1. List and explain the advantages of Internet protocol. OR Write a note on business case for IP. (VTU July 2019):

key advantages of using Internet Protocol (IP) for the Internet of Things (IoT) in simpler English:

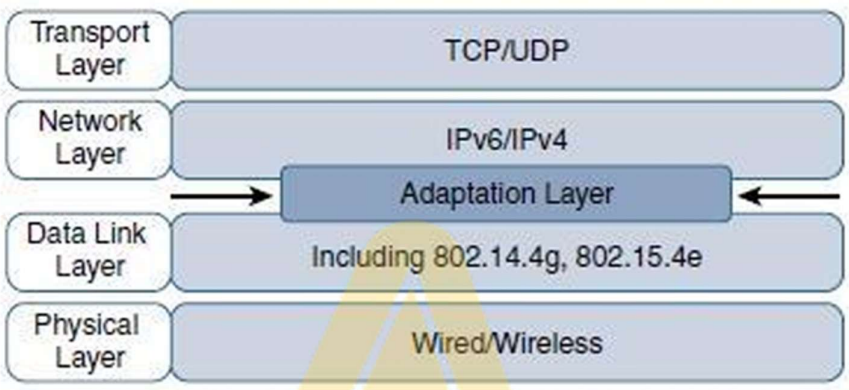
1. Open standard - IP is not owned by any one company, allowing any devices and applications to work together seamlessly.
2. Flexible - IP can be used over many different network types like Ethernet, Wi-Fi, cellular etc. to connect IoT devices.
3. Widely available - Modern operating systems and industrial systems already have IP support built-in.
4. Scalable - IP has been used to build massively scaled networks like the public internet with millions of devices.
5. Secure and manageable - Proven tools exist to secure and manage IP networks.
6. Reliable - IP has been used reliably for years in critical systems like finance and defense.
7. Consumer friendly - Consumer IoT devices will likely connect over common broadband and mobile IP networks.
8. Enables innovation - IP allows IoT to leverage new web, mobile and cloud innovations.

In simple terms, using the common IP technology allows IoT to work across different systems, scale easily, leverage existing tools/skills, and drive future innovations - providing an open, flexible and future-proof networking foundation.

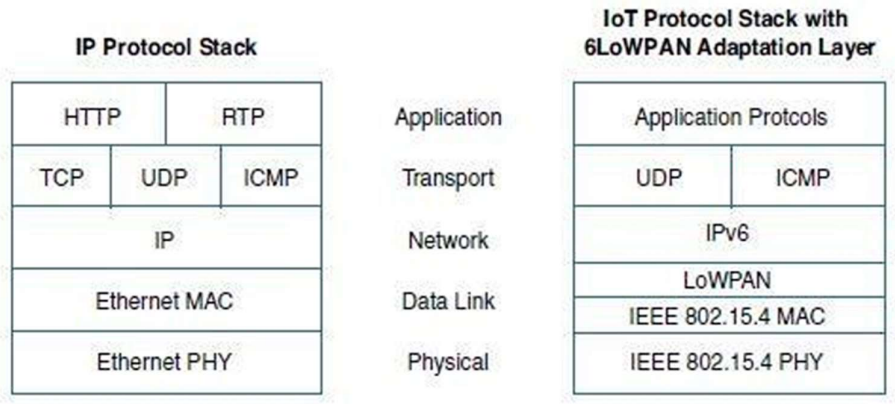
2. Illustrate with a neat block diagram, How to optimize IP for IOT using adaptation Layer.

- For IP to work over any specific physical layer (Layer 1) and data link layer (Layer 2) protocol, an adaptation layer is required to define how IP packets get encapsulated and transmitted over that protocol. This model of packaging IP over lower layers is called an adaptation layer.
- Adaptation layers are typically standardized by the Internet Engineering Task Force (IETF) and published as Request for Comments (RFC) documents.
- For constrained IoT devices and networks with limited bandwidth, power, and processing capabilities, the adaptation layer needs to include optimizations to make IP transmission more efficient.

