UNIT - V

Syllabus: IOT Transport & Session Layer Protocols: Transport Layer (TCP, MPTCP, UDP, DCCP, SCTP)-(TLS, DTLS) – Session Layer-HTTP, CoAP, XMPP, AMQP, MQTT.

The Transport Layer

With the TCP/IP protocol, two main protocols are specified for the transport layer:

1) Transmission Control Protocol (TCP): This connection-oriented protocol requires a session to get established between the source and destination before exchanging data. You can view it as an equivalent to a traditional telephone conversation, in which two phones must be connected and the communication link established before the parties can talk.

2) User Datagram Protocol (UDP): With this connectionless protocol, data can be quickly sent between source and destination—but with no guarantee of delivery. This is analogous to the traditional mail delivery system, in which a letter is mailed to a destination. Confirmation of the reception of this letter does not happen until another letter is sent in response.

With the predominance of human interactions over the Internet, TCP is the main protocol used at the transport layer. This is largely due to its inherent characteristics, such as its ability to transport large volumes of data into smaller sets of packets.

In addition, it ensures reassembly in a correct sequence, flow control and window adjustment, and retransmission of lost packets. These benefits occur with the cost of overhead per packet and per session, potentially impacting overall packet per second performances and latency.

In contrast, UDP is most often used in the context of network services, such as Domain Name System (DNS), Network Time Protocol (NTP), Simple Network Management Protocol (SNMP), and Dynamic Host Control Protocol (DHCP), or for real-time data traffic, including voice and video over IP.

In these cases, performance and latency are more important than packet retransmissions because re-sending a lost voice or video packet does not add value. When the reception of packets must be guaranteed error free, the application layer protocol takes care of that function. When considering the choice of a transport layer by a given IoT application layer protocol, it is recommended to evaluate the impact of this choice on both the lower and upper layers of the stack. For example, most of the industrial application layer protocols are implemented over TCP, while their specifications may offer support for both transport models.

While the use of TCP may not strain generic compute platforms and high-data-rate networks, it can be challenging and is often overkill on constrained IoT devices and networks. This is particularly true when an IoT device needs to send only a few bytes of data per transaction. When using TCP, each packet needs to add a minimum of 20 bytes of TCP overhead, while UDP adds only 8 bytes. TCP also requires the establishment and potential maintenance of an open logical channel.

IoT nodes may also be limited by the intrinsic characteristics of the data link layers. For example, low-power and lossy networks (LLNs). A new IoT application protocol, such as Constrained Application Protocol (CoAP), almost always uses UDP and why implementations of industrial application layer protocols may call for the optimization and adoption of the UDP transport layer if

run over LLNs. For example, the Device Language Message Specification/Companion Specification for Energy Metering (DLMS/COSEM) application layer protocol, a popular protocol for reading smart meters in the utilities space, is the de facto standard in Europe. Adjustments or optimizations to this protocol should be made depending on the IoT transport protocols that are present in the lower layers. For example, if you compare the transport of DLMS/COSEM over a cellular network versus an LLN deployment, you should consider the following:

■ Select TCP for cellular networks because these networks are typically more robust and can handle the overhead. For LLNs, where both the devices and network itself are usually constrained, UDP is a better choice and often mandatory.

■ DLMS/COSEM can reduce the overhead associated with session establishment by offering a "long association" over LLNs. Long association means that sessions stay up once in place because the communications overhead necessary to keep a session established is much less than is involved in opening and closing many sepa- rate sessions over the same time period. Conversely, for cellular networks, a short association better controls the costs by tearing down the open associations after transmitting.

■ When transferring large amounts of DLMS/COSEM data, cellular links are preferred to optimize each open association. Smaller amounts of data can be handled efficiently over LLNs. Because packet loss ratios are generally higher on LLNs than on cellular networks, keeping the data transmission amounts small over LLNs limits the retransmission of large numbers of bytes.

Multicast requirements are also impacted by the protocol selected for the transport layer. With multicast, a single message can be sent to multiple IoT devices. This is useful in the IoT context for upgrading the firmware of many IoT devices at once. Also, keep in mind that multicast utilizes UDP exclusively.

To guarantee interoperability, certification and compliance profiles, such as Wi-SUN, need to specify the stack from Layer 1 to Layer 4. This enables the chosen technology to be compatible with the different options of the stack while also being compatible with IP.

TRANSPORT LAYER

Transport protocols such as TCP make use of this service to provide applications with reliable, in-order, data stream delivery. They use either UDP or TCP as a transport mechanism. Remember that UDP is unreliable and offers no flow-control, so in this case, the application has to provide its own error recovery, flow control, and congestion control functionality. It is often easier to build applications on top of TCP because it is a reliable stream, connection-oriented, congestion-friendly, flow control-enabled protocol. As a result, most application protocols will use TCP, but there are applications built on UDP to achieve better performance through increased protocol efficiencies. Most applications use the client/server model of interaction

TCP (Transmission Control Protocol):

TCP is a layer 4 protocol which provides acknowledgement of the received packets and is also reliable as it resends the lost packets. It is better than UDP but due to these features it has an additional overhead. It is used by application protocols like HTTP and FTP.

UDP (User Datagram Protocol):

UDP is also a layer 4 protocol but unlike TCP it doesn't provide acknowledgement of the sent packets. Therefore, it isn't reliable and depends on the higher layer protocols for the same. But on

the other hand it is simple, scalable and comes with lesser overhead as compared to TCP. It is used in video and voice streaming.

TCP Vs UDP –

Session Multiplexing:

A single host with a single IP address is able to communicate with multiple servers. While using TCP, first a connection must be established between the server and the receiver and the connection is closed when the transfer is completed. TCP also maintains reliability while the transfer is taking place.

UDP on the other hand sends no acknowledgement of receiving the packets. Therefore, provides no reliability.

Segmentation:

Information sent is first broken into smaller chunks for transmission.

Maximum Transmission Unit or MTU of a Fast Ethernet is 1500 bytes whereas the theoretical value of TCP is 65495 bytes. Therefore, data has to be broken into smaller chunks before being sent to the lower layers. MSS or Maximum Segment Size should be set small enough to avoid fragmentation. TCP supports MSS and Path MTU discovery with which the sender and the receiver can automatically determine the maximum transmission capability.

UDP doesn't support this; therefore it depends on the higher layer protocols for data segmentation.

Flow Control:

If sender sends data faster than what receiver can process then the receiver will drop the data and then request for a retransmission, leading to wastage of time and resources. TCP provides end-to-end flow control which is realized using a sliding window. The sliding window sends an acknowledgement from receiver's end regarding the data that the receiver can receive at a time.

UDP doesn't implement flow control and depends on the higher layer protocols for the same.

Connection Oriented:

TCP is connection oriented, i.e., it creates a connection for the transmission to take place, and once the transfer is over that connection is terminated. UDP on the other hand is connectionless just like IP (Internet Protocol).

Reliability:

TCP sends an acknowledgement when it receives a packet. It requests a retransmission in case a packet is lost.

UDP relies on the higher layer protocols for the same.

Headers: The size of TCP header is 20-bytes (16-bits for source port, 16-bits for the destination port, 32-bits for seq number, 32-bits for ack number, 4-bits header length).The size of the UDP header is 8-bytes (16-bits for source port, 16-bits for destination port, 16-bits for length, 16-bits for checksum); it's significantly smaller than the TCP header. Both UDP and TCP header is comprised of 16-bit Source port (these are used for identifying the port number of the source) fields and 16-bits destination port (these are used for specifying the offered application) fields.

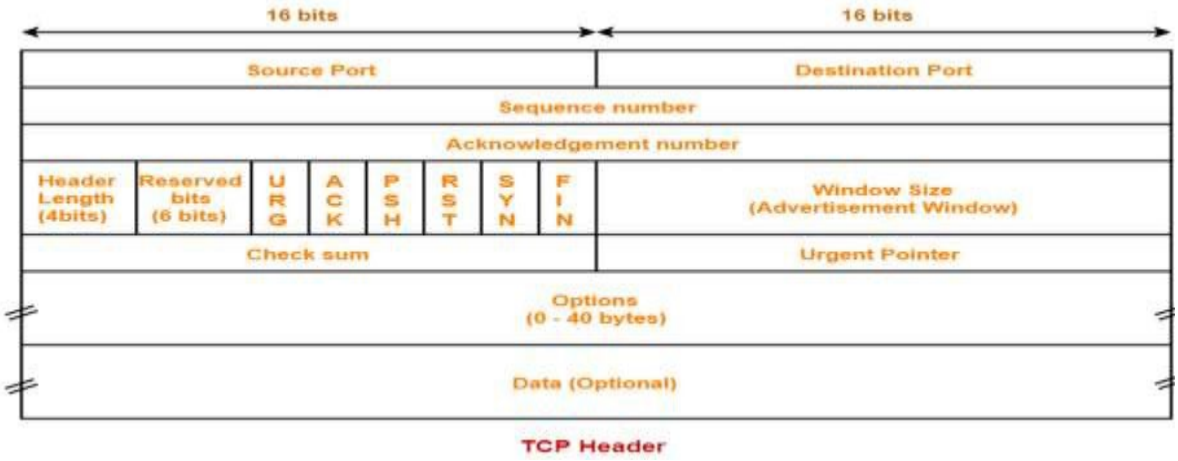


Fig : TCP header

Source port: this is a 16 bit field that specifies the port number of the sender. Destination port: this is a 16 bit field that specifies the port number of the receiver.

Sequence number: the sequence number is a 32 bit field that indicates how much data is sent during the TCP session. When you establish a new TCP connection (3 way handshake) then the initial sequence number is a random 32 bit value. The receiver will use this sequence number and sends back an acknowledgment. Protocol analyzers like wireshark will often use a relative sequence number of 0 since it's easier to read than some high random number.

Acknowledgment number: this 32 bit field is used by the receiver to request the next TCP segment. This value will be the sequence number incremented by 1.

DO: this is the 4 bit data offset field, also known as the header length. It indicates the length of the TCP header so that we know where the actual data begins.

RSV: these are 3 bits for the reserved field. They are unused and are always set to 0.

Flags: there are 9 bits for flags, we also call them control bits. We use them to establish connections, send data and terminate connections:

URG: urgent pointer. When this bit is set, the data should be treated as priority over other data.

ACK: used for the acknowledgment.

PSH: this is the push function. This tells an application that the data should be transmitted immediately and that we don't want to wait to fill the entire TCP segment.

RST: this resets the connection, when you receive this you have to terminate the connection right away. This is only used when there are unrecoverable errors and it's not a normal way to finish the TCP connection.

SYN: we use this for the initial three way handshake and it's used to set the initial sequence number.

FIN: this finish bit is used to end the TCP connection. TCP is full duplex so both parties will have to use the FIN bit to end the connection. This is the normal method how we end an connection.

Window: the 16 bit window field specifies how many bytes the receiver is willing to receive. It is used so the receiver can tell the sender that it would like to receive more data than what it is currently receiving. It does so by specifying the number of bytes beyond the sequence number in the acknowledgment field.

Checksum: 16 bits are used for a checksum to check if the TCP header is OK or not.

Urgent pointer: these 16 bits are used when the URG bit has been set, the urgent pointer is used to indicate where the urgent data ends.
Options: this field is optional and can be anywhere between 0 and 320 bits.

UDP protocol

In computer networking, the UDP stands for User Datagram Protocol. The David P. Reed developed the UDP protocol in 1980. It is defined in RFC 768, and it is a part of the TCP/IP protocol, so it is a standard protocol over the internet.

The UDP protocol allows the computer applications to send the messages in the form of datagram's from one machine to another machine over the Internet Protocol (IP) network. The UDP is an alternative communication protocol to the TCP protocol (transmission control protocol). Like TCP, UDP provides a set of rules that governs how the data should be exchanged over the internet. The UDP works by encapsulating the data into the packet and providing its own header information to the packet. Then, this UDP packet is encapsulated to the IP packet and sent off to its destination. Both the TCP and UDP protocols send the data over the internet protocol network, so it is also known as TCP/IP and UDP/IP. There are many differences between these two protocols. UDP enables the process-to-process communication, whereas the TCP provides host to host communication. Since UDP sends the messages in the form of datagrams, it is considered the best-effort mode of communication. TCP sends the individual packets, so it is a reliable transport medium. Another difference is that the TCP is a connection-oriented protocol whereas, the UDP is a connectionless protocol as it does not require any virtual circuit to transfer the data. UDP also provides a different port number to distinguish different user requests and also provides the checksum capability to verify whether the complete data has arrived or not; the IP layer does not provide these two services. The User Datagram Protocol (UDP) is simplest Transport Layer communication protocol available of the TCP/IP protocol suite. It involves minimum amount of communication mechanism. UDP is said to be an unreliable transport protocol but it uses IP services which provides best effort delivery mechanism. In UDP, the receiver does not generate an acknowledgement of packet received and in turn, the sender does not wait for any acknowledgement of packet sent. This shortcoming makes this protocol unreliable as well as easier on processing.

Requirement of UDP

Deploy UDP where the acknowledgement packets share significant amount of bandwidth along with the actual data. For example, in case of video streaming, thousands of packets are forwarded towards its users. Acknowledging all the packets is troublesome and may contain huge amount of bandwidth wastage. The best delivery mechanism of underlying IP protocol ensures best efforts to deliver its packets, but even if some packets in video streaming get lost, the impact is not calamitous and can be ignored easily. Loss of few packets in video and voice traffic sometimes goes unnoticed.

Features

- UDP is used when acknowledgement of data does not hold any significance.
- UDP is good protocol for data flowing in one direction.
- UDP is simple and suitable for query based communications.
- UDP is not connection oriented.

- UDP does not provide congestion control mechanism.
- UDP does not guarantee ordered delivery of data.
- UDP is stateless.
- UDP is suitable protocol for streaming applications such as VoIP, multimedia streaming.

The fields in a UDP header are:

Source port – The port of the device sending the data. This field can be set to zero if the destination computer doesn't need to reply to the sender.

Destination port – The port of the device receiving the data. UDP port numbers can be between 0 and 65,535.

Length – Specifies the number of bytes comprising the UDP header and the UDP payload data. The limit for the UDP length field is determined by the underlying IP protocol used to transmit the data.

Checksum – The checksum allows the receiving device to verify the integrity of the packet header and payload.

It is optional in IPv4 but was made mandatory in IPv6.

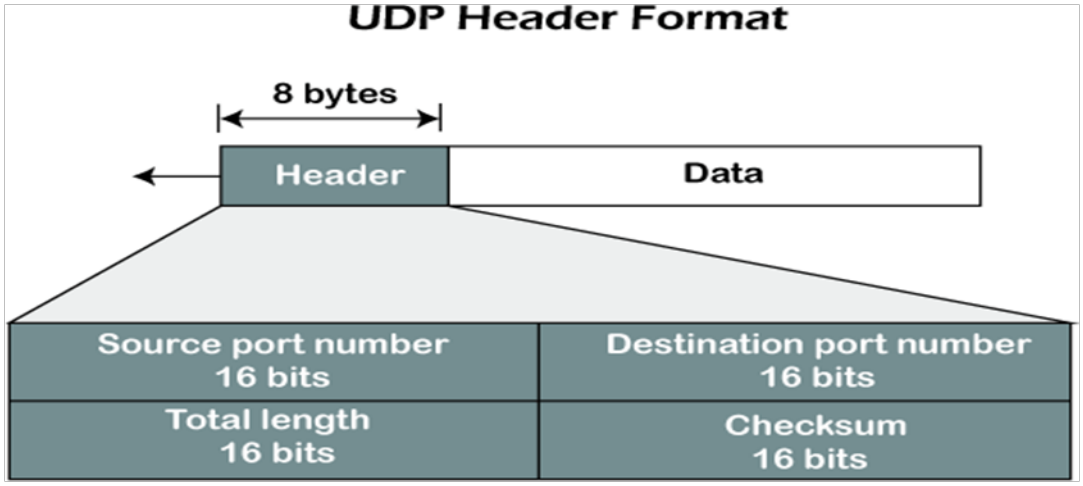
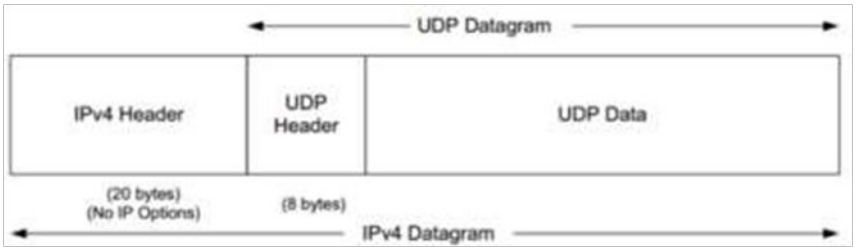


Fig : UDP header

The following figure shows the encapsulation of a UDP datagram as a single IPv4 datagram. The IPv6 encapsulation is similar, but other details differ slightly.

