C VENKATA SUBBAIAH

HOD, DPET. OF CSE,

AITS-KADAPA

UNIT III

IoT Architecture and Protocols

Architecture Reference Model:

Introduction:

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

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

State of the art

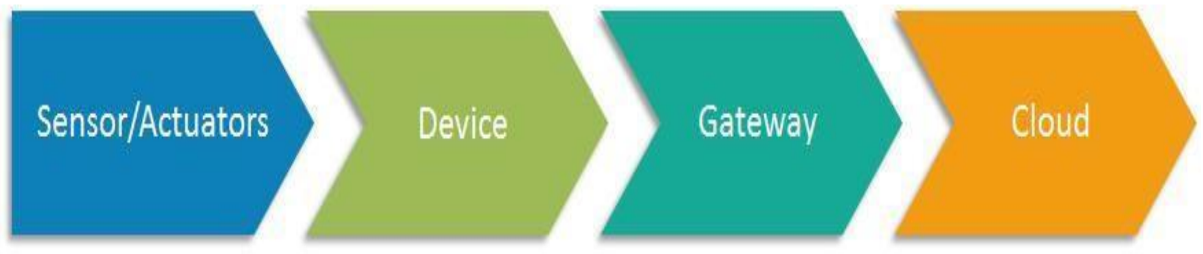
IoT architecture varies from solution to solution, based on the type of solution which we intend to build. IoT as a technology majorly consists of four main components, over which an architecture is framed.

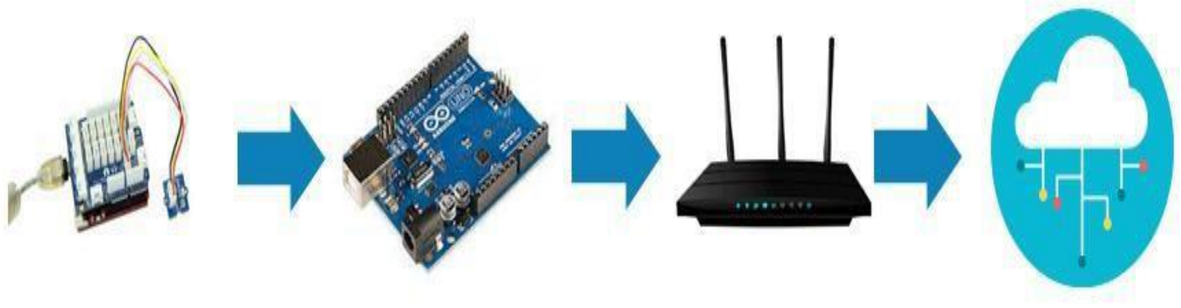
1) Sensors

2) Devices

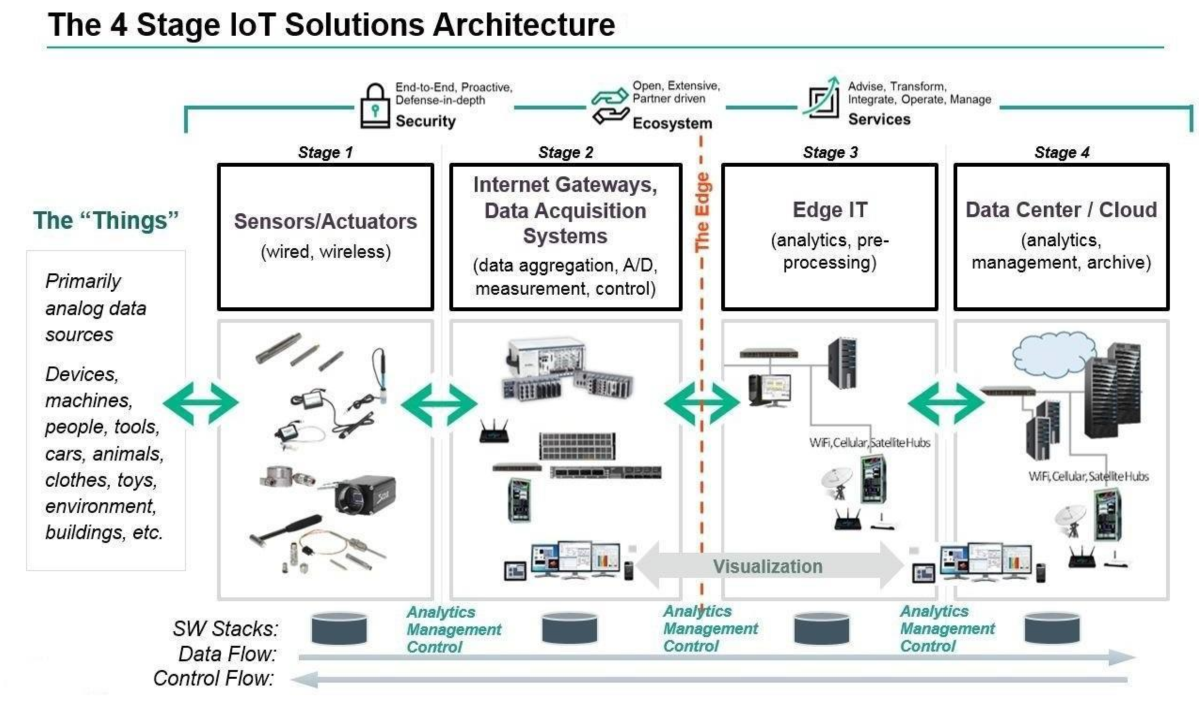
3) Gateway

4) Cloud





Stages of IoT Architecture:


The 4 Stage IoT Solutions Architecture
End-to-End, Proactive, Defense-in-depth
Security
Open, Extensive, Partner driven
Ecosystem
Advise, Transform, Integrate, Operate, Manage
Services
Stage 1
Stage 2
Stage 3
Stage 4
The "Things"
Primarily analog data sources
Devices, machines, people, tools, cars, animals, clothes, toys, environment, buildings, etc.
Sensors/Actuators
(wired, wireless)
Internet Gateways, Data Acquisition Systems
(data aggregation, A/D, measurement, control)
The Edge
Edge IT
(analytics, pre-processing)
Data Center / Cloud
(analytics, management, archive)
WiFi, Cellular, Satellite Hubs
WiFi, Cellular, Satellite Hubs
Visualization
SW Stacks:
Data Flow:
Control Flow:
Analytics Management Control
Analytics Management Control
Analytics Management Control

1. Sensors/actuators

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

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

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

2. The Internet gateways

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

For example, a pump might contain a half-dozen sensors and actuators that feed data into a data aggregation device that also digitizes the data. This device might be physically attached to the pump. An adjacent gateway device or server would then process the data and forward it to the Stage 3 or Stage 4 systems. Intelligent gateways can build on additional, basic gateway functionality by adding such capabilities as analytics, malware protection, and data management services. These systems enable the analysis of data streams in real time.

3. Edge IT

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

4.The data centre and cloud

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

Reference Model and architecture:

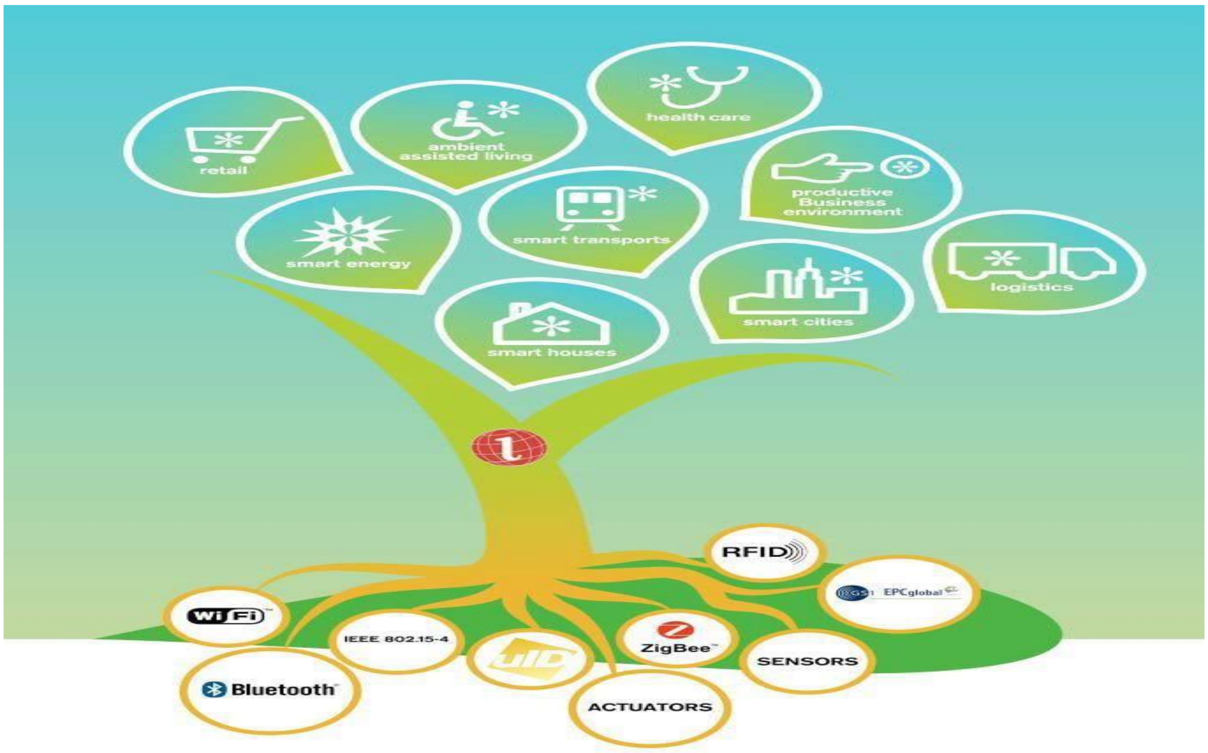
Reference Architecture that describes essential building blocks as well as design choices to deal with conflicting requirements regarding functionality, performance, deployment and security. Interfaces should be standardised, best practices in terms of functionality and information usage need to be provided.

