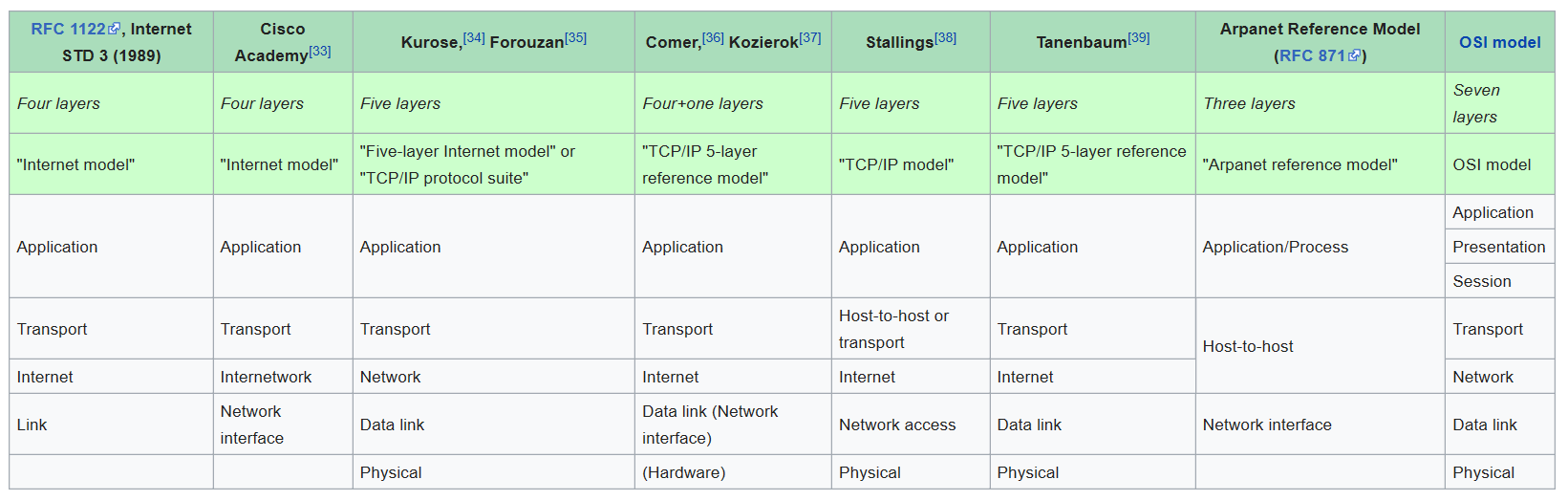
**Internet Protocol Suite**

The network layers are defined in various ways in different kinds of literature which are enlisted in the below figure.

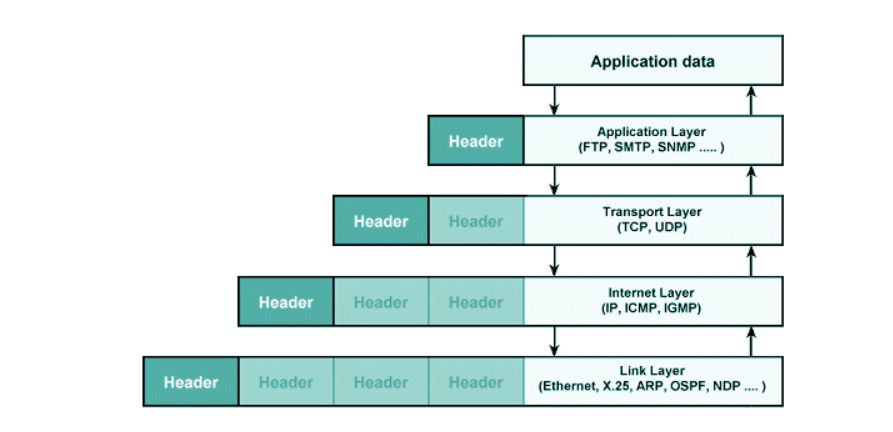
(contributers, 2019)

According to RFC 1122, Internet STD 3(1989) there are four layers.

