basically, any safety critical system —
 - GPS ✓
- Real Time, but not Embedded:
 - Stock trading system
 - Skype
 - Pandora, Netflix
- Embedded, but not Real Time:
 - Home temperature control ⇒
 - Sprinkler system
 - Washing machine, refrigerator, etc.

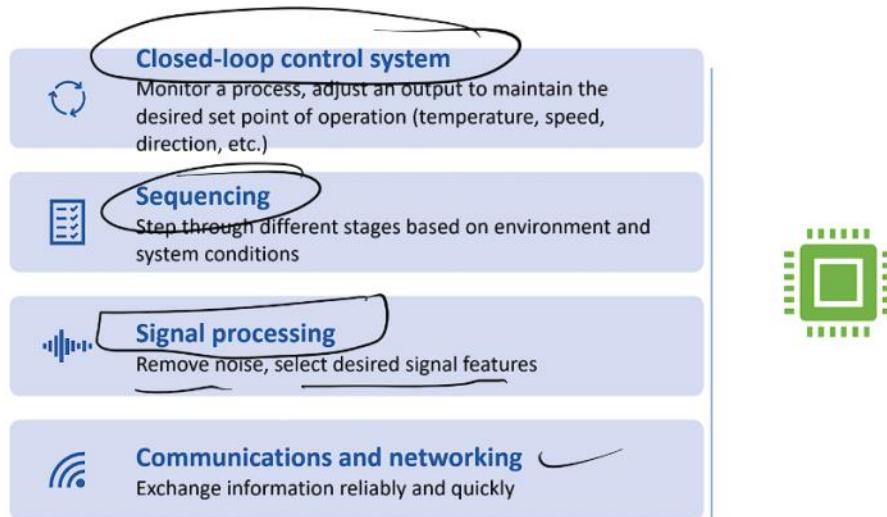
Benefits of embedded systems



Constraints specific to embedded devices



Functions of embedded systems





ChatGPT

Event-driven (reactive) and time-driven real-time systems are two fundamental paradigms for designing and managing real-time systems, each with distinct characteristics and approaches to handling tasks and events within a system. Here's a comparison between the two:

1. **Event-Driven (Reactive) Real-Time Systems:**

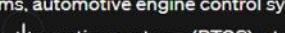
- **Definition:** In event-driven or reactive real-time systems, tasks are triggered by external events or interrupts rather than being executed at fixed intervals. These events can include sensor inputs, user interactions, communication signals, or other external stimuli.
- **Characteristics:**
 - Tasks are initiated in response to specific events or occurrences, allowing the system to react dynamically to changes in its environment.
 - The system is event-triggered, meaning that tasks execute asynchronously in response to events, which can lead to non-deterministic behavior.
 - Event-driven systems are well-suited for applications where events occur sporadically and unpredictably, and where timely responses to these events are critical.
 - Programming models for event-driven systems often involve event loops, callbacks, and event handlers to manage the flow of control and handle asynchronous events.



- **Examples:** Automotive control systems (e.g., ABS triggering based on wheel speed sensors), interactive multimedia applications, graphical user interfaces (GUIs), network protocols, etc.

2. **Time-Driven Real-Time Systems:**

- **Definition:** In time-driven real-time systems, tasks are executed according to predetermined schedules or deadlines, with fixed time intervals governing task execution. These systems rely on periodic timers or clock signals to synchronize task execution.
- **Characteristics:**
 - Tasks are scheduled to run at specific time intervals or deadlines, providing determinism and predictability in task execution.
 - The system operates according to a fixed time-triggered schedule, ensuring that tasks execute predictably without being influenced by external events or stimuli.
 - Time-driven systems are suitable for applications where tasks must be executed with precise timing and where predictability and determinism are paramount.
 - Programming models for time-driven systems often involve cyclic executive algorithms, periodic task scheduling, and time-triggered communication protocols to ensure timely task execution and coordination.
- **Examples:** Aircraft flight control systems, automotive engine control systems, industrial automation systems, real-time operating systems (RTOS), etc.



Message ChatGPT

In summary, event-driven (reactive) real-time systems respond to external events or stimuli asynchronously, while time-driven real-time systems execute tasks according to fixed schedules or deadlines. Event-driven systems offer flexibility and responsiveness to unpredictable events, while time-driven systems provide determinism and predictability in task execution. The choice between these paradigms depends on the specific requirements of the application, including timing constraints, event occurrence patterns, and the need for determinism and predictability.



Sensors

- A sensor is a device that receives a stimulus and responds with an electrical signal.

Sensors

Sensors are devices that measure a particular property of the environment and convert that into an electrical output

Typically, sensors have a linear function between the physical property measured and the electrical output

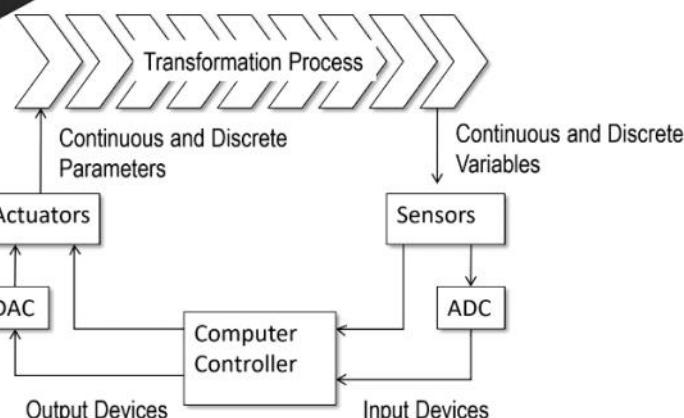
The sensitivity of a sensor indicates the minimum value of the measured input that can produce a certain output signal

An analog-to-digital converter employed to turn a sensor output into signals that can be processed by MCUs

Different physical phenomena underpin the operation of sensors, e.g., the piezoelectric effect (accumulation of electric charge when mechanical pressure is applied)

IoT-Lecture-07 Sensor.pdf Open... Share L

Computer Process Control System





Show drafts ▾



A Deeper Dive into Computer Process Control Systems (CPCS)

Expanding on our previous discussions, here's a more in-depth breakdown of CPCS:

What they are:

CPCS are automated systems that monitor and control physical processes using computers. They continuously measure key variables (e.g., temperature, pressure, flow rate) and adjust them to achieve desired outcomes, ensuring consistent quality, efficiency, and safety.

Key components:

1. **Sensors:** Collect data about various process parameters in real-time. Examples include temperature sensors, pressure gauges, flow meters, and analytical instruments.
2. **Controllers:** The central processing unit of the system. They receive sensor data, compare it to setpoints (desired values), and send control signals to actuators based on programmed algorithms.
3. **Actuators:** Physical devices that translate control signals into actions. Examples include valves, pumps, heaters, and motors.
4. **Software:** Includes control algorithms, user interfaces, and communication protocols. Algorithms analyze sensor data and determine control actions, while interfaces allow human interaction and monitoring.
5. **Communication network:** Connects all components, enabling data exchange and coordinated operation.

How they work:

1. Sensors continuously gather data and transmit it to the controller.
2. The controller compares the data to setpoints and applies the control algorithm.
3. Based on the analysis, the controller sends signals to actuators.
4. Actuators adjust process variables accordingly.
5. The cycle repeats continuously, maintaining the process within desired parameters.

What is a Stimulus?

- Motion, position, displacement
- Velocity and acceleration
- Force, strain
- Pressure
- Flow
- Sound
- Moisture
- Light
- Radiation
- Temperature
- Chemical presence



Visual Sensor



Ultrasound Sensor



Infrared Sensor

Transducers are fundamental components in the Internet of Things (IoT) ecosystem, acting as the bridge between the physical world and the digital realm. They essentially **convert one form of energy into another**, enabling communication and interaction between physical systems and IoT devices.

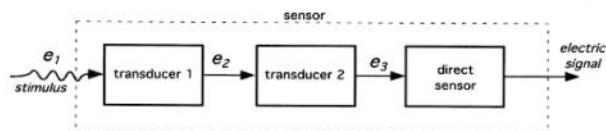
Here's a breakdown of their role in IoT:

What they do:

- **Sensors:** These are input transducers that convert physical phenomena (like temperature, pressure, light, or motion) into electrical signals that can be processed by digital systems. In IoT, various sensors collect data from the environment or equipment, forming the basis for further analysis and decision-making.
- **Actuators:** These are output transducers that convert electrical signals into physical actions. In IoT, actuators receive instructions from controllers and translate them into real-world changes, like turning on lights, adjusting thermostats, or controlling motors.

Sensor as Energy Converter

- This conversion can be direct or it may require transducers



- Example:

A chemical sensor may have a part which converts the energy of a chemical reaction into heat (transducer) and another part, a thermopile, which converts heat into an electrical signal.

Microphone, Loud Speaker, Biological Senses (e.g. touch, sight,..., etc)

Physical Principles of Sensing

- Charges, fields, & potentials
- Capacitance
- Magnetism
- Induction
- Resistance
- Piezoelectric effect
- Seebeck and Peltier effects
- Thermal properties of materials
- Heat transfer
- Light

Sensor Types: HW & SW

- **Hardware-based sensors**

- Physical components built into a device
- They derive their data by directly measuring specific environmental properties

- **Software-based sensors**

- Not physical devices, although they mimic hardware-based sensors
- They derive their data from one or more hardware-based sensors

L give examples of software based sensors in real life

Software-based sensors are becoming increasingly prominent in various domains of real life. Here are some compelling examples:

In your pocket:

- **Activity tracking:** Your smartwatch or fitness tracker uses software algorithms to analyze data from motion sensors (accelerometer, gyroscope) to estimate steps, distance, and even sleep patterns.
- **Heart rate monitoring:** Software interprets signals from optical sensors on your wrist, translating them into your heart rate, providing insights into health and fitness.
- **Fingerprint/Face ID:** Algorithms analyze camera data to accurately recognize your unique features for secure device unlocking.

In your home:

Hardware Sensors:

- **Physical devices:** These are tangible components that directly interact with the environment to measure physical phenomena.
- **Examples:** Temperature sensors, pressure sensors, light sensors, motion sensors, microphones, cameras, etc.
- **Functionality:** They convert physical properties into electrical signals that can be processed by microcontrollers or computers.
- **Analog or digital:** Some sensors provide analog outputs proportional to the measured value, while others have digital outputs with discrete levels.
- **Advantages:**
 - Direct measurement of physical phenomena.
 - Reliable and accurate data acquisition.
 - Wide range of available sensors for various applications.
- **Disadvantages:**
 - Can be costly depending on the sensor type and accuracy.
 - Require physical integration into the system, adding complexity.
 - May have limitations in terms of size, power consumption, or environmental resistance.

Software Sensors:

Types of Sensors

- **Active**

- send signal into environment and measure interaction of signal w/ environment
- e.g. radar, sonar

- **Passive**

- record signals already present in environment
- e.g. video cameras

Active and passive sensors are two fundamental types of sensors that differ in their operation and energy requirements.

Active sensors emit their own energy source to interact with the environment and collect data. They generate a signal that is then reflected or absorbed by the target. The sensor then measures the change in the signal to determine the state of the environment.

Examples of active sensors include:

- **Radar:** Emits radio waves and measures the reflected signal to determine the distance and speed of objects.
- **Sonar:** Emits sound waves and measures the reflected signal to determine the depth of water or the location of objects.
- **Laser rangefinder:** Emits laser pulses and measures the time it takes for the pulses to return to determine the distance to objects.

Passive sensors do not emit their own energy source and instead rely on energy from the environment to collect data. They absorb or reflect energy from the environment and measure the change in the signal.

Examples of passive sensors include:

- **Infrared sensors:** Detect infrared radiation emitted by objects, such as body heat or heat from appliances.
- **Photodiodes:** Convert light into electrical signals.
- **Magnetometers:** Detect magnetic fields.

Actuators

- Hardware devices that convert a controller command signal into a change in a physical parameter
 - The change is usually mechanical (e.g., position or velocity)
 - An actuator is also a transducer because it changes one type of physical quantity into some alternative form
 - An actuator is usually activated by a low-level command signal, so an amplifier may be required to provide sufficient power to drive the actuator

Types of Actuators

1. Electrical actuators

- Electric motors
 - DC servomotors
 - AC motors
 - Stepper motors
- Solenoids



2. Hydraulic actuators

- Use hydraulic fluid to amplify the controller comm signal



3. Pneumatic actuators

- Use compressed air as the driving force



Actuators are the workhorses of the physical world, converting various forms of energy into mechanical motion to perform actions. Depending on the energy source used, we have three main types: electrical, hydraulic, and pneumatic. Each possesses unique characteristics and suits specific applications:

1. Electrical Actuators:

- **Energy source:** Electricity from batteries, power lines, or other sources.
- **Types:**
 - **Solenoids:** Generate linear motion using electromagnetism. Ideal for quick, short-stroke movements like valves or locks.
 - **Electric motors:** Offer a wide range of speed, torque, and power options. Can be AC or DC, linear or rotary, geared or brushless. Used in robots, drones, conveyor belts, etc.
 - **Piezoelectric actuators:** Utilize small changes in shape due to applied electric fields, providing precise, high-frequency movements for applications like inkjet printers or microfluidics.

2. Hydraulic Actuators:

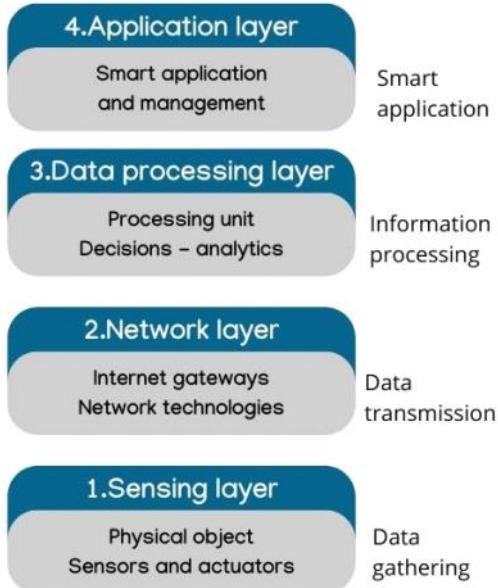
- **Energy source:** Incompressible fluid (usually oil) pressurized by pumps.
- **Types:**
 - **Hydraulic cylinders:** Provide linear motion by converting fluid pressure into force. Used in heavy machinery, construction equipment, agricultural vehicles, etc.
 - **Hydraulic motors:** Convert fluid pressure into rotary motion. Used in industrial robots, machine tools, wind turbines, etc.
- **Advantages:**
 - High force and power output for heavy-duty applications.
 - Precise control through valves and flow control.
 - Relatively efficient energy transfer.
- **Disadvantages:**
 - Requires pumps, hoses, and valves, increasing complexity and cost.
 - Leaks can create environmental hazards.
 - Maintenance requirements due to fluid contamination and wear.

3. Pneumatic Actuators:

- **Energy source:** Compressed air.
- **Types:**
 - **Pneumatic cylinders:** Similar to hydraulic cylinders, but use compressed air instead of fluid. Often used in lighter-duty applications like packaging machinery, robotic arms, or door openers.
 - **Pneumatic motors:** Convert compressed air into rotary motion. Used in automation equipment, textile machinery, and other industrial applications.
- **Advantages:**
 - Simple design and relatively low cost.
 - Clean and quiet operation.
 - High response speed.
- **Disadvantages:**
 - Limited force compared to hydraulic actuators.
 - Air compressibility can make precise control challenging.
 - Requires an air compressor and distribution system.

Choosing the Right Actuator:

IOT ARCHITECTURE :



1. Sensing Layer –

The sensing layer is the first layer of the IoT architecture and is responsible for collecting data from different sources. This layer includes sensors and actuators that are placed in the environment to gather information about temperature, humidity, light, sound, and other physical parameters. These devices are connected to the network layer through wired or wireless communication protocols.

2. Network Layer –

The network layer of an IoT architecture is responsible for providing communication and connectivity between devices in the IoT system. It includes protocols and technologies that enable devices to connect and communicate with each other and with the wider internet. Examples of network technologies that are commonly used in IoT include WiFi, Bluetooth, Zigbee, and cellular networks such as 4G and 5G. Additionally, the network layer may include gateways and routers that act as intermediaries between devices and the wider internet, and may also include security features such as encryption and authentication to protect against unauthorized access.

3. Data processing Layer –

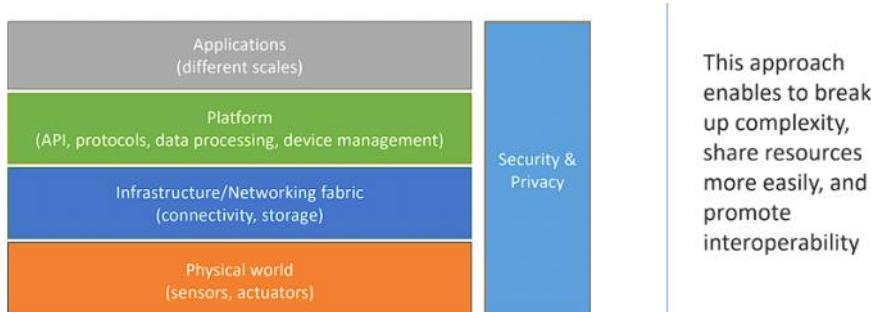
The data processing layer of IoT architecture refers to the software and hardware components that are responsible for collecting, analyzing, and interpreting data from IoT devices. This layer is responsible for receiving raw data from the devices, processing it, and making it available for further analysis or action. The data processing layer includes a variety of technologies and tools, such as data management systems, analytics platforms, and machine learning algorithms. These tools are used to extract meaningful insights from the data and make decisions based on that data. Example of a technology used in the data processing layer is a data lake, which is a centralized repository for storing raw data from IoT devices.

4. Application Layer –

The application layer of IoT architecture is the topmost layer that interacts directly with the end-user. It is responsible for providing user-friendly interfaces and functionalities that enable users to access and control IoT devices. This layer includes various software and applications such as mobile apps, web portals, and other user interfaces that are designed to interact with the underlying IoT infrastructure. It also includes middleware services that allow different IoT devices and systems to communicate and share data seamlessly. The application layer also includes analytics and processing capabilities that allow data to be analyzed and transformed into meaningful insights. This can include machine learning algorithms, data visualization tools, and other advanced analytics capabilities.

A layered view to IoT architecture

At a high level, stakeholders may converge to a shared vision

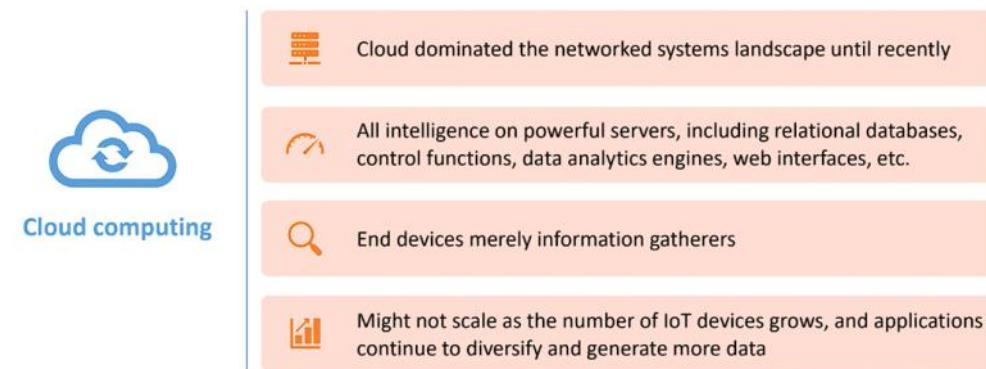


Advantages of the layer approach

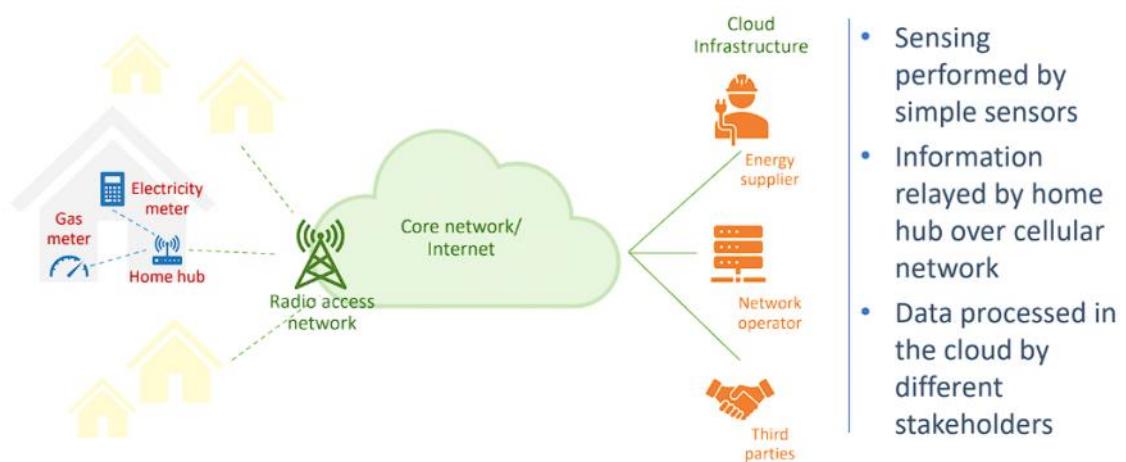
- Allows IoT device manufacturers to focus strictly on improving their performance, power consumption, etc. – expose only well-defined interfaces to software platforms.
- Easier to share and partition strictly the network and computing resources (slicing); reducing the burden on service providers to build and manage networks – Infrastructure/Network as a Service (IaaS/NaaS)
- Enables software/app developers to build applications without having to understand the specifics of a device – Platform as a Service (PaaS)

Cloud vs. Fog vs. Edge

The information processing view



Example: Smart metering



Cloud vs. Fog vs. Edge

The information processing view



Pushing some of the intelligence closer to the device, for e.g., to access networks or gateways



This includes data aggregation, compression, (partial) processing, making localized decisions



IoT devices kept simple, no direct communication with end servers, still battery powered



Resource management implemented across different network layers – management could be regarded as an application



Fog computing

The role of gateways in fog architectures



Data filtering and processing (for e.g., aggregation of summaries, compression, etc.)



Protocol translation and interfacing between different connectivity technologies



Data flow multiplexing, packet routing



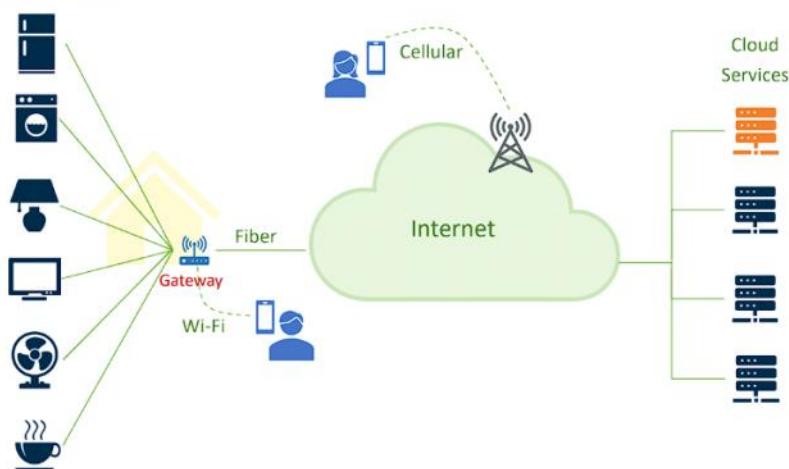
Security (for e.g., data encryption, firewalling)



Scalability problem: as the number of devices grows, so will the number of gateways that are required

Example: Home automation

Home appliances



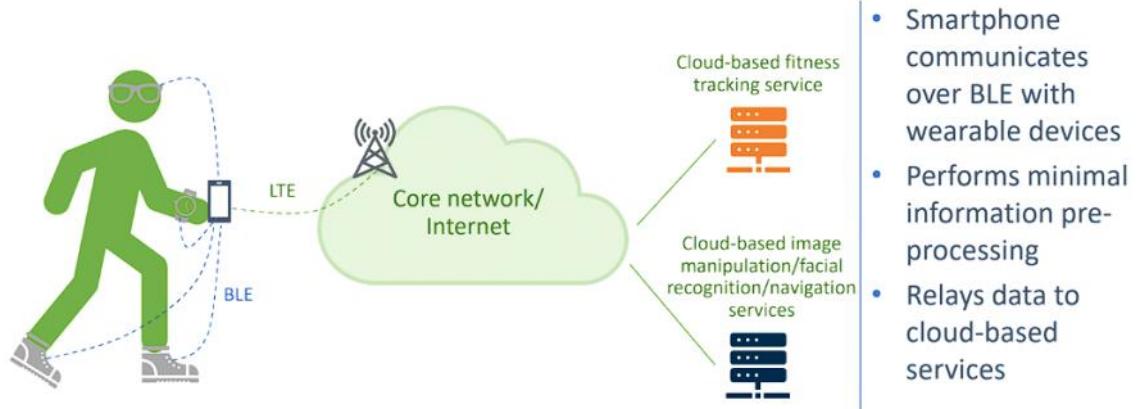
- Gateway performs protocol translation
- Incorporates basic network intrusion detection system
- Cloud services continue to perform analytics

Smartphones as gateways

The fitness and healthcare domain

- Embed multiple networking technologies (Wi-Fi, 3G/4G, Bluetooth/BLE, NFC, etc.)
- Run full TCP/IP stacks, thus maintain end-to-end connectivity with cloud
- Can connect to multiple devices within close proximity simultaneously
- Ability to enforce secure transport (e.g., TLS/HTTPS)
- Sufficient computing power to pre-process/augment collected data

Example: Wearables



Cloud vs. Fog vs. Edge

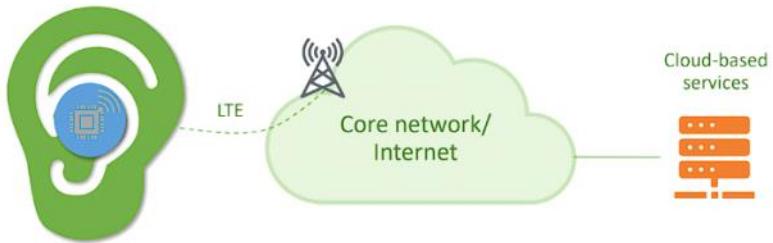
The information processing view



Edge computing

- Pushing compute power, communication capabilities, intelligence down at device level
- Processing as much as possible where data is collected
- Transmitting only key information or summaries
- Enabling new applications: automotive IoT, virtual/augmented reality, in-ear computing

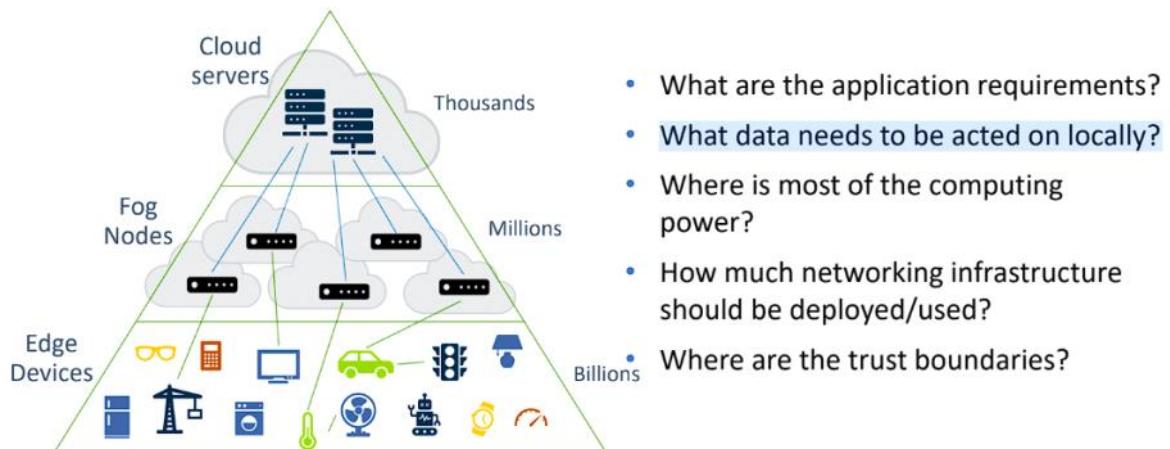
Example: Hearables



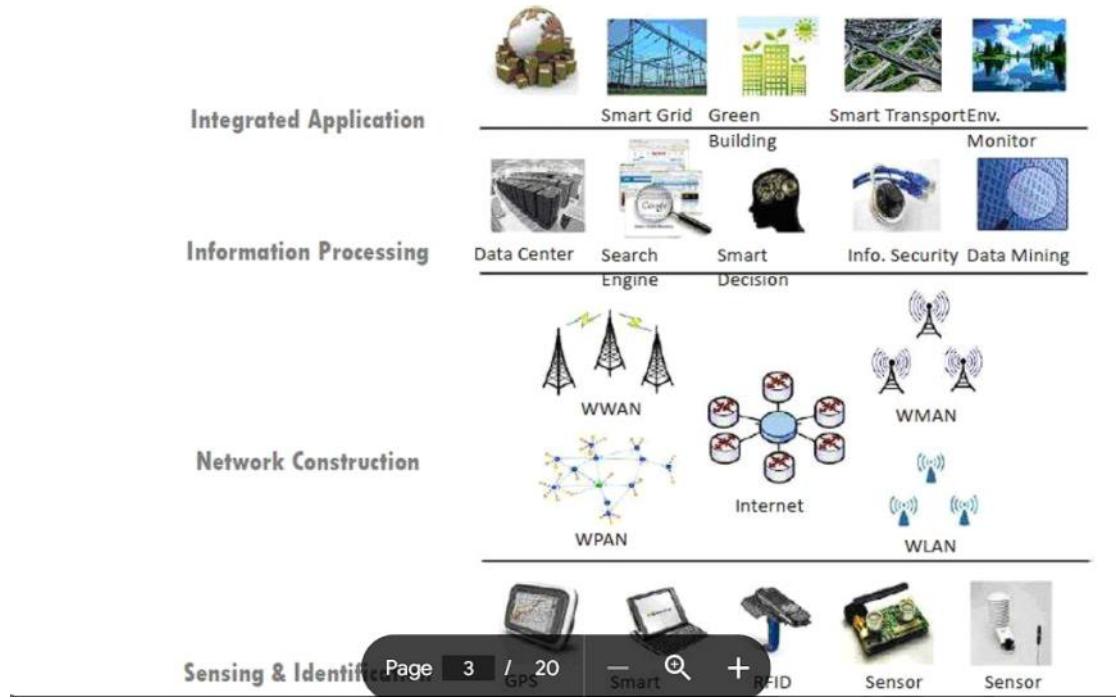
- **Hardware:** Low-power chips specialized in computationally intensive tasks (Arm Ethos)
- **Software:** AI libraries optimized for constrained devices (uTensor)
- **Neural networks:** compressed/pruned models

Choosing the right IoT architecture

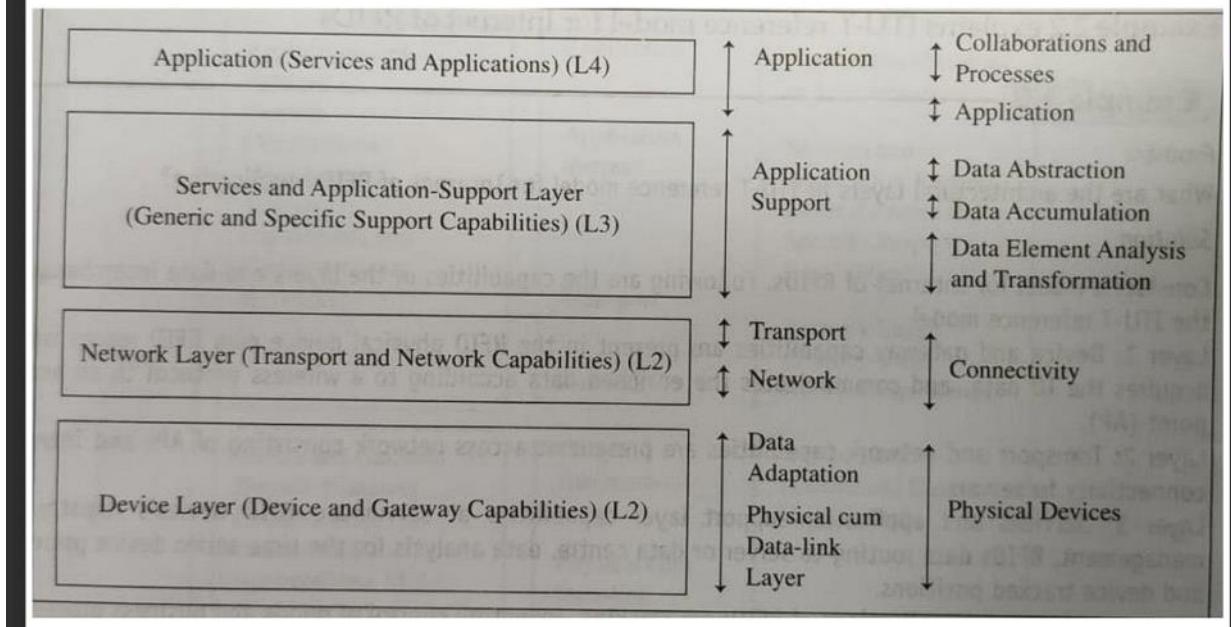
Performance and cost remain the dominant architectural drivers



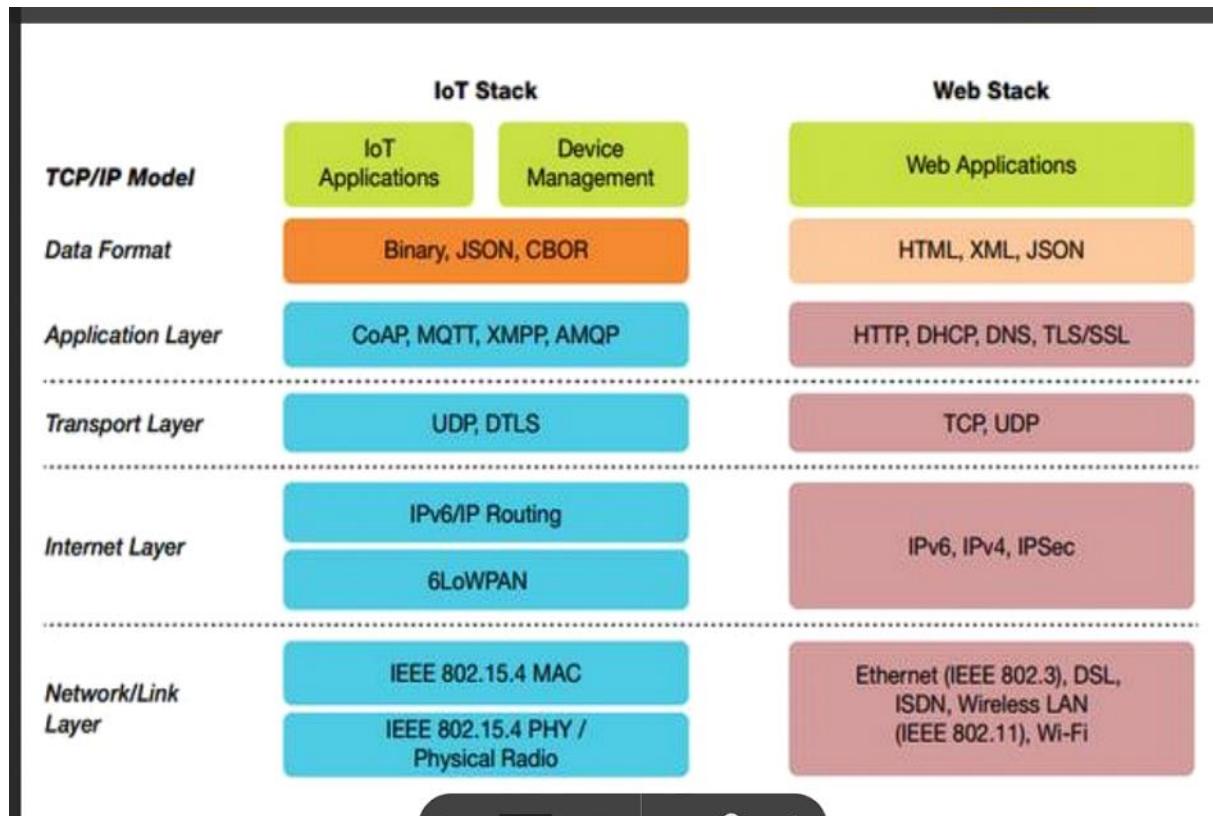
IoT 4 Layers model



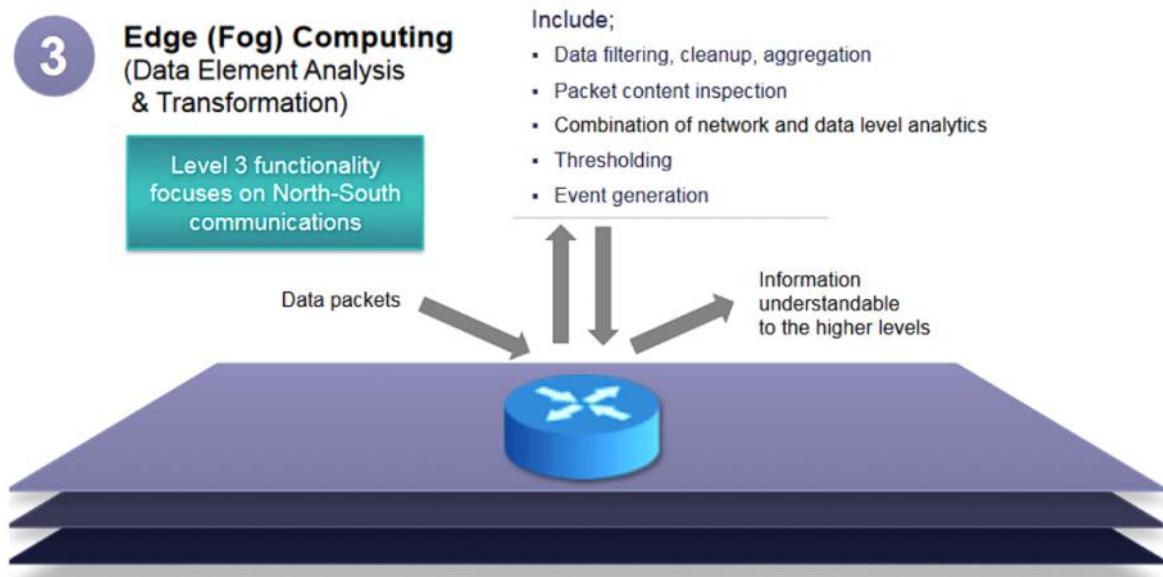
Comparison



COMPARISON BTW TCP/IP STACK AND IOT STACK !!



3 MAIN FEATURES OF THE PLATFORM LAYER !! IN IOT ARCHITECTURE :



4

Data Accumulation (Storage)

- Event filtering/sampling
- Event comparison
- Event joining for CEP
- Event based rule evaluation
- Event aggregation
- Northbound/southbound alerting
- Event persistence in storage

Query Based Data Consumption



Event Based Data Generation

or



Making network data usable by applications

1. Converts data-in-motion to data-at-rest
2. Converts format from network packets to database relational tables
3. Achieves transition from 'Event based' to 'Query based' computing
4. Dramatically reduces data through filtering and selective storing

**5**

Data Abstraction (Aggregation & Access)

Abstracting the data interface for applications

Information Integration

1. Creates schemas and views of data in the manner that applications want
2. Combines data from multiple sources, simplifying the application
3. Filtering, selecting, projecting, and reformatting the data to serve the client applications
4. Reconciles differences in data shape, format, semantics, access protocol, and security

Remarks



OSI MODEL

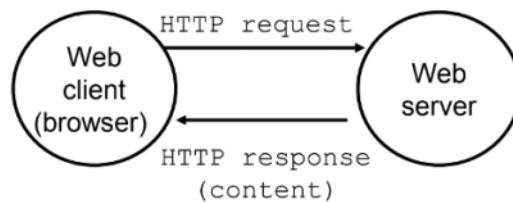
TCP/IP MODEL

Application Layer	Application Layer
Presentation Layer	
Session Layer	
Transport Layer	Transport Layer
Network Layer	Internet Layer
Data Link Layer	Network Access Layer
Physical Layer	

PROTOCOLS USED BY IOT ECOSYSTEM :

HyperText Transfer Protocol (HTTP)

- Clients and servers communicate using the **HyperText Transfer Protocol (HTTP)**
 - Client and server establish TCP connection
 - Client requests content
 - Server responds with requested content
 - Client and server close connection (usually)



You're correct! MIME stands for **Multipurpose Internet Mail Extensions**, and it's a technical standard that revolutionized email by allowing us to send and receive more than just plain text messages. Here's how it works:

Before MIME:

- Early email messages were limited to plain text only.
- Images, sounds, and other file types couldn't be attached.
- Non-English characters displayed incorrectly due to lack of character encoding support.

With MIME:

- MIME acts as an "add-on" to email protocols like SMTP.
- It defines different **content types** to identify the data attached to an email (e.g., image/jpeg, audio/mpeg, text/html).
- It allows **encoding** the data for secure transmission and proper display on different email clients.
- Email messages can now have **multiple parts** with text, attachments, and alternative versions for compatibility.

Web Content

- Web servers return **content** to clients
 - a sequence of bytes with an associated **MIME (Multipurpose Internet Mail Extensions) type**
- **Example MIME types**

• text/html	HTML document
• text/plain	Unformatted text
• application/postscript	Postscript document
• image/gif	Binary image encoded in GIF format
• image/jpeg	Binary image encoded in JPEG format

Static & Dynamic Content

- The content returned in HTTP responses can be either **static** or **dynamic**
 - Static content: content stored in files and retrieved in response to an HTTP request
 - Examples: HTML files, images, audio clips
 - Dynamic content: content produced on-the-fly in response to an HTTP request
 - Example: content produced by a program executed by the server on behalf of the client
- Bottom line: all web content is associated with a **file** that is managed by the server

URLs

- Each file managed by a server has a unique name called a **URL** (**Universal Resource Locator**)
- URLs for static content:
 - `http://www.cse.nd.edu:80/index.html`
 - `http://www.cse.nd.edu/index.html`
 - `http://www.cse.nd.edu`
 - Identifies a file called `index.html`, managed by a web server at `www.cse.nd.edu` that is listening on port 80
- URLs for dynamic content:
 - `http://www.cse.nd.edu:8000/cgi-bin/adder?15000&213`
 - Identifies an executable file called `adder`, managed by a web server at `www.cse.nd.edu` that is listening on port 8000, that should be called with two argument strings: 15000 and 213

HTTP Requests

- HTTP request is a **request line**, followed by zero or more **request headers**
- Request line:
 - `<method> <uri> <version>`
 - `<version>` is HTTP version of request (HTTP/1.0 or HTTP/1.1)
 - `<uri>` is typically URL for proxies, URL suffix for servers
 - `<method>` is either GET, POST, OPTIONS, HEAD, PUT, DELETE, or TRACE

REST ARCHITECTURE STYLE :

Nouns, Verbs, and Representations in REST APIs: Building Blocks of Resourceful Communication

In RESTful APIs, nouns, verbs, and representations work together to create a clear and consistent way to interact with resources. Here's how they fit into the picture:

Nouns:

- **Represent resources:** RESTful APIs treat real-world entities like data, objects, or actions as **resources**.
- These resources are **identified by nouns**, making them easy to understand and predictable.
- Examples: `/users` , `/products` , `/orders` , `/comments` .

Verbs:

- **Represent actions on resources:** HTTP methods serve as the verbs in RESTful communication.
- Standardized methods like **GET, POST, PUT, DELETE** map to CRUD operations (Create, Read, Update, Delete) on the identified resource.
- Understanding the meaning of each verb clarifies the intended action on the resource.

- Understanding the meaning of each verb clarifies the intended action on the resource.

Representations:

- **Describe the format of data exchanged:** Resources are not just abstract entities; they have data associated with them.
- RESTful APIs use **representations** to specify the format of this data.
- Common formats include JSON, XML, plain text, and others, negotiated through content-type headers in requests and responses.

Putting it all together:

Imagine you want to update the address of a user with ID 123. Here's how it would look in a RESTful API:

- **Resource (noun):** `/users/123`
- **Action (verb):** PUT (update)
- **Representation (format):** JSON in request body containing new address data

This approach combines the clarity of nouns for resources, the action-oriented nature of verbs, and the flexibility of representations to create a well-defined communication style.

Benefits:

- **Clarity and predictability:** Anyone familiar with nouns, verbs, and common data

Benefits:

- **Clarity and predictability:** Anyone familiar with nouns, verbs, and common data formats can easily understand RESTful API interactions.
- **Interoperability:** Standardized methods and formats enable different tools and libraries to access RESTful APIs seamlessly.
- **Flexibility:** Representations allow for various data formats to cater to different needs.

CoAP

- Constrained Application Protocol
 - REST-based web transfer protocol
 - manipulates Web resources using the same methods as HTTP: GET, PUT, POST, and DELETE
 - subset of HTTP functionality re-designed for low power embedded devices such as sensors (for IoT and M2M)

Basics of CoAP

- CoAP – **Constrained Application Protocol.**
- CoAP is an IoT Protocol.
- CoAP is designed to allow simple devices to join IoT through the low bandwidth-restricted network.
- CoAP is designed for M2M and IoT applications.
- CoAP is an Application Layer Protocol that follows the request-response model.
- CoAP runs over UDP Protocol.
- CoAP uses fewer resources than HTTP.
- CoAP client can use GET, PUT, and DELETE commands during the session.



CoAP Layers

- CoAP is divided into two layers.
- Upper Layer (Application Layer) is designed for the communication method based on the Request-Response model.
- Lower Layer (UDP) is designed to deal with UDP and asynchronous messages.

Application Layer
Request/Response

UDP
Messages

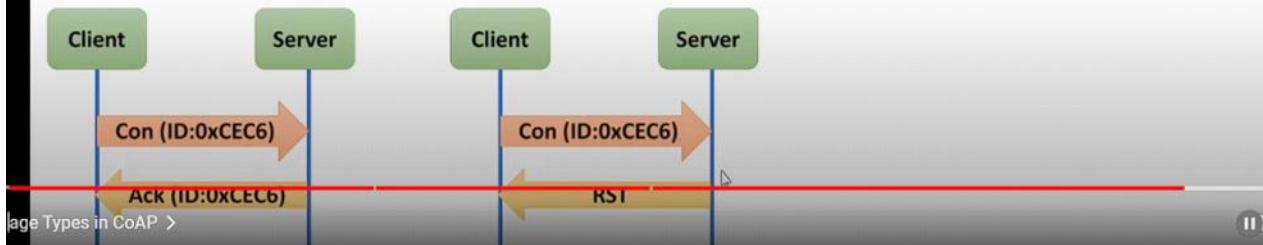


Message Types CoAP

- CoAP Supports Four different message types:

1. Confirmable Message {Reliable Message} (Con):

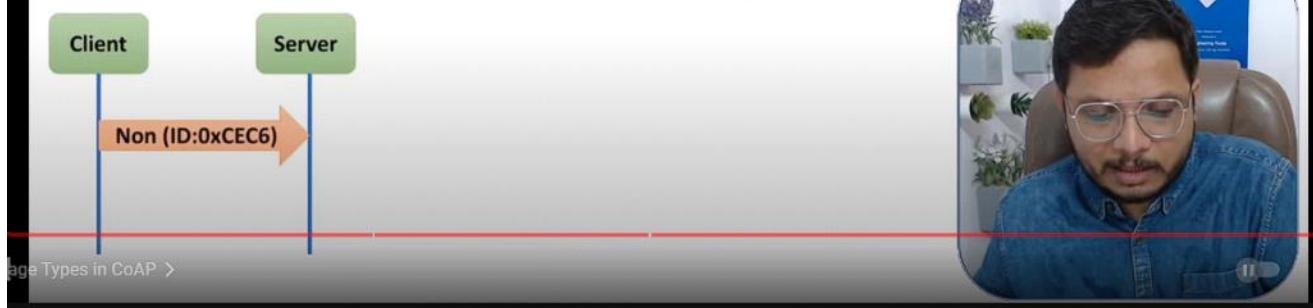
- A Confirmable message requires a response, either a Positive Acknowledgement or a Negative Acknowledgement.
- If acknowledgement is not received then retransmissions are made until all attempts are exhausted.
- The Acknowledgement message contains the same ID as the confirmation message.
- If the server has trouble managing the incoming request, it can send back a reset message (RST) instead of an acknowledge message (ACK).



Message Types CoAP

2. Non-confirmable Message {Unreliable Message} (Non):

- A Non-confirmable message doesn't require a response in the form of Acknowledgement.
- These messages do not contain critical information, like a request for a sensor measurement made in periodic basis.
- Even if one value is missed, there is not too much impact.
- Even if these messages are unreliable, they have a unique ID



Message Types CoAP

3. Acknowledgement (Ack):

- It is sent to Acknowledge a confirmable message (Con).

4. Reset (RST):

- It represents negative Acknowledgment.
- It generally indicates some kind of failure, like unable to process received data.



CoAP

- Four message types:
 - **Confirmable** – requires an ACK
 - **Non-confirmable** – no ACK needed
 - **Acknowledgement** – ACKs a Confirmable
 - **Reset** - indicates a Confirmable message has been received but context is missing for processing

1. CoAP Provides Reliability Without Using TCP as Transport Protocol:

This statement highlights a key advantage of CoAP: **lightweight reliability**. Unlike HTTP, which heavily relies on the reliable TCP protocol, CoAP uses UDP for communication. While UDP doesn't guarantee delivery or order of messages, CoAP implements its own mechanisms to achieve reliability without the overhead of TCP. This makes it ideal for resource-constrained devices with limited processing power and battery life.

Here's how CoAP ensures reliability:

- **Confirmable messages:** Clients request confirmation from the server for received messages.
- **Non-confirmable messages:** Used for best-effort delivery when confirmation is not crucial.
- **Re-transmission with backoff:** If a confirmation is not received, the message is re-sent with increasing delays to avoid overwhelming the network.

2. CoAP Enables Asynchronous Communication:

This refers to the ability of CoAP to handle requests even when the server is not immediately available.

Example:

- A sensor sends a temperature reading to the server using CoAP.
- If the server is busy processing other requests, it still acknowledges the message (ACK) to let the sensor know it was received.
- Once the server frees up, it can process the reading and send a response at its own pace.

This asynchronous approach improves efficiency and reduces latency, especially for devices with limited resources or intermittent connectivity.

devices with limited resources or intermittent connectivity.

3. Multicast and Congestion Control:

CoAP supports:

- **Multicast:** Sending a message to a group of devices simultaneously, ideal for applications like smart lighting control or emergency notifications.
- **Congestion control:** Techniques like backoff and random delay help regulate data flow and prevent network overload in scenarios with multiple devices communicating concurrently.

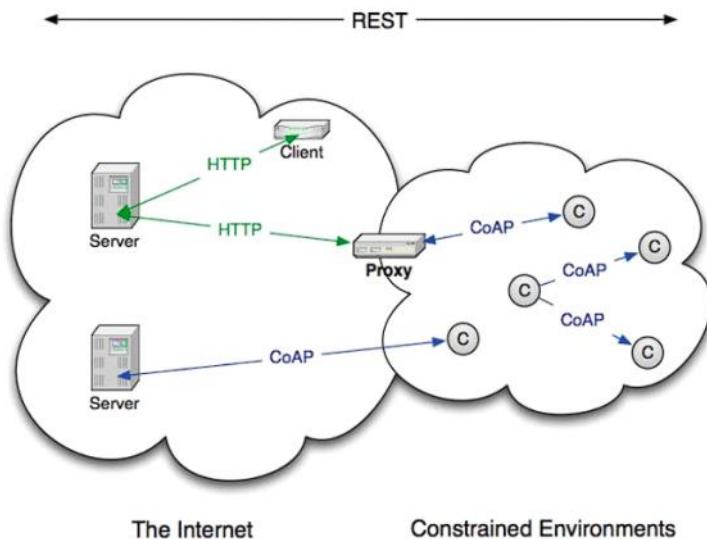
These features enhance CoAP's suitability for resource-constrained networks with potentially high device density.

In summary:

CoAP is a powerful protocol for resource-constrained devices, offering:

- **Lightweight reliability:** Achieves reliable communication without the overhead of TCP.
- **Asynchronous communication:** Enables efficient handling of requests even when the server is busy.
- **Multicast and congestion control:** Optimizes communication in scenarios with

CoAP



BASICALLY WITHIN CONSTRAINTS ENV USE :
COAP AND THEN : HTTP WHEN COAP SERVER SE
DATA INTERNET / CLOUD PE BHEJNA HO !! AND
THIS COULD HAPPENED ASYNCRONOUSLY !!

What CoAP Is

- CoAP is
 - A RESTful protocol
 - Both synchronous and asynchronous
 - For constrained devices and networks
 - Specialized for M2M applications
 - Easy to proxy to/from HTTP

Basics of MQTT

- MQTT – Message Queuing Telemetry Transport
- MQTT is used for M2M and IoT connectivity.
- It is a publish-subscribe-based messaging protocol that transports messages between devices.
- It is very lightweight and thus suited for M2M (Mobile to Mobile, WSN – Wireless Sensor Network) and IoT sensors, where communication nodes communicate with applications through the MQTT message broker.
- It usually runs over TCP/IP Protocol.
- MQTT was developed by IBM and Eurotech.
- MQTT is designed for limited devices, high latency, and low bandwidth communication.

Generative AI is experimental. Learn more : :



En



Listen

Telemetry is the process of collecting, transmitting, and measuring data from remote sources. The word comes from the Greek words tele, meaning "remote", and metron, meaning "measure".

Show more ▾

2.20

MQTT Terminologies

PUBLISH is mechanism used to SEND Messages

SUBSCRIBE is mechanism used to RECEIVE Messages

TOPIC is like address or name, e.g. for temp. sensor we assign topic as device/temp

PAYLOAD is an actual MESSAGE or DATA to be SENT from Client

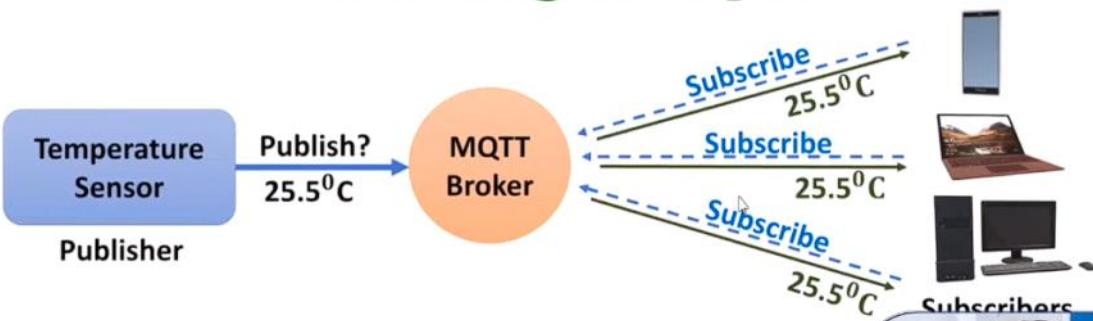
MQTT Components



- **MQTT Client** – A client can be either the publisher or the subscriber. Subscribers can subscribe to topics and receive messages. Publishers can publish the topics.
- **MQTT Broker** – It is a central point of communication. It is responsible for dispatching all messages between clients. It receives subscriptions from clients on topics and based on subscriptions it forwards the messages to clients.
- **Topic** – It is an identifier for subscribers. {Temperature, Pressure, Humidity, Gases Properties, etc.}
- **Message** – Subscribers receives messages based on topics of subscription from Broker.



Working of MQTT



- **Step – 1:** Publishers can publish topics. {Ex. Temperature}
- **Step – 2:** Subscribers can subscribe to one or more topics.
- **Step – 3:** Publisher gives messages on topics to MQTT Broker.
- **Step – 4:** MQTT Broker publishes the messages to subscribed users.



MQTT Methods

- **Connect** – The client requests a connection with the MQTT Broker.
- **Disconnect** – The client requests a disconnection with the MQTT Broker.
- **Subscribe** – The client subscribes to the topic.
- **Unsubscribe** – The client unsubscribes from the topic.
- **Publish** – The client publishes messages.



MQTT Advantages

- It is a simple protocol for IoT devices.
- It is a lightweight protocol, so it is easy to implement in software.
- Low Network usage.
- Low Power usage.
- Lower data rate for protocol implementation.

IT IS ALSO AN APPLICATION LAYER PROTOCOL LIKE THE COAP PROTOCOL !!

CoAP Vs MQTT

Parameters	CoAP	MQTT
Full Form	❖ Constrained Application Protocol	❖ Message Queuing Telemetry Transport
Architecture Model	❖ Request Response Model	❖ Public Subscribe Model
Messaging Mode	❖ Asynchronous and Synchronous	❖ Asynchronous only
Communication Mode	❖ One to One	❖ Many to Many
Transport Layer	❖ Mainly UDP	❖ Mainly TCP
Header Size	❖ 4 Byte	❖ 2 Byte
Message Labelling	❖ Yes (2byte ID with message)	❖ No
LNN Effectiveness	❖ Excellent	❖ Low
Sorting of Data	❖ Universal Resource Identifier (URI)	❖ Broker Topic
Port Used	❖ 61631	❖ Secure (8883) and Nonsecure (1883)
Persistence Support	❖ No Support	❖ It supports live data Communication
Reliability	❖ It takes care of confirmable messages, non-confirmable messages, acknowledgments, and retransmissions.	❖ Three quality services: 1. Delivery not Guaranteed 2. Delivery confirmation 3. Delivery double confirmation

Comparison CoAP & MQTT

Both used in IoT

- CoAP:

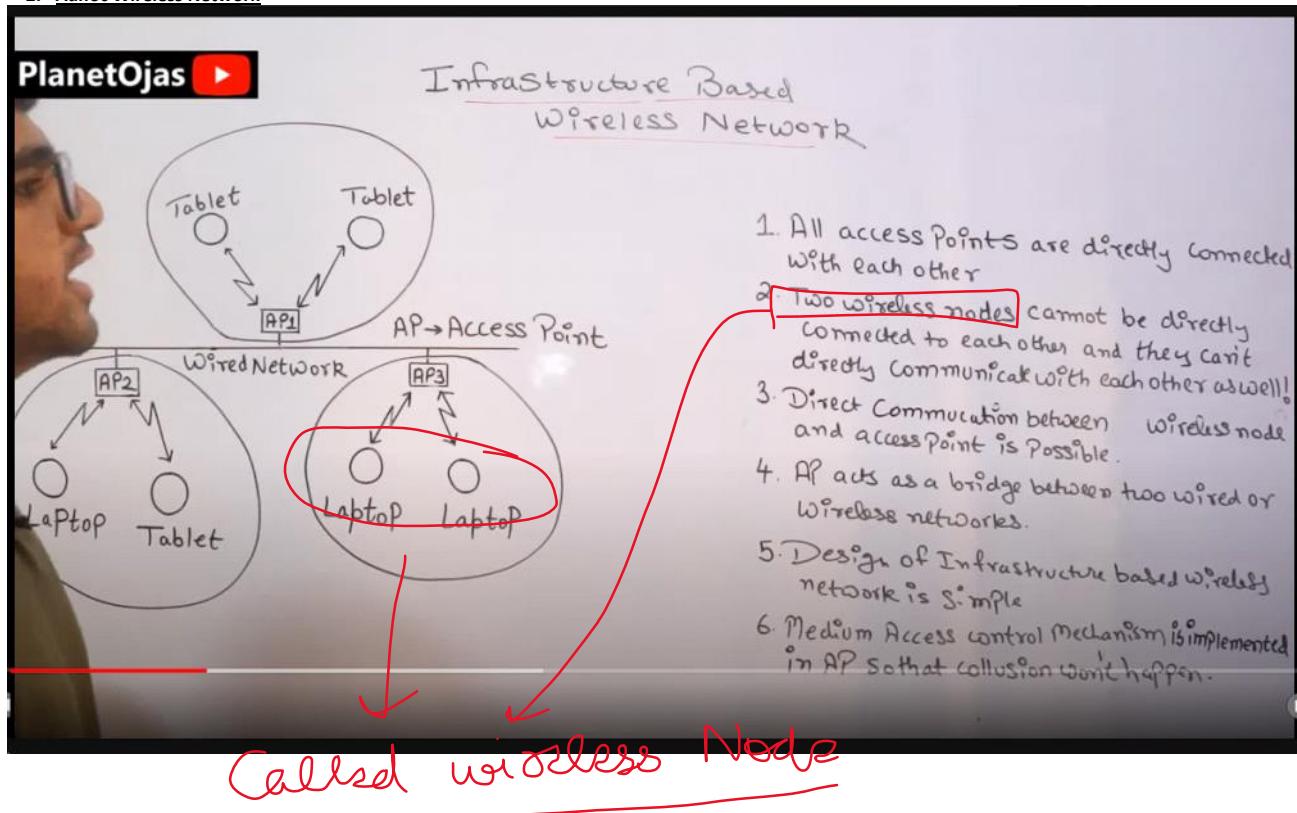
- one-to-one communication
- UDP/IP
- unreliable
- lightweight and easy to implement

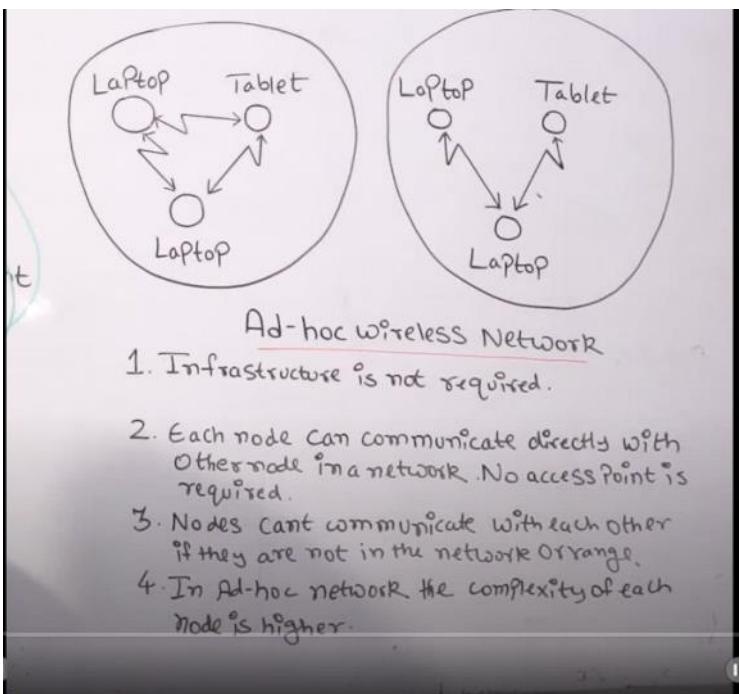
- MQTT:

- many-to-many communication
- TCP/IP
- focus on message delivery; reliable
- higher overheads (protocol data, processing costs)

WIRELESS LANs ARE OF 2 TYPES :

1. **Infrastructure Based Wireless Network**
2. **Adhoc Wireless Network**





ADHOC MEANS TEMPORARY !!

EG : BLUETOOTH { VVIMP }

MEDIUM ACCESS CONTROL MECHANISM AT EACH WIRELESS NODE { THEREFORE : COMPLEXITY OF EACH NODE INCREASES }

Same hi cheez hai below ss :

Easy Engineering Classes – Best YouTube Channel for University/College Semester Exams

IEEE 802.11

- ↳ Family of WLANs.
- It is a standard that specifies the Physical or MAC Layer adapted.
- Defines separate standard for infrastructure based and adhoc NW.

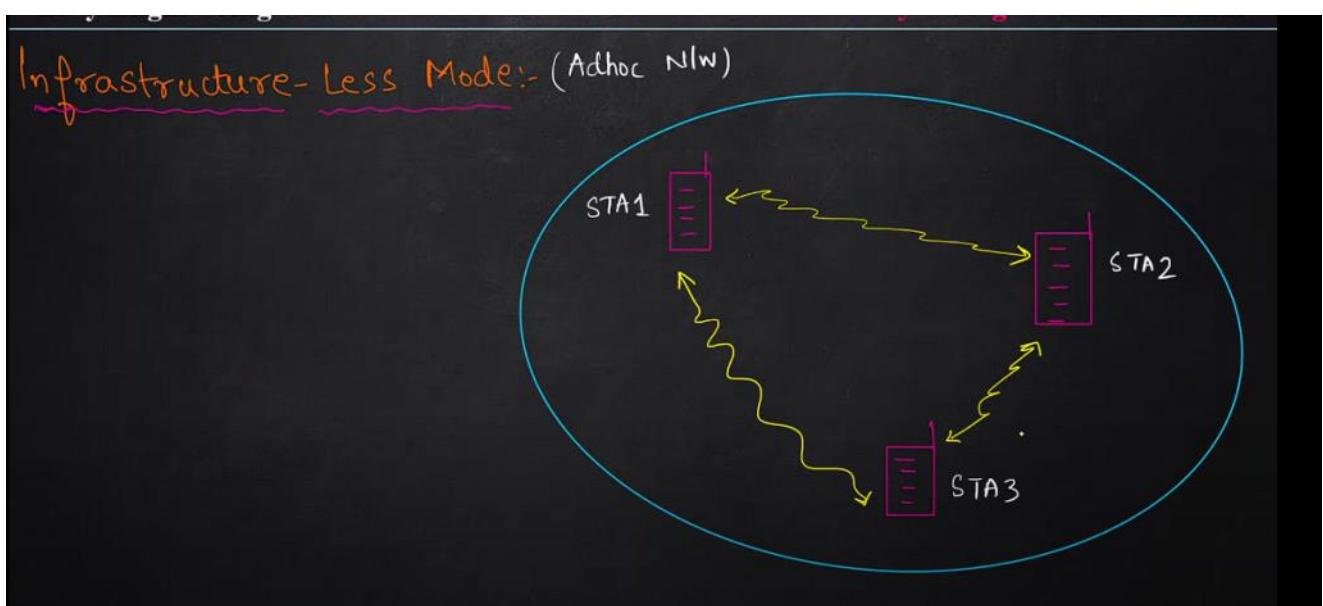
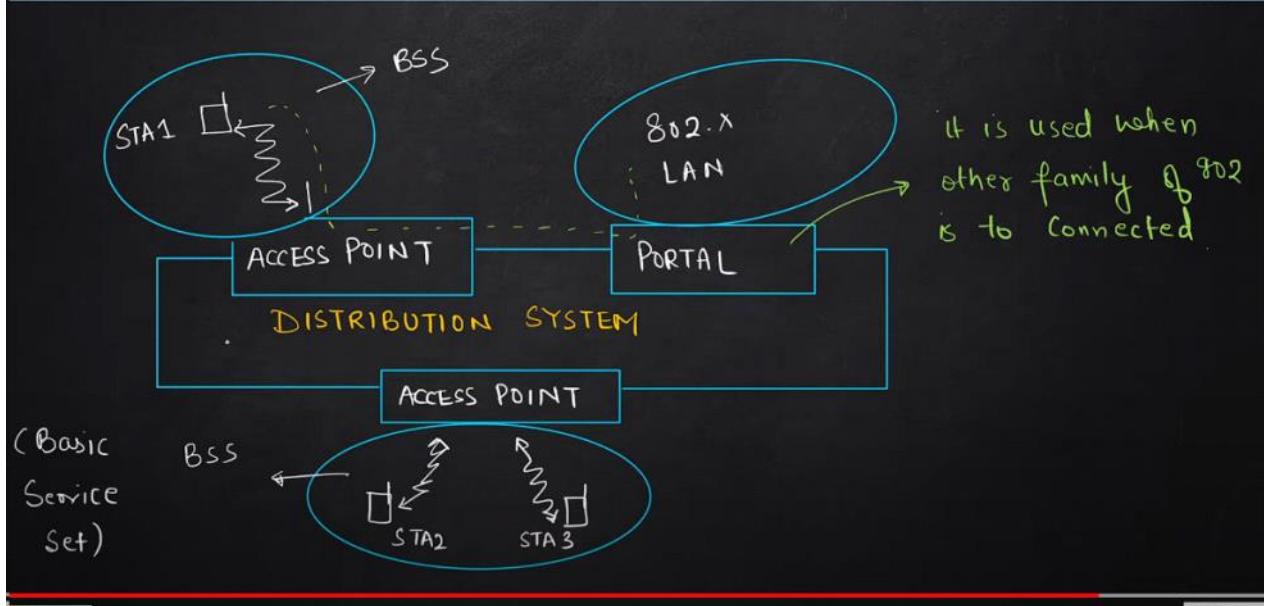
Infrastructure based Mode:-

- ↳ Helps in providing Wi-Fi Hot spots for internet access.
- Based on CSMA/CA.
- Multiple access point are connected to form a distribution NW.



Like and Share the Video – Don't Forget to Subscribe Channel and Press





L Infrastructure based and adhoc wireless networks are basically using the ieee 802.11 standard ? like what is the role of this standard / protocol in these networks ?

Show drafts ▾

You're absolutely correct! Both infrastructure-based and ad hoc wireless networks can leverage the IEEE 802.11 standard, often referred to as Wi-Fi. However, the **role of the standard varies slightly** depending on the network type:

Infrastructure-based networks:

- Centralized access points (APs): These manage all communication within the network, acting as a central hub.
- IEEE 802.11 defines the communication protocol between devices and APs.** This includes:
 - Media access control (MAC):** Regulates how devices share the wireless medium to avoid collisions.
 - Physical layer (PHY):** Defines the radio frequencies and modulation techniques used for data transmission.
 - Data link layer:** Handles packet framing, addressing, and error checking.
 - Security:** Provides encryption and authentication mechanisms to protect data.
- Specific variations of the standard are used:

In summary:

- IEEE 802.11 provides the foundation for communication in both infrastructure-based and ad hoc wireless networks.
- In infrastructure networks, it governs communication between devices and the central AP.
- In ad hoc networks, it applies to direct device-to-device communication and might require additional protocols for routing and discovery.

This standard ensures interoperability and consistency, allowing different devices to connect and communicate reliably within both network types.



Wi-Fi scanning, within the context of the 802.11 standard, refers to the process of a wireless device searching for available Wi-Fi networks. This scanning can be done in two main ways: **passive scanning** and **active scanning**.

Passive Scanning:

- In passive scanning, the device listens for **beacon frames** transmitted by nearby access points (APs).
- Beacon frames contain information about the network, such as its Service Set Identifier (SSID), supported data rates, security settings, and other details.
- The device doesn't transmit any frames itself, making it a **stealthy** way to detect available networks.
- This approach is often used by battery-powered devices to conserve energy, as it requires less radio activity.
- However, passive scanning takes longer because the device has to wait for APs to transmit beacon frames, which may occur at varying intervals.
- It also has limited range, as it can only detect APs that are within its reach and actively broadcasting beacons.

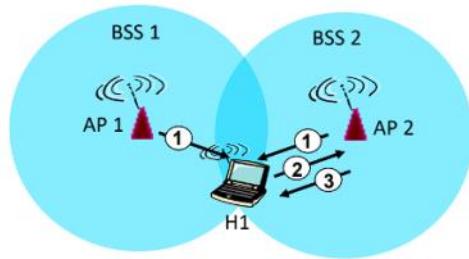
Active Scanning:

Active Scanning:

- In active scanning, the device transmits **probe request frames** specifically asking for information about nearby networks.
- This is a more direct and **aggressive** way to find networks, as the device actively seeks them out.
- As a result, active scanning usually returns results faster than passive scanning.
- However, it also consumes more battery power due to the additional transmissions.
- Additionally, some networks may ignore probe requests for security reasons.

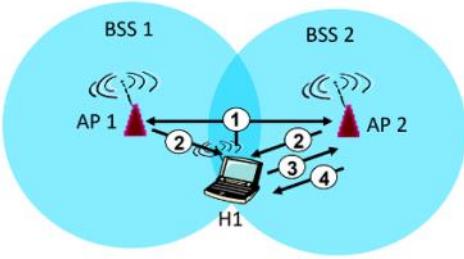
Choosing the Right Scanning Method

Wi-Fi (802.11) Scanning



Passive Scanning

- (1) Beacons sent from APs
- (2) Association Request sent from H1 to selected AP
- (3) Association Response sent from AP to H1



Active Scanning

- (1) Probe Request (broadcast) sent from H1
- (2) Probe Response sent from APs
- (3) Association Request sent from H1 to selected AP
- (4) Association Response sent from AP to H1

Easy Engineering Classes – Free YouTube Lectures
GGSIPU, UPTU, Mumbai Univ., Pune Univ., GTU, Anna Univ., PTU and Others EEC Classes

Reactive Routing Protocols:-

Are also known as On-demand routing protocols.

Dynamic Source Routing (DSR) Protocol:

- ↳ discovers the route b/w Source and destination when required. Sender knows the complete path.
- ↳ is based on Source Routing Path.
- ↳ Intermediate nodes do not maintain routing info to route the packets to the destination.
- ↳ Less NW overhead as the no. of message exchanges b/w nodes is very low.

Phases of DSR Protocol:

Route Discovery
↳ RREQ Packet
↳ (Route Req.)
↳ Source Node ID
↳ Destination Node ID
↳ (Broadcast)

Route maintenance
↳ RERR message (Route Error)
↳ Route Cache.

Route Cache.
Stores the Path.

Diagram illustrating the DSR protocol phases:

A network diagram showing nodes A, B, C, D, and E. A dashed line connects node A to node B, and another dashed line connects node B to node D. A solid line connects node A to node C, and another solid line connects node C to node E. A red arrow points from node A to node B, indicating the path being discovered. Labels include "Sender Path des." and "unicast". The path A → B → D is highlighted in green.

Example of Dynamic Source Routing (DSR):

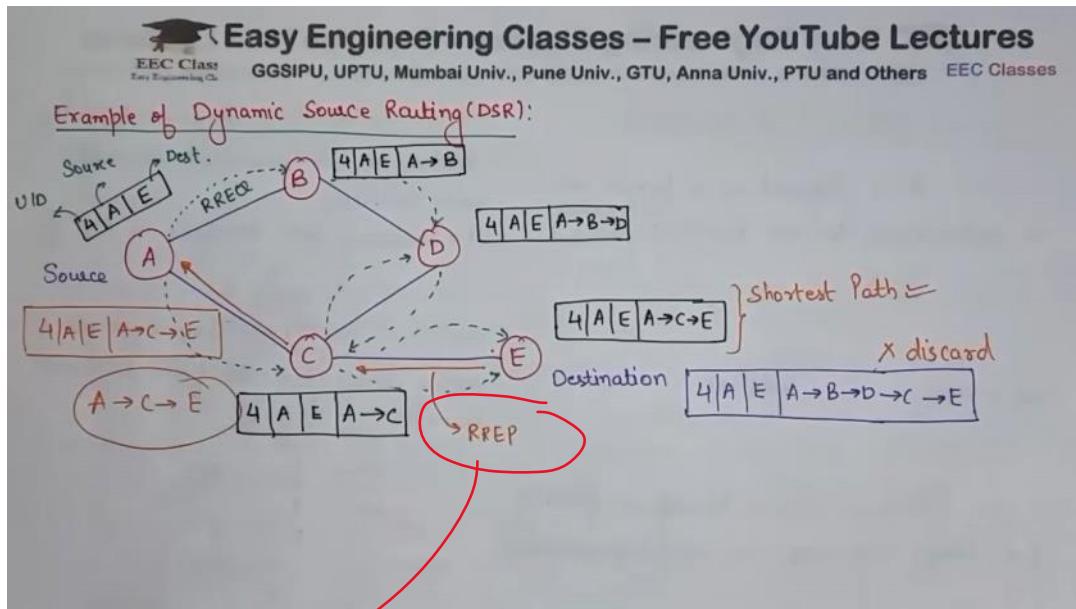
The diagram shows a network with nodes A, B, C, D, and E. A dashed line connects node A to node B, and another dashed line connects node B to node D. A solid line connects node A to node C, and another solid line connects node C to node E. A red arrow points from node A to node B, indicating the path being discovered. Labels include "Source" and "Dest.". The path A → B → D is highlighted in green.

Shortest Path =
x discard

Destination: 4|A|E|A → B → D → C → E

Other paths shown:
 - 4|A|E|A → C → E
 - 4|A|E|A → C → B → D → E

The above ss is of rreq packets { broadcast }



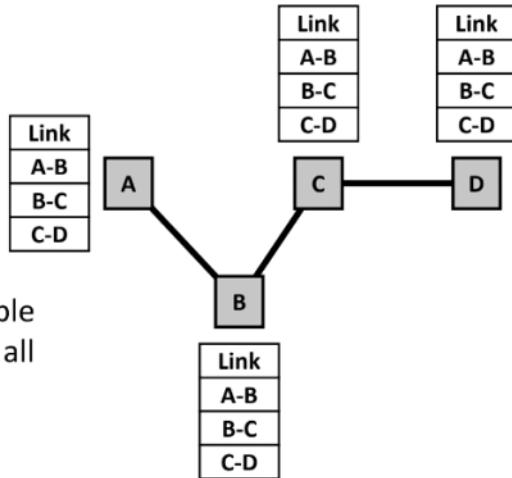
THIS IS RREP { ROUTE REPLY } { UNICAST }
WHERE THE DESTINATION WILL SEND THE
OPTIMAL / SHORTEST PATH TO THE SENDER !!

Ad-Hoc Routing Protocol

- An ad-hoc routing protocol is a convention that controls how nodes decide which way to route packets between computing devices in a mobile ad-hoc network
- Foundation in most protocols: **neighbor discovery**
 - Nodes send periodic announcements as broadcast packets (beacon messages, alive messages, ...)
 - Can embed “neighbor table” into such messages; allows nodes to learn “2-hop neighborhood”
- Popular types of routing protocols:
 - Proactive
 - Reactive
 - Geographic

Proactive: “Link-State” Algorithms

- Each node shares its link information so that all nodes can build a map of the full network topology



- Assuming the topology is stable for a sufficiently long period, all nodes will have the same topology information

Path Calculation:

- Each node **individually calculates the shortest path** to all other nodes based on its local routing table and a chosen algorithm (e.g., Dijkstra's algorithm).
- This calculation happens **locally** on each node, not on the sender beforehand.
- The calculated path doesn't include the entire route from source to destination; it only lists the **next hop** towards the destination based on the current network information.

Packet Forwarding:

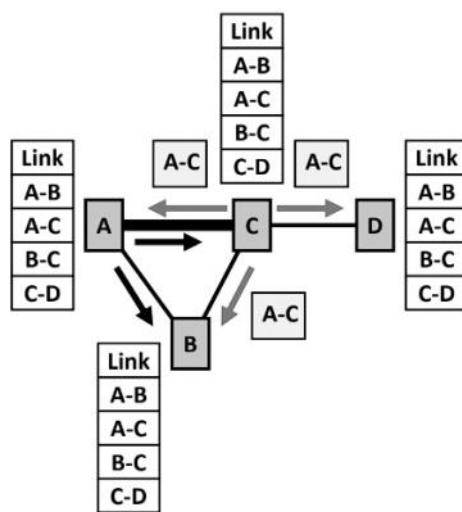
- When a node wants to send a packet, it checks its routing table for the next hop towards the destination based on the calculated path.
- The packet only includes the **address of the next hop**, not the entire route.
- This **hop-by-hop forwarding** approach adapts to network changes efficiently.

Proactive: “Link-State” Algorithms

- Link information is updated when a link changes state (goes up or down)

- by sending small “hello” packets to neighbors

- Nodes A and C propagate the existence of link A-C to their neighbors and, eventually, to the entire network



Reactive: DSR

- **Dynamic Source Routing**
- Search for route when needed only
 - Search using **Route Request (RREQ)** broadcasts
 - Response using **Route Reply (RREP)** message
- Every message along route contains entire path to help intermediate nodes to decide what to do with message

Proactive vs Reactive

- **Reactive:**
 - Only establish/maintain routes between nodes needed them (in contrast: tables store ALL routes)
 - Store entire route in each message; message size grows with route length
 - Route requests cause “flooding”
- **Proactive:**
 - Route information always available; no need to search for route (but route information can be outdated)
 - Continuous exchange of route change updates

1. Geographic Routing:

- Geographic routing is a routing technique used in wireless ad hoc networks, where nodes utilize location information to make routing decisions. Instead of using traditional routing protocols that rely on network topology information, geographic routing uses the geographical positions of nodes to determine the next hop for forwarding packets.
- In geographic routing, nodes must know their own locations, the location of the destination, and the locations of their neighboring nodes. This information can be obtained through GPS (Global Positioning System) or other positioning technologies.
- The sender can make routing decisions based on the location information of itself, the destination, and its neighbors. Alternatively, a location broker may be used to query and obtain location information.

2. Unicast Location-Based Routing:

- Unicast location-based routing is a type of geographic routing where a packet is forwarded to a single destination node.
- In unicast location-based routing, each forwarding node makes routing decisions based on the location of the destination node and the locations of its neighboring nodes. This is often done using a greedy forwarding strategy, where nodes forward packets to the neighbor closest to the destination.
- A challenge in unicast location-based routing is the presence of voids or holes in the network, where a packet may arrive at a node without neighbors that could bring the packet closer to the destination. Strategies such as using backup routes or local repair mechanisms may be employed to overcome this challenge.

3. Geocasting:

- Geocasting is a communication paradigm where a packet is sent to all or some nodes within a specific geographic region.
- In geocasting, the sender specifies a target geographic area, and the packet is forwarded to all nodes within that area or to a subset of nodes based on certain criteria.
- An example of geocasting is sending a query to all sensors within a particular geographic area of interest.
- Routing challenges in geocasting include propagating the packet near the target region (similar to unicast routing) and distributing the packet within the target region (similar to flooding). Various algorithms and protocols have been proposed to address these challenges, such as zone-based routing and area-based flooding.

In summary, geographic routing, unicast location-based routing, and geocasting are routing techniques used in wireless ad hoc networks that leverage location information to make routing decisions. These techniques enable efficient communication in dynamic and resource-constrained environments by exploiting the spatial properties of the network.

