RGPVNOTES.IN

Please do not share these notes on apps like WhatsApp or Telegram.

The revenue we generate from the ads we show on our website and app funds our services. The generated revenue **helps us prepare new notes and improve the quality of existing study materials**, which are available on our website and mobile app.

If you don't use our website and app directly, it will hurt our revenue, and we might not be able to run the services and **have to close them.** So, it is a humble request for all to **stop sharing the study material** we provide on various apps. Please **share the website's URL instead.**

Downloaded from www.rgpvnotes.in, whatsapp: 8989595022

UNIT-IV (CO4)

MQTT, MQTT Methods and Components, MQTT Communication, Topics and Applications, SMQTT, CoAP, CoAP Message Types, CoAP Request-Response Model, XMPP, AMQP Features and Components, AMQP Frame Types.

Message Queuing Telemetry Transport (MQTT)

MQTT (Message Queuing Telemetry Transport) is a lightweight messaging protocol that was developed by IBM and first released in 1999. It uses the pub/sub pattern and translates messages between devices, servers, and applications. MQTT protocol was initially created in order to link sensors on oil pipelines with communications satellites, with an emphasis on minimal battery loss and bandwidth consumption.

The connected devices in the MQTT protocol are known as “clients,” which communicate with a server referred to as the “broker.” The broker handles the task of data transmission between clients. Whenever a client (known as the “publisher”) wants to distribute information, it will publish to a particular topic, the broker then sends this information to any clients that have subscribed to that topic (known as “subscribers”).

The publisher does not need any data on the number or the locations of subscribers. In turn, subscribers do not need any data about the publisher. Any client can be a publisher, subscriber, or both. The clients are typically not aware of each other, only of the broker that serves as the intermediary. This setup is popularly known as the “pub/sub model”.

MQTT Methods

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, found on a server.

Methods in MQTT are:

Connect -Waits for connection to be established with the server.

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

Subscribe – Requests the server to let the client subscribe to one or more topics.

Unsubscribe – Requests the server to let the client unsubscribe from one or more topics.

Publish – Returns immediately to application thread after passing request to the MQTT client.

follow us on instagram for frequent updates: www.instagram.com/rgpvnotes.in

Downloaded from www.rgpvnotes.in, whatsapp: 8989595022

MQTT Components:

MQTT main components are as follows:

- **A broker -** broker which is the server that handles the data transmission between the clients.
- **A topic-** which is the place a device wants to put or retrieve a message to/from.
- **The message**-which is the data that a device receives "when subscribing" from a topic or send "when publishing" to a topic.
- **Publish-** publish is the process a device does to send its message to the broker.
- **Subscribe**- where a device does to retrieve a message from the broker.

Like any other internet protocol, MQTT is based on clients and a server. Likewise, the server is the node who is responsible for handling the client's requests of receiving or sending data between each other.

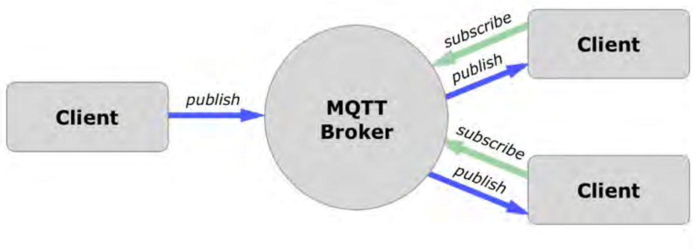

Figure 4.1: MQTT working

MQTT server is called a broker and the clients are simply the connected devices.

- When a device (a client) wants to send data to the broker, we call this operation a "publish".
- When a device (a client) wants to receive data from the broker, we call this operation a "subscribe".

MQTT Communication

MQTT (Message Queuing Telemetry Transport) Communication is a process for exchanging messages among devices. It is frequently used in IoT applications. MQTT is intended for large networks with low data traffic and designed to minimize data volumes. Data transfer using MQTT takes place over TCP. It may be encrypted with SSL. A "publisher-subscriber" data transfer model is used. This means that messages are exchanged using one central hub (a MQTT broker).

MQTT broker

A MQTT broker is a central hub (typically in a cloud in the public internet) that connects MQTT publishers with MQTT subscribers. MQTT publishers send messages and MQTT subscribers subscribe to receive the messages. There can be several MQTT subscribers to the same "topic". Messages are divided into "topics"; a device may either "publish" a given topic, or "subscribe" to the topic. Within a topic, messages are exchanged as they are received by the MQTT broker and then sent to the subscribed devices. A device (electrical socket) can be simultaneously a publisher for some topics (publishes the measured values) and a subscriber for other topics (reacts to commands for controlling the output).

MQTT subscriber

A MQTT subscriber receives MQTT messages from the MQTT broker. Messages are categorized into topics that can be subscribed too. It uses the pub/sub pattern to connect interested parties with each other. It does it by

follow us on instagram for frequent updates: www.instagram.com/rgpvnotes.in

Downloaded from www.rgpvnotes.in, whatsapp: 8989595022

decoupling the sender (publisher) with the receiver (subscriber). The publisher sends a message to a central topic which has multiple subscribers waiting to receive the message.

MQTT publisher

A MQTT publisher sends MQTT messages to the MQTT broker. A MQTT client can publish messages as long as it is connected to a MQTT broker. The MQTT protocol categorizes the messages by the topic. Every message must contain a topic that can be used by the MQTT broker to pass the message on to the subscribed MQTT subscribers. Every message has a payload that is delivered to the subscribers in this way. It can carry any content.

MQTT Topics and Application

Another important concept is the topics. Topics are the way to register interest for incoming messages or how you specify where you want to publish the message.

Topics are represented with strings separated by a forward slash. Each forward slash indicates a topic level. Here's an example on how you would create a topic for a lamp in your home office:

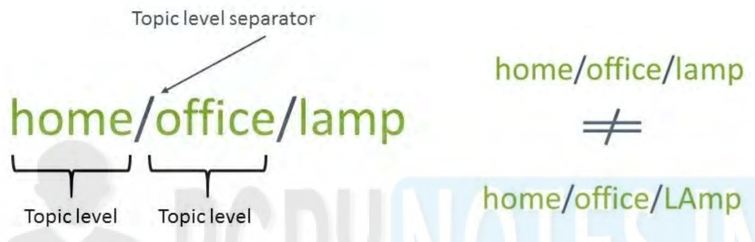

Figure 4.2: Example of Topic level separator

Note: topics are case-sensitive, which makes these two topics different:

Below figure shows the scenario to turn on a lamp in the home or office using MQTT can imagine the following scenario:

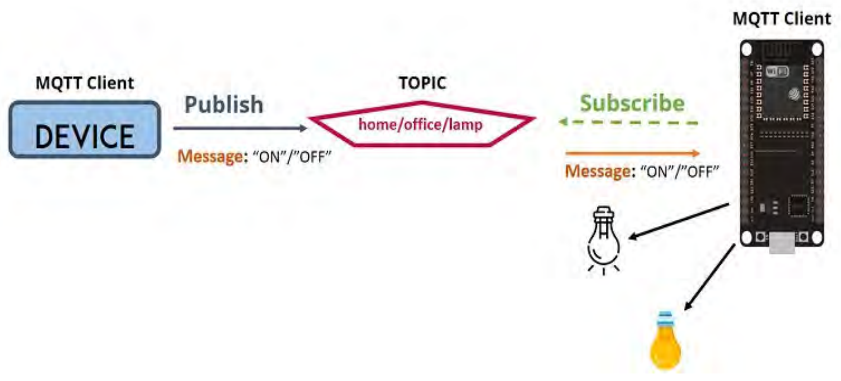

Figure 4.3: Example of MQTT

1. Should have a device that publishes "on" and "off" messages on the home/office/lamp topic.
2. Should have a device that controls a lamp (it can be an ESP32, ESP8266, or any other board). The ESP32 that controls your lamp is subscribed to that topic: home/office/lamp.

follow us on instagram for frequent updates: www.instagram.com/rgpvnotes.in

Downloaded from www.rgpvnotes.in, whatsapp: 8989595022

3. So, when a new message is published on that topic, the ESP32 receives the “on” or “off” message and turns the lamp on or off.