**Application Layer**: It is a layer used by applications to provide user services, exchange data over network connections created by lower level protocols. Some protocols in application layers are HTTP, FTP, SMTP, DHCP, DNS, etc. Data encoded is given to the transport layer for transmission of data over the network.

**Transport Layer**: The layer provides data channels identified by port numbers for providing process-specific transmission channels for applications. It establishes host-to-host connectivity. It may provide services like error control, segmentation, flow control, congestion control. Some of the transport layer protocols are TCP, UDP, DCCP, SCTP, RSVP, etc.

**Internet Layer**: The layer acts as an interface for the different topology of the underlying network. It is used to exchange datagrams over network boundaries also known as routing. It performs two basic functions Host addressing and identification, Packet routing. This layer provides the IP Address for network interface identification and location addressing. The internet protocol version 4 (IPV4) and IPV6 are used in present-day internet.

**Link Layer**: It has two sublayers, Logical link layer and Medium Access Control (MAC). The logical layer is responsible for the framing of the datagrams received from the Internet layer for the final transmission of frames over the physical link. Logical layer is also responsible for error control, flow control, multiplexing of packets for transmission and de-multiplexing while receiving. MAC protocol is used for link access i.e. when multiple nodes try to share a single broadcast link. It can be implemented on both the software level and hardware level (Network Interface Cards). Some of link layer protocols are MAC (Ethernet, Wi-Fi, DSL, ISDN, FDDI), ARP, OSFP, Tunnels, etc.

**Functions of Transport Layer**

The transport layer may support the following services:

1. Connection-Oriented communication.
2. In order delivery.
3. Reliability
4. Flow Control
5. Congestion avoidance

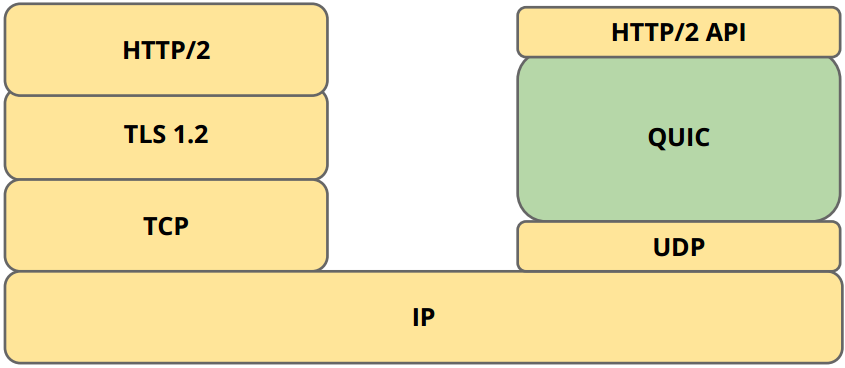
**Transport Layer Protocols:**

* User Datagram Protocol (UDP)
* UDP-Lite
* Reliable UDP (RUDP)
* Transmission Control Protocol (TCP)
* Multipath TCP (MPTCP)
* Stream Control Transmission Protocol (SCTP)
* Datagram Congestion Control Protocol (DCCP)
* Fibre Channel Protocol (FCP)
* IL protocol
* Reliable Data Protocol (RDP)
* Sequenced Packet Exchange (SPX)
* Structured Stream Transport (SST)
* Micro Transport Protocol (µTP)
* AppleTalk Transaction Protocol (ATP)
* Cyclic UDP (CUDP)

**Introduction To QUIC**

QUIC is a transport protocol initially developed by Google. Initially, it was an acronym for **Q**uick **U**DP **I**nternet **C**onnection but later IETF did not recognize it as an acronym rather as a protocol name itself. QUIC is a transport layer protocol build on UDP to provide facilities same as TCP and even more. QUIC supports a reliable end to end connection. It has stream multiplexing, encryption, and low-latency features. QUIC even has modular congestion control so as to allow users their own custom-built algorithms.

The below image shows the stack arrangement of TCP and QUIC.



QUIC has been adopted by the majority of Google’s server and browsers like Chrome, Mozilla Firefox are using QUIC protocol. Applications like YouTube, Google Search Engine have seen a major improvement in terms of speed. QUIC reduced the latