CHANDIGARH UNIVERSITY

Discover. Learn. Empower.

University Institute of Engineering

AIT-CSE

Privacy and Security in IoT - CSD- 433

Topic – Security Aspects of API ,Transport Encryption,MQTT, CoAP

Lecture – 1.8

Delivered by

Er. Gaurav Soni (E9610)

Assistant Professor, AIT-CSE

DISCOVER . LEARN . EMPOWER

Privacy and Security in IoT

Course Objectives

CO Number	Title
CO1	To identify various privacy and security requirements in Internet of Things
CO2	To learn cryptographic techniques for a secure IoT system
CO3	To understand various Trust Models used in IoT

Privacy and Security in IoT

Course Outcome

CO Number	Title	Level
CO1	After successful completion of this course students will be able to understand the security requirements in IoT.	**Understand**
CO2	After successful completion of this course students will be able to understand the authentication credentials and access control.	**Understand**
CO3	After successful completion of this course students will be able to implement security algorithms to make a secure IoT system.	Implement

This will be covered in this lecture

What are some of the most common API security best practices?

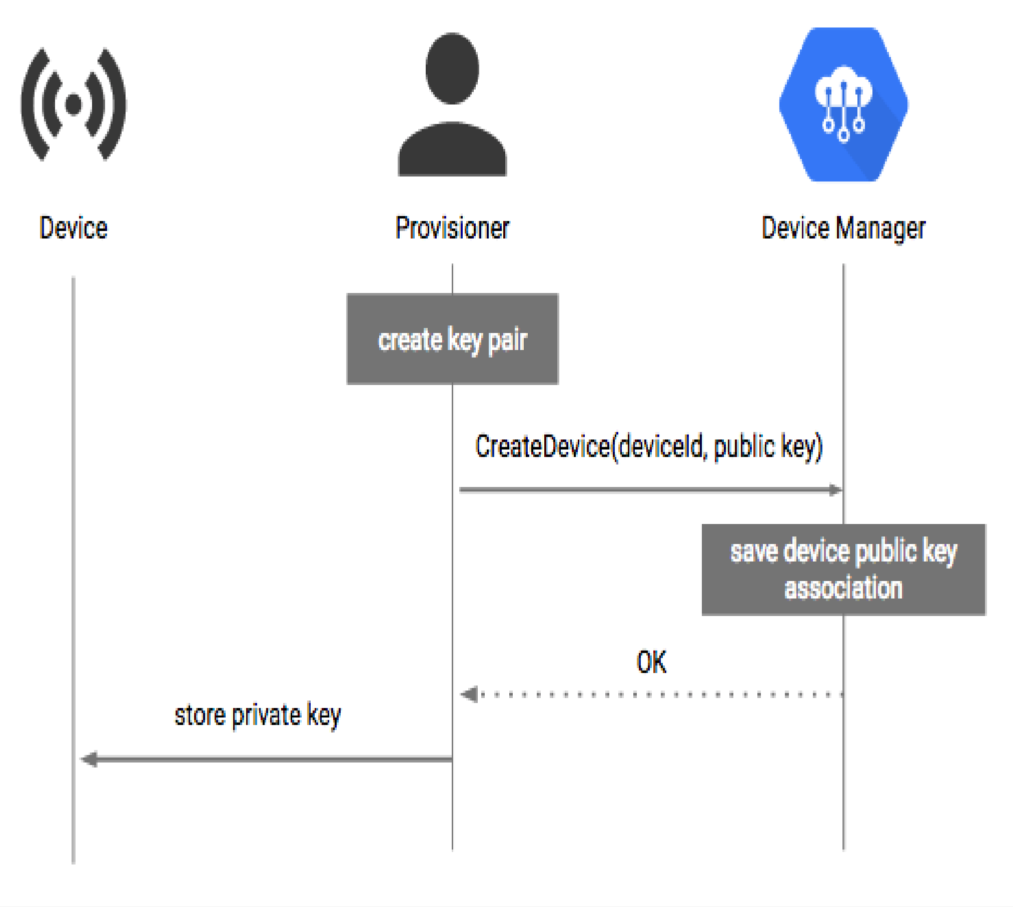
- **Use tokens**. Establish trusted identities and then control access to services and resources by using tokens assigned to those identities.
- **Use encryption and signatures**. Encrypt your data using a method like TLS(see above). Require signatures to ensure that the right users are decrypting and modifying your data, and no one else.
- **Identify vulnerabilities**. Keep up with your operating system, network, drivers, and API components. Know how everything works together and identify weak spots that could be used to break into your APIs. Use sniffers to detect security issues and track data leaks.
- **Use quotas and throttling**. Place quotas on how often your API can be called and track its use over history. More calls on an API may indicate that it is being abused. It could also be a programming mistake such as calling the API in an endless loop. Make rules for throttling to protect your APIs from spikes and Denial-of-Service attacks.
- **Use an API gateway**. API gateways act as the major point of enforcement for API traffic. A good gateway will allow you to authenticate traffic as well as control and analyze how your APIs are used.

API management and security

- **API management and security**
- API security often comes down to good API management. Many API management platforms support three types of security schemes. These are:
- **An API key** that is a single token string (i.e. a small hardware device that provides unique authentication information).
- **Basic Authentication** (APP ID / APP Key) that is a two token string solution (i.e. username and password).
- **OpenID Connect** (OIDC) that is a simple identity layer on top of the popular OAuth framework (i.e. it verifies the user by obtaining basic profile information and using an authentication server).

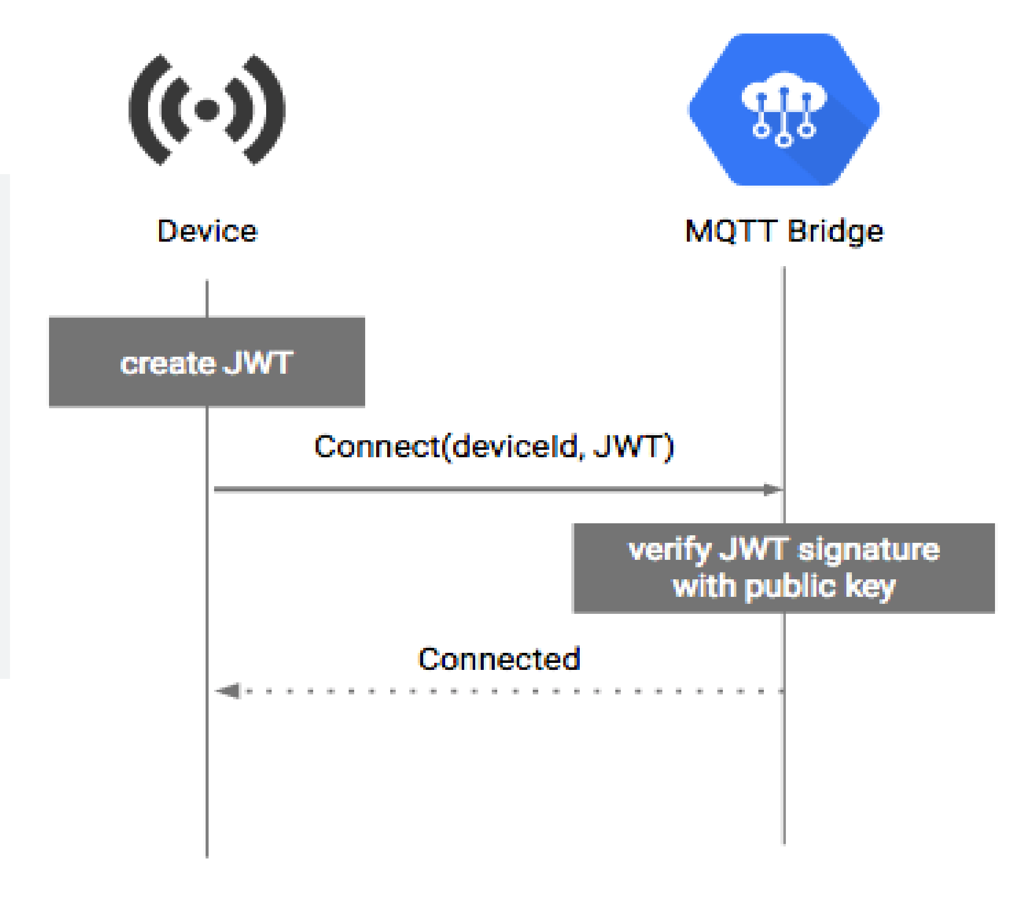
Provisioning credential Authentication in Cloud IoT Core using MQTT

- The public-private key pair is generated by the provisioner.
- The provisioner creates the device using the Cloud IoT Core API, gcloud commands, or the Cloud Platform Console, specifying the public key just created. This will be used to verify the device's identity.
- The Cloud IoT Core device manager stores the device resource and the public key.
- The device manager responds to the provisioner, indicating that the device was created.
- The private key is stored on the device to use later for authentication. The hardware Trusted Platform Module (TPM) can be used for this step


Device
Provisioner
Device Manager
create key pair
CreateDevice(deviceId, public key)
save device public key association
OK
store private key

Authentication in Cloud IoT Core using MQTT

1.The device prepares a JSON Web Token (JWT), as described in [Using JSON Web Tokens](). The JWT is signed with the private key from the authentication flow.
2.When connecting to the MQTT bridge, the device presents the JWT as the password in the MQTT CONNECT message. The username content is ignored; however, some MQTT client libraries will not send the password unless the username is specified. For best results, set the username to an arbitrary value like unused or ignored.
3.The MQTT bridge verifies the JWT against the device's. public key
4.The MQTT bridge accepts the connection.
5.The connection is closed when the JWT expires (after accounting for the allowed clock drift).



TRANSPORT ENCRYPTION

Transport Trust in IoT

- In IoT, a number of lightweight protocols have been developed to match the needs of security, transmission, and resource consumption. The Message Queuing Telemetry Transport (MQTT) and Constrained Application Protocol (CoAP) are the two most promising resource limited devices in IoT.

Both MQTT and CoAP have following features:

- Are open standard
- Easy to implement
- Provide bandwidth-efficient and uses energy-efficient communication

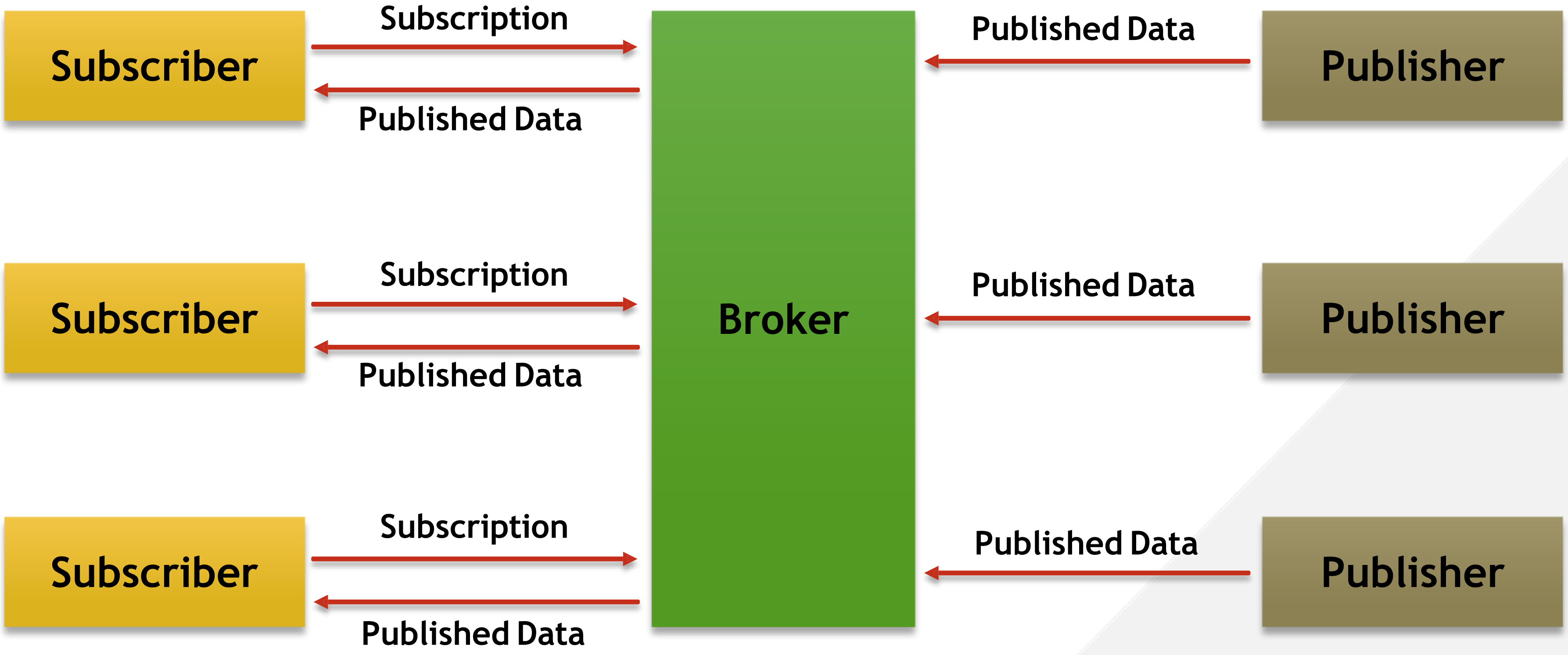
MQTT (Message Queuing Telemetry Transport)

- MQTT is a M2M (Machine-To-Machine) connectivity protocol.
- MQTT is based on ***Publisher/Subscriber*** model.
- It was created to connect to the systems situated in remote place and to get
    - data from those sensor.
- Decoupling can be done because of ***Publisher/Subscriber*** model which is hard to achieve in ***Client/Server*** model.

MQTT Components

- MQTT has 3 basic components
  - Publisher (e.g. Motor, Mobile Devices, etc.)
  - Subscriber (e.g. Temperature sensor, Motion Sensor, etc.)
  - Broker

MQTT Diagram



MQTT Publisher

- System or sensors which collects data and send it to the broker which further
  - sends it to multiple subscribers.
- Example motion sensor, water level sensor, etc.
- Publisher publish data in following formats
  - Binary
  - JSON(JavaScript Object Notation)
  - SDC Record
  - Text

MQTT Subscriber

- subscriber can be a mobile device, data server, monitoring stations, etc. which receives publish data from broker so that it can act according to it or monitor it or store it.
- Subscriber send request to broker to send required publisher's data.
- Broker has the table in which it maintains all subscriptions request, and send publish data according to it.

•Example Mobile devices, Monitoring system, etc.

MQTT Broker

Broker is the component which take care of receiving data from Publisher and sending it to Subscriber accordingly.

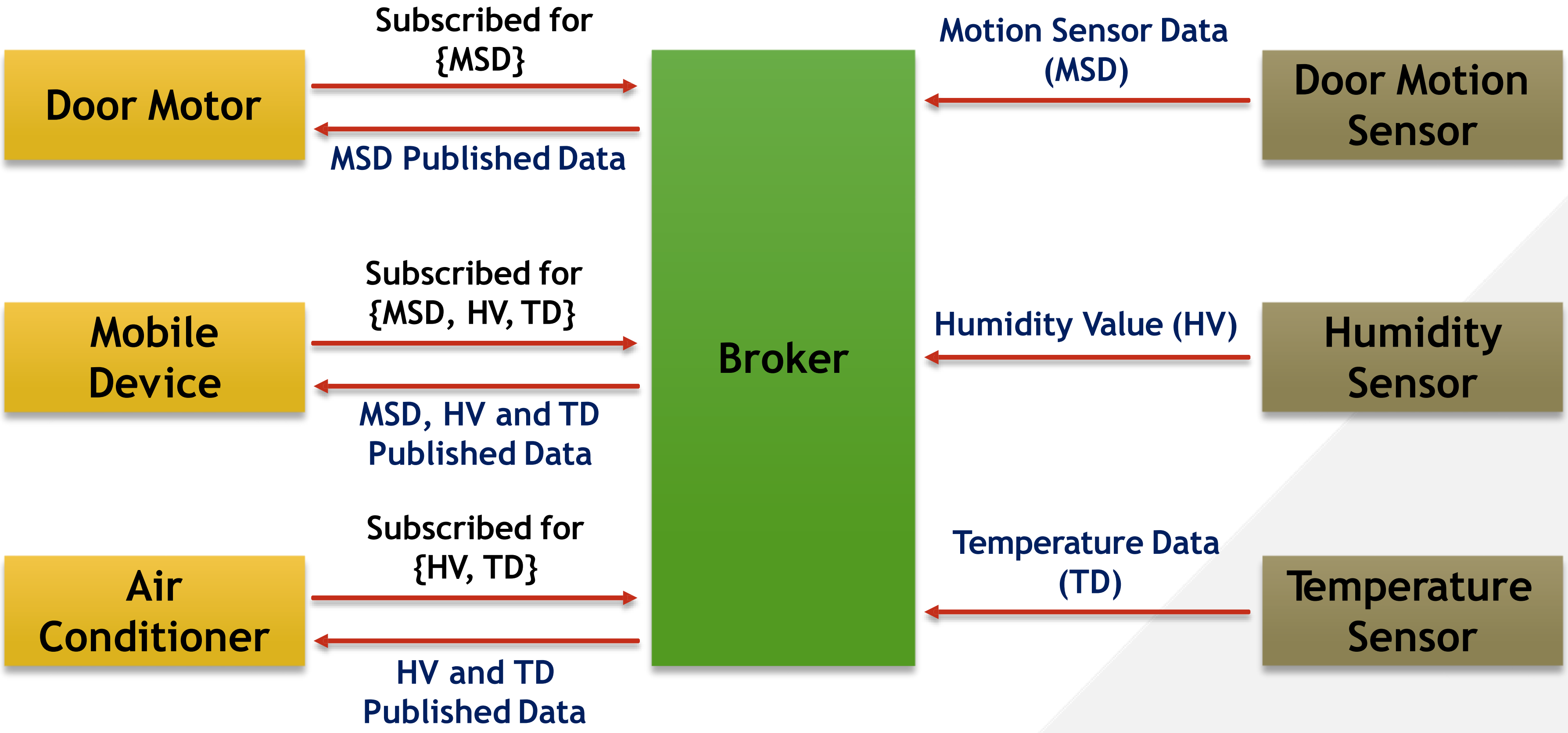
Broker has to filter messages, broker can filter messages in following ways.

- Subject based
- Content based
- Type based

Broker has a subscription table in which it store all the request from the subscriber for the publisher’s data.

Broker sends publish data to multiple subscriber according to this list.

Simple MQTT Example



Door Motor

Subscribed for {MSD}

MSD Published Data

Mobile Device

Subscribed for {MSD, HV, TD}

MSD, HV and TD Published Data

Air Conditioner

Subscribed for {HV, TD}

HV and TD Published Data

Broker

Motion Sensor Data (MSD)

Door Motion Sensor

Humidity Value (HV)

Humidity Sensor

Temperature Data (TD)

Temperature Sensor

Decoupling in Pub/Sub

Space decoupling

- In MQTT there no need for publisher and subscriber to know each other, since it is Pub/Sub model there no need to exchange IP address.

Time decoupling

- Since it is Pub/Sub model there is no need for publisher and subscriber to run on same time.

Synchronization decoupling

- Operations on both publisher and subscriber do not need to be interrupt during the publishing and receiving.

CoAP (Constrained Application Protocol)

- CoAP is developed BY ***IETF*** (Internet Engineering Task Force)
- As the name suggest CoAP has been developed for data transmission in
  - ***constrained network***.
- In network where devices are low power, and network is lossy, we need protocols like UDP for transmission, but UDP has its own disadvantages, here comes CoAP in scene which uses UDP but it is more reliable, secured, etc.
- CoAP has become more important for IoT and M2M because of its features.
- CoAP runs on UDP enable devices, it support ***less memory devices*** and also
  - ***low power devices***.

CoAP (Constrained Application Protocol)

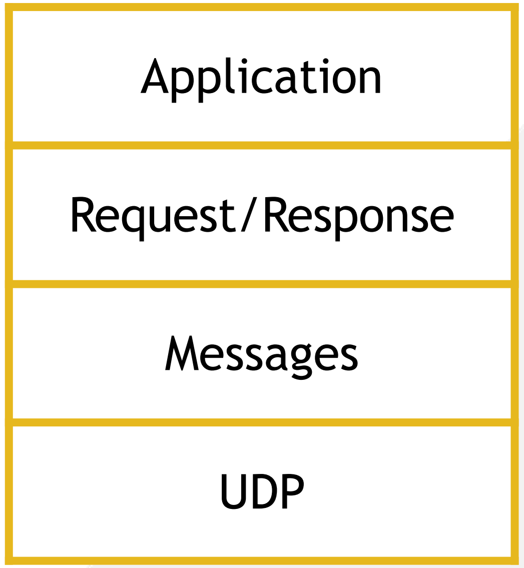
- Unlike MQTT it is ***1 to 1*** protocol.
- CoAP is based on ***Client/Server model***.
- It supports in ***RESTful*** web services.
- It uses less resources then HTTP.
- HTTP runs on TCP and it not suitable for devices with constraints, hence CoAP runs on UDP, although it is connection less protocol.

CoAP (Constrained Application Protocol)

- In CoAP client can use ***<u>GET, PUT, DELETE</u>*** methods during request.
- CoAP provides support to optimize the datagram length.
- It also support ***<u>IP multicast</u>***
- Retransmission is supported by TCP and not by UDP, but CoAP supports •***<u>retransmission</u>***.
- CoAP also supports ***<u>Piggy-backed</u>*** package.
- When server sends response with Acknowledgement it is known as Piggy- backed.

CoAP Layers

- Mostly it is divided in two layers.
  - Upper Layer (Request and Response)
  - Lower Layer (Message)
- There are 4 types of message
  - Confirmable (CON)
  - Non Confirmable (NON)
  - Acknowledgement (ACK)
  - Reset (RST)



Application

Request/Response

Messages

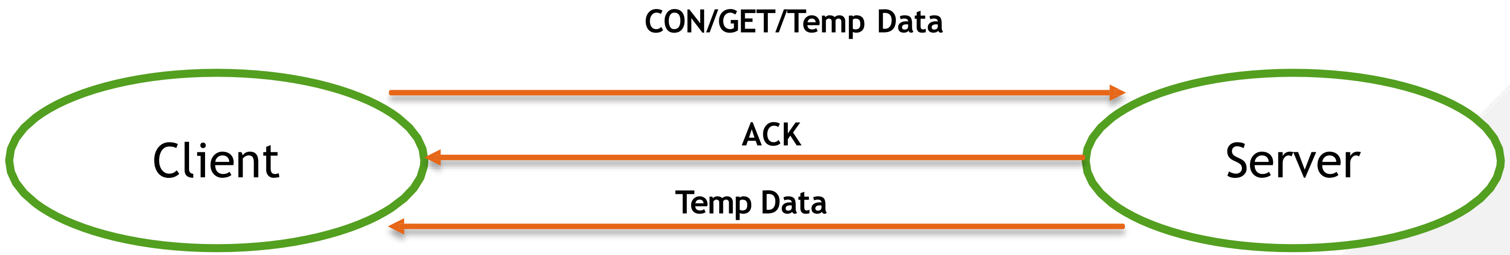
UDP

- Reset (RST) – RST message is sent when recipient fails to send response in time.

Types of message

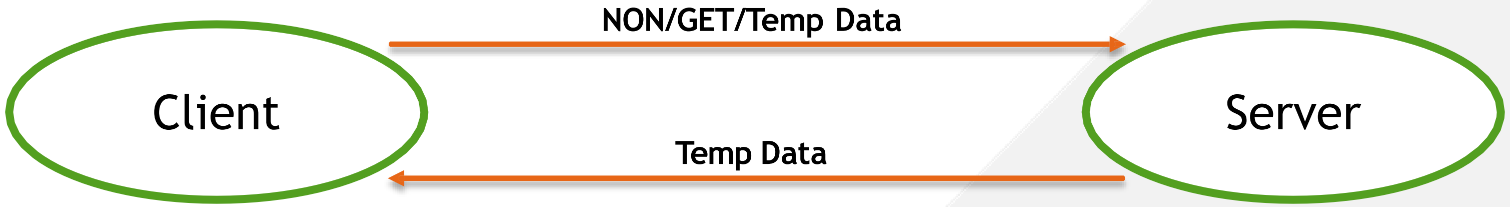
Confirmable (CON)

- If sender wants receiver to send ***ACK*** for its message sender will add ***CON*** in header.



Non Confirmable (NON)

- If sender don’t want receiver to send ***ACK*** for its message sender will add ***NON*** in header.



References

1. Li Da Xu, Securing Internet of Things, Algorithms, and Implementations, Elsevier
2. https://www.hivemq.com/blog/how-to-get-started-with-mqtt/

Home Assignment

1. Explore the concept of MQTT

THANK YOU

For queries
Email: gaurav.e9610@cumail.in