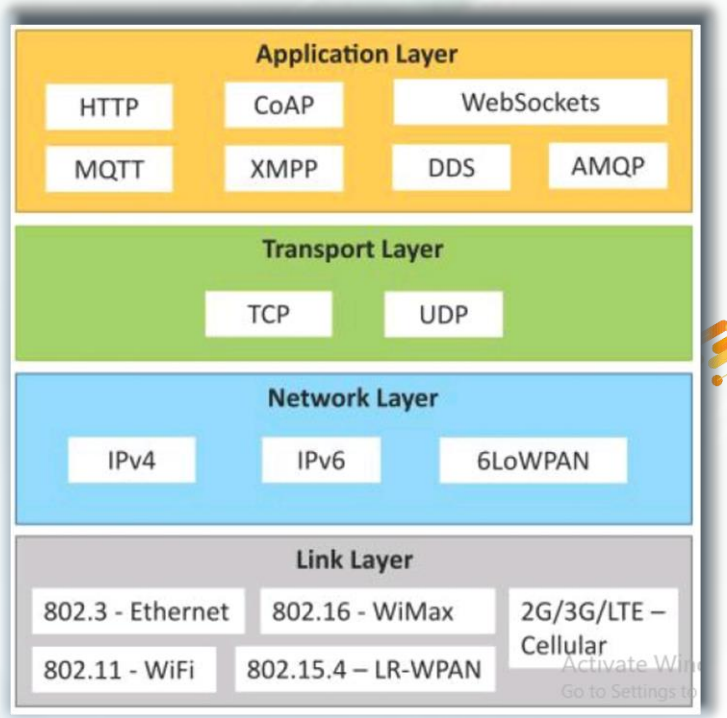
National Forensic Sciences University
Knowledge | Wisdom | Fulfilment
An Institution of National Importance
(Ministry of Home Affairs, Government of India)

Unit 3:
Protocols for IoT



Dr. Ujjaval Patel

Assistant Professor, School of Cyber Security & Digital Forensics, NFSU, Gandhinagar

ujjaval.patel@nfsu.ac.in

+91 987 987 9746

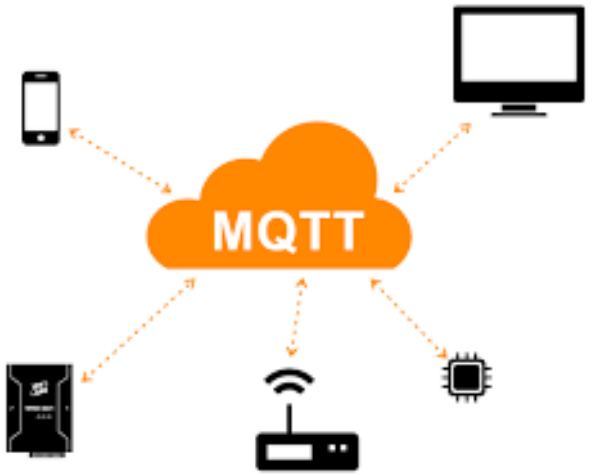
Messaging Protocols

Section - 1

Messaging Protocols in IoT

- Messaging protocols are very important for the transfer of data in terms of messages.
- They are useful for send/receive of a message to/from the cloud in IoT applications.
- In the section, two messaging protocols are discussed.

1. Message Queuing Telemetry Transport (MQTT)
2. Constrained Application Protocol (CoAP)



CoAP
Constrained **A**pplication
Protocol

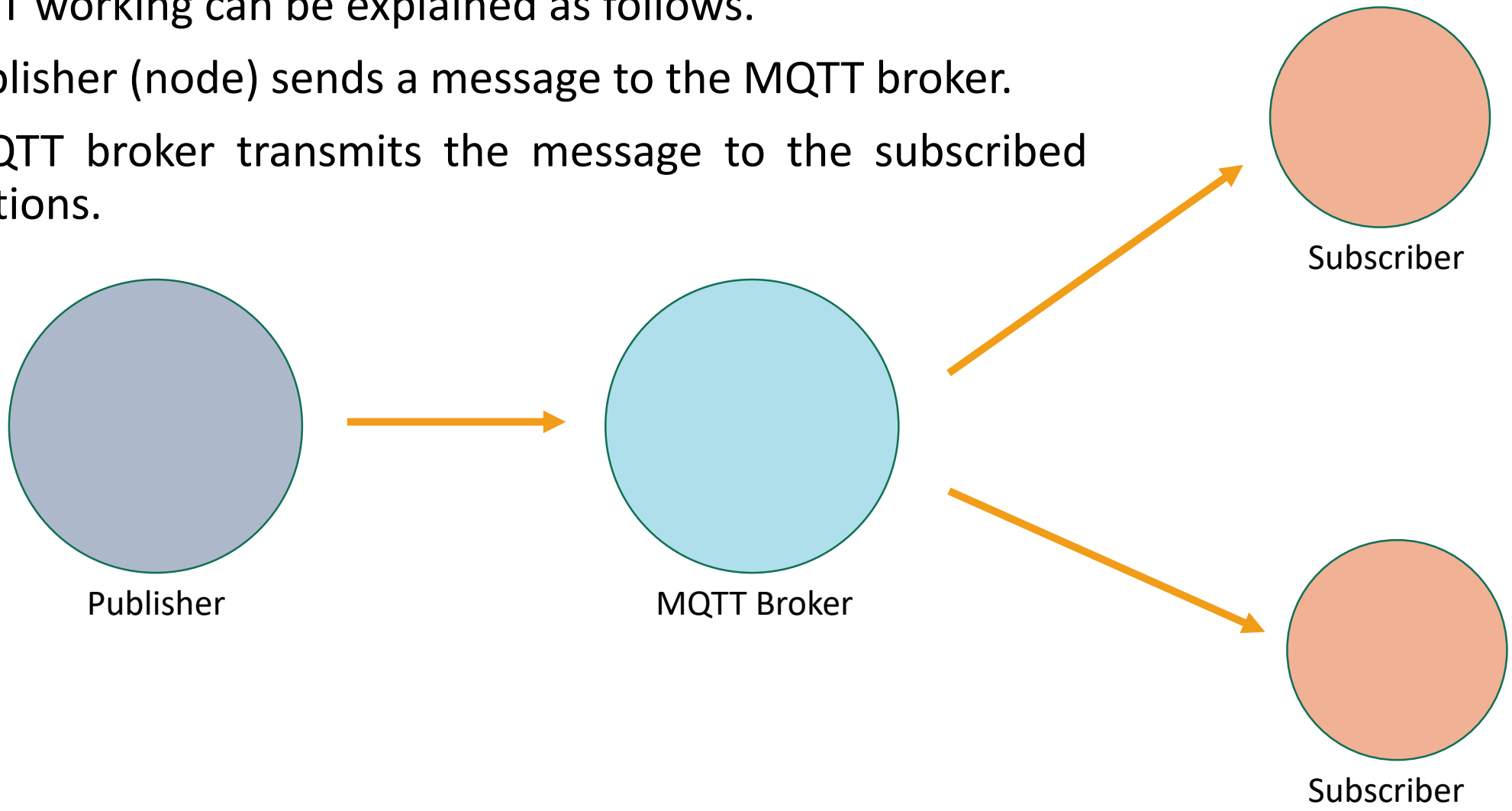
Message Queuing Telemetry Transport (MQTT)

- As we know, IoT has one of the biggest challenges of resource constraints, the lightweight protocol is more preferred.
- MQTT is widely used in IoT applications as it is a lightweight protocol.
- Here, lightweight means, it can work with minimal resources and does not require any specific hardware architecture or additional resources.

MQTT

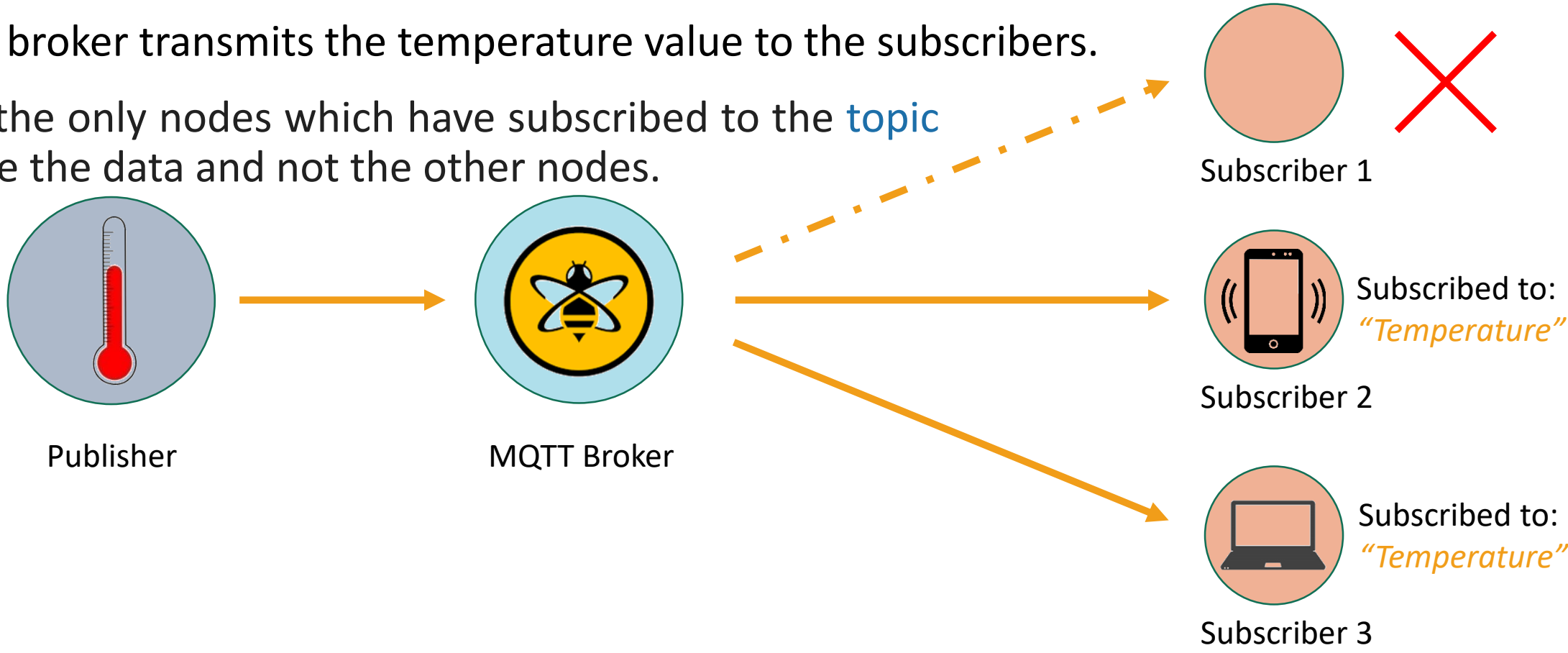
- MQTT works with “publish – subscribe” pattern.
- The MQTT working can be explained as follows.

1. The publisher (node) sends a message to the MQTT broker.
2. The MQTT broker transmits the message to the subscribed destinations.



MQTT - Example

- Suppose a temperature sensor is connected to any system and we want to monitor that temperature.
- The controller connected to the temperature sensor publishes the temperature value to the MQTT broker with a topic name: *"Temperature"*. Hence, it is considered a publisher.
- The MQTT broker transmits the temperature value to the subscribers.
- Note that the only nodes which have subscribed to the topic will capture the data and not the other nodes.



MQTT - Example

- Can publisher and subscriber interchange their role?
- Let us take an example of a Driverless car.
- A GPS sensor is connected to the car which gives the current location of the car.
- The system in car, publishes the location to MQTT broker with topic name – CurrentLocation.
- The server/computer subscribes to the topic CurrentLocation and receives the current location of the car.

Publisher: CurrentLocation | MQTT Broker | Subscriber: CurrentLocation

MQTT - Example

- Based on the currently selected destination, the server/computer computes the direction of the car.
- This data is published to an MQTT broker with a topic named – Direction.
- To get the command from the server, the system in the car has to subscribe to the topic named Direction.
- According to the command received from the server, the car will run in a particular direction.

Publisher: CurrentLocation

Subscriber: Direction

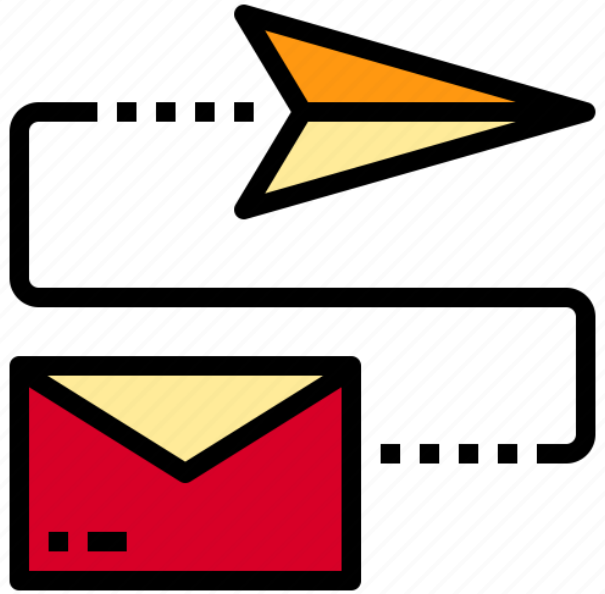
MQTT Broker

Subscriber: CurrentLocation

Publisher: Direction

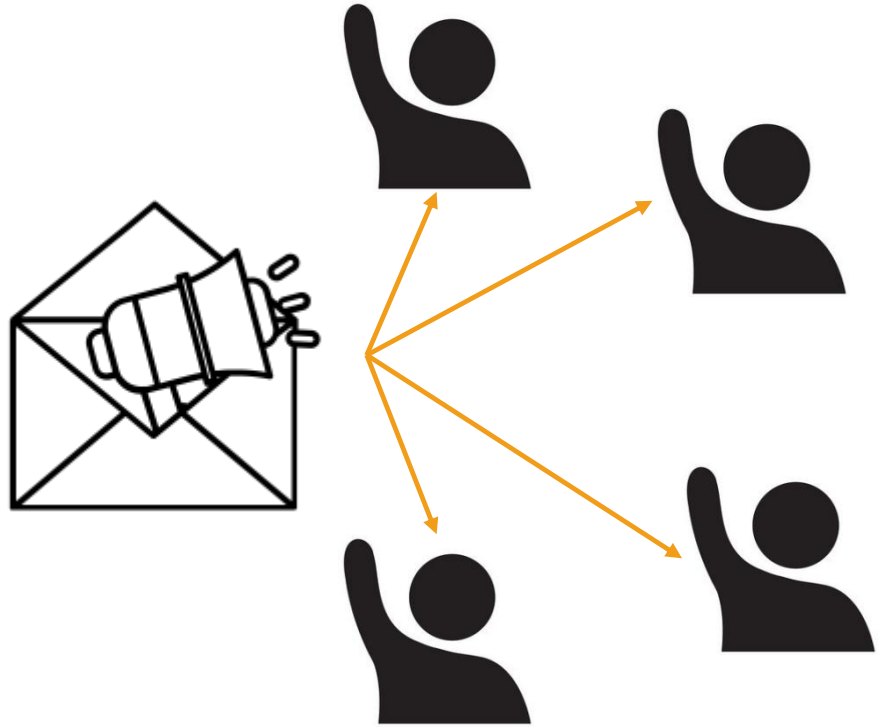
MQTT - Working

- In MQTT, clients do not have any address like a typical networking scheme.
- The broker filters the messages based on the subscribed topics.
- The messages will then be circulated to respective subscribers.
- The topic name can be anything in string format.
- The MQTT working can be explained with an example of the television broadcast.



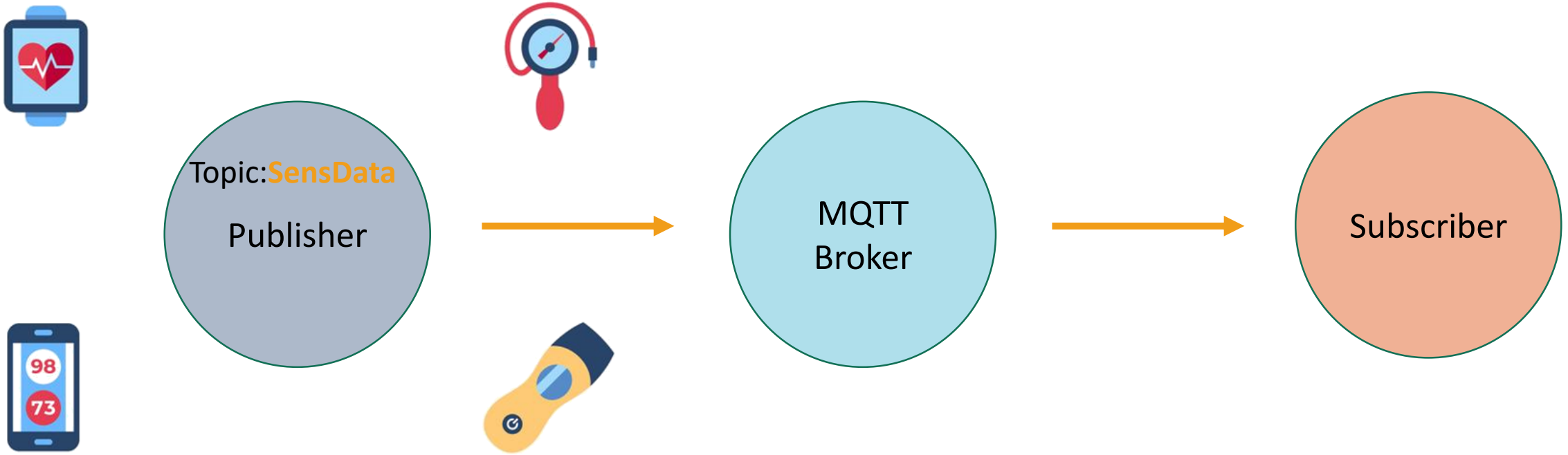
MQTT - Working

- Television broadcaster station broadcasts all the channels.
- The viewers subscribe to their favorite channels only.
- Similarly, in MQTT, the publisher node publishes data with the topic name and interested subscribers subscribe to the topic.
- Note that there can be multiple subscribers of a same topic as well as a single subscriber can subscribe to multiple topics.



MQTT – Node(s)

- Node(s) in MQTT collects the information from sensors or other i/p devices in case of the publisher.
- Connects it to the messaging server known as the MQTT broker.
- A specific string – Topic is used to publish the message and let other nodes understand the information. These nodes are considered subscribers.
- Any node can be publisher or subscriber and node is also referred to as client.



QoS in MQTT

- MQTT supports different QoS (Quality of Service) levels.
- There are 3 layers of QoS supported by MQTT.

1. Level 0 = At most once (Best effort, No Acknowledgement)
2. Level 1 = At least once (Acknowledged, retransmitted if *ack* not received)
3. Level 2 = Exactly once (Requested to send, clear to send)

MQTT Publisher – Code Implementation

- The code for MQTT publisher is as shown in below.

MQTT_publisher.ino

```
1  #include <ESP8266WiFi.h>
2  #include <PubSubClient.h>
3
4  const char* ssid = "IoT";
5  const char* pwd = "Adani@123";
6  const char* mqtt_server =
          "broker.mqtt-dashboard.com";
7
8  WiFiClient espClient;
9  PubSubClient client(espClient);
10 int value = 0;
11 unsigned long lastMsg = 0;
12 #define MSG_BUFFER_SIZE  (50)
13 char msg[MSG_BUFFER_SIZE];
```

Including the "ESP8266WiFi" library for enabling Wi-Fi connection with protocols.

Including the "PubSubClient" library for use of MQTT protocol and their functions.

Defining parameters for Wi-Fi connection.

Defining MQTT server/broker for publishing the message on particular topic.

MQTT Publisher – Code Implementation

MQTT_publisher.ino

```
14 void setup_wifi() {
15   delay(10);
16   Serial.println();
17   Serial.print("Connecting to ");
18   Serial.println(ssid);
19   WiFi.mode(WIFI_STA);
20   WiFi.begin(ssid, pwd);
21
22   while(WiFi.status()!=WL_CONNECTED)
23   {
24     delay(500);
25     Serial.print(".");
26   }
27   Serial.println("WiFi connected");
28   Serial.println("IP address: ");
29   Serial.println(WiFi.localIP());
30 }
```

Initializing Wi-Fi module as station mode.

Connecting Wi-Fi module with given SSID and Password

Try until Wi-Fi is not connected to given SSID with delay of half second.

MQTT Publisher – Code Implementation

MQTT_publisher.ino

```
31 void setup() {
32   Serial.begin(115200);
33   setup_wifi();
34  client.setServer(mqtt_server,1883);
35   while (!client.connected()) {
36     Serial.print("Attempting MQTT");
37     String clientId="ESPClient1234";
38     if (client.connect(
39                   clientId.c_str())) {
40       Serial.println("connected");
         client.publish("Adani/7thsem
41         /ICT", "You are at AIIE");
42     }
43   }
}
```

Connects the client to the MQTT Server stored in *mqtt_server* with port number 1883.

Execute the syntax in loop until ESP is not connected to MQTT broker.

Defining a *clientId* for MQTT server/broker.

Returns the pointer of string *clientId* to the array.

Publishing announcement message after connecting to the MQTT broker with Topic name : Adani/7thsem/ICT

MQTT Publisher – Code Implementation

MQTT_publisher.ino

```
44 void loop() {
45
46   client.loop();
47   unsigned long now = millis();
48   if (now - lastMsg > 2000)
49   {
50     lastMsg = now;
51     ++value;
52     snprintf (msg, MSG_BUFFER_SIZE,
53         "hello world #%ld", value);
54    Serial.print("Publish message:");
55     Serial.println(msg);
56     client.publish(" Adani/7thsem
         /ICT ", msg);
57   }
58 }
```

This function allows client to process incoming messages, publish data and refresh the connection.

Creating a string in *msg* with given buffer size and assigning the value “hello world” with concatenation of integer named *value*.

Publishes the *msg* in the given topic name: Adani/7thsem/ICT

MQTT Subscriber – Code Implementation

- The code for MQTT subscriber is as shown in below.

MQTT_subscriber.ino

```
1  #include <ESP8266WiFi.h>
2  #include <PubSubClient.h>
3
4  const char* ssid = "IoT";
5  const char* pwd = "Adani@123";
6  const char* mqtt_server =
       "broker.mqtt-dashboard.com";
7
8  WiFiClient espClient;
9  PubSubClient client(espClient);
```

Including the "ESP8266WiFi" library for enabling Wi-Fi connection with protocols.

Including the "PubSubClient" library for use of MQTT protocol and their functions.

Defining parameters for Wi-Fi connection.

Defining MQTT server/broker for publishing the message on particular topic.

MQTT Subscriber – Code Implementation

MQTT_subscriber.ino

```
10 void setup_wifi() {
11   delay(10);
12   Serial.println();
13   Serial.print("Connecting to ");
14   Serial.println(ssid);
15   WiFi.mode(WIFI_STA);
16   WiFi.begin(ssid, pwd);
17
18   while(WiFi.status()!=WL_CONNECTED)
19   {
20     delay(500);
21     Serial.print(".");
22   }
23   Serial.println("WiFi connected");
24   Serial.println("IP address: ");
25   Serial.println(WiFi.localIP());
26 }
```

Initializing Wi-Fi module as station mode.

Connecting Wi-Fi module with given SSID and Password

Try until Wi-Fi is not connected to given SSID with delay of half second.

MQTT Subscriber – Code Implementation

MQTT_subscriber.ino

```
27 void callback(char* topic,
   byte* payload, unsigned int length)
28 {
29   Serial.print("Message arrived [");
30   Serial.print(topic);
31   Serial.print("] ");
32   for (int i = 0; i < length; i++)
33   {
34     Serial.print((char)payload[i]);
35   }
36   Serial.println();
37 }
```

A *callback* function for receiving messages every time when *client.loop( )* executes.

This function accepts the arguments *char* of topic*, *char* of payload* and the *length of payload*.

Executing the for loop until all the characters of the payload are printed.

Typecasting of *payload[i]* into equivalent character and then printing on serial monitor.

MQTT Subscriber – Code Implementation

MQTT_subscriber.ino

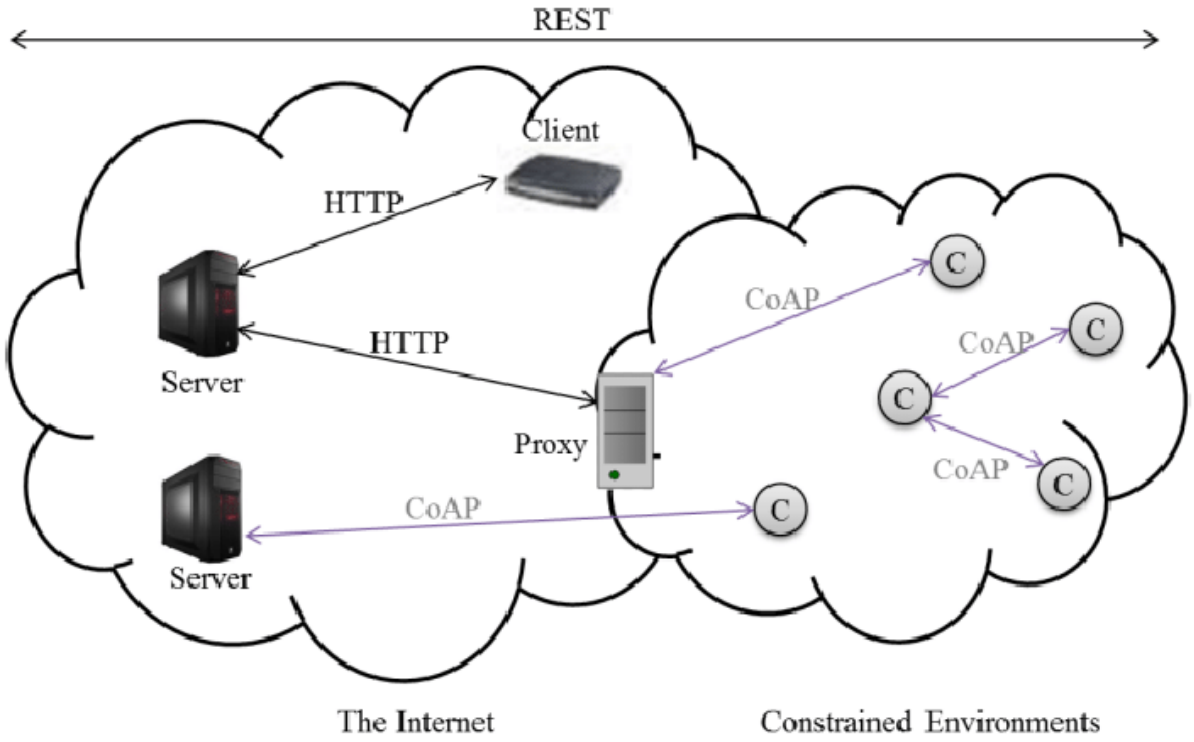
```
38 void setup() {
39   Serial.begin(115200);
40   setup_wifi();
41  client.setServer(mqtt_server,1883);
42   client.setCallback(callback);
43   while (!client.connected()) {
44    Serial.print("Attempting MQTT");
45    String clientId="ESPClient1234";
46    if (client.connect(
47                   clientId.c_str())) {
48     Serial.println("connected");
49    client.subscribe("Adani/Msg");
50     }
51   }
52 }
53 void loop() {
54   client.loop(); }
```

This is built-in function *client.setCallback( )* in *PubSubClient* library and it calls a function *callback( )* every time when *client.loop( )* function is executed.

Subscribing to the Topic named “Adani/Msg”

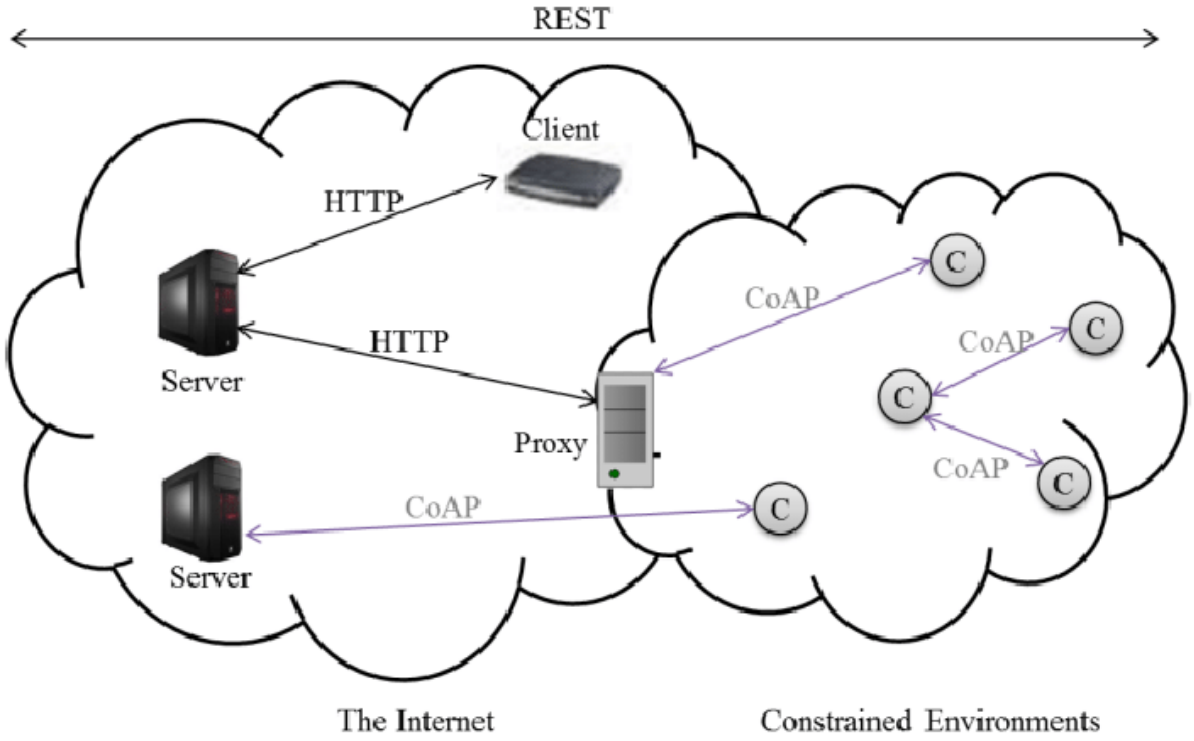
Constrained Application Protocol (CoAP)

- CoAP is designed by IETF (Internet Engineer Task Force) to work in constrained environment.
- It is a one-to-one communication protocol.
- CoAP is also light weight like MQTT protocol, and it uses less resources than HTTP.
- HTTP runs over TCP and is connection oriented while CoAP runs over UDP and it is connection less.



CoAP Architecture

- CoAP is based on the RESTful architecture (Representational State Transfer).
- REST approach ensures the secure, fault tolerant and scalable system.
- The CoAP optimizes the length of the datagram.
- CoAP is connectionless protocol and it requires retransmission support.

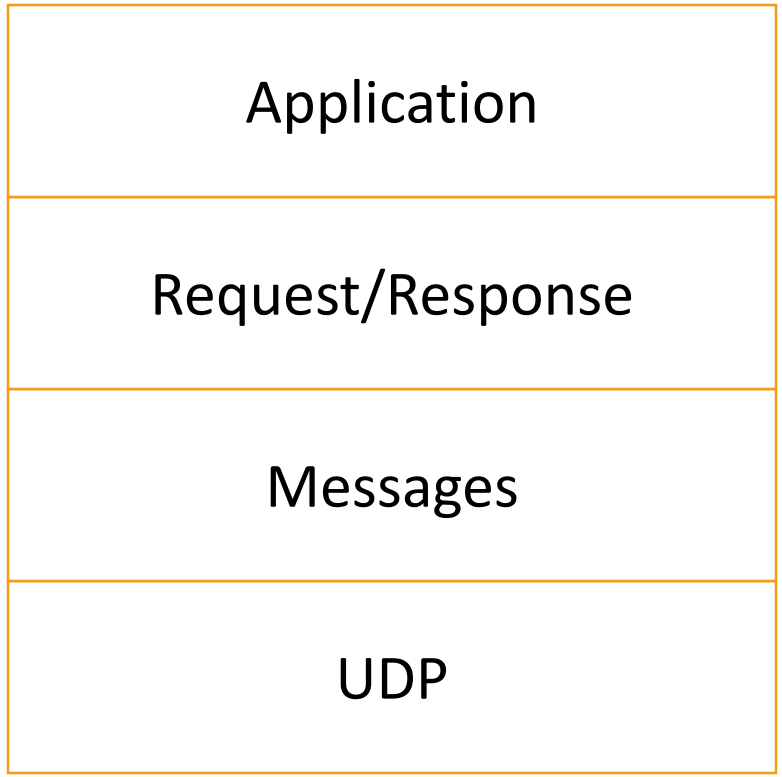


CoAP Layers

- CoAP has four layers:

1. UDP
2. Messages
3. Request-Response
4. Application

- The message layer is designed to deal with UDP and asynchronous switching.
- The request/response layer concerns communication method and deal with request/response messages.
- In the request/response layer, clients may use GET/PUT/DELET methods to transmit the message.



Message Layer in CoAP

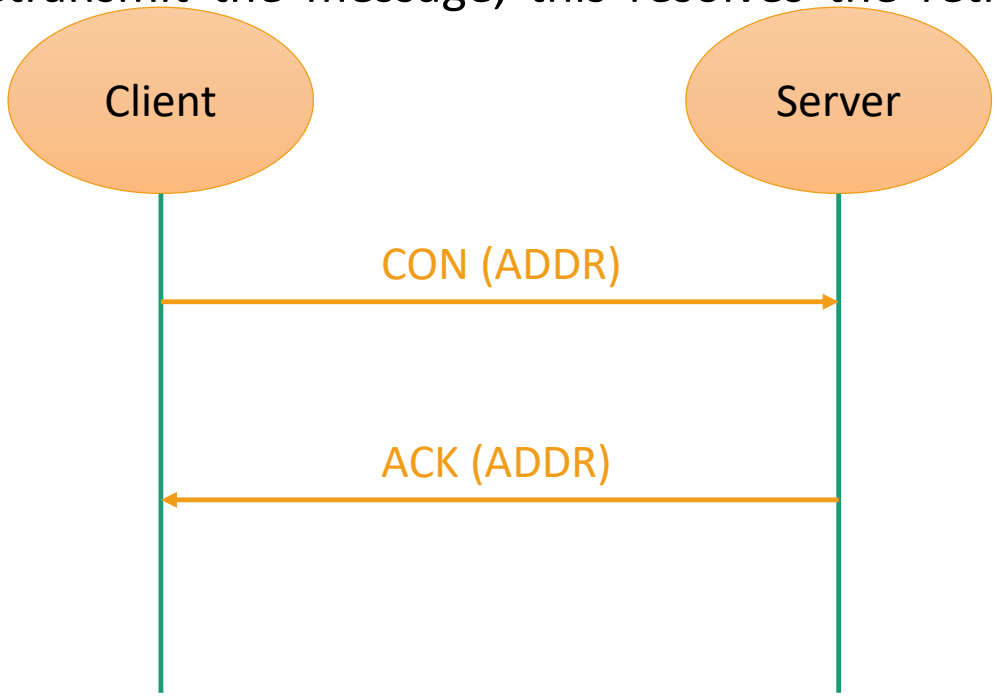
- CoAP message layer supports four types of messages:

1. Confirmable – Reliable Messaging (CON)
2. Non-confirmable – Non-reliable Messaging (NON)
3. Acknowledgement (ACK)
4. Reset (RST)

Message Layer in CoAP

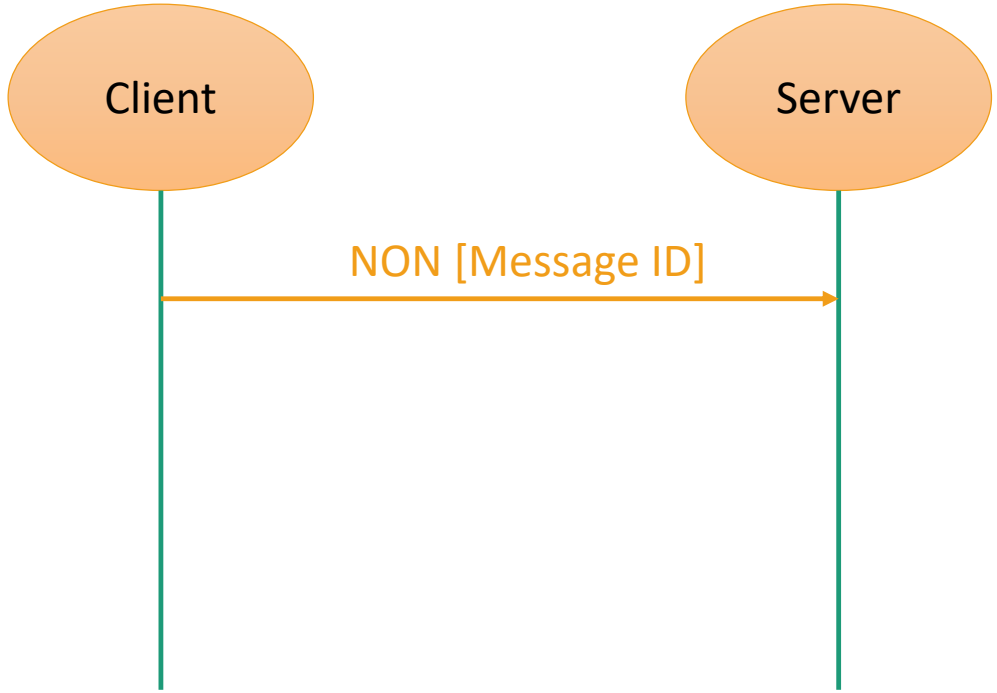
- CoAP message layer supports four types of messages:

1. Confirmable – Reliable Messaging (CON):
   - This is reliable approach because the retransmission of message occurs until the acknowledgement of the same message ID is received.
   - If there is a timeout or fail of acknowledgement, the RST message will be sent from the server as a response.
   - Hence, the client will retransmit the message, this resolves the retransmission and it is considered a reliable approach.



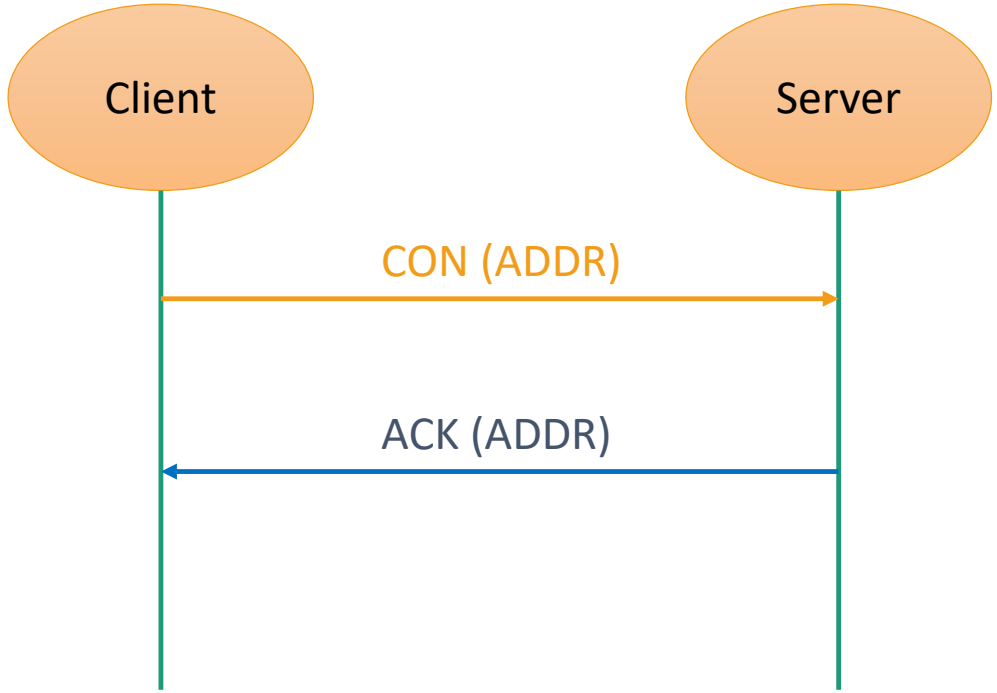
Message Layer in CoAP (Contd.)

2. Non-confirmable – Non-reliable Messaging (NON):
   - In this message transmission style, there is no reliability is ensured.
   - The acknowledgement is not issued by the server.
   - The message has ID for supervision purpose, if the message is not processed by the server it transmits the RST message.



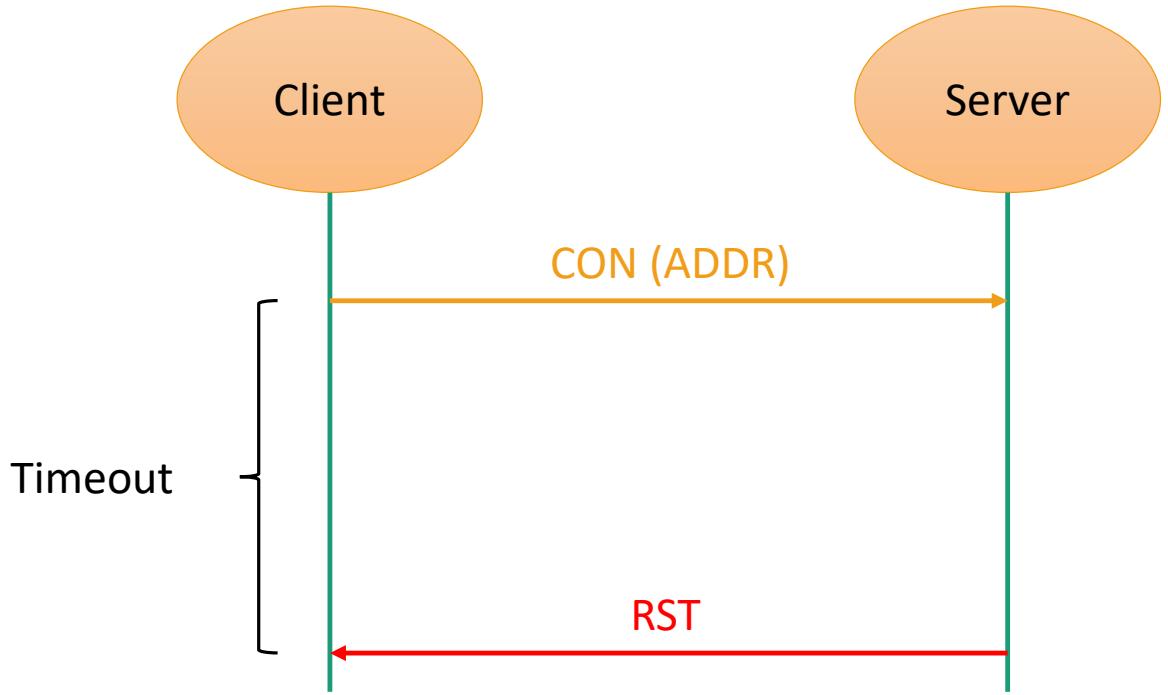
Message Layer in CoAP (Contd.)

3. Acknowledgement (ACK):
   - In this messaging, the traditional acknowledgement message sent as usual protocol.
   - We can compare this with regular handshaking scheme.
   - The handshake is automated process that establishes a link for communication before actual data transfer begins.



Message Layer in CoAP (Contd.)

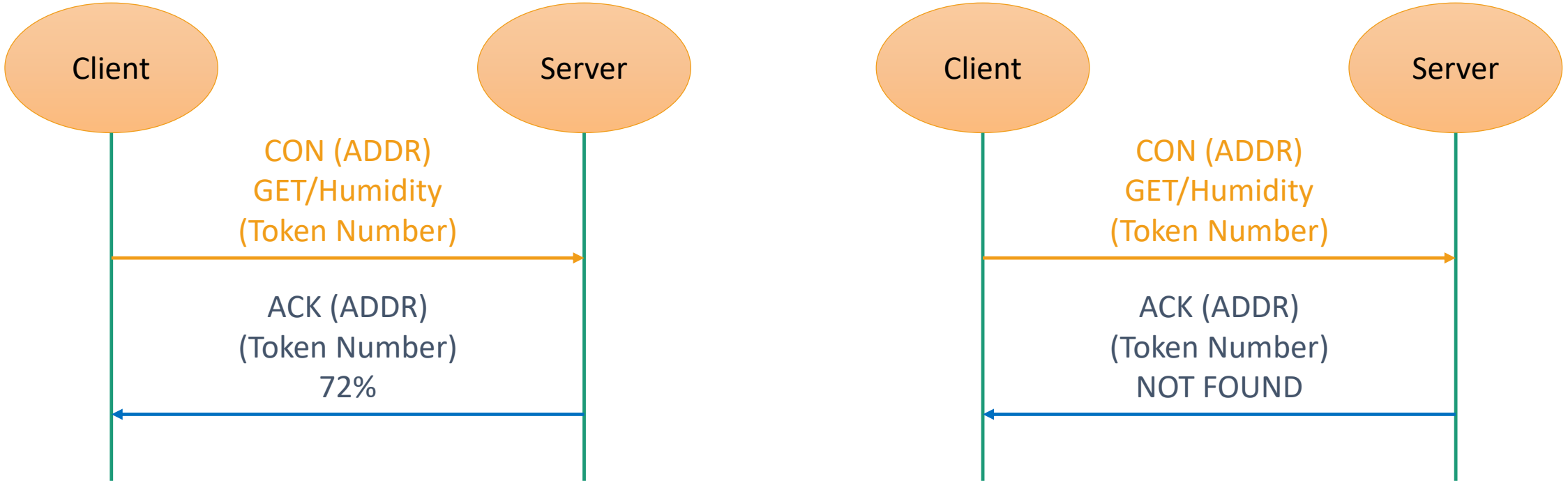
4. Reset (RST):
   - The receiver is expecting the message from sender of a particular message ID.
   - If no message is processed or received before specific amount of time (called timeout), the message called RESET is transmitted from receiver.
   - This message informs the sender that there is a trouble in transmission of messages.



Request-Response Layer in CoAP

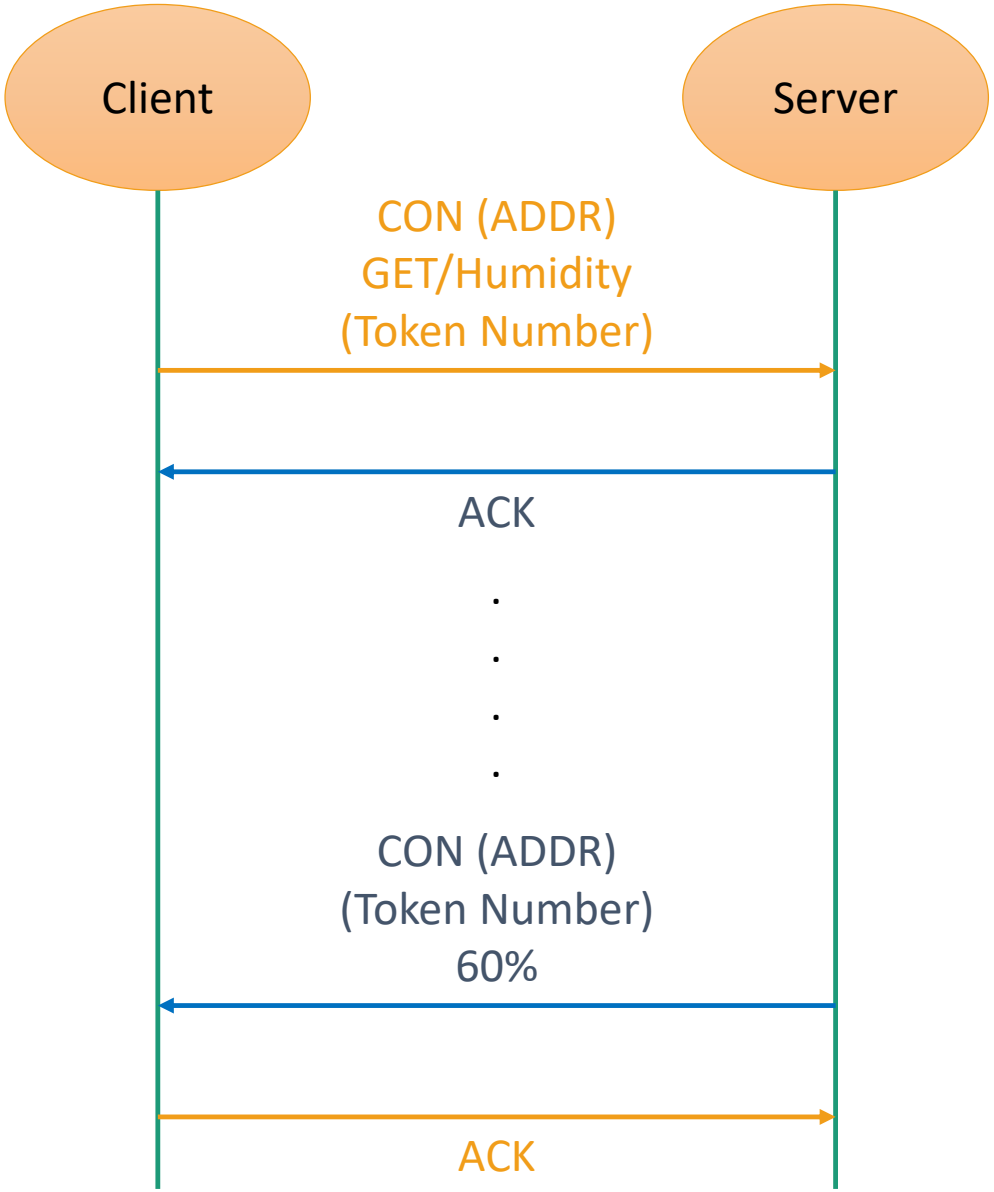
- There are three modes in CoAP request-response layer.

1. Piggy-Backed:
   - In this mode, the client sends the data with particular method (GET, PUT etc.) with token number and CON/NON messaging method.
   - The ACK is transmitted by server immediately with corresponding token number and message.
   - If the message is not received by server or data is not available then the failure code is embedded in ACK.



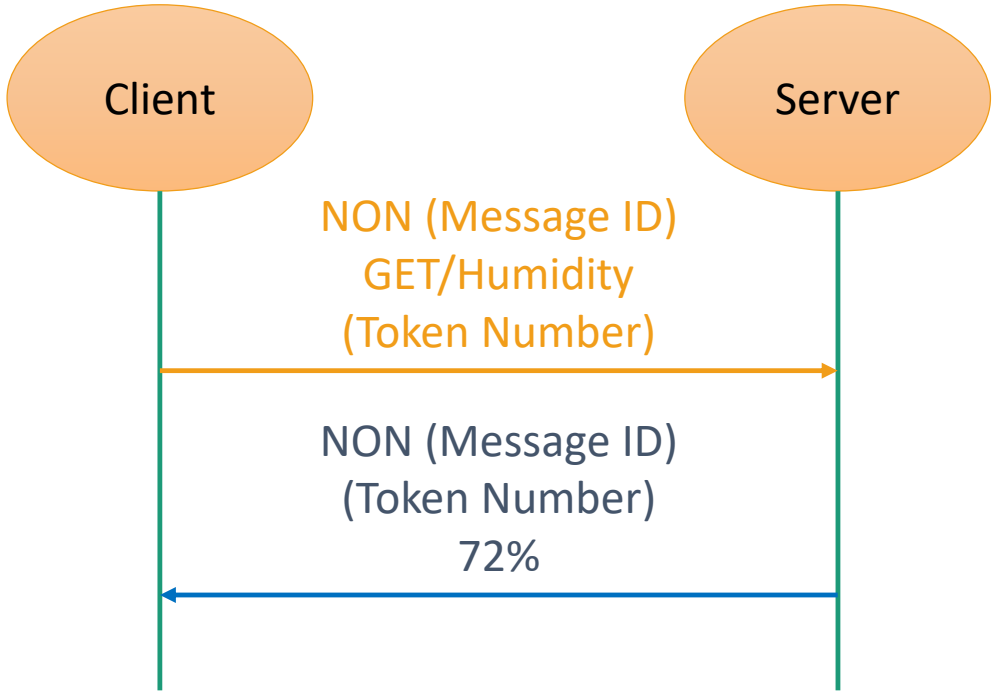
Request-Response Layer in CoAP (Contd.)

2. Separate Response:
   - In this mode, the client sends the data with particular method (GET, PUT etc.) with token number and CON messaging method.
   - If the server is unable to respond immediately, an empty ACK will be reverted.
   - After some time, when the server is able to send the response, it sends a CON message with data.
   - In response to that, ACK is sent back from client to server.



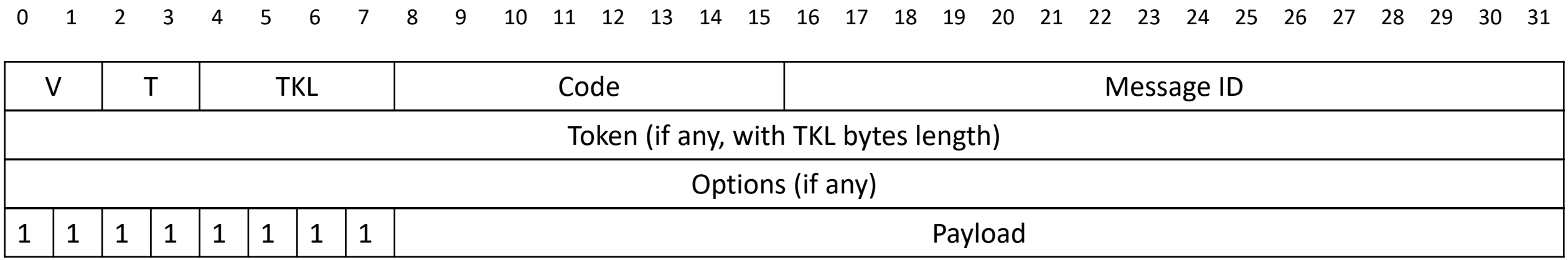
Request-Response Layer in CoAP (Contd.)

3. Non-Confirmable Request and Response:
   - In this mode, the client sends the data with particular method (GET, PUT etc.) with token number and NON messaging method.
   - The server does not give ACK in NON messaging method, so it will send a NON type response in return with token number and its data.



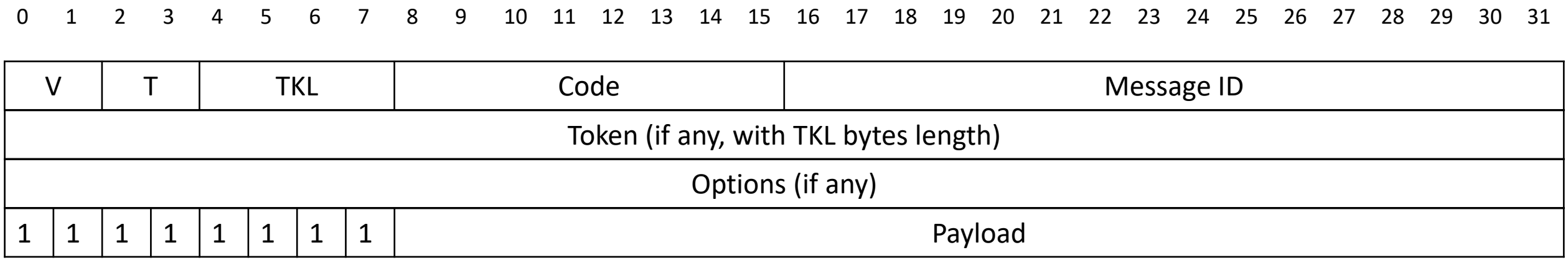
CoAP Message Format

- The CoAP message format is of 16 Bytes consisting various fields.
- V : It refers to version of CoAP. It is two bit unsigned integer.
- T : It refers to message type. It is also two bit unsigned integer.
  - Confirmable (0)
  - Non-Confirmable (1)
  - ACK (2)
  - RESET (3)
- TKL : It refers to the token length and is four bit unsigned integer.
- Code: It refers to the response code.



CoAP Message Format (Contd.)

- Message ID: It is identifier of the message and to avoid duplication.
- Token : Optional field whose size is indicated by the Token Length field.
- Options : This field is used if any options available and having length of 4 Bytes.
- Payload : This field refers to the payload data where actual data is transmitted.



Difference between MQTT & CoAP

	MQTT	CoAP
Underlying protocol	TCP (connection oriented)	UDP (connectionless)
Communication	**M:N (Many to many)**	**1:1 (One-to-one)**
Power	**Higher than CoAP. Lesser than other protocols.**	**Lowest, consumes less power than MQTT, making it the best for point to point communication.**
Model	**Publisher/Subscriber**	**Request-response (RESTful)**

XMPP Protocol

- The full form of XMPP is Extensible Messaging and Presence Protocol.
- It is designed for messaging, chat, video, voice calls and collaboration.
- The fundamental base of this protocol is XML and it is used for messaging services such as WhatsApp.
- The XMPP was originally named as Jabber and later known as XMPP.

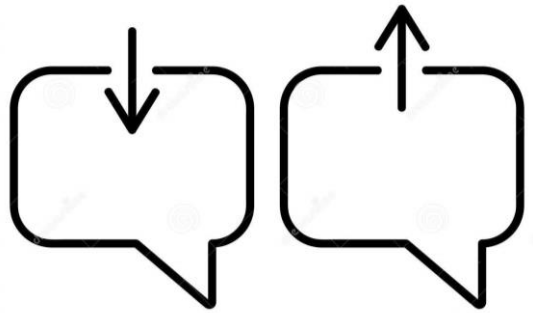
XMPP denotation

- The denotation for XMPP can be given as following:
- X – X denotes eXtensible.
  - It is considered extensible as it is designed to accept and accommodate any changes.
  - The protocol is defined in open standard and it is extensible because of open standard approach.
- M – M denotes Messaging.
  - XMPP supports sending message to the recipients in real time.
- P – P denotes Presence.
  - It is helpful to **identify the status** such as “Offline” or “Online” or “Busy”.
  - It also helps in understanding if the recipient is ready to receive the message or not.
- P – P denotes Protocol.
  - The set of standards together acting as a protocol.

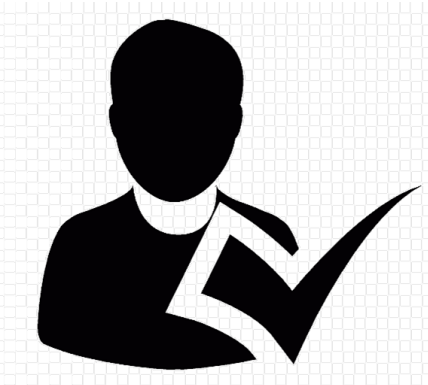
Messenger Requirements

- Any messenger requires the following features inbuilt :

1. Message sending and receiving features.
2. Understanding the presence/absence status.
3. Managing subscription details.
4. Maintaining the contact list.
5. Message blocking services.



- Note that all the listed features are built in XMPP.



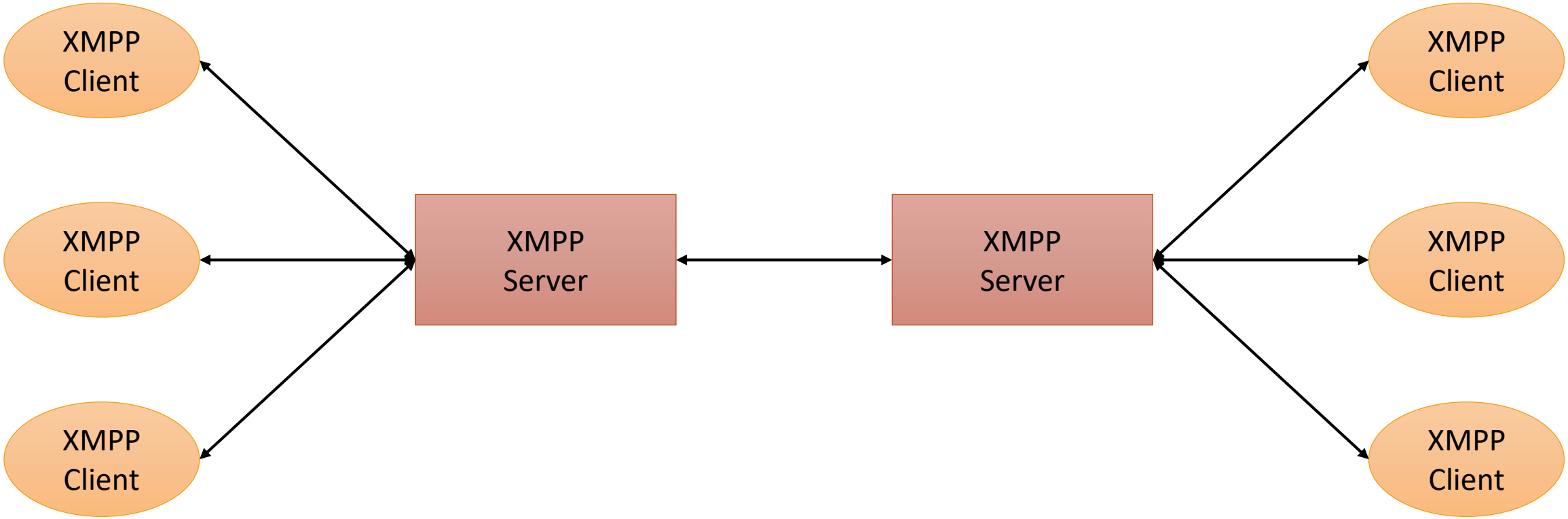
XMPP Features:

- The following are features of XMPP:

1. **Decentralized:** XMPP is similar to that of e-mail. Anyone can run their own XMPP server.
2. **Secure:** XMPP is very secure with end-to-end encryption.
3. **Stable & functionally proven:** It is functionally proven since its initial version Jabber launched in 1998.
4. **Standardization:** IETF published XMPP specifications as RFCs in RFC3920 & 3921 standards.
5. **Flexibility:** XMPP is not just messaging oriented; it is flexible enough for file sharing, gaming & even remote monitoring of systems.

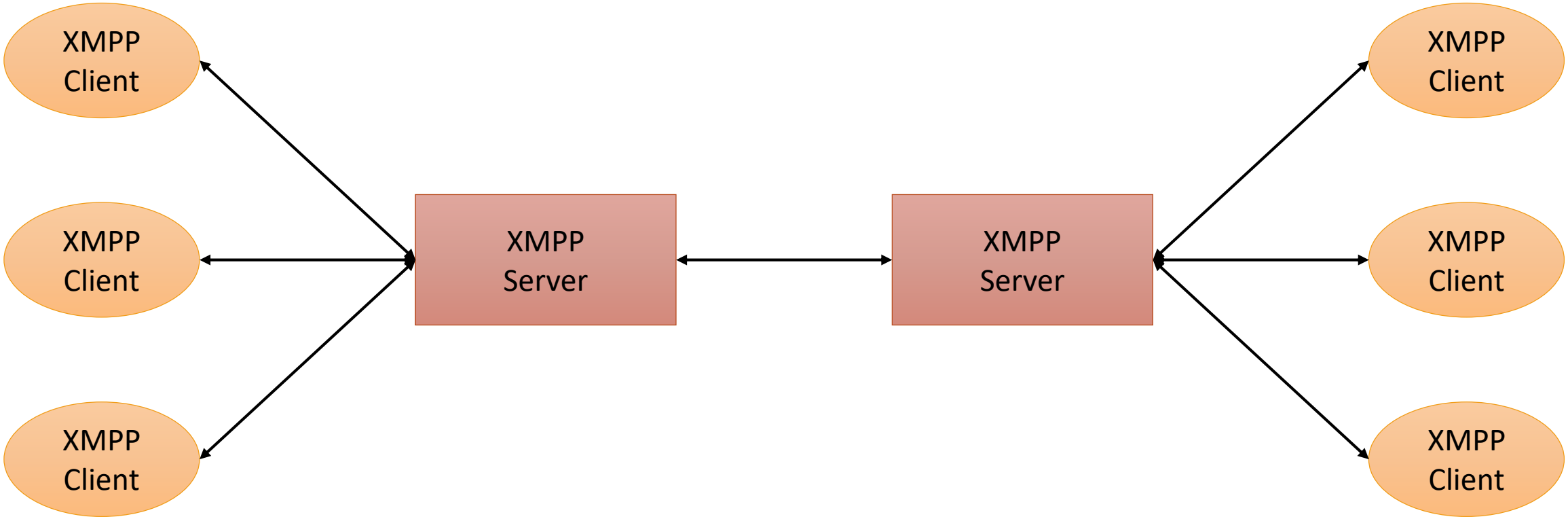
XMPP Architecture

- The architecture of XMPP protocol is shown in the figure :
- The XMPP clients communicate with XMPP server bidirectional.
- There can be any numbers of clients connected to XMPP server.
- The XMPP servers may be connected to the other clients or other XMPP servers.



XMPP Architecture (Contd.)

- The initial version of the XMPP was with TCP using open ended XML.
- After certain period of time, the XMPP was developed based on HTTP.
- The XMPP can work with HTTP is through two different methods Polling and Binding.
- What is Polling and Binding?



XMPP Architecture (Contd.)

- In polling method, the messages stored in the server are pulled or fetched.
- The fetching is done by XMPP client through HTTP GET and POST requests.
- In binding method, Bidirectional-streams Over Synchronous HTTP (BOSH) enables the server to push the messages to the clients when they are sent.

- The binding approach is more effective than polling.
- Also both method uses HTTP, hence the firewall allows the fetch and post activity without any difficulties.

Addressing & Identification Protocols

Section - 3

Internet Protocol Version 4 (IPv4)

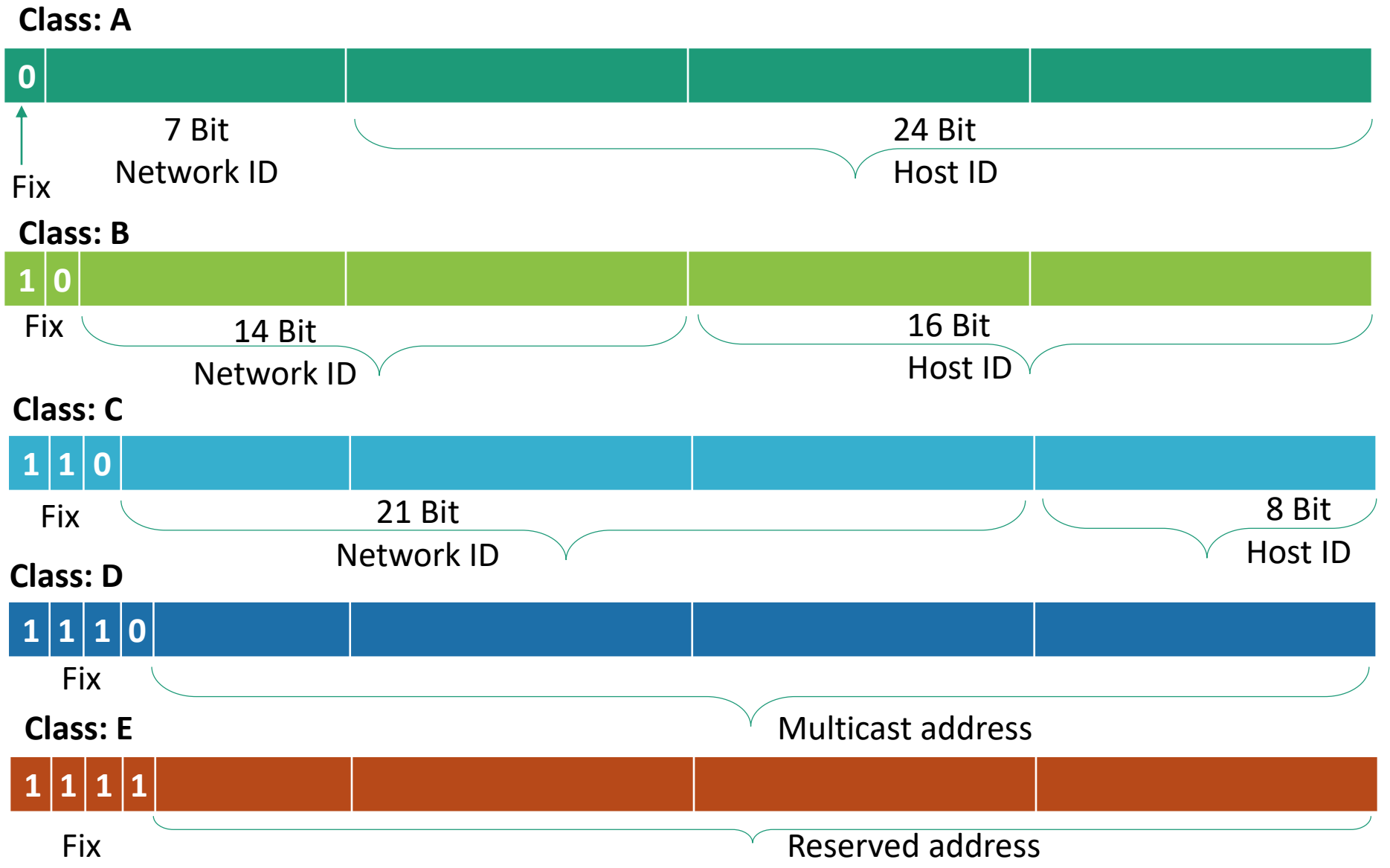
- IP addresses are useful in identifying a specific host in a network.
- IP addresses are 32-bit numbers which are divided into 4 octets. Each octet represents 8 bit binary number.
- Below is an example of an IP address:

10101100	00010000	11111110	00000001
172	**16**	**254**	**1**

IP addresses are divided into 2 parts:
Network ID & Host ID

<NID> <HID> = IP Address

Classification of IP Address

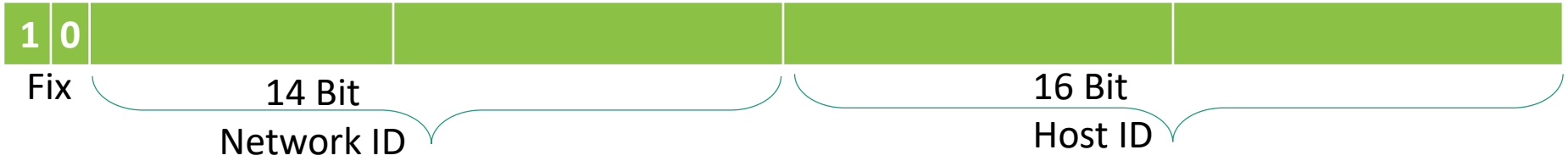


Class A

- Only 126 addresses are used for network address.
- All 0's and 1's in Network-ID are dedicated for special IP address. So, total number of IP address in class A can be represented as following.
- The range of Class A is 0.0.0.0 to 127.255.255.255 around 17 million hosts/network.
- Ping Address: 127.0.0.1 used for self checking for trouble shooting & network testing.

0.0.0.0	Special IP Address
00000001.0.0.1 1.0.0.2 1.0.0.3 . . . 126.255.255.254	$2^{24} - 2$ are Host IP
127.255.255.255	Special IP Address – Loopback

Class B



- No special network address here. All are usable.
- The total number of hosts connected in network can be given as following:
- It is used in medium size network.
- The range of Class B is 128.0.0.0 to 191.255.255.255

128.0.0.0	Special IP Address
10000001.0.0.1 130.0.0.2 130.0.0.3 . . . 190.255.255.254	$2^{16} - 2$ are Host IP
10111111.255.255.255	Special IP Address – Loopback

Class C

- This class is used in small networks.
- The calculation for number of hosts can be given as following:
- Normally, the LANs are configured with Class C.
- The range for Class C is given as 192.0.0.0 to 223.255.255.255

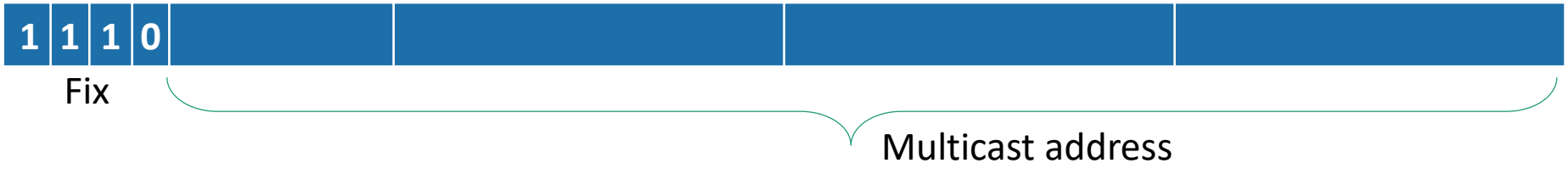
192.0.0.0	Special IP Address
11000001.0.0.1 194.0.0.2 194.0.0.3 . . . 222.255.255.254	$2^8 - 2$ are Host IP
11011111.255.255.255	Special IP Address – Loopback

Class D

- Very first four bits of the first octet in Class D IP addresses are set to 1110, giving a range of:

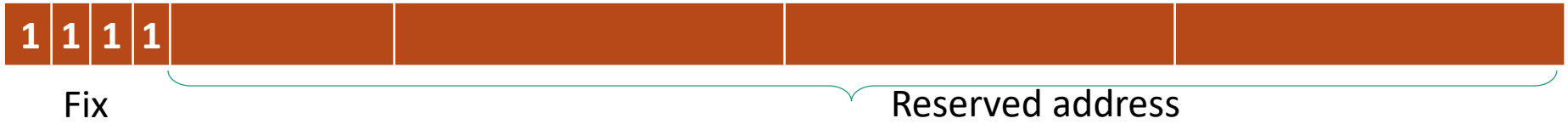
11100000 – 11101111
224 – 239

- Class D has IP address rage from 224.0.0.0 to 239.255.255.255.
- Class D is reserved for Multicasting.
- In multicasting data is not destined for a particular host, that is why there is no need to extract host address from the IP address, and Class D does not have any subnet mask.



Class E

- This IP Class is reserved for experimental purposes only for R&D or Study.
- IP addresses in this class ranges from 240.0.0.0 to 255.255.255.254.
- Like Class D, this class too is not equipped with any subnet mask.



IPv4 Format

- Figure shows the format for IPv4.
- VER (Version):
  - It is a 4 bits field which indicates the version of IP.
- IHL (Internet Header Length):
  - It specifies the Internet Header Length in 32 bits word length and points the beginning of data.
- Types of Service:
  - It is a 8 bit field which specifies the Quality of Service (QoS). The format for these 8 bits are given here.

<table>
<tr><td>VER (4)</td><td>IHL (4)</td><td>Types of Service (8)</td><td colspan="2">Total Length (16)</td></tr>
<tr><td colspan="3">Identification (16)</td><td>Flags(3)</td><td>Fragment Offset (13)</td></tr>
<tr><td colspan="2">Time to Live (8)</td><td>Protocol (8)</td><td colspan="2">Header Checksum (16)</td></tr>
<tr><td colspan="5">Source Address (32)</td></tr>
<tr><td colspan="5">Destination Address (32)</td></tr>
<tr><td colspan="5">Options (if any)</td></tr>
</table>

IPv4 Format

- Types of Service:
  - Precedence (3) : They are used for future options.
  - Delay (1): It indicates the delay of either normal or lower.
  - Throughput (1): Indicates the speed of throughput of either normal or higher rate.
  - Reliability (1): It indicates normal reliability or high reliability.