- The key examples of such optimized adaptation layers are the ones developed by the IETF's 6LoWPAN (IPv6 over Low-Power Wireless Personal Area Networks) and its successor 6Lo working groups.



Optimizing IP for IoT Using an Adaptation Layer



Comparison of an IoT Protocol Stack Utilizing 6LoWPAN and an IP Protocol Stack

3. Describe with a neat diagrams the header stacks of 6LoWPAN

Header Compression:

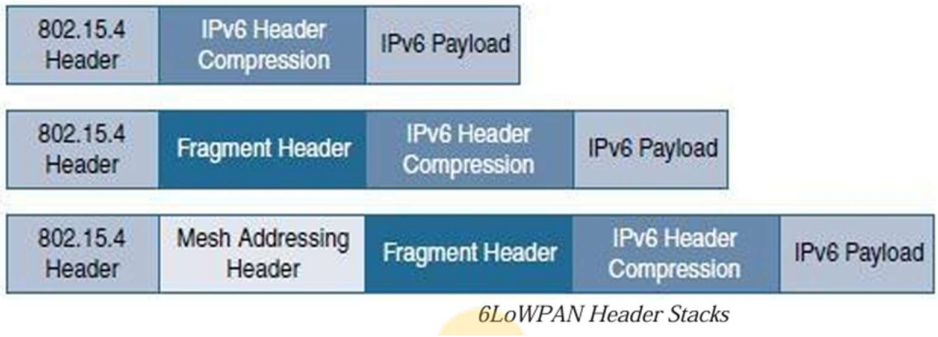
- IPv6 headers are 40 bytes and UDP headers are 8 bytes, which is overhead for constrained networks
- RFCs 4944 and 6282 defined header compression techniques for 6LoWPAN
- This can shrink the combined IPv6 and UDP headers down to as low as 6 bytes in some cases
- Reduces overhead significantly for transmission over low-bandwidth IoT networks

Fragmentation:

- IPv6 requires a minimum MTU of 1280 bytes
- Constrained IoT networks may have much smaller frame sizes
- 6LoWPAN defines fragmentation to break up large IP packets into smaller fragments
- Allows transmitting IPv6 packets over networks with limited frame sizes

Mesh Addressing:

- Enables forwarding of packets over multiple hops in a mesh network
- Defines Hop Limit, Source Address, and Destination Address fields in the 6LoWPAN header
- Hop Limit is decremented at each hop, preventing infinite loops
- Once it hits 0, the packet is dropped and no longer forwarded
- Allows multi-hop routing for extended coverage in IoT mesh networks



6LoWPAN Header Stacks

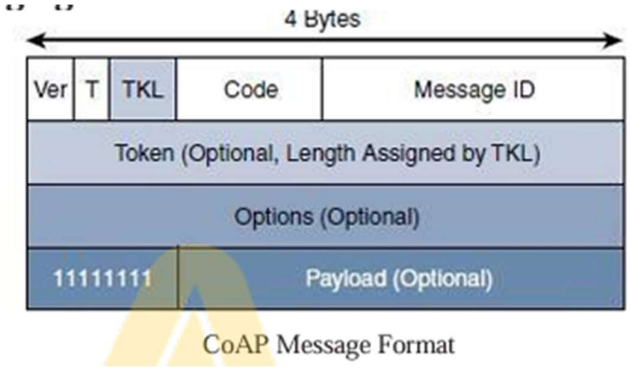
7. Explain the following with respect to CoAP i) Messaging format ii) CoAP Communications in IoT Infrastructures iii) CoAP Reliable Transmission

i) Messaging format

According to the provided information, the CoAP (Constrained Application Protocol) message format consists of the following fields:

1. Ver (Version): Identifies the CoAP version.
2. T (Type): Defines one of four message types - Confirmable (CON), Non-confirmable (NON), Acknowledgement (ACK), or Reset (RST).
3. TKL (Token Length): Specifies the size (0-8 bytes) of the Token field.
4. Code: Indicates the request method (e.g., GET, POST) for a request message and a response code for a response message.
5. Message ID: Detects message duplication and used to match ACK and RST message types to CON and NON message types.
6. Token: With a length specified by TKL, correlates requests and responses.
7. Options: Specifies option number, length, and option value. Capabilities provided by the Options field include specifying the target resource of a request and enabling proxy functions.
8. Payload: Carries the CoAP application data. This field is optional. The purpose of this byte is to delineate the end of the Options field and the beginning of the Payload.

The CoAP message format is designed to be lightweight and efficient, with a short fixed-length header (4 bytes) and variable-length fields, delivering low overhead while decreasing parsing complexity. This format facilitates the exchange of messages over UDP between endpoints, including the secure transport protocol Datagram Transport Layer Security (DTLS).



CoAP Message Format

ii) CoAP Communications in IoT Infrastructures

a. CoAP communications can take various paths across an IoT infrastructure.

b. Connections can occur between devices on the same or different constrained networks, or between devices and generic Internet or cloud servers, all operating over IP.

c. Since both HTTP and CoAP are IP-based protocols, the proxy function can be located anywhere in the network, not necessarily at the border between constrained and non-constrained networks.

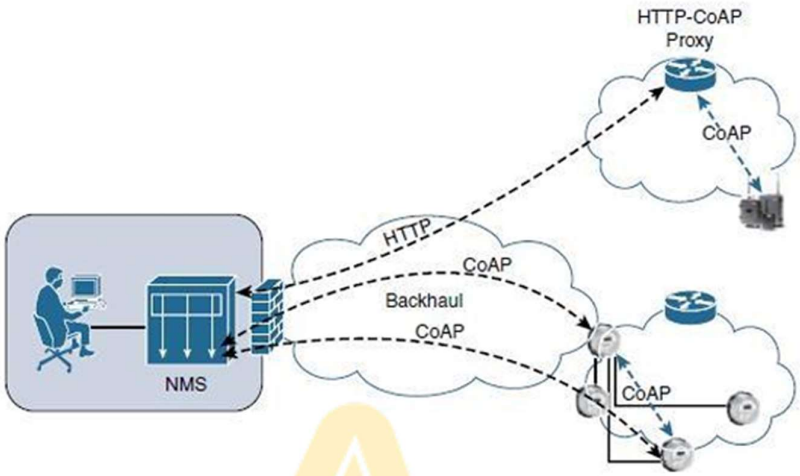
d. CoAP is based on the REST (Representational State Transfer) architecture, where a "thing" can act as both a client and a server.

e. Through the exchange of asynchronous messages, a client requests an action via a method code (GET, POST, PUT, DELETE) on a server resource.

f. A Uniform Resource Identifier (URI) localized on the server identifies the targeted resource.

g. The server responds with a response code that may include a representation of the resource.

In summary, CoAP follows a RESTful architecture, enabling flexible communication paths between constrained and non-constrained devices and servers across an IP-based IoT infrastructure, with the capability of using proxies at various points in the network.



CoAP Communications in IoT Infrastructures

iii) CoAP Reliable Transmission

The key points regarding CoAP's reliable transmission are:

a. While running over UDP, CoAP offers reliable transmission of messages when the CoAP header is marked as "confirmable".

b. CoAP supports basic congestion control with a default timeout, simple stop-and-wait retransmission with exponential back-off mechanism, and detection of duplicate messages through a message ID.

c. If a request or response is tagged as confirmable, the recipient must explicitly acknowledge or reject the message using the same message ID.

d. If a recipient cannot process a non-confirmable message, a reset message is sent.

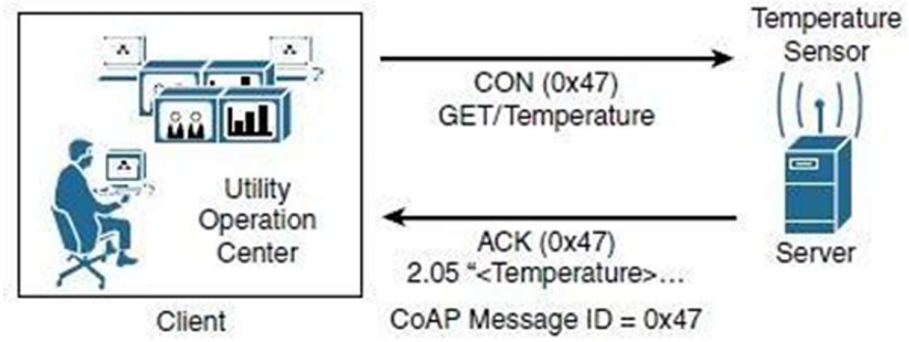
Example:

e. The example shows a CoAP client (utility operations center) and a CoAP server (temperature sensor) communicating with a message ID of 0x47 for reliability and duplicate detection.

f. The client sends a confirmable (CON) GET message with ID 0x47 to get the temperature from the sensor.

g. The sensor replies with an ACK message with the same ID 0x47, piggybacking the successful response (code 2.05) and the requested temperature data.

In summary, CoAP's reliable transmission is achieved through confirmable messages, acknowledgments, message IDs for deduplication, and retransmissions with congestion control mechanisms, despite running over the unreliable UDP protocol.



CoAP Reliable Transmission Example

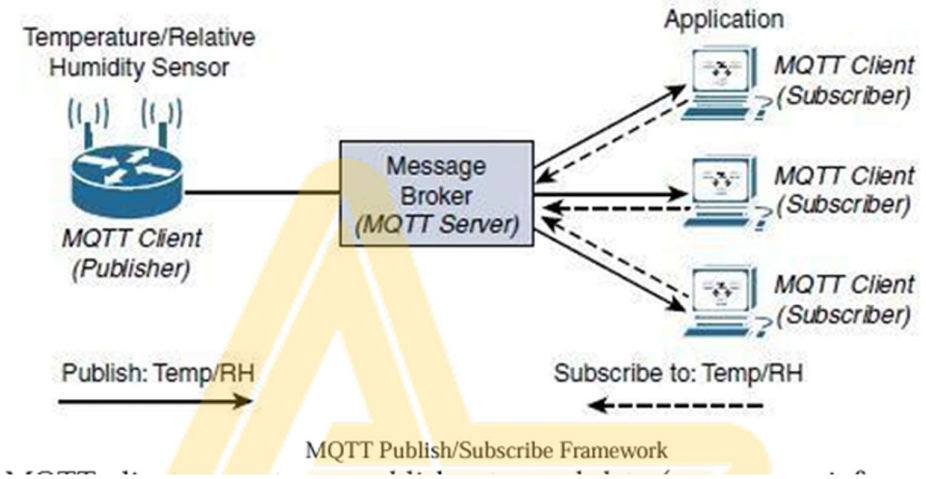
8. Message Queuing Telemetry Transport (MQTT) with respect to following: i) MQTT Publish/Subscribe Framework ii) MQTT Message Format iii) MQTT QoS

i) MQTT Publish/Subscribe Framework

The key points regarding the MQTT publish/subscribe framework are:

1. An MQTT client can act as a publisher to send data or resource information to an MQTT server acting as a message broker.
2. The MQTT server (message broker) accepts network connections and application messages from publishers.
3. The MQTT server handles subscription and unsubscription processes, and pushes application data to MQTT clients acting as subscribers.
4. Clients can subscribe to all data (using wildcards) or specific data from a publisher's information tree.
5. The presence of a message broker in MQTT decouples data transmission between publisher and subscriber clients.
6. This decoupling allows the message broker to buffer and cache information in case of network failures, so publishers and subscribers do not need to be online simultaneously.
7. MQTT control packets run over a TCP transport using port 1883, ensuring an ordered and lossless byte stream between client and server.
8. MQTT is a lightweight protocol, with each control packet consisting of a 2-byte fixed header, optional variable header fields, and an optional payload of up to 256 MB.

In summary, MQTT follows a publish/subscribe model with a central message broker that decouples publishers and subscribers, allowing for asynchronous communication, data buffering, and flexible subscription patterns, all over a lightweight TCP-based protocol.



ii) MQTT Message Format

The key fields in the MQTT message header are:

1. Message Type:

- Identifies the kind of MQTT control packet
- 14 different packet types specified, each with a unique value coded in this field
- Values 0 and 15 are reserved

2. DUP (Duplication Flag):

- When set, allows the client to indicate that the packet has been sent before but not acknowledged

3. QoS (Quality of Service):

- Allows selection of 3 different QoS levels

4. Retain Flag:

- Notifies the server to hold onto (retain) the message data
- Allows new subscribers to instantly receive the last known value

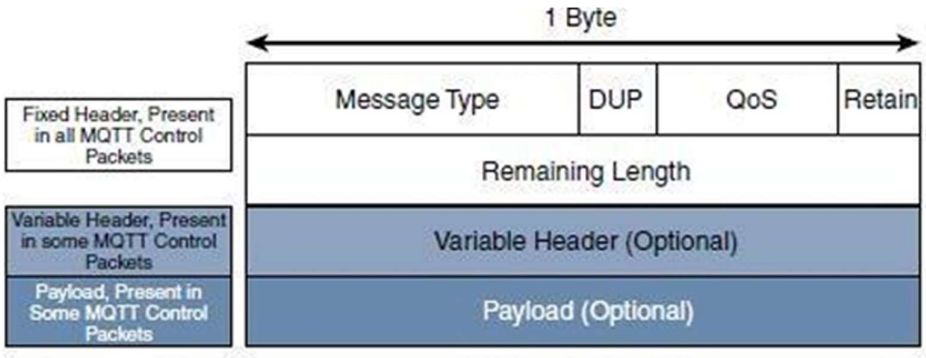
5. Remaining Length:

- Specifies the number of bytes in the MQTT packet following this header field

Other key points:

- MQTT has a smaller 2-byte fixed header compared to 4 bytes in CoAP
- The compact header design contributes to MQTT's lightweight nature
- Various flags and fields enable features like QoS levels, deduplication, and retained messages

So in summary, the MQTT header packs several important pieces of metadata into just 2 bytes, making it highly efficient for constrained IoT environments while still supporting crucial functionality.



MQTT Message Format

iii) MQTT QoS

You have clearly explained the three levels of Quality of Service (QoS) supported by the MQTT protocol:

1. QoS 0 (At most once):

- Best effort, unacknowledged delivery
- Message is sent once by publisher to server, and server sends once to subscribers
- No retries, message may be lost

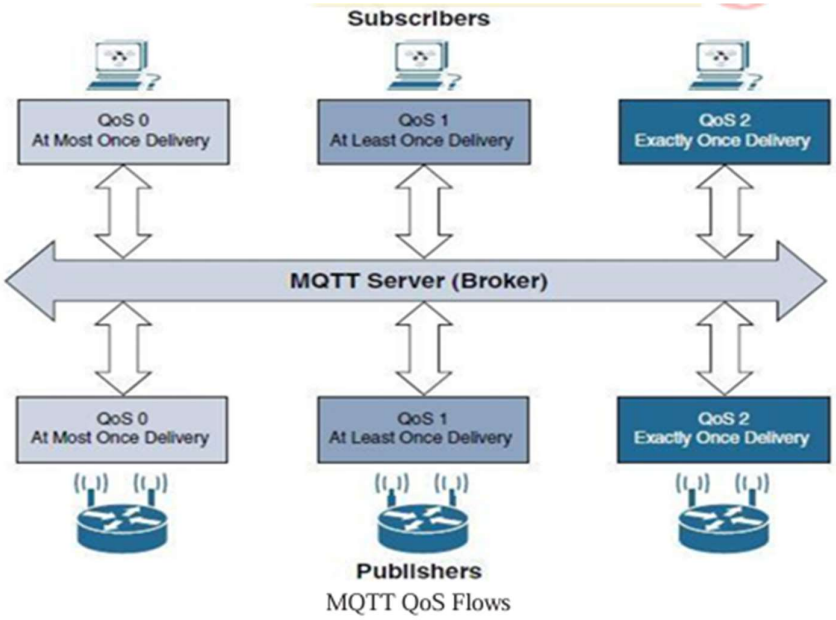
2. QoS 1 (At least once):

- Ensures message is delivered at least once
- Uses PUBLISH and PUBACK packets with packet identifiers
- Message is retried until PUBACK is received
- May result in duplicate deliveries

3. QoS 2 (Exactly once):

- Ensures message is delivered exactly once with no duplicates
- Highest QoS level with most overhead
- Uses a two-step handshake with PUBREC/PUBREL packets in addition to PUBLISH/PUBACK
- Guaranteed delivery, duplicates are dropped

The different QoS levels allow applications to trade off between delivery guarantees and protocol overhead based on their reliability requirements. QoS 0 is fire-and-forget, QoS 1 guarantees at least once delivery, and QoS 2 guarantees exactly once delivery at the cost of increased packet exchanges.



MQTT QoS Flows

12. Discuss the need for optimization of IP in IOT (VTU July 2019)

The key reasons for optimizing IP for IoT networks as mentioned are:

1. Constrained Nodes:

- Different classes of IoT devices coexist with varying resource constraints
- Some very constrained devices may only send a few bytes infrequently, requiring an IP adaptation model with gateways/proxies
- Devices with moderate resources can use a stripped-down IP stack or communicate via gateways
- Relatively powerful devices similar to PCs can use a full IP stack but need to account for bandwidth constraints

2. Constrained Networks:

- Many IoT devices operate over low-power, low-bandwidth links (wireless or wired)
- Link speeds range from just a few kbps to a few hundred kbps
- Packet delivery rates can oscillate, with bursts of errors and intermittent connectivity losses
- Such constrained network conditions create challenges like latency and control plane overhead

3. IP Version Transition:

The key factors related to IPv4 and IPv6 support in IoT solutions are:

1. Application Protocol:

- IoT devices with Ethernet or Wi-Fi interfaces can generally communicate over both IPv4 and IPv6

- However, the specific application protocol being used may dictate the choice of IP version

2. Cellular Provider and Technology:

- For IoT devices using cellular modems, IPv4/IPv6 support depends on the cellular generation and data services offered by the provider

3. Serial Communications:

- Many legacy devices in industries like manufacturing and utilities use serial communications

- Encapsulation of serial protocols over IP is done via raw socket TCP or UDP

- While raw sockets can run over IPv4 or IPv6, current implementations are mostly IPv4-only

4. IPv6 Adaptation Layer:

- Some recently standardized IoT protocols have IPv6-only adaptation layers for specific PHY/MAC layers

- Devices implementing technologies that require these IPv6 adaptation layers must communicate over IPv6-only subnetworks

The transition from IPv4 to IPv6 is driven by IPv4 address exhaustion, with IPv6 providing a much larger address space. The main factors influencing IPv4/IPv6 support are the application protocol requirements, cellular network capabilities, legacy serial communications predominantly using IPv4, and the increasing adoption of IPv6-only adaptation layers in new IoT standards.