IOT-UNIT-3 NOTES

Syllabus :

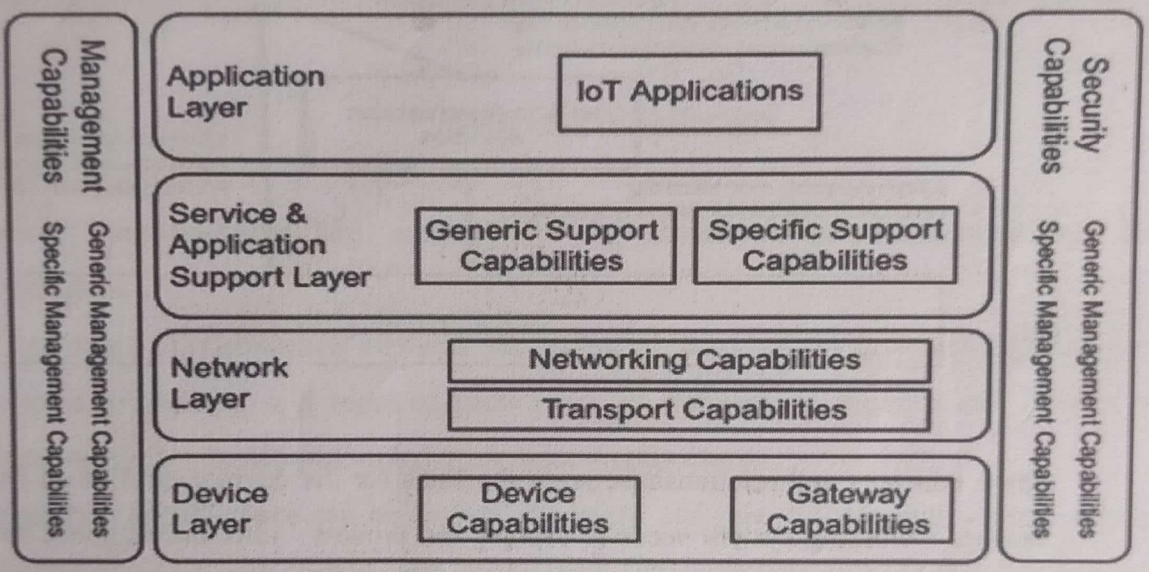
IoT Architecture and Protocols

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

3.1. Architecture Reference Model- Introduction :

This concept provides an overview of the Architecture Reference Model (ARM) for IoT, including descriptions of the domain, information, and functional models.

Architecture Reference Model



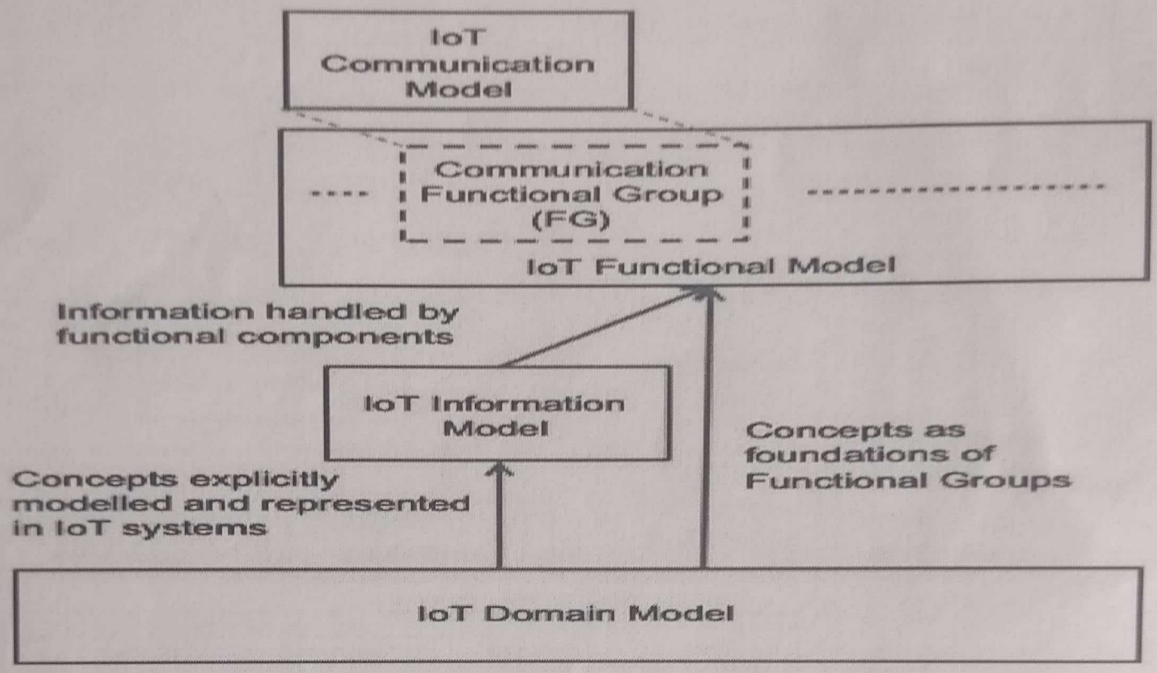
3.2. Reference Model and Architecture :

- An ARM consists of two main parts:
  - → Reference model
  - → Reference Architecture.
- For describing an IoT ARM, we have chosen to use the IoT-A ARM (Carrez et al. 2013) as a guide because it is currently the most complete model and reference.
- However, a real system may not have all the modeled entities or architectural elements described , or it could contain other non-IoT-related entities.
- The foundation of an IoT Reference Architecture description is an IoT reference model.
- A reference model describes the domain using a number of sub-models .

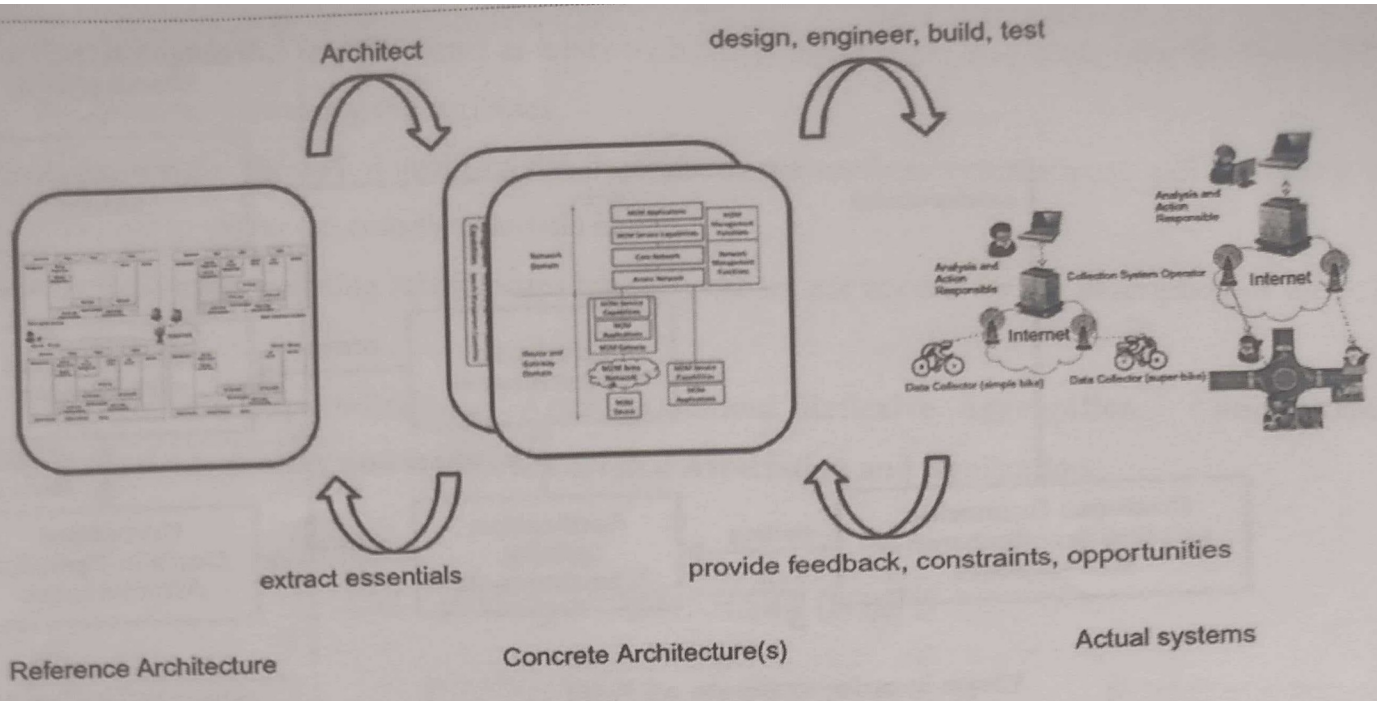
- The **domain model** of an architecture model captures the main concepts or entities in the domain in question, in this case M2M and IoT.

  When these common language references are established, the domain model adds descriptions about the relationship between the concepts.

IoT Reference Model



- These concepts and relationships serve the basis for the development of an **information model** because a working system needs to capture and process information about its main entities and their interactions.
- A working system that captures and operates on the domain and information model contains concepts and entities of its own, and these need to be described in a separate model, the **functional model**.
- Apart from the reference model, the other main component of an ARM is the **Reference Architecture**.
- A **System Architecture** is a communication tool for different stakeholders of the system. Developers, component and system managers, partners, suppliers, and customers have different views of a single system based on their requirements and their specific interactions with the system.
- The high-level abstraction is called **Reference Architecture** as it serves as a reference for rete architectures and actual systems, as shown in the Figure.

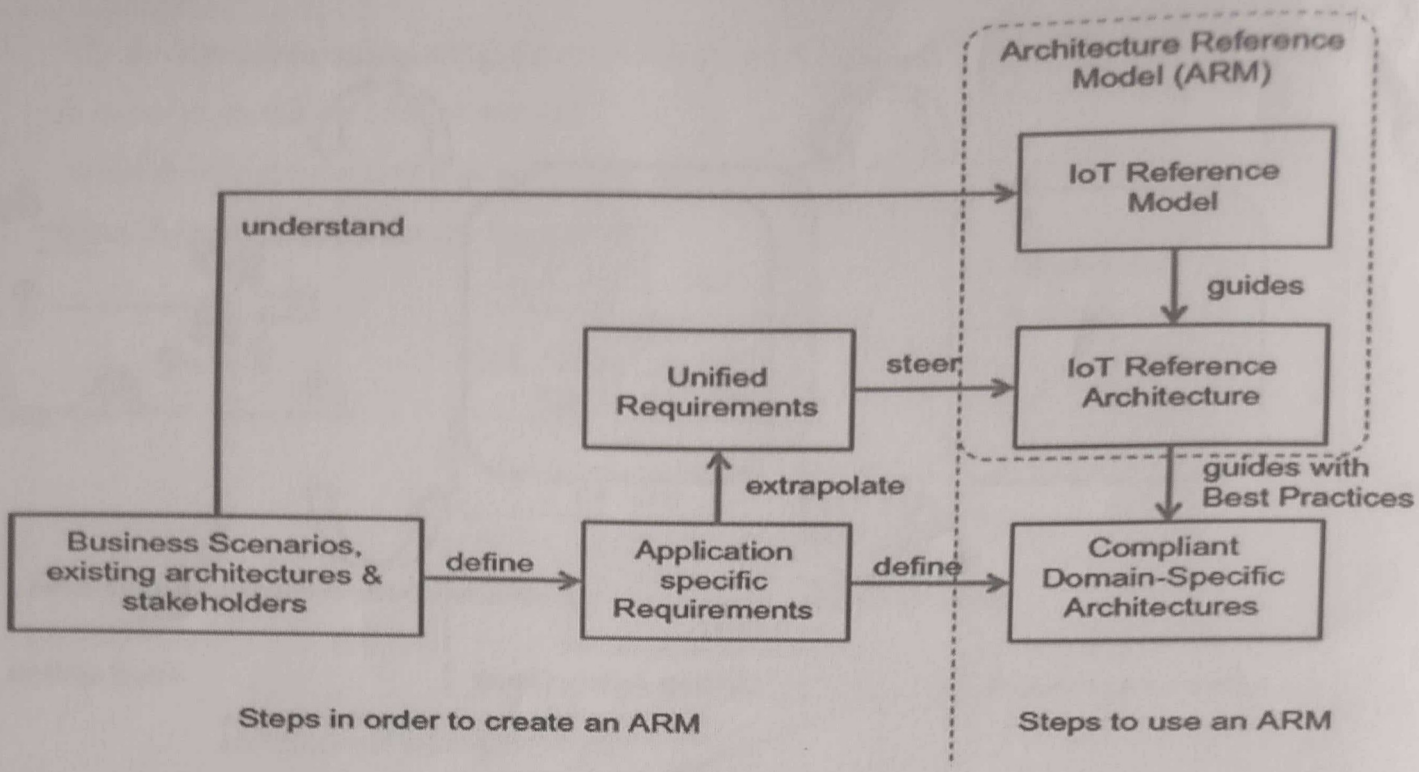


Reference Architecture

- **Concrete architectures** are instantiations of rather abstract and high-level Reference Architectures.
- A **Reference Architecture** captures the essential parts of an architecture, such as design principles, guidelines, and required parts (such as entities), to monitor and interact with the physical world for the case of an IoT Reference Architecture.
- A **concrete architecture** can be further elaborated and mapped into real world components by designing, building, engineering, and testing the different components of the actual system. As the figure implies, the whole process is iterative.
- The general essentials out of multiple concrete architectures can then be aggregated, and contribute to the evolution of the Reference Architecture.
- The IoT architecture model is related to the IoT Reference Architecture as shown in Figure 7.3.
- This figure shows two facets of the IoT ARM:

  (a) how to actually create an IoT ARM, and

  (b) how to use it with respect to building actual systems.
- The IoT reference model guides the process of creating an IoT Reference Architecture.



IoT Reference Architecture

3.3. IoT Reference Model :

1. IoT Domain Model :

- The domain model captures the basic attributes of the main concepts and the relationship between these concepts.
- A domain model also serves as a tool for human communication between people working in the domain and between people who work across different domains.

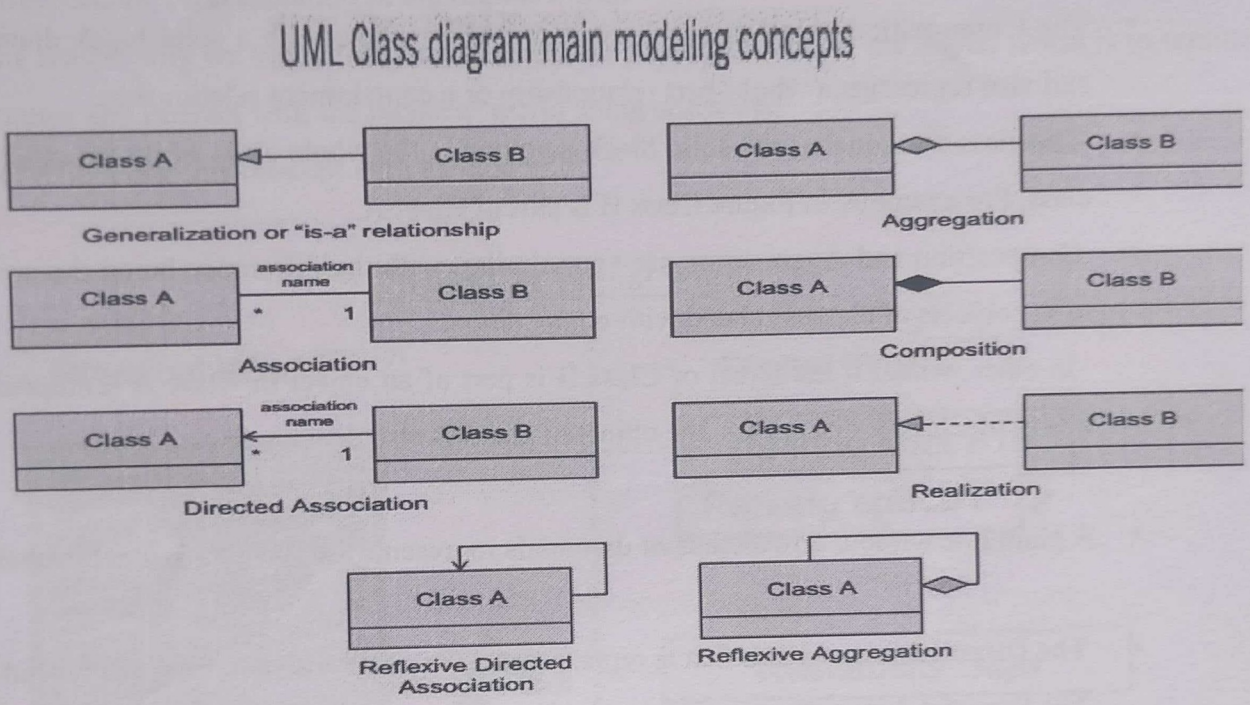
➢ Model notation and semantics :

- In order to present the relationships between the main concepts of the IoT domain model we use the **Unified Modeling Language (UML) Class diagrams .**

The Class diagrams consist of boxes that represent the different classes of the model connected with each other through typically continuous lines or arrows, which represent relationships between the respective classes.

A class contains a name and a set of attributes and operations. For the description of the IoT domain model, we will use only the class name and the class attributes, and omit the class

- Notation-wise this is represented as a box with two compartments, one containing the class name and the other containing the attributes.
- However, for the IoT domain model description, the attribute compartment will be empty in order not to clutter the complete domain model.
- The following modeling relationships between classes are needed for the description of the IoT Domain Model:
- **Generalization/Specialization, Aggregation and Reflexive Aggregation, Composition, Directed Association and Reflexive Directed Association and Realization.**



- The **Generalization/Specialization** relationship is represented by an arrow with a solid line and a hollow triangle head. Depending on the starting point of the arrow, the relationship can be viewed as a generalization or specialization.
- For example, in Figure Class A is a general case of Class B or Class B is special case or specialization of Class A. Generalization is also called an "is-a" relationship. For example, in Figure Class B "is-a" Class A.

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

- The class that touches the hollow diamond is the whole class while the other class is the part class.
- For example, in Figure 7.4, class B represents a part of the whole Class A, or in other words, an object of Class A "contains" or "has-a" object of Class B.
-
- When the line with the hollow diamond starts and ends in the same class, then this relationship of one class to itself is called **Reflexive Aggregation** and it denotes that objects of a class A contain objects of the same class.
-
- The **Composition relationship** is represented by a line with a solid black diamond in one end, and also represents a whole-part relationship or a containment relationship.
- The class that touches the solid black diamond is the whole class while the other class is the part class. For example, in Figure Class B is part of Class A.
- Composition and Aggregation are very similar, with the difference being the coincident lifetime to the objects of classes related with composition.

  In other words, if an object of Class B is part of an object of Class A (composition), when the object of Class A disappears, the object of Class B also disappears.

- A plain line without arrowheads or diamonds represents the **Association relationship.**

- The **Directed Association** that is represented with a line with a normal arrowhead.

  The Directed Association implies navigability from a Class B to a Class A in Figure.

  Navigability means that objects of Class B have the necessary attributes to know that they relate to objects of Class A while the reverse is not true: objects of Class A can exist without having references to objects of Class B.
- When the arrow starts and ends at the same class, then the class is associated to itself with a **Reflexive Directed Association**, which means that an object of this class is associated with objects of the same class with the specific named association.

An arrow with a hollow triangle head and a dashed line represents the **Realization relationship**. This relationship represents a association between the class that specifies the functionality and the class that realizes the functionality. For example, Class A in Figure specifies the functionality while Class B realizes it.

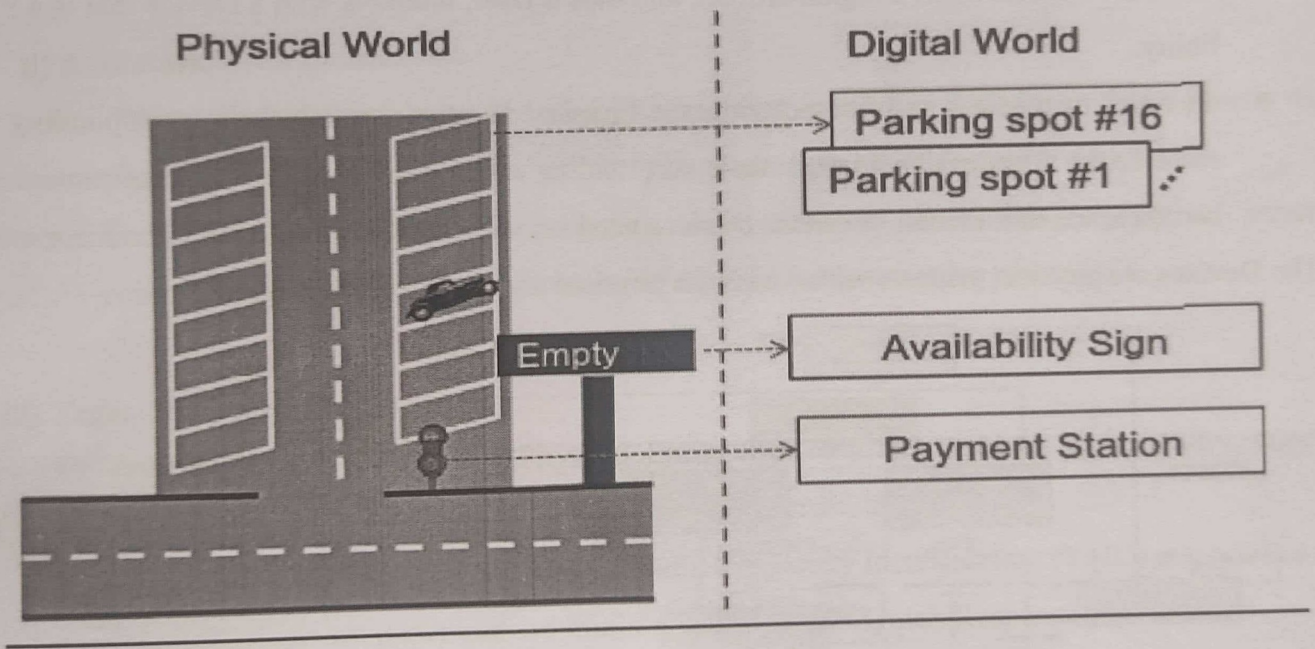
- Aggregations, Reflexive Aggregations, Associations (Directed or not) and Reflexive Associations (Directed or not) may contain multiplicity information such as numbers (e.g "1"), ranges (e.g. "0-1", open ranges "1. . ."), etc. in one or the other end of the relationship line/arrow.
- These multiplicities denote the potential number of class objects that are related to the other class object.

➢ Main concepts :

- The IoT is a support infrastructure for enabling objects and places in the physical world to have a corresponding representation in the digital world.
- The reason why we would like to represent the physical world in the digital world is to remotely monitor and interact with the physical world using software.

Let's illustrate this concept with an example

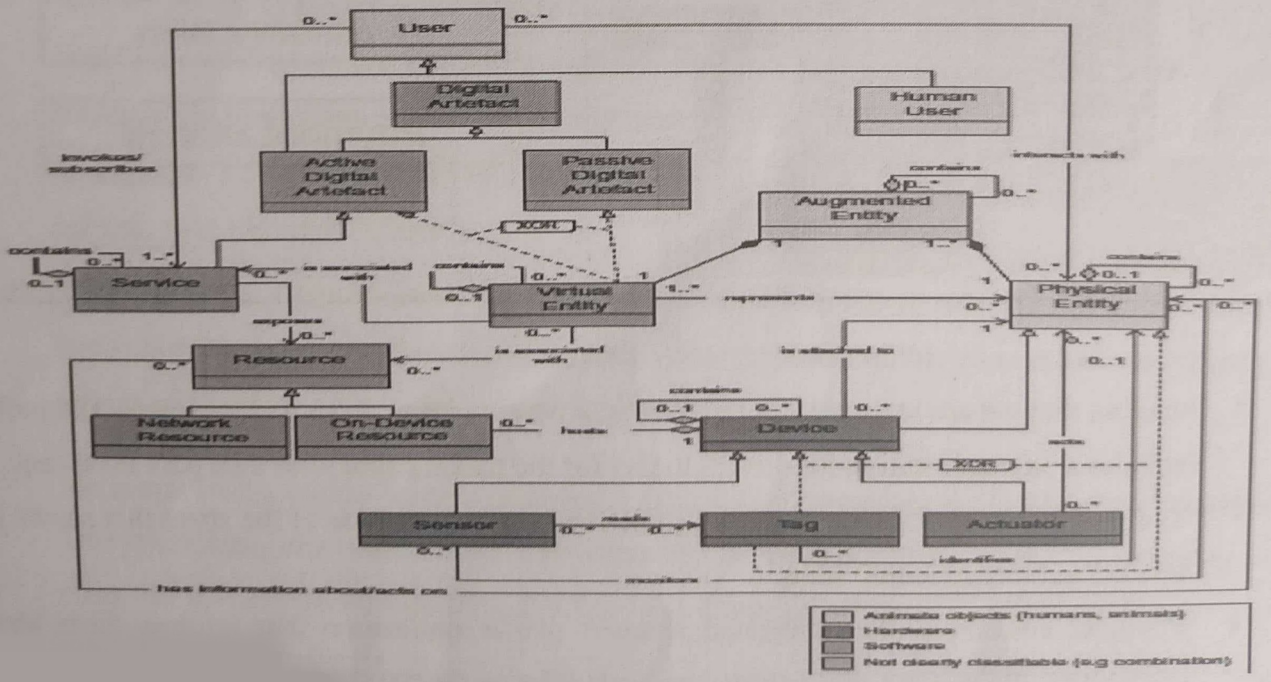
Physical vs Virtual World



- Imagine that we are interested in monitoring a parking lot with 16 parking spots. The parking lot includes a payment station for drivers to pay for the parking spot after they park their cars.
- The parking lot also includes an electronic road sign on the side of the street that shows in real-time the number of empty spots.
- Frequent customers also download a smart phone application that informs them about the [redacted] f a parking spot before they even drive on the street where the parking lot is located.

- A User and a Physical Entity are two concepts that belong to the domain model. A User can be a Human User, and the interaction can be physical (e.g. parking the car in the parking lot).
- The physical interaction is the result of the intention of the human to achieve a certain goal (e.g. park the car).
- A Physical Entity is represented in the digital world as a Virtual Entity.
- A Virtual Entity can be a database entry, a geographical model (mainly for places), an image or avatar, or any other Digital Artifact.
- One Physical Entity can be represented by multiple Virtual Entities, each serving a different Purpose.
- Each Virtual Entity also has a unique identifier for making it addressable among other Digital Artifacts.
- A Digital Artifact is an artifact of the digital world, and can be passive (e.g. a database entry) or active (e.g. application software).
- The model captures human-to-machine, application (active digital artifact)- to-machine, and M2M interaction when a digital artifact, and thus a User, interacts with a Device that is a Physical Entity.
- In order to monitor and interact with the Physical Entities through their corresponding virtual entities, the Physical Entities or their surrounding environment needs to be instrumented with certain kinds of Devices, or certain Devices need to be embedded/attached to the environment.

The Devices are physical artifacts with which the physical and virtual worlds interact.



- For the IoT Domain Model, three kinds of Device types are the most important:
  - → Sensors
  - → Actuators
  - → Tags

i) Sensors:

These are simple or complex Devices that typically involve a transducer that converts physical properties such as temperature into electrical signals.

- These Devices include the necessary conversion of analog electrical signals into digital signals, e.g. a voltage level to a 16-bit number, processing for simple calculations, potential storage for intermediate results, and potentially communication capabilities to transmit the digital representation of the physical property as well receive commands.
- A video camera can be another example of a complex sensor that could detect and recognise people.

ii) Actuators:

- These are also simple or complex Devices that involve a transducer that converts electrical signals to a change in a physical property (e.g. turn on a switch or move a motor).
- These Devices also include potential communication capabilities, storage of intermediate commands, processing, and conversion of digital signals to analog electrical signals.

iii) Tags:

- Tags in general identify the Physical Entity that they are attached to. In reality, tags can be Devices or Physical Entities but not both, as the domain model shows.
- An example of a Tag as a Device is a Radio Frequency Identification (RFID) tag, while a tag as a Physical Entity is a paper-printed immutable barcode or Quick Response (QR) code.
- Either electronic Devices or a paper-printed entity tag contains a unique identification that can be read by optical means (bar codes or QR codes) or radio signals (RFID tags).
- The reader Device operating on a tag is typically a sensor, and sometimes a sensor and an actuator combined in the case of writable RFID tags.

- IoT Services can be classified into three main classes according to their level of abstraction:
  → Resource-Level Services.
  → Virtual Entity-Level Services
  →Integrated Services

i) Resource-Level Services :

- Resource-Level Services typically expose the functionality of a Device by exposing the on-Device Resources.
- In addition, these services typically handle quality aspects such as security, availability, and performance issues.
- Apart from the on-Device resources, there are also Network Resources hosted in more powerful machines or in the cloud that exposes Resource-Level Services and abstracts the location of the actual resources.
- An example of such a Network Resource is a historical database of measurements of a specific resource on a specific Device.
- Resource-Level Services typically include interfaces that access resource information based on the identity of the resource itself.

ii) Virtual Entity-Level Services :

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

iii) Integrated Services

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

  An example of an instantiation of the IoT Domain Model is shown in Figure . For this instantiation, we use the example of a simple parking lot management system presented earlier, and we model only part of the real system.

