SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF ELECTRICAL AND ELECTRONICS

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

UNIT – I – Overview of M2M – SECA7019

UNIT 1 OVERVIEW OF M2M

Introduction to communication needs, Terminology - Machine to Machine to Communication (M2M), Internet of Things (IoT), Web of Things (WoT), M2M vsIoT, IoTvs. WoT, Communication Standards

1.1COMMUNICATION NEEDS:

M2M (Machine-to-Machine) has come of age. It has been almost a decade since the idea of expanding the scope of entities connected to "the network" (wireless, wireline; private, public) beyond mere humans and their preferred communication gadgets has emerged around the notions of the "Internet of Things" (IoT), the "Internet of Objects" or M2M. The initial vision was that of a myriad of new devices, largely unnoticed by humans, working together to expand the footprint of end-user services. This will create new ways to care for safety or comfort, optimizing a variety of goods-delivery mechanisms, enabling efficient tracking of people or vehicles, and at the same time creating new systems and generating new value.

As with every vision, it has taken time to materialize. Early efforts concentrated on refining the initial vision by testing new business models, developing point solutions to test feasibility, and forecasting the impact of insufficient interoperability. Over the past few years, the realization that there are new viable sources of demand that can be met and monetized has created the push for a joint effort by industry to turn a patchwork of standalone elements and solutions into a coherent "system of systems", gradually turning the focus from the "what" to the "how" and developing the appropriate technologies and standards.

A cornerstone to this connectivity landscape is and will be machine-to-machine (M2M). M2M generally refers to information and communications technologies (ICT) able to measure, deliver, digest, and react upon information in an autonomous fashion, i.e., with no or really minimal human interaction during deployment, configuration, operation, and maintenance phases. Flagship examples of M2M technologies are telemetry readings of status of oil and brakes of cars on the move, health state measurements of blood pressure and heartbeat of the elderly, monitoring of corrosion state of oil or gas pipelines, occupancy

measurements of parking in cities, remote metering of water consumption, and real-time monitoring of critical parts of a piece of machinery. While machines do not excel humans in writing poetry, they are definitely industry favorites when it comes to (i) repetitive jobs, like delivering water meter data once a day, and (ii) time-critical jobs with decisions taken within a few milliseconds based on the input of an enormous amount of data, like the real-time monitoring of rotating machinery parts. M2M is all about big data, notably about (i) real-time, (ii) scalable, (iii) ubiquitous, (iv) reliable, and (v) heterogeneous big data, and thus associated opportunities. These technical properties are instrumental to the ecosystem:

- Indeed, real time allows making optimal and timely decisions based on a large amount of prior collected historical data. The trend is to move away from decision making based on long-term averages to decisions based on real-time or short-term averages, making a real difference to the large amount of nonergodic industrial processes.
- Scalability implies that all involved stakeholders, i.e., technology providers, service companies, and finally end user in a given industry vertical, can scale up the use and deployment where and when needed without jeopardizing prior deployments. Wireless is instrumental when it comes to scalability!
- Ubiquitous deployment and use are important since it allows reaching this critical mass required to make technologies survive long term. A sparse or punctual use of specific M2Mtechnologies makes sense in the bootstrapping phase of the market but is unsustainable in the long term.
- Reliability, often overlooked, means the industry customer gets sensor readings that can be relied on because the system had been calibrated, tested, and certified. A prominent example is the vertical of urban parking, where Google's crowdsourced platform to get parking occupancy had been discontinued since stakeholders in this space preferred to get reliable occupancy messages 24/7 from certified sensors installed at each parking spot, rather than unreliable crowdsourced data, even when offered for free.
- Finally, the big data dream can only be materialized if heterogeneous data, i.e., data from different verticals, are combined to give unprecedented insights into

behavior, trends, and opportunities and then also act upon them. In fact, when referring to “big” in big data, it is less about quantity of data (i.e., large volumes of terabytes of information) but rather the quality of data (i.e., different information streams giving a new comprehension of the field, where one stream could be only a single bit).

The data being delivered byM2Msystems are often referred to as the oil of the twentyfirst century. It might be coincidental but the M2M data flow resembles the oil processing flow in that it is composed of data (i) upstream, (ii) mash-up, and (iii) downstream:

Upstream: Here, data are being collected from sensors in the field that deliver the data wirelessly to a gateway via single-hop or mesh networking technologies. The data traverse the Internet and reach dedicated servers or cloud platforms, where they are stored and profiled for further analytics.

Downstream: Here, business intelligence is applied to the data and intelligent decisions are taken. These are pushed back down to the customer, by either controlling some actuators, displaying some information/alerts in control/service platforms, or simply informing users about issues pertaining to this specific M2M application. Human–computer interfaces (HCI) or even computer–computer interfaces (beyond the typical APIs) will be instrumental in ensuring the offtake of M2M technologies.

Mash-up: The data processing and business intelligent platforms are likely the most important constituent of emerging M2M systems. Data flows from machines, humans, and the Internet in general will feed intelligent analytics engines, which are able to find trends, optimize processes, and thus make the industry vertical more efficient and effective.These three constituents are depicted in Figure 1.1

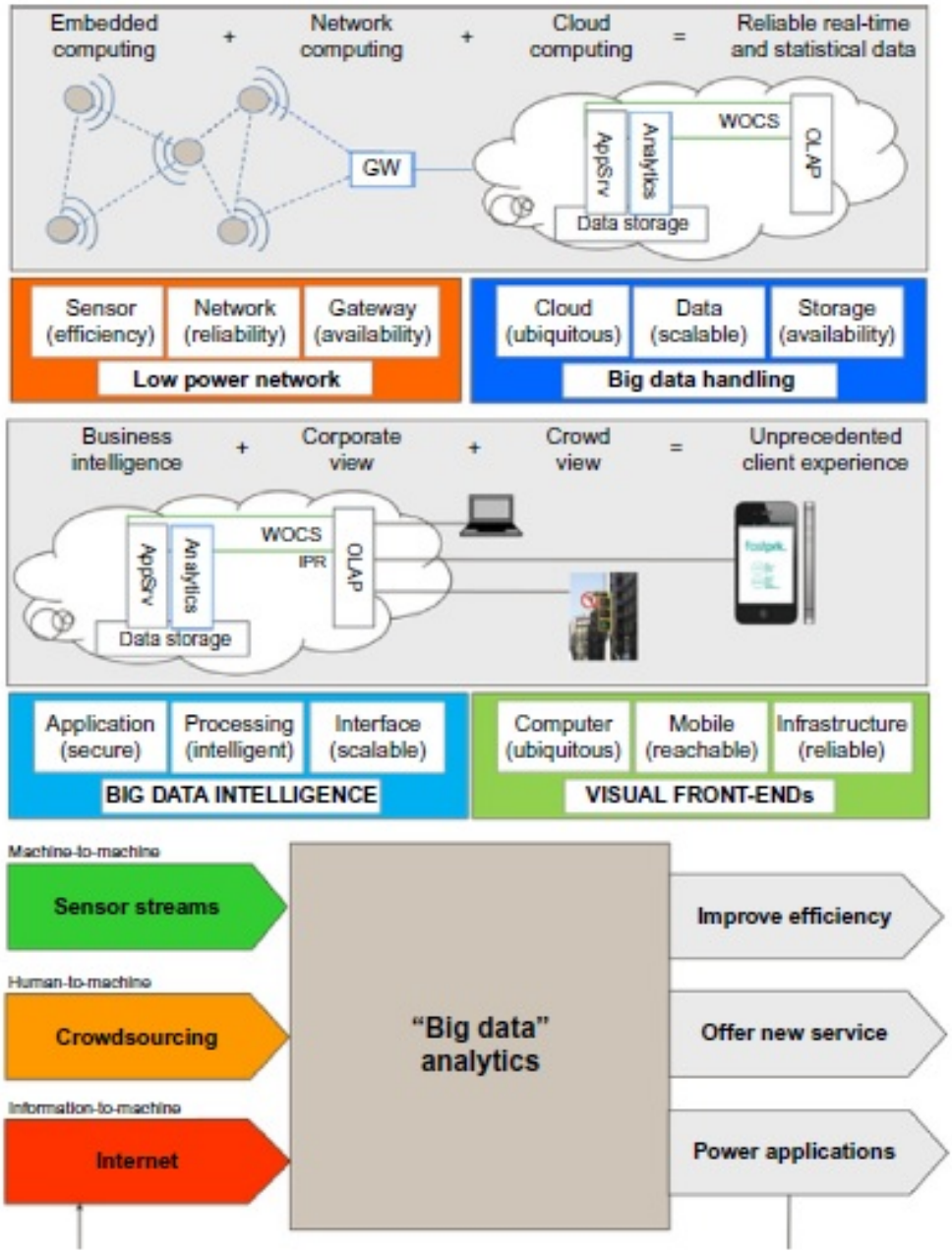


Figure 1.1 : Upstream (top), downstream (middle), and mash-up (bottom) components for typical industrial M2M turnkey solutions.

Machine-to-machine technology landscape:

As for wireless M2M technologies, the ecosystem has so far relied on ZigBee-like and 2G/3G cellular technologies; however, new players are entering the space, such as low-power

Wi-Fi, Bluetooth low energy, and proprietary cellular systems. The pros and cons of these are as follows:

- ZigBee-like technologies: IEEE 802.15.4 and derivatives were (and still are) perceived as the holy grail for wireless sensor networking and M2M usage. Indeed, with the latest IEEE 802.15.4e amendments, it has become a very energy-efficient technology, even in the case of multi-hop mesh. However, in its very nature of providing fairly high data rates over short distances, it is against the essence of M2M, which mainly requires very low data rates over large distances. The need for frequent repeaters/gateways and skilled engineers to handle connectivity/radio planning and the lack of a global critical mass when it comes to coverage and deployment/adoption have prevented the predicted growth and will likely be the demise of the technology itself. ZigBee and derivatives have never made the jump from being a technology to being a turnkey solution, i.e., a system that is easy to use to customers whose core business is not dealing with dimensioning wireless. However, this community has achieved something that none of the others have yet: they have penetrated the control community and were able, despite clearly being technically inferior compared to any of the below systems, to become the certified choice for wireless SCADA-like systems (see, e.g., WirelessHART, ISA100.11a, and Wireless M-Bus).
- Low-power Wi-Fi: An interesting contender in the M2M space is emerging in the form of a well-tuned Wi-Fi system. Wi-Fi enjoys an enormous popularity in both the business-toconsumer (B2C) and B2B markets, with more than 2bn access points installed and a truly vibrant standardization ecosystem active. There is Wi-Fi coverage virtually anywhere where it is worth taking M2M data. Interestingly, it consumes significantly less energy when transmitting information. This is mainly because data are transmitted in a single hop only when needed, thus requiring the radio to be switched on only at events or regular alive beacons. In contrast, ZigBee-like technologies need to listen periodically for neighbors to transmit data; this consumes in the end more energy than a simple single-hop network. It is thus to no surprise that chip manufacturing giants, like Broadcom,

have decided to ditch the IEEE 802.15.4 stack for the more promising IEEE 802.11 stack in its gamble for the IoT and M2M market.

- Proprietary cellular: Cellular networks have the enormous advantage of covering large areas to reach multiple M2M devices without the need for repeaters or numerous base stations. A problem with current standardized cellular solutions (2G/3G/LTE) is the fairly slow standardization cycle and high licensing costs associated with enabling every sensor and actuator with such technology. Proprietary solutions have thus emerged, which bridge this gap. Notable players in the field are SIGFOX, Neul, Cycleo, On-Ramp, and lately also the chip giants Texas Instruments (TI) and STMicroelectronics. These technologies are, in general, fairly narrowband—thus enjoying very large link budgets—and connect hundreds if not thousands of devices to a single base station over a large area. While these solutions are technically well elaborate, they are not deemed to be future-proof since they are not standardized. Also, while coverage is guaranteed by the said systems, mobility and roaming are not supported.
- Standardized cellular: Therefore, standardized cellular M2M technologies will be a crucial constituent to the M2M landscape in the next years to come. Cellular systems enjoy worldwide coverage and interoperability, allowing for M2M devices to roam and be on the move.In addition, interference is being handled by centralized radio resource management (RRM) algorithms, which, to a large extent, ensure that there are no coexistence, delay, or throughput problems. It is thus to no surprise that the major standardization bodies, such as 3GPP, IEEE, and ETSI, started standardizing M2M architectures and protocols.

Cellular M2M requirements:

Before standardized cellular technologies can be used in the M2M space, the following technical requirements need to be addressed, which contrast completely the requirement on human-driven communications:

- There will be a lot ofM2Mnodes, i.e., by orders of magnitude more than human connections. l More and more applications are delay-intolerant, mainly control.
- There will be little traffic per node, mainly in the uplink.

- Nodes need to run autonomously for a long time.
- Automated security and trust mechanisms need to be in place.

Machine-to-machine standards and initiatives:

Market fragmentation and lack of global standards are frequent criticisms of M2Mcommunication systems. The harmonization of technologies at a global scale is addressed in standards development organizations (SDOs), on the one hand, and industrial associations or special interest groups (SIGs), on the other. Next, a number of supra-SDO and supra-SIG initiatives are succinctly described. In addition to all of these, special attention must be paid to enhancing current M2M technologies in order to make them futureproof. Thus focuses on a number of research and innovation projects, which investigate tomorrow's M2M communication networks.

1.2 IOT VS M2M:

The difference between IoT and M2M can be baffling. In truth, the false impression that M2M and IoT is the identical has been a continuing situation of discourse within the tech-sphere. But now, extra than ever, as each technology preserve to evolve at wreck-neck speed.

- **Remote Access:** M2M set the principles of device connectivity on which IoT stepped forward on, IoT development takes place and is almost constructed upon. IoT is the larger vision on connectivity this is fueled by the improvements of M2M packages.M2M is, first and fundamental, a business answer that connects corporations to their likewise connected machines. M2M revolutionizes commercial enterprise operations by way of enabling them to screen and manipulate their machines or pieces of equipment remotely. The primary deliverable of M2M is to connect a device to the cloud so that business can control that tool and gather information from it remotely. IoT is a mass marketplace generation that applies to both clients and corporations. Customer IoT connects people to their devices and similarly, allows for far-off control over the one's devices. Corporation IoT, then again, takes it similarly through facilitating asset tracking and control.
- **Business Performance:** A vital software of both technologies is its' information-centricity. Due to the furnished connectivity, data from these devices may be gleaned for comments on its overall performance, consumer experience, and protection. For example,

facts from M2M systems are used to find machine mistakes and cut down on preservation fees by means of doing away with the want for consistent manual preservation. Certainly, the connectivity that both M2M and IoT offer lets in for the gathering of analytics and different treasured insights that agencies can use to leverage within the development of their structures.Undeniably, M2M and IoT share common factors. The center similarity is that each offer far off gets right of entry to gadget records and each alternate records among machines without human intervention. Although, aren't synonymous. Right here are where IoT and M2M diverge.

- **Connectivity:** As the call implies, M2M connects machines to machines, while IoT takes device-to-device connectivity, integrates internet programs, and connects it to a cloud. While IoT converges disparate systems into an expansive machine to allow new applications, M2M employs isolated structures of sensors and islands of remotely gathered and measured statistics. Actually placed, IoT is extra than device connectivity, as it's for the community of linked gadgets.
- **Scalability:** Because of its cloud-primarily based architecture, IoT is inherently extra scalable than M2M. Cloud-based totally structure eliminates the want for extra difficult-stressed connections and M2M sim card set up.

1.3 WEB OF THINGS (WOT):

The goals of the Web of Things (WoT) are to improve the interoperability and usability of the Internet of Things (IoT). Through a collaboration involving many stakeholders over many years, several building blocks have been identified that help address these challenges.

This specification is focused on the scope of W3C WoT standardization, which can be broken down into these building blocks as well as the abstract architecture that defines how they are related. The building blocks are defined and described in detail in separate specifications. However, in addition to defining the abstract architecture and its terminology and conceptual framework, this specification also serves as an introduction to the WoT building blocks, and explains their interworking:

- The Web of Things (WoT) Thing Description [WOT-THING-DESCRIPTION] normatively provides a machine-readable data format for describing the metadata

and network-facing interfaces of Things. It is based upon the fundamental concepts introduced in this document, such as interaction affordances.

- The Web of Things (WoT) Binding Templates [WOT-BINDING-TEMPLATES] provides informational guidelines on how to define network-facing interfaces in Things for particular protocols and IoT ecosystems, which we call Protocol Bindings. The document also provides examples for a number of existing IoT ecosystems and standards.
- The Web of Things (WoT) Scripting API [WOT-SCRIPTING-API], which is optional, enables the implementation of the application logic of a Thing using a common JavaScript API similar to the Web browser APIs. This simplifies IoT application development and enables portability across vendors and devices.
- The Web of Things (WoT) Security and Privacy Guidelines [WOT-SECURITY] represent a cross-cutting building block. This informational document provides guidelines for the secure implementation and configuration of Things, and discusses issues which should be considered in any systems implementing W3C WoT. However, it should be emphasized that security and privacy can only be fully evaluated in the context of a complete set of concrete mechanisms for a specific implementation, which the WoT abstract architecture does not fully specify. This is especially true when the WoT architecture is used descriptively for pre-existing systems, since the W3C WoT cannot constrain the behavior of such systems, it can only describe them.

This specification also covers non-normative architectural aspects and conditions for the deployment of WoT systems. These guidelines are described in the context of example deployment scenarios, although this specification does not normatively define specific concrete implementations. This specification serves as an umbrella for W3C WoT specifications and defines the basics such as terminology and the underlying abstract architecture of the W3C Web of Things. In summary, the purpose of this specification is to provide:

- a set of use cases in use Cases that lead to the W3C WoT Architecture,

- a set of requirements for WoT implementations in Requirements,
- a definition of the abstract architecture in WoT Architecture
- an overview of a set of WoT building blocks and their interplay in WoT Building Blocks,
- an informative guideline on how to map the abstract architecture to possible concrete implementations in Abstract Servient Architecture,
- informative examples of possible deployment scenarios inExample WoT Deployments,
- and a discussion, at a high level, of security and privacy considerations to be aware of when implementing a system based on the W3C WoT architecture in Security and Privacy Considerations.

Terminology:

This specification uses the following terms as defined here. The WoT prefix is used to avoid ambiguity for terms that are (re)defined specifically for Web of Things concepts.

- **Action**: An Interaction Affordance that allows to invoke a function of the Thing, which manipulates state (e.g., toggling a lamp on or off) or triggers a process on the Thing (e.g., dim a lamp over time).

- **Binding Templates:** A re-usable collection of blueprints for the communication with different IoT platforms. The blueprints provide information to map Interaction Affordances to platform-specific messages through WoT Thing description as well as implementation notes for the required protocol stacks or dedicated communication drivers.
- **Consumed Thing:** A software abstraction that represents a remote Thing used by the local application. The abstraction might be created by a native WoT Runtime, or instantiated as an object through the WoT Scripting API.

- **Consuming a Thing:** To parse and process a TD document and from it create a Consumed Thing software abstraction as interface for the application in the local runtime environment.
- **Consumer:** An entity that can process WoT Thing Descriptions (including its JSON-based representation format) and interact with Things (i.e., consume Things).
- **Data Schema:** A data schema describes the information model and the related payload structure and corresponding data items that are passed between Things and Consumers during interactions.
- **Digital Twin:** A digital twin is a virtual representation of a device or a group of devices that resides on a cloud or edge node. It can be used to represent real-world devices which may not be continuously online, or to run simulations of new applications and services, before they get deployed to the real devices.
- **Domain-specific Vocabulary Linked Data:** vocabulary that can be used in the WoT Thing Description, but is not defined by W3C WoT.
- **Edge Device:** A device that provides an entry point into enterprise or service provider core networks. Examples include gateways, routers, switches, multiplexers, and a variety of other access devices.
- **Event:** An Interaction Affordance that describes an event source, which asynchronously pushes event data to Consumers (e.g., overheating alerts).
- **Exposed Thing:** A software abstraction that represents a locally hosted Thing that can be accessed over the network by remote Consumers. The abstraction might be created by a native WoT Runtime, or instantiated as an object through the WoT Scripting API. Exposing a Thing To create an Exposed Thing software abstraction in the local runtime environment to manage the state of a Thing and interface with the behavior implementation.
- **Hypermedia Control:** A serialization of a Protocol Binding in hypermedia, that is, either a Web link [RFC8288] for navigation or a Web form for performing other operations. Forms can be seen as request templates provided by the Thing to be completed and sent by the Consumer.

- **Interaction Affordance:** Metadata of a Thing that shows and describes the possible choices to Consumers, thereby suggesting how Consumers may interact with the Thing. There are many types of potential affordances, but W3C WoT defines three types of Interaction Affordances: Properties, Actions, and Events. A fourth Interaction Affordance is navigation, which is already available on the Web through linking.
- **Interaction Model:** An intermediate abstraction that formalizes and narrows the mapping from application intent to concrete protocol operations. In W3C WoT, the defined set of Interaction Affordances constitutes the Interaction Model.
- **Intermediary:** An entity between Consumers and Things that can proxy, augment, or compose Things and republish a WoT Thing Description that points to the WoT Interface on the Intermediary instead of the original Thing. For Consumers, an Intermediary may be indistinguishable from a Thing, following the Layered System constraint of REST.
- **IoT Platform:**A specific IoT ecosystem such as OCF, oneM2M, or Mozilla Project Things with its own specifications for application-facing APIs, data model, and protocols or protocol configurations.
- **Metadata:** Data that provides a description of an entity's abstract characteristics. For example, a Thing Description is Metadata for a Thing.
- **Personally Identifiable Information (PII):** Any information that can be used to identify the natural person to whom such information relates, or is or might be directly or indirectly linked to a natural person. We use the same definition as [ISO-IEC-29100].
- **Privacy:**Freedom from intrusion into the private life or affairs of an individual when that intrusion results from undue or illegal gathering and use of data about that individual. We use the same definition as [ISO-IEC-2382]. See also Personally Identifiable Information and Security, as well as other related definitions in [ISO-IEC-29100].
- **Private Security:** Data Private Security Data is that component of a Thing's Security Configuration that is kept secret and is not shared with other devices or users. An example would be private keys in a PKI system. Ideally such data is

stored in a separate memory inaccessible to the application and is only used via abstract operations, such as signing, that do not reveal the secret information even to the application using it.

- **Property:** An Interaction Affordance that exposes state of the Thing. This state can then be retrieved (read) and optionally updated (write). Things can also choose to make Properties observable by pushing the new state after a change. Protocol Binding The mapping from an Interaction Affordance to concrete messages of a specific protocol, thereby informing Consumers how to activate the Interaction Affordance. W3C WoT serializes Protocol Bindings as hypermedia controls.
- **Public Security Metadata:**Public Security Metadata is that component of a Thing's Security Configuration which describes the security mechanisms and access rights necessary to access a Thing. It does not include any secret information or concrete data (including public keys), and does not by itself, provide access to the Thing. Instead, it describes the mechanisms by which access may be obtained by authorized users, including how they must authenticate themselves.
- **Security:** Preservation of the confidentiality, integrity and availability of information. Properties such as authenticity, accountability, non-repudiation, and reliability may also be involved. This definition is adapted from the definition of Information Security in [ISO-IEC-27000], which also includes additional definitions of each of the more specific properties mentioned. We additionally note that it is desirable that these properties be maintained both in normal operation and when the system is subject to attack.
- **Security Configuration:** The combination of Public Security Metadata, Private Security Data, and any other configuration information (such as public keys) necessary to operationally configure the security mechanisms of a Thing.
- **Servient:** A software stack that implements the WoT building blocks. A Servient can host and expose Things and/or host Consumers that consume Things. Servients can support multiple Protocol Bindings to enable interaction with different IoT platforms.

- **Subprotocol:** An extension mechanism to a transfer protocol that must be known to interact successfully. An example is long polling for HTTP.
- **TD:** Short for WoT Thing Description.
- **TDVocabulary:** A controlled Linked Data vocabulary by W3C WoT to tag the metadata of Things in the WoT Thing Description including communication metadata of WoT Binding Templates.
- **Thing or Web Thing:** An abstraction of a physical or a virtual entity whose metadata and interfaces are described by a WoT Thing Description, whereas a virtual entity is the composition of one or more Things.
- **Thing Directory:** A directory service for TDs that provides a Web interface to register TDs (similar to [CoRE-RD]) and look them up (e.g., using SPARQL queries or the CoRE RD lookup interface [CoRE-RD]).
- **Transfer Protocol:** The underlying, standardized application layer protocol without application-specific requirements or constraints on options or subprotocol mechanisms. Examples are HTTP, CoAP, or MQTT.
- **Virtual Thing:** An instance of a Thing that represents a Thing that is located on another system component.
- **WoT Interface:** The network-facing interface of a Thing that is described by a WoT Thing Description.
- **WoT Runtime:** A runtime system that maintains an execution environment for applications, and is able to expose and/or consume Things, to process WoT Thing Descriptions, to maintain Security Configurations, and to interface with Protocol Binding implementations. A WoT Runtime may have a custom API or use the optional WoT Scripting API.
- **WoT Scripting API:** The application-facing programming interface provided by a Servient in order to ease the implementation of behavior or applications running in a WoT Runtime. It is comparable to the Web browser APIs. The WoT Scripting API is an optional building block for W3C WoT.
- **WoT Servient:**Synonym for Servient.
- **WoT Thing Description or Thing Description:** Structured data describing a Thing. A WoT Thing Description comprises general metadata, domain-specific metadata,

Interaction Affordances (which include the supported Protocol Bindings), and links to related Things. The WoT Thing Description format is the central building block of W3C WoT.

Common Patterns:

This section introduces common use case patterns that illustrate how devices/things interact with controllers, other devices, agents and servers. In this section, we use the term client role as an initiator of a transport protocol, and the term server role as a passive component of a transport protocol. This does not imply prescribing a specific role on any system component.

A device can be in a client and server role simultaneously. One example of this dual role is a sensor, that registers itself with a cloud service and regularly sends sensor readings to the cloud. In the response messages the cloud can adjust the transmission rate of the sensor's messages or select specific sensor attributes, that are to be transmitted in future messages. Since the sensor registers itself with the cloud and initiates connections, it is in the 'client' role. However, since it also reacts to requests, that are transmitted in response messages, it also fulfills a 'server' role. The following sections illustrate the roles, tasks, and use case patterns with increasing complexity. They are not exhaustive and are presented to motivate for the WoT architecture and building blocks.

Web Thing:

A Web Thing has four architectural aspects of interest: its behavior, its Interaction Affordances, its security configuration, and its Protocol Bindings, as depicted in Figure. The behavior aspect of a Thing includes both the autonomous behavior and the handlers for the Interaction Affordances. The Interaction Affordances provide a model of how Consumers can interact with the Thing through abstract operations, but without reference to a specific network protocol or data encoding. The protocol binding adds the additional detail needed to map each Interaction Affordance to concrete messages of a certain protocol. In general, different concrete protocols may be used to support different subsets of Interaction Affordances, even within a single Thing. The security configuration aspect of a Thing

represents the mechanisms used to control access to the Interaction Affordances and the management of related Public Security Metadata and Private Security Data.

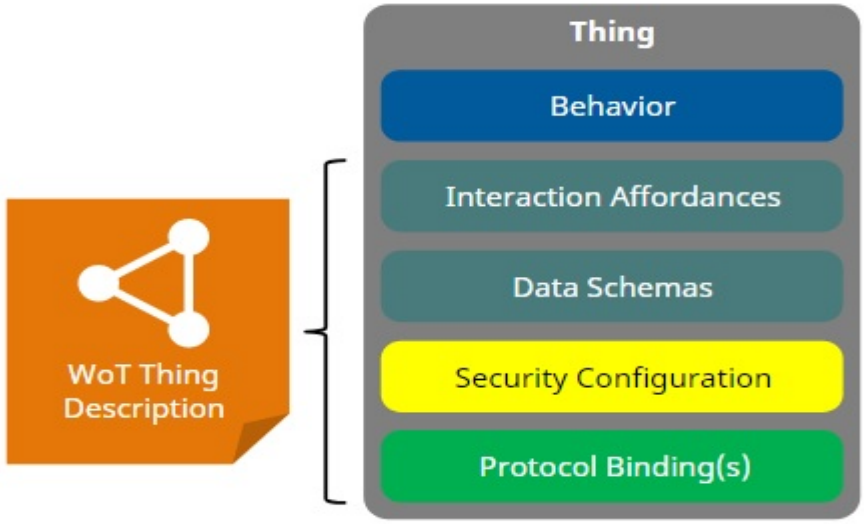


Figure 1.2: Architectural Aspects of a Thing

1.4 COMMUNICATION STANDARDS:

Computers require very precise, rigidly defined rules or protocol for successful communication. Even slight variations can render communication impossible. In order for two computers using different protocols to communicate, some kind of translation must take place. The device that performs this translation is called a gateway. Protocol translation, like language translation, is an imperfect art. Concepts that are easily and clearly expressed in one language or protocol may be difficult or even impossible to translate into another. Consider the English word "snow." Native peoples of the arctic have many words in their languages that all translate to snow.

All of the detail and nuance that was associated with the original word is lost when it is translated to snow. For the English speaking person this may not be much of a problem because the lost details may not have been useful information. However, when the word "snow" is translated in the reverse direction the problem is more difficult. Much more detail is expected.

Which possibility is the best match for the current context? What problems might be caused by choosing the wrong word? These issues must be addressed when designing gateways. How will the gateway handle differences in the way concepts such as trends, schedules, alarms, and prioritized commands are represented and manipulated? How will missing details be filled in without causing unintended consequences? These difficulties often place severe limitations on the kind of information that can be transferred. The result is often connectivity on a limited scale. Some control system manufacturers use gateways within their own product line even though they are using their own proprietary protocols.

A typical example would be field panel controllers and workstations that communicate using one protocol while the field panels communicate with unitary controllers that they supervise via a second protocol. The field panel in this example is also a gateway. Since the same company designs both protocols and they were intended to be used together, the semantic translation issues are avoided. Typically, translation problems occur only when trying to connect products made by different manufacturers or, in some cases, different versions of products made by the same manufacturer. The example shows that gateways may be stand-alone devices or be included as part of a device that has other control functionality. Gateways are sometimes confused with routers because gateways also perform routing tasks in addition to translation. The distinction between a gateway and a router is important.

Router Different from a Gateway:

Like a gateway, a router is used to connect two or more communication networks. Routers and gateways filter message traffic because messages pass through only if the source of the message is on one side and the intended destination is on the other. The difference is that for a router, the connected networks all use the same application protocol messages. Translation is not needed. A router's job is to forward the message to the next network in the path to its destination. This makes routers much simpler than gateways.

A router may be used to connect similar or dissimilar networks. If the networks are similar the router merely forwards the message, changing only the address so that it will go to the next stop. For the case of dissimilar networks the situation is slightly more complicated. This case is illustrated in figure 1 for two BACnet local area network (LAN) options. In Figure 1, a

message originating with a device on a Master-Slave/Token-Passing (MS/TP) LAN passes through a router in order to reach its destination, a workstation residing on an ISO 8802-3 or "Ethernet" LAN. The content of the message and the binary representation of the message do not change as it passes through the router. However, some other changes do take place when the router forwards the message.

Gateway Benefits:

In some situations, a gateway can provide connectivity between devices that would not be possible any other way. If this connectivity is vital to the building automation system, this factor can be so critical that it outweighs any negative factors. This is one reason why an initial critical analysis to determine exactly what information needs to be exchanged is so important. Can the gateway provide all of the information needed? If not, is the information that it provides important enough to proceed with the gateway? In some circumstances, options are available that do not involve a gateway at all.

These options may be available from only a single source or a very limited number of sources. In this case it may be beneficial to consider allowing gateways simply to create a competitive bidding situation. The financial savings resulting from competition may make any limitations of the gateway worthwhile. This approach may permit the involvement of a vendor in the process with whom you have a very good business or service relationship. This business relationship may be important enough during the life cycle of the system to make it worthwhile to accept any limitations that the gateway imposes. Similarly, a vendor’s product may have some unique feature that is very desirable. It may be beneficial to permit the use of a gateway just to get this feature.

Traditionally, building automation functions such as lighting control, HVAC control, and life safety systems have been isolated from each other. Sometimes hard-wired connections between an HVAC control system and a life safety system exist that permits monitoring of some limited status information. A growing consensus in the building industry indicates that important benefits may be obtained through more complete integration of these systems. However, complete integration may cause concern because it could possibly have an adverse impact on the ability to guarantee the integrity of critical life safety systems. A gateway provides an

opportunity to meet both goals by significantly improving the ability to integrate and manage the systems at the same time retaining some isolation because the gateway limits the interaction. In some cases this may be important for meeting code requirements.

A policy decision might be made to require the use of standard communication protocols, like BACnet, for all future purchases at a facility. This can be a prohibitively expensive change if the entire building automation system must be replaced at one time. Gateways provide a way to retain installed proprietary equipment that is still functioning satisfactorily and to migrate towards standards-based products in a step-by-step manner as equipment reaches the end of its useful life. These potential benefits can make gateways desirable. It is important to keep in mind that all gateways impose limitations. A building automation system with gateways is more complicated to design and maintain than one where all of the components communicate using the same protocol. There is a cost to using gateways that must be balanced against the potential benefits.

Gateway Limitations:

All gateways have finite information storage and processing resources. The gateway must somehow map the information and concepts from one protocol to the other. This requires translation algorithms for each kind of information that needs to be translated and tables that provide mapping information. Sometimes the gateway stores a local copy of the information from one network in a form that makes it easily accessible to devices on the other network. The computer resources necessary to implement this process vary from case to case. In general, both the number of different types of information that are to be translated and the number of occurrences of each type, affect the amount of memory and processor time needed. The more information exchange your application requires, the more probable it is that a gateway will not have the capacity to do the job.

Future expandability is usually limited and sometimes impossible. Specifications for gateways must clearly state what information must be available through the gateway and what expansion capability is required. Any proposed gateway must be evaluated in the context of these requirements. Often, using a gateway will require a compromise between getting all of the desired information and keeping within budget. As described earlier, gateways have a limited

ability to translate dissimilar concepts. The more differences there are between the protocols involved, the more likely it is that the gateway will only be able to translate a portion of the available information. This will probably not be a problem for simple concepts such as temperatures and on/off status.

However, problems may arise for more complex concepts such as schedules, alarms, prioritized commands, and trending data. Different approaches to handling these building automation functions make gateway design and implementation very difficult. In some cases one or more of these functions will be simply impossible to do through the gateway. Many control systems provide a way to program or configure controllers over the network. It is usually not possible to accomplish these tasks by passing messages through a gateway. Instead, a separate connection that bypasses the gateway is usually required. Gateways can also introduce time delays when attempting to retrieve information. In addition to the processing time required to do the translation, the connected networks may have different transmission speeds and different rules for gaining access to the media that also contribute to the delay.

Sometimes gateways are designed only to pass a limited amount of very specific information. The gateway may poll devices for this information and store a local copy. When a request for this information is received the response is prepared by the gateway based on the local copy of the data. This approach can speed up the response but it does not scale easily to large systems. It can also result in returning old and misleading data. What happens, for example, if the proprietary communication fails temporarily and the stored data cannot be updated? Does the gateway pass the previously stored data that is now obsolete? Will the recipient be able to tell the difference? Gateways make troubleshooting network problems more difficult.

Different tools are needed to see and interpret the protocols on both sides of the gateway. Any ambiguity introduced by the gateway's imperfect translation introduces the possibility of error. Even the limitation on the amount of information accessible through the gateway can make troubleshooting difficult, because it may not be possible to access all of the information needed to diagnose the problem. It should be clear that, under the right circumstances, a gateway is very useful. However, it comes with a price. A wrench is very useful and effective tool to tighten a nut. It is not a particularly helpful tool to drive a nail, even though with persistence it may be used for that purpose. A gateway is a useful and effective tool to get limited connectivity

between otherwise incompatible devices. As the amount and variety of information that needs to be exchanged grows, the viability of using a gateway to accomplish the task diminishes.

In some circumstances, a gateway is the only option for a critical integration and is definitely worthwhile. In other cases, a gateway is acceptable because the advantages and disadvantages roughly balance each other. However, sometimes a gateway is an unnecessary liability and should not be used. The key point is that no two jobs are the same and the costs and benefits must be weighed on a case-by-case basis. The most important part of the process is to know where to look for the benefits, as well as the trouble, so that an informed decision can be made. First, determine what information needs to be exchanged and prioritize the list. Then, request detailed information about the gateway design from the vendor so that the potential benefits and weaknesses.

Questions to Practice:

PART A

1. Discuss the basic principles of communication needed to frame an IoT Device
2. Quote in brief about the Communication Standards
3. List the Special types of terminology in Gateway Design
4. Evaluate the function of Machine to Machine Communication
5. Explain how Smart Utility Management used in M2M Communication
6. Find the function of Strategy analytics in M2M Communication
7. Simplify the role and importance of Web of things

PART B

1. Analyze in detail about Machine to Machine Communication.
2. Elucidate the selection of M2M Communication Process with suitable diagram in detail
3. Investigate Internet of Things in Communication Standards

4. List and explain Web of Things with suitable applications
5. Enumerate Machine to Machine Communication and Internet of Things with similarities and differences in detail.

References:

1. Mischa Dohler, Carles Anton-Haro, “Machine-to-machine (M2M) Communications”, Woodhead Publishing, 1st Edition, 2014.
2. Perry Lea, “Internet of Things for Architects: Architecting IoT solutions by implementing sensors, communication infrastructure, edge computing, analytics, and security”, Packt, 1st Edition, 2018.
3. Maciej Kranz , “Building the Internet of Things”, John Wiley & Sons, 1st Edition, 2017.
4. David Boswarthick, Omar Elloumi, Olivier Hersent, “M2M Communications: A Systems Approach”, John Wiley & Sons, 1 st Edition, 2012.
5. Anupama C. Raman and Pethuru Raj, “The Internet of Things: Enabling Technologies, Platforms, and Use Cases”, CRC press, 1st Edition, 2017.

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF ELECTRICAL AND ELECTRONICS

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

UNIT – II – Communication Models and Data Exchange Formats – SECA7019

UNIT 2 COMMUNICATION MODELS AND DATA EXCHANGE FORMATS

Communication Models - Request-Response, Publish Subscribe, Communication Patterns - Telemetry, Inquiry, Status, Notifications, Data exchange formats e.g. JSON

2.1CommunicationModels:

The concept of combining computers, sensors, and networks to monitor and control devices has been around for decades, the recent confluence of key technologies and market trends is emerging in a new reality for the "Internet of Things''. The main vision of IoT is to usher in a revolutionary, fully interconnected "smart" world, with relationships between objects and their environment and objects and people becoming more tightly intertwined. However, if the system is more complex, a number of potential challenges may stand in the way of this vision – particularly in the areas of security, privacy, interoperability and standards, legal, regulatory, and rights issues, as well as the inclusion of emerging economies. Clearly, the Internet of Things involves a complex and evolving set of technological, social, and policy considerations across a diverse set of stakeholders.

While the end user benefits from effective communication models, it should be mentioned that effective IoT communication models also enhance technical innovation and open opportunity for commercial growth. New products and services can be designed to take advantage of IoT data streams that didn't exist previously, acting as a catalyst for further innovation. The fact is that IoT devices will be found everywhere and will enable ambient intelligence in the future. Therefore, the overview of the various communication models used in IOT, given below, are of great value. The IoTs allow people and things to be connected anytime, anyplace, with anything and anyone, ideally using any path/network and any service. From an operational perspective, it is useful to think about how IoT devices connect and communicate in terms of their technical communication models, before making decisions:

- The device-to-device communication model represents two or more devices that

directly connect and communicate between one another, rather than through an intermediary application server. These devices communicate over many types of networks, including IP networks or the Internet. Often, however these devices use protocols like Bluetooth, Z-Wave, or ZigBee to establish direct device-to-device communications.

- In a device-to-cloud communication model, the IoT device connects directly to an Internet cloud service like an application service provider to exchange data and control message traffic. This approach frequently takes advantage of existing communications mechanisms like traditional wired Ethernet or Wi-Fi connections to establish a connection between the device and the IP network, which ultimately connects to the cloud service. Frequently, the device and cloud service are from the same vendor. If proprietary data protocols are used between the device and the cloud service, the device owner or user may be tied to a specific cloud service, limiting or preventing the use of alternative service providers. This is commonly referred to as "vendor lock-in'', a term that encompasses other facets of the relationship with the provider such as ownership of and access to the data. At the same time, users can generally have confidence that devices designed for the specific platform can be integrated.
- In the device-to-gateway model, or more typically, the device-to-application-layer gateway (ALG) model, the IoT device connects through an ALG service as a conduit to reach a cloud service. In simpler terms, this means that there is application software operating on a local gateway device, which acts as an intermediary between the device and the cloud service and provides security and other functionality such as data or protocol translation. Several forms of this model are found in consumer devices. In many cases, the local gateway device is a smartphone running an app to communicate with a device and relay data to a cloud service. The other form of this device-to-gateway model is the emergence of "hub" devices in home automation applications. These are devices that serve as a local gateway between individual IoT devices and a cloud service, but they can also bridge the interoperability gap between devices themselves.

- The back-end data-sharing model refers to a communication architecture that enables users to export and analyze smart object data from a cloud service in combination with data from other sources. This architecture supports “the [user’s] desire for granting access to the uploaded sensor data to third parties”. This approach is an extension of the single device-to-cloud communication model, which can lead to data silos where “IoT devices upload data only to a single application service provider’’. A back-end sharing architecture allows the data collected from single IoT device data streams to be aggregated and analyzed and suggests a federated cloud services approach, or cloud applications programmer interfaces (APIs), to achieve interoperability of smart device data hosted in the cloud.

By enabling the user to achieve better access to an IoT device and its data, the overall value of the device is clearly amplified, however, these networked benefits come with trade-offs. Careful consideration needs to be paid to the incurred cost burdens placed on users to connect to cloud resources when considering an architecture, especially in regions where user connectivity costs are high.

Request-Response Model:

Request-response model is communication model in which the client sends requests to the server and the server responds to the requests. When the server receives a request, it decides how to respond, fetches the data, retrieves resource representation, prepares the response, and then sends the response to the client. Request-response is a stateless communication model and each request-response pair is independent of others.

HTTP works as a request-response protocol between a client and server. A web browser may be the client, and an application on a computer that hosts a web site may be the server.

Example: A client (browser) submits an HTTP request to the server; then the server returns a response to the client. The response contains status information about the request and may also contain the requested content.

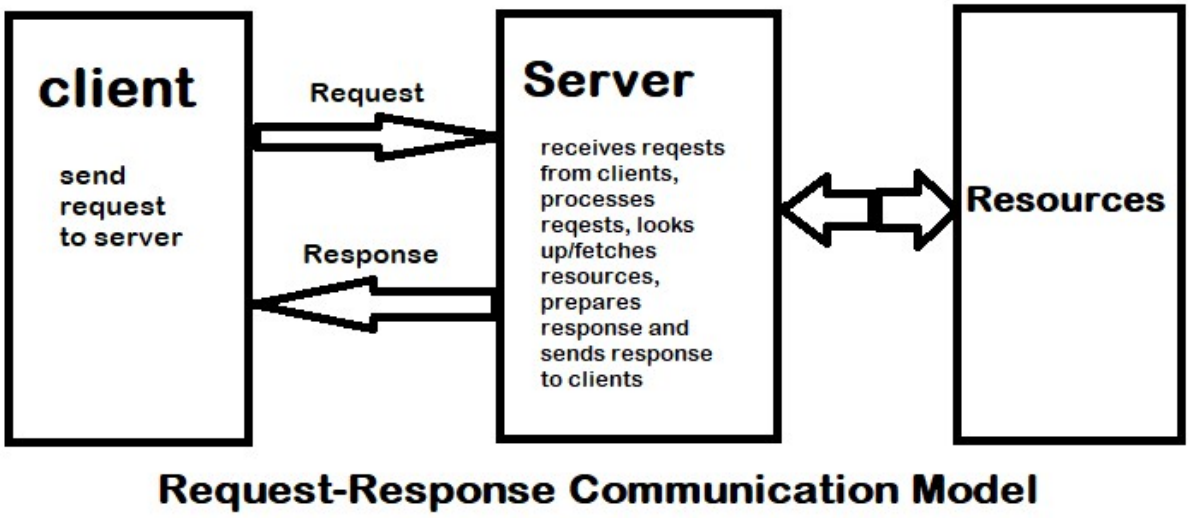


Figure 2.1: Request Response Communication Model

The request-response model in System.Net is defined by the WebRequest and WebResponse classes. These abstract base classes provide an implementation pattern that's used by protocol-specific implementations such as the FILE and HTTP handlers described in the previous section. This common pattern for request-response enables developers to resolve resources containing different URI schemes in a consistent and protocol-independent manner. For example, this common model enables a developer to write the same code to download a resource on an FTP server as that needed to download a resource on an HTTP server. This model is extensible, so third-party implementers can plug in their own protocol handlers as well. There are also a number of helper classes that work with this model and apply to the various supported protocols. Figure 2.2 highlights the request-response model, including the key methods and properties of WebRequest and WebResponse, as well as the helper classes that work with the model.

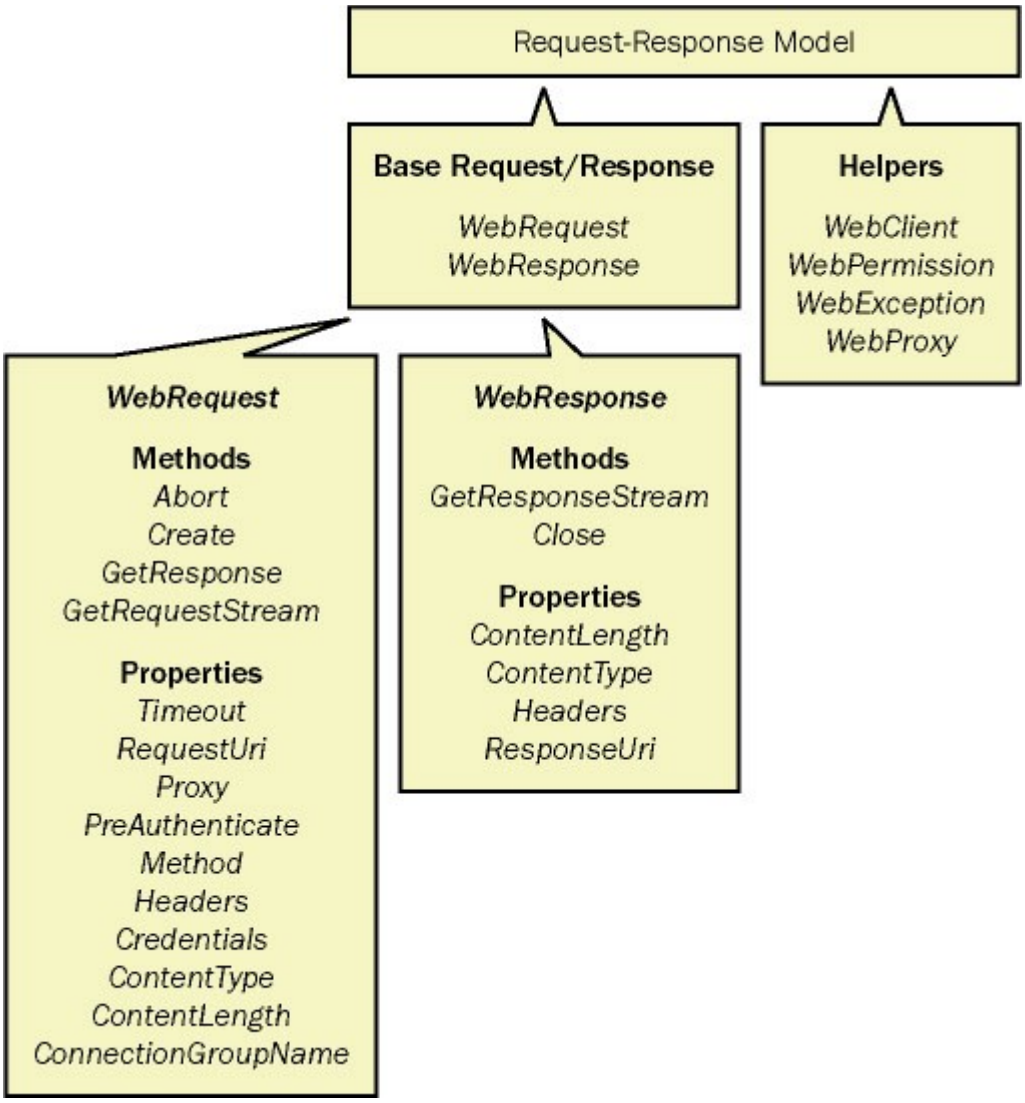


Figure 2.2 The System.Net request-response model

The WebRequest and WebResponse classes contain methods and properties that apply generically across the different supported protocols and enable applications to download and upload data given a specific URI and to specify authentication details, proxy information, and content details such as the type and size of the content. An asynchronous pattern is also provided for any operation that could be blocking on network activity. Extensibility is achieved through the WebRequest.RegisterPrefix method or through entries in the System.Net section of the application-specific or machine-wide configuration file.

Request-response model helper classes perform a specific function that's usually common to the different protocols that implement WebRequest and WebResponse. WebProxy and WebException are both examples of such helper classes. Another example of a request-response model helper is the WebPermission class. WebPermission is used to define code access security privileges for classes that implement the WebRequest and WebResponse base classes. Privileges that are controlled through WebPermission include the ability to access a specific URI, a wildcard set of URIs as supplied by a regular expression, or all URIs as identified by the System.Security.Permissions.PermissionState enumeration. Finally, the WebClient class provides a number of task-focused methods that perform operations that are commonly useful to applications that deal with resources on the Web. Here are the operations available as one-line method calls with WebClient:

- Download a URI-based resource to memory.
- Download a URI-based resource and save it to a file.
- Given a URI, return a readable stream.
- Given a URI, return a writable stream.
- Upload data from memory to a URI-based resource.
- Upload data from a file to a URI-based resource.
- Post a name-value collection to a Web form.

Because the WebClient class uses the WebRequest and WebResponse classes in its implementation, it works with any protocol handler that has been registered into the request-response model.

Publish-Subscribe Model:

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

publisher, it sends the data to all the subscribed consumers.

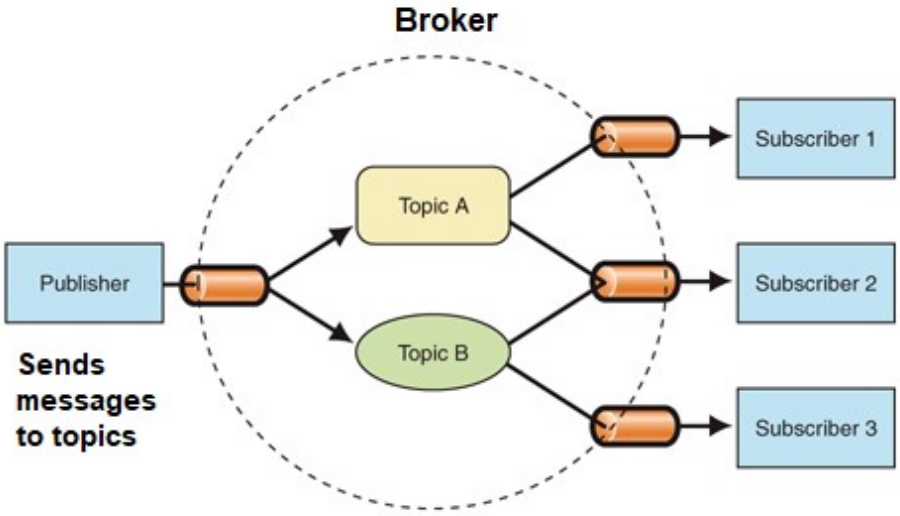


Figure 2.3 Publish Subscribe Model

This section describes various concepts related to publish-subscribe.

Queue: A queue is an entity that supports the notion of named subjects of interest. Queues can be characterized as:

- Non-persistent queue (lightweight queue): The underlying queue infrastructure pushes the messages published to connected clients in a lightweight, at-best-once, manner.
- Persistent queue: Queues serve as durable containers for messages. Messages are delivered in a deferred and reliable mode.
- Agent: Publishers and subscribers are internally represented as agents. There is a distinction between an agent and a client. An agent is a persistent logical subscribing entity that expresses interest in a queue through a subscription. An agent has properties, such as an associated subscription, an address, and a delivery mode for messages. In this context, an agent is an electronic proxy for a publisher

or subscriber. A client is a transient physical entity. The attributes of a client include the physical process where the client programs run, the node name, and the client application logic. There could be several clients acting on behalf of a single agent. Also, the same client, if authorized, can act on behalf of multiple agents.

- Rule on a queue: A rule on a queue is specified as a conditional expression using a predefined set of operators on the message format attributes or on the message header attributes. Each queue has an associated message content format that describes the structure of the messages represented by that queue. The message format may be unstructured (RAW) or it may have a well-defined structure (ADT). This allows both subject- or content-based subscriptions.
- Subscriber: Subscribers (agents) may specify subscriptions on a queue using a rule. Subscribers are durable and are stored in a catalog.
- Database event publication framework: The database represents a significant source for publishing information. An event framework is proposed to allow declarative definition of database event publication. As these pre-defined events occur, the framework detects and publishes such events. This allows active delivery of information to end-users in an event-driven manner as part of the publish-subscribe capability.
- Registration: Registration is the process of associated delivery information by a given client, acting on behalf of an agent. There is an important distinction between the subscription and registration related to the agent/client separation.Subscription indicates an interest in a particular queue by an agent. It does not specify where and how delivery must occur. Delivery information is a physical property that is associated with a client, and it is a transient manifestation of the logical agent (the subscriber). A specific client process acting on behalf of an agent registers delivery information by associating a host and port, indicating where the delivery should be done, and a callback, indicating how there delivery should be done.
- Publishing a message: Publishers publish messages to queues by using the

appropriate queuing interfaces. The interfaces may depend on which model the queue is implemented on. For example, an enqueue call represents the publishing of a message.

- Rules engine: When a message is posted or published to a given queue, a rules engine extracts the set of candidate rules from all rules defined on that queue that match the published message.
- Subscription services: Corresponding to the list of candidate rules on a given queue, the set of subscribers that match the candidate rules can be evaluated. In turn, the set of agents corresponding to this subscription list can be determined and notified.
- Posting: The queue notifies all registered clients of the appropriate published messages. This concept is called posting. When the queue needs to notify all interested clients, it posts the message to all registered clients.
- Receive a message: A subscriber may receive messages through any of the following mechanisms:
    - A client process acting on behalf of the subscriber specifies a callback using the registration mechanism. The posting mechanism then asynchronously invokes the callback when a message matches the subscriber's subscription. The message content may be passed to the callback function (non-persistent queues only).
    - A client process acting on behalf of the subscriber specifies a callback using the registration mechanism. The posting mechanism then asynchronously invokes the callback function, but without the full message content. This serves as a notification to the client, which subsequently retrieves the message content in a pull fashion (persistent queues only).
    - A client process acting on behalf of the subscriber simply retrieves messages from the queue in a periodic, or some other appropriate, manner. While the messages are deferred, there is no asynchronous

delivery to the end-client.

2.2 Telemetry:

Telemetry is the process by which an object's characteristics are measured (such as velocity of an aircraft), and the results transmitted to a distant station where they are displayed, recorded, and analyzed. The transmission media may be air and space for satellite applications, or copper wire and fiber cable for static ground environments like power generating plants.

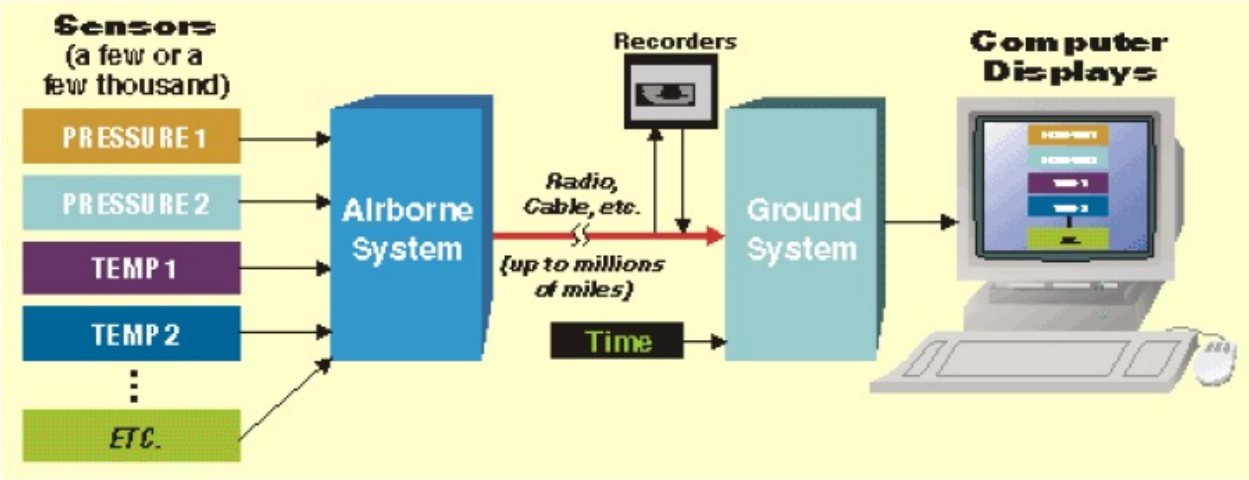


Figure 2.4 Telemetry System

In today's telemetry applications, which support large numbers of measurands, it is too costly and impractical to use separate transmission channels for each measured quantity. The telemetry process involves grouping measurements (such as pressure, speed, and temperature) into a format that can be transmitted as a single data stream. Once received, the data stream is separated into the original measurement's components for analysis.

Telemetry lets you stay in a safe (or convenient) location while monitoring what's taking place in an unsafe (or inconvenient) location. Aircraft development, for example, is a major application for telemetry systems. During initial flight testing, an aircraft performs a variety of test maneuvers. The critical flight data from a maneuver is transmitted to flight test engineers at a ground station where results are viewed in real time or analyzed within seconds of the maneuver. Real-time monitoring allows the "safety officer" to make instant decisions on whether to proceed with or terminate a test. With real-time analysis, the flight test engineer can request a maneuver be repeated, the next maneuver be performed, or test plan alternatives be substituted. Real-time data is also captured to storage media, such as disk and tape, for later analysis and archiving.

Today's telemetry systems are built from commercial-off-the-shelf (COTS) products. But while they all have many common elements, they are each uniquely configured to meet specific application requirements. A telemetry system is often viewed as two components, the Airborne System and the Ground System. In actuality, either or both may be in the air or on the ground.

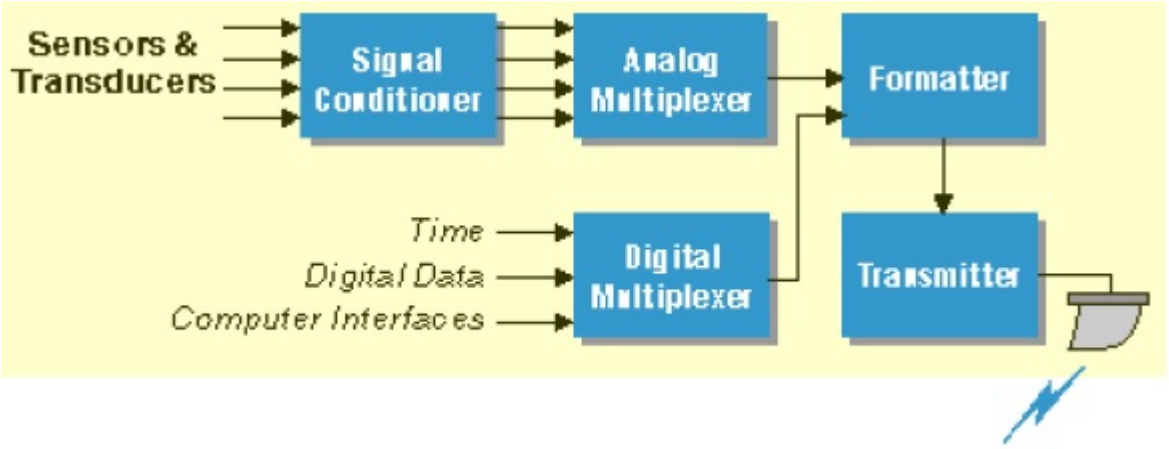


Figure 2.5 : Airborne/Data Acquisition System

Data acquisition begins when sensors (aka, transducers) measure the amount of a physical attribute and transform the measurement to an engineering unit value. Some sensors produce a voltage directly (thermocouples for temperature or piezoelectric strain gages for

acceleration), while others require excitation (resistive strain gages, potentiometers for rotation, etc.). Sensors attached to signal conditioners provide power for the sensors to operate or modify signals for compatibility with the next stage of acquisition. Since maintaining a separate path for each source is cumbersome and costly, a multiplexer (historically known as a commutator) is employed. It serially measures each of the analog voltages and outputs a single stream of pulses, each with a voltage relative to the respective measured channel. The rigorous merging of data into a single stream is called Time Division Multiplexing or TDM.

The scheme where the pulse height of the TDM stream is proportional to the measured value is called Pulse Amplitude Modulation (PAM). A unique set of synchronization pulses is added to identify the original measurands and their value. PAM has many limitations, including accuracy, constraints on the number of measurands supported, and the poor ability to integrate digital data.

Pulse Code Modulation (PCM) is today's preferred telemetry format for the same reasons that PAM is inadequate. Accuracy is high, with resolution limited only by the analog to digital converter (ADC), and thousands of measurands can be acquired along with digital data from multiple sources, including the contents of the computer's memory and data buses. In a PCM-based system, the original PAM multiplexer's analog output is digitized to a parallel format. The Output Formatter along with synchronization data for measurand identification merges this, plus other sources of digital data. The Output Formatter serializes the composite parallel data stream to a binary string of pulses (1's and 0's) for transmission on copper wire, fiber cable, or "the ether." All components from after the sensor to the formatter comprise the encoder (see figure below}. Other, often remote encoders are used to multiplex additional sensor data into the main encoder's output. Not only does this expand the number of measurands to thousands per stream, but it also eliminates the weight of cables required for each sensor.

The output of the main encoder is filtered and transmitted via radio transmitter and antenna, coax cable, telephone line, tape recorder, etc. Filtering rounds or smoothes the

square data pulses to reduce frequencycontent and thus the required transmitter bandwidth. At the Ground Station, the received data stream is amplified. Since the transmission path often distorts the already rounded signal, a bit synchronizer reconstructs it to the original serial square wave train. Then, a decommutator or decom (similar to that found in L-3' Visual Test System or System 550) recognizes the synchronization pattern and returns the serial digital stream to parallel data. The decom also separates the PCM stream into its original measurands (also known as prime parameters) and data.

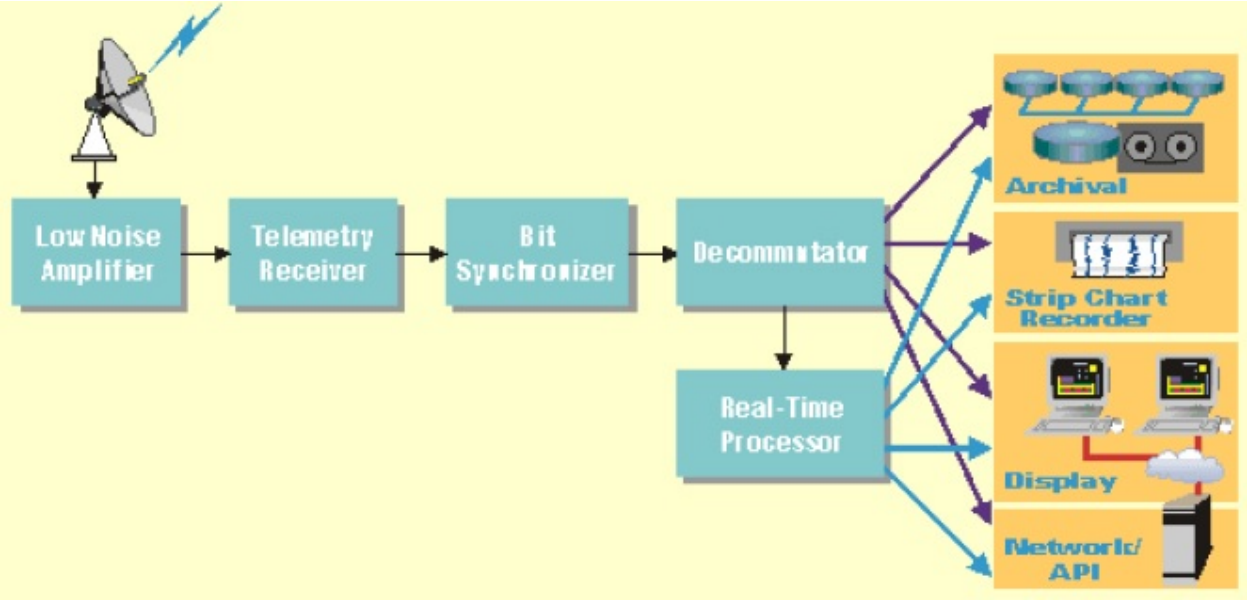


Figure 2.6: Ground station Block Diagram

The computer (in the Visual Test System) or the telemetry front end (System 550) selects prime parameters for real-time processing; archiving to disk or tape; display; output to strip chart recorders and annunciators; or distribution to other computing resources according to the test plan.

L-3 Telemetry-West and its sister divisions manufacture virtually the entire telemetry system — from signal conditioners to antennas for the Airborne System and from antennas to telemetry receivers for the Ground System. The following table breaks down which L-3 divisions provide what for today's telemetry system requirements. Just remember that when you need to put everything together, L-3 Telemetry-West will help specify, integrate, and install all L-3 and

third-party components for a total telemetry solution.

Airborne Systems:

Data Acquisition:A wide variety of sensors (also known as transducers) are used to measure and acquire a physical property's value. It is up to the instrumentation engineer to select the device to meet the environmental, response, accuracy, size, and cost specifications for the application. Signal conditioners serve as the interface of the data acquisition system from the transducers. Many transducers require ac or dc power (e.g., thermistors, strain gages, and linear variable differential transformers - LVDTs) while others generate signals (tachometers, thermocouples, and piezoelectric strain gages). They provide excitation (power), network calibration, signal amplification, and filtering.

In airborne data acquisition, sensor output characteristics must be transformed, filtered, or modified for compatibility with the next stage of the system. The absolute relationship between the output and the actual property value of the measurand may vary with time, altitude, pressure, temperature, etc. Therefore, signal conditioners also incorporate calibration features to assist in defining the relationships.

A system under test may be subjected to known physical characteristics and the output measured to ascertain and verify the relationship between the sensor and its output. For example, when on the ground, an airplane's flaps may be moved at known angles, while measurements are taken on sensor or airborne system output. The plot of angle vs. output will be used by the ground system for real-time data display in engineering units.

Multiplexer: Whatever the quantities monitored at the data source (whether electrical or physical), the cost to transmit each quantity through a separate channel would be prohibitive. Think of the equipment and cables or frequency spectrum required to monitor and transmit several hundred or thousands of measurands! One way to conserve resources is to share time or frequency spectrum with techniques such as Time-Division Multiplexing (TDM) and Frequency Multiplexing (FM), respectively.

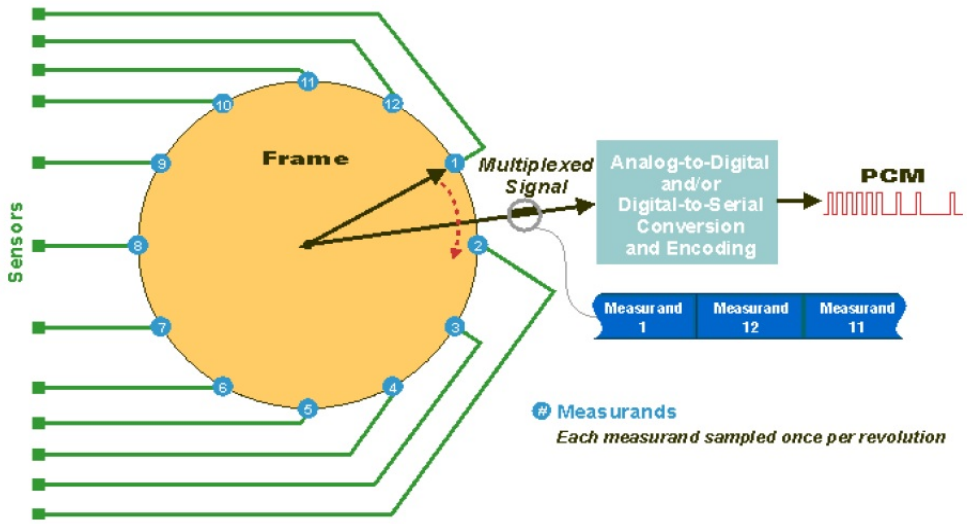


Figure 2.7: Basic Communication

Today, the most popular form of telemetry multiplexing (originally called commutation, as in an electric motor's commutator) is TDM. Here, each channel is serially sampled for an instant by the multiplexer.

When all channels have been sampled, the sequence restarts at the first channel. Thus, samples from a particular channel are interleaved in time between samples from all of the other channels. An example of a simple interleaved data stream is shown below as the output of the commutator: Since no measurand is monitored continuously, sampling must be accomplished fast enough so that the value of each measurand does not change significantly during intervals. In practice, each parameter is measured at a rate of three to five times the highest frequency of interest. There's a definite science to selecting efficient sampling rates and measurand positions.

Commutation:A complete scan by the multiplexer (one revolution of the commutator) produces a frame of the stream of words containing the value of each measurand. Every scan produces the same sequence of words. Only the value of a measurand is captured, not its address (name). If only the measurand's data is captured, there is no way to distinguish the owner of one value from the next. Thus, a unique word called the frame sync is added at the end of each frame to serve as a reference for the process of decommutating the stream's data (i.e., extracting it into individual

measurand values).

In a simple commutator, each data word is sampled once per revolution at a rate compatible with the measurand with the fastest changing data. Since the rate of change of a measurand's value varies tremendously, the sampling frequency rate must accommodate it. As an example, to characterize vibration requires many more samples per second (thousands) than temperature (fractions). According to the Nyquist Theorem, you must sample data at twice the maximum frequency component for the signal to be acquired. Sampling rates of 5 times the maximum frequency component are typical.

A low pass filter is used to eliminate any frequencies that you cannot accurately digitize to prevent aliasing. If we were to take a worst-case approach to sampling all measurands at the highest rate, we could expect much waste in carrier frequency spectrum and power. Sampling rates should therefore vary with respect to frequency content and be somewhat independent of other measurands with different periodic acquisition rates. Highly sampled measurands are super-commutated with multiple occurrences of the measurand in each frame.

Frame Synchronization Pattern: Identifying the end of each minor frame period is the synchronization (sync) word, which is a unique sequence of 1's and 0's. The pattern is generally a pseudo-random sequence that is unlikely to occur randomly in the acquired data and usually occupies two words (or more) in the minor frame. The IRIG-106 Standard lists recommended patterns for lengths 16 through 33 bits. The first three bits transmitted in a frame sync pattern are always a "1," regardless of LSB or MSB alignment.

Pattern Length	Patterns										
16	111	010	111	001	000	0					
17	111	100	110	101	000	00					
18	111	100	110	101	000	000					
19	111	110	011	001	010	000	0				
20	111	011	011	110	001	000	00				
21	111	011	101	001	011	000	000				
22	111	100	110	110	101	000	000	0			
23	111	101	011	100	110	100	000	00			
24	111	110	101	111	001	100	100	000			
25	111	110	010	110	111	000	100	000	0		
26	111	110	100	110	101	100	110	000	00		
27	111	110	101	101	001	100	110	000	000		
28	111	101	011	110	010	110	011	000	000	0	
29	111	101	011	110	011	001	101	000	000	00	
30	111	110	101	111	001	100	110	100	000	000	
31	111	111	100	110	111	110	101	000	010	000	0
32	111	111	100	110	101	100	101	000	010	000	00
33	111	110	111	010	011	101	001	010	010	011	000

Figure 2.8: Frame Synchronization Pattern

The length of the frame sync is longer than usual data words to reduce the probability of actual data matching it. The frame sync should also be commensurate with the number of words in the minor frame (typically, it occupies 1 to 5 percent of the total minor frame). An identical pattern is repeated for every minor frame on the assumption that random data will not consistently match the defined pattern. The decommutator can then be programmed to lock onto this pattern to begin regenerating the original commutated measurands.

Data exchange formats:

Data exchange is the process of taking data structured under a source schema and transforming it into data structured under a target schema, so that the target data is an accurate representation of the source data.Data exchange allows data to be shared between different computer programs. It is similar to the related concept of data integration except that data is actually restructured (with possible loss of content) in data exchange. There may be no way to transform an instance given all of the constraints. Conversely, there may be numerous ways to transform the instance (possibly infinitely many), in which case a "best" choice of solutions has to be identified and justified.

JSON - JavaScript Object Notation:

JavaScript Object Notation is a schema-less, text-based representation of structured data that is based on key-value pairs and ordered lists. Although JSON is derived from JavaScript, it is supported either natively or through libraries in most major programming languages. JSON is commonly, but not exclusively, used to exchange information between web clients and web servers. Over the last 15 years, JSON has become ubiquitous on the web. Today it is the format of choice for almost every publicly available web service, and it is frequently used for private web services as well.

Features and design goals The main features and design goals of JSON include It's text based (so, also humans, not only machines, can read and create JSON documents) It's minimal: JSON defines only a very small set of rules Portability Subset of JavaScript Because of this simplicity, JSON has become popular and many websites use JSON instead of XML to serialize data for communication between server applications and their clients.

Javascript origin:

JSON originates from the object literals of JavaScript (ECMAScript) and is specified in RFC 7159 (The JavaScript Object Notation (JSON) Data Interchange Format). In fact, JSON is referred in ECMA-262 (see also ECMA-404 (The JSON Data Interchange Format)). Because the JSON syntax originates in JavaScript, it is possible to use eval in JavaScript to parse a JSON document. Of course, this poses a security risk (like that of SQL injection) if

the JSON document contains executable code.

Tokens:

The grammar of JSON has only six structural characters: { and } [and] : , Besides them, there are three other kinds of tokens: strings (enclosed in ") is a sequence of Unicode characters numbers literals (true, false or null).

Primitive data types (scalars): JSON specifies four primitive data types. Each scalar value has one of these data types.

Data Type	**~Example`**	**Comment**
String	"string"	A sequence of zore or more Unicode characters (UTF-8 (default), UTF-16 or UTF-32)
Number	42, -13.9	Represented in base 10; leading zeros are not allowed
Boolean	true, false	The only two values whose data type is *boolean*
Null	null	The only value whose data type is *null*

Structured types:

The primitive data types are building blocks to create structured types. There are twostructured types:

objects	Unordered set name/value pairs
arrays	Ordered list of values (whose data type need not to be the same)

The terms object and array have their origins in JavaScript (see JavaScript objects and arrays)

Objects:

JSON objects are embedded in curly braces ({ … }). A JSON object consists of comma separated key-value pairs. The key is a string and is separated from the value by a colon (:). These key value pairs are referred to as members.

```
{
   "item": "Lemon",
   "price": {
         "val":      1.84,
         "currency: "CHF"
   }
}
```

Arrays:

Arrays are embedded in square brackets: […]. An array is an ordered, comma-separated list of JSON values.

```
[ "foo",
   42,
   null,
   { "foo": "bar" }
]
```

Values:

A value is either a primitive or a structured type.

Questions to Practice:

Part-A

1. List the advantages and disadvantages of Communication Models in Gateway Design
2. Explain the functions of Communication Models
3. List out the steps involved in Communication Models
4. Give the summary of Request and Response in Communication Models
5. Discuss the impacts of Publish and Subscribe in Communication Models
6. List out the properties of Communication Patterns

Part-B

1. Analyze the Performance of Telemetry in Communication Patterns

2. Categorize Inquiry and Status in Communication Patterns

3. Enumerate structural characters of Tokens in JSON.

4. Classify the concepts of Primitive Data Types and Structural types in JSON

5. Highlight the impacts of Data Exchange Formats with suitable example

References:

1. Mischa Dohler, Carles Anton-Haro, "Machine-to-machine (M2M) Communications", Woodhead Publishing, 1st Edition, 2014.
2. Perry Lea, "Internet of Things for Architects: Architecting IoT solutions by implementing sensors, communication infrastructure, edge computing, analytics, and security", Packt, 1st Edition, 2018.
3. Maciej Kranz , "Building the Internet of Things", John Wiley & Sons, 1st Edition, 2017.
4. David Boswarthick, Omar Elloumi, Olivier Hersent, "M2M Communications: A Systems Approach", John Wiley & Sons, 1 st Edition, 2012.
5. Anupama C. Raman and Pethuru Raj, "The Internet of Things: Enabling Technologies, Platforms, and Use Cases", CRC press, 1st Edition, 2017.

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF ELECTRICAL AND ELECTRONICS

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

UNIT – III – MQTT and COAP Protocols – SECA7019

UNIT 3 MQTT AND COAP PROTOCOLS

IoT protocols, Introduction to MQTT, MQTT – packet format, broker, topics, QOS Levels, LWT, keep Alive and Client take over, security, Connectivity to IoT cloud Platform through MQTT, MQTT over Web sockets, CoAP overview, Client – server implementation through CoAP Protocol, Resource registration and discovery, Use case of CoAP, CoAPvs MQTT.

3.1 Introduction:

We have discussed various telecommunication mediums and technologies to move that data from the WPAN, WLAN, and WAN. There are many complexities and subtleties in building and bridging these network connections from non-IP-based PAN networks to IP-based WAN networks. There are also protocol conversions that need to be understood. Standard protocols are the tools that bind and encapsulate raw data from a sensor to something meaningful and formatted for the cloud to accept. One of the principal differences between an IOT system and M2M system is that M2M may communicate over a WAN to a dedicated server or system with no encapsulated protocol whatsoever. The IoT by definition is based on communication between endpoints and services with the internet being the common network fabric.

IOT Protocols:

A natural question is, why are there any protocols outside of HTTP to transport data across the WAN? HTTP has provided significant services and abilities for the internet for over 20 years, yet was designed and architected for general purpose computing in client/server models. IoT devices can be very constrained, remote, and bandwidth limited. Therefore, more efficient, secure, and scalable protocols are necessary to manage a plethora of devices in various network topologies such as mesh networks.

In transporting data to the internet, designs are relegated to the TCP/IP foundation layers. TCP and UDP protocols are the obvious and only choice in data communication with TCP being significantly more complex in its implementation than UDP (being a multicast protocol). UDP, however, does not have the stability and reliability of TCP, forcing some designs to compensate by adding resiliency in the application layers above UDP. Many of the protocols listed in this chapter are Message Orientated Middleware (MOM) implementations.

The basic idea of a MOM is that communication between two devices takes places using distributed message queues. A MOM delivers messages from one userspace application to another. Some devices produce data to be added to a queue while others consume data stored in a queue. Some implementations require a broker or middleman be the central service. In that case, producers and consumers have a publish and subscribe-type relationship with the broker. AMQP, MQTT, and STOMP are MOM implementations; others include CORBA and Java messaging services. A MOM implementation using queues can use them for resilience in design. Data can persist in the queues, even if the server fails.

The alternative to MOM implementation is RESTful. In a RESTful model, a server owns the state of a resource but the state is not transferred in a message to the server from the client. RESTful designs use HTTP methods such as GET, PUT, POST, and DELETE to place requests upon a resource's Universal Resource Identifier (URI) (see the following figure). No broker or middle agent is needed in this architecture. As they are based on the HTTP stack, they enjoy most services offered, such as HTTPS security.

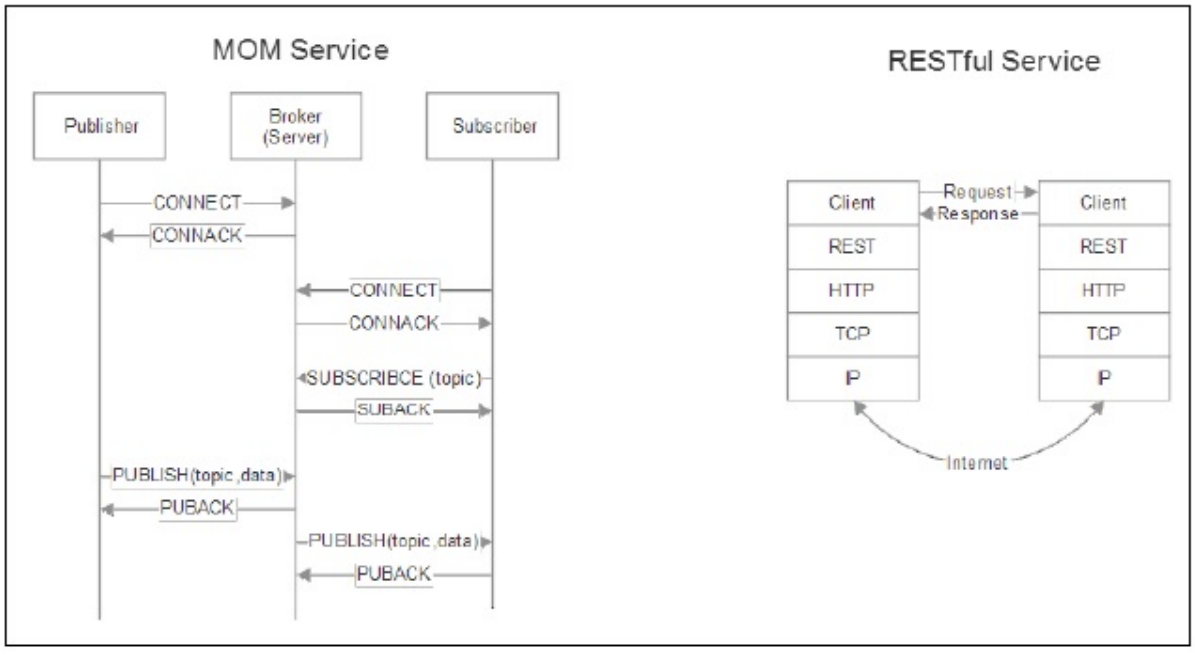


Figure 3.1: An example comparing MOM and Restful Service

RESTful designs are typical of client-server architectures. Clients initiate access to resources through synchronous request-response patterns. Additionally, clients are

responsible for errors, even if the server fails. The figure below illustrates a MOM versus a RESTful service. On the left is a messaging service (based on MQTT) using a middle broker server and publishers and subscribers to an event. Here many clients can be both publishers and subscribers and information may or may not be stored in queues for resiliency. On the right is a RESTful design where the architecture is built upon HTTP and uses HTTP paradigms for communicating from the client to the server.

MQTT:

The IBM Websphere Message Queue technology was first conceived in 1993 to address problems in independent and non-concurrent distributed systems to securely communicate. A derivative of the WebSphere Message Queue was authored by Andy Stanford-Clark and Arlen Nipper at IBM in 1999 to address the constraints of connecting remote oil and gas pipelines over a satellite connection. That protocol became known as the MQTT.

The goals of this IP-based transport protocol are:

- It must be simple to implement
- To provide a form of quality of service
- To be very lightweight and bandwidth efficient
- To be data agnostic
- To have continuous session awareness
- To address security issues

MQTT was an internal and proprietary protocol for IBM for many years until being released in version 3.1 in 2010 as a royalty-free product. In 2013, MQTT was standardized and accepted into the OASIS consortium. In 2014, OASIS released it publicly as version MQTT 3.1.1. MQTT is also an ISO standard (ISO/IEC PRF 20922).

MQTT publish-subscribe:

While client-server architectures have been the mainstay for data center services for years, publish-subscribe models represent an alternative that is useful for IoT uses. Publishsubscribe, also known as pub/sub, is a way to decouple a client transmitting a message from another client receiving a message. Unlike a traditional client-server model, the clients are not aware of any physical identifiers such as the IP address or port. MQTT is a pub/sub architecture but is not a message queue. Message queues by nature store messages while MQTT does not. In MQTT, if no one is subscribing (or listening) to a topic, it is simply ignored and lost. Messages queues also maintain a client-server topology where one consumer is paired with one producer.

A client transmitting a message is called a publisher; a client receiving a message is called a subscriber. In the center is an MQTT broker who has the responsibility of connecting clients and filtering data. Such filters provide:

- Subject filtering: By design, clients subscribe to topics and certain topic branches and do not receive more data than they want. Each published message must contain a topic and the broker is responsible for either re-broadcasting that message to subscribed clients or ignoring it.
- Content filtering: Brokers have the ability to inspect and filter published data. Thus, any data not encrypted can be managed by the broker before being stored or published to other clients.
- Type filtering: A client listening to a subscribed data stream can apply their own filters as well. Incoming data can be parsed and the data stream either processed further or ignored.

MQTT may have many producers and many consumers, as shown in the following figure:

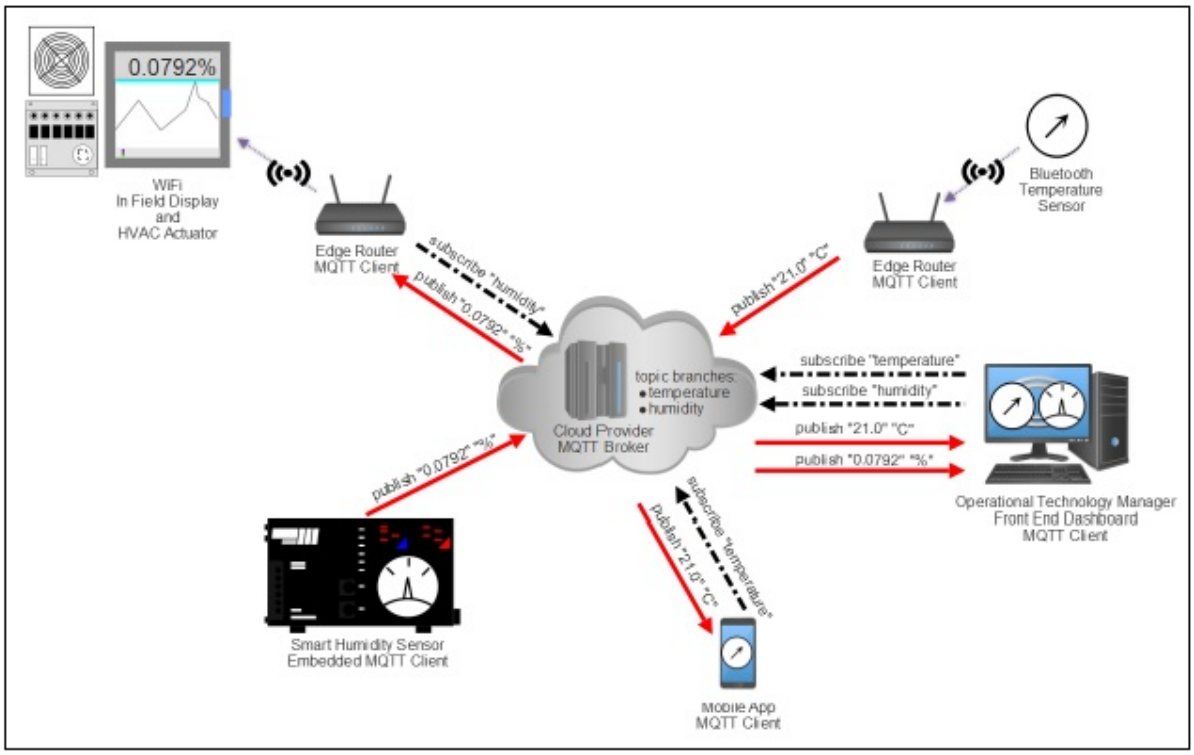


Figure 3.2: MQTT Publish-Subscribe Topology

MQTT successfully decouples publishers from consumers. Since the broker is the governing body between publishers and consumers there is no need to directly identify a publisher and consumer based on physical aspects (such as an IP address). This is very useful in IoT deployments as the physical identity may be unknown or ubiquitous. MQTT and other pub/sub models are also time-invariant. This implies that a message published by one client may be read and responded to at any time by a subscriber. A subscriber could be in a very low power/bandwidth-limited state (for example, Sigfox communication) and respond minutes or hours later to a message. Because of the lack of physical and time relationships, pub/sub models scale very well to extreme capacity.

Cloud-managed MQTT brokers typically can ingest millions of messages per hour and support tens of thousands of publishers. MQTT is data format agnostic. Any type of data may reside in the payload which is why both the publishers and subscribers must both understand and agree to the data format. It is possible to transmit text messages, image data, audio data, encrypted data, binary data, JSON objects, or virtually any other structure in the payload.

JSON text and binary data, however, are the most common data payload types. The maximum allowable packet size in MQTT is 256 MB, which allows for am extremely large payload.

MQTT architecture:

MQTT by name is a misnomer. There are no message queues inherent in the protocol. While it is possible to queue messages, it's not necessary and often not done. MQTT is TCP based, and therefore includes some guarantee that a packet is transferred reliably. MQTT is also an asymmetric protocol, whereas HTTP is a nonsymmetric protocol. Say node A needs to communicate with node B.

An asymmetric protocol between A and B would only require one side (A) to use the protocol, however, all information needed for the reassembly of packets must be contained in the fragmentation header sent by A. Asymmetric systems have one master and one slave (FTP being a classic example). In a symmetric protocol, both A and B would have the protocol installed. A or B can assume the role of master or slave (telnet being a primary example). MQTT has distinct roles which makes sense in sensor/cloud topology.

MQTT can retain a message on a broker indefinitely. This mode of operation is controlled by a flag on a normal message transmission. A retained message on a broker is sent to any client that subscribes to that MQTT topic branch. The message is transmitted immediately to that new client. This allows a new client to receive a status or signal from a newly subscribed topic without waiting. Typically, a client subscribing to a topic may need to wait hours, or even days, before a client publishes new data.

MQTT defines an optional facility called Last Will and Testament (LWT). An LWT is a message a client specifies during the connect phase. The LWT contains the Last Will topic, QoS, and the actual message. If a client disconnects from a broker connection ungracefully (for example, keep-alive timeout, I/O error, or the client closing the session without a disconnect), the broker is then obligated to broadcast the LWT message to all other subscribed clients to that topic. Even though MQTT is based on TCP, connections can still be lost, especially in a wireless sensor environment.

A device may lose power, lose signal strength, or simply crash in the field, and a session will enter a half-open state. Here, the server will believe the connection is still reliable and expect data. To remedy this half-open state, MQTT uses a keep-alive system. Using this system, both the MQTT broker and client have assurance that the connection is still valid even if there hasn't been a transmission for some time. The client sends a PINGREQ packet to the broker, which in turn acknowledges the message with a PINGRESP. A timer is preset on the client and broker side. If a message has not been transmitted from either within a predetermined time limit, a keep-alive packet should be sent. Both a PINGREQ or a message will reset the keep-alive timer. If a keep-alive is not received and the timer expires, the broker will close the connection and send out the LWT packet to all clients. The client may at some point later attempt to reconnect. In that case, a half-open connection will be closed by the broker and open up a new connection to the client.

While keep-alive assists with broken connections, re-establishing all of a client's subscriptions and QoS parameters can result in unnecessary overhead on a data-capped connection. To alleviate this extra data, MQTT allows for persistent connections. A persistent connection will save the following on the broker side:

- All client's subscriptions
- All QoS messages that were not confirmed by the client
- All new QoS messages missed by the client

This information is referenced by the client_id parameter to identify unique clients. A client can request a persistent connection, however, the broker can deny the request and force a clean session to restart. Upon connection, the cleanSession flag is used by the broker to allow or deny persistent connections. The client can determine if a persistent connection was stored using the CONNACK message.

There are three levels of quality of service in MQTT:

- QoS-0 (non-assured transmission): This is the minimal QoS level. This is analogous to a fire-and-forget model detailed in some of the wireless protocols. It is a best-effort delivery process without the receiver acknowledging a message or the sender reattempting transmission.

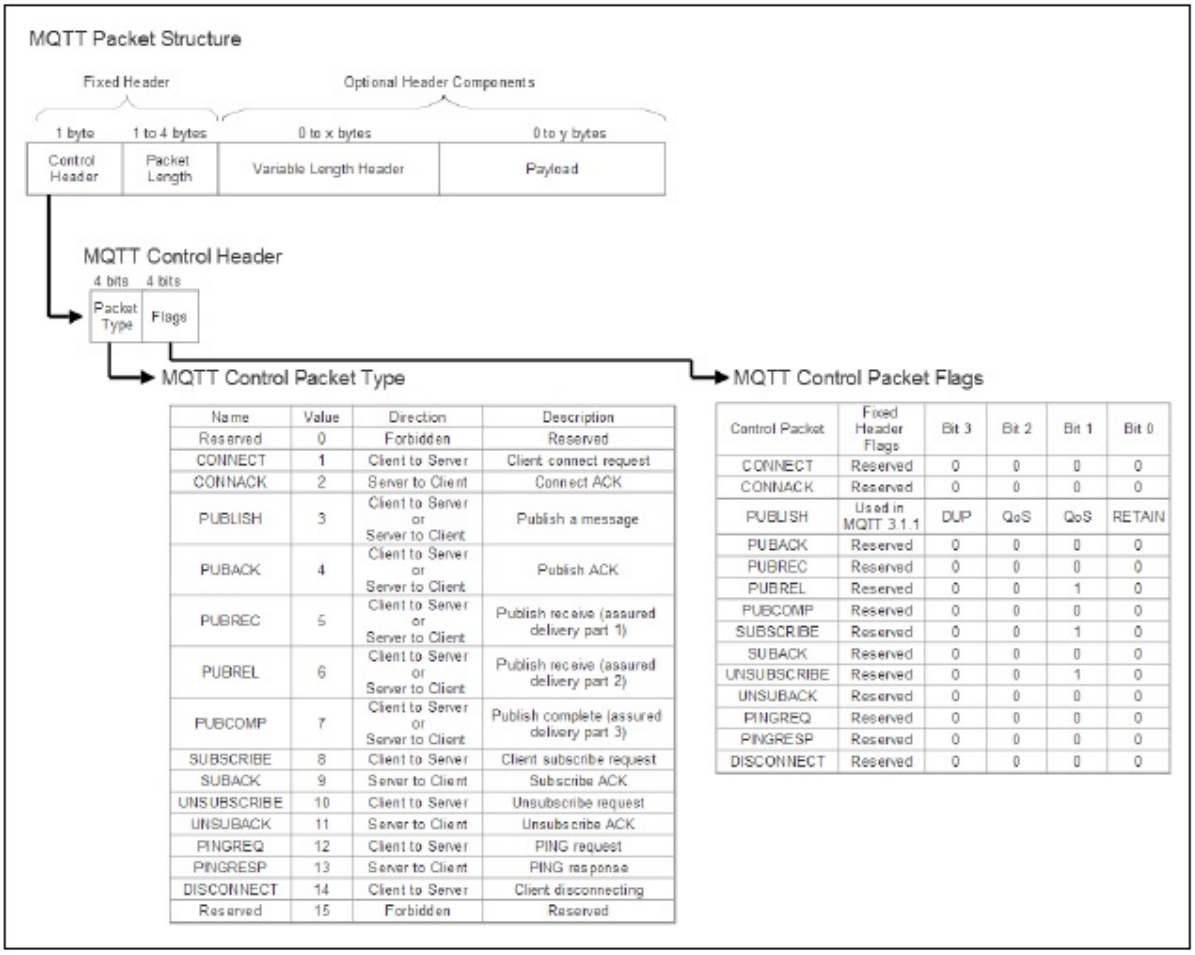
- Qos-1 (assured transmission): This mode will guarantee delivery of the message at least once to the receiver. It may be delivered more than once, and the receiver will send an acknowledgment back with a PUBACK response.
- QoS-2 (assured service on applications): This is the highest level of QoS that ensures and informs both the sender and receiver that a message has been transmitted correctly. This mode generates more traffic with a multi-step handshake between the sender and receiver. If a receiver gets a message set to QoS-2, it will respond with a PUBREC message to the sender. This acknowledges the message and the sender will respond with a PUBREL message. PUBREL allows the receiver to safely discard any re-transmissions of the message. The PUBREL is then acknowledged by the receiver with a PUBCOMP. Until the PUBCOMP message is sent, the receiver will cache the original message for safety.

QoS in MQTT is defined and controlled by the sender and each sender can have a different policy. Typical use cases:

- QoS-0: Should be used when message queuing isn't needed. It is best for wired connections or when the system is severely constrained on bandwidth.
- Qos-1: This should be the default usage. QoS1 is much faster than QoS2 and greatly reduces transmission cost.
- QoS-2: For mission-critical applications. Also, cases where retransmission of duplicate message could cause faults.

MQTT packet structure:

An MQTT packet sits on top of the TCP layer of the OSI model network stack. The packet consists of a 2 byte fixed header, which must always be present, a variably sized header (optional) and concludes with the payload (again, optional):



Name	Value	Direction	Description
Reserved	0	Forbidden	Reserved
CONNECT	1	Client to Server	Client connect request
CONNACK	2	Server to Client	Connect ACK
PUBLISH	3	Client to Server or Server to Client	Publish a message
PUBACK	4	Client to Server or Server to Client	Publish ACK
PUBREC	5	Client to Server or Server to Client	Publish receive (assured delivery part 1)
PUBREL	6	Client to Server or Server to Client	Publish receive (assured delivery part 2)
PUBCOMP	7	Client to Server or Server to Client	Publish complete (assured delivery part 3)
SUBSCRIBE	8	Client to Server	Client subscribe request
SUBACK	9	Server to Client	Subscribe ACK
UNSUBSCRIBE	10	Client to Server	Unsubscribe request
UNSUBACK	11	Server to Client	Unsubscribe ACK
PINGREQ	12	Client to Server	PING request
PINGRESP	13	Server to Client	PING response
DISCONNECT	14	Client to Server	Client disconnecting
Reserved	15	Forbidden	Reserved

Control Packet	Fixed Header Flags	Bit 3	Bit 2	Bit 1	Bit 0
CONNECT	Reserved	0	0	0	0
CONNACK	Reserved	0	0	0	0
PUBLISH	Used in MQTT 3.1.1	DUP	QoS	QoS	RETAIN
PUBACK	Reserved	0	0	0	0
PUBREC	Reserved	0	0	0	0
PUBREL	Reserved	0	0	1	0
PUBCOMP	Reserved	0	0	0	0
SUBSCRIBE	Reserved	0	0	1	0
SUBACK	Reserved	0	0	0	0
UNSUBSCRIBE	Reserved	0	0	1	0
UNSUBACK	Reserved	0	0	0	0
PINGREQ	Reserved	0	0	0	0
PINGRESP	Reserved	0	0	0	0
DISCONNECT	Reserved	0	0	0	0

Figure 3.3 MQTT Packet Structure

MQTT communication formats:

A communication link using MQTT starts with the client sending a CONNECT message to a broker. Only a client can initiate a session and no client may communicate directly with another client. The broker will always respond back to a CONNECT message with a CONNACK response and status code. Once established the connection remains open. The following are the MQTT messages and formats:

- CONNECT format (client to server): A typical CONNECT message will contain the following (only the clientID is required to initiate a session):

Field	Requirement	Description
clientID	Required	Identifies client to the server. Each client has a unique client ID. It can be between 1 and 23 UTF-8 bytes long.
cleanSession	Optional	0: Server must resume communications with the client. Client and server must save session state after disconnect. 1: Client and server must discard and the previous session and start a new one.
username	Optional	Name used by the server for authentication.
password	Optional	0 to 65536 byte binary password prefixed by 2 bytes of length.
lastWillTopic	Optional	Topic branch to publish will message.
lastWillQos	Optional	2 bits specifying QoS level when publishing will message.
lastWillMessage	Optional	Defines the Will message payload.
lastWillRetain	Optional	Specifies if the Will is to be retained when published.
keepAlive	Optional	Time interval in seconds. The client is responsible for sending a message or a PINGREQ packet before the keepAlive timer expires. The server will disconnect from the network at 1.5x the Keep Alive period. A value of zero (0) will disable the keepAlive mechanism.

- CONNECT return codes (server to client): The broker will respond to a CONNECT message with a response code. A designer should be aware that not all connections may be approved by a broker.
- PUBLISH format (client to server): At this point, a client may publish data to a topic branch.
- SUBSCRIBE format (client to server): The payload of a subscribe packet includes at least one pair of UTF-8 encoded topicIDs and QoS levels.

Wildcards can be used to subscribe to multiple topics in a single message. For these examples, the topic will be the full path of

"{country}/{states}/{cities}/{temperature,humidity}".

+ Single level wildcard: Substitutes a single level in a topic string name. For example, US/+/Milwaukee will substitute the state level and replace it with all 50 states, Alaska to Wyoming.

* Multi-level wildcard: Replaces multiple levels rather than a single level. It is always the last character in the topic name. For example, US/Wisconsin/# will subscribe to all Wisconsin cities: Milwaukee, Madison, Glendale, Whitefish Bay, Brookfield, and so on.

$ Special topics: This is a special statistical mode for MQTT brokers. Clients cannot publish to $ topics. There is no official standard for use at the moment. One model uses $SYS in the following manner: $SYS/broker/clients/connected.

MQTT-SN:

A derivative of MQTT is called MQTT-SN (sometimes called MQTT-S) for sensor networks. It keeps to the same philosophy of MQTT as a lightweight protocol for edge devices but is architected specifically for the nuances of a wireless personal area network typical in sensor environments. These traits include support for low-bandwidth links, link failure, short message length, and resource-constrained hardware. MQTT-SN is, in fact, so light that it can be run successfully over BLE and Zigbee. MQTT-SN does not require TCP/IP stack. It can be used over a serial link (preferred way), where a simple link protocol (to distinguish different devices on the line) overhead is really small. Alternatively it can be used over UDP, which is less hungry than TCP.

MQTT-SN architecture and topology :

There are four fundamental components in an MQTT-SN topology:

- Gateways: In MQTT-SN, a gateway has the responsibility of protocol conversion from MQTT-SN to MQTT and vice versa (although other translations are possible). Gateways can also be aggregating or transparent (covered later in this chapter).
- Forwarders: A route between a sensor and an MQTT-SN gateway may take many paths and hop across several routers along the way. Nodes between the source client and the MQTT-SN gateway are called forwarders and simply reencapsulate

MQTT-SN frames into new and unchanged MQTT-SN frames that are sent to the destination until they arrive at the correct MQTT-SN gateway for protocol conversion.

- Clients: Clients behave in the same way as in MQTT and are capable of subscribing and publishing data.
- Brokers: Brokers behave in the same way as in MQTT

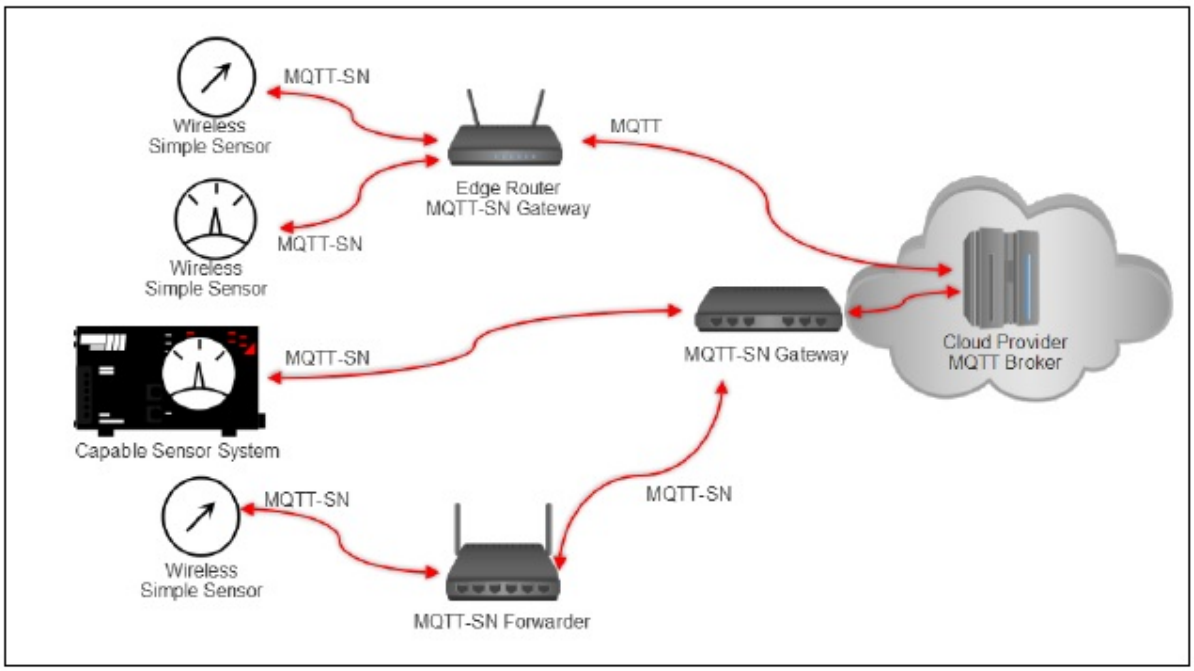


Figure 3.4: MQTT SN Topology

Transparent and aggregating gateways:

In MQTT-SN, gateways can take on two distinct roles. First, a transparent gateway will manage many independent MQTT-SN streams from sensor devices and convert each stream into an MQTT message. An aggregating gateway will coalesce a number of MQTTSN streams into a reduced number of MQTT streams sent to the cloud MQTT broker. An aggregating gateway is more complex in design but will reduce the amount of communication overhead and number of simultaneous connections left open on the server. For an aggregating gateway topology to function, the clients need to publish or subscribe to the same topic.

Gateway advertisement and discovery:

Since the topology of MQTT-SN is slightly more complicated than MQTT, a discovery process is used to establish routes through multiple gateways and forwarder nodes. Gateways joining an MQTT-SN topology start by binding to an MQTT broker first. Afterwards, it may issue an ADVERTISE packet to connected clients or other gateways. Multiple gateways can exist on the network, but a client may only connect to a single gateway. The client has the responsibility of storing a list of active gateways and their network addresses. This list is constructed from the various ADVERTISEMENT and GWINFO messages being broadcast. Because of the new types of gateways and topologies in MQTT-SN, several new messages are available to assist in discovery and advertising:

- ADVERTISE: Broadcast periodically from a gateway to advertise its presence.
- SEARCHGW: Broadcast by a client when searching for a particular gateway. Part of the message is a radius parameter, which states how many hops the SEARCHGW message should follow in a network topology. For example, a value of 1 indicates a single hop in a very dense network where every client is reachable with a single hop.
- GWINFO: This is the response by gateways upon receiving a SEARCHGW message. It contains the gateway ID and gateway address, which is only broadcast when the SEARCHGW is sent from a client.

Differences between MQTT and MQTT-SN:

The main differences between MQTT-SN and MQTT are as follows:

- There are three CONNECT messages in MQTT-SN versus one in MQTT. The additional two are used to transport the Will topic and Will message explicitly.
- MQTT-SN can run over simplified medium and UDP.
- Topic names are replaced by short, two-byte long topic ID messages. This is to assist with bandwidth constraints in wireless networks.

- Pre-defined topic IDs and short topic names can be used without any registration. To use this feature, both the client and server need to use the same topic ID. Short topic names are short enough to be embedded in the PUBLISH message.
- A discovery procedure is introduced to assist clients and to allow them to find the network addresses of servers and gateways. Multiple gateways may exist in the topology and can be used to load share communication with clients.
- The cleanSession is extended to the Will feature. A client subscription can be retained and preserved, but now the Will data is also preserved.
- A revised keepAlive procedure is used in MQTT-SN. This is to support sleeping clients in which all messages destined for them are buffered by a server or edgerouter and delivered upon awakening.

Constrained Application Protocol:

The Constrained Application Protocol (CoAP) is the product of the IETF (RFC7228). The IETF Constrained RESTful Environments (CoRE) working group created the first draft of the protocol in June 2014 but had worked for several years on its creation. It is specifically intended as a communication protocol for constrained devices. The core protocol is now based on RFC7252. The protocol is unique as it is first tailored for machine-to-machine (M2M) communication between edge nodes. It also supports mapping to HTTP through the use of proxies. This HTTP mapping is the on-board facility to get data across the internet. CoAP is excellent at providing a similar and easy structure of resource addressing familiar to anyone with experience of using the web but with reduced resources and bandwidth demands.Additionally, some implementations of CoAP perform up to 64x better than HTTP equivalents on similar hardware.

CoAP architecture:

CoAP is based on the concept of mimicking and replacing heavy HTTP abilities and usage with a lightweight equivalent for the IoT. It is not an HTTP replacement as it does lack features; HTTP requires more powerful and service-orientated systems.

CoAP features can be summarized as follows:

- HTTP-like
- Connection-less protocols
- Security through DTLS rather than TLS in a normal HTTP transmission
- Asynchronous message exchanges
- Lightweight design and resource requirements and low header overhead
- Support for URI and content-types
- Built upon UDP versus TCP/UDP for a normal HTTP session
- A stateless HTTP mapping allowing for proxies top bridge to HTTP sessions

CoAP has two basic layers:

- Request/Response layer: Responsible for sending and receiving RESTful-based queries. REST queries are piggybacked on CON or NON messages. A REST response is piggybacked on the corresponding ACK message.
- Transactional layer: Handles single message exchanges between endpoints using one of the four basic message types. The transaction layer also supports multicasting and congestion control.

CoAP shares context, syntax, and usage similarly to HTTP. Addressing in CoAP is also styled like HTTP. An address extends to the URI structure. As in HTTP URIs, the user must know the address beforehand to gain access to a resource. At the top-most level, CoAP uses requests such as GET, PUT, POST, and DELETE, as in HTTP.

Similarly, response codes mimic HTTP, such as:

- 2.01: Created
- 2.02: Deleted
- 2.04: Changed
- 2.05: Content
- 4.04: Not found (resource)
- 4.05: Method not allowed

The form of a typical URI in CoAP would be:

coap://host[:port]/[path][?query]

A CoAP system has seven main actors:

- Endpoints: These are the sources and destinations of a CoAP message. The specific definition of an endpoint depends on the transport being used.
- Proxies: A CoAP endpoint that is tasked by CoAP clients to perform requests on its behalf. Reducing network load, access sleeping nodes, and providing a layer of security are some of the roles of a proxy. Proxies can be explicitly selected by a client (forward-proxying) or can be used as in-situ servers (reverse-proxying). Alternatively, a proxy can map from one CoAP request to another CoAP request or even translate to a different protocol (cross-proxying). A common situation is an edge router proxying from a CoAP network to HTTP services for cloud-based internet connections.
- Client: The originator of a request. The destination endpoint of a response.
- Server: The destination endpoint of a request. The originator of a response.
- Intermediary: A client acting as both a server and client towards an origin server. A proxy is an intermediary.
- Origin servers: The server on which a given resource resides.
- Observers: An observer client can register itself using a modified GET message. The observer is then connected to a resource and if the state of that resource changes, the server will send a notification back to the observer.

Below is an example of CoAP architecture. Being a lightweight HTTP system allows for CoAP clients to communicate to one another or services in the cloud supporting CoAP. Alternatively, a proxy can be used to bridge to an HTTP service in the cloud. CoAP endpoints can establish relationships with each other, even at the sensor level. Observers allow for subscription-like attributes to resource that change in a similar manner to MQTT. The graphic also illustrates origin servers holding the resource being shared.

The two proxies allow for CoAP to perform HTTP translation or allow forwarding requests on behalf of a client.

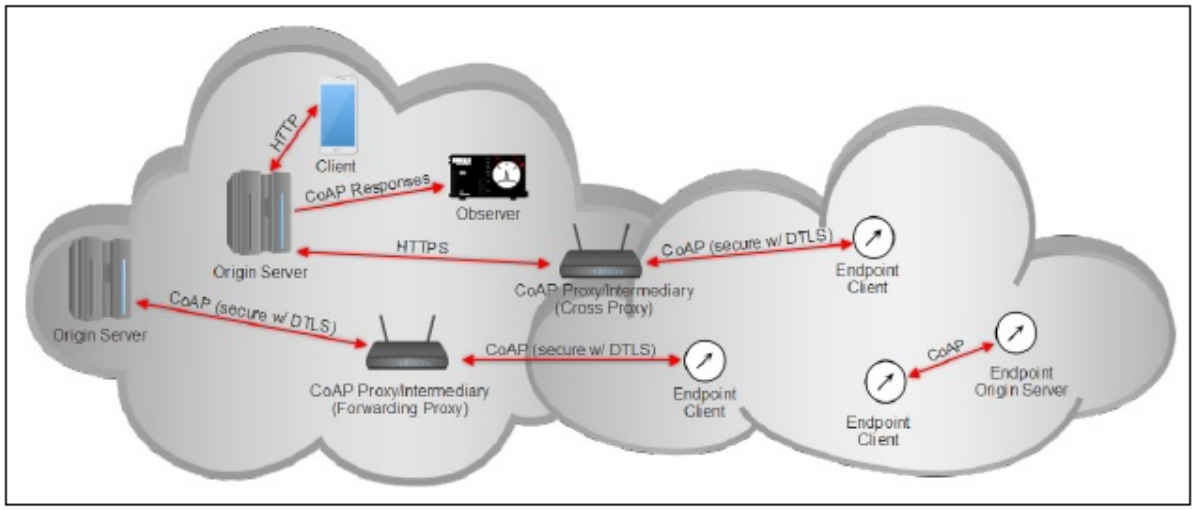


Figure 3.5: CoAP Architecture

CoAP Messaging Formats:

Protocols based on UDP transport imply that the connection may not be inherently reliable. To compensate for reliability issues, CoAP introduces two message types that differ by either requiring acknowledgment or not. An additional feature of this approach is that messages can be asynchronous.

In total there are only four messages in CoAP:

- Confirmable (CON): Requires an ACK. If a CON message is sent an ACK must be received within a random time interval between ACK_TIMEOUT and (ACK_TIMEOUT * ACK_RANDOM_FACTOR). If the ACK is not received, the sender transmits the CON message over and over at exponentially increasing intervals until it receives the ACK or a RST. This essentially is the CoAP form of congestion control. There is a maximum number of attempts set by MAX_RETRANSMIT. This is the resiliency mechanism to compensate for the lack of resiliency in UDP.

- Non-confirmable (NON): Requires no ACK. Essentially a fire-and-forget message or broadcast.
- Acknowledgement (ACK): Acknowledges a CON message. The ACK message can piggyback along with other data.
- Reset (RST): Indicates that a CON message has been received but the context is missing. The RST message can piggyback along with other data.

CoAP is a RESTful design using request/response messages piggybacked on CoAP messages.The graphic shows three examples of CoAP non-confirmable and confirmable request/response transactions. These are enumerated and described as follows:

- Non-confirmable request/response (left): A message broadcast between client A and B using the typical HTTP GET construct. B reciprocates with Content data sometime later and returns the temperature of 20 degrees celsius.
- Confirmable request/response (middle): Included is the Message-ID, a unique identifier for each message. The Token represents a value that must match during the duration of the exchange.
- Confirmable request/response (right): Here the message is confirmable. Both client A and B will wait for an ACK after each message exchange. To optimize communication, Client B can elect to piggyback the ACK with the returned data as shown in the far right.

While other messaging architecture requires a central server to propagate messages between clients, CoAP allows messages to be sent between any CoAP client including sensors and servers. CoAP includes a simple caching model. Caching is controlled through response codes in the message header. An option number mask will determine if it is a cache key. The Max_Age option is used to control cache element lifetimes and ensure freshness of the data. That is, Max_Age sets the maximum time a response can be cached for before it must be refreshed. Max_Age defaults to 60 seconds and can span up to 136.1 years.

Proxies play a role in caching; for example, a sleeping edge sensor may use a proxy to cache data and to preserve power. The CoAP message header is uniquely designed for maximum efficiency and bandwidth preservation. The header is four bytes in length with a typical request message taking only 10 to 20-byte headers. This is typically 10x smaller than an HTTP header.

The structure consists of Message Type Identifiers (T), which must be included in each header along with an associated unique Message-ID. The Code field is used to signal errors or success states across channels. After the header, all other fields are optional and include variable length Tokens, Options, and Payload.

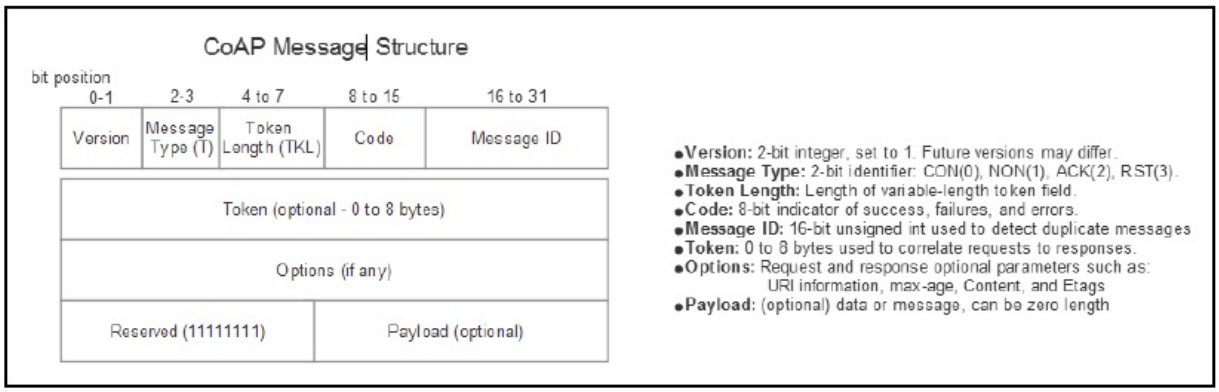


Figure 3.6: CoAP Message Structure

UDP also can cause duplicate messages to arrive for both CON and NON transmissions. If identical Message_IDs are delivered to a recipient within the prescribed EXCHANGE_LIFETIME, a duplicate is said to exist. This clearly can occur, as shown in the previous figures, when an ACK is missing or dropped and the client retransmits the message with the same Message_ID. The CoAP specification states the recipient should ACK each duplicate message it receives but should only process one request or response. This rule may be relaxed if a CON message transports a request that is idempotent.

As mentioned, CoAP allows for the role of an observer in a system. This is unique as it allows CoAP to behave in a similar manner to MQTT. The observation process allows a client to register for observation and the server will notify the client whenever the resource being monitored changes state. The duration of observation can be defined during registration. Additionally, the observation relationship ends when the initiating client sends a RST or another GET message:

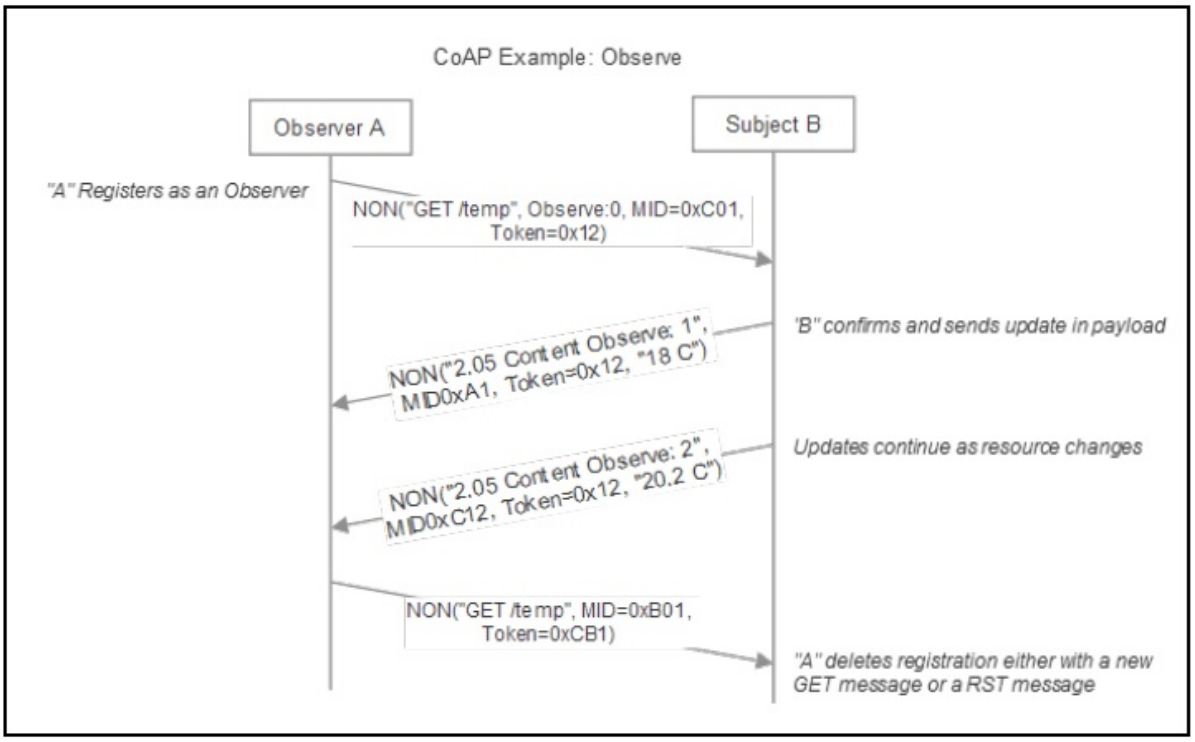


Figure 3.7: CoAP Observer registration and update process

As mention previously, there is no inherent authentication or encryption in the CoAP standard, rather, the user must rely on DTLS to provide that level of security.

If DTLS is used then an example URI would be:

//insecure coap://example.net:1234/~temperature/value.xml

//secure coaps://example.net:1234/~temperature/value.xml

CoAP offers resource discovery mechanisms as well. Simply sending a GET request to /.well-known/core will disclose a list of known resources on the device. Additionally, query strings can be used in the request for applying specific filters.

Other protocols:

There are many other messaging protocols in use or proposed for IoT and M2M deployments. By far the most prevalent are MQTT and CoAP.

STOMP: STOMP stands for Simple (or Streaming) Text Message-Oriented Middleware Protocol. It is a text-based protocol designed by Codehaus to operate with message-oriented middleware. A broker developed in one programming language can receive messages from a client written in another. The protocol has similarities to HTTP and operates over TCP. STOMP consists of a frame header and a frame body. The current specification is STOMP 1.2, dated October 22, 2012, and is available under a free license. It is different from many protocols presented as it does not deal with subscription topics or queues. It simply uses HTTP-like semantics such as SEND with a destination string. A broker must dissect the message and map to a topic or queue for the client. A consumer of the data will SUBSCRIBE to the destinations provided by the broker. STOMP has clients written in python (Stomp.py), TCL (tStomp), and Erlang (stomp.erl). Several servers have native STOMP support, such as RabbitMQ (via a plugin), and some servers have been designed in specifics languages (Ruby, Perl, or OCaml).

AMQP: AMQP stands for Advanced Message Queuing Protocol. It is a hardened and proven MOM protocol used by massive data sources such as JP Morgan Chase in processing over 1 billion messages per day, and the Ocean Observatory Initiative in collecting over 8 terabytes of oceanographic data each day. It was originally devised at JP Morgan Chase in 2003, who led the creation of a working group of 23 companies in 2006 chartered with the architecture and governance of the protocol. In 2011, the working group was merged into the OASIS group where it is currently hosted.

Today it is well established in banking and credit transaction industries but has a place in IoT as well. Furthermore, AMQP is standardized by ISO and IEM as ISO/IEC 19464:2014. A formal AMQP working group can be found at www.amqp.org. The AMQP protocol resides on top of the TCP stack and uses port 5672 for communication. Data is serialized over AMQP, meaning that messages are broadcast in unit frames. Frames are transmitted in virtual channels identified with a unique channel_id. Frames consist of headers, channel_ids, payload information, and footers.

A channel, however, can only be associated with a single host. Messages are assigned a unique global identifier. AMQP is a flow-controlled, message-orientated communication system. It is a wire-level protocol and a low-level interface. A wire protocol refers to APIs

immediately above the physical layer of a network. A wire-level API allows for different messaging services such as .NET (NMS) and Java (JMS) to communicate with each other. Similarly, AMQP attempts to decouple publishers from subscribers. Unlike MQTT it has mechanisms for load balancing and formal queuing. A well-used protocol that is based on AMQP is RabbitMQ. RabbitMQ is an AMQP message broker written in Erlang. Additionally, several AMQP clients are available, such as the RabbitMQ client written in Java, C#, Javascript, and Erlang, as well as Apache Qpid written in Python, C++, C#, Java, and Ruby. One or more virtual hosts with their own namespaces, exchanges, and message queues will reside on a central server(s).

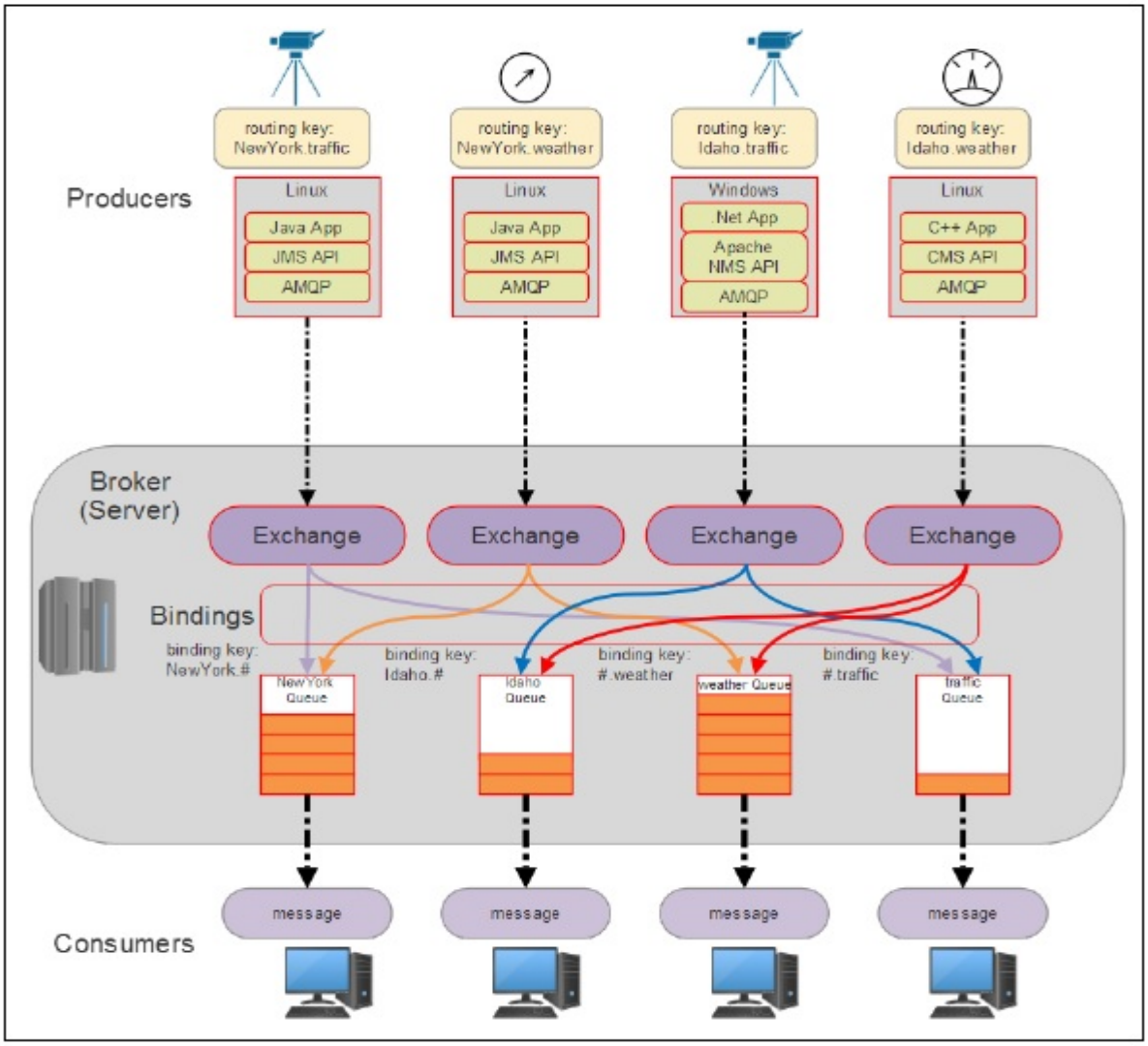


Figure 3.8: AMQP Architecture Topology

Producers and consumers subscribe to the exchange service. The exchange service receives messages from a publisher and routes the data to an associated queue. This relationship is called a binding and the binding can either be direct to one queue or fanned out to multiple queues (as in a broadcast). Alternatively, the binding can associate one exchange with one queue using a routing key; this is formally called a direct exchange. Another type of exchange is the topic exchange.

The network topology of an AMQP deployment is hub-and-spoke with the ability for hubs to communicate with one another. AMQP consists of nodes and links. A node is a named source or a sink of messages. A message frame moves between nodes across unidirectional links. If a message is passed through a node, the global identifier is unchanged. If the Node performs any transformation, a new ID is assigned. A link can filter messages. There are three distinct messaging patterns that can be used in AMQP:

- Asynchronous Directed Messages: Message is transmitted without requiring a receiver acknowledgment.
- Request/Reply or Pub/Sub: Like MQTT with a central server acting as a pub/sub service.
- Store and Forward: This is used for hub relaying, where a message is sent to an intermediate hub and then sent on towards its destination.

Questions to Practice:

Part-A

1. Determine the IoT Protocols

2. Explain the features of MQ Telemetry Transport

3. Justify the merits and demerits of Connectivity to IoT Cloud Platform

4. Give reasons in detail about Web Sockets

5. Infer the principle of Constrained Application Protocol

Part-B

1. Evaluate a MQ Telemetry Transport by considering packet format, broker topics and QOS Levels

2. Validate Client – server implementation through CoAP Protocol

3. Determine an algorithm for solving MQTT by using Keep Alive, Client take over and Security

4. Evaluate a Resource registration and discovery using CoAP Protocol

References:

1. Mischa Dohler, Carles Anton-Haro, "Machine-to-machine (M2M) Communications", Woodhead Publishing, 1st Edition, 2014.
2. Perry Lea, "Internet of Things for Architects: Architecting IoT solutions by implementing sensors, communication infrastructure, edge computing, analytics, and security", Packet, 1st Edition, 2018.
3. Maciej Kranz , "Building the Internet of Things", John Wiley & Sons, 1st Edition, 2017.
4. David Boswarthick, Omar Elloumi, Olivier Hersent, "M2M Communications: A Systems Approach", John Wiley & Sons, 1 st Edition, 2012.
5. Anupama C. Raman and Pethuru Raj, "The Internet of Things: Enabling Technologies, Platforms, and Use Cases", CRC press, 1st Edition, 2017.

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF ELECTRICAL AND ELECTRONICS

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

UNIT – IV – HTTP REST and WEBSOCKETS Protocols – SECA7019

UNIT 4 HTTP REST AND WEBSOCKETS PROTOCOLS

Introduction to HTTP, HTTP REST model, CRUD operations, connectivity to IoT cloud platforms using HTTP REST Services, Introduction to Websocket, use cases of Websocket, Websocket vs HTTP REST, Significance of gateway design, characteristics, Protocol bridging, implementation and case studies.

4.1 Introduction to HTTP:

HTTP (Hypertext Transfer Protocol) is perhaps the most popular application protocol used in the Internet (or The WEB). HTTP is an asymmetric request-response client-server protocol as illustrated. An HTTP client sends a request message to an HTTP server. The server, in turn, returns a response message. In other words, HTTP is a pull protocol, the client pulls information from the server (instead of server pushes information down to the client). HTTP is a stateless protocol. In other words, the current request does not know what has been done in the previous requests. HTTP permits negotiating of data type and representation, so as to allow systems to be built independently of the data being transferred.

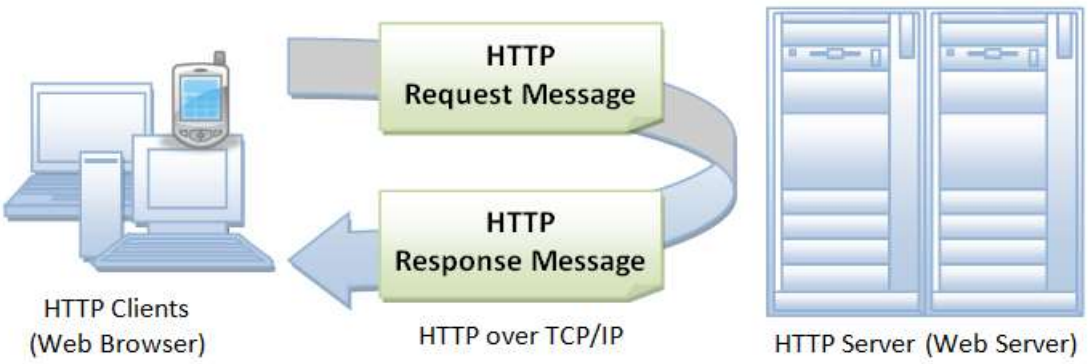


Figure 4.1: HTTP Protocol

Quoting from the RFC2616: "The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless, protocol which can be used for many tasks beyond its use for hypertext, such as name servers and distributed object management systems, through extension of its request methods, error codes and headers."

Whenever you issue a URL from your browser to get a web resource using HTTP, e.g. http://www.nowhere123.com/index.html, the browser turns the URL into a request message and sends it to the HTTP server. The HTTP server interprets the request message, and returns you an appropriate response message, which is either the resource you requested or an error message. This process is illustrated below:

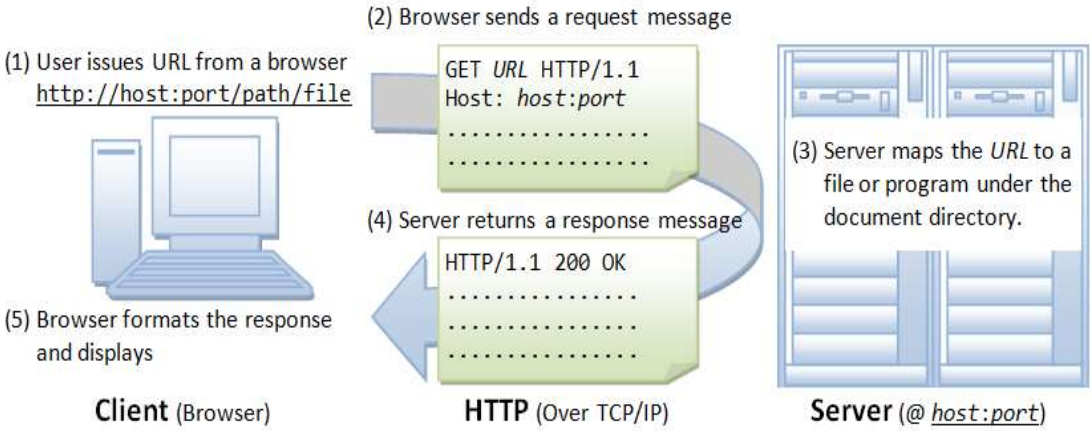


Figure 4.2 Browser Process

A URL (Uniform Resource Locator) is used to uniquely identify a resource over the web. URL has the following syntax: protocol://hostname:port/path-and-file-name. There are 4 parts in a URL: Protocol: The application-level protocol used by the client and server, e.g., HTTP, FTP, and telnet.

Hostname: The DNS domain name (e.g., www.nowhere123.com) or IP address (e.g., 192.128.1.2) of the server.

Port: The TCP port number that the server is listening for incoming requests from the clients.

Path-and-file-name: The name and location of the requested resource, under the server document base directory.

For example, in the URL http://www.nowhere123.com/docs/index.html, the communication protocol is HTTP; the hostname is www.nowhere123.com. The port number was not specified in the URL, and takes on the default number, which is TCP port 80 for HTTP. The path and file name for the resource to be located is "/docs/index.html".

HTTP Protocol:

As mentioned, whenever you enter a URL in the address box of the browser, the browser translates the URL into a request message according to the specified protocol; and sends the request message to the server. For example, the browser translated the URL http://www.nowhere123.com/doc/index.html into the following request message:

```
GET/docs/index.html HTTP/1.1
Host: www.nowhere123.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
(blank line)
```

- When this request message reaches the server, the server can take either one of these actions: The server interprets the request received, maps the request into a file under the server's document directory, and returns the file requested to the client.
- The server interprets the request received, maps the request into a program kept in the server, executes the program, and returns the output of the program to the client.
- The request cannot be satisfied, the server returns an error message.

An example of the HTTP response message is as shown:

```
HTTP/1.1 200 OK
Date: Wed, 15Apr 202008:56:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Thu, 16 Apr 2020 07:16:26 GMT
ETag: "10000000565a5-2c-3e94b66c2e680"
Accept-Ranges: bytes
Content-Length: 44
Connection: close
Content-Type: text/html
X-Pad: avoid browser bug

<html><body><h1>It works!</h1></body></html>
```

The browser receives the response message, interprets the message and displays the contents of the message on the browser's window according to the media type of the response (as in the Content-Type response header). Common media type include "text/plain", "text/html", "image/gif", "image/jpeg", "audio/mpeg", "video/mpeg", "application/msword", and "application/pdf". In its idling state, an HTTP server does nothing but listening to the IP address(es) and port(s) specified in the configuration for incoming request. When a request

arrives, the server analyzes the message header, applies rules specified in the configuration, and takes the appropriate action. The webmaster's main control over the action of web server is via the configuration, which will be dealt with in greater details in the later sections.

HTTP over TCP/IP:

HTTP is a client-server application-level protocol. It typically runs over a TCP/IP connection, as illustrated. (HTTP needs not run on TCP/IP. It only presumes a reliable transport. Any transport protocols that provide such guarantees can be used.)

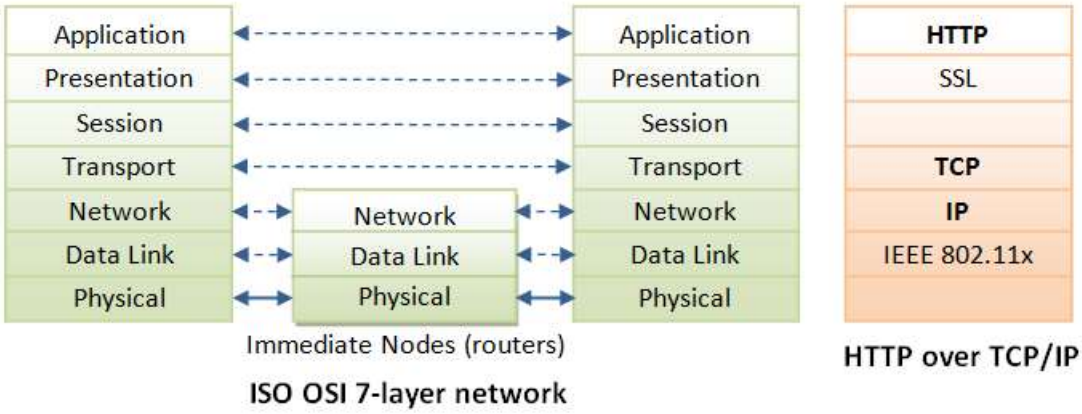


Figure 4.3 HTTP over TCP/IP

TCP/IP (Transmission Control Protocol/Internet Protocol) is a set of transport and network-layer protocols for machines to communicate with each other over the network. IP (Internet Protocol) is a network-layer protocol, deals with network addressing and routing.

In an IP network, each machine is assigned an unique IP address (e.g., 165.1.2.3), and the IP software is responsible for routing a message from the source IP to the destination IP. In IPv4 (IP version 4), the IP address consists of 4 bytes, each ranges from 0 to 255, separated by dots, which is called a quad-dotted form. This numbering scheme supports up to 4G addresses on the network. The latest IPv6 (IP version 6) supports more addresses. Since memorizing number is difficult for most of the people, an english-like domain name, such as www.nowhere123.com is used instead.

The DNS (Domain Name Service) translates the domain name into the IP address (via distributed lookup tables). A special IP address 127.0.0.1 always refers to your own machine. It's domian name is "localhost" and can be used for local loopback testing. TCP

(Transmission Control Protocol) is a transport-layer protocol, responsible for establish a connection between two machines. TCP consists of 2 protocols: TCP and UDP (User Datagram Package). TCP is reliable, each packet has a sequence number, and an acknowledgement is expected. A packet will be re-transmitted if it is not received by the receiver. Packet delivery is guaranteed in TCP. UDP does not guarantee packet delivery, and is therefore not reliable. However, UDP has less network overhead and can be used for applications such as video and audio streaming, where reliability is not critical. TCP multiplexes applications within an IP machine. For each IP machine, TCP supports (multiplexes) up to 65536 ports (or sockets), from port number 0 to 65535. An application, such as HTTP or FTP, runs (or listens) at a particular port number for incoming requests. Port 0 to 1023 are pre-assigned to popular protocols, e.g., HTTP at 80, FTP at 21, Telnet at 23, SMTP at 25, NNTP at 119, and DNS at 53. Port 1024 and above are available to the users.

4.2 HTTP REST model:

REST is acronym for REpresentational State Transfer. It is architectural style for distributed hypermedia systems and was first presented by Roy Fielding in 2000 in his famous dissertation. Like any other architectural style, REST also does have it's own 6 guiding constraints which must be satisfied if an interface needs to be referred as RESTful. These principles are listed below.

Guiding Principles of REST:

- Client–server – By separating the user interface concerns from the data storage concerns, we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components.
- Stateless – Each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client.
- Cacheable – Cache constraints require that the data within a response to a request be implicitly or explicitly labeled as cacheable or non-cacheable. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests.

- Uniform interface – By applying the software engineering principle of generality to the component interface, the overall system architecture is simplified and the visibility of interactions is improved. In order to obtain a uniform interface, multiple architectural constraints are needed to guide the behavior of components. REST is defined by four interface constraints: identification of resources; manipulation of resources through representations; self-descriptive messages; and, hypermedia as the engine of application state.
- Layered system – The layered system style allows an architecture to be composed of hierarchical layers by constraining component behavior such that each component cannot "see" beyond the immediate layer with which they are interacting.
- Code on demand (optional) – REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts. This simplifies clients by reducing the number of features required to be pre-implemented.

The key abstraction of information in REST is a resource. Any information that can be named can be a resource: a document or image, a temporal service, a collection of other resources, a non-virtual object (e.g. a person), and so on. REST uses a resource identifier to identify the particular resource involved in an interaction between components. The state of the resource at any particular timestamp is known as resource representation. A representation consists of data, metadata describing the data and hypermedia links which can help the clients in transition to the next desired state. The data format of a representation is known as a media type. The media type identifies a specification that defines how a representation is to be processed. A truly RESTful API looks like hypertext. Every addressable unit of information carries an address, either explicitly (e.g., link and id attributes) or implicitly (e.g., derived from the media type definition and representation structure).

Another important thing associated with REST is resource methods to be used to perform the desired transition. A large number of people wrongly relate resource methods to HTTP GET/PUT/POST/DELETE methods. Roy Fielding has never mentioned any recommendation around which method to be used in which condition. All he emphasizes is that it should be uniform interface. If you decide HTTP POST will be used for updating a resource – rather than most people recommend HTTP PUT – it's alright and application interface will be RESTful.

Ideally, everything that is needed to change the resource state shall be part of API response for that resource – including methods and in what state they will leave the representation.

4.3 CRUD operations

Within computer programming, the acronym CRUD stands for create, read, update and delete. These are the four basic functions of persistent storage. Also, each letter in the acronym can refer to all functions executed in relational database applications and mapped to a standard HTTP method, SQL statement or DDS operation. It can also describe user-interface conventions that allow viewing, searching and modifying information through computer-based forms and reports. Entities are read, created, updated and deleted. Those same entities can be modified by taking the data from a service and changing the setting properties before sending the data back to the service for an update. Plus, CRUD is data-oriented and the standardized use of HTTP action verbs.

Most applications have some form of CRUD functionality. In fact, every programmer has had to deal with CRUD at some point. Not to mention, a CRUD application is one that utilizes forms to retrieve and return data from a database. The first reference to CRUD operations came from Haim Kilov in 1990 in an article titled, "From semantic to object-oriented data modeling." However, the term was first made popular by James Martin's 1983 book, Managing the Database Environment. Here's a breakdown:

- CREATE procedures: Performs the INSERT statement to create a new record.
- READ procedures: Reads the table records based on the primary keynoted within the input parameter.
- UPDATE procedures: Executes an UPDATE statement on the table based on the specified primary key for a record within the WHERE clause of the statement.
- DELETE procedures: Deletes a specified row in the WHERE clause.

Based on the requirements of a system, varying user may have different CRUD cycles. A customer may use CRUD to create an account and access that account when returning to a particular site. The user may then update personal data or change billing information. On the other hand, an operations manager might create product records, then call them when needed or modify line items.

Here's an example, illustrating an asp.net MVC 4 CRUD operation using ADO.NET:

```
CREATE PROCEDURE Usp_InsertUpdateDelete_Customer
@CustomerID BIGINT = 0
,@Name NVARCHAR(100) = NULL
,@Mobileno NVARCHAR(15) = NULL
,@Address NVARCHAR(300) = 0
,@Birthdate DATETIME = NULL
,@EmailID NVARCHAR(15) = NULL
,@Query INT
AS
BEGIN
IF (@Query = 1)
BEGIN
INSERT INTO Customer(
NAME
,Address
,Mobileno
,Birthdate
,EmailID
)
VALUES (
@Name
,@Address
,@Mobileno
,@Birthdate
,@EmailID
)

IF (@@ROWCOUNT > 0)
BEGIN
SELECT 'Insert'
END
END

IF (@Query = 2)
BEGIN
UPDATE Customer
SET NAME = @Name
,Address = @Address
,Mobileno = @Mobileno
,Birthdate = @Birthdate
,EmailID = @EmailID
WHERE Customer.CustomerID = @CustomerID

SELECT 'Update'
END

IF (@Query = 3)
BEGIN
DELETE
FROM Customer
WHERE Customer.CustomerID = @CustomerID

SELECT 'Deleted'
END

IF (@Query = 4)
BEGIN
SELECT *
FROM Customer
END
END

IF (@Query = 5)
BEGIN
SELECT *
FROM Customer
WHERE Customer.CustomerID = @CustomerID
END
```

During the Web 2.0 era, CRUD operations were at the foundation of most dynamic websites. However, you should differentiate CRUD from the HTTP action verbs. For example, if you want to create a new record you should use "POST." To update a record, you would use "PUT" or "PATCH." If you wanted to delete a record, you would use "DELETE." Through CRUD, users and administrators had the access rights to edit, delete, create or browse online records. An application designer has many options for executing CRUD operations. One of the most efficient of choices is to create a set of stored procedures in SQL to execute operations. With regard to CRUD stored procedures, here are a few common naming conventions:

- The procedure name should end with the implemented name of the CRUD operation. The prefix should not be the same as the prefix used for other user-defined stored procedures.
- CRUD procedures for the same table will be grouped together if you use the table name after the prefix.
- After adding CRUD procedures, you can update the database schema by identifying the database entity where CRUD operations will be implemented.

Benefits of CRUD:

Instead of using ad-hoc SQL statements, many programmers prefer to use CRUD because of its performance. When a stored procedure is first executed, the execution plan is stored in SQL Server's procedure cache and reused for all applications of the stored procedure.

When a SQL statement is executed in SQL Server, the relational engine searches the procedure cache to ensure an existing execution plan for that particular SQL statement is available and uses the current plan to decrease the need for optimization, parsing and recompiling steps for the SQL statement.

If an execution plan is not available, then the SQL Server will create a new execution plan for the query. Moreover, when you remove SQL statements from the application code, all the SQL can be kept in the database while only stored procedure invocations are in the client application. When you use stored procedures, it helps to decrease database coupling. Furthermore, using CRUD operations helps to prevent SQL injection attacks. By utilizing stored procedures instead of string concatenation to build dynamic queries from user input data for all SQL Statements means that everything placed into a parameter gets quoted.

4.4 Websocket

Web sockets are defined as a two-way communication between the servers and the clients, which mean both the parties communicate and exchange data at the same time. The key points of Web Sockets are true concurrency and optimization of performance, resulting in more responsive and rich web applications. Description of Web Socket Protocol This protocol defines a full duplex communication from the ground up. Web sockets take a step forward in bringing desktop rich functionalities to the web browsers. It represents an evolution, which was awaited for a long time in client/server web technology.

The main features of web sockets are as follows − Web socket protocol is being standardized, which means real time communication between web servers and clients is possible with the help of this protocol.

Web sockets are transforming to cross platform standard for real time communication between a client and the server. This standard enables new kind of the applications. Businesses for real time web application can speed up with the help of this technology. The biggest advantage of Web Socket is it provides a two-way communication (full duplex) over a single TCP connection.

Before diving to the need of Web sockets, it is necessary to have a look at the existing techniques, which are used for **duplex communication** between the server and the client. They are as follows −

- Polling
- Long Polling
- Streaming
- Postback and AJAX
- HTML5

Polling

Polling can be defined as a method, which performs periodic requests regardless of the data that exists in the transmission. The periodic requests are sent in a synchronous way. The client makes a periodic request in a specified time interval to the Server. The response of the server includes available data or some warning message in it.

Long Polling

Long polling, as the name suggests, includes similar technique like polling. The client and the server keep the connection active until some data is fetched or timeout occurs. If the connection is lost due to some reasons, the client can start over and perform sequential request.Long polling is nothing but performance improvement over polling process, but constant requests may slow down the process.

Streaming

It is considered as the best option for real-time data transmission. The server keeps the connection open and active with the client until and unless the required data is being fetched. In this case, the connection is said to be open indefinitely. Streaming includes HTTP headers which increases the file size, increasing delay. This can be considered as a major drawback.

Internet was conceived to be a collection of Hypertext Mark-up Language (HTML) pages linking one another to form a conceptual web of information. During the course of time, static resources increased in number and richer items, such as images and began to be a part of the web fabric. Server technologies advanced which allowed dynamic server pages - pages whose content was generated based on a query. Soon, the requirement to have more dynamic web pages lead to the availability of Dynamic Hypertext Mark-up Language (DHTML). All thanks to JavaScript. Over the following years, we saw cross frame communication in an attempt to avoid page reloads followed by HTTP Polling within frames. However, none of these solutions offered a truly standardized cross browser solution to real-time bi-directional communication between a server and a client. This gave rise to the need of Web Sockets Protocol. It gave rise to full-duplex communication bringing desktop-rich functionality to all web browsers.

Web Socket represents a major upgrade in the history of web communications. Before its existence, all communication between the web clients and the servers relied only on HTTP. Web Socket helps in dynamic flow of the connections that are persistent full duplex. Full duplex refers to the communication from both the ends with considerable fast speed. It is termed as a game changer because of its efficiency of overcoming all the drawbacks of existing protocols.

Web Socket for Developers and Architects:

Importance of Web Socket for developers and architects –

- Web Socket is an independent TCP-based protocol, but it is designed to support any other protocol that would traditionally run only on top of a pure TCP connection.
- Web Socket is a transport layer on top of which any other protocol can run. The Web Socket API supports the ability to define sub-protocols: protocol libraries that can interpret specific protocols.
- Examples of such protocols include XMPP, STOMP, and AMQP.
- The developers no longer have to think in terms of the HTTP request-response paradigm.
- The only requirement on the browser-side is to run a JavaScript library that can interpret the Web Socket handshake, establish and maintain a Web Socket connection.
- On the server side, the industry standard is to use existing protocol libraries that run on top of TCP and leverage a Web Socket Gateway.

The following diagram describes the functionalities of Web Sockets –

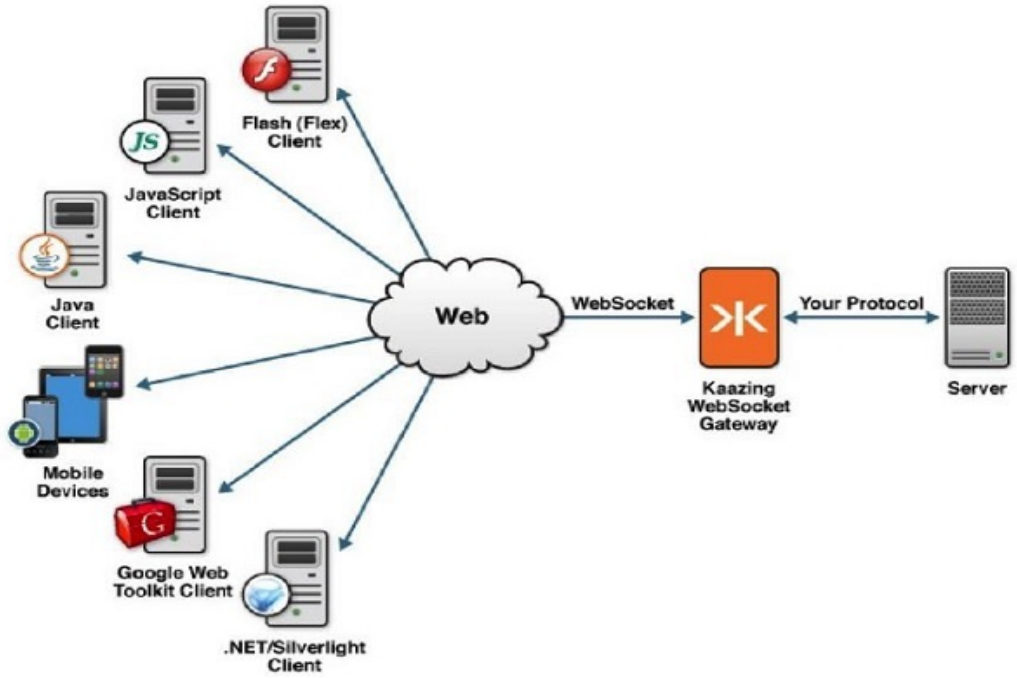


Figure 4.4 Functionalities of Web Sockets

Web Socket connections are initiated via HTTP; HTTP servers typically interpret Web Socket handshakes as an Upgrade request. Web Sockets can both be a complementary add-on to an existing HTTP environment and can provide the required infrastructure to add web functionality. It relies on more advanced, full duplex protocols that allow data to flow in both directions between client and server.

Functions of Web Sockets:

Web Sockets provide a connection between the web server and a client such that both the parties can start sending the data.

The steps for establishing the connection of Web Socket are as follows –

- The client establishes a connection through a process known as Web Socket handshake.
- The process begins with the client sending a regular HTTP request to the server.
- An Upgrade header is requested. In this request, it informs the server that request is for Web Socket connection. Web Socket URLs use the ws scheme.
- They are also used for secure Web Socket connections, which are the equivalent to HTTPs.

A simple example of initial request headers is as follows –

```
GET ws://websocket.example.com/ HTTP/1.1
Origin: http://example.com
Connection:Upgrade
Host: websocket.example.com
Upgrade: websocket
```

Web Sockets occupy a key role not only in the web but also in the mobile industry. The importance of Web Sockets is given below.

- Web Sockets as the name indicates, are related to the web.
- Web consists of a bunch of techniques for some browsers; it is a broad communication platform for vast number of devices, including desktop computers, laptops, tablets and smart phones.
- HTML5 app that utilizes Web Sockets will work on any HTML5 enabled web browser. Web socket is supported in the mainstream operating systems.

- All key players in the mobile industry provide Web Socket APIs in own native apps. Web sockets are said to be a full duplex communication.
- The approach of Web Sockets works well for certain categories of web application such as chat room, where the updates from client as well as server are shared simultaneously.

Web Sockets, a part of the HTML5 specification, allow full duplex communication between web pages and a remote host. The protocol is designed to achieve the following benefits, which can be considered as the key points –

- Reduce unnecessary network traffic and latency using full duplex through a single connection (instead of two).
- Streaming through proxies and firewalls, with the support of upstream and downstream communication simultaneously.

It is necessary to initialize the connection to the server from client for communication between them. For initializing the connection, creation of Javascript object with the URL with the remote or local server is required.

```
var socket = new WebSocket(“ ws://echo.websocket.org ”);
```

The URL mentioned above is a public address that can be used for testing and experiments. The websocket.org server is always up and when it receives the message and sends it back to the client. This is the most important step to ensure that application works correctly.

Web Sockets – Events:

There are four main Web Socket API events –

- Open
- Message
- Close
- Error

Each of the events are handled by implementing the functions like onopen, onmessage, onclose and onerror functions respectively. It can also be implemented with the help of addEventListener method.

The brief overview of the events and functions are described as follows –

- Open: Once the connection has been established between the client and the server, the open event is fired from Web Socket instance. It is called as the initial handshake between client and server. The event, which is raised once the connection is established, is called onopen.
- Message: Message event happens usually when the server sends some data. Messages sent by the server to the client can include plain text messages, binary data or images. Whenever the data is sent, the onmessage function is fired.
- Close: Close event marks the end of the communication between server and the client. Closing the connection is possible with the help of onclose event. After marking the end of communication with the help of onclose event, no messages can be further transferred between the server and the client. Closing the event can happen due to poor connectivity as well.
- Error: Error marks for some mistake, which happens during the communication. It is marked with the help of onerror event. Onerror is always followed by termination of connection.

Web Sockets – Actions:

Events are usually triggered when something happens. On the other hand, actions are taken when a user wants something to happen. Actions are made by explicit calls using functions by users. The Web Socket protocol supports two main actions, namely −

- send( )
- close( )

send ( ) :

This action is usually preferred for some communication with the server, which includes sending messages, which includes text files, binary data or images.

A chat message, which is sent with the help of send() action, is as follows –

```
// get text view and button for submitting the message
var textsend = document.getElementById("text-view");
var submitMsg = document.getElementById("tsend-button");

//Handling the click event
submitMsg.onclick =function(){
// Send the data
   socket.send( textsend.value);
```

```
}
```

close ( ):

This method stands for goodbye handshake. It terminates the connection completely and no data can be transferred until the connection is re-established.

```
var textsend = document.getElementById("text-view");
var buttonStop = document.getElementById("stop-button");

//Handling the click event
buttonStop.onclick =function(){
// Close the connection if open
if(socket.readyState ===WebSocket.OPEN){
      socket.close();
}
}
```

Once a connection has been established between the client and the server, the open event is fired from Web Socket instance. It is called as the initial handshake between client and server. The event, which is raised once the connection is established, is called the onopen. Creating Web Socket connections is really simple. All you have to do is call the WebSocket constructor and pass in the URL of your server.

The following code is used to create a Web Socket connection –

```
// Create a new WebSocket.
var socket =newWebSocket('ws://echo.websocket.org');
```

Once the connection has been established, the open event will be fired on your Web Socket instance. onopen refers to the initial handshake between client and the server which has lead to the first deal and the web application is ready to transmit the data.

The following code snippet describes opening the connection of Web Socket protocol –

```
socket.onopen =function(event){
   console.log("Connection established");
// Display user friendly messages for the successful establishment
of connection
var.label = document.getElementById("status");
   label.innerHTML ="Connection established";
}
```

It is a good practice to provide appropriate feedback to the users waiting for the Web Socket connection to be established. However, it is always noted that Web Socket connections are comparatively fast.

Example:

Building up the client-HTML5 file

```
<!DOCTYPE html>
<html>
<metacharset="utf-8"/>
<title>WebSocket Test</title>

<scriptlanguage="javascript"type="text/javascript">
var wsUri ="ws://echo.websocket.org/";
var output;

function init(){
          output = document.getElementById("output");
          testWebSocket();
}

function testWebSocket(){
          websocket =newWebSocket(wsUri);

          websocket.onopen =function(evt){
             onOpen(evt)
};
}

function onOpen(evt){
          writeToScreen("CONNECTED");
}

       window.addEventListener("load", init,false);

</script>

<h2>WebSocket Test</h2>
<divid="output"></div>

</html>
```

The output will be as follows –

The above HTML5 and JavaScript file shows the implementation of two events of Web Socket, namely –

- onLoad which helps in creation of JavaScript object and initialization of connection.
- onOpen establishes connection with the server and also sends the status.

Bridge: Bridge is a network device which works in data link layer. Bridge connects two different LAN working on same protocol. Also In bridge, format of packet is not changed.

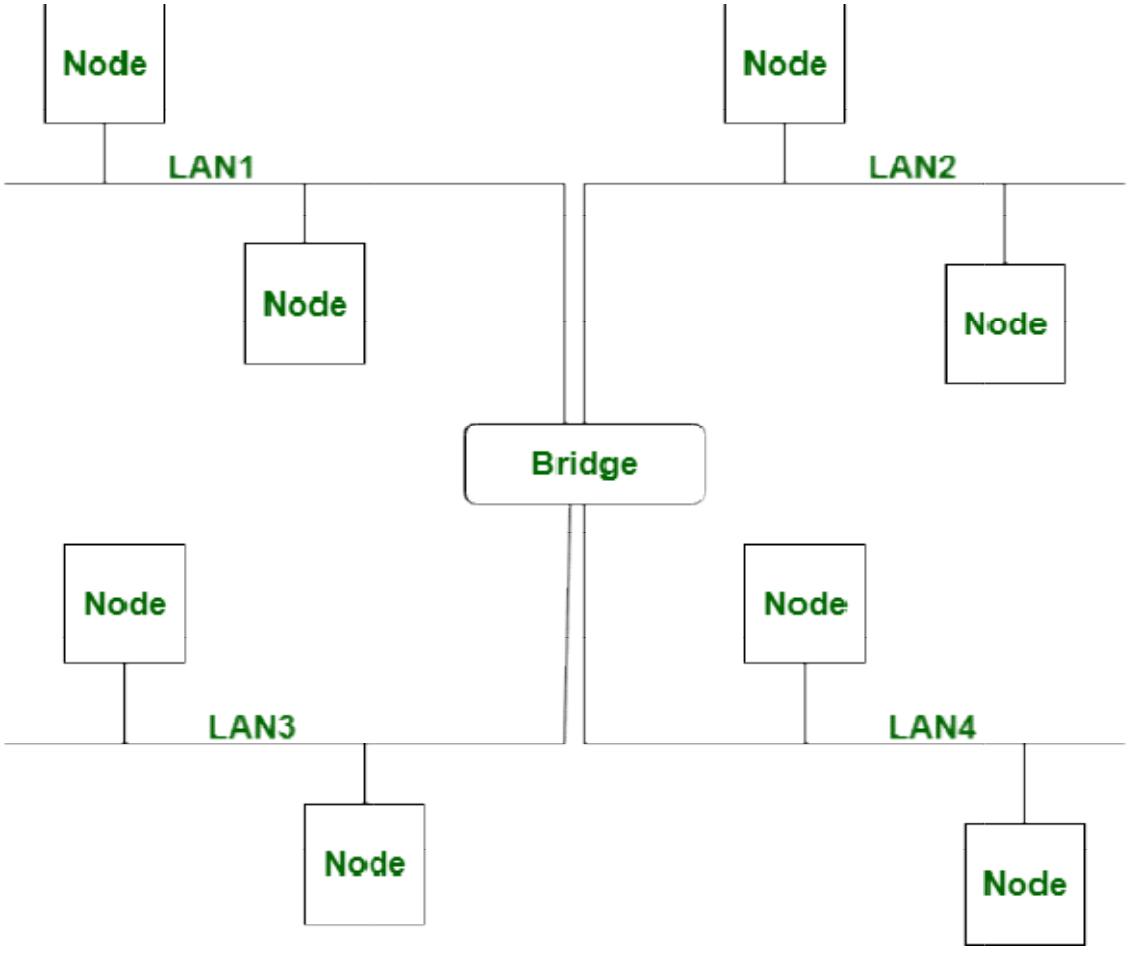


Figure 4.5 Bridge-Network Device

Gateway:

Gateway works in OSI model's all layer. It converts the protocol. Gateway will settle for and transfer the packet across networks employing a completely different protocol. In gateway, format of packet is changed which oppose to the bridge.

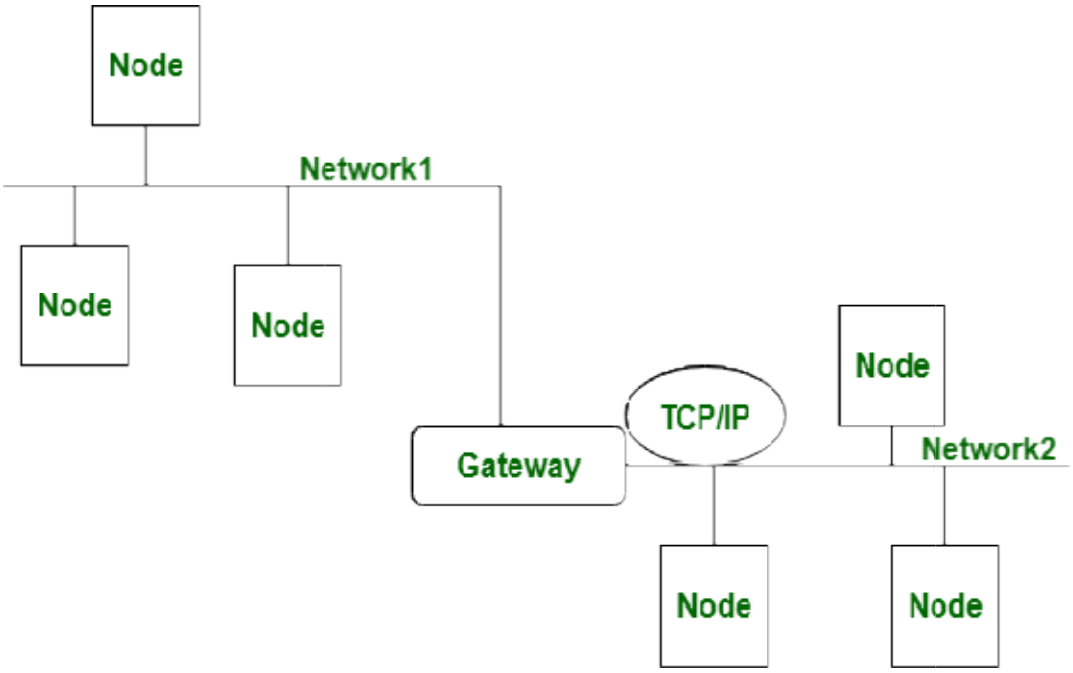


Figure 4.6 Gateway-Network Device

Difference of Bridge and Gateway:

S.No.	Bridge	Gateway
1.	Bridge works in data link layer.	While it works in all layer.
2.	Bridge connects two different LANs.	While it converts the protocol.
3.	Bridge connects two different LAN working on same protocol.	While gateway will settle for and transfer the packet across networks employing a completely different protocol.
4.	In bridge, data or	Whereas in gateway, data or information is

	information is in the form of packet.	also in the form of packet.
5.	In bridge, format of packet is not changed.	While in gateway, format of packet is changed.
6.	Bridge is not installed in router.	While it installed in router.

Questions for Practice:

Part-A

1. Describe the features of HTTP
2. Classify HTTP and HTTP Rest model
3. Explain in detail about Web Sockets.
4. Estimate the significance of Gateway Design.
5. Infer the properties of Gateway Design

Part-B

1. Evaluate the CRUD Operations based on HTTP.
2. Validate connectivity to IoT cloud platforms using HTTP REST Services
3. Measure the Significance of gateway design and its characteristics.
4. Justify the differences between HTTP, HTTP REST and Protocol Bridging with an example?

References:

1. Mischa Dohler, Carles Anton-Haro, "Machine-to-machine (M2M) Communications", Woodhead Publishing, 1st Edition, 2014.
2. Perry Lea, "Internet of Things for Architects: Architecting IoT solutions by implementing sensors, communication infrastructure, edge computing, analytics, and security", Packet, 1st Edition, 2018.

3. Maciej Kranz , “Building the Internet of Things”, John Wiley & Sons, 1st Edition, 2017.
4. David Boswarthick, Omar Elloumi, Olivier Hersent, “M2M Communications: A Systems Approach”, John Wiley & Sons, 1 st Edition, 2012.
5. Anupama C. Raman and Pethuru Raj, “The Internet of Things: Enabling Technologies, Platforms, and Use Cases”, CRC press, 1st Edition, 2017.

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF ELECTRICAL AND ELECTRONICS

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

UNIT – V – EDGE and FOG Computing – SECA7019

UNIT 5 EDGE AND FOG COMPUTING

Introduction to Edge Computing, Edge computing vs FOG computing, Edge Analytics at devices and Gateways, Down sampling of data, aggregations, filters, Threshold prediction, Detecting Anomalies.

5.1 Introduction to Edge Computing:

The rapid development of technology has seen an increase in the number of IoT devices in recent years. This number is expected to increase more in the future as well. In the year 2019 itself, more than 6 billion IoT devices are in use. The rise in the number of devices means the increase in data generation. This voluminous data generated from the devices is creating problems as the processing of this data is a challenging task. IoT devices generate data which is stored on the cloud. This data needs faster processing and response. However, due to the increase in transmission latency the IoT device user might face some performance troubles. Transmission latency increases due to the increase in the distance between the cloud and the IoT device user. Thus, the closer the user to the cloud, the better is the performance speed of the device.

The concept of Edge computing came into the feature in order to sort out this problem. Edge computing is somewhat similar to the Content Delivery Network (CDN) but edge server processes the data generated from IoT devices in edge computing instead of the centralized cloud server. There is an edge server near the IoT device user which serves as a bridge between the cloud and the user. This decreases the data transmission latency and the user experiences a speedy performance of the device. There are some areas that require minimal transmission latency such as smart homes, cloud gaming, video streaming, smart devices, etc. If there is a delay in data transmission in these areas then the impacts can be dangerous. For example, in the case of self-driving cars the response from the cloud should be very quick. If there is a delay in transmission of response after the car sends the data to the central server, consequences can be dangerous as the car cannot take human-like decisions on its own. Thus, edge computing reduces the transmission latency when voluminous data has to be transmitted. Edge computing helps in real-time data processing without any sort of transmission latency. Since the data is

processed near the user device, the usage of Internet bandwidth is also reduced. These features of edge computing are making it a fundamental constituent for organizations.

According to one of the statistics of Gartner, 50% of the data generated by an enterprise will be created and processed by edge computing rather than centralized cloud data centres by 2022. Edge Computing: Advantages and difficulties The decrease in response time to microseconds can be taken as the major advantage of edge computing. There is less usage of internet bandwidth that ultimately saves the network resources. Although edge computing seems similar to cloud computing, there is a difference in the advantages and applications of cloud computing.

Edge computing has a handful of challenges despite its capability to mitigate latency in data transmission and network usage. Some of them are:

Network Bandwidth: The storage of more data demands the expansion of bandwidth. Thus, the network bandwidth needs to be increased with the increase in the amount of data while edge computing is used.

Security: Due to the distributed structure, edge computing is vulnerable to security exploits. This type of architecture increases attack vectors that may result in malware infections.

Compliances: The data stored over Edge servers need verification for compliances and regulations as data is one of the key components of any business organization.

Latency: The distance between the user device and cloud affects the transmission latency. Sometimes there is an increase in latency that may result in disastrous consequences.

Hence, edge computing along with solutions brings some challenges too. These difficulties can be overcome by consulting with cloud experts to start utilizing the benefits of edge computing. So, the implementation of edge computing in any business surely helps in the growth of the business.

5.2 Edge computing vs FOG computing

The fundamental objective of the internet of things (IoT) is to obtain and analyze data from assets that were previously disconnected from most data processing tools. This data is generated by physical assets or things deployed at the very edge of the network—such as motors, light bulbs, generators, pumps, and relays—that perform specific tasks to support a business

process. The internet of things is about connecting these unconnected devices (things) and sending their data to the cloud or Internet to be analyzed.

In traditional IoT cloud architecture, all data from physical assets or things is transported to the cloud for storage and advanced analysis. Once in the cloud, the data is used for cognitive prognostics (that is, predictive maintenance, forensic failure analysis, and process optimization).

Fog and edge computing in manufacturing and automation applications are network and system architectures that attempt to collect, analyze, and process data from these assets more efficiently than traditional cloud architecture.

These architectures share similar objectives:

- To reduce the amount of data sent to the cloud
- To decrease network and internet latency
- To improve system response time in remote mission-critical applications

However, there is a key difference between the two concepts. Both fog computing and edge computing involve pushing intelligence and processing capabilities down closer to where the data originates—at the network edge. The key difference between the two architectures is exactly where that intelligence and computing power is placed.

- Fog computing pushes intelligence down to the local area network (LAN) level of network architecture, processing data in a fog node or IoT gateway.
- Edge computing pushes the intelligence, processing power, and communication capabilities of an edge gateway or appliance directly into devices like PLCs (programmable logic controllers), PACs (programmable automation controllers), and especially EPICs (edge programmable industrial controllers).

In both architectures data is generated from the same source—physical assets such as pumps, motors, relays, sensors, and so on. These devices perform a task in the physical world such as pumping water, switching electrical circuits, or sensing the world around them. These are the “things” that make up the internet of things.

Fog computing: In fog computing, transporting data from things to the cloud requires many steps.

- First the electrical signals from things are traditionally wired to the I/O points of an automation controller (PLC or PAC). The automation controller executes a control system program to automate the things.
- Next the data from the control system program is sent to an OPC server or protocol gateway, which converts the data into a protocol Internet system understand, such as MQTT or HTTP.
- Then the data is sent to another system, such as a fog node or IoT gateway on the LAN, which collects the data and performs higher-level processing and analysis. This system filters, analyzes, processes, and may even store the data for transmission to the cloud or WAN later.

So fog computing involves many layers of complexity and data conversion. Its architecture relies on many links in a communication chain to move data from the physical world of our assets into the digital world of information technology. In a fog computing architecture, each link in the communication chain is a potential point of failure.

Edge computing: Edge computing simplifies this communication chain and reduces potential points of failure. In edge computing, physical assets like pumps, motors, and generators are again physically wired into a control system, but this system is controlled by an edge programmable industrial controller, or EPIC.

The EPIC automates the physical assets by executing an onboard control system program, just like a PLC or PAC. But the EPIC has edge computing capabilities that allow it to also collect, analyze, and process data from the physical assets it's connected to—at the same time it's running the control system program. EPICs then use edge computing capabilities to determine what data should be stored locally or sent to the cloud for further analysis.

In edge computing, intelligence is literally pushed to the network edge, where our physical assets or things are first connected and where IoT data originates. Edge computing

saves time and money by streamlining IoT communication, reducing system and network architecture complexity, and decreasing the number of potential failure points in an IoT application. Reducing system architecture complexity is key to the success of IIoT applications.

Encountering failure with edge and fog computing:

Of course, there are patterns of failure, as well:

- If you place too much at the edge, it's easy to overwhelm the smaller processor and storage platforms that exist there. In some cases, storage could be limited to a few gigabytes and processing using a single CPU. Power and size restrictions are really what set the limits.
- Another pattern is failure to integrate security from concept to production. Security is systemic to both edge and fog computing architectures and centralized processing. Security needs to span both and use mechanisms such as identity and access management. Encryption is not a nice-to-have, but rather a requirement for device safety. Imagine if a jet engine could be hacked in flight.

Fog and edge could create a tipping point, of sorts. Network latency limited IoT's full evolution and maturity given the limited processing that can occur at sensors. Placing a micro-platform at the IoT device, as well as providing tools and approaches to leverage this platform, will likely expand the capabilities of IoT systems and provide more use cases for IoT in general. Fog and edge are enabling technologies and standards that give IoT users and technology providers with more options. Removing the limits of centralized cloud servers means IoT is much more distributed and flexible in the services providers can offer. The first step is to understand what edge and fog are, how they can be best exploited within your own problem domain, and then face real business problems. Consider the number of ways edge and fog computing can make consumers' and workers' lives better. The value is limitless.

5.3 Edge Analytics at devices and Gateways

Edge Computing/ Edge Analytics is a relatively new approach for many companies. Most architectures are used to sending all data to the Cloud/Lake. But in Edge computing, that does not happen. i.e. data can be processed near the source and not all data is sent back to the Cloud. For large-scale IoT deployments, this functionality is critical because of the sheer volumes of Data being generated.The significance and evolution of IoT edge analytics. It is believed that hardware capabilities will converge for large vendors (Cisco, Dell, HPE, Intel, IBM and others). Hence, IoT analytics will be the key differentiator.

Companies like Cisco, Intel and others were early proponents of Edge Computing by positioning their Gateways as Edge devices. Historically, gateways performed the function of traffic aggregation and routing. In the Edge computing model, the core gateway functionality has evolved. Gateways do not just route data but also store data and to perform computations on the data as well.

Edge analytics allows us to do some pre-processing or filtering of the data closer to where the data is being created. Thus, data that falls within normal parameters can be ignored or stored in a low cost storage and abnormal readings may be sent to the Lake or the in-memory database. Now, a new segment of the market is developing driven by vendors like Dell. HPE and others.

These vendors are positioning their servers as Edge devices by adding additional storage, computing power and analytics capabilities. This has implications for Edge Analytics for IoT. IoT Edge Analytics is typically applicable for Oil Rigs, Mines and Factories which operate in low bandwidth, low latency environments. Edge Analytics could apply not just to sensor data but also to richer forms of data such as Video analytics. IoT datasets are massive. A typical Formula One car carries 150-300 sensors.

An airlines for example, the current Airbus A350 model has close to 6,000 sensors and generates 2.5 Tb of data per day,. A city (for example the Smart city of Santander in Spain) includes a network comprising more than 25,000 sensors. To avoid these sensors from constantly pinging the Cloud, we need some form of interim processing. Hence, the need for

Edge processing in IoT analytics. We can consider Edge devices from two perspectives: Evolution of the traditional Gateway vendors and Evolution of the traditional server vendors.

Impact on IoT analytics:

The two stages are Creating the analytics model and Executing the Analytics model. Creating the analytics model involves : collecting data, storing data, preparing the data for analytics(some ETL functions), choosing the analytics algorithm, training the algorithms, validating the analytic goodness of fit etc. The output of this trained model will be rules, recommendations, scores etc. Only then, can we deploy this model. So, when we say, we are implementing analytics at the 'edge' – what exactly are we doing? If you follow the examples above ex from Dell, the more general case is: creating the model in one location and potentially deploying the model at multiple points (ex from Cloud to gateway, server, factory etc)

PMML becomes important for the ability to deploy models in multiple locations: Predictive Model Markup Language (PMML) PMML is an XML-based predictive model interchange format. PMML provides a way for analytic applications to describe and exchange predictive models produced by data mining and machine learning algorithms. It supports common models such as logistic regression and feedforward neural networks.

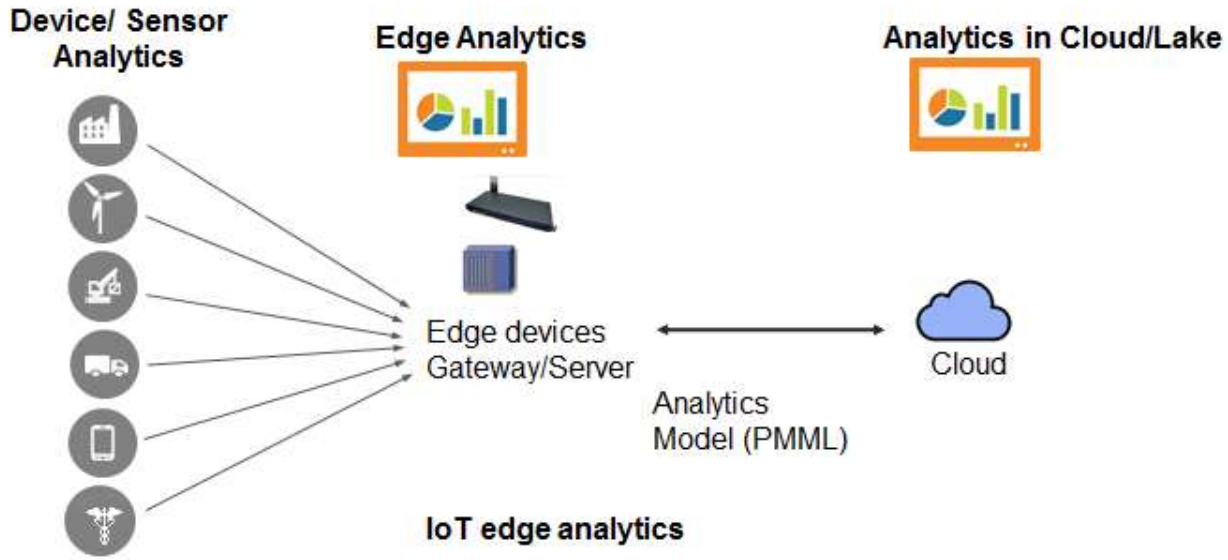


Figure 5.1: IOT Edge Analytics

Decentralized processing has inherent complexity: When you decentralize processing, you face some inherently complex situations – ex management and replication of master data, security, storage etc. In addition, the process of creating the analytics model on one machine and deploying it on another machine is new.

Peer to Peer node communication may be a real possibility over time: IoT is currently deployed through silos. Edge networks offer the possibility of Peer to Peer communication by the Edge devices if they have enough processing capability

Advantages of Edge Analytics: In edge analytics, however, the devices or the gateways can handle the analysis.

Reduction of network bottlenecks: Some data, for example video employed in smart city applications such as traffic management, is so large that it could congest the network. A network with a bandwidth of 100 Gbps, for example, can support uploads of 1080p streams from only 12,000 users at YouTube's recommended upload rate of 8.5 Mbps, according to a recent article in Pervasive Computing. A million concurrent uploads would require 8.5 Tera bytes per second.

Fast response times: Applications such as energy production from wind or solar power plants and monitoring of ill patients require response times of a minute or less. When that data is sent to a central location for analysis, it loses its value.

Data filtering: Allows analytics to be performed on actionable data; only necessary data is analyzed or sent on for further analysis.

IoT edge analytics is an exciting space. It is also a nascent space with many more developments yet to come. Concepts such as PMML(deploying models), Peer edge nodes etc could be key drivers of IoT deployments in future.

5.4 Data Analytics and Performance Enhancement

Based on the evolving communications, computing and embedded systems technologies, Internet of Things (IoT) systems can interconnect not only physical users and devices but also virtual services and objects, which have already been applied to many different application scenarios, such as smart home, smart healthcare, and intelligent transportation. With the rapid development, the number of involving devices increases tremendously. The huge number of devices and correspondingly generated data bring critical challenges to the

IoT systems. To enhance the overall performance, this thesis aims to address the related technical issues on IoT data processing and physical topology discovery of the subnets self-organized by IoT devices.

First, the issues on outlier detection and data aggregation are addressed through the development of recursive principal component analysis (R-PCA) based data analysis framework. The framework is developed in a cluster-based structure to fully exploit the spatial correlation of IoT data. Specifically, the sensing devices are gathered into clusters based on spatial data correlation. Edge devices are assigned to the clusters for the R-PCA based outlier detection and data aggregation. The outlier-free and aggregated data are forwarded to the remote cloud server for data reconstruction and storage. Moreover, a data reduction scheme is further proposed to relieve the burden on the trunk link for data uploading by utilizing the temporal data correlation. Kalman filters (KFs) with identical parameters are maintained at the edge and cloud for data prediction. The amount of data uploading is reduced by using the data predicted by the KF in the cloud instead of uploading all the practically measured data.

Data Analytics in IoT Systems:

With the pervasive deployment of IoT technology, the number of connected IoT end devices increases in an explosive trend, which continuously generates a massive amount of data. Timely data analytics can provide useful information for decision making in the IoT systems, which is able to enhance both the system efficiency and reliability. More specifically, data analytics in IoT systems is utilized to effectively and effciently process the discrete IoT data series and provide services such as data classification, pattern analysis, and tendency prediction. However, the continuous generation of data from heterogeneous devices brings huge technical challenges to IoT data analytics. Thus, how to timely and fully process and analyze the massive and heterogeneous IoT data needs to be seriously considered in the design of IoT systems.

With the rapid development of communications and embedded systems technologies, IoT systems have been pervasively deployed in different kinds of application scenarios. The numberof involving IoT end devices keeps increasing in an explosive trend. These devices directly interact with the real world and continuously generate a massive amount of data,

which brings huge challenges to the data analytics in IoT systems, particularly the data analytics with a critical requirement of completion time.

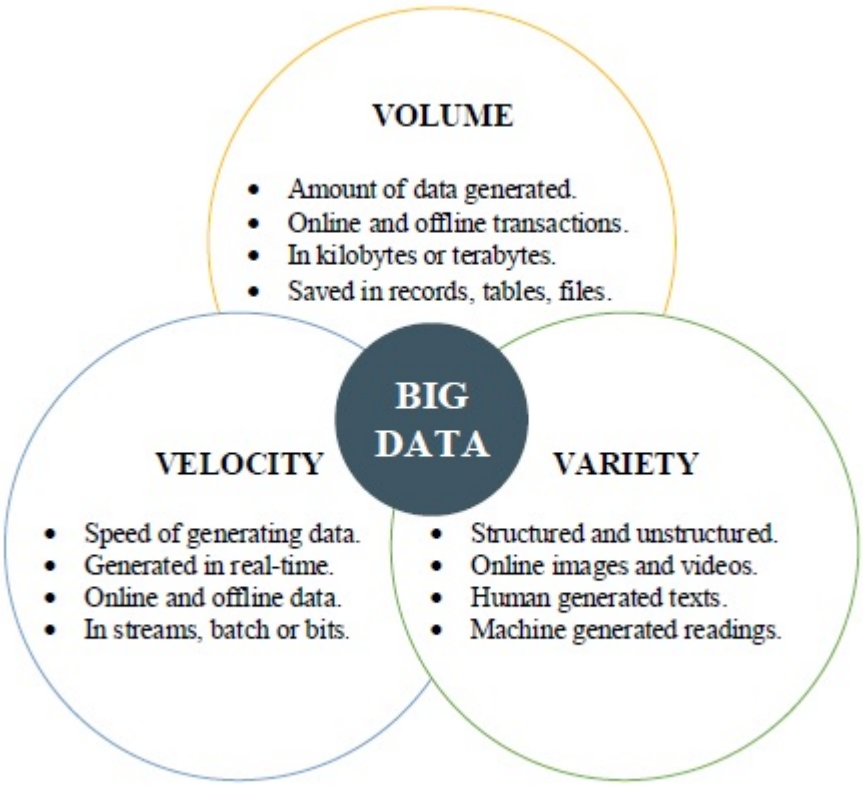


Figure 5.2 Three Vs of big data: volume, velocity, and variety

Thus, data analytics needs to be seriously considered in the IoT systems.

IoT Data Characteristics:

With the tremendous increment in the number of IoT end devices, a massive amount of IoT data are generated consequently. However, due to the unique characteristics of IoT data, data analytics in IoT systems is not identical to the conventional big data analytics. Thus, the characteristics of IoT data are firstly identified in this subsection. The renowned properties of big data are the three Vs, namely, volume, velocity, and variety, as depicted in Figure 5.2. Though they have three Vs in common, IoT data still have several aspects different from the conventional big data. The unique characteristics of IoT data are listed as follows.

- Large scale: With the pervasive deployments of large-scale IoT systems, a large number of IoT end devices are involved in the systems and continuously generate a massive amount of data. In most of IoT systems, not only the real-time data but also the historical data are needed to provide the descriptions of user patterns, environmental trends, etc. Thus, both the real-time and historical data have to be processed, analyzed and stored in the IoT systems, which finally labels the characteristic of large scale to IoT data.
- Heterogeneity: The sensing layer of an IoT system is in high diversity, which comprises heterogeneous devices and subnets. Different from the traditional homogeneous wireless networks, data generated by the heterogeneous IoT devices are not identical in formats and even unstructured, which finally results in heterogeneity.
- Temporal and spatial correlation: IoT data are generally labeled with both location information and timestamp, as most of the IoT systems are context-aware. The labeled IoT data are highly correlated in temporal and spatial domains because the environmental parameters sensed and sampled by the IoT end devices are varied in mild trends. Providing the statistical characteristic of temporal and spatial correlation, IoT data can be easily processed with the statistical tools and the machine learning methods.
- Taint: Due to the low-cost feature of IoT end devices, these tiny devices are vulnerable to different kinds of attacks and also inner malfunctions, which can finally lead to abnormal IoT data. Therefore, data pre-processing, particularly data cleaning, is generally needed before eventually performing data analysis.

Sensor Data Aggregation in IoT Systems:

IoT is emerging as the underlying technology of our connected society, which enables many advanced applications. In IoT-enabled applications, information of application surroundings is gathered by networked sensors, especially wireless sensors due to their advantage of infrastructurefree deployment. However, the pervasive deployment of wireless sensor nodes generates a massive amount of sensor data, and data outliers are frequently incurred due to the dynamic nature of wireless channels. As the operation of IoT systems

relies on sensor data, data redundancy and data outliers could significantly reduce the effectiveness of IoT applications or even mislead systems into unsafe conditions. In this chapter, a cluster-based data analysis framework is proposed using R-PCA, which can aggregate the redundant data and detect the outliers in the meantime. More specifically, at a cluster head, spatially correlated sensor data collected from cluster members are aggregated by extracting the PCs, and potential data outlier is determined by the abnormal SPE score, which is defined as the square of residual value after extraction of PCs. With R-PCA, the parameters of the PCA model can be recursively updated to adapt to the changes in IoT systems. The cluster-based data analysis framework also relieves the computational and processing burdens on sensor nodes.

Spatial Correlation based Data Outlier Detection:

Existing studies on spatial correlation based outlier detection can be summarized into the following categories.

- Majority Voting is a classical spatial correlation based data outlier detection method. A local sensor node is detected as abnormal when its reading is substantially different from the majority of its neighbors. For instance, in distributed fault detection (DFD) algorithm, general and differential Euclidean distances between sensor data generated from local sensor node and its neighbors were used as outlier detection criteria. The local node was detected as abnormal when the Euclidean distances between most of its neighbors were over a certain threshold. However, due to the excessive dependence of a sensor node on its neighbors, the data outlier detection accuracy was low when the network was sparse.
- Classifiers are applied to detect data outlier by training a “normal” model and then classifying the under detecting data into normal and abnormal. Support vector machine (SVM) is among the most commonly used classifiers, especially the lightweight quarter-sphere SVM. Generally, the radius of the quarter sphere is trained by “normal” data, and any sensor data that falls out of the quarter sphere is detected as abnormal. The major concern of the classifier-based outlier detection is its high computational complexity, since the local-based algorithms are processed at sensor nodes.

Spatial Correlation based Sensor Data Aggregation:

- In Conventional Data Aggregation algorithms, sensor nodes are clustered by spatial correlation, but sensor data are simply aggregated by basic operations, such as mean and median, without full exploitation of data correlation and accuracy for data aggregation. For instance, a trust-based framework for data aggregation in WSNs. Every sensor data was assigned a weight, according to the trustworthiness ranked by comparison with historical data and neighbor data. Afterwards, the weighted mean value calculated at the aggregator was used as the aggregated data.
- Compressive Sensing (CS) transforms raw sensor data into the sparse domain at the sender, to reduce the overall communication overload, while increases the complexity of receiver for data recovery. CS-based data aggregation scheme. Particularly, they adopted diffusion wavelets to sparsify the raw sensor data, which further reduced the communication overload while the data recovery faced high computational complexity. Besides, sparsity might exist in the environments, but the compressive sensing method was still limited by the restricted isometry property.
- PCA has been widely used to aggregate sensor data, due to the essence of principal component extraction. A PCA-based hierarchical data aggregation algorithm was proposed. At each level, sensor data from the lower level was aggregated and forwarded. However, only the temperature reading was investigated without consideration of multivariate sensor readings in current IoT systems. Considering the multivariate data aggregation, a novel principal components-based context compression (PC3) algorithm was proposed. PC3 algorithm was also able to adaptively update the transformation basis of PCA by alternating the learning and compression operations at sensor nodes. However, PC3 was too complex to be deployed on a sensor node with limited computational capacity. In this chapter, an improved R-PCA algorithm is proposed relying on cluster processing, which recursively updates the transformation basis with only the newest data so that the memory occupied by historical data incould be released.

Data Sampling at Sensor Nodes:

Given the limited computational capacity and power supply of a sensor node, a sensor node is designed to be simply responsible for sampling the application surroundings and transmitting the sampled sensor data to its cluster head in the proposed data analysis framework. Considering the general condition that multiple sensors are embedded on one sensor node, the data matrix generated by a sensor node i is multivariate and mathematically expressed as

$$\mathbf{X}_i = \begin{bmatrix} x_{i,1}(1) & x_{i,1}(2) & \dots & x_{i,1}(n) \\ x_{i,2}(1) & x_{i,2}(2) & \dots & x_{i,2}(n) \\ \vdots & \vdots & \ddots & \vdots \\ x_{i,m}(1) & x_{i,m}(2) & \dots & x_{i,m}(n) \end{bmatrix},$$

where m is the number of physical variables (like temperature and humidity), while n is the number of samples. At a certain time t, data vector generated by sensor node i is

$Xi(t) = [xi,1(t); xi,2(t); : : : ; xi,m(t)]^T$

5.5 Detecting Anomalies:

The Internet of Things (IoT) refers to a network of billions of interconnected devices thathave the ability to communicate and exchange data over the Internet. Such IoT devices range from sensors, smart phones, computers, vehicles, building appliances, and health devices. The extension of Internet connectivity to physical devices and everyday objects has substantially increased worldwide real-time data collection and transmission. As of 2019 there are approximately 9 billion IoT devices across the world and by 2020 this number will surge to over 25 billion. These IoT edge devices, such as smart light bulbs, wearable medical devices, doors, heaters, sensors for smart agriculture, etc., which typically host a variety of sensors for temperature, pressure, humidity, light, motion and acceleration, are often resource-constrained. For example, many of them are battery-powered, with limited processing power which is just sufficient for the task at hand so that they can be mass-produced while minimizing costs. On the other hand, the amount of data produced by these sensors at this scale is staggering. As IoT devices grow in number, the tremendous amount of sensing data collected hasraised great challenges for data transmission overhead (time and energy) and cloud storage. Applications of cost-cutting edge mining techniques to reduce

packet transmission and remote storage requirements are rapidly growing throughout IoT networks. One of the most basic edge mining techniques is random sampling to reduce the number of observations sent to the cloud. More sophisticated methods, such as anomaly detection, isolate and transmit only the contextually relevant observations. The guiding principle behind anomaly detection is that only unexpected behavior needs to be notified to the centralized cloud. Contemporary works specify different anomaly detection methods ranging from basic thresholding to machine learning algorithms. However, as the resources available at each IoT edge device can be rather limited in terms of power, memory, connectivity, bandwidth, and computation, it is critical to choose appropriate anomaly detection algorithms that can not only effectively identify abnormal behaviors but also consume limited resources at the edge devices.

Anomaly detection is one of the most popular edge mining techniques explored in IoT scenarios anomaly detection is used as a method to implement an Intrusion Detection System (IDS) for Wireless Sensor Networks (WSN). The use of LDA to reduce the dimensionality of network intrusion datasets and applies both Naive Bayes and KNN algorithms for anomaly classification. The performance of anomaly detection (i.e., false positive rates) using an unsupervised outlier detection technique based on the RF algorithm. Furthermore, the effectiveness of autoencoders for an unsupervised IDS and proposes a novel splitting and learning mechanism to lower false positive detection. Although these works explore novel applications of anomaly detection on the IoT edge, they do not focus on resource constrained scenarios and therefore do not delve into the time and energy consumption of these methods. Inanomaly detection is investigated in healthcare applications. Cardiac anomaly detection with low false negative counts and stress the importance of capturing outliers in healthcare applications. Similar to the IDS studies, this work focuses extensively on anomaly detection implementation but does not consider the factor of resource consumption. Several works also tackle anomaly detection in IoT applications outside of IDS and healthcare. In non-machine learning anomaly detection algorithms are proposed for a set of heterogenous sensors in an IoT WSN. In an autoencoder neural network is used for determining anomaly readings from a testbed of eighttemperature and humidity sensors.

Anomaly Detection Models And Datasets:

Random Forests RF models are applicable to a wide range of classification problems. In a random forest, each node is split using a subset of features randomly chosen at that node. This strategy is robust against overfitting and enables RF to perform better than many other classifiers, including discriminant analysis, support vector machines, and neural networks. The only major downside of RF is that a large number of trees can slow down the algorithm for real-time predictions.

Multilayer Perceptron MLP is the most known and frequently used type of neural network using the backpropagation training algorithm. In recent years, neural networks have been extensively used for pattern recognition and optimization. MLP models contain three types of layers: input layer, output layer, and hidden layer. Each node in the input layer, from top to bottom, passes an input data point to each neuron in the first hidden layer. Then, each hidden layer neuron multiplies each value with a weight vector and computes the sum of the multiplied values. Subsequently each hidden layer neuron applies its activation function to this sum, and sends the resulting value to the next layer and eventually to the output layer. Suppose there are N training samples, d features, k hidden layers each containing h neurons, o output neurons, and i iterations. It is advisable to start with a small number of hidden layers and nodes given the high time complexity of MLP's backpropagation algorithm and time-consuming grid search procedure. One of the most notable advantages of MLP, is its low prediction complexity, O(Nhk), which makes it suitable for time-sensitive anomaly detection tasks.

K-Nearest Neighbors K-Nearest Neighbor Classifier (KNN) is a relatively simple learning algorithm. It is very commonly used in text mining, agricultural predictions, and stock market forecasting. Because KNN does not make any assumptions about the underlying data distribution, it is particularly suitable for applications with little or no prior knowledge about the distribution of the dataset. KNNs are generally reputed for high prediction accuracy with respect to precision and recall. Furthermore, the training phase of KNN classifiers is very efficient. Nevertheless, there are several drawbacks of KNNs. First, the model has high space complexity

as it stores all training data instances. Second, finding the most optimal value for K is not trivial. KNN's most significant drawback, however, is its exorbitant prediction phase overhead, which is O(NPMlogK) with N training samples, M test samples, K neighbors, and P distance metric time complexity. The high overhead at the prediction phase may make it less suitable to be carried on by resource constrained IoT edge devices.

Discriminant Analysis In this work, we use both LDA and QDA. LDA first projects a dataset onto lower-dimensional space to prevent overfitting and to generate linear class-separability. QDA also performs dimensionality reduction but generates a quadratic line to fit the training data. Despite its potential for non-linear data patterns, the number of the parameters needed by QDA scales quadratically with that of the variables, making it slower for very high dimensional datasets. Thus,the use of anomaly detection as an impactful edge mining technique in different IoT scenarios.

Questions for Practice:

Part-A

1. Tabulate the different models of Edge Computing.
2. Formulate Edge and Fog Computing for IoT devices
3. Integrate Threshold Prediction.
4. Design a Filter Structure for Edge Computing.
5. Design a Filter Structure for Fog Computing.
6. Discuss the features of Detecting Anomalies with real time applications.

Part-B

1. Design a Mathematical Description for change in Down Sampling of Data
2. Elaborate Edge and Fog Computing for Gateway Design
3. Design a filter that implements Edge and Fog Computing.

4. Design a Structure for Edge Analytics at devices and Gateways

5. Identify how aggregations differ from Protocols

6. Create a methodology how Edge and Fog Computing act as a key player in IoT techniques with a real time application

References:

1. Mischa Dohler, Carles Anton-Haro, “Machine-to-machine (M2M) Communications”, Woodhead Publishing, 1st Edition, 2014.
2. Perry Lea, “Internet of Things for Architects: Architecting IoT solutions by implementing sensors, communication infrastructure, edge computing, analytics, and security”, Packet, 1st Edition, 2018.
3. Maciej Kranz , “Building the Internet of Things”, John Wiley & Sons, 1st Edition, 2017.
4. David Boswarthick, Omar Elloumi, Olivier Hersent, “M2M Communications: A Systems Approach”, John Wiley & Sons, 1 st Edition, 2012.
5. Anupama C. Raman and Pethuru Raj, “The Internet of Things: Enabling Technologies, Platforms, and Use Cases”, CRC press, 1st Edition, 2017.