MQTT was thus created with these features to greatly fit-in for constrained IoT networks. There was one more great advantage of MQTT, MQTT was able to Separate data creators & data consumers. Data creators are sensor nodes and data consumers could be cloud based applications or could even be other connected devices. By incorporating a publish-subscribe model which has a server also called broker in between, MQTT enables massive scalability of devices, which is much essential for IoT.

- Simple to Implement,
- Quality of Service for Data Delivery
- Light Weight
- Bandwidth Efficient
- Data Agnostic – So that we could send different types of data such as values, images, etc.
- Continuous Session Awareness- To Know the current status in Real-time

SMQTT (Secure Message Queue Telemetry Transport)

SMQTT (Secure Message Queue Telemetry Transport) is an extension of MQTT protocol which uses encryption based on lightweight attribute encryption. The main advantage of this encryption is that it has a broadcast encryption feature. In this features, one message is encrypted and delivered to multiple other nodes. The process of message transfer and receiving consists of four major stages:

1. Setup: In this phase, the publishers and subscribers register themselves to the broker and get a secret master key.
2. Encryption: When the data is published to broker, it is encrypted by broker.
3. Publish: The broker publishes the encrypted message to the subscribers.
4. Decryption: Finally, the received message is decrypted by subscribers with the same master key. SMQTT is propose used to enhance MQTT security feature.

CoAP (Constrained Application Protocol)

CoAP (Constrained Application Protocol**)** is a session layer protocol that provides the RESTful (HTTP) interface between HTTP client and server. It is designed by IETF Constrained RESTful Environment (CoRE) working group. It is designed to use devices on the same constrained network between devices and general nodes on the Internet. CoAP enables low-power sensors to use RESTful services while meeting their low power constraints. This protocol is specially built for IoT systems primarily based on HTTP protocols.

This network is used within the limited network or in a constrained environment. The whole architecture of CoAP consists of CoAP client, CoAP server, REST CoAP proxy, and REST internet.

follow us on instagram for frequent updates: www.instagram.com/rgpvnotes.in

Downloaded from www.rgpvnotes.in, whatsapp: 8989595022

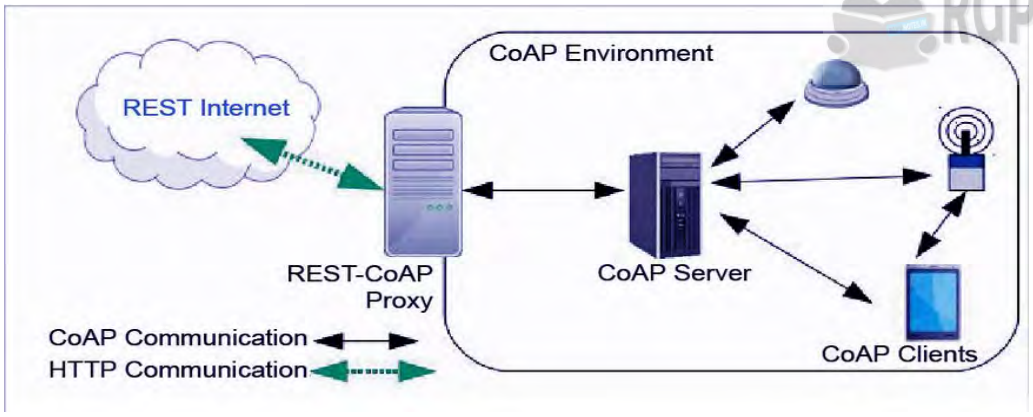

Figure 4.4: CoAP Working

The data is sent from CoAP clients (such as smart phones, RFID sensors, etc.) to the CoAP server and the same message is routed to REST CoAP proxy. The REST CoAP proxy interacts outside the CoAP environment and uploads the data over REST internet.

CoAP Message Types

CoAP makes use of two message types, requests and responses, using a simple, binary, base header format. The base header may be followed by options in an optimized Type-Length-Value format. CoAP is by default bound to UDP and optionally to DTLS, providing a high level of communications security. Any bytes after the headers in the packet are considered the message body. The length of the message body is implied by the datagram length. The entire message must fit within a single datagram when bound to UDP. When used with 6LoWPAN, as defined in RFC 4944, messages SHOULD also fit into a single IEEE 802.15.4 frame to minimize fragmentation.

One must take security into account when dealing with IoT protocols. For example, CoAP uses UDP to transport information. CoAP relies on UDP security features to protect information. As HTTP uses TLS over TCP, CoAP uses Datagram TLS over UDP. DTLS supports RSA, AES, and so on.

CoAP Request/Response Model

The CoAP Request/Response is the second layer in the CoAP abstraction layer. The request is sent using a Confirmable (CON) or Non-Confirmable (NON) message. There are several scenarios depending on if the server can answer immediately to the client request or the answer if not available.

If the server can answer immediately to the client request, then if the request is carried using a Confirmable message (CON), the server sends back to the client an Acknowledge message containing the response or the error code:

follow us on instagram for frequent updates: www.instagram.com/rgpvnotes.in

Downloaded from www.rgpvnotes.in, whatsapp: 8989595022

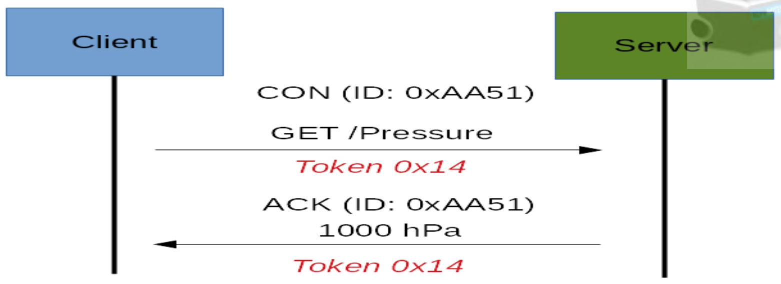

Figure 4.5: CoAP Request/Response Model

As it can be noticed in the CoAP message, there is a Token. The Token is different from the Message-ID and it is used to match the request and the response.

If the server can't answer to the request coming from the client immediately, then it sends an Acknowledge message with an empty response. As soon as the response is available, then the server sends a new confirmable message to the client containing the response. At this point, the client sends back an Acknowledge message:

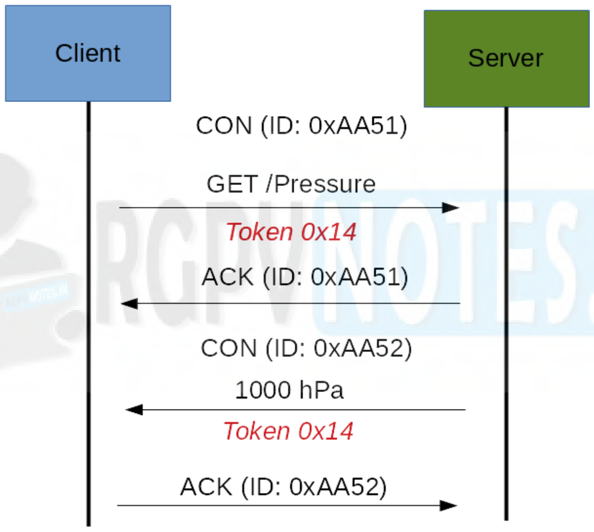

Figure 4.6: CoAP Request/Response Model

If the request coming from the client is carried using a NON-confirmable message, then the server answer using a NON-confirmable message.