The IPv4 Protocol field has the value 17 to indicate UDP. IPv6 uses the same value (17) in the Next Header field.



UDP application

Here are few applications where UDP is used to transmit data:

- Domain Name Services
- Simple Network Management Protocol

- Trivial File Transfer Protocol
- Routing Information Protocol
- Kerberos

	TCP	UDP
Full form	It stands for **Transmission Control Protocol**.	It stands for **User Datagram Protocol**.
Type of connection	It is a connection-oriented protocol, which means that the connection needs to be established before the data is transmitted over the network.	It is a connectionless protocol, which means that it sends the data without checking whether the system is ready to receive or not.
Reliable	TCP is a reliable protocol as it provides assurance for the delivery of data packets.	UDP is an unreliable protocol as it does not take the guarantee for the delivery of packets.
Speed	TCP is slower than UDP as it performs error checking, flow control, and provides assurance for the delivery of	UDP is faster than TCP as it does not guarantee the delivery of data packets.
Header size	The size of TCP is 20 bytes.	The size of the UDP is 8 bytes.
Acknowledgment	TCP uses the three-way-handshake concept. In this concept, if the sender receives the ACK, then the sender will send the data. TCP also has the ability to resend the lost data.	UDP does not wait for any acknowledgment; it just sends the data.
Flow control mechanism	It follows the flow control mechanism in which too many packets cannot be sent to the receiver at the same time.	This protocol follows no such mechanism.
Error checking	TCP performs error checking by using a checksum. When the data is corrected, then the data is retransmitted to the receiver.	It does not perform any error checking, and also does not resend the lost data packets.
Applications	This protocol is mainly used where a secure and reliable communication process is required, like military services, web browsing, and e-mail.	This protocol is used where fast communication is required and does not care about the reliability like VoIP, game streaming, video and music streaming, etc.

MPTCP (Multipath TCP):

Goals for MPTCP are:

• It should be capable of using multiple network paths for a single connection.

• It must be able to use the available network paths at least as well as regular TCP, but without starving TCP.

• It must be as usable as regular TCP for existing applications.

• Enabling MPTCP must not prevent connectivity on a path where regular TCP works.

MPTCP manages the underlying TCP connections (called subflows6) that are used to carry the actual data. From an architectural viewpoint, MPTCP acts as a shim layer between the socket interface and one or more TCP subflows, as shown in figure 1. MPTCP requires additional signaling between the end hosts.

It achieves this by using TCP options to achieve the following goals:

- Establish a new MPTCP connection.
- Add subflows to an MPTCP connection.
- Transmit data on the MPTCP connection.

An MPTCP connection is established by using the three-way handshake with TCP options to negotiate its usage. The MP_CAPABLE option in the SYN segment indicates that the client supports MPTCP. This option also contains a random key used for security purposes. If the server supports MPTCP, then it replies with a SYN+ACK segment that also contains the MP_CAPABLE option. This option contains a random key chosen by the server. The third ACK of the three-way handshake also includes the MP_CAPABLE option to confirm the utilization of MPTCP and the keys to enable stateless servers.

The three-way handshake shown in figure 2 creates the first TCP subflow over one interface. To use another interface, MPTCP uses a three-way handshake to establish one subflow over this interface. Adding a subflow to an existing MPTCP connection requires the corresponding MPTCP connection to be uniquely identified on each end host. With regular TCP, a TCP connection is always identified by using the tuple. Unfortunately, because NAT is present, the addresses and port numbers that are used on the client may not be the same as those exposed to the server. Although on each host the 4-tuple is a unique local identification of each TCP connection, this identification is not globally unique.

MPTCP needs to be able to link each subflow to an existing MPTCP connection. For this, MPTCP assigns a locally unique token to each connection. When a new subflow is added to an existing MPTCP connection, the MP_JOIN option of the SYN segment contains the token of the associated MPTCP connection. This is illustrated in figure 3. The astute reader may have noticed that the MP_CAPABLE option does not contain a token. Still, the token is required to enable the establishment of subflows. To reduce the length of the MP_CAPABLE option and avoid using all the limited TCP options space (40 bytes) in the SYN segment, MPTCP derives the token as the result of a truncated hash of the key. The second function of the MP_JOIN option is to authenticate the addition of the subflow. For this, the client and the server exchange random nonces, and each host computes an HMAC (hash- based message authentication code) over the random nonce chosen by the other host and the keys exchanged during the initial handshake. Now that the subflows have been established, MPTCP can use them to exchange data. Each host can send data over any of the established subflows. Furthermore, data transmitted over one subflow can be retransmitted on another to recover from losses. This is achieved by using two levels of sequence numbers.6 The regular TCP sequence number ensures that data is received in order over each subflow and allows losses to be detected. MPTCP uses the data sequence number to reorder the data received over different subflows before passing it to the application.

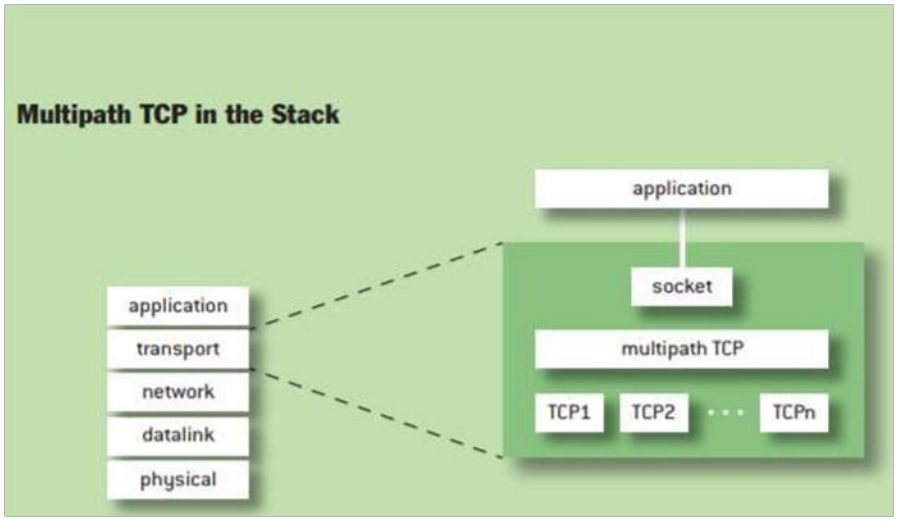


Fig 3: Multipath TCP in stack

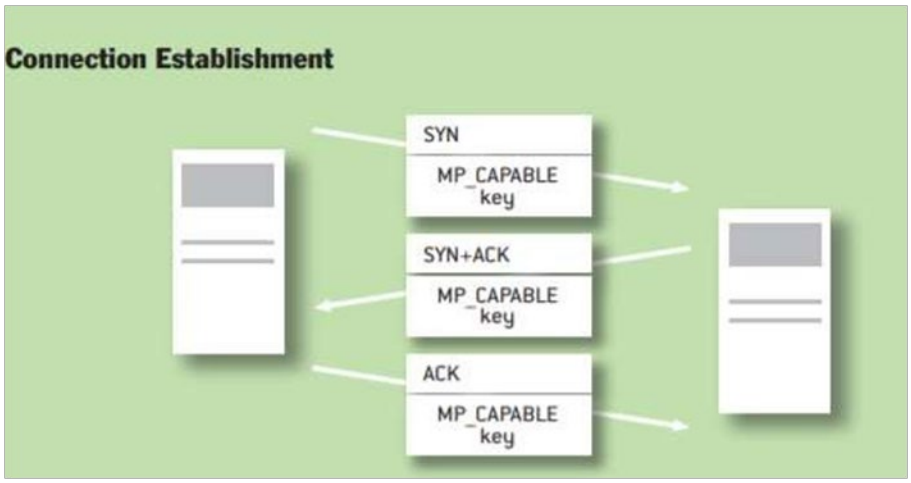


Fig 4: Connection Establishment

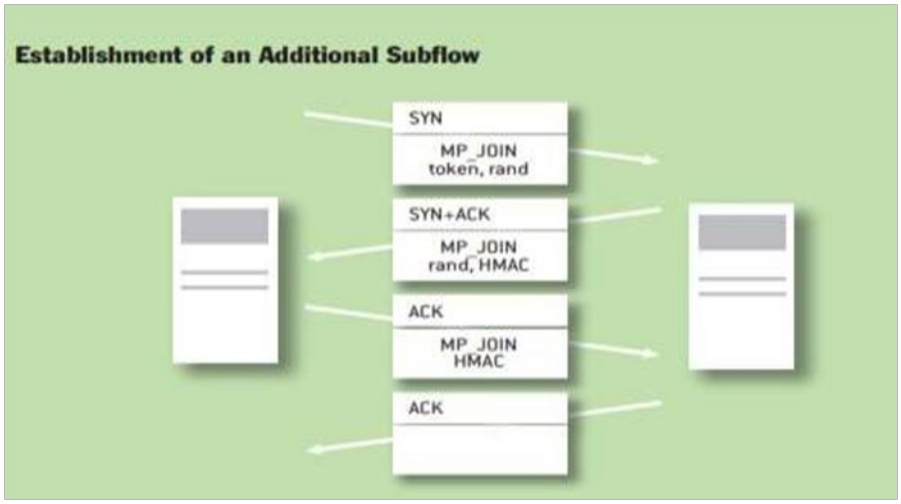


Fig 5: Establishment of an additional subflow

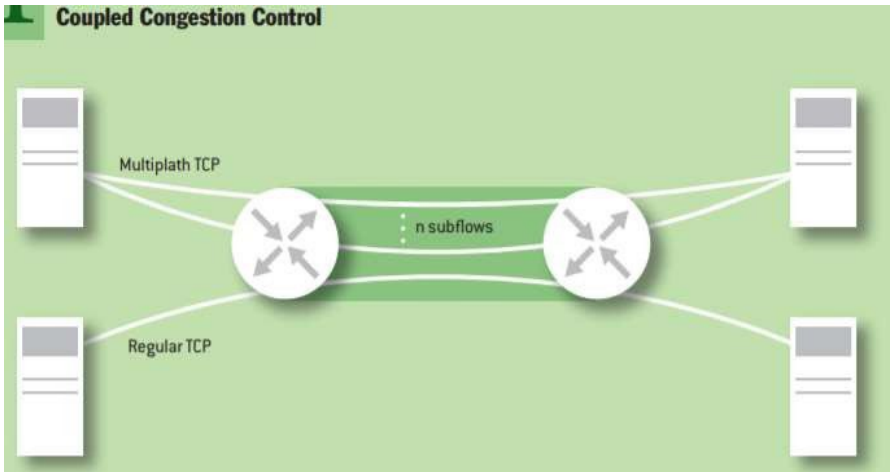


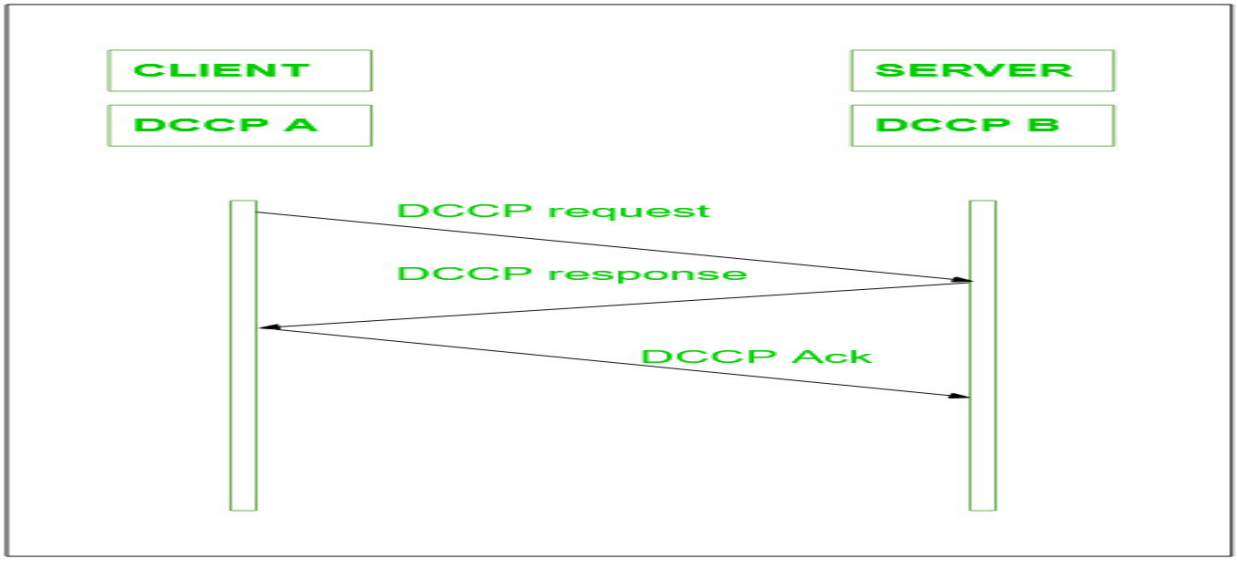
Fig: Coupled congestion control

From a congestion-control viewpoint, using several subflows for one connection leads to an interesting problem. With regular TCP, congestion occurs on one path between the sender and the receiver. MPTCP uses several paths, and two paths will typically experience different levels of congestion. A naive solution to the congestion problem in MPTCP would be to use the standard TCP congestion-control scheme on each subflow. This can be easily implemented but leads to unfairness with regular TCP. In the network depicted in figure 4, two clients share the same bottleneck link. If the MPTCP-enabled client uses two subflows, then it will obtain two-thirds of the shared bottleneck. SYN MP JOIN token, rand SYN+ACK MP JOIN rand, HMAC ACK ACK MP JOIN HMAC Establishment of an Additional Subflow n subows Multiplath TCP Regular TCP Coupled Congestion Control NETWORKS 5 This is unfair because if this client used regular TCP, it would obtain only half of the shared bottleneck.

DCCP:
DCCP provides an unreliable transport service, similar to UDP, but with adaptive congestion control, similar to TCP and Stream Control Transmission Protocol (SCTP). DCCP can be viewed equally well as either UDP- plus-congestion-control or TCP-minus-reliability (although, unlike TCP, DCCP offers multiple congestion control algorithms)
DCCP is basically a message-based transport-level protocol. The setting of a secure connection is easily maintained using it, its closure i.e. ECN (Explicit Congestion Notification), congestion control, and negotiation of features. DCCP is a great technique to access congestion control mechanisms, also we don't need to implement them at the application level also.

DCCP basically allows similar Transfer Control Protocol feeds also, but delivery in the order of transmission cannot be done. Sequential delivery of multiple streams (as in SCTP- Stream Control Transmission Protocol) is not available in DCCP.

DCCP is widely used in applications package delivery is composed of time constraints. The examples that come under this category include multiplayer online games, internet telephony, streaming media (video, audio), etc. The most important feature of these applications is that old messages quickly become expired automatically, lose their usefulness by default.
DCCP connection setup can be explained through the below image, it is basically similar to TCP connection setup :



Features of DCCP:
DCCP is a non-reliable datagram stream, with a good feature of confirmation.
DCCP helps to secure negotiation of options, including negotiation of the most suitable mechanism for congestion control.
It provides a secure handshake protocol with the purpose of initializing and closing the connection of DCCP.
It plays a vital role in the discovery of the maximum transmitting unit on the chosen path by the user.

It provides techniques that allow servers to avoid storing states for attempted unconnected, unconfirmed disconnections, or for already closed connections as well.
Confirmation mechanisms are a very good feature of DCCP which helps to communicate packet loss and ECN information.
Optional mechanisms are also some good techniques that communicate to the emitting application with high security, which packets have reached the receiver and which are not, also whether they have been marked by ECN or not, or they are corrupted or removed in the receiver buffer.
DCCP can support multiple concurrent streams within a single connection, which enables applications to transmit multiple data flows over the same connection.
It provides a mechanism for applications to prioritize their data flows, which helps in achieving better Quality of Service (QoS).
DCCP supports both connection-oriented and connectionless communication modes.
It offers a congestion control mechanism that is more flexible than the TCP congestion control mechanism.
DCCP can be used over both IP version 4 and IP version 6 networks.

Advantages

Congestion control: Unlike UDP, which has no built-in mechanism for controlling congestion, DCCP includes congestion control algorithms that help to prevent network overload and ensure reliable delivery of data.
Quality of Service (QoS) support: DCCP provides support for QoS, which allows network administrators to prioritize different types of traffic based on their importance. This can be useful for applications such as video streaming or voice over IP, where low latency and high reliability are essential.
Flexibility: DCCP is designed to be a flexible protocol, allowing network administrators to choose from a variety of congestion control algorithms based on the specific requirements of their network and applications.
Compatibility: DCCP is designed to work with existing IP networks and is compatible with traditional IP protocols like TCP and UDP.
Real-World Applications
Streaming media: DCCP is often used in streaming media applications, such as video conferencing, where low latency and high reliability are important. The congestion control algorithms built into DCCP help to ensure that these applications can run smoothly and effectively, even in networks with high levels of congestion.
Gaming: Some online gaming applications also make use of DCCP, as its congestion control algorithms can help to prevent network slowdowns and ensure that game data is delivered quickly and reliably.
Telemetry: DCCP is also used in telemetry applications, where large amounts of data need to be transmitted from remote devices back to a central control center. The congestion control algorithms built into DCCP help to ensure that this data is delivered reliably and efficiently, even in congested networks.
Remote Access: DCCP can be used for remote access applications, as it provides a reliable and secure connection for remote access to servers and other resources.

DTLS

Datagram Transport Layer Security (DTLS) is a protocol used to secure datagram-based communications. It's based on the stream-focused Transport Layer Security (TLS), providing a similar level of security. ...

However, DTLS gains the benefits of datagram protocols, too; in particular, the lower overhead and reduced latency

DTLS over DCCP

The approach here is very straightforward -- DTLS records are transmitted in the Application Data fields of DCCP-Data and DCCP-DataAck packets (in the rest of the document assume that "DCCP-Data packet" means "DCCP-Data or DCCP-DataAck packet"). Multiple DTLS records MAY be sent in one DCCP-Data packet, as long as the resulting packet is within the Path Maximum Transfer Unit (PMTU) currently in force for normal data packets, if fragmentation is not allowed (the Don't Fragment (DF) bit is set for IPv4 or no fragmentation extension headers are being used for IPv6), or within the current DCCP maximum packet size if fragmentation is allowed. A single DTLS record MUST be fully contained in a single DCCP-Data packet; it MUST NOT be split over multiple packets.

DCCP and DTLS Sequence Numbers

Both DCCP and DTLS use sequence numbers in their packets/records. These sequence numbers serve somewhat, but not completely, overlapping functions. Consequently, there is no connection between the sequence number of a DCCP packet and the sequence number in a DTLS record contained in that packet, and there is no connection between sequence number-related features such as DCCP synchronization.

DCCP and DTLS Connection Handshakes

Unlike UDP, DCCP is connection-oriented, and has a connection handshake procedure that precedes the transmission of DCCP-Data and DCCP-DataAck packets. DTLS is also connection-oriented, and has a handshake procedure of its own that must precede the transmission of actual application information. Using the rule of mapping DTLS records to DCCP-Data and DCCP-DataAck packets, the two handshakes are forced to happen in series, with the DCCP handshake first, followed by the DTLS handshake. This is how TLS over TCP works. However, the DCCP handshake packets DCCP-Request and DCCP-Response have Application Data fields and can carry user data during the DCCP handshake, and this creates the opportunity to perform the handshakes partially in parallel. DTLS client implementations MAY choose to transmit one or more DTLS records (typically containing DTLS handshake messages or parts of them) in the DCCP-Request packet. A DTLS server implementation MAY choose to process these records as usual, and if it has one or more DTLS records to send as a response (typically containing DTLS handshake messages or parts of them), it MAY include those records in the DCCP-Response packet. DTLS servers MAY also choose to delay the response until the DCCP handshake completes and then send the DTLS response in a DCCP-Data packet.

Note that even though the DCCP handshake is a reliable process (DCCP handshake messages are retransmitted as required if messages are lost), the transfer of Application Data in DCCP-Request and DCCP-Response packets is not necessarily reliable. For example, DCCP server implementations are free to discard Application Data received in DCCP-Request packets. And if DCCP-Request or

DCCP-Response packets need to be retransmitted, the DCCP implementation may choose to not include the Application Data present in the initial message.

SCTP Requirements

Number of Inbound and Outbound Streams: An association between the endpoints A and Z provides n streams from A to Z and m streams from Z to A. A pair consisting of two streams with the same stream identifier is considered and used as one bi-directional stream. Thus an SCTP association can be considered as a set of min(n,m) bi- directional streams and (max(n,m) - min(n,m)) uni-directional streams.

Fragmentation of User Messages

To avoid the knowledge and handling of the MTU inside TLS, SCTP MUST provide fragmentation of user messages, which is an optional feature of [RFC2960]. Since SCTP is a message oriented protocol, it must be able to transmit all TLS records as SCTP user messages. Thus the supported maximum length of SCTP user messages MUST be at least 2^14+ 2048 + 5 = 18437 bytes, which is the maximum length of a TLS Ciphertext, as defined in [RFC2246]. Note that an SCTP implementation might need to support the partial delivery API to be able to support the transport of user messages of this size. Therefore, SCTP takes care of fragmenting and reassembling the TLS records in order to avoid IP-fragmentation.

Features and Services	DCCP	SCTP	UDP
Connection Oriented	Yes	Yes	No
Un-ordered Data Delivery	Yes	Yes	Yes
Ordered Data Delivery	No	Yes	No
Reliable	No	Yes	No
Congestion Control	Yes	Yes	No
Flow Control	Optional	Yes	No
Multi-streaming	No	Yes	No
Multi-homing	No	Yes	No

Comparison of DCTP,SCTP,UDP

Session Layer Protocols

HTTP:

HTTP stands for HyperText Transfer Protocol.

It is a protocol used to access the data on the World Wide Web (www).

The HTTP protocol can be used to transfer the data in the form of plain text, hypertext, audio, video, and so on.
This protocol is known as HyperText Transfer Protocol because of its efficiency that allows us to use in a hypertext environment where there are rapid jumps from one document to another document.
HTTP is similar to the FTP as it also transfers the files from one host to another host. But, HTTP is simpler than FTP as HTTP uses only one connection, i.e., no control connection to transfer the files.
HTTP is used to carry the data in the form of MIME-like format.
HTTP is similar to SMTP as the data is transferred between client and server.
The HTTP differs from the SMTP in the way the messages are sent from the client to the server and from server to the client. SMTP messages are stored and forwarded while HTTP messages are delivered immediately.
HTTP represents "Hypertext Transfer Protocol." HTTP is the protocol that can transfer information over the network. It is the Internet protocol suite method and defines commands and functions used for sharing web page data.

HTTP uses a server-client model. A client, for example, maybe a laptop or telephone device. The HTTP server is frequently a web host running web server software, such as Apache or IIS.

HTTP also represents commands such as GET and POST, which are used to handle submissions on websites. The CONNECT command can act as a fast connection that is encrypted using a secure socket layer (SSL).

HTTP is equivalent to SMTP as the data is transferred between client and server. The HTTP differs from the SMTP in how the messages are sent from the client to the server and from the server to the client. SMTP messages are saved and advanced, while HTTP messages are delivered directly.

Features
The features of HTTP are as follows −

1) **Connectionless protocol**

HTTP is a connectionless protocol. HTTP user initiates a request and waits for a response from the server. When the server gets the request, the server processes the request and sends back the response to the HTTP user, after which the client disconnects the connection.

2) **Media independent**

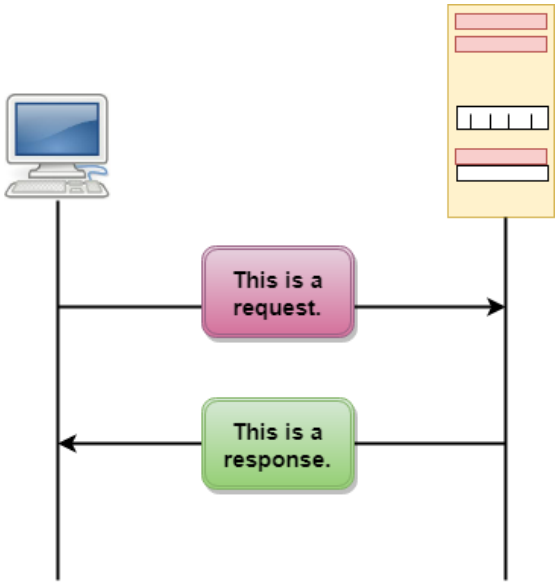
HTTP protocol is media independent as data can be transmitted as long as both the user and server know how to manage the data content. It is necessary for both the user and server to specify the content type in the MIME-type header.

3) **Stateless**

It is a stateless protocol as both the client and server learn each other only during the current request. In HTTP every client connection opens a new session that sends its request the stateless nature keeps the protocol very simple and straightforward. This consumes very few resources on

the server and can support more simultaneous users since there are no client information overheads to be maintained throughout the sessions.

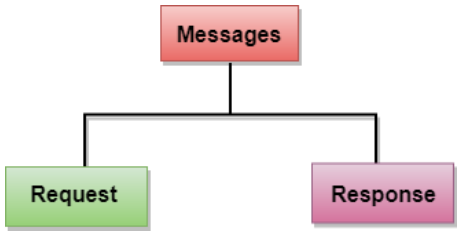
Most of the IP applications, including IoT applications use TCP or UDP for transport. However, there are several message distribution functions that are common among many IoT applications; it is desirable that these functions be implemented in an interoperable standard ways by different applications. These are the so called “Session Layer” protocols.



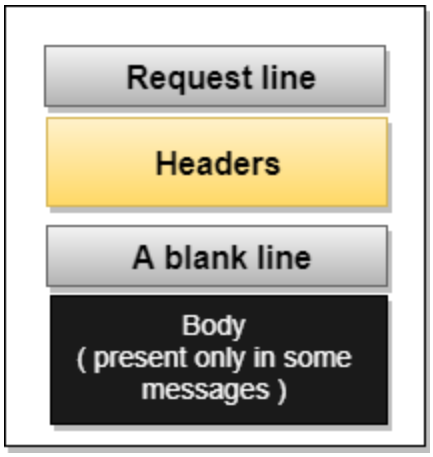
The above figure shows the HTTP transaction between client and server. The client initiates a transaction by sending a request message to the server. The server replies to the request message by sending a response message.

Messages

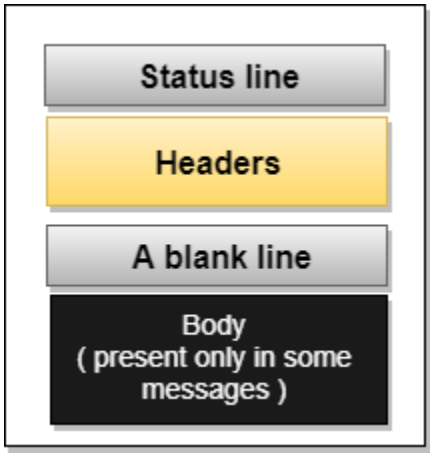
HTTP messages are of two types: request and response. Both the message types follow the same message format.



Request Message: The request message is sent by the client that consists of a request line, headers, and sometimes a body.



Response Message: The response message is sent by the server to the client that consists of a status line, headers, and sometimes a body.



MQTT

MQTT stands for Message Queuing Telemetry Transport. MQTT is a machine to machine internet of things connectivity protocol. It is an extremely lightweight and publish-subscribe messaging transport protocol. This protocol is useful for the connection with the remote location where the bandwidth is a premium. These characteristics make it useful in various situations, including constant environment such as for communication machine to machine and internet of things contexts. It is a publish and subscribe system where we can publish and receive the messages as a client. It makes it easy for communication between multiple devices. It is a simple messaging protocol designed for the constrained devices and with low bandwidth, so it's a perfect solution for the internet of things applications.

Characteristics of MQTT

The MQTT has some unique features which are hardly found in other protocols. Some of the features of an MQTT are given below:

It is a machine to machine protocol, i.e., it provides communication between the devices.

It is designed as a simple and lightweight messaging protocol that uses a publish/subscribe system to exchange the information between the client and the server.

It does not require that both the client and the server establish a connection at the same time.

It provides faster data transmission, like how WhatsApp/messenger provides a faster delivery. It's a real-time messaging protocol.
It allows the clients to subscribe to the narrow selection of topics so that they can receive the information they are looking for.

MQTT Architecture

To understand the MQTT architecture, we first look at the components of the MQTT.
1 Message
2 Client
3 Server or Broker
4 TOPIC

Message

The message is the data that is carried out by the protocol across the network for the application. When the message is transmitted over the network, then the message contains the following parameters:
Payload data
Quality of Service (QoS)
Collection of Properties
Topic Name

Client

In MQTT, the subscriber and publisher are the two roles of a client. The clients subscribe to the topics to publish and receive messages. In simple words, we can say that if any program or device uses an MQTT, then that device is referred to as a client. A device is a client if it opens the network connection to the server, publishes messages that other clients want to see, subscribes to the messages that it is interested in receiving, unsubscribes to the messages that it is not interested in receiving, and closes the network connection to the server.
In MQTT, the client performs two operations:
In MQTT, the client performs two operations:
Publish: When the client sends the data to the server, then we call this operation as a publish.
Subscribe: When the client receives the data from the server, then we call this operation a subscription.

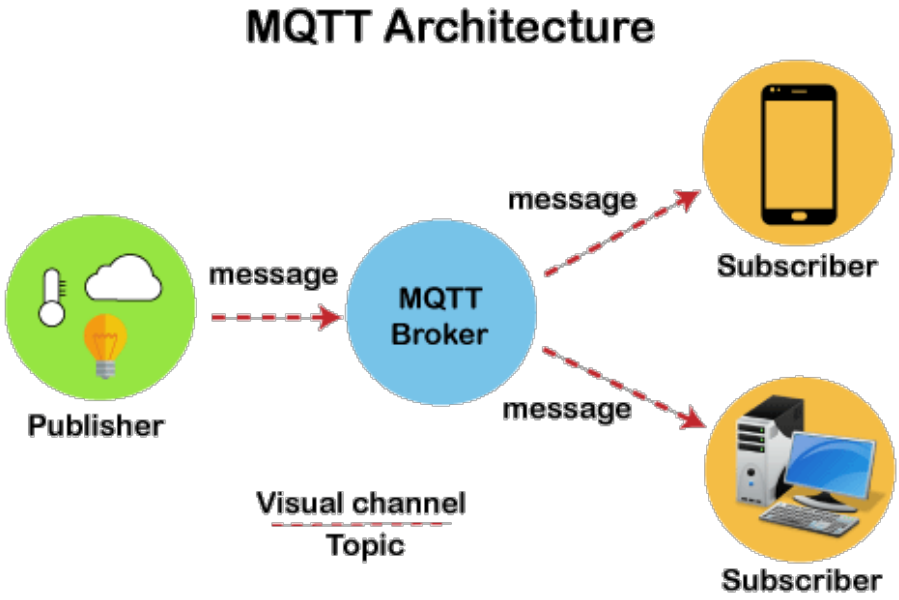
Server

The device or a program that allows the client to publish the messages and subscribe to the messages. A server accepts the network connection from the client, accepts the messages from the client, processes the subscribe and unsubscribe requests, forwards the application messages to the client, and closes the network connection from the client.

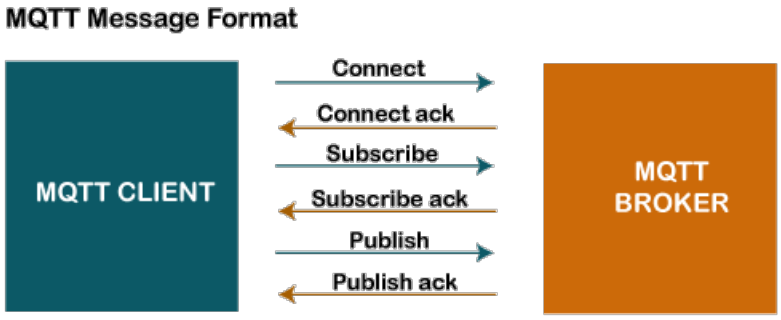
TOPIC

The label provided to the message is checked against the subscription known by the server is known as TOPIC.

Architecture of MQTT

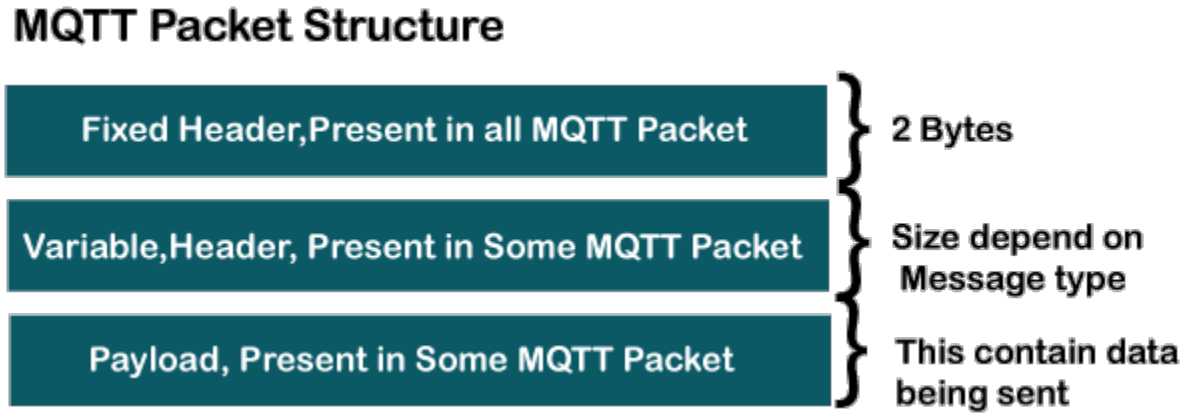


Now we will look at the architecture of MQTT. To understand it more clearly, we will look at the example. Suppose a device has a temperature sensor and wants to send the rating to the server or the broker. If the phone or desktop application wishes to receive this temperature value on the other side, then there will be two things that happened. The publisher first defines the topic; for example, the temperature then publishes the message, i.e., the temperature's value. After publishing the message, the phone or the desktop application on the other side will subscribe to the topic, i.e., temperature and then receive the published message, i.e., the value of the temperature. The server or the broker's role is to deliver the published message to the phone or the desktop application.



The MQTT uses the command and the command acknowledgment format, which means that each command has an associated acknowledgment. As shown in the above figure that the connect command has connect acknowledgment, subscribe command has subscribe acknowledgment, and publish command has publish acknowledgment. This mechanism is similar to the handshaking mechanism as in TCP protocol.

Now we will look at the packet structure or message format of the MQTT.



The MQTT message format consists of 2 bytes fixed header, which is present in all the MQTT packets. The second field is a variable header, which is not always present. The third field is a payload, which is also not always present. The payload field basically contains the data which is being sent. We might think that the payload is a compulsory field, but it does not happen. Some commands do not use the payload field, for example, disconnect message.

SMQTT

An extension of MQTT is Secure MQTT (SMQTT) which uses encryption based on lightweight attribute based encryption. The main advantage of using such encryption is the broadcast encryption feature, in which one message is encrypted and delivered to multiple other nodes, which is quite common in IoT applications. In general, the algorithm consists of four main stages: setup, encryption, publish and decryption. In the setup phase, the subscribers and publishers register themselves to the broker and get a master secret key according to their developer's choice of key generation algorithm. Then, when the data is published, it is encrypted, published by the broker which sends it to the subscribers and finally decrypted at the subscribers which have the same master secret key. The key generation and encryption algorithms are not standardized. SMQTT is proposed only to enhance MQTT security feature

AMQP

The Advanced Message Queuing Protocol (AMQP) is another session layer protocol that was designed for financial industry. It runs over TCP and provides a publish/ subscribe architecture which is similar to that of MQTT. The difference is that the broker is divided into two main components: exchange and queues, as shown in Figure 6. The exchange is responsible for receiving publisher messages and distributing them to queues based on pre-defined roles and conditions.

Queues basically represent the topics and subscribed by subscribers which will get the sensory data whenever they are available in the queue

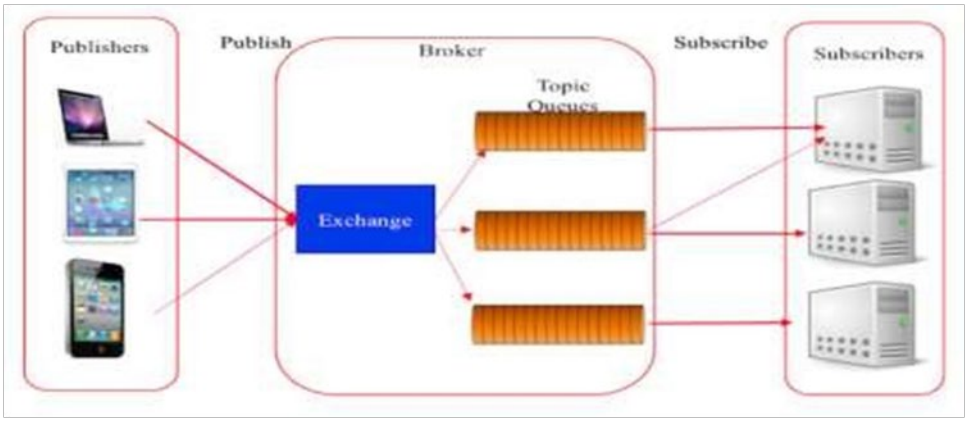


Fig : AMQP

CoAP

The Constrained Application Protocol (CoAP) is another session layer protocol designed by IETF Constrained RESTful Environment (Core) working group to provide lightweight RESTful (HTTP) interface. Representational State Transfer (REST) is the standard interface between HTTP client and servers. However, for lightweight applications such as IoT, REST could result in significant overhead and power consumption. CoAP is designed to enable low-power sensors to use RESTful services while meeting their power constrains. It is built over UDP, instead of TCP commonly used in HTTP and has a light mechanism to provide reliability. CoAP architecture is divided into two main sublayers: messaging and request/response.

The messaging sublayer is responsible for reliability and duplication of messages while the request/response sublayer is responsible for communication. As shown in Figure 7 , CoAP has four messaging modes: confirmable, non- confirmable, piggyback and separate. Confirmable and non-confirmable modes represent the reliable and unreliable transmissions, respectively while the other modes are used for request/response.

Piggyback is used for client/server direct communication where the server sends its response directly after receiving the message, i.e., within the acknowledgment message. On the other hand, the separate mode is used when the server response comes in a message separate from the acknowledgment, and may take some time to be sent by the server. As in HTTP, CoAP utilizes GET, PUT, PUSH, DELETE messages requests to retrieve, create, update, and delete, respectively

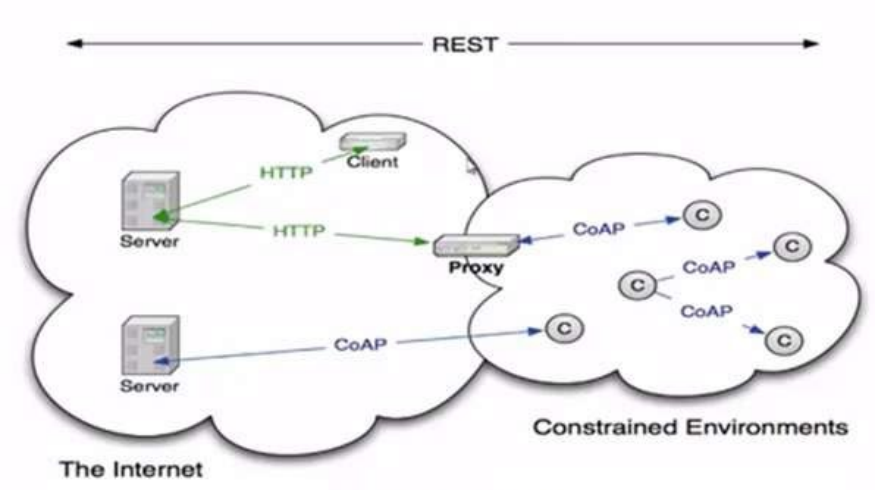
• It is a web transfer protocol which is used in constrained networks such as IoT, WSN, M2M etc. to provide communication between constrained devices having less memory/less power specifications.

• It follows client/server model. GET, PUT, POST and DELETE methods are used.

• It uses both request/response and publish/subscribe models.

• It supports binding to UDP, TCP and SMS.

• It is very efficient RESTful protocol as defined in RFC 7252.



CoAP Architecture

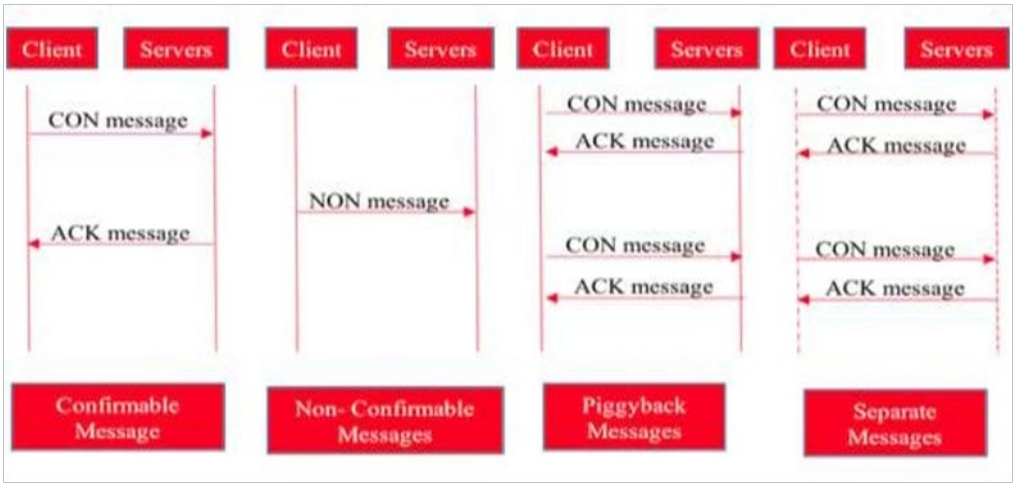


Fig: COAP messages

Following are the benefits or advantages of CoAP protocol:

➡It is simple protocol and uses less overhead due to operation over UDP. It allows short wake up times and long sleepy states. This helps in achieving long battery life for use in IoT (Internet of Things) and M2M (Machine to Machine) communication.

➡It uses IPSEC or DTLS to provide secure communication.

➡Synchronous communication is not necessity in CoAP protocol.

➡It has lower latency compare to HTTP.

➡It consumes less power than HTTP.

➡It uses ACK message and hence it becomes reliable like HTTP. Moreover it avoids unnecessary retransmissions.

➡CoAP protocol is used as best protocol choice for home communication networks. It is used in information appliances, communication equipments and control equipments in smart home networks

Drawbacks or disadvantages of CoAP protocol

Following are the drawbacks or disadvantages of CoAP protocol:

➨CoAP is unreliable protocol due to use of UDP. Hence CoAP messages reach unordered or will get lost when they arrive at destination. To make CoAP as reliable protocol, stop and wait with exponential backoff retransmission feature is incorporated in it. Duplicate detection is also introduced.
➨It acknowledges each receipt of the message and hence increases processing time. Moreover it does not verify whether the received message has been decoded properly or not.
➨It is unencrypted protocol like MQTT and uses DTLS to provide security at the cost of implementation overhead.
➨CoAP has communication issues for devices behind NAT (Network Address Translation).

XMPP

Extensible Messaging and Presence Protocol (XMPP) is a messaging protocol that was designed originally for chatting and message exchange applications. It was standardized by IETF more than a decade ago. Hence, it is well known and has proven to be highly efficient over the internet. Recently, it has been reused for IoT applications as well as a protocol for SDN. This reusing of the same standard is due to its use of XML which makes it easily extensible. XMPP supports both publish/ subscribe and request/ response architecture and it is up to the application developer to choose which architecture to use. It is designed for near real-time applications and, thus, efficiently supports low-latency small messages. It does not provide any quality of service guarantees and, hence, is not practical for M2M communications. Moreover, XML messages create additional overhead due to lots of headers and tag formats which increase the power consumption that is critical for IoT application. Hence, XMPP is rarely used in IoT but has gained some interest for enhancing its architecture in order to support IoT applications

XMPP Advantages:

It is free and decentralized which means anyone can set up an XMPP server.

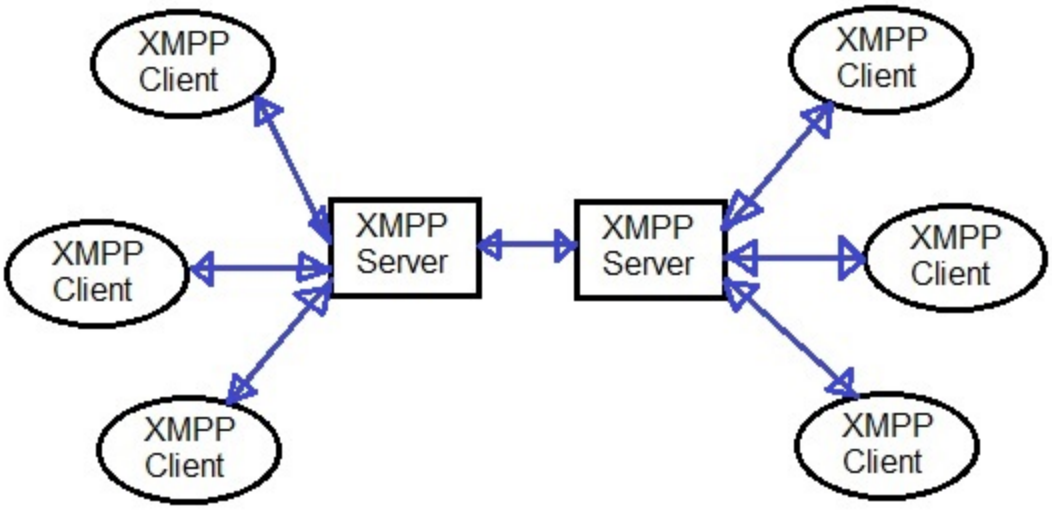
It is based on open standards.

It supports multiple implementations of clients and servers.

It is flexible, XML-based and can be extended. So, suitable for both instant messaging features and custom cloud services.

Security is supported via SASL and TLS.

It is efficient, can support million of concurrent users on a single service such as GTalk.



Following are features of XMPP protocol used between XMPP client and XMPP server for communication:

• XMPP uses port number 5222 for client to server (C2S) communication.

• XMPP uses port number 5269 for server to server (S2S) communication.

• Discovery and XML streams are used for S2S and C2S communications.

• Uses security mechanisms such as TLS and SASL.

• There are no intermediate servers for federation unlike E-mail.

The features of XMPP viz. addressing, scalability, Federation and security are ideal for Internet of Things (IoT) applications. Federation is a feature by which two business domain users can talk to each other.

Example: connectivity from thermostat to web server which can be later accessed easily using mobile phone.

DDS

Data Distribution Service (DDS) is another publish/subscribe protocol that is designed by the Object Management Group (OMG) for M2M communications [DDS]. The basic benefit of this protocol is the excellent quality of service levels and reliability guarantees as it relies on a broker-less architecture, which suits IoT and M2M communication. It offers 23 quality-of-service levels which allow it to offer a variety of quality criteria including: security, urgency, priority, durability, reliability, etc. It defines two sublayers: data-centric publish- subscribe and data-local reconstruction sublayers. The first takes the responsibility of message delivery to the subscribers while the second is optional and allows a simple integration of DDS in the application layer. Publisher layer is responsible for sensory data distribution. Data writer interacts with the publishers to agree about the data and changes to be sent to the subscribers. Subscribers are the

receivers of sensory data to be delivered to the IoT application. Data readers basically read the published data and deliver it to the subscribers and the topics are basically the data that are being published. In others words, data writers and data reader take the responsibilities of the broker in the broker-based architectures.

Protocols	UDP/TCP	Architecture	Security and QoS	Header Size (bytes)	Max Length(bytes)
MQTT	TCP	Pub/Sub	Both	2	5
AMQP	TCP	Pub/Sub	Both	8	-
CoAP	UDP	Req/Res	Both	4	20 (typical)
XMPP	TCP	Both	Security	-	-
DDS	TCP/UDP	Pub/Sub	QoS	-	-