The central choice of the IoT-A project was to base its work on the current state of the art, rather than using a clean-slate approach. Due to this choice, common traits are derived to form the base line of the Architectural Reference Model (ARM). This has the major advantage of ensuring backward compatibility of the model and also the adoption of established, working solutions to various aspects of the IoT. With the help of end users, organised into a stakeholder's group, new requirements for IoT have been collected and introduced in the main model building process. This work was conducted according to established architecture methodology.

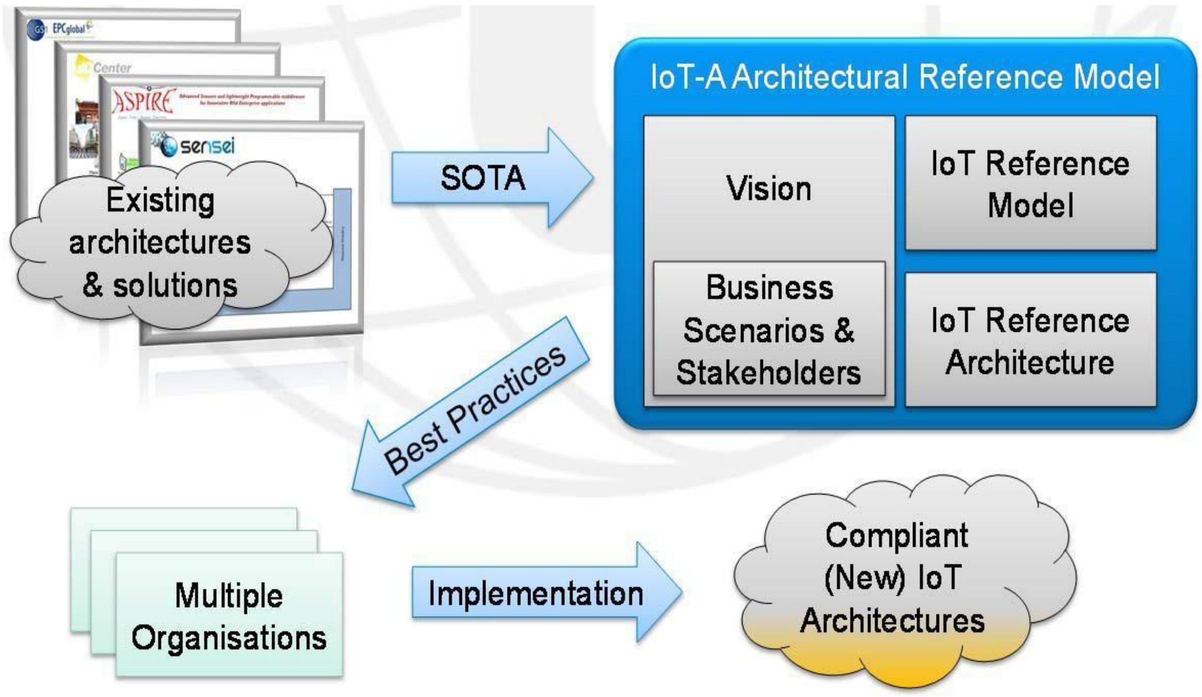
A Reference Architecture (RA) can be visualised as the —Matrix that eventually gives birth ideally to all concrete architectures. For establishing such a Matrix, based on a strong and exhaustive analysis of the State of the Art, we need to envisage the superset of all possible functionalities, mechanisms and protocols that can be used for building such concrete architecture and to show how interconnections could take place between selected ones (as no concrete system is likely to use all of the functional possibilities). Giving such a foundation along with a set of design-choices, based on the characterisation of the targeted system w.r.t. various dimensions (like distribution, security, real-time, semantics) it becomes possible for a system architect to select the protocols, functional components, architectural options, needed to build their IoT systems.

As any metaphoric representation, this tree does not claim to be fully consistent in its depiction; it should therefore not be interpreted too strictly. On the one hand, the roots of this tree are spanning across a selected set of communication protocols (6LoWPAN, Zigbee, IPv6,) and device technologies (sensors, actuators, tags,) while on the other hand the blossoms / leaves of the tree represent the whole set of IoT applications that can be built from the sap (i.e., data and information) coming from the roots. The trunk of the tree is of utmost importance here, as it represents the Architectural Reference Model (ARM). The ARM is the combination of the Reference Model and the Reference Architecture, the set of models, guidelines, best practices, views and perspectives that can be used for building fully

interoperable concrete IoT architectures and systems. In this tree, we aim at selecting a minimal set of interoperable technologies (the roots) and proposing the potentially necessary set of enablers or building blocks (the trunk) that enable the creation of a maximal set of interoperable IoT systems (the leaves).



IoT-A architectural reference model building blocks


Existing architectures & solutions
SOTA
IoT-A Architectural Reference Model
Vision
IoT Reference Model
Business Scenarios & Stakeholders
IoT Reference Architecture
Best Practices
Multiple Organisations
Implementation
Compliant (New) IoT Architectures

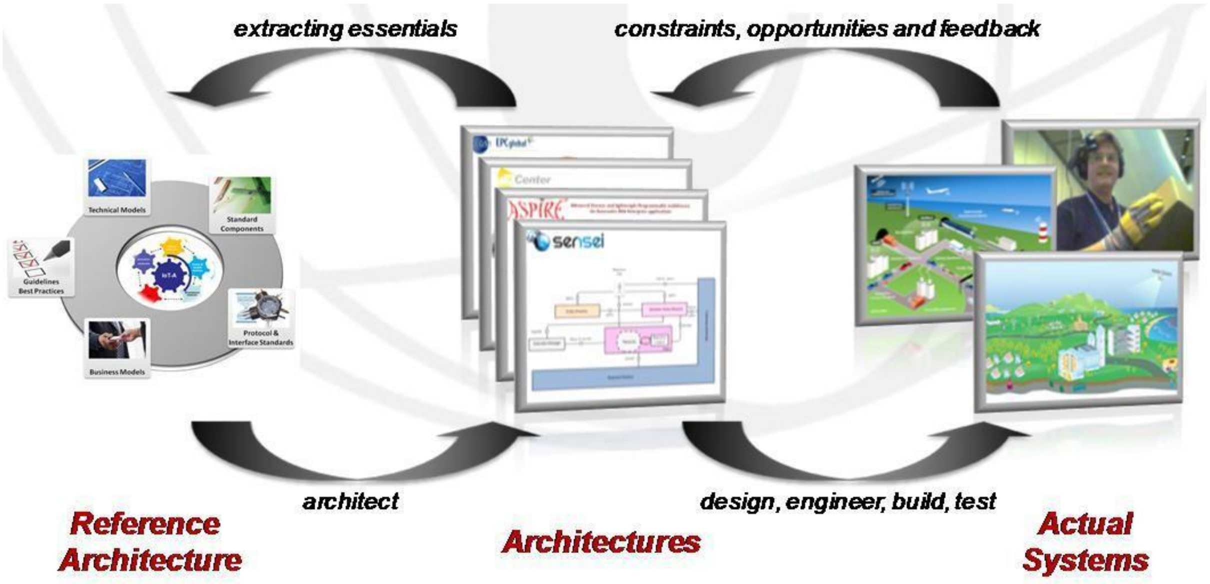
Starting with existing architectures and solutions, generic baseline requirements can be extracted and used as an input to the design. The IoT-A ARM consists of four parts:

The vision summarises the rationale for providing an architectural reference model for the IoT. At the same time, it discusses underlying assumptions, such as motivations. It also discusses how the architectural reference model can be used, the methodology applied to the architecture modelling, and the business scenarios and stakeholders addressed.

Business scenarios defined as requirements by stakeholders are the drivers of the architecture work. With the knowledge of businesses aspirations, a holistic view of IoT architectures can be derived.

The IoT Reference Model provides the highest abstraction level for the definition of the IoT-A Architectural Reference Model. It promotes a common understanding of the IoT domain. The description of the IoT Reference Model includes a general discourse on the IoT domain, an IoT Domain Model as a top-level description, an IoT Information Model explaining how IoT information is going to be modelled, and an IoT Communication Model in order to understand specifics about communication between many heterogeneous IoT devices and the Internet as a whole.

The IoT Reference Architecture is the reference for building compliant IoT architectures. As such, it provides views and perspectives on different architectural aspects that are of concern to stakeholders of the IoT. The terms' view and perspectives are used according to the general literature and standards the creation of the IoT Reference Architecture focuses on abstract sets of mechanisms rather than concrete application architectures. To organisations, an important aspect is the compliance of their technologies with standards and best practices, so that interoperability across organisations is ensured.

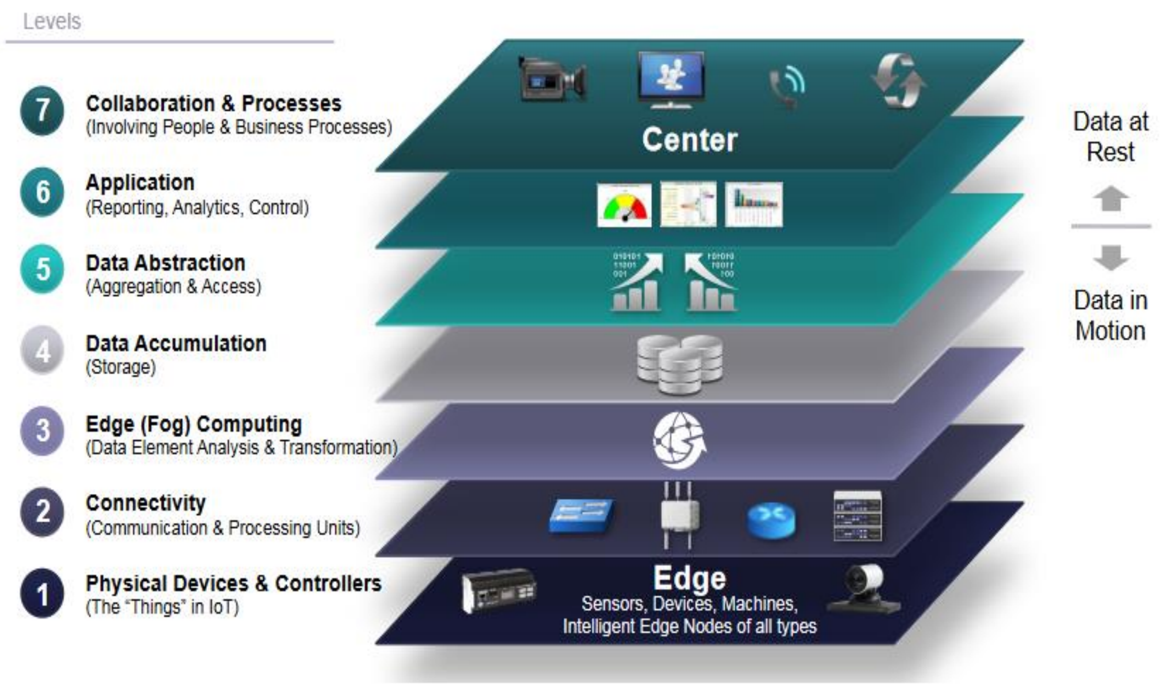


In an IoT system, data is generated by multiple kinds of devices, processed in different ways, transmitted to different locations, and acted upon by applications. The proposed IoT reference model is comprised of seven levels. Each level is defined with terminology that can be standardized to create a globally accepted frame of reference.

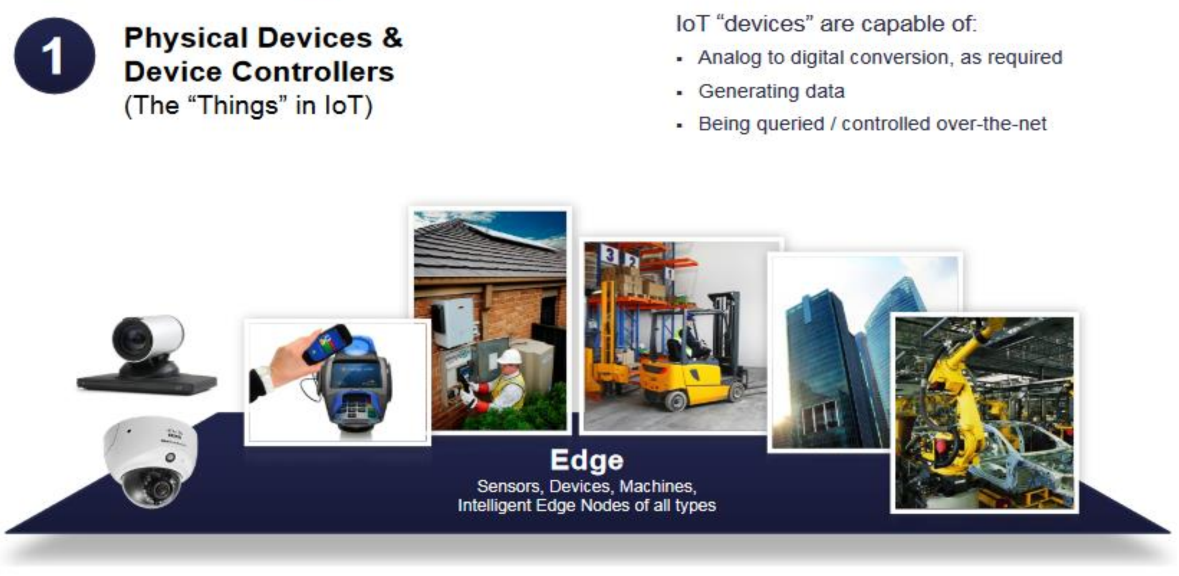
- Simplifies: It helps break down complex systems so that each part is more understandable. Clarifies: It provides additional information to precisely identify levels of the IoT and to establish common terminology.
- Identifies: It identifies where specific types of processing is optimized across different parts of the system.

- Standardizes: It provides a first step in enabling vendors to create IoT products that work with each other.
- Organizes: It makes the IoT real and approachable, instead of simply conceptual.

IoT reference Model:


Levels
7 Collaboration & Processes
(Involving People & Business Processes)
6 Application
(Reporting, Analytics, Control)
5 Data Abstraction
(Aggregation & Access)
4 Data Accumulation
(Storage)
3 Edge (Fog) Computing
(Data Element Analysis & Transformation)
2 Connectivity
(Communication & Processing Units)
1 Physical Devices & Controllers
(The "Things" in IoT)
Center
Edge
Sensors, Devices, Machines,
Intelligent Edge Nodes of all types
Data at Rest
Data in Motion

1. **Physical Devices and Controllers:**
   The IoT Reference Model starts with Level 1: physical devices and controllers that might control multiple devices. These are the "things" in the IoT, and they include a wide range of endpoint devices that send and receive information. Today, the list of devices is already extensive. It will become almost unlimited as more equipment is added to the IoT over time. Devices are diverse, and there are no rules about size, location, form factor, or origin. Some devices will be the size of a silicon chip. Some will be as large as vehicles. The IoT must support the entire range. Dozens or hundreds of equipment manufacturers will produce IoT devices. To simplify compatibility and support manufacturability, the IoT Reference Model generally describes the level of processing needed from Level 1 devices. Figure 2 describes basic capabilities for a device


1 Physical Devices & Device Controllers
(The "Things" in IoT)
IoT "devices" are capable of:
Analog to digital conversion, as required
Generating data
Being queried / controlled over-the-net
Edge
Sensors, Devices, Machines,
Intelligent Edge Nodes of all types

2. **Connectivity:**

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

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

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

- **Connectivity includes:**
  - Communicating with and between the Level devices
  - Reliable delivery across the network(s)
  - Implementation of various protocols
  - Switching and routing
  - Translation between protocols
  - Security at the network level(Self Learning) Networking Analytics

3. **Edge (Fog) Computing**

The functions of Level 3 are driven by the need to convert network data flows into information that is suitable for storage and higher-level processing at Level 4 (data accumulation). This means that Level 3 activities focus on high-volume data analysis and transformation. For example, a Level 1 sensor device might generate data samples multiple times per second, 24 hours a day, 365 days a year. A basic tenet of the IoT Reference Model is that the most intelligent system initiates information processing as early and as close to the edge of the network as possible. This is sometimes referred to as fog computing. Level 3 is where this occurs.

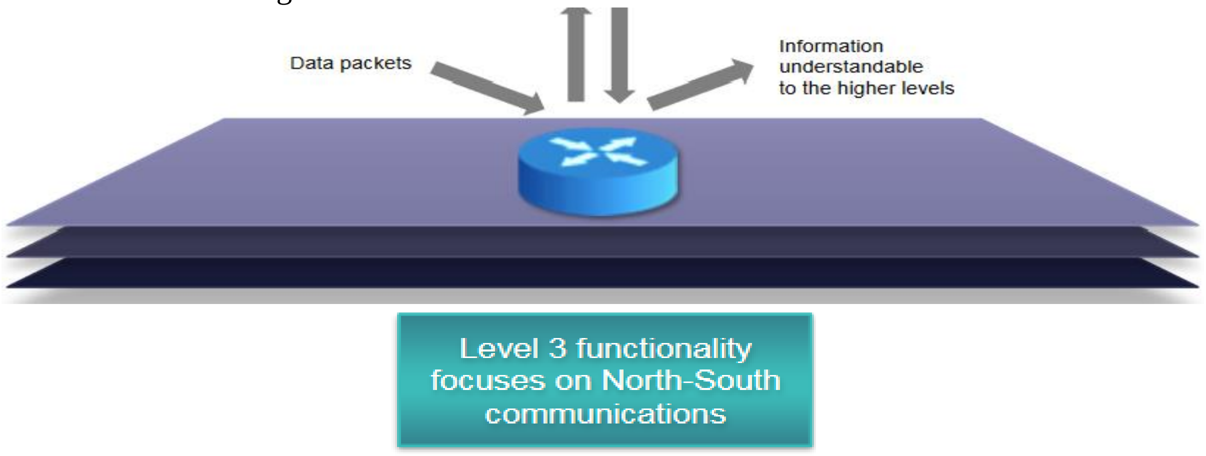
Given that data is usually submitted to the connectivity level (Level 2) networking equipment by devices in small units, Level 3 processing is performed on a packet-by-packet basis. This processing is limited, because there is only awareness of data units—

not “sessions” or “transactions.” Level 3 processing can encompass many examples, such as
Evaluation: Evaluating data for criteria as to whether it should be processed at a higher level

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

Include;

- Data filtering, clean up, aggregation
- Packet content inspection
- Combination of network and data level analytics
- Thresholding
- Event generation


Data packets
Information understandable to the higher levels
Level 3 functionality focuses on North-South communications

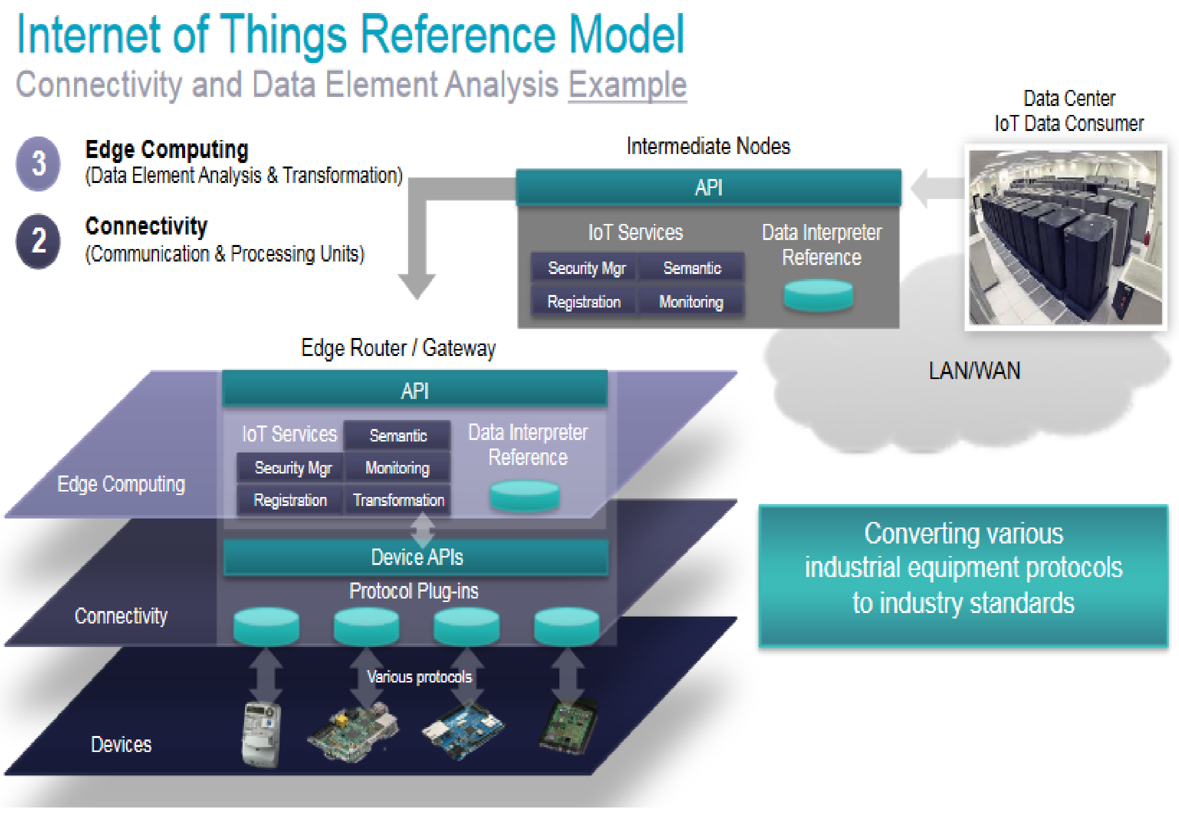


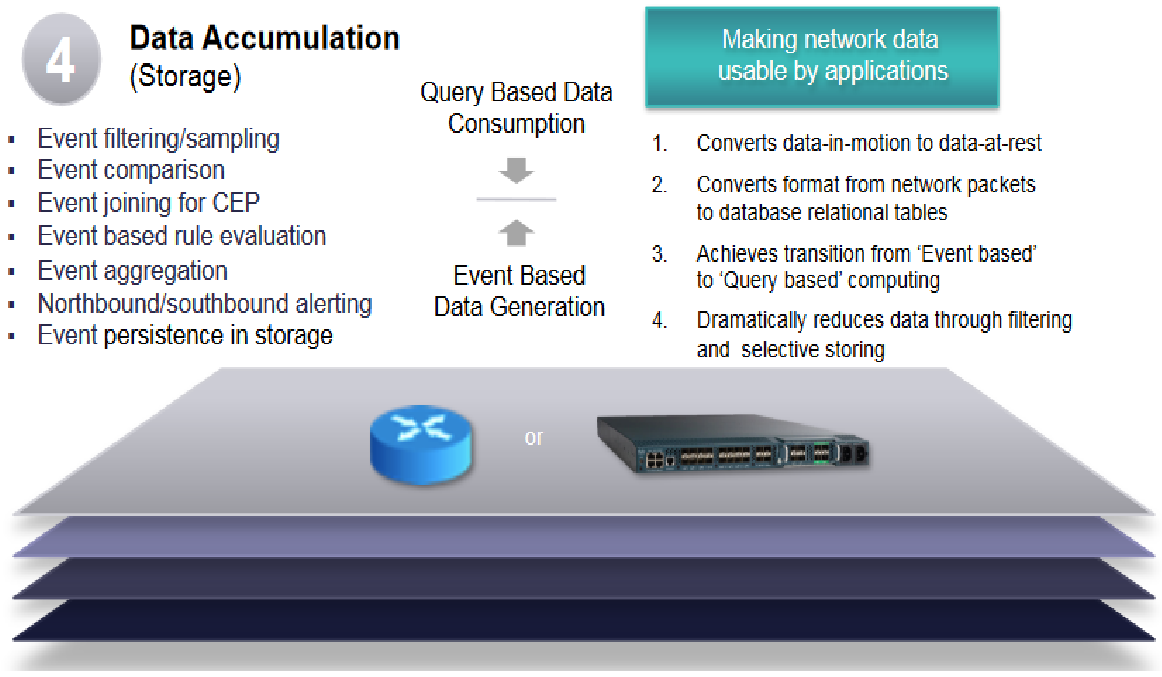
Fig : Level 2 and 3 Connectivity and Data Element Analysis Example

4. Data Accumulation

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

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

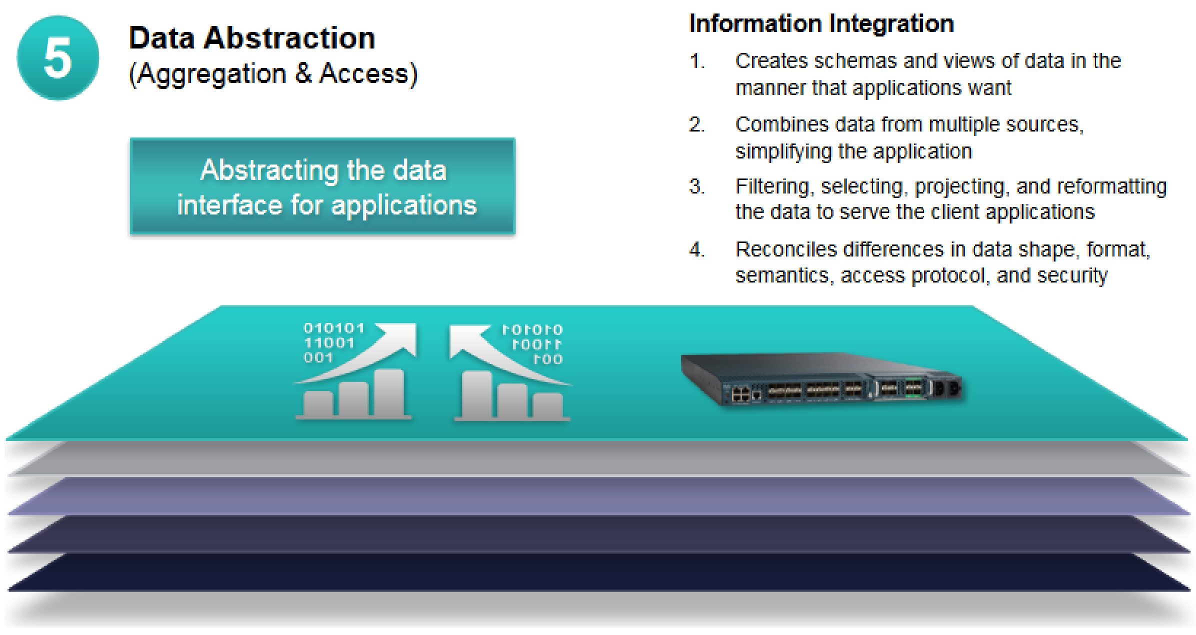
As Level 4 captures data and puts it at rest, it is now usable by applications on a non-real-time basis. Applications access the data when necessary. In short, Level 4 converts event-based data to query-based processing. This is a crucial step in bridging the differences between the real-time networking world and the non-real-time application world. Figure 6 summarizes the activities that occur at Level 4.


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
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
or

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

For these reasons, the data abstraction level must process many different things. These include:

- Reconciling multiple data formats from different sources
- Assuring consistent semantics of data across sources
- Confirming that data is complete to the higher-level application
- Consolidating data into one place (with ETL, ELT, or data replication) or providing access to multiple data
- stores through data virtualization
- Protecting data with appropriate authentication and authorization
- Normalizing or de-normalizing and indexing data to provide fast application access



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

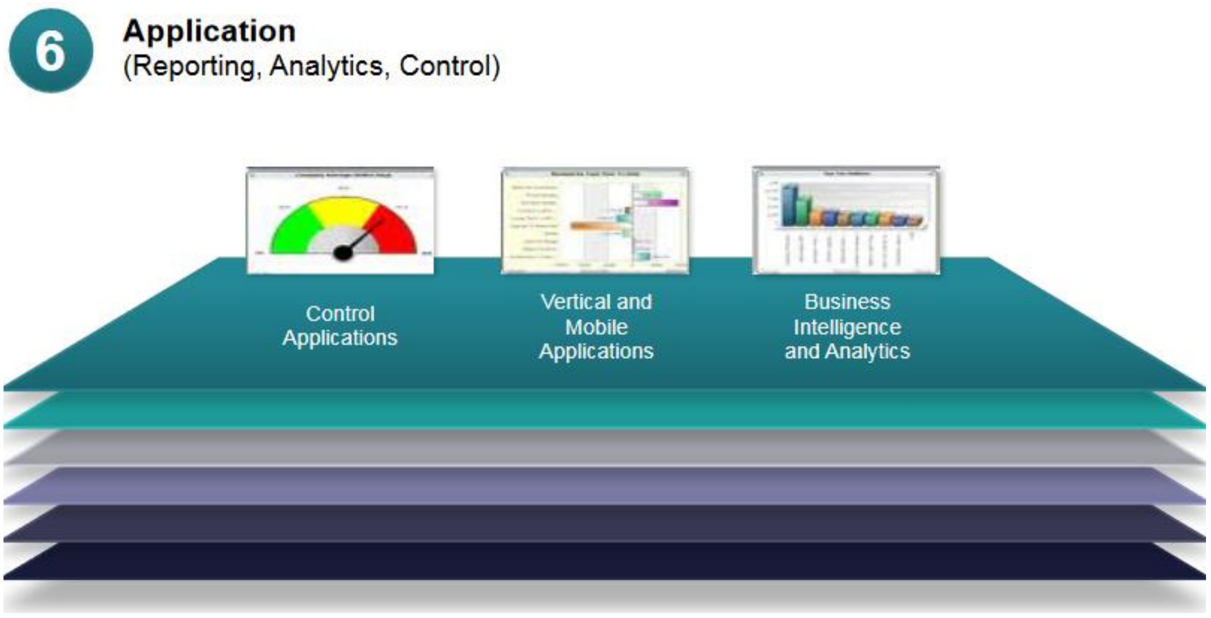
6. Application

Level 6 is the application level, where information interpretation occurs. Software at this level interacts with Level 5 and data at rest, so it does not have to operate at network speeds. The IoT Reference Model does not strictly define an application. Applications vary based on vertical markets, the nature of device data, and business needs. For example, some applications will focus on monitoring device data. Some will focus on controlling devices. Some will combine device and non-device data. Monitoring and control applications represent many different application models, programming patterns, and software stacks, leading to discussions of operating systems, mobility, application servers, hypervisors, multi-threading, multi-tenancy, etc. These topics are beyond the scope of the IoT Reference Model discussion. Suffice it to say that application complexity will vary widely.

Examples include:

- Mission-critical business applications, such as generalized ERP or specialized industry solutions
- Mobile applications that handle simple interactions
- Business intelligence reports, where the application is the BI server
- Analytic applications that interpret data for business decisions
- System management/control center applications that control the IoT system itself and don't act on the data produced by it

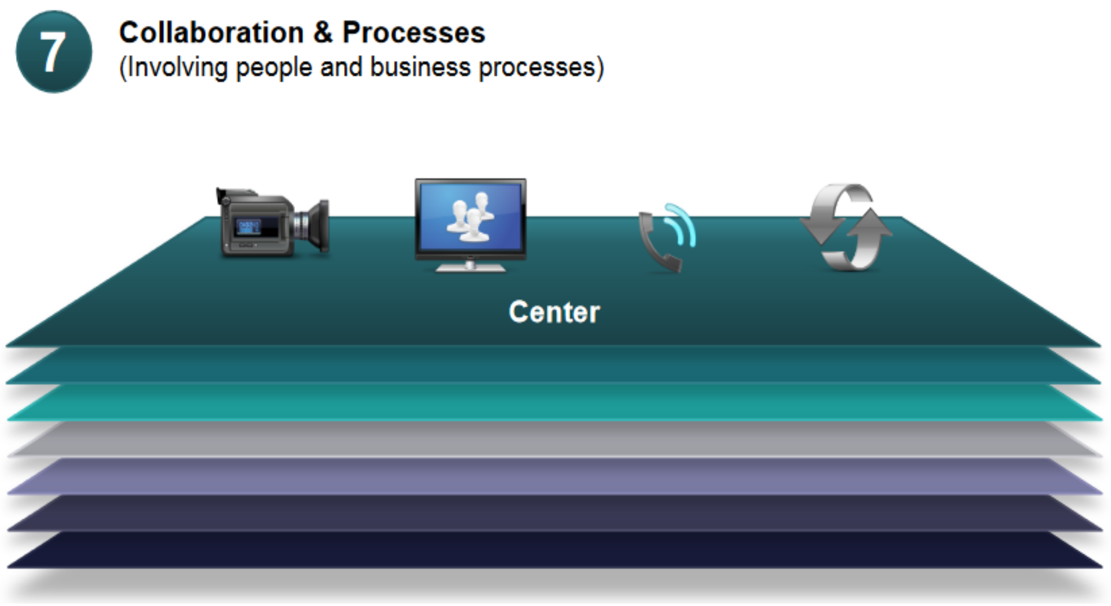
If Levels 1-5 are architected properly, the amount of work required by Level 6 will be reduced. If Level 6 is designed properly, users will be able to do their jobs better. Figure 8 depicts Level6


6
Application
(Reporting, Analytics, Control)
Control Applications
Vertical and Mobile Applications
Business Intelligence and Analytics

7. Collaboration and Processes

One of the main distinctions between the Internet of Things (IoT) and IoT is that IoT includes people and processes. This difference becomes particularly clear at Level 7: Collaboration and Processes. The IoT system, and the information it creates, is of little value unless it yields action, which often requires people and processes. Applications execute business logic to empower people. People use applications and associated data for their specific needs. Often, multiple people use the same application for a range of different purposes. So the objective is not the application—it is to empower people to do their work better. Applications (Level 6) give business people the right data, at the right time, so they can do the right thing.

But frequently, the action needed requires more than one person. People must be able to communicate and collaborate, sometimes using the traditional Internet, to make the IoT useful. Communication and collaboration often require multiple steps. And it usually transcends multiple applications. This is why Level 7, as shown in Figure 9, represents a higher level than a single application.

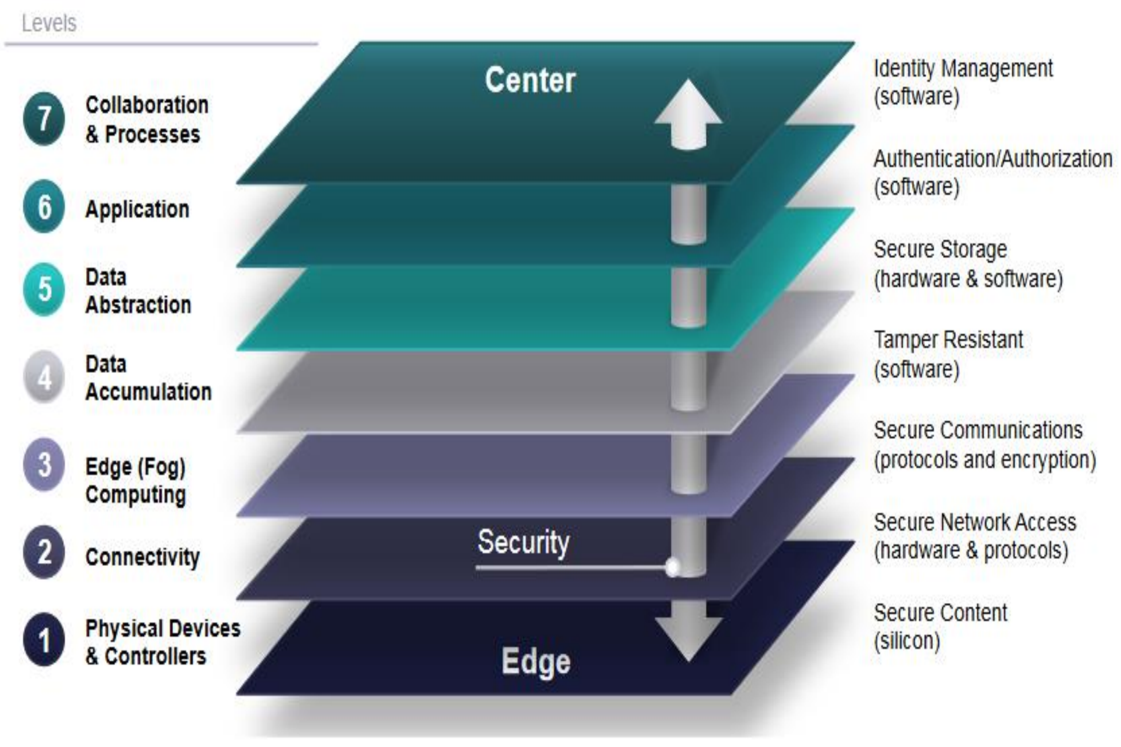

7
Collaboration & Processes
(Involving people and business processes)
Center

Security in the IoT:

Discussions of security for each level and for the movement of data between levels could fill a multitude of papers. For the purpose of the IoT Reference Model, security measures must:

- Secure each device or system
- Provide security for all processes at each level
- Secure movement and communication between each level, whether north- or south-bound

As shown in Figure 10, security must pervade the entire mode


Levels
7 Collaboration & Processes
6 Application
5 Data Abstraction
4 Data Accumulation
3 Edge (Fog) Computing
2 Connectivity
1 Physical Devices & Controllers
Center
Security
Edge
Identity Management (software)
Authentication/Authorization (software)
Secure Storage (hardware & software)
Tamper Resistant (software)
Secure Communications (protocols and encryption)
Secure Network Access (hardware & protocols)
Secure Content (silicon)

Protocols:

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

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

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

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

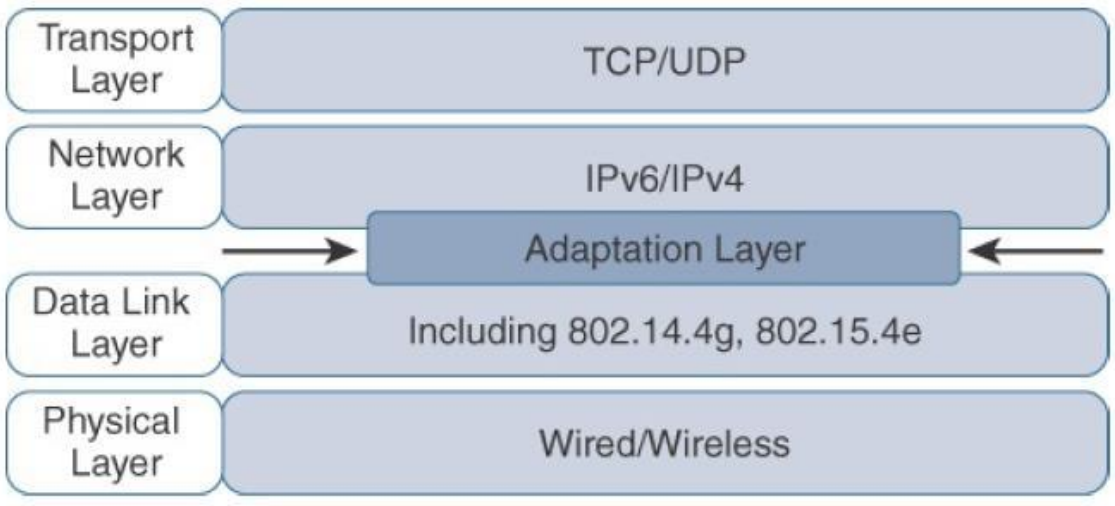


Figure : Optimizing IP for IoT Using an Adaptation Layer

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

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

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

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

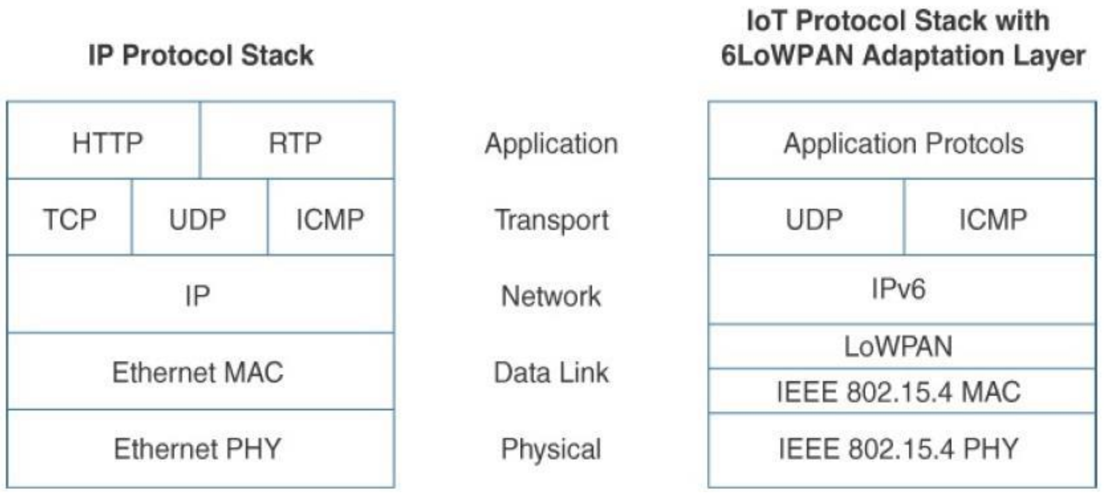


Fig: Comparison of an IoT Protocol Stack Utilizing 6LoWPAN and an IP Protocol Stack

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

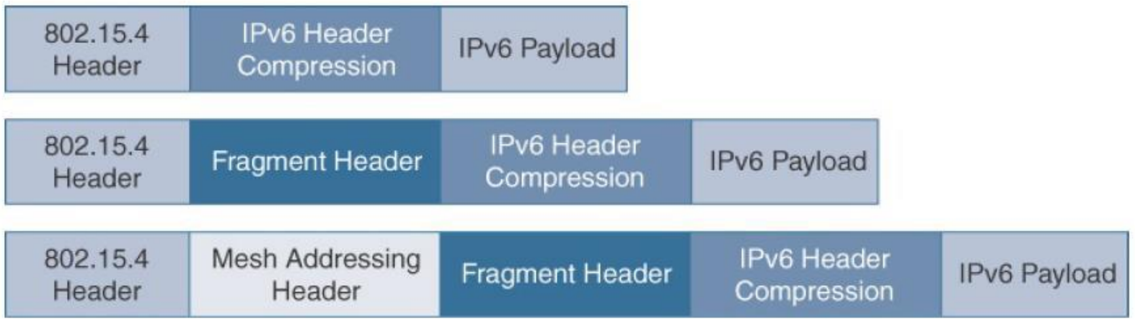


Figure :6LoWPAN Header Stack

Header Compression

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

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

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

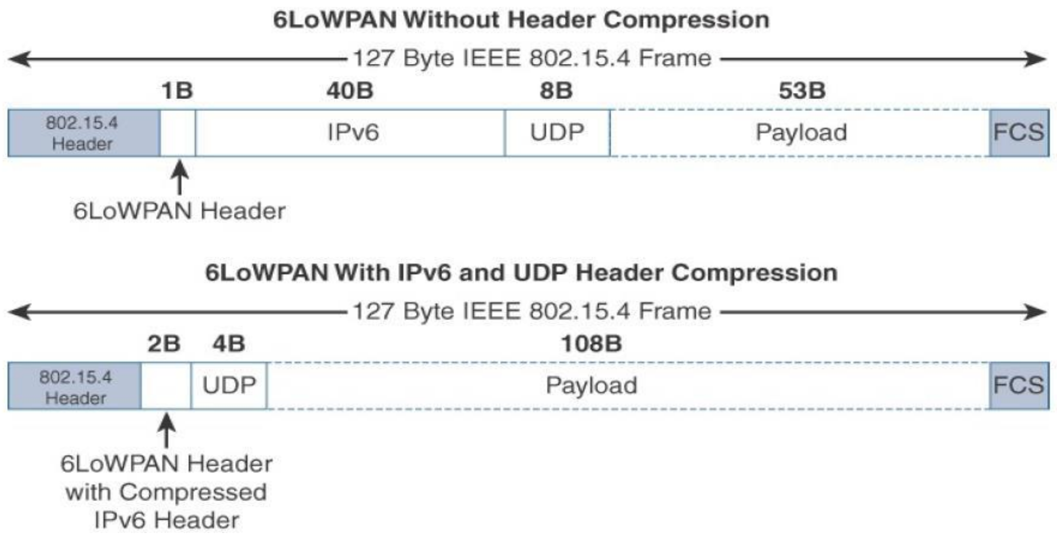

6LoWPAN Without Header Compression
127 Byte IEEE 802.15.4 Frame
1B
40B
8B
53B
802.15.4 Header
IPv6
UDP
Payload
FCS
6LoWPAN Header
6LoWPAN With IPv6 and UDP Header Compression
127 Byte IEEE 802.15.4 Frame
2B
4B
108B
802.15.4 Header
UDP
Payload
FCS
6LoWPAN Header with Compressed IPv6 Header

Figure : 6LoWPAN Header Compression

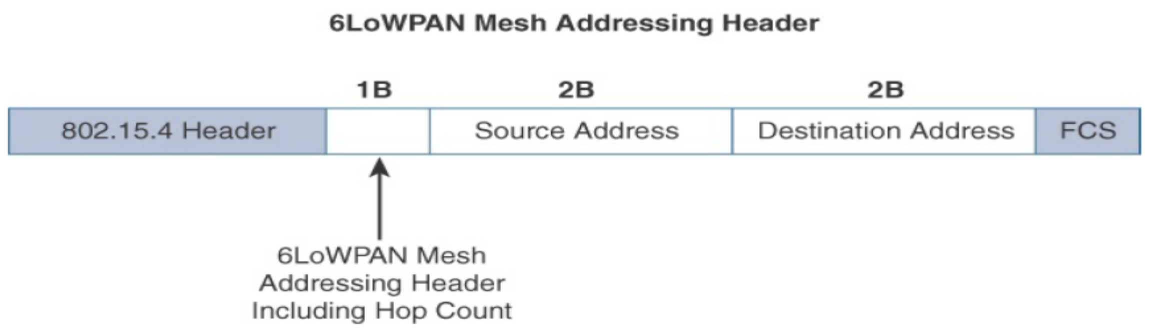
At the top of Figure above, you see a 6LoWPAN frame without any header compression enabled: The full 40- byte IPv6 header and 8-byte UDP header are visible. The 6LoWPAN header is only a single byte in this case. Notice that uncompressed IPv6 and UDP headers leave only 53 bytes of data payload out of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

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

Mesh Addressing

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

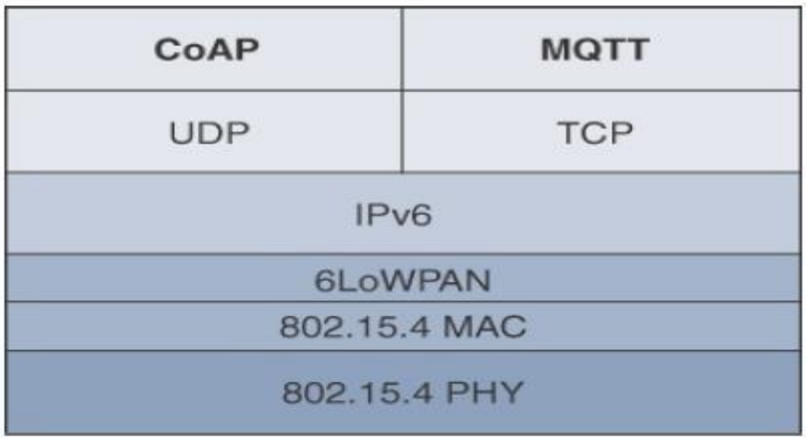
The Source Address and Destination Address fields for mesh addressing are IEEE 802.15.4 addresses indicating the endpoints of an IP hop. Figure below details the 6LoWPAN mesh addressing header fields.


6LoWPAN Mesh Addressing Header
1B
2B
2B
802.15.4 Header
Source Address
Destination Address
FCS
6LoWPAN Mesh Addressing Header Including Hop Count

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

IoT Application Layer Protocols (COAP ANS MQTT)

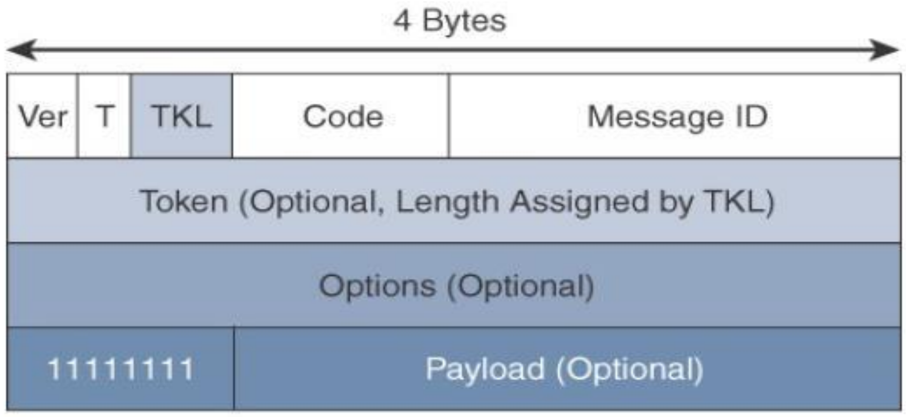
When considering constrained networks and/or a large-scale deployment of constrained nodes, verbose web-based and data model protocols, may be too heavy for IoT applications. To address this problem, the IoT industry is working on new lightweight protocols that are better suited to large numbers of constrained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



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

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

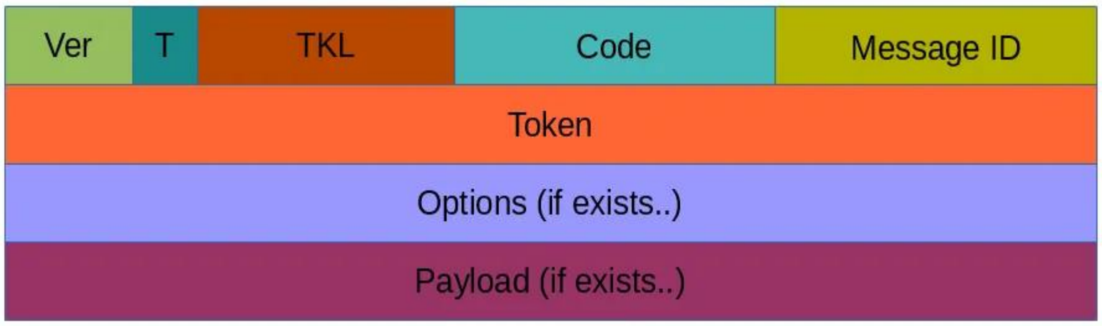
   From a formatting perspective, a CoAP message is composed of a short fixed-length Header field (4 bytes), a variable-length but mandatory Token field (0–8 bytes), Options fields if necessary, and the Payload field. Figure below details the CoAP message format, which delivers low overhead while decreasing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



CoAP Message Format

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

CoAP Message Fields

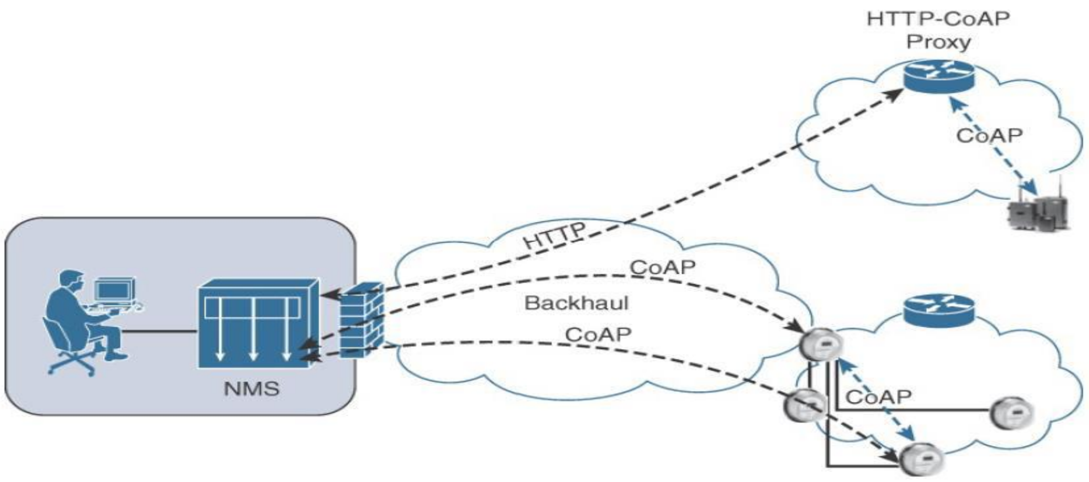


- **Ver**: It is a 2 bit unsigned integer indicating the version
- **T**: it is a 2 bit unsigned integer indicating the message type: 0 confirmable, 1 non-confirmable
- **TKL**: Token Length is the token 4 bit length
- **Code**: It is the code response (8 bit length)
- **Message ID**: It is the message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

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

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

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

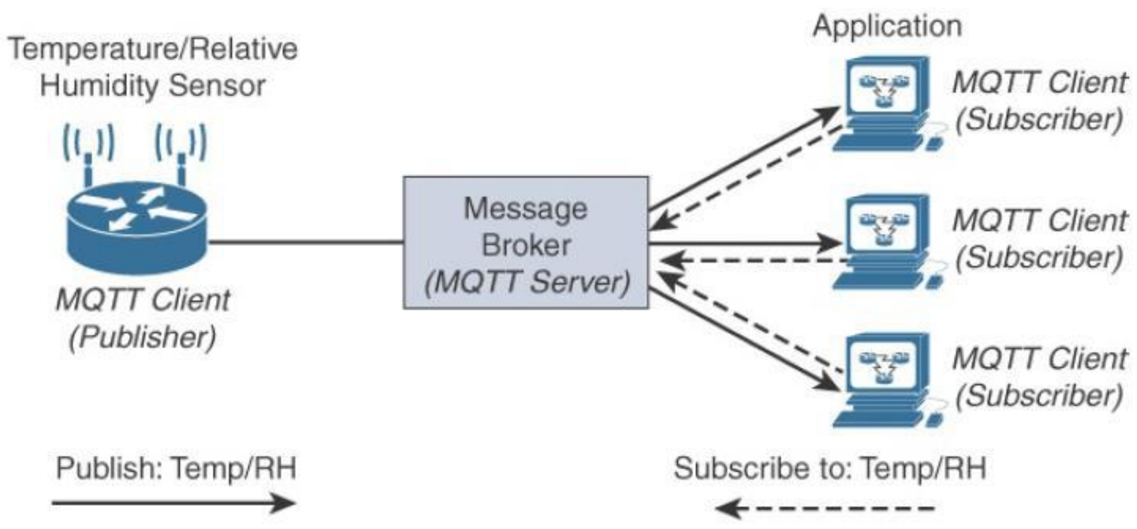


Just like HTTP, CoAP is based on the REST architecture, but with a “thing” acting as both the client and the server. Through the exchange of asynchronous messages, a client requests an action via a method code on a server resource. A uniform resource identifier (URI) localized on the server identifies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

Message Queuing Telemetry Transport (MQTT):

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

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

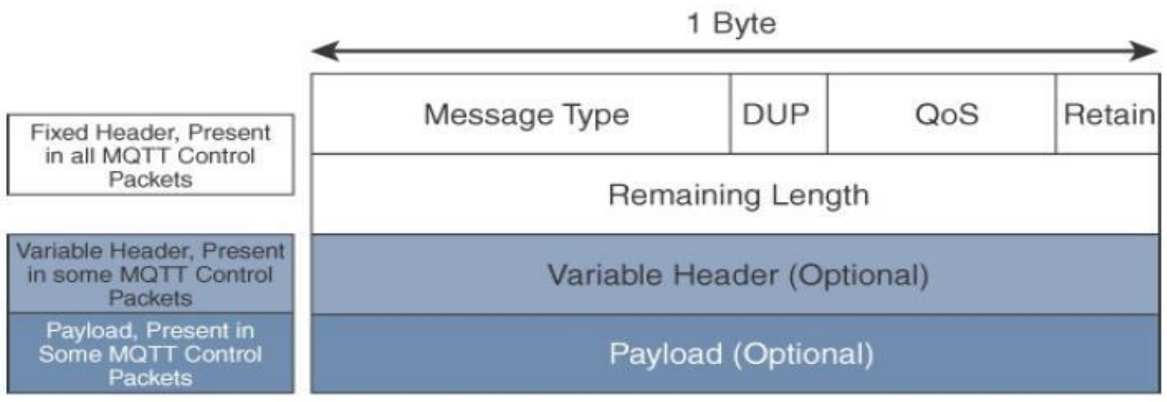


An MQTT client can act as a publisher to send data (or resource information) to an MQTT server acting as an MQTT message broker. In the example illustrated in Figure 2.22, the MQTT client on the left side is a temperature (Temp) and relative humidity (RH) sensor that publishes its Temp/RH data. The MQTT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

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

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

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


1 Byte
Fixed Header, Present in all MQTT Control Packets
Message Type
DUP
QoS
Retain
Remaining Length
Variable Header, Present in some MQTT Control Packets
Variable Header (Optional)
Payload, Present in Some MQTT Control Packets
Payload (Optional)

MQTT Message Format

Compared to the CoAP message format, MQTT contains a smaller header of 2 bytes compared to 4 bytes for CoAP. The first MQTT field in the header is Message Type, which identifies the kind of MQTT packet within a message. Fourteen different types of control packets are specified in MQTT version 3.1.1. Each of them has a unique value that is coded into the Message Type field. Note that values 0 and 15 are reserved. MQTT message types are summarized in Table 2.5.