XMPP Protocol

XMPP is a short form for Extensible Messaging Presence Protocol. It's protocol for streaming XML elements over a network in order to exchange messages and presence information in close to real time. This protocol is mostly used by instant messaging applications like WhatsApp.

Meaning each character of word XMPP:

- X: It means extensible. XMPP is an open-source project which can be changed or extended according to the need.
- M: XMPP is designed for sending messages in real time. It has very efficient push mechanism compared to other protocols.
- P: It determines whether you are online/offline/busy. It indicates the state.

follow us on instagram for frequent updates: www.instagram.com/rgpvnotes.in

Downloaded from www.rgpvnotes.in, whatsapp: 8989595022

- P: XMPP is a protocol, that is, a set of standards that allow systems to communicate with each other.

These are the basic requirements of any Instant Messenger which are fulfilled by XMPP:

1. Send and receive messages with other users.
2. Check and share presence status
3. Manage subscriptions to and from other users.
4. Manage contact list
5. Block communications (receive message, sharing presence status, etc) to specific users.

Some other XMPP features:

1. Decentralized

XMPP is based on client-server architecture, i.e., clients don’t communicate directly, and they do it with the help of server as intermediary. It is decentralized means there is no centralized XMPP server just like email, anyone can run their own XMPP server. Resource is used in case the application support mobile as well as desktop or web application, so it can be optional in case a Instant Messenger Application support only single kind of resource.

2. XMPP implementation

The original protocol for XMPP is Transmission Control Protocol, using open ended XML streams over long-lived TCP connections. In some cases, there are restricted firewalls, XMPP (port 5222) is blocked, so it can’t be used for web applications and users behind restricted firewalls, to overcome this, XMPP community also developed a HTTP transport. And as the client uses HTTP, most firewalls allow clients to fetch and post messages without any problem. Thus, in scenarios where the TCP port used by XMPP is blocked, a server can listen on the normal HTTP port and the traffic should pass without problems.

AMQP Features

The Advanced Message Queuing Protocol (AMQP) is an open internet protocol for messaging. This protocol allows the reliable message exchange between two parties (client and server). AMQP was development in order to give possibility different applications and system working together regardless of their internal structure.

Uses of AMQP

AMQP can be used in any situation if there is a need for high-quality and secure message delivery between client and broker.

AMQP provides the following possibilities:

- Monitoring and sharing updates;

- Ensuring quick response of the server to requests and transmission of time-consuming tasks for further processing;

- Distribute messages to multiple recipients;

- Connection offline clients for further data retrieval;

follow us on instagram for frequent updates: www.instagram.com/rgpvnotes.in

Downloaded from www.rgpvnotes.in, whatsapp: 8989595022

- Increase the reliability and smooth operation of applications.;
- Reliability of message delivery;
- High speed message delivery;
- Message Acceptance.

Components of AMQP

Components of AMQP are as follows :

1. Message queue

A queue acts as a buffer that stores messages that are consumed later. A queue can also be declared with a number of attributes during creation. For instance, it can be marked as durable, auto-delete and exclusive, where exclusive means that it can be used by only one connection and this queue will be deleted when that connection closes.

2. Exchanges and Exchange Types

A channel routes messages to a queue depending on the exchange type and bindings between the exchange and the queue. For a queue to receive messages, it must be bound to at least one exchange.

AMQP 0.9.1 brokers should provide four exchange types - direct exchange, fanout exchange, topic exchange, and header exchange. A deeper understanding of the different exchange types, bindings, and routing keys and how or when you should use them can be found in RabbitMQ for beginners - Exchanges, routing keys and bindings.

An exchange can be declared with a number of attributes during creation. For instance, it can be marked as durable so that it survives a broker restart, or it can be marked as auto-delete meaning that it's automatically deleted when the last queue is unbound.

3. Binding

A binding is a relation between a queue and an exchange consisting of a set of rules that the exchange uses (among other things) to route messages to queues.

4. Message and Content

A message is an entity sent from the publisher to the queue and finally subscribed to by the consumer. Each message contains a set of headers defining properties such as life duration, durability, and priority.

AMQP 0.9.1 also has a built-in feature called message acknowledgment that is used to confirm message delivery and/or processing.

5. Connection

A connection in AMQP 0.9.1 is a network connection between your application and the AMQP broker, e.g. a TCP/IP socket connection.

follow us on instagram for frequent updates: www.instagram.com/rgpvnotes.in

Downloaded from www.rgpvnotes.in, whatsapp: 8989595022

6. Channel

A channel is a virtual connection inside a connection, between two AMQP peers. Message publishing or consuming to or from a queue is performed over a channel (AMQP). A channel is multiplexed; one single connection can have multiple channels.

More information about connections and channels and the relationship between them can be found here.

- **Virtual Hosts**

Virtual hosts (vhost) provide a way to segregate applications in the broker. Different users can have different access privileges to different vhost. Queues and exchanges are created so they only exist in one vhost.

- **AMQP Methods**

AMQP 0.9.1 provides a number of methods or operations that can be performed. Some examples of AMQP methods are opening a channel, declaring a queue or deleting an exchange.

AMQP Frame types

A frame is the basic unit with AMQP. A connection consists of the ordered sequence of frames. Order in this case means that the last frame must not arrive at the receiver until all other frames have first reached their destination. Each frame can be divided into three segments (in version 1.0):

- **Frame header**: This mandatory header has a size of 8 bytes. Here you will find information that determines the routing of the message.
- **Extended header**: This area is optional and has no set scope. It serves to expand the header in the future with further information.
- **Frame body**: The body contains the actual data to be transferred. The size is freely selectable. However, this area can also be left empty, and then the frame only serves to maintain the connection.

The body of a frame, in turn, can take nine different forms:

- **open**: Negotiates the connection parameters between broker and client.
- **begin**: Indicates that a connection is starting.
- **attach**: The message is appended with a link that is necessary in order to use the data transfer.
- **flow**: Changes the status of a link.
- **transfer**: The actual message is transmitted with the transfer frame.
- **disposition**: A disposition frame provides information on changes to the information delivery.
- **detach**: Removes the link.
- **end**: Indicates that the connection will be terminated.
- **close**: Terminates the connection and declares that no further frames will be sent.

Queues & messages

Each queue has its own name, which identifies it to the other participants. Either a client or the broker determines the name. A queue is realized by a memory, which either can permanently sit on a hard disk or briefly in RAM. The permanent variant guarantees that the queue will continue to exist even after restarting a broker. But it does not guarantee that messages are also permanently secured here, it depends on the message as to whether they are still available after a restart. It happens when a consumer cannot recall the message in queue, because, for example, the client or the connection has broken down. It is possible to decide whether a

follow us on instagram for frequent updates: www.instagram.com/rgpvnotes.in

Downloaded from www.rgpvnotes.in, whatsapp: 8989595022

consumer must properly acknowledge the receipt of a message or whether the mere delivery is sufficient for success. If the first variant is chosen and the consumer does not need to send back a message, the broker tries to send the message to another consumer or to reach the actual recipient again. However, if the variant is activated without confirmation and the consumer does not retrieve the message, it will be lost. But it is also possible for a client to deliberately reject the acceptance of a message. This may be useful if message processing does not work. The feedback from the consumer causes the broker to either completely delete the message or incorporate it back into the queue.

follow us on instagram for frequent updates: www.instagram.com/rgpvnotes.in

RGPV NOTES.IN

RGPVNOTES.IN

Thank you for using our services. Please support us so that we can improve further and help more people.
https://www.rgpvnotes.in/support-us

If you have questions or doubts, contact us on
WhatsApp at +91-8989595022 or by email at hey@rgpvnotes.in.

For frequent updates, you can follow us on
Instagram: https://www.instagram.com/rgpvnotes.in/.