layers in IoT have many similarities. But there are differences in the protocols used at various layers of IoT. Figure 2.6 shows the comparison of the protocols of IoT stack and Web stack.

We have already seen the four layers of IoT in Chapter 1. The four layers include the sensing layer, network layer, service layer and interface layer. The sensing layer in IoT is equivalent to physical layer and data link layer of OSI reference model. The protocols used in IoT are IEEE 802.15.4 MAC and IEEE 802.15.4 PHY/Radio for this layer while protocols for OSI layer are Ethernet, DSL, ISDN, Wireless LAN, Wi-Fi. Similarly in the network layer, the protocols used are IPv6 and 6LowPAN in IoT whereas the protocols used in OSI are IPv4, IPv6 and IPSec. Again the protocols used in service layer in IoT are UDP and DTLS while the protocols in transport layer for web are TCP and UDP. The application layer in OSI is similar to the interface layer in IoT. The protocols used in IoT and web are different as they are intended for different purposes. Unlike web stack, we can see in addition to the protocols in IoT, there is a management component in IoT stack. This is essential because there are a large number of devices, networks and other resources which needs to be managed efficiently.

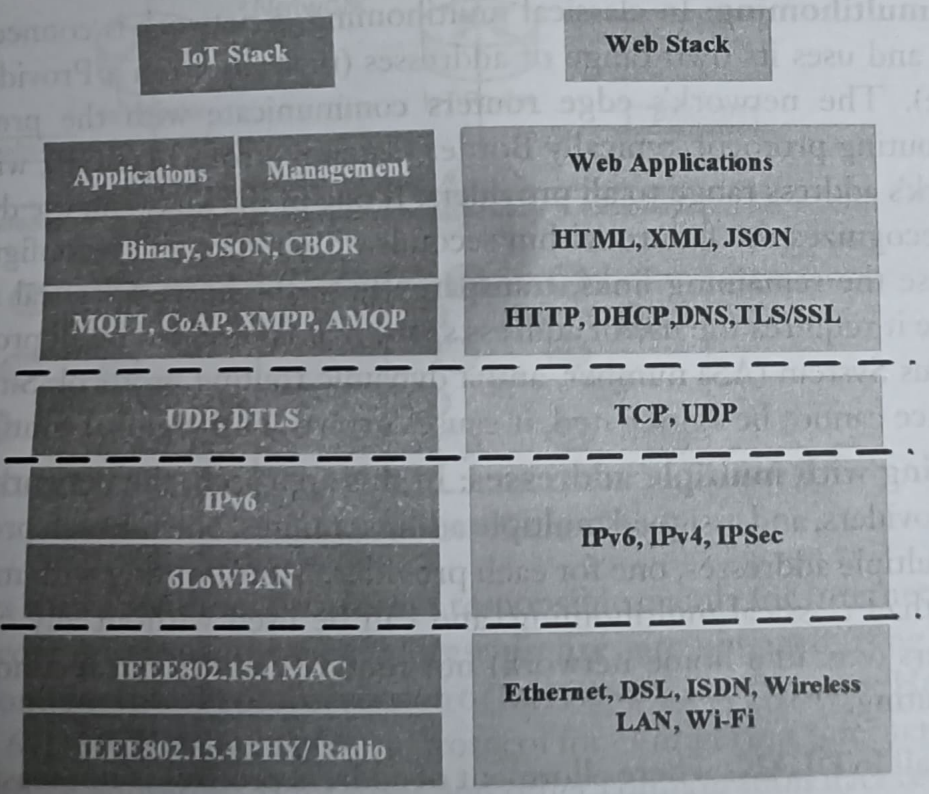

Fig.2.6: Comparisons of IoT and Web Stacks

2.6 IoT Identification and Data Protocols

The Internet of Things covers a huge range of industries and use cases that scale from a single constrained device up to massive cross-platform deployments of embedded technologies and cloud systems connecting in real-time. There are all together numerous legacy and emerging communication protocols that allow devices and servers to talk to each other in new, more

interconnected ways. Rather than trying to fit all of the IoT protocols on top of existing architecture models like OSI Model, we have broken the protocols into the following layers to provide some level of organization.

Infrastructure (ex: 6LowPAN, IPv4/IPv6, RPL)
Identification (ex: EPC, uCode, IPv6, URIs)
Comms / Transport (ex: Wi-Fi, Bluetooth, and LPWAN)
Discovery (ex: Physical Web, mDNS, DNS-SD)
Data Protocols (ex: MQTT, CoAP, AMQP, Websocket, Node)
Device Management (ex: TR-069, OMA-DM)
Semantic (ex: JSON-LD, Web Thing Model)
Multi-layer Frameworks (ex: Alljoyn, IoTivity, Weave, Homekit)

In this Chapter we will be discussing about infrastructure protocols like IPv4, IPv6 and data protocols like MQTT, CoAP, AMQP etc. Other commonly used protocols will be discussed in the upcoming Chapter.

2.6.1 IPv4

Internet Protocol Version 4 (IPv4) is the fourth revision of the Internet Protocol and a widely used protocol in data communication over different kinds of networks. IPv4 is a connectionless protocol used in packet-switched layer networks, such as Ethernet. It provides the logical connection between network devices by providing identification for each device. There are many ways to configure IPv4 with all kinds of devices – including manual and automatic configurations – depending on the network type.

IPv4 is based on the best-effort model. This model guarantees neither delivery nor avoidance of duplicate delivery. These aspects are handled by the upper layer transport. IPv4 is defined and specified in IETF publication RFC 791. It is used in the packet-switched link layer in the OSI model. IPv4 uses 32-bit addresses for Ethernet communication in five classes: A, B, C, D and E. Classes A, B and C have a different bit length for addressing the network host. Class D addresses are reserved for multicasting, while class E addresses are reserved for future use.

Class A has subnet mask 255.0.0.0 or /8, B has subnet mask 255.255.0.0 or /16 and class C has subnet mask 255.255.255.0 or /24. For example, with a /16 subnet mask, the network 192.168.0.0 may use the address range of 192.168.0.0 to 192.168.255.255. Network hosts can take any address from this range; however, address 192.168.255.255 is reserved for broadcast within the network. The maximum number of host addresses IPv4 can assign to end users is 232. IPv6 presents a standardized solution to overcome IPv4's limitations. Because of its 128-bit address length, it can define up to 2,128 addresses.

Presently, the Internet is mainly IPv4 based with little or no IPv6 uplink facilities or support. Due to the lack of a universal solution to IPv6, lots of un-optimized solutions are

being used for IoT deployment. These makeshift solutions mainly address IPv6 to IPv4 translation, IPv6 tunneling over IPv4 and application layer proxies like data relaying. Figure 2.7 shows an IPv4 header format.

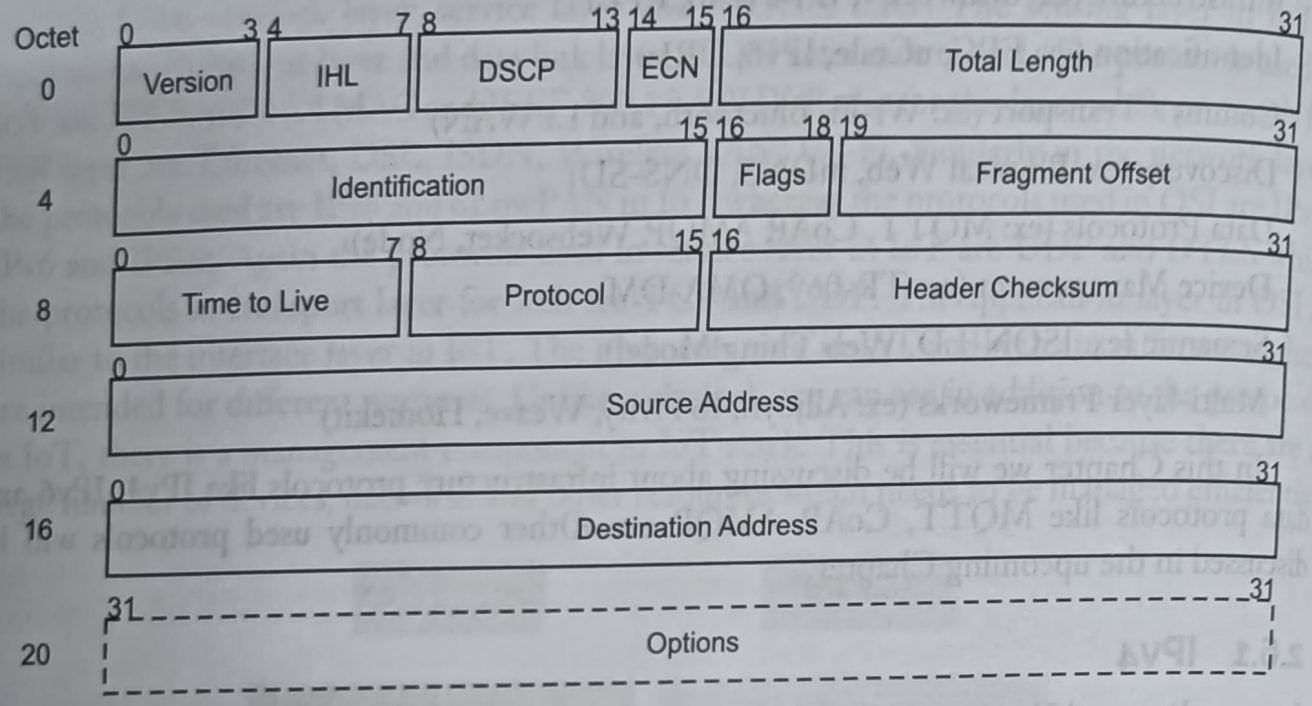

Fig.2.7: IPv4 Header Format

IPv4 header includes several relevant information as follows.

Version: Version no. of Internet Protocol used (e.g. IPv4).

IHL: Internet Header Length. Gives the length of entire IP header.

DSCP: Differentiated Services Code Point. This is a type of service.

ECN: Explicit Congestion Notification. It carries information about the congestion seen in the route.

Total Length: Length of entire IP Packet (including IP header and IP Payload).

Identification: If IP packet is fragmented during the transmission, all the fragments contain same identification number. This is to identify original IP packet they belong to.

Flags: As required by the network resources, if IP Packet is too large to handle, these 'flags' tells if they can be fragmented or not. In this 3-bit flag, the MSB is always set to '0'.

Fragment Offset: This offset tells the exact position of the fragment in the original IP Packet.

Time to Live: To avoid looping in the network, every packet is sent with some TTL value set, which tells the network how many routers (hops) this packet can cross. At each hop, its value is decremented by one and when the value reaches zero, the packet is discarded.

Protocol: Tells the Network layer at the destination host, to which Protocol this packet belongs to, i.e. the next level Protocol. For example protocol number of ICMP is 1, TCP is 6

and UDP is 17.

Header Checksum: This field is used to keep checksum value of entire header which is then used to check if the packet is received error-free.

Source Address: 32-bit address of the Sender (or source) of the packet.

Destination Address: 32-bit address of the receiver (or destination) of the packet.

Options: This is optional field, which is used if the value of IHL is greater than 5. These options may contain values for options such as security, record route, time stamp, etc.

IPv4 emphasizes more on reliable transmission, as is evident by fields such as type of service, total length, identification, fragment offset, TTL and checksum fields.

2.6.2 IPv6

IPv6 (Internet Protocol version 6) is a set of specifications from the Internet Engineering Task Force (IETF) that's essentially an upgrade of IP version 4 (IPv4). The basics of IPv6 are similar to those of IPv4 -- devices can use IPv6 as source and destination addresses to pass packets over a network, and tools like ping work for network testing as they do in IPv4, with some slight variations. The most obvious improvement in IPv6 over IPv4 is that IP addresses are lengthened from 32 bits to 128 bits. This extension anticipates considerable future growth of the Internet and provides relief for what was perceived as an impending shortage of network addresses. IPv6 also supports auto-configuration to help correct most of the shortcomings in version 4, and it has integrated security and mobility features.

IPv4 is Internet Protocol version 4 and it describes what the numbers used by every device that connects to the Internet must look like (for example, 192.168.0.254). Based on this model, the early Internet architects made an estimated 4.3 billion numbers available and at the beginning of the Internet age, that number probably seemed inexhaustible. However, over the last couple of decade's network, cloud, and mobile technologies have rapidly reduced the number of available IP addresses as each new technology added devices to the global Internet. When the IP addresses inventory first began to shrink and it appeared the IPv4 supply of numbers were being consumed at a higher than expected rate, plans were devised to transition to a new protocol called IPv6 that would provide for a significantly larger supply of numbers. Although implementing IPv6 would solve a problem that has been known for over a decade, no one has been in a hurry to adopt the new standard.

The reason for this is that to make the switch to IPv6 would require anyone who connects to the Internet to move from the IPv4 protocol to IPv6. IPv4 and IPv6 are not compatible and finding ways to run both protocols without affect users, businesses, and service providers. Still, the transition to IPv6 will solve the problem of running out of available IP addresses. The number of IPv4 addresses in the pool is 232 or 4,294,967,296 (approximately 4.29 billion). Switching to IPv6 would make 2128 or 340 Undecillion (or 340,282,366,920,938,000,000,0 00,000,000,000,000,000) addresses available. With so many addresses available, it would be unlikely that a transition of this type would ever need to happen again.

Figure 2.8 shows the IPv6 header format.

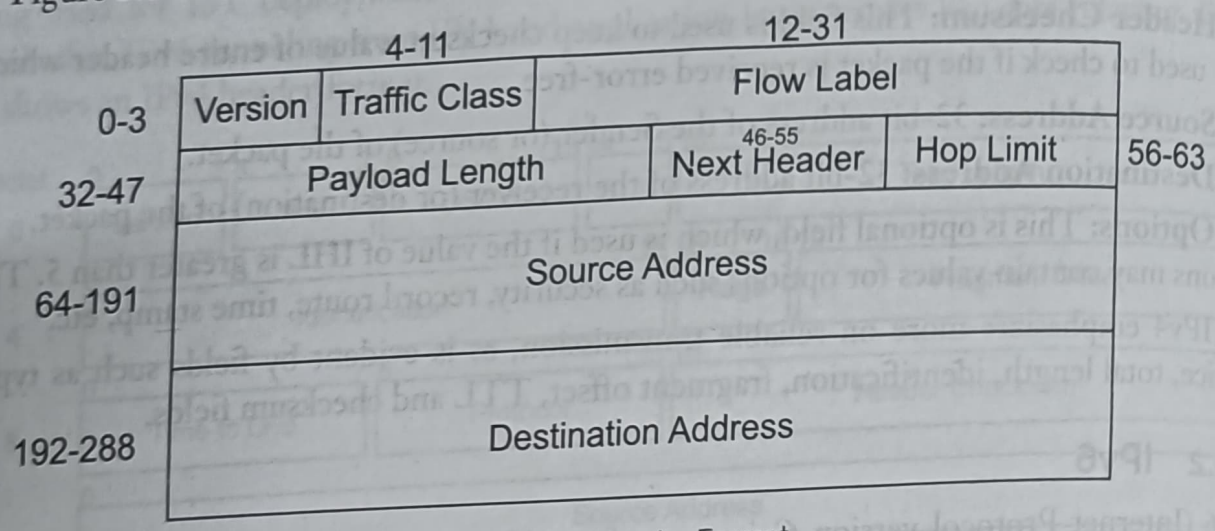

Fig.2.8: IPv6 Header Format

IPv6 header includes several relevant information as follows.

Version (4-bits): It represents the version of Internet Protocol, i.e. 0110.

Traffic Class (8-bits): These 8 bits are divided into two parts. The most significant 6 bits are used for Type of Service to let the router known what services should be provided to this packet. The least significant 2 bits are used for Explicit Congestion Notification (ECN).

Flow Label (20-bits): This label is used to maintain the sequential flow of the packets belonging to a communication. The source labels the sequence to help the router identify that a particular packet belongs to a specific flow of information. This field helps avoid re-ordering of data packets. It is designed for streaming/real-time media.

Payload Length (16-bits): This field is used to tell the routers how much information a particular packet contains in its payload. Payload is composed of Extension Headers and Upper Layer data. With 16 bits, up to 65535 bytes can be indicated; but if the Extension Headers contain Hop-by-Hop Extension Header, then the payload may exceed 65535 bytes and this field is set to 0.

Next Header (8-bits): This field is used to indicate either the type of Extension Header, or if the Extension Header is not present then it indicates the Upper Layer PDU. The values for the type of Upper Layer PDU are same as IPv4's.

Hop Limit (8-bits): This field is used to stop packet to loop in the network infinitely. This is same as TTL in IPv4. The value of Hop Limit field is decremented by 1 as it passes a link (router/hop). When the field reaches 0 the packet is discarded.

Source Address (128-bits): This field indicates the address of originator of the packet.

Destination Address (128-bits): This field provides the address of intended recipient of the packet.

IPv6 header structure is simpler as it mainly focuses on the addressing part of the source

and destination. It is concerned more with addressing than with reliability of data delivery. A comparison of IPv4 and IPv6 is given in Figure 2.9.

	IPv4	IPv6
Developed by	IETF 1974	IEF 1998
Length (bits)	32	128
No: of addresses	2^{32}	2^{128}
Notation	Dotted Decimal	Hexadecimal
Dynamic allocation	DHCP	SLAAC/DHCPv6
IPSec	Optional	Compulsory
Header Size	Variable	Fixed
Header Checksum	Yes	No
Header Options	Yes	No
Broadcast Addresses	Yes	No
Multicast Address	No	Yes

Fig.2.9: IPv4 versus IPv6

2.6.3 MQTT

MQTT or Message Queuing Telemetry Transport is an ISO standard (ISO/IEC PRF 20922). It is a publish-subscribe-based light weight messaging protocol used in conjunction with TCP/IP protocol. It works on top of the TCP/IP protocol. It is designed for connections with remote locations or the network bandwidth is limited. MQTT was introduced by IBM in 1999 and standardized by OASIS in 2013. It is designed to provide connectivity mostly embedded between applications and middlewares on one side and networks & communications on the other side. Figure 2.10 shows the components of MQTT.

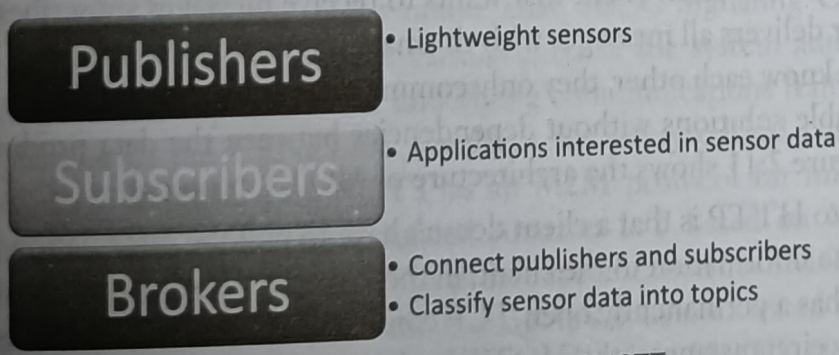

Fig.2.10: Components of MQTT

Publishers are usually light weight sensors and subscribers are applications which are interested in sensor data. A topic to which a client/subscriber is subscribed is updated in the form of messages and distributed by the message broker. MQTT client can be any device, from a micro controller to a fully-fledged server, which runs the MQTT library and is connected

to MQTT broker over any network. MQTT client libraries are available across a wide variety of programming language platforms like C, C++, C#, Java, JavaScript, Android, iOS etc. A message broker controls the publish-subscribe messaging pattern. Brokers connect publishers and subscribers. MQTT Broker is responsible for receiving all messages, filtering, decision making and sending messages to subscribed clients. MQTT is based on TCP/IP, hence both client and broker is expected to have TCP/IP stack.

Methods in MQTT

MQTT methods are also referred to as verbs. MQTT defines methods to indicate desired actions to be performed on identified resources. Resources can be files or the outputs of an executable program found on a server.

(a) **Connect** – Waits for connection to be established with the server.

(b) **Disconnect** – Waits for the MQTT client to finish any work, which needs to be done and for the TCP/IP session to disconnect.

(c) **Subscribe** – Requests the server to let the client subscribe to one or more topics.

(d) **Unsubscribe** – Requests the server to let the client unsubscribe from one or more topics.

(e) **Publish** – Publishes the data from sensors or end devices. Returns immediately to application thread after passing request to the MQTT client.

Working of MQTT

The protocol uses a publish/subscribe architecture in contrast to HTTP with its request/response paradigm. Publish/Subscribe is event-driven and enables messages to be pushed to clients. The central communication point is the MQTT broker, which is in charge of dispatching all messages between the senders and the rightful receivers. Each client that publishes a message to the broker includes a topic into the message. The topic is the routing information for the broker. Each client that wants to receive messages subscribes to a certain topic and the broker delivers all messages with the matching topic to the client. Therefore the clients don't have to know each other, they only communicate over the topic. This architecture enables highly scalable solutions without dependencies between the data producers and the data consumers. Figure 2.11 shows the architecture of MQTT protocol.

The difference to HTTP is that a client doesn't have to pull the information it needs, but the broker pushes the information to the client, in the case there is something new. Therefore each MQTT client has a permanently open TCP connection to the broker. If this connection is interrupted by any circumstances, the MQTT broker can buffer all messages and send them to the client when it is back online.

The central concepts in MQTT to dispatch messages are topics. A topic is a simple string that can have more hierarchy levels, which are separated by a slash. A sample topic for sending temperature data of the living room could be ***house/living-room/temperature***. On one hand the client can subscribe to the exact topic or on the other hand use a **wildcard**.

The subscription to ***house/+/temperature*** would result in all message send to the previously mentioned topic ***house/living-room/temperature*** as well as any topic with an arbitrary value in the place of living room, for example ***house/kitchen/temperature***. The plus sign is a single level wild card and only allows arbitrary values for one hierarchy. If you need to subscribe to more than one level, for example to the entire sub-tree, there is also a **multilevel wildcard (#)**. It allows subscribing to all underlying hierarchy levels. For example ***house/#*** is subscribing to all topics beginning with house.

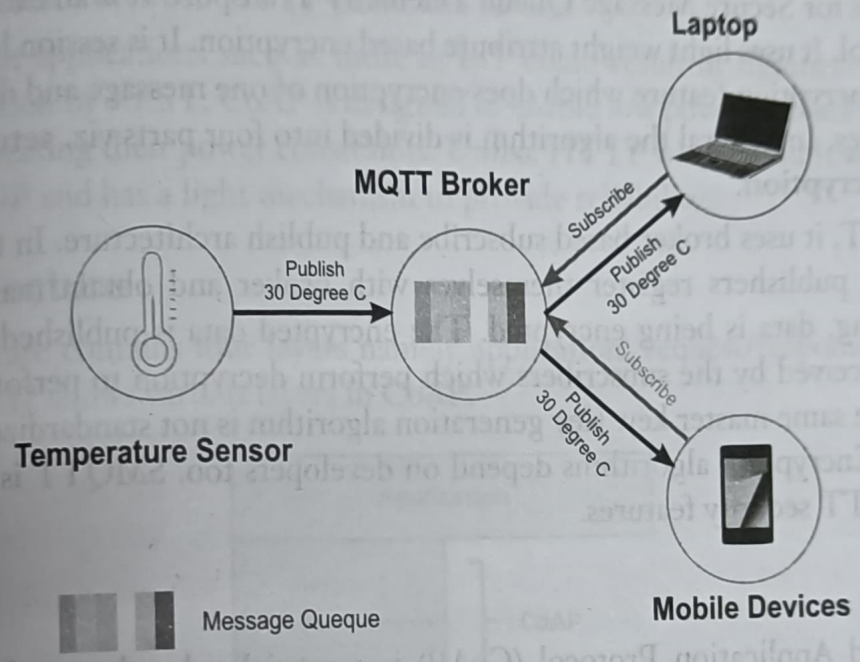

Fig.2.11: Architecture of MQTT

Applications of MQTT

There are several projects that implement MQTT. Facebook Messenger has used aspects of MQTT in Facebook Messenger for online chat. IECC Signaling Control System uses MQTT for communications within the various parts of the system and other components of the signaling system. It provides the underlying communications framework for a system that is compliant with the CENELEC standards for safety-critical communications. The EVRYTHNG IoT platform uses MQTT as an M2M protocol for millions of connected products. Amazon Web Services announced Amazon IoT based on MQTT. The Open Geospatial Consortium SensorThings API standard specification has a MQTT extension in the standard as an additional message protocol binding. It was demonstrated in a US Department of Homeland Security IoT Pilot. Ad fruit launched a free MQTT cloud service for IoT experimenters and learners called Ad fruit IO. Microsoft Azure IoT Hub uses MQTT as its main protocol for telemetry messages. XIM, Inc. launched an MQTT client called MQTT Buddy in 2017. It's a MQTT app for Android and iOS, but not F-Droid, users available in English, Russian and Chinese languages. Open-source software home automation platform Home Assistant is MQTT enabled and offers four options for MQTT brokers. Pimatic

home automation framework for Raspberry Pi and based on Node.js offers MQTT plug in providing full support for MQTT protocol. McAfee Open DXL is based on MQTT with enhancements to the messaging brokers themselves so that they can intrinsically understand the DXL message format in support of advanced features such as services, request/response (point-to-point) messaging and service zones.

SMQTT

SMQTT stands for Secure Message Queue Telemetry Transport. It is an extension to simple MQTT protocol. It uses light weight attribute based encryption. It is session layer protocol. It has broadcast encryption feature which does encryption of one message and delivers the same to multiple nodes. In general the algorithm is divided into four parts viz. **setup, encryption, publish** and **decryption.**

Like MQTT, it uses broker based subscribe and publish architecture. In the setup phase, subscribers and publishers register themselves with broker and obtain master secret key. Before publishing, data is being encrypted. The encrypted data is published by the broker. This is being received by the subscribers which perform decryption to perform decoding of the data with the same master key. Key generation algorithm is not standardized and depends on developers. Encryption algorithms depend on developers too. SMQTT is proposed only to enhance MQTT security features.

2.6.4 CoAP

The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use with constrained nodes and constrained (e.g., low-power, lossy) networks. The nodes often have 8-bit microcontrollers with small amounts of ROM and RAM, while constrained networks such as IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs) often have high packet error rates and a typical throughput of 10s of Kbit/s. 6LoWPANs will be discussed in the next Chapter. The protocol is designed for machine- to-machine (M2M) applications such as smart energy and building automation.

Features of CoAP

Following are the features of CoAP.

(a) Overhead and parsing complexity.

(b) URI and content-type support.

(c) Support for the discovery of resources provided by known CoAP services.

(d) Simple subscription for a resource and resulting push notifications.

(e) Simple caching based on maximum message age.

CoAP provides a request/response interaction model between application end points. It supports built-in discovery of services and resources and includes key concepts of the Web

such as URIs and Internet media types. CoAP is designed to easily interface with HTTP for integration with the Web while meeting specialized requirements such as multicast support, very low overhead and simplicity for constrained environments. In CoAP, client-server interaction is asynchronous over a datagram oriented transport protocol such as User Datagram Protocol (UDP). The Constrained Application Protocol (CoAP) is a session layer protocol designed by IETF constrained RESTful Environment (CORE) working group to provide light weight RESTful interface. REST is Representational State Transfer, which is the standard interface between HTTP client and servers.

Light weight applications such as those in IoT could result in significant overhead and power consumption by REST. CoAP is designed to enable low power sensors to use RESTful services while meeting their power constraints. Unlike HTTP which is built with TCP, CoAP is built with UDP and has a light mechanism to provide reliability.

CoAP Architecture

CoAP architecture contains four layers namely application, request/response, messages and UDP. Figure 2.12 shows various layers in CoAP.

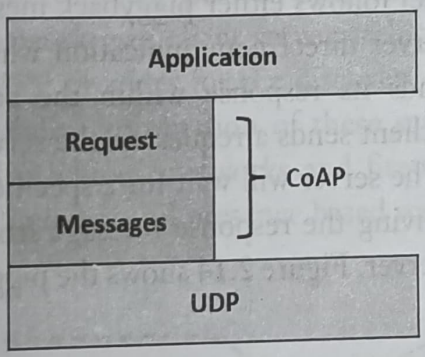

Fig.2.12: Layers in CoAP

The two main sub layers in CoAP are messaging and request/response sub layers. The messaging sub layer is responsible for reliability and duplication of messages while the request/response sub layer is responsible for communication. CoAP has four messaging modes.

1. Confirmable
2. Non-confirmable
3. Piggyback
4. Separate

Confirmable messaging mode represents reliable transmission whereas non-confirmable messaging mode represents unreliable transmission. Figure 2.13 shows the confirmable and non-confirmable messaging.

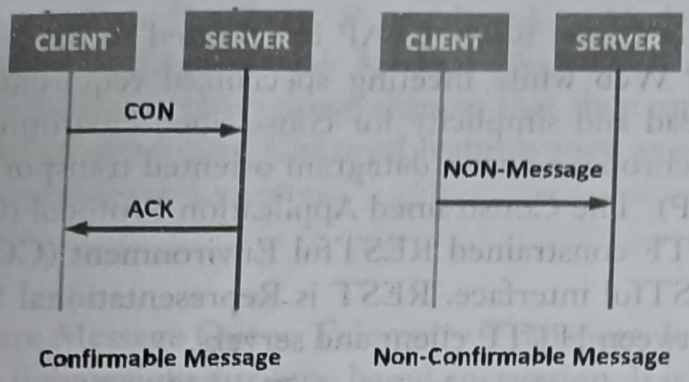

Fig.2.13: Confirmable and Non-confirmable Messaging

In confirmable messaging, the client keeps retransmission until it gets ACK with the same message ID. The client uses a default time out and decreases counting time exponentially when transmitting CON. If recipient fail to process message, it responses by replacing ACK with RST (Reset). In non-confirmable messaging, the client sends a message and does not wait for the acknowledgement. It doesn't need to be ACKed, but has to contain message ID for supervising in case of retransmission. If recipient fail to process message, server replies RST.

The request/response model follows either piggyback messaging or separate messaging. Piggyback is used for client/server direct communication where the client sends a request to the server. The server sends its response within the acknowledgement message. In separate messaging mode, the client sends a request to the server. The server sends back the acknowledgement separately. The server will wait for a specified time interval and then send the response message. On receiving the response message from the server, the client sends back acknowledgement to the server. Figure 2.14 shows the piggyback and separate messaging modes.

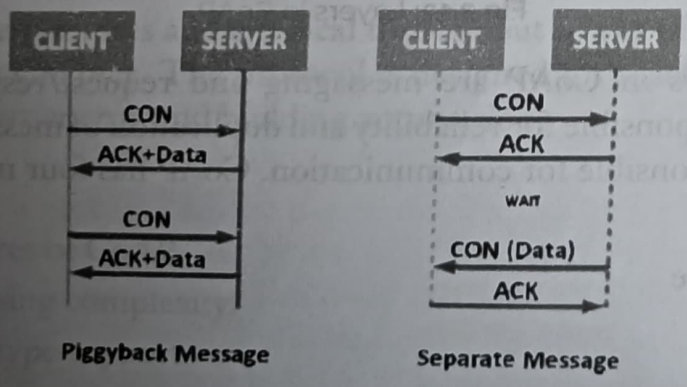

Fig.2.14: Piggyback and Separate Messaging

In piggybacking, client sends request using CON type or NON type message and receives response ACK with confirmable message immediately. For successful response, ACK contain response message (identify by using token), and for failure response, ACK contain failure response code. In separate message mode, if server receives a CON type message but not able to response this request immediately, it will send an empty ACK. In this case, client resends

this message. When server is ready to response this request, it will send a new CON to client and client reply a confirmable message with acknowledgment. ACK is just to confirm CON message, no matter CON message carry request or response. Similar to HTTP, CoAP utilizes GET, PUT, PUSH and DELETE messages to retrieve, create, update and delete messages respectively.

2.6.5 XMPP

Extensible Messaging and Presence Protocol (XMPP) is a communication protocol for message-oriented middleware based on XML (Extensible Markup Language). It enables the real time exchange of structured yet extensible data between any two or more network entities. Designed to be extensible, the protocol has been used also for publish-subscribe systems, signaling for VoIP, video, file transfer, gaming, Internet of Things (IoT) applications such as the smart grid, and social networking services.

Unlike most instant messaging protocols, XMPP is defined in an open standard and uses an open systems approach of development and application, by which anyone may implement an XMPP service and interoperate with other organizations' implementations. Since this is an open standard, it supports M2M or peer-to-peer communications across a diverse set of networks. The XMPP network uses client server architecture. By design, there is no central authoritative server. XMPP provides for the discovery of services residing locally or across a network and the availability information of these services. This is well suited for cloud computing where virtual machines, networks and firewalls would otherwise present obstacles to alternative service discovery and presence based solutions. Figure 2.15 shows the architecture of XMPP.

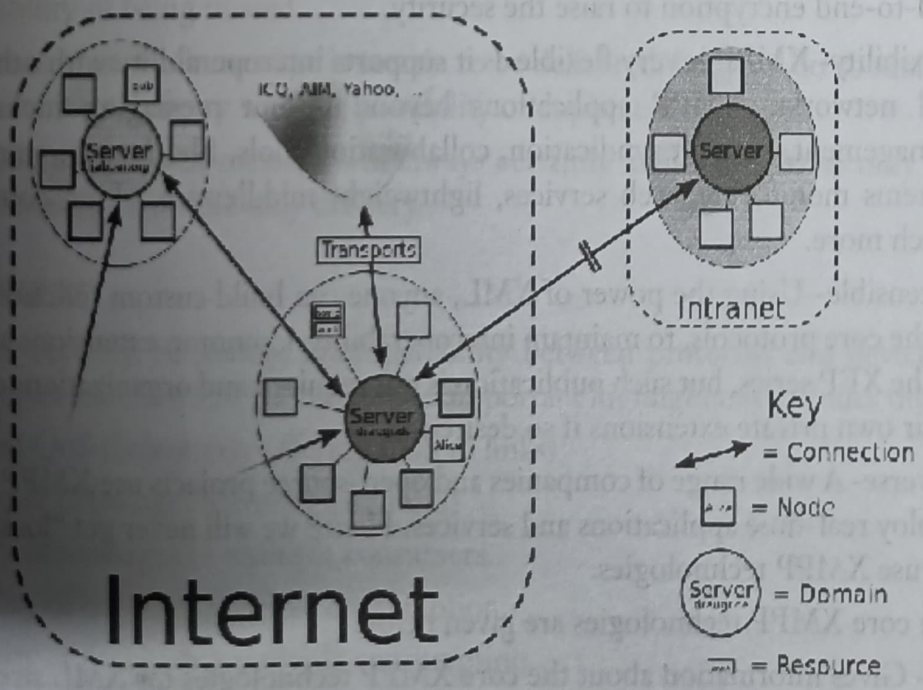

Fig.2.15: Architecture of XMPP

Following are the features of XMPP.

1. Proven – The first Jabber/XMPP technologies were developed by Jeremie Miller in 1998 and are now quite stable. Hundreds of developers are working on this technology. There are tens of thousands of XMPP servers running on the Internet today and millions of people use XMPP for instant messaging through public services such as Google Talk and XMPP deployments at organizations worldwide.
2. Decentralization – There is no central server. Anyone can run their own XMPP server. The architecture of the XMPP network is similar to email. This enables individuals and organizations to take control of their communications experience.
3. Open Standards – The XMPP protocols are free, open, public, and easily understandable. In addition, multiple implementations exist in the form clients, servers, server components, and code libraries. Since this is an open standard, no royalties or granted permissions are required to implement these specifications. The Internet Engineering Task Force (IETF) has formalized the core XML streaming protocols as an approved instant messaging and presence technology. The XMPP specifications were published as RFC 3920 and RFC 3921 in 2004 and the XMPP Standards Foundation continues to publish many XMPP Extension Protocols. In 2011 the core RFCs were revised, resulting in the most up-to-date specifications (RFC 6120, RFC 6121, and RFC 7622).
4. Security – Security in XMPP can be achieved by authentication, encryption etc. Any XMPP server may be isolated from the public network (e.g., on a company intranet) and robust security using SASL and TLS has been built into the core XMPP specifications. In addition, the XMPP developer community is actively working on end-to-end encryption to raise the security.
5. Flexibility- XMPP is very flexible as it supports interoperability with other machines and networks. XMPP applications beyond instant messaging include network management, content syndication, collaboration tools, file sharing, gaming, remote systems monitoring, web services, lightweight middleware, cloud computing, and much more.
6. Extensible- Using the power of XML, anyone can build custom functionality on top of the core protocols, to maintain interoperability. Common extensions are published in the XEP series, but such publication is not required and organizations can maintain their own private extensions if so desired.
7. Diverse- A wide range of companies and open-source projects use XMPP to build and deploy real-time applications and services. Hence we will never get "locked in" when we use XMPP technologies.

Some of the core XMPP technologies are given below.

Core — Gives information about the core XMPP technologies for XML streaming.

Jingle — Technology for multimedia signaling for voice, video, file transfer, and other applications.

Multi-User Chat — Provides flexible, multi-party communication.

PubSub — Supports alerts and notifications for data syndication, rich presence and more.

BOSH — This is an HTTP binding for XMPP (and other) traffic.

2.6.6 AMQP

Advanced Message Queuing Protocol (AMQP) is an open standard application layer protocol for message-oriented middleware. The defining features of AMQP are message orientation, queuing, routing (including point-to-point and publish-and-subscribe), reliability and security. Advanced Message Queuing Protocol (AMQP) is an open source published standard for asynchronous messaging by wire. AMQP enables encrypted and interoperable messaging between organizations and applications. The protocol is used in client/server messaging and in IoT device management. It is a binary application layer protocol.

AMPQ is efficient, portable, multichannel and secure. The binary protocol offers authentication and encryption by way of SASL or TLS, relying on a transport protocol such as TCP. The messaging protocol is fast and features guaranteed delivery with acknowledgement of received messages. AMQP works well in multi-client environments and provides a means for delegating tasks and making servers handle immediate requests faster. Because AMQP is a streamed binary messaging system with tightly mandated messaging behavior, the interoperability of clients from different vendors is assured. AMQP allows for various guaranteed messaging modes specifying a message be sent.

1. At-most-Each message is delivered once or never. It is sent one time with the possibility of being missed.
2. At-least-once-Each message is certain to be delivered, but may do so multiple times. It guarantees delivery with the possibility of duplicated messages.
3. Exactly-once- Each message will always certainly arrive and do so only once. This guarantees a one-time only delivery.

AMQP Features

In addition to security, reliability, interoperability between protocols and devices, routing, queuing and open standard AMQP has certain important highlights. It includes the following.

(a) Targeted QoS (Selectively offering QoS to links)
(b) Persistence (Message delivery guarantees)
(c) Delivery of messages to multiple consumers.
(d) Possibility of ensuring multiple consumption.
(e) Possibility of preventing multiple consumption.
(f) High speed protocol.

AMQP Architecture

AMQP connects systems, feeds business processes with the information they need and reliably transmits onward the instructions that achieve their goals. AMQP connects across organizations, technologies, time and space. The architecture of AMQP is shown in Figure 2.16.

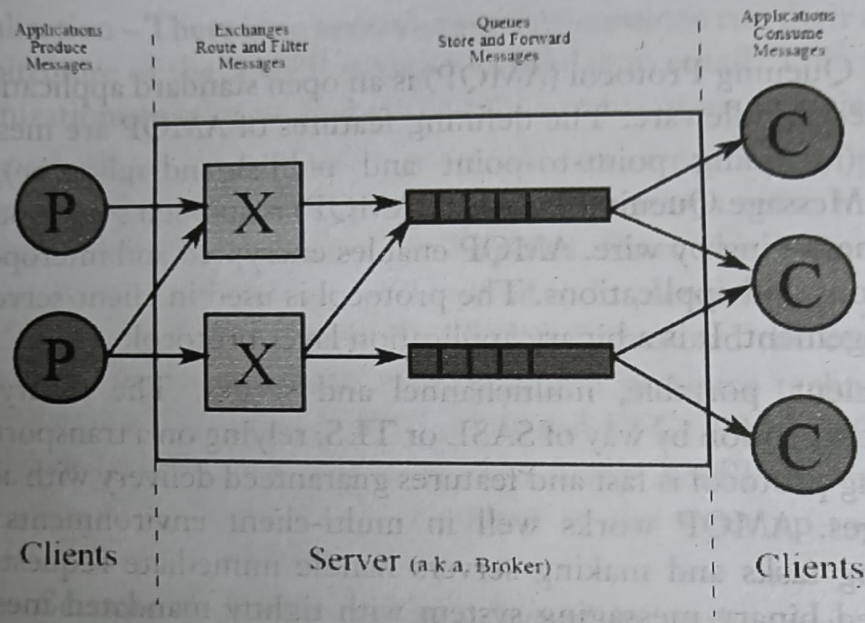

Fig.2.16: Architecture of AMQP

There are applications that produce messages on one end and applications that consume messages on the other end. In Figure 2.16, clients that produce messages are denoted as P and clients that consume messages are denoted as C and X represents exchanges, route and filter messages. The servers in between contains brokers which receives messages and routes them to queues. The queues store and forward messages to business clients. There will be separate queues for separate business processes. The consumer clients receive messages from these queues. In addition, there are bindings which are rules for distributing messages like who can access what message, destination of the message etc.

After receiving messages from the publishing clients, the exchanges process them and route them to one or more queues. The type of routing performed depends on the type of the exchange and there are currently four of them.

Direct Exchange: Direct exchange type involves the delivery of messages to queues based on routing keys. Routing keys can be considered as additional data defined to set where a message will go. Typical use case for direct exchange is load balancing tasks in a round-robin way between workers.

Fan-out Exchange: Fan-out exchange completely ignores the routing key and sends any message to all the queues bound to it. Use cases for fan-out exchanges usually involve distribution of a message to multiple clients for purposes similar to notifications: Sharing of

messages (e.g. chat servers) and updates (e.g. news), Application states (e.g. configurations).

Topic Exchange: Topic exchange is mainly used for pub/sub (publish-subscribe) patterns. Using this type of transferring, a routing key alongside binding of queues to exchanges are used to match and send messages. Whenever a specialized involvement of a consumer is necessary (such as a single working set to perform a certain type of actions), topic exchange comes in handy to distribute messages accordingly based on keys and patterns.

Headers Exchange: Headers exchange constitutes of using additional headers (i.e. message attributes) coupled with messages instead of depending on routing keys for routing to queues. Being able to use types of data other than strings headers exchange allow differing routing mechanism with more possibilities but similar to direct exchange through keys.

In AMQP, basic unit of data is a ***frame***. There are nine AMQP frame types defined that are used to initiate, control and tear down the transfer of messages between two peers. They are the following.

1. Open (the connection)
2. Begin (the session)
3. Attach (initiate new link)
4. Transfer (for sending actual messages)
5. Flow (controls message flow rate)
6. Disposition(informs the changes in state of transfer)
7. Detach (terminate the link)
8. End (the session)
9. Close (the connection)

AMQP Applications

AMQP is ideal for several applications as follows.

1. Monitoring and globally sharing updates.
2. Connecting different systems and processes to talk to each other.
3. Allowing servers to respond to immediate requests quickly and delegate time consuming tasks for later processing.
4. Distributing a message to multiple recipients for consumption.
5. Enabling offline clients to fetch data at a later time.
6. Introducing fully asynchronous functionality for systems.
7. Increasing reliability and uptime of application deployments.

2.7 Conclusion

This Chapter gives an overview about IoT networking. This Chapter explains the connectivity terminologies, gateway prefix allotment, impact of mobility on addressing, multihoming and