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

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

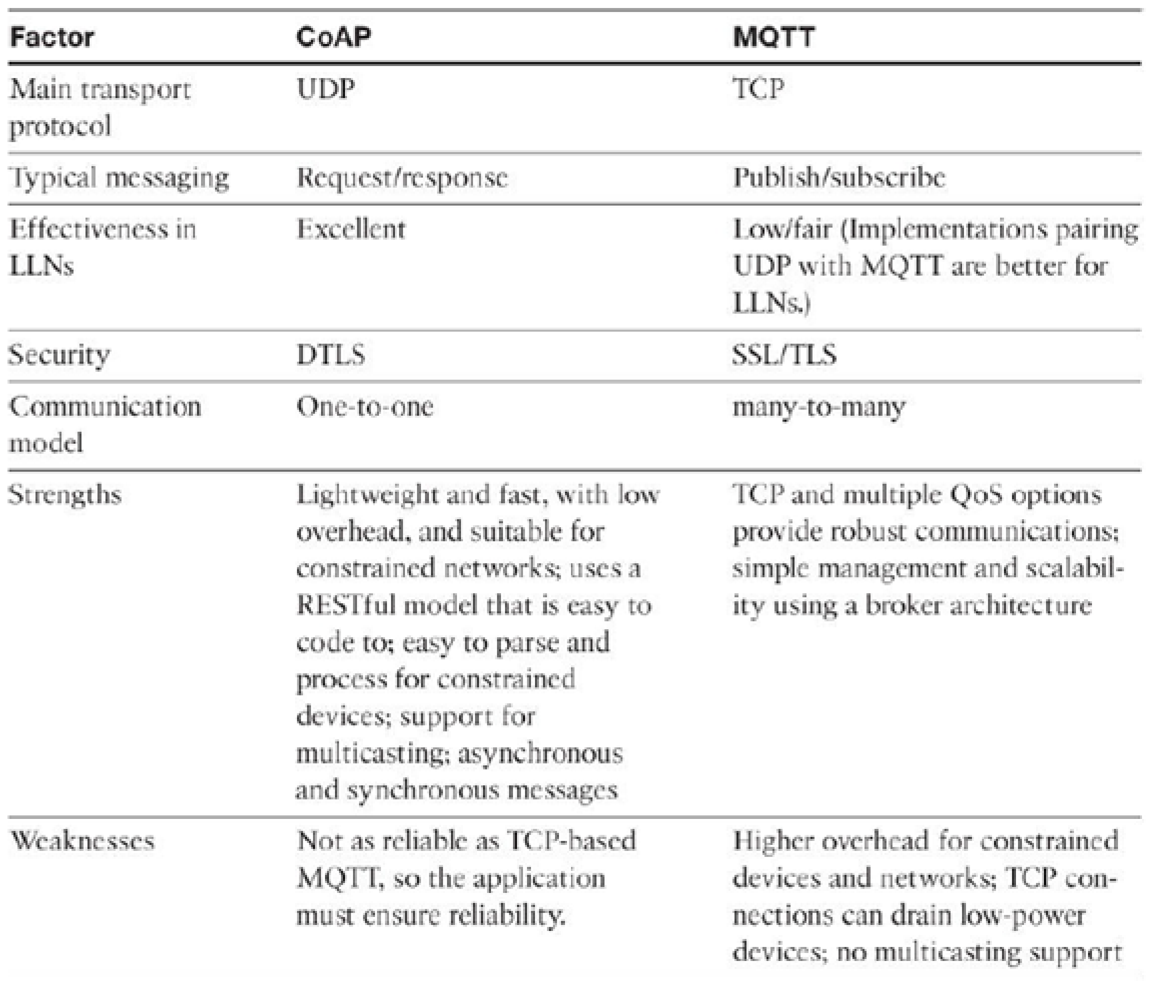
The last mandatory field in the MQTT message header is Remaining Length. This field specifies the number of bytes in the MQTT packet following this field.

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

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

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

Comparison of CoAP and MQTT

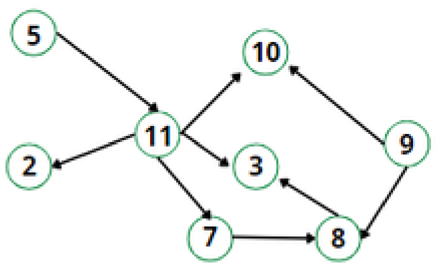


Factor	CoAP	MQTT
Main transport protocol	UDP	TCP
Typical messaging	Request/response	Publish/subscribe
Effectiveness in LLNs	Excellent	Low/fair (Implementations pairing UDP with MQTT are better for LLNs.)
Security	DTLS	SSL/TLS
Communication model	One-to-one	many-to-many
Strengths	Lightweight and fast,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

RPL:(RPL (IPv6 Routing protocol):

RPL stands for Routing Protocol for Low Power and Lossy Networks for heterogeneous traffic networks. It is a routing protocol for Wireless Networks. This protocol is based on the same standard as by Zigbee and 6 Lowpan is IEEE 802.15.4 It holds both many-to-one and one-to-one communication. It is a Distance Vector Routing Protocol that creates a tree-like routing topology called the Destination Oriented Directed Acyclic Graph (DODAG), rooted towards one or more nodes called the root node or sink node.

The **Directed Acyclic Graphs** (DAGs) are created based on user-specified specific Objective Function (OF). The OF defines the method to find the best-optimized route among the number of sensor devices.



The IETF chartered the ROLL (Routing Over Low power and Lossy networks) working group to evaluate all three routing protocols and determine the needs and requirements for developing a routing solution for IP smart objects. After the study of various use cases and a survey of existing protocols, the consensus was that a new routing protocol should be developed for IP smart objects, given the characteristics and requirements of the constrained network. This new Distance Vector Routing Protocol was named the IPv6 Routing Protocol for Low power and Lossy networks(RPL). The RPL specification was published as RFC 6550 by the ROLL working group.

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

Modes of RPL:

This protocol defines two modes:

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

Non-Storing mode: Only the border router(s) of the RPL domain contain(s) the full routing table. All other nodes in the domain maintain their list of parents only and use this as a list of default routes towards the border router. The abbreviated routing table saves memory space and CPU. When communicating in non-storing mode, a node always forwards its packet to the border router, which knows how to ultimately reach the final destination.

RPL is based on the concept of a Directed Acyclic Graph (DAG). A DAG is Directed Graph where no cycle exists. This means that from any vertex or point in the graph, we cannot follow an edge or a line back to this same point. All of the edges are arranged in a path oriented toward and terminating at one or more root nodes.

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

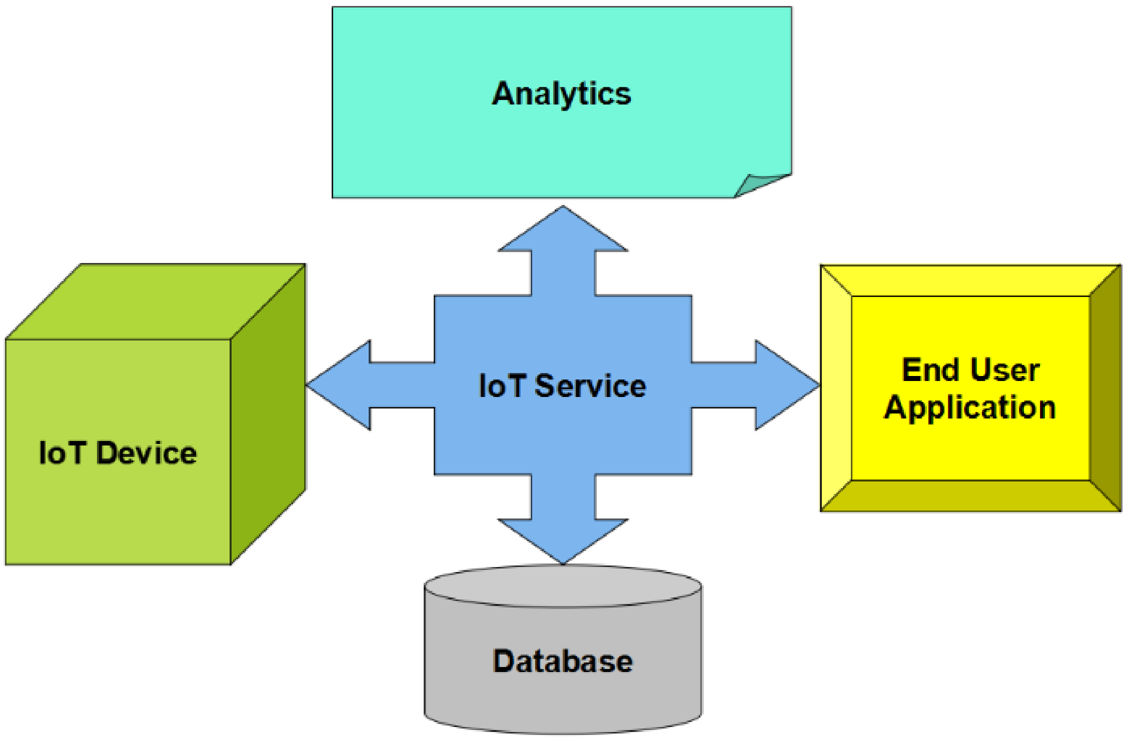
Implementation of RPL Protocol:

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

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

IOT FRAMEWORK-THING SPEAK:

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

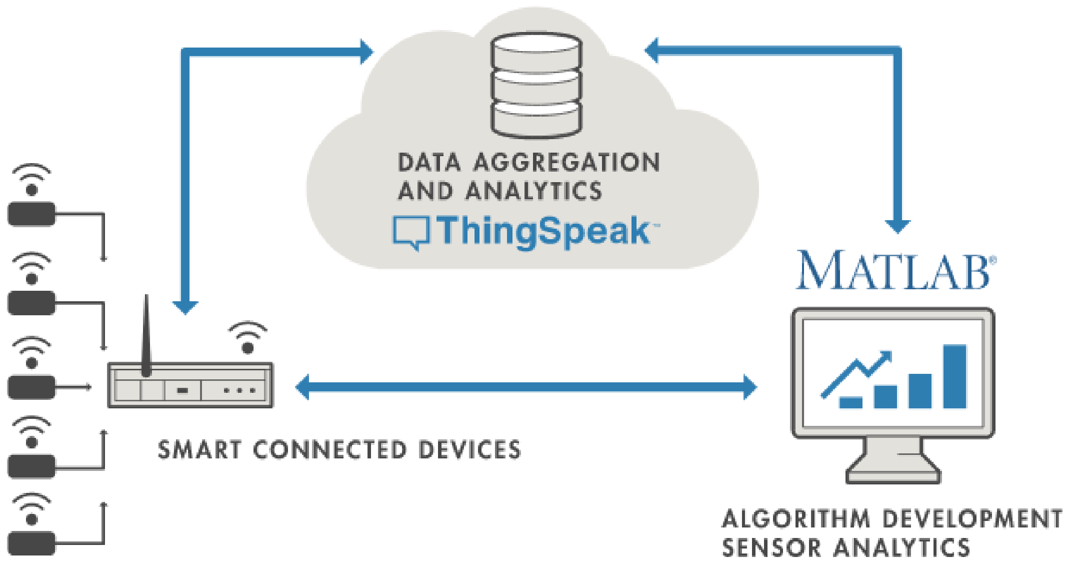


What is Thing Speak:

Thing Speak is a platform providing various services exclusively targeted for building IoT applications. It offers the capabilities of real-time data collection, visualizing the collected data in the form of charts, ability to create plugins and apps for collaborating with web services, social network and other APIs. We will consider each of these features in detail below.
The core element of Thing Speak is a 'Thing Speak Channel'. A channel stores the data that we
send to Thing Speak and comprises of the below elements:

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

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



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

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

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