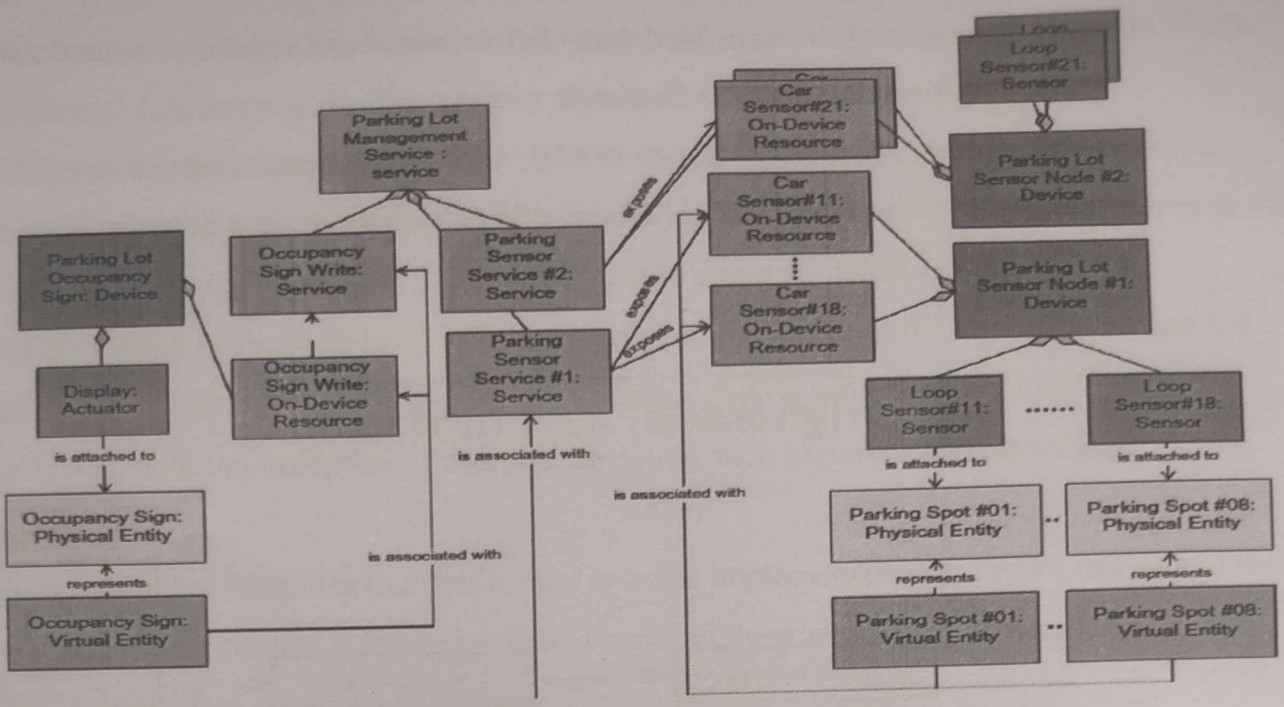


FIGURE 7.7

IoT Domain Model instantiation.

➢ **Further considerations :**

- Identification of Physical Entities is important in an IoT system in order for any User to interact with the physical world though the digital world.
- Furness et al. (2009) describe two ways:

  (a) primary identification that uses natural features of a Physical Entity,

  (b) secondary identification when using tags or labels attached to the Physical Entity.
- Both types of identification are modeled in the IoT Domain Model. Extracting natural features can be performed by a camera Device (Sensor) and relevant Resources that produce a set of features for specific Physical Entities.

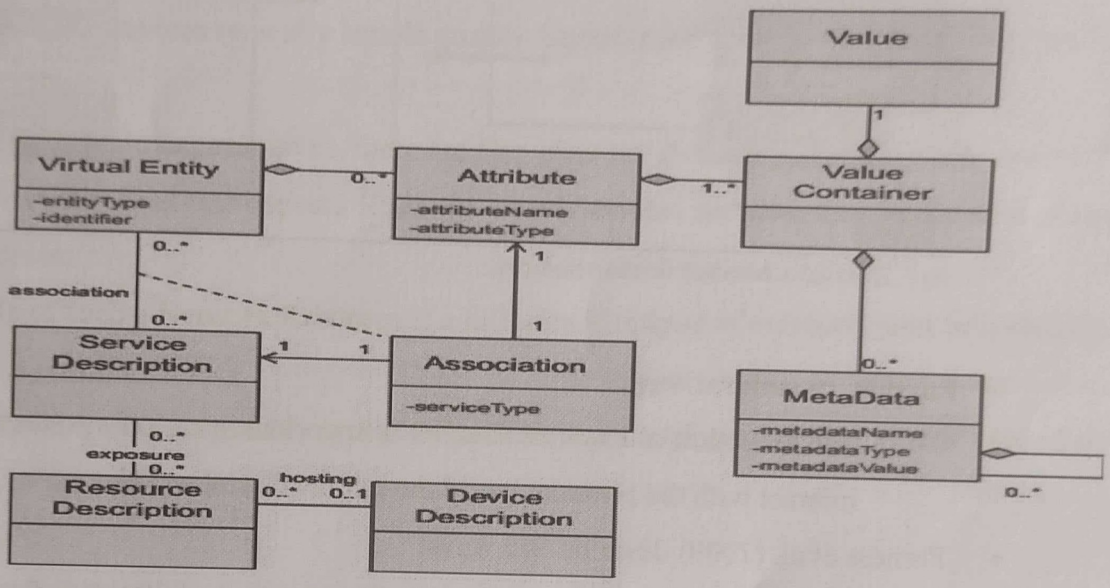
2. Information model :

- According to the Data-Information-Knowledge-Wisdom Pyramid (Rowley 2007), information is defined as the enrichment of data (raw values without relevant or usable context) with the right context, so that queries about who, what, where, and when can be answered.

Because the Virtual Entity in the IoT Domain Model is the "Thing" in the Internet of Things, the [redacted] captures the details of a Virtual Entitycentric model.

- Similar to the IoT Domain Model, the IoT Information Model is presented using Unified Modeling Language (UML) diagrams.
- As mentioned earlier, each class in a UML diagram contains zero or more attributes. These attributes are typically of simple types such as integers or text strings, and are represented with red text under the name of the class.

High-level IoT Information Model.



- A more complex attribute for a specific class A is represented as a class B, which is contained in class A with an aggregation relationship between class A and class B.
- Moreover, the UML diagram for describing the IoT Information Model contains additional notation not presented earlier.
- More specifically, the Association class contains information about the specific association between a Virtual Entity and a related Service.

In other words, while in the IoT Domain Model, one is interested in capturing the fact that a Virtual Entity and a Service are associated, and the IoT Information Model explicitly represents this association as part of the information maintained by an IoT system.

- On a high-level, the IoT Information Model maintains the necessary information about Virtual Entities and their properties or attributes.

- These properties/attributes can be static or dynamic and enter into the system in various forms, e.g. by manual data entry or reading a sensor attached to the Virtual Entity.
- Virtual Entity attributes can also be digital synchronized copies of the state of an actuator as mentioned earlier: by updating the value of an Virtual Entity attribute, an action takes place in the physical world.

  In the presentation of the high-level IoT information model, we omit the attributes that are not updated by an IoT Device (sensor, tag) or the attributes that do not affect any IoT Device (actuator, tag), with the exception of essential attributes such as names and identifiers.

- Examples of omitted attributes that could exist in a real implementation are room names and floor numbers, in general, context information that is not directly related to IoT Devices, but that is nevertheless important for an actual system.
- The IoT Information Model describes Virtual Entities and their attributes that have one or more values annotated with meta-information or metadata.

  The attribute values are updated as a result of the associated services to a Virtual Entity. The associated services, in turn, are related to Resources and Devices as seen from the IoT Domain Model.

- The IoT Information Model captures the above associations as follows . A Virtual Entity object contains simple attributes/properties:

  (a) EntityType to denote the type of entity, such as a human, car, or room .

  (b) a unique identifier

  (c) zero or more complex attributes of the class Attributes.

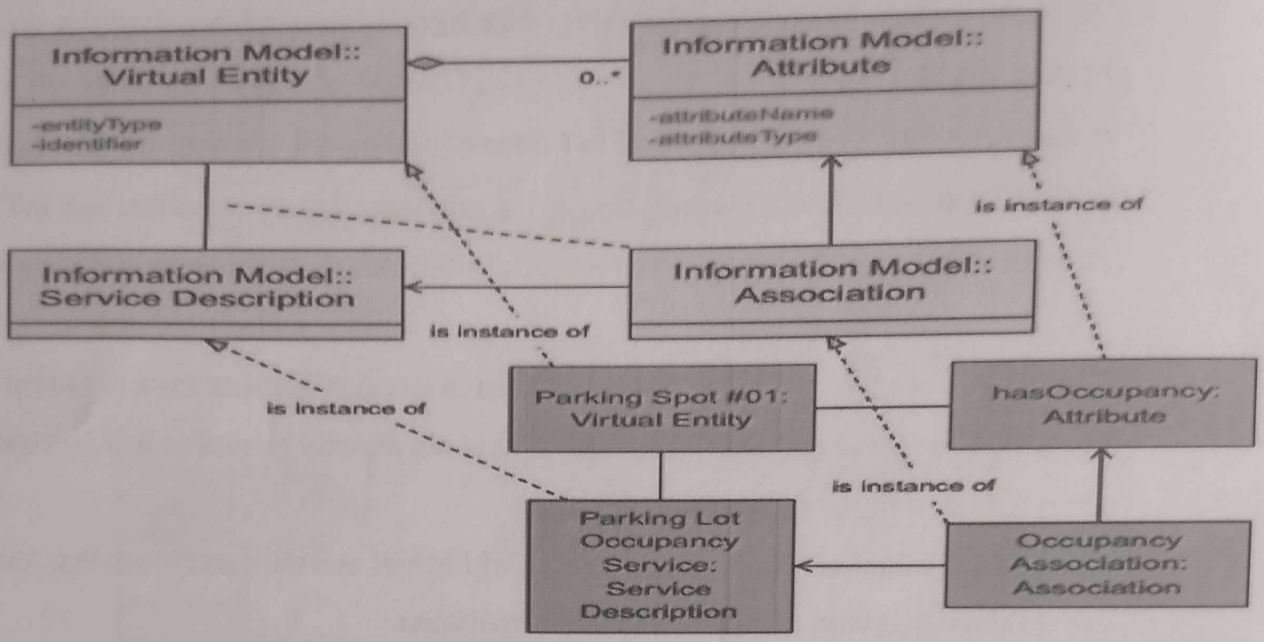
  Because the class Association describes the relationship between a Virtual Entity and Service Description through the Attribute class, there is a dashed line between Association class and the line between the Virtual Entity and Service Description classes.
- The attribute service Type can take two values:

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

  (b) "ACTUATION," if the associated service is an actuation service (i.e. allows an action executed on an actuator).

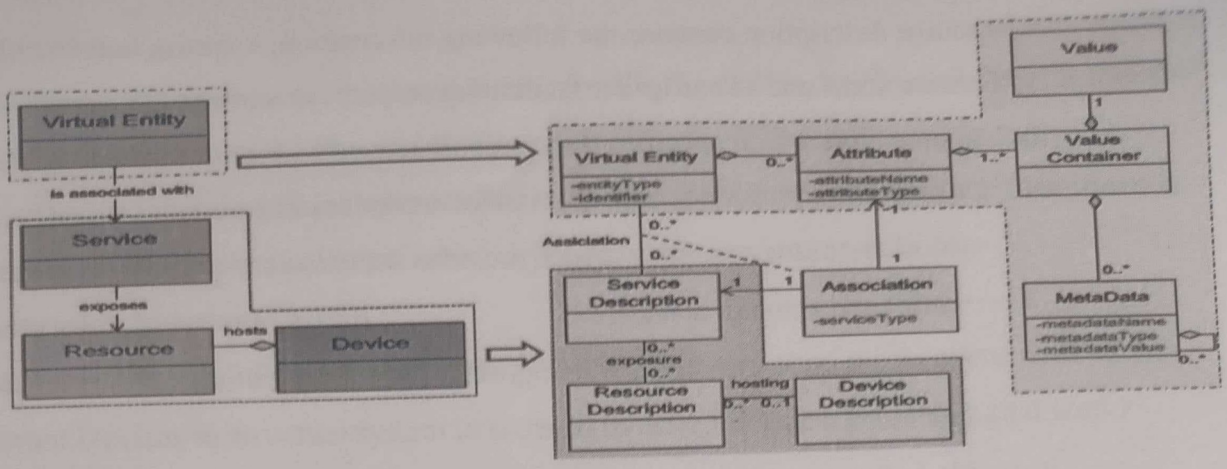
  In both cases, the eventual value of the attribute will be a result of either reading a sensor or [redacted]tor.

An example of an instantiation of the high-level information model following the parking lot example presented earlier.



- Throughout the description of the IoT Information Model, the reader might wonder about the mapping between the IoT Domain Model and the Information Model.
- It presents the relationship between the core concepts of the IoT Domain Model and the IoT Information Model.
- The Information Model captures the Virtual Entity in the Domain Model being the "Thing" in the Internet of Things as several associated classes that basically capture the description of a Virtual Entity and its context.

  The Device, Resource, and Service in the IoT Domain Model are also captured in the IoT Information Model because they are used as representations of the instruments and the digital interfaces for interaction with the Physical Entity associated with the Virtual Entity.



Relationship between core concepts of IoT Domain Model and IoT Information Model.

- The Virtual Entity in the IoT Information Model is described with only a few simple attributes, the complex Attribute associated with sensor/actuator/tag services.
- As mentioned earlier, there are several other attributes or properties that could exist in a Virtual Entity description:

(i) Location and its temporal information are important because Physical Entities represented by Virtual Entities exist in space and time.

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

(iii) Information such as ownership is also important in commercial settings because it may determine access control rules or liability issues.

- The Services in the IoT Domain Model are mapped to the Service Description in the IoT Information Model.
- The Service Description contains (among other information) the following :

(i) Service type, which denotes the type of service, such as Big Web Service or RESTful Web Service.

(ii) Service area and Service schedule are properties of Services used for specifying the geographical area of interest for a Service and the potential temporal availability of a Service, respectively.

(iii) Associated resources that the Service exposes.

(iv) Metadata or semantic information used mainly for service composition.

- The IoT Information Model also contains Resource descriptions because Resources are associated [redacted]s and Devices in the IoT Domain model.

- A Resource description contains the following information:

(i) Resource name and identifier for facilitating resource discovery.

(ii) Resource type, which specifies if the resource is

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

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

(c) **a processor resource**, which provides processing of sensor data and output of processed data;

(d) **a storage resource**, which provides storage of data about a Physical Entity;

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

(iii) Free text attributes or tags used for capturing typical manual input such as "fire alarm, ceiling."

(iv) Indicator of whether the resource is an on-Device resource or network resource.

(v) Location information about the Device that hosts this resource in case of an on-Device resource.

(vi) Associated Service information.

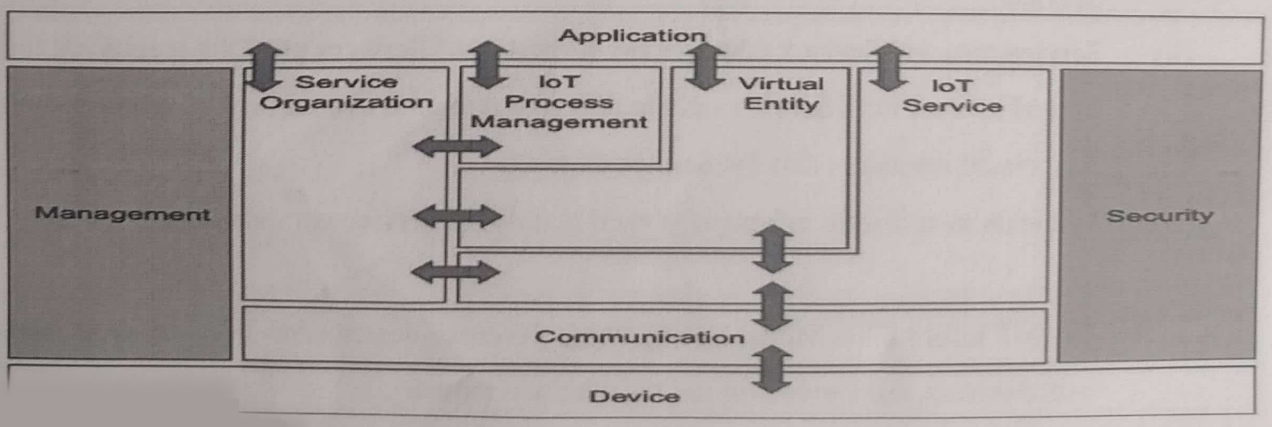
(vii) Associated Device description information.

3. Functional model :

- The IoT Functional Model aims at describing mainly the Functional Groups (FG) and their interaction with the ARM, while the Functional View of a Reference Architecture describes the functional components of an FG, interfaces, and interactions between the components.

The Functional View is typically derived from the Functional Model in conjunction with high-level requirements.

IoT-A Functional Model



(i) Device functional group :

- The Device FG contains all the possible functionality hosted by the physical Devices that are used for instrumenting the Physical Entities.

  This Device functionality includes sensing, actuation, processing, storage, and identification components, the sophistication of which depends on the Device capabilities.

(ii) Communication functional group :

- The Communication FG abstracts all the possible communication mechanisms used by the relevant Devices in an actual system in order to transfer information to the digital world components or other Devices.
- Examples of such functions include **wired bus** or **wireless mesh** technologies through which sensor Devices are connected to Internet Gateway Devices.

  Communication technologies used between Applications and other functions such as functions from the IoT Service FG are out of scope because they are the typical Internet technologies.

(iii) IoT Service functional group :

- The IoT Service FG corresponds mainly to the Service class from the IoT Domain Model, and contains single IoT Services exposed by Resources hosted on Devices or in the Network (e.g. processing or storage Resources).

  Support functions such as directory services, which allow discovery of Services and resolution to Resources, are also part of this FG.

(iv) Virtual Entity functional group :

- The Virtual Entity FG corresponds to the Virtual Entity class in the IoT Domain Model, and contains the necessary functionality to manage associations between Virtual Entities with themselves as well as associations between Virtual Entities and related IoT Services, i.e. the Association objects for the IoT Information Model.

  Associations between Virtual Entities can be static or dynamic depending on the mobility of the Physical Entities related to the corresponding Virtual Entities.
- An example of a static association between Virtual Entities is the hierarchical inclusion relationship of a building, floor, room,corridor,open space.
- An example of a dynamic association between Virtual Entities is a car moving from one block of a city to another.

(v) **IoT Service Organization functional group :**

- The purpose of the IoT Service Organization FG is to host all functional components that support the composition and orchestration of IoT and Virtual Entity services.
- Moreover, this FG acts as a service hub between several other functional groups such as the IoT Process Management FG.
- For example, service requests from Applications or the IoT Process Management are directed to the Resources implementing the necessary Services.
- Therefore, the Service Organization FG supports the association of Virtual Entities with the related IoT Services, and contains functions for discovery, composition, and choreography of services.
- Choreography is the brokerage of Services so that Services can subscribe to other services in a system.

(vi) IoT Process Management functional group :

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

(vii) Management functional group :

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

(viii) Security functional group :

- The Security FG contains the functional components that ensure the secure operation of the system as well as the management of privacy.

  The Security FG contains components for Authentication of Users (Applications, Humans), Authorization of access to Services by Users, secure communication (ensuring integrity and confidentiality of messages) between entities of the system such as Devices, Services, Applications, and last but not least, assurance of privacy of sensitive information relating to Human Users.

(ix) Application functional group :

- The Application FG is just a placeholder that represents all the needed logic for creating an IoT application.

  The applications typically contain custom logic tailored to a specific domain such as a Smart Grid.

(x) Modular IoT functions :

- It is important to note that not all the FGs are needed for a complete actual IoT system.
- The Functional Model, as well as the Functional View of the Reference Architecture, contains a complete map of the potential

  functionalities for a system realization.
- The functionalities that will eventually be used in an actual system are dependent on the actual system requirements.

  What is important to observe is that the FGs are organized in such a way that more complex functionalities can be built based on simpler ones, thus making the model modular.

4. Communication model :

- The communication model for an IoT Reference Model consists of the identification of the endpoints of interactions, traffic patterns (e.g. unicast vs. multicast), and general properties of the underlying technologies used for enabling such interactions.
- The potential communicating endpoints or entities are the Users, Resources, and Devices from the IoT Domain Model.

  Users include Human Users and Active Digital Artifacts (Services, internal system components, external applications).
- Devices with a Human-Machine Interface mediate the interactions between a Human User and the physical world (e.g. keyboards, mice, pens, touch screens, buttons, microphones, cameras, eye tracking, and brain wave interfaces, etc.), and therefore the Human User is not a communication model endpoint.
- The User (Active Digital Artifact) Service-to-Service interactions include the User-to-Service and Service-to-Service interactions (in the case of an enterprise service/ application accessing another service, or in the case of IoT service composition) as well as the Service-Resource-Device interactions.

The Userto-Service and Service-to-Service communication is typically based on Internet protocols

- Service-to-Service interactions when one or both Services are hosted on constrained/low-end Devices such as embedded systems.
- Typically, constrained Devices have a different communication stack from the ones used for the Internet-type of networks.

Therefore, the communication model for these interactions includes several types of gateways (e.g. network, application layer gateways) to bridge between two or more disparate communication technologies.

- A similar problem occurs between the Service-to-Resource communications.
- The Devices may be so constrained that they cannot host the Services, while the Resources could be hosted or not depending on the Device capabilities.
- This inability of the Device to host Resources or Services results in moving the corresponding Resources and/or Services out of the Device and into more powerful Devices or machines in the cloud.

Then the Resource-to-Device or the Service-to-Resource communication needs to involve multiple types of communication stacks.

5. Safety, privacy, trust, security model :

- An IoT system enables interactions between Human Users and Active Digital Artifacts (Machine Users) with the physical environment.
- The fact that Human Users are part of the system that could potentially harm humans if malfunctioning, or expose private information, motivates the Safety and Privacy needs for the IoT Reference Model and Architecture.

The Trust and Security Model are needed in every ICT system with the objective to protect the digital world.

(i) Safety :

- System safety is highly application- or application domain-specific, and is typically closely related to an IoT system that includes actuators that could potentially harm animate objects (humans, animals).

- For example, the operation of an IoT system controlling an elevator could harm humans if it allowed the elevator doors to open with a normal user interaction when the elevator room is not behind the doors.
- Critical infrastructure protection is also related to safety because the loss of such infrastructure due to a malicious user attack could be detrimental to humans,
  e.g. attacks to a Smart Grid could result in damages ranging from simple loss of electricity in a home to electricity loss in a hospital.
- A system designer of such critical systems typically follows an iterative process with two steps:
  (a) identification of hazards,
  (b) the mitigation plan.
- Not all hazards or mitigation steps include IoT technology, but a system designer could add safety assertions in relevant points in the interaction between Users, Services, Resources, and Devices.
- For example, a Human User interaction of pressing the elevator button should result in the elevator door opening only when a Sensor Device detects the elevator room to be behind the doors.
- If the system designer would like to provide for a safer elevator system, the system should include mechanical safety locks that work even in the absence of the loss of electricity.

(ii) Privacy :

- Because interactions with the physical world may often include humans, protecting the User privacy is of utmost importance for an IoT system.
- The IoT-A Privacy Model (Carrez et al. 2013, Gruschka & Gessner 2012) depends on the following functional components: Identity Management, Authentication, Authorization, and Trust & Reputation.
- **Identity Management** offers the derivation of several identities of different types for the same architectural entity with the objective to protect the original User identity for anonymization purposes.
- **Authentication** is a function that allows the verification of the identity of a User whether this is the original or some derived identity.
- **Authorization** is the function that asserts and enforces access rights when Users (Services, Human Users) interact with Services, Resources, and Devices.
- **The Trust and Reputation** functional component maintain the static or dynamic trust relationships between interacting entities.

(iii) Trust :

- According to the Internet Engineering Task Force (IETF) Internet Security Glossary (Shirey 2007), "Generally, an entity is said to 'trust' a second entity when the first entity makes the assumption that the second entity will behave exactly as the first entity expects."
- Trust and Reputation could be represented by a score, as seen earlier. This score could be used to impact the behavior of technical components interacting with each other.
- The necessary aspects of a trust model according to IoT-A (Gruschka & Gessner 2012) are as follows:
- **Trust Model Domains :**

  Because ICT and IoT systems may include a large number of interacting entities with different properties, maintaining trust relationships for every pair of interacting entities may be prohibitive. Therefore, groups of entities with similar trust properties can define different trust domains.

- **Trust Evaluation Mechanisms:** These are well-defined mechanisms that describe how a trust score could be computed for a specific entity.

  The evaluation mechanism needs to take into account the source of information used for computing the trust level/score of an entity; two related aspects are the federated trust and trust anchor. A related concept is the IoT support for evaluation of the trust level of a Device, Resource, and Service.
- **Trust Behavior Policies:** These are policies that govern the behavior between interacting entities based on the trust level of these interacting entities;

  for example, how a User could use sensor measurements retrieved by a Sensor Service with a low trust level.
- **Trust Anchor:** This is an entity trusted by default by all other entities belonging to the same trust model, and is typically used for the evaluation of the trust level of a third entity.
- **Federation of Trust:** A federation between two or more Trust Models includes a set of rules that specify the handling of trust relationships between entities with different Trust Models.

  Federation becomes important in large-scale systems.

(iv) Security :

The Security Model for IoT consists of communication security that focuses mostly on the confidentiality and integrity protection of interacting entities and functional components such as

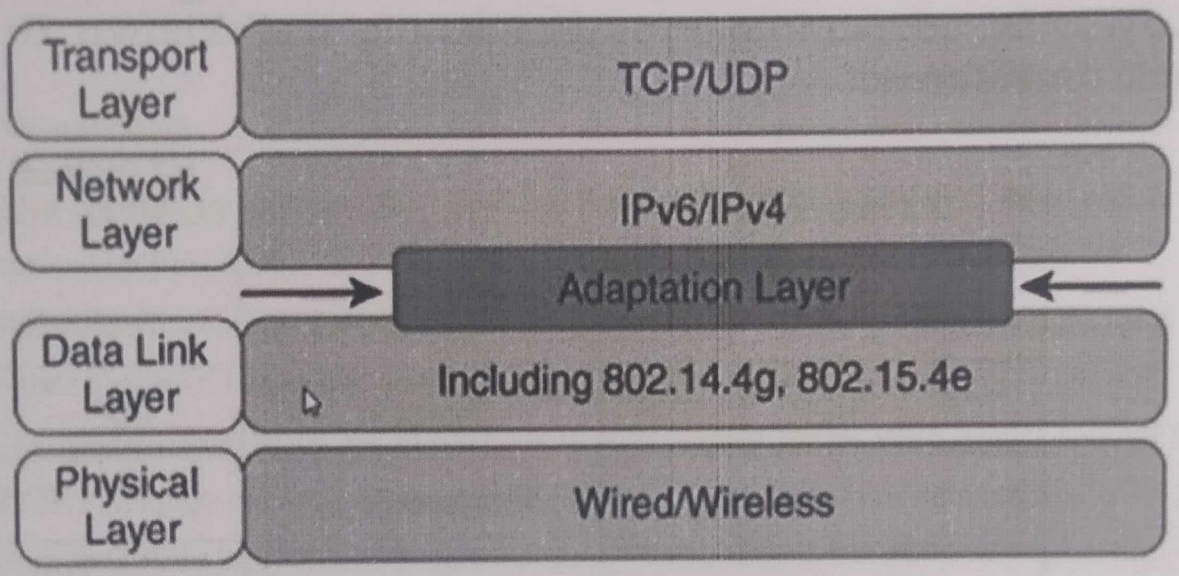
...gement, Authentication, Authorization, and Trust & Reputation, as seen earlier.

...ommunications Technology

3.4. 6LowPAN : (IPV6 over Low-power Wireless Personal Area Networks)

Optimizing IP for IoT :

- While the internet protocol is key for a successful internet of Things, constrained nodes and Constrained networks mandate optimization at various layers and on multiple protocols of the IP architecture.
- Figure 5.1 highlights the TCP/IP layers where optimization is applied.



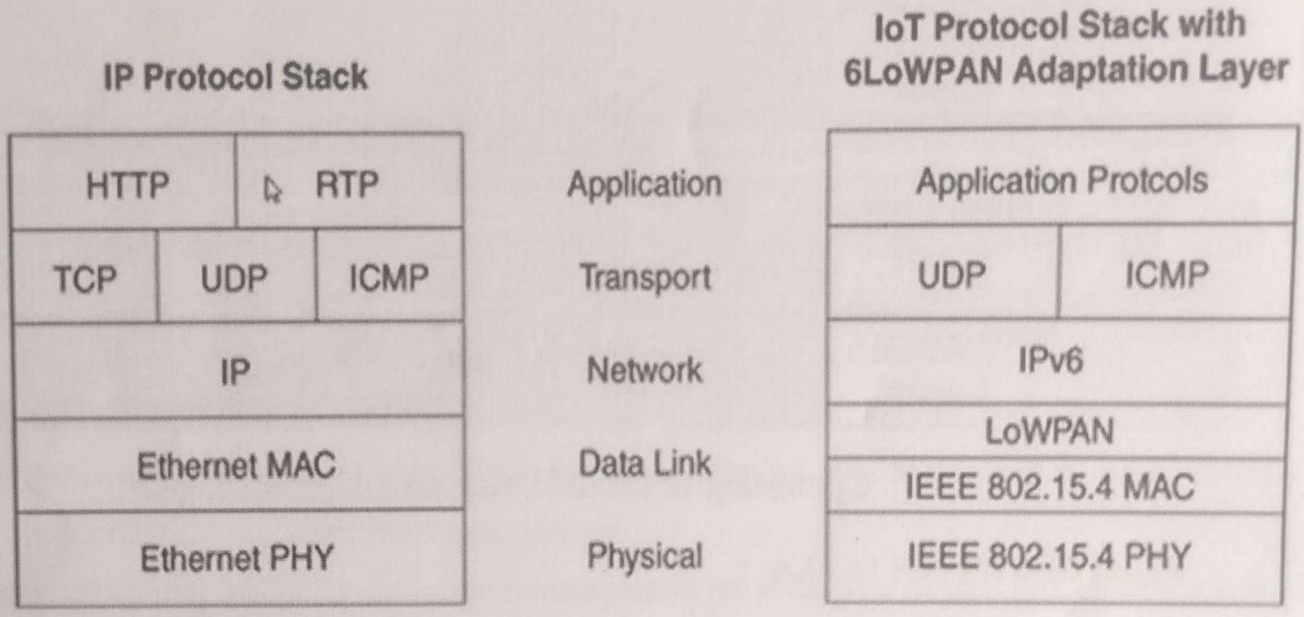
Optimizing IP for IoT Using an Adaptation Layer

From 6LoWPAN to 6Lo :

- In the IP architecture, the transport of IP packets over any given Layer 1 (PHY) and Layer 2 (MAC) protocol must be defined and documented.
- The model for packaging IP into lower-layer protocols is often referred to as an **adaptation layer.**
- Unless the technology is proprietary. IP adaptation layers are typically defined by an IETF working group and released as a request for comments (RFC).
- An RFC is a publication from IETF that officially documents internet standards, specifications, protocols, procedures, and events.
- For ex : RFC 864 describes how an IPV4 packet gets encapsulated over an Ethernet frame and RFC 2464 describes how the same function is performed for an IPV6 packets.

- IoT-related protocols follow a similar process. The main difference is that an adaptation lay designed for IoT may include some optimizations to deal with constrained nodes and networks.
- The main examples of adaptation layers optimized for constrained nodes or "things" are the ones under the 6LoWPAN working group and its successor, the 6LoWPAN working group and its successor, the 6Lo working group.
- The initial focus of the 6LoWPAN working group was to optimize the transmission of IPV6 packets over constrained networks such as IEEE 802.15.4.

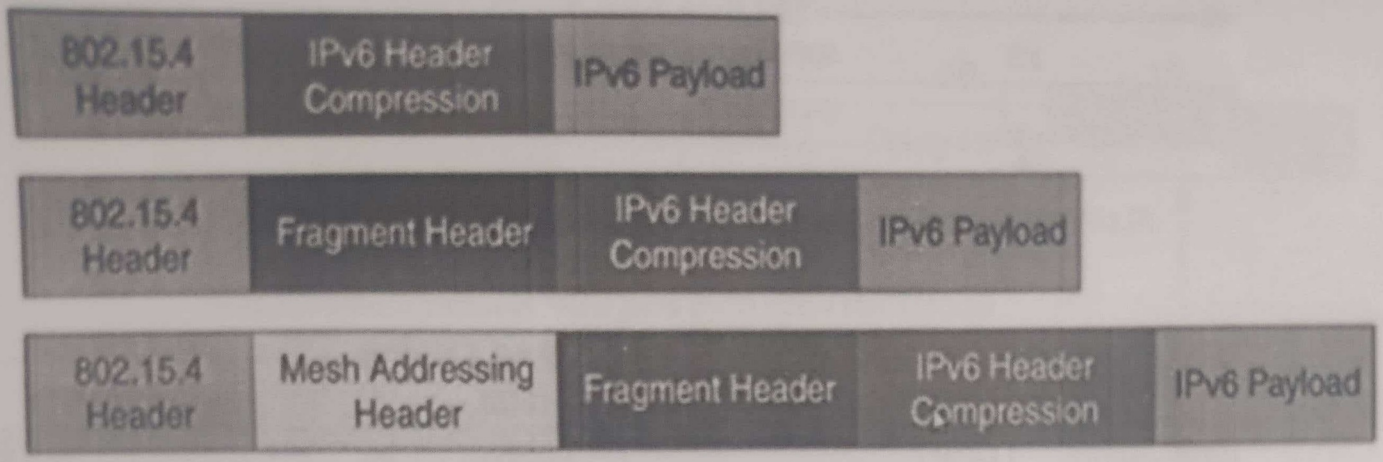
an example of an IoT protocol stack using the 6LoWPAN adaptation layer beside the well-known IP protocol stack for reference.



Comparison of an IoT Protocol Stack Utilizing 6LoWPAN and an IP Protocol Stack

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

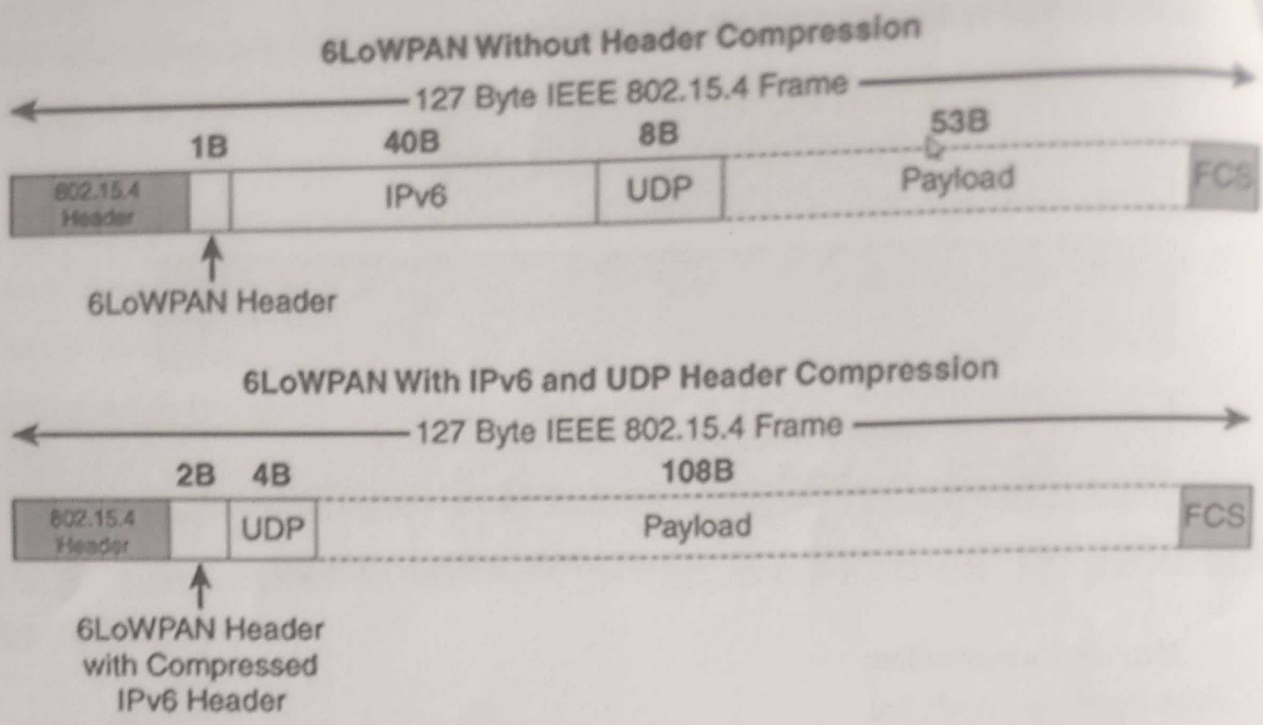
examples of typical 6LoWPAN header stacks



6LoWPAN Header Stacks

Header Compression:

- IPV6 header compression for 6LoWPAN was define initially in RFC 4944 and subsequently updated by RFC 6282.
- This capability shrinks the size of IPV6's 40-byte headers and User Datagram Protocol's (UDP's) 8-byte headers down as low as 6 bytes combined in some cases.
- The header compression for 6LoWPAN is only defined for an IPV6 header and not IPV4. The 6LoWPAN protocol does not support IPV4, and, in fact, there is no standardized IPV4 adaptation layer for IEEE 802.15.4.
- 6LoWPAN header compression is stateless , and conceptually it is not too complicated.
- However, a number of factors affect the amount of compression, such as implementation of RFC 4944 versus RFC 6922 , whether UDP is included ,and various IPV6 addressing scenarios.
- At a high level , 6LoWPAN works by taking advantage of shared information known by all nodes from their participation in the local network.
- In addition ,it omits some standard header fields by assuming commonly used values.
- Figure 5.4 highlights an example that shows the amount of reduction that is possible with 6LoWPAN header compression.
- At the top of Figure , we see a 6LoWPAN frame without any header compression enabled :
  - The full 40-byte IPv6 header and 8-byte UDP header are visible. The 6LoWPAN header is only single byte in this case.

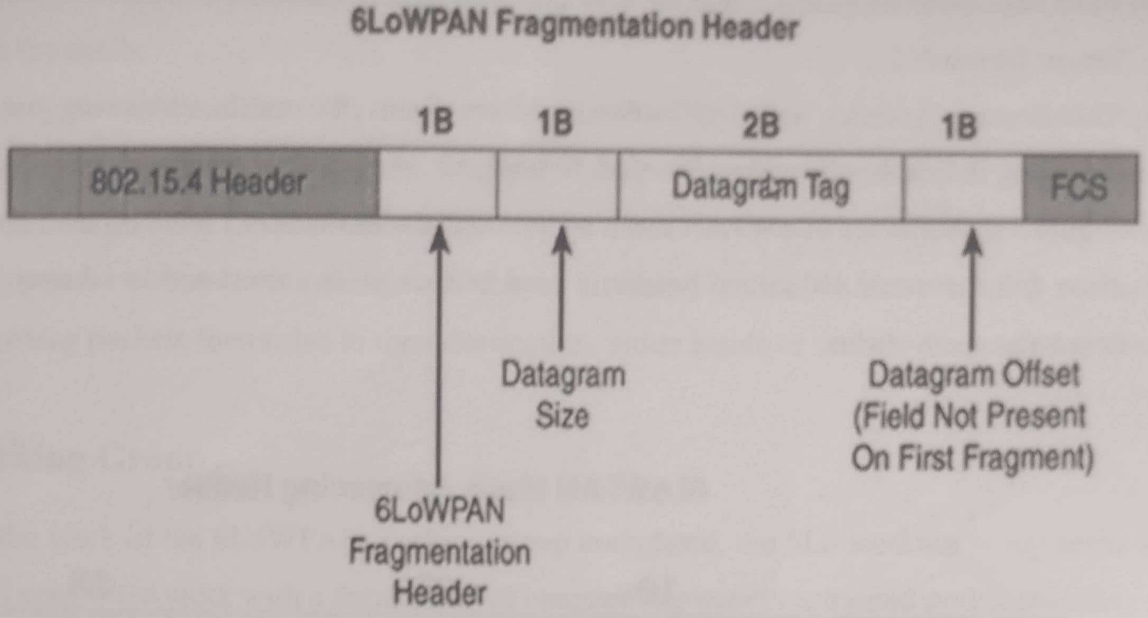


6LoWPAN Header Compression

- Notice that uncompressed IPV6 and UDP headers leave only 53 bytes of data payload out of the 127-bytes maximum frame size in the case of IEEE 802.15.4.
- The bottom half of figure shows a frame where header compression has been enabled for a best-case scenario.
- The 6LoWPAN header increase to 2 bytes to accommodate the compression IPV6 header and UDP has been reduced in half to 4 bytes from 8.
- Most importantly the header compression has allowed the payload to more than double from 53 bytes to 108bytes which is obviously much more efficient.

Fragmentation :

- The maximum transmission unit (MTU) for an IPv6 network must be at least 1280 bytes.
- The term MTU defines the size of the largest protocol data unit that can be passed. For IEEE 802.15.4, 127 bytes is the MTU.
- We can see that this is a problem because IPv6 with a much larger MTU, is carried inside the 802.15.4 frame with a much smaller one.
- To remedy this situation , large IPv6 packets must be fragmented across multiple 802.15.4 frames at Layer 2.
- The fragment header utilized by 6LoWPAN is composed of three primary fields.



6LoWPAN Fragmentation Header

I. Datagram Size

I. Datagram Tag and

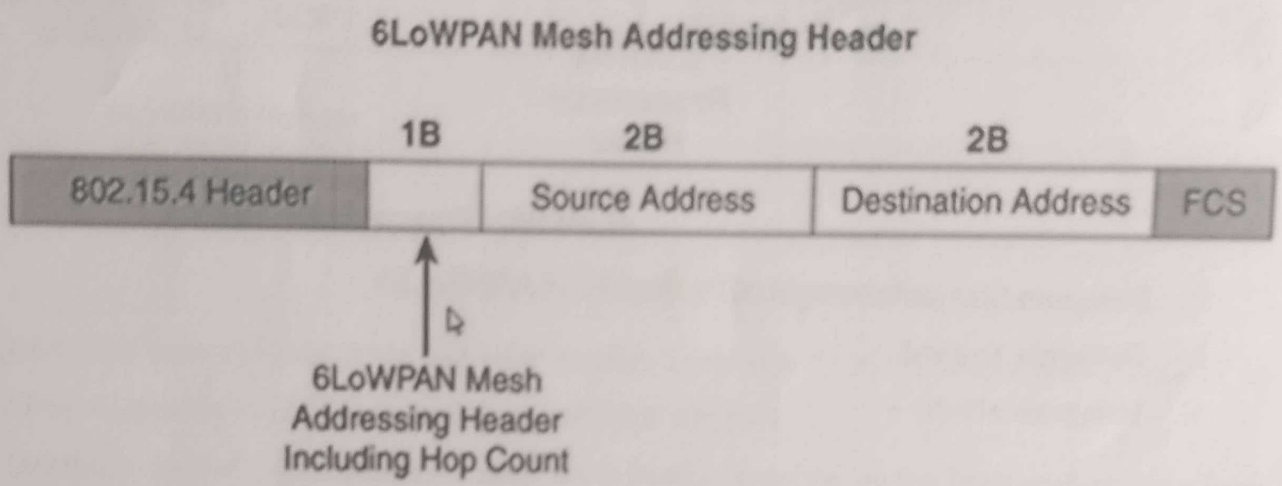
II. Datagram offset

- The 1-byte Datagram Size field specifies the total size of the unfragmented payload.
- Datagram Tag identifies the set of fragmented for a payload.Finally, the Datagram Offset field delineates how far into a payload a particular fragment occurs.
- Figure 5.5 provides an overview of a 6LoWPAN fragmentation header. The 6LoWPAN fragmentation header field itself uses a unique bit value to identify that the subsequent fileds behind it are fragment fields are opposed to another capability, such as header compression.
- Also, in the first fragment, the Datagram Offset field is not present because it would be set to 0.
- This results in the first fragmentation header for an IPv6 payload being only 4 bytes long. The remainder of the fragments have a 5-bytes header field so that the appropriate offset can be specified.

Mesh Adressing :

- The purpose of the 6LoWPAN mesh addressing function is to forward packets over multiple hops.
- Three fields are defined for this header: Hop Limit, Source Adress, and Destination Address.
- Analogous to the IPv6 hop limit field, the hop limit for mesh addressing also provides an upper limit on how many times the frame can be forwarded.

- Each hop decrements this value by 1 as it is forwarded. Once the value hits 0, it is dropped and no longer forwarded.
- The Source Address and Destination Address fields for mesh addressing are IEEE 802.15.4 adressing indicating the endpoints of an IP hop.
- Figure 5.6 details the 6LoWPAN mesh addressing header fields.
- Note that the mesh addressing header is used in a single ip subnet and is a Layer 2 type of routing known as mesh-under.



6LoWPAN Mesh Addressing Header

Mesh-under versus Mesh-over Routing :

- For network technologies such as IEEE 802.15.4, IEEE 802.15.4g, and IEEE 1901.2a that support mesh topologies and operate at the physical and data link layers, two main options exit for establishing reachability and forwarding packets.
- With the first option, mesh-under, the routing of packets is handled at the 6LoWPAN adaption layer.
- The other option, known as **"mesh-over"** or **"route-over"**, utilizes IP routing for getting packets to their destination.
- With mesh-under routing, the routing of IP packets leverages the 6LoWPAN mesh addressing header to route and forward packets at the link layer.
- The term **mesh-under** is used because multiple link layer hops can be used to complete a single IP hop.

- Nodes have a Layer 2 forwarding table that they consult to route the packets to their final destination within the mesh.
- A edge gateway terminates the mesh-under domain. The edge gateway must also implement a mechanism to translate between the configured Layer 2 protocol and any IP routing mechanism implemented on other Layer IP interfaces.
- In mesh-over or route-over scenarios, IP Layer 3 routing is utilized for computing reachability and then getting packets forwarded to their destination, either inside or outside the mesh domain.

6Lo Working Group

- With the work of the 6LoWPAN working group completed, the 6Lo working group seeks to expand on this completed work with a focus on IPv6 connectivity over constrained-node networks.
- While the 6LoWPAN working group initially focus its optimizations on IEEE 802.15.4 LLNs, standardizing IPv6 over other link layer technologies is still needed.
- Therefore, the charter of the 6Lo working group, now called the IPv6 connectivity over constrained-node networks.
- In particular, this working group is focused on the following:

➢ **IPv6-over-foo adaption layer specifications using 6LoWPAN technologies(RFC4944, RFC6282, RFC6775) for link layer technologies:**

For example,this includes:

- IPv6 over Bluetooth Low Energy
- Transmission of IPv6 packets over near-field communication
- IPv6 OVER 802.11 ah
- Transmission of IPv6 packets over DECT Ultra Low Energy
- Transmission of IPv6 packets on WIN-PA (Wireless Networks for Industrial Automation-Process Automation)

3.5. RPL (Routing Protocol for low power and lossy Network) :

INTRODUCTION

- Low Power and Lossy Networks(LLN) are resource constrained.
- Routers are usually limited in terms of processing power, battery and memory, and their interconnects are characterized by unstable links with high loss rates, low data rates and low packet delivery rates
- The traffic patterns could be P2P or P2MP or MP2P
- Lossy means the packet drop rate will be high.
- RPL is a distance vector routing protocol.
- RPL mainly targets collection-based networks, where nodes periodically send measurements to a collection point.
- The protocol was designed to be highly adaptive to network conditions and to provide alternate routes,whenever default routes are inaccessible.
- RPL provides a mechanism to disseminate information over the dynamically formed network topology
- Contains thousands of nodes.

RPL topology

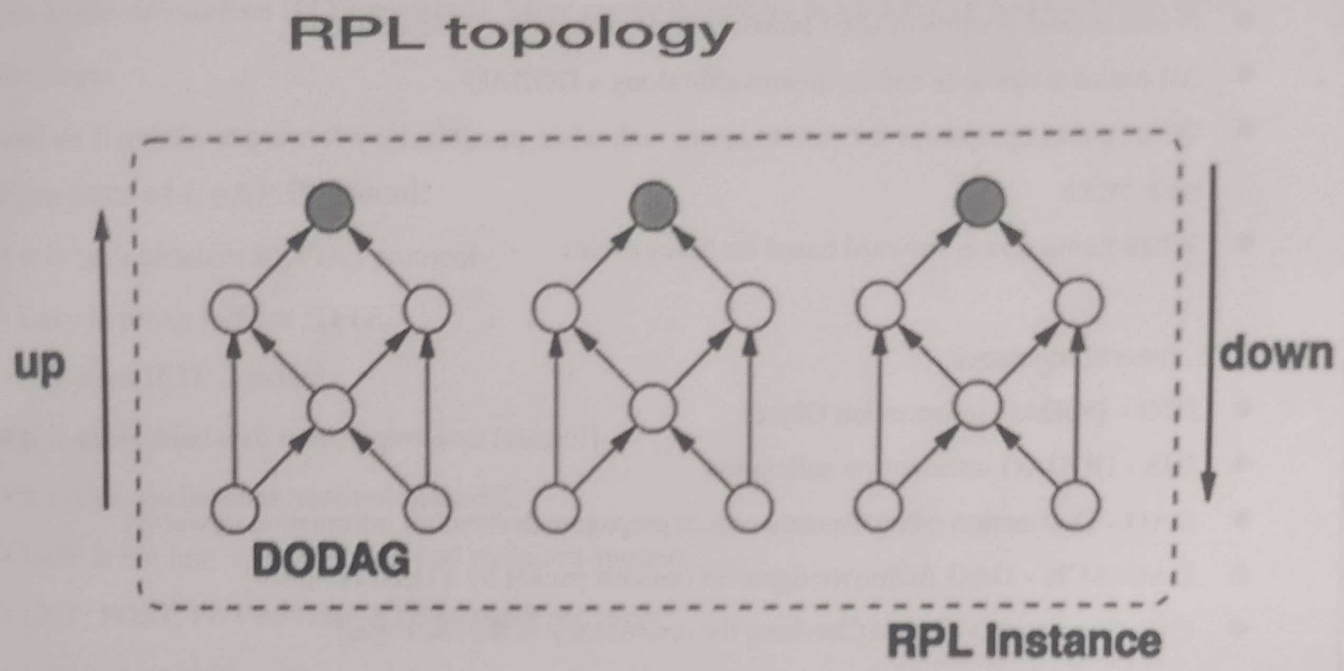
- DODAG(Destination Oriented Directed ACYLIC Graphs)
  - A DODAG is a DAG rooted at a single destination. The DODAG root has no outgoing Edges. A DODAG is a uniquely identified by a combination of RPL Instance ID and DODAG ID.

- Rank
  - A nodes Rank defines the nodes individual position relative to other nodes with Respect to a DODAG root.Rank strictly increases in the DOWN direction and strictly decreases in the up direction.

- DODAG Root
  - the DODAG root is the DAG root of the DODAG. The DODAG root may act as a border router for the DODAG, and aggregate routes in the DODAG and may redistribute DODAG routes into other routing protocols.

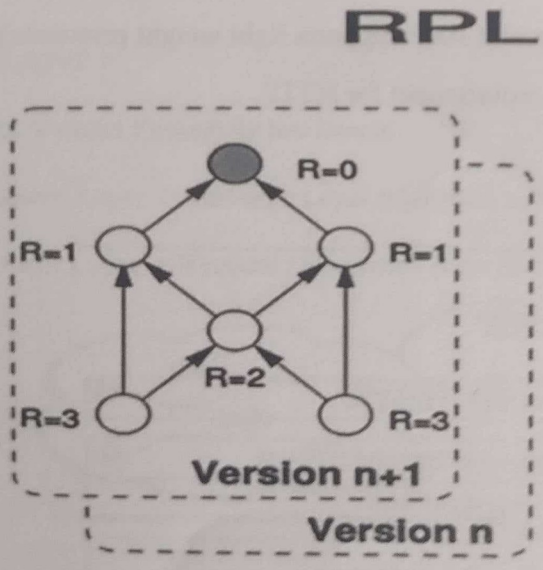
- Upward path is so common(mp2p)
- Downward path is optional mainly for p2p and p2mp

- An RPL Instance consists of multiple Destination Oriented Directed Acyclic Graphs(DODAG).Traffic moves either up towards the DODAG root or down towards the DODAG leafs.



RPL Instance

- DODAGS are disjoint(no shared nodes)
- Link properties: (reliability,latency,..)\node properties:(powered or not,..)
- RPL Instance has an optimization objective
- Multiple RPL Instances with different optimization objectives can coexist



- A node's Rank defines the node's individual position relative to other nodes with respect to a DODAG root. The scope of Rank is a DODAG Version.

Forwarding and routing

- Up routes towards nodes of decreasing rank (parents), Down routes towards nodes of nodes of increasing rank.
- Nodes inform parents of their presence and reachability to descendants.
- All routes g upwards and/or downwards along a DODAG
- When going up, always forward to lower rank when possible, may forward to sibling if no lower rank exists
- When going down, forward based on down routes

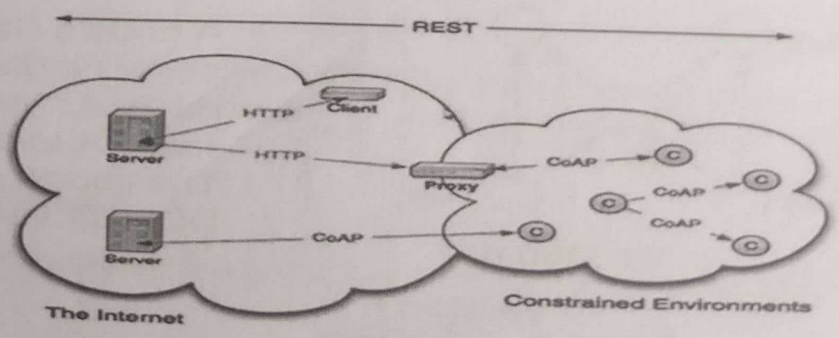
RPL control messages

- **DIO** - DODAG information Object
- **DIS** - DODAG information soliciation
- **DAO** - Destination advertisement object(propagate destination information upwards)
- **DAO-ACK** - DAO Acknowledgement (unicast packet by a DAO recipient)
- **CC** - Consistency Check(Checking for consistency in the messages)

3.6. CoAP : (Constrained Application Protocol)

CoAP is the short form of Constrained Application Protocol. The CoAP specification is maintained by the Internet Engineering Task Force (IETF). CoAP is very conversational by nature because it is request- and response-driven. CoAP developed to enable smart devices to connect to the internet and it is a 1 to 1 communication protocol. CoAP supports light weight protocols and it certainly uses lesser resources than HTTP and it is not a replacement for HTTP.

CoAP Architecture :



CoAP Architecture

It extends normal HTTP clients to clients having resource constraints. These clients are known as CoAP clients. Proxy device bridges gap between constained environment and typical internet environment based on HTTP protocols. Same server takes care of both HTTP and CoAP protocol messages.

Features of CoAP Protocol:

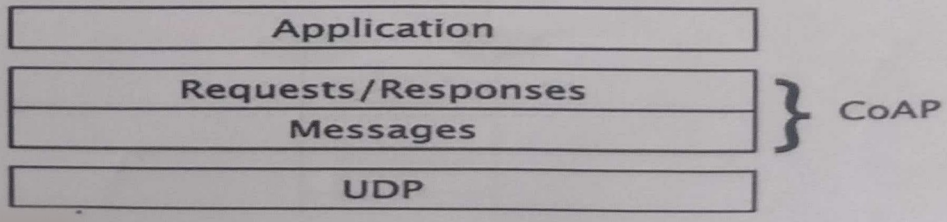
- It is very efficient RESTful protocol.
- Easy to proxy to/from HTTP.
- It is open IETF standard
- It is Embedded web transfer protocol (coap://)
- It uses asynchronous transaction model.
- UDP is binding with reliability and multicast support.
- GET, POST, PUT and DELETE methods are used.
- URI is supported.
- It uses small and simple 4 byte header.
- Supports binding to UDP, SMS and TCP.
- DTLS based PSK, RPK and certificate security is used.
- uses subset of MIME types and HTTP response codes.
- Uses built in discovery mechanism.

CoAP Layer :

The protocol works through its two layers:

Lower Layer -> Message Layer (Happens with UDP)

Upper Layer -> Request / Response (everything happens here)



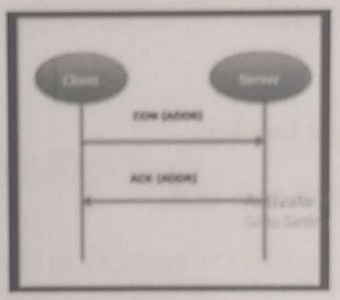
(i) Message Layer :

In Message layer, it supports 4 types of messages. They are :

1. Confirmable Message (CON)
2. Non Confirmable Message (NON)
3. Acknowledgement (ACK)
4. Reset (RST)

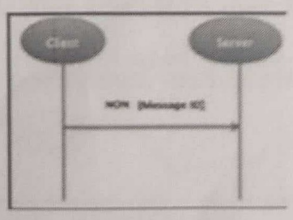
➔ Confirmable Message (CON) :

Confirmable Message (CON) is a reliable Messaging concepts and it also supports retransmission until acknowledgement arrives with the same message ID. If any Timeout or Recipient fail to process massage, it will send RST message with the response.



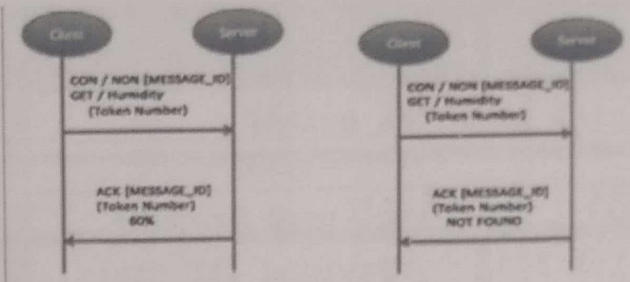
➔ Non Confirmable Message (NON) :

Non Confirmable Message (NON) are unreliable message. No Acknowledgement here, if no processing done by receiver, RST message will be sent.



➔ **Acknowledgement (ACK) :**

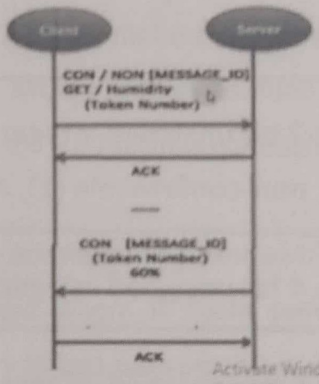
Acknowledgement shall be received immediately with corresponding token number and message. If not available, the failure code shall be embedded as a part of the ACK.



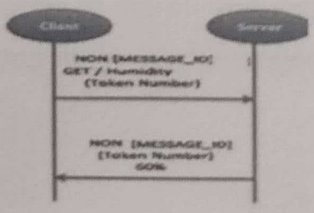
Piggybacking. To reduce message traffic and processing overhead, CoAP supports the concept of piggybacking. With this, the response data to a request can be included (piggybacked) on the ACK message.

(ii) Request / Response Layer Messages :

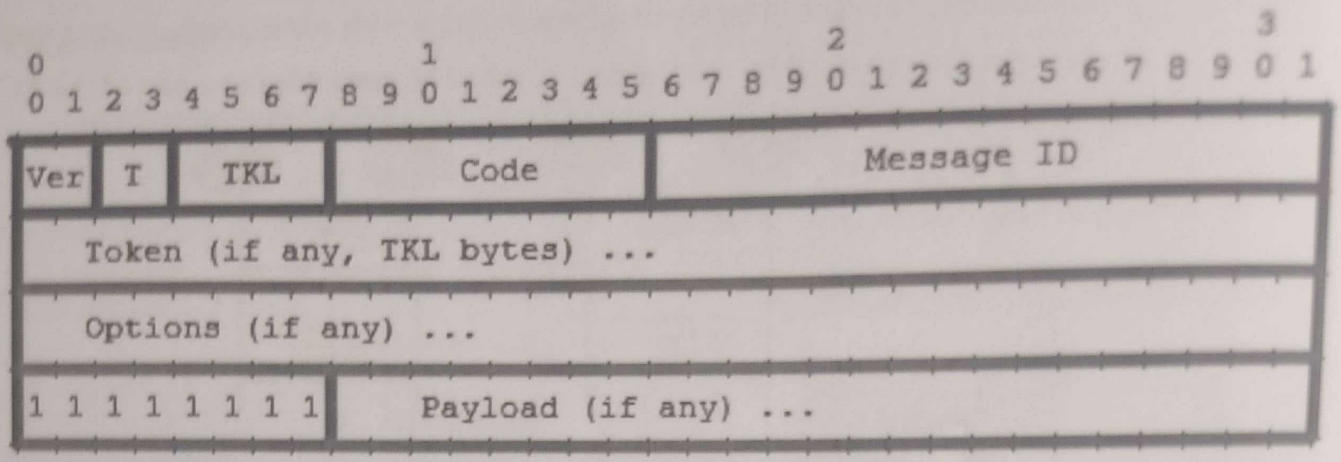
➔ **Separate Response :** When a CON type message is sent to the server from the client and in case, the server is unable to respond immediately, an empty ACK shall be revereted. After some time , when the server can send the response, it shall send a CON message with data. ACK shall be sent back from the client.



➔ **Non Confirmable Request and Response :** Here , NON type message shall be sent from the client to server. Server need not give ACK. Server can send a NON type response in turn.



CoAP Message Format | CoAP Header :



CoAP Message Format

CoAP message format consists of 4 bytes header followed by token value (from 0 to 8 bytes). The table below mentions header which consists of 4 bytes i.e. 32 bits.

CoAP message header	Description
Ver	It is 2 bit unsigned integer. It mentions CoAP version number. Set to one.
T	It is 2 bit unsigned integer. Indicates message type viz. confirmable (0), non-confirmable (1), ACK (2) or RESET(3).
TKL	It is 4 bit unsigned integer, Indicates length of token (0 to 8 bytes).
Code	It is 8 bit unsigned integer, It is split into two parts viz. 3 bit class (MSBs) and 5 bit detail (LSBs).
Message ID	16 bit unsigned integer. Used for matching responses. Used to detect message duplication.

CoAP vs MQTT :

CoAP and MQTT are distinct from each other on various fronts:

	MQTT	CoAP
BASE PROTOCOL	TCP	UDP
COMMUNICATIONNODE	M:N MANY TO MANY	1:1 ONE TO ONE
POWER CONSUMPTION	HIGHIER than CoAP	LOWER than MQTT
ETC	Pub/Sub Model	Req/Resp Model Support RESTful

3.7. MQTT : (MQ Telemetry Transport)

MQTT (MQ Telemetry Transport) is a lightweight open messaging protocol that provides resource-constrained network clients with a simple way to distribute telemetry information in low-bandwidth environments.

While the *TT* in MQTT stands for Telemetry Transport, the *MQ* is in reference to a product called IBM MQ. Although the spell-out for *MQTT* is sometimes given as Message Queuing Telemetry Transport, there is no message queuing in MQTT communication. The protocol, which employs a publish/subscribe communication pattern, is used for machine-to-machine (M2M) communication.

Created as a low-overhead protocol to accommodate bandwidth and CPU limitations, MQTT was designed to run in an embedded environment where it could provide a reliable, effective path for communication. MQTT is a good choice for wireless networks that experience varying levels of latency due to occasional bandwidth constraints or unreliable connections.

Although MQTT started as a proprietary protocol used to communicate with supervisory control and data acquisition (SCADA) systems in the oil and gas industry, it has become popular in the smart device arena and today is the leading open source protocol for connecting internet of things (IoT) and industrial IoT (IIoT) devices.

➢ **How does MQTT work :**

- Aimed at maximizing the available bandwidth, MQTT's publish/subscribe (pub/sub) communication model is an alternative to traditional client-server architecture that communicates directly with an endpoint.
- By contrast, in the pub/sub model, the client that sends a message (the publisher) is decoupled from the client or clients that receive the messages (or the subscribers). Because neither the publishers nor the subscribers contact each other directly, third parties -- the brokers -- take care of the connections between them.
- MQTT clients include publishers and subscribers, terms that refer to whether the client is publishing messages or subscribed to receive messages.
- These two functions can be implemented in the same MQTT client. When a device (or client) wants to send data to a server (or broker) it is called a *publish*.
- When the operation is reversed, it is called a *subscribe*. Under the pub/sub model, multiple clients can connect to a broker and subscribe to topics in which they are interested.
- If the connection from a subscribing client to a broker is broken, then the broker will buffer messages and push them out to the subscriber when it is back online.
- If the connection from the publishing client to the broker is disconnected without notice, then the broker can close the connection and send subscribers a cached message with instructions from the publisher.
- An IBM writeup describes the pub/sub model: "Publishers send the messages, subscribers receive the messages they are interested in, and brokers pass the messages from the publishers to the subscribers.
- Publishers and subscribers are MQTT clients, which only communicate with an MQTT broker. MQTT clients can be any device or application (from microcontrollers like the Arduino to a full application server hosted in the Cloud) that runs an MQTT library."

➢ **What is an MQTT broker :**

- An MQTT broker acts as a go-between for the clients who are sending messages and the subscribers who are receiving those messages.
- In a post office analogy, the broker is the post office itself. All messages have to go through the broker before they can be delivered to the subscriber.

- Brokers may have to handle millions of concurrently connected MQTT clients, so when choosing an MQTT broker, enterprises should rate them based on their scalability, integration, monitoring and failure-resistance capabilities.

➢ **Types of MQTT messages :**

- An MQTT session is divided into four stages:
  - → connection,
  - → authentication,
  - → communication
  - → termination.

(i) **Connection :**

- A client starts by creating a Transmission Control Protocol/Internet Protocol (TCP/IP) connection to the broker by using either a standard port or a custom port defined by the broker's operators.
- When creating the connection, it is important to recognize that the server might continue an old session if it is provided with a reused client identity.

(ii) Authentication :

- The standard ports are 1883 for nonencrypted communication and 8883 for encrypted communication -- using Secure Sockets Layer (SSL)/Transport Layer Security (TLS).
- During the SSL/TLS handshake, the client validates the server certificate and authenticates the server.
- The client may also provide a client certificate to the broker during the handshake. The broker can use this to authenticate the client.
- While not specifically part of the MQTT specification, it has become customary for brokers to support client authentication with SSL/TLS client-side certificates.

(iii) Communication :

- During the communication phase, a client can perform publish, subscribe, unsubscribe and ping operations. The publish operation sends a binary block of data -- the content -- to a topic that is defined by the publisher.
- MQTT supports message binary large objects (BLOBs) up to 256 MB in size. The format of the content will be application-specific. Topic subscriptions are made using a

SUBSCRIBE/SUBACK packet pair, and unsubscribing is similarly performed using UNSUBSCRIBE/UNSUBACK packet pair.

(iv) Termination :

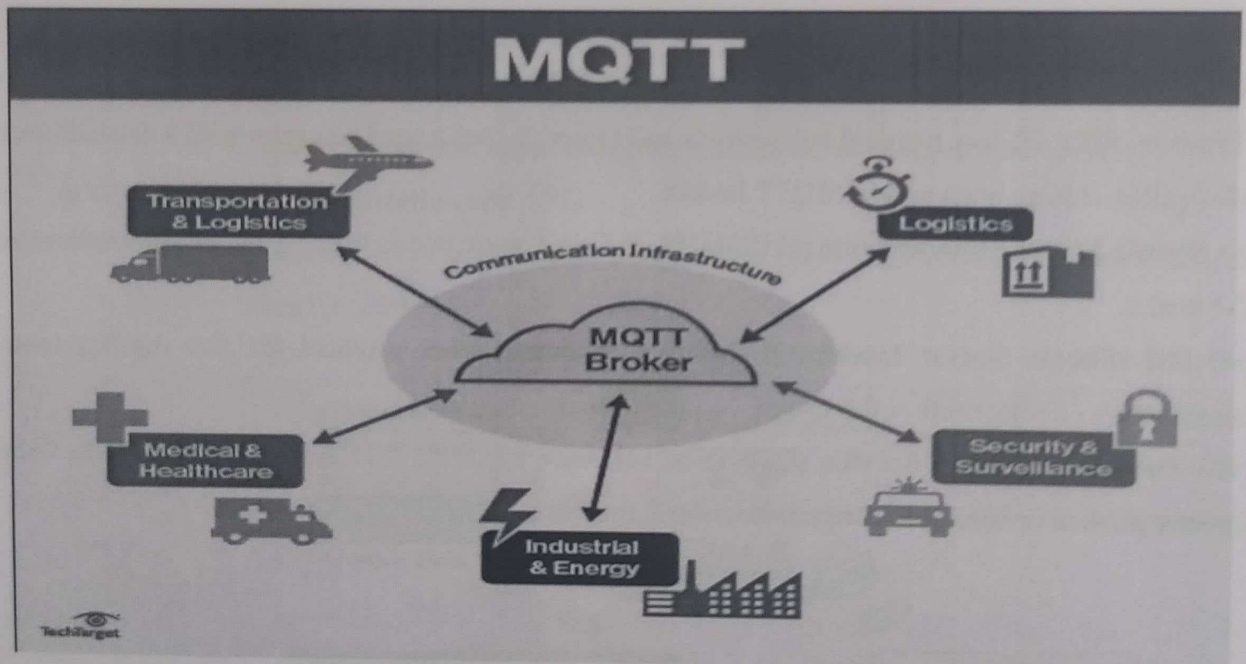
- When a publisher or subscriber wants to terminate an MQTT session, it sends a DISCONNECT message to the broker and then closes the connection.
- This is called a *graceful shutdown* because it gives the client the ability to easily reconnect by providing its client identity and resuming where it left off.

MQTT Message	Description
CONNECT	Client request to connect to server
CONNACK	Connect acknowledgement
PUBLISH	Publish message
PUBACK	Publish acknowledgement
PUBREC	Publish received
PUBREL	Publish release
PUBCOMP	Publish complete
SUBSCRIBE	Client subscribe request
SUBACK	Subscribe acknowledgement
UNSUBSCRIBE	Unsubscribe request
UNSUBACK	Unsubscribe acknowledgement
PINGREQ	PING request
PINGRESP	PING response
DISCONNECT	Client is disconnecting

➢ What are the benefits of using MQTT? :

- Efficient data transmission and quick to implement, due to its being a lightweight protocol
- Low network usage, due to minimized data packets
- Efficient distribution of data
- Successful implementation of remote sensing and control;
- Fast, efficient message delivery;
- Uses small amounts of power, which is good for the connected devices; and
- Optimizes network bandwidth.

- **What are the drawbacks of MQTT? :**
  - MQTT has slower transmit cycles compared to Constrained Application Protocol (CoAP).
  - MQTT's resource discovery works on flexible topic subscription, whereas CoAP uses a stable resource discovery system.
  - MQTT is unencrypted. Instead, it uses TLS/SSL (Transport Layer Security/Secure Sockets Layer) for security encryption.
  - It is difficult to create a globally scalable MQTT network.
  - Other MQTT challenges relate to security, interoperability and authentication.
- **MQTT protocol applications and use cases :**
  - Due to its lightweight properties MQTT works well for applications involving remote monitoring, including the following:
  - Synchronization of sensors, such as fire detectors or motion sensors for theft detection, to determine if a hazard is valid.
  - Monitoring health parameters using sensors for patients leaving a hospital.
  - Sensors alerting people of danger.



➢ **How is MQTT used in IoT? :**

- Examples of MQTT use in IoT or IIoT instructure include the following:

(i) Smart metering: The MQTT protocol can be used to transmit data with guaranteed message delivery to provide accurate meter readings in real time. This helps make billing more accurate.

(ii) Gathering ambient sensor data: Sensors used in remote environments are often low-power devices, so MQTT is a good fit for IoT sensor buildouts with lower-priority data transmission needs.

(iii) Machine health data. Ably, which provides a pub/sub messaging platform, gives the example of a wind turbine requiring "guaranteed delivery of machine health data to local teams even before that information hits a data center."

(iv) Billing systems. MQTT helps eliminate duplicate or lost message packets in billing or invoicing.

➢ **Competing protocols :**

- Other transfer protocols that compete with MQTT include the following:

(i) Constrained Application Protocol (CoAP). Well suited for IoT, it uses a request/response communication pattern.

(ii) Advanced Message Queuing Protocol (AMQP). Like MQTT, it uses a publish/subscribe communication pattern.

(iii) Simple/Streaming Text Oriented Messaging Protocol (STOMP). It is a text-based protocol. However, STOMP does not deal with queues and topics; it uses a send semantic with a destination string.

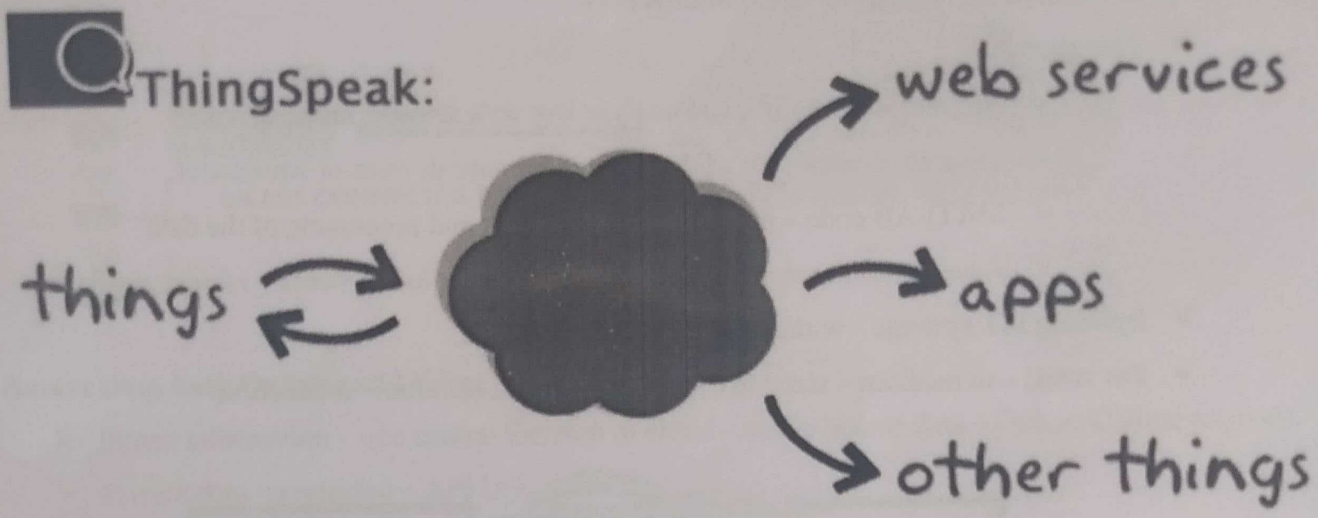
Mosquitto. It is an open source MQTT broker.

(v) Simple Media Control Protocol (SMCP). A CoAP stack that is used in embedded environments, is C-based.

(vi) SSI (Simple Sensor Interface). This is a communications protocol for data transfer between a combination of computers and sensors.

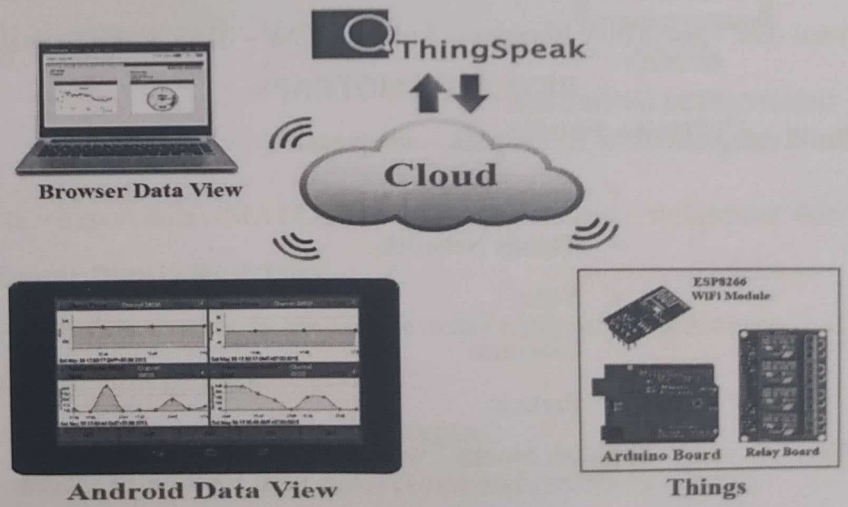
(vii) Data Distribution Service (DDS). For real-time systems, it is a middleware standard that can directly publish or subscribe communications in real time in embedded systems.

3.8. IoT frameworks- Thing Speak :



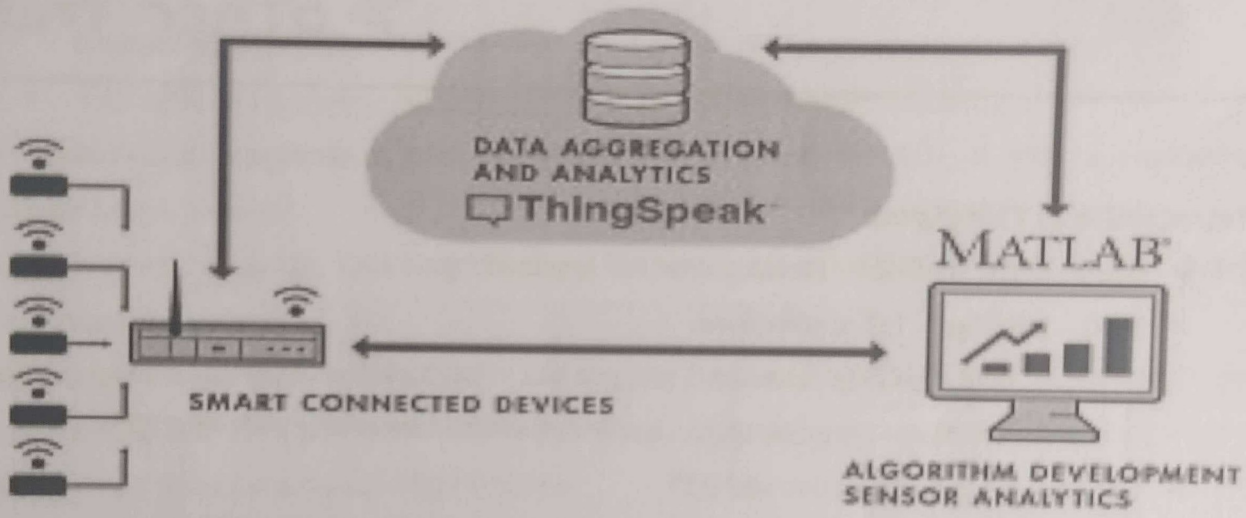
Introduction to ThingSpeak

- Open-Source platform - Development IoT applications.
- 2010 – ioBridge - IoT applications.
  - ✓ Real time data collection , analysis and visualization – charts.
  - ✓ Creation – plug-ins, apps – e services, social networking and other APIs.
- Open source IoT application and API
  - ✓ Store and retrieve data from things using HTTP and MQTT protocol – Internet/Local Area Network.



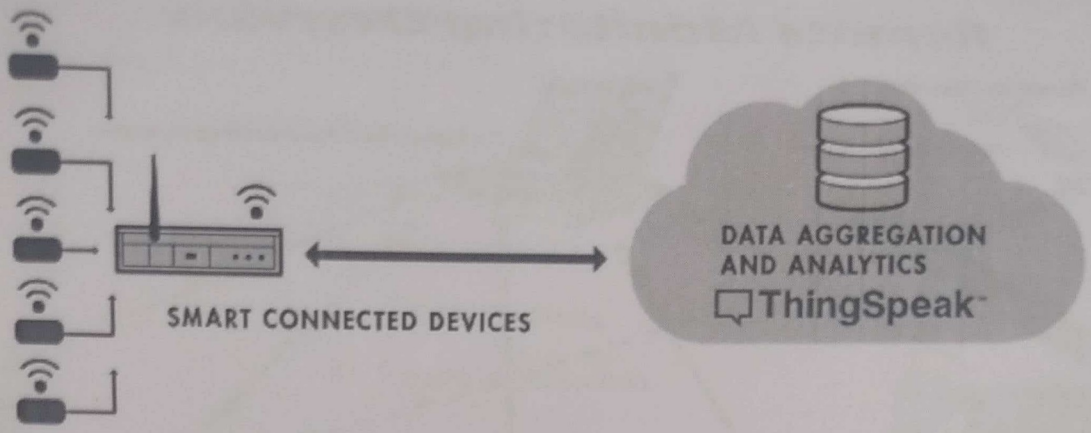
ThingSpeak for IoT

- Integrated support – MATLAB – analyze and visualize uploaded data – without license.
- IoT analytics platform service – MathWorks.
- Thingspeak
  - ✓ Aggregate ,visualize, and analyze live data streams in the cloud.
  - ✓ Instant visualizations of data posted by your devices or equipment.
  - ✓ MATLAB code – perform online analysis and processing of the data.
  - ✓ Accelerates development of proof-of-concept IoT systems – require analytics.
- Buliding IoT systems – without servers / web software.
- For small – to medium – sized IoT systems, - hosted solution – production.



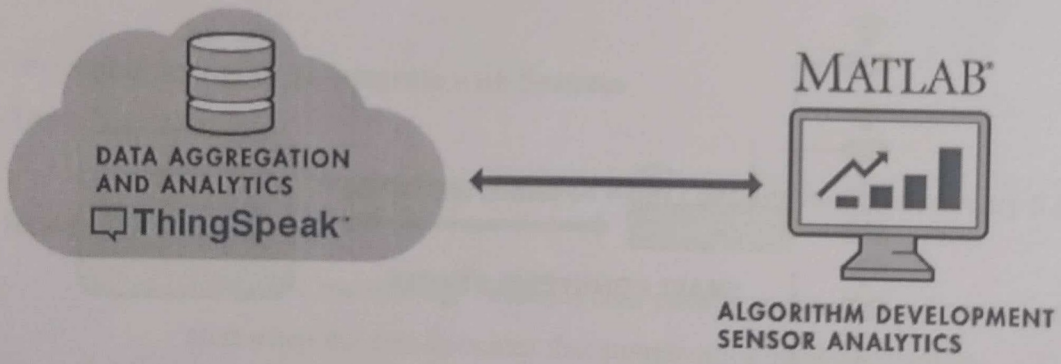
Connect Hardware to ThinkSpeak

- Internet- connected device with ThinkSpeak.
- Send data – use native libraries – Arduino , ESP – 8266, and Raspberry Pi.
  - REST API or MQTT API.
- Build integrations to ThingSpeak – setup easier.
  - ✓ LoRaWAN
  - ✓ Things Network
  - ✓ Senet
  - ✓ Libelium
  - ✓ Particle
- Simulink user – Simulink blocks – write data to ThingSpeak.



Access Data both Online and Offline

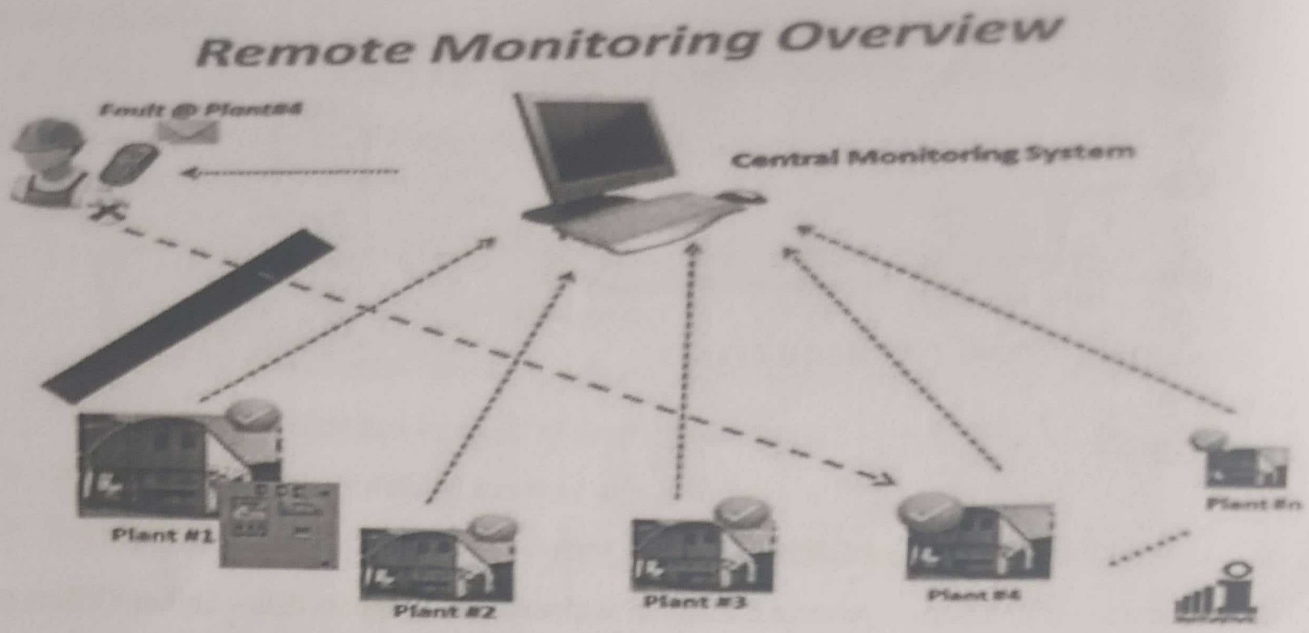
- Stores information – one central location in cloud – easily access data – Online/Offline analysis.
- Private data – protected – API key.
- ThingSpeak account – securely download the data stored in the cloud.
  - Read data – CSV/JSON formats – REST API call & API key
- Read data – ThingSpeak channel – MQTT topic.
- Import data – third party web services – NOAA , public utility data from local utility providers ,and stock and pricing data from financial providers.
- Data together – investigate correlations and develop predictive algorithms.



- MATLAB users – import data – MATLAB desktop environment – thingspeak Read function.

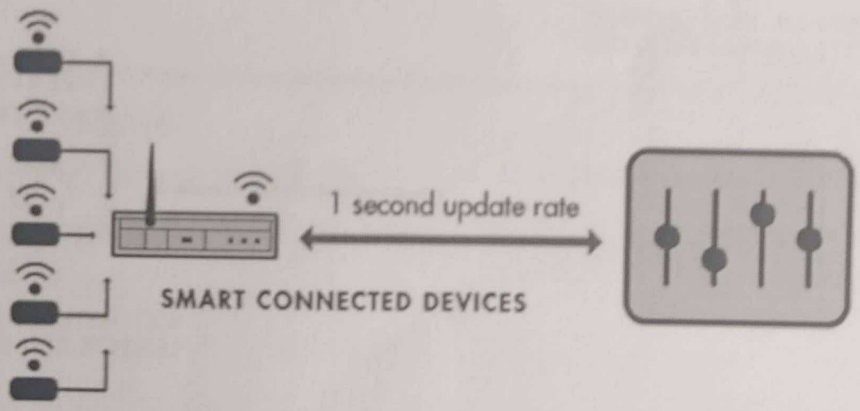
Remotely Visualize Sensor Data in Real Time

- ThingSpeak – charts data – remotely monitor devices / equipment – anywhere.
- View data – any web browser / mobile devices.
- Share read-only views of data – clients and colleagues.
- ThingSpeak – manage data – own front end – clients and customers – log in.



Control Devices Online

- Commercial ThingSpeak license – send data to ThingSpeak – once per second.
  - ✓ Enables near – real time monitoring of devices.
  - ✓ Allows to set up control loops from cloud.
- For applications – require faster response times – best practice – control loop at the edge closer to the hardware.



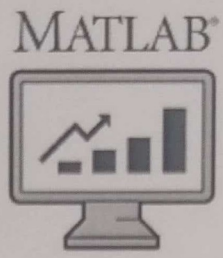
Perform Computations & Build Custom Visulaizations

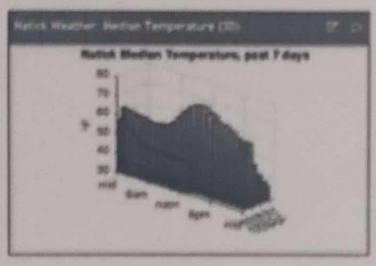
- MATLAB engine in ThingSpeak
  - ✓ Perform calibrations.
  - ✓ Develop analytics.
  - ✓ Transform IoT data.

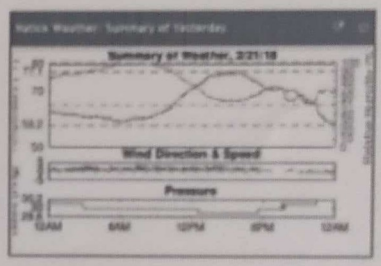
- ✓ Build custom charts.

➢ ThingSpeak license

- ✓ MATLAB calculations – 60 secs.
- ✓ Use MATLAB ToolBoxes.
  - Machine learning .
  - Signal processing.
  - System identification.







Create Streaming Analytics & Integrate with Systems

➢ **Time Control app :**

- ✓ schedule computation – run once a day / once an hour / once every 5 mins.

➢ **React App :**

- ✓ Condition monitoring – monitor data coming in from your devices and set up an alert when the data indicates that something is needing attention.

➢ Analyses – trigger events that push data from ThingSpeak to other web applications – Salesforce – REST APIs.