Precedence	Delay	T	R	Reserved Bits

VER (4)	IHL (4)	Types of Service (8)	Total Length (16)	
Identification (16)			Flags(3)	Fragment Offset (13)
Time to Live (8)	Protocol (8)		Header Checksum (16)	
Source Address (32)				
Destination Address (32)				
Options (if any)				

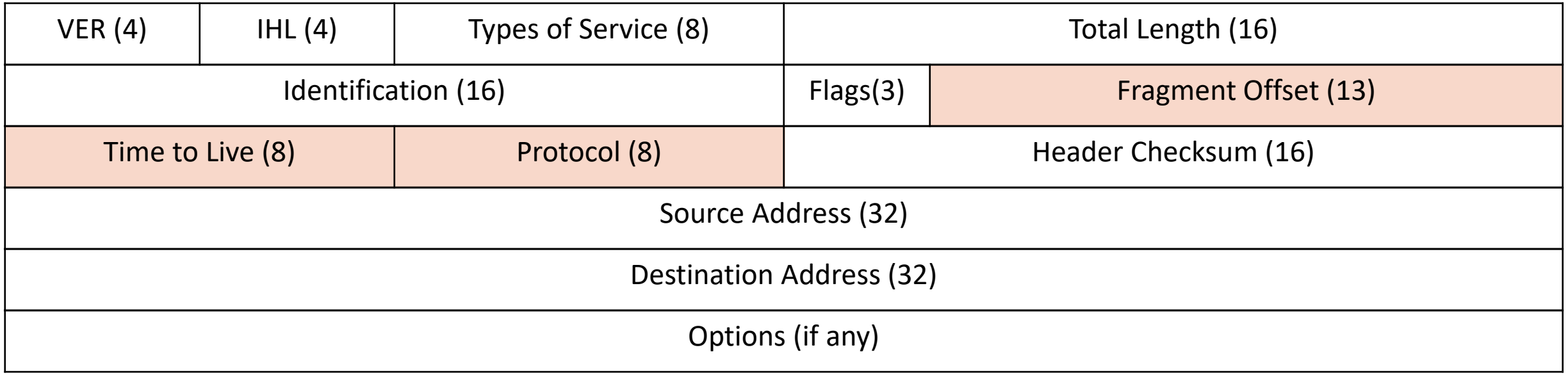
IPv4 Format

- Total Length:
  - This is a 16 bit field defining the length of IPv4 datagram. The minimum length of datagram is 20 bytes and maximum is 65535.
- Identification:
  - This field is of 16 bits which helps in assembling the fragments and added by the senders.
- Flags:
  - There are3 bits in flags. First is always ‘1’. Second is DF (Don’t Fragment) and third is MF (More Fragment).

VER (4)	IHL (4)	Types of Service (8)	Total Length (16)	
Identification (16)			Flags(3)	Fragment Offset (13)
Time to Live (8)		Protocol (8)	Header Checksum (16)	
Source Address (32)				
Destination Address (32)				
Options (if any)				

IPv4 Format

- Fragment Offset:
  - When fragmentation is done, this specifies the offset or position of overall message.
- Time to Live:
  - This is 8 bit field that indicates the time period of datagram to survive.
- Protocol:
  - This field identifies the protocol for higher layer in the IP datagram.



IPv4 Format

- Header Checksum:
  - It is 16 bit field to provide basic protection in transmission of header via checksum.
- Source Address:
  - This is 32 bit IP address of the source of the datagram.
- Destination Address:
  - This is 32 bit IP address of the destination of the datagram.
- Options:
  - There are many optional header available for debug and test purpose.

<table>
<tr><td>VER (4)</td><td>IHL (4)</td><td>Types of Service (8)</td><td colspan="2">Total Length (16)</td></tr>
<tr><td colspan="3">Identification (16)</td><td>Flags(3)</td><td>Fragment Offset (13)</td></tr>
<tr><td colspan="2">Time to Live (8)</td><td>Protocol (8)</td><td colspan="2">Header Checksum (16)</td></tr>
<tr><td colspan="5">Source Address (32)</td></tr>
<tr><td colspan="5">Destination Address (32)</td></tr>
<tr><td colspan="5">Options (if any)</td></tr>
</table>

IPv6 Addressing

- As the number of devices grow connected to the internet, it is obvious that we need more number of IP addresses to assign all the devices.
- IPv4 IP addressing uses 32 bit IP addresses which can generate
- $2^{32} = 4294967296$
- The above value is just above 4 billion. In fact the actual available IP addresses in IPv4 is even less than theoretical value.
- The actual IP addresses that can be assigned in IPv4 are 3,720,249,092.
- The IPv6 IP addressing uses 128 bit IP addresses which can generate
- $2^{128} = 3.40282366920938463463374607431177\ x\ 10^{38}$
- The above value is very large that if we assign a unique IP to every item in the world then also it will have reserved IP addresses.,

Features of IPv6 Addressing

- It has better and efficient routing than IPv4.
- The additional flow label is added in header of IPv6 for improvement in QoS.
- IPv6 has new application like IP telephony, video/audio, interactive games etc. with ensured QoS.
- The plug and play abilities have been improved in IPv6.
- IPv6 has eliminated the need of Network Address Translation due to the huge availability of IP addresses.

IPv6 Header Format

- IPv6 header format has less fields than IPv4 header format.
- IPv6 header is 40 bytes which is twice than IPv4 header.
- The figure shows IPv6 header format.
- The IPv6 has simple packet handling and improved forwarding efficiency.
- The fields of IPv6 header can be explained as following:

IP Version Number (4)	Traffic Class (8)	Flow Label (20)
Payload Length (16)	Next Header (8)	Hop Limit (8)
Source Address (128)		
Destination Address (128)		

IPv6 Header Format

- IP version :
  - It is 4 bit field to represent the IP version being used.
- Traffic Class :
  - It is 8 bit field to identify the different classes or priorities of IPv6 packets.
  - It replaces the type of services in IPv4.
  - Upper 6 bits are used for type of services lower 2 bits are used for Explicit Congestion Notification.
- Flow Label :
  - It is 20 bit field used to identify the sequence of packets.
  - It is also useful for prioritizing the delivery of packet and providing real time service.
  - The higher priority packets can be delivered ahead of lower priority packets.

IP Version Number (4)	Traffic Class (8)	Flow Label (20)
Payload Length (16)	Next Header (8)	Hop Limit (8)
Source Address (128)		
Destination Address (128)		

IPv6 Header Format

- Payload Length :
  - It is a 16 bit field which identifies the length of payload of IPv6.
- Next Header :
  - It is a 8 bit field which is similar to protocol field in IPv4 header.
  - It represents the type of extension header that follows the primary IPv6 header.
- Hop Limit :
  - It is 8 bit field equivalent to Time to Live in IPv6 Header.
  - The value is decremented by 1 every time when it is forwarded by the host.
  - When this value reaches 0, the packet is discarded.

IP Version Number (4)	Traffic Class (8)	Flow Label (20)
Payload Length (16)	Next Header (8)	Hop Limit (8)
Source Address (128)		
Destination Address (128)		

IPv6 Header Format

- Source Address :
  - It represents the 128 bit IP address of source or sender.
- Destination Address :
  - It represents the 128 bit IP address of destination or receiver.

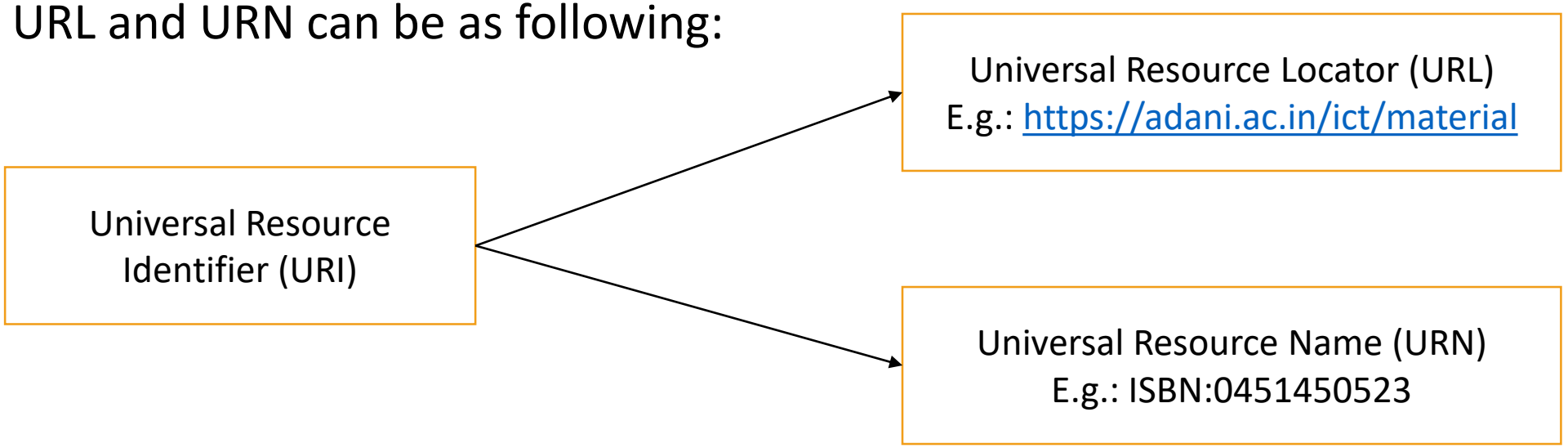
IP Version Number (4)	Traffic Class (8)	Flow Label (20)
Payload Length (16)	Next Header (8)	Hop Limit (8)
Source Address (128)		
Destination Address (128)		

IPv4 versus IPv6

	IPv4	IPv6
Developed	IETF 1974	IEF 1998
Length (bits)	32	128
No. of Addresses	2^32	2^128
Notation	Dotted Decimal	Hexadecimal
Dynamic Allocation of addresses	DHCP	SLAAC/ DHCPv6
IPSec	Optional	Compulsory

Uniform Resource Identifier (URI)

- Uniform Resource Identifier (URI) is used to identify the resources with the help of sequence of characters.
- The URI protocol was developed by IETF.
- The URI can be URL or URN.
- The sample for URL and URN can be as following:



- URN mostly provides information of unique name without locating or retrieving the resource information. When you add “access mechanism, location” then URN becomes URL.
- URL provides locating and retrieving information on a network.

Uniform Resource Locator (URL)

- The URL contains the information of how to fetch a resource information from its location.
- URL always begin with a protocol like HTTP/HTTPS
- A URL is used when client request the server for the service.
- The sample URL and its fields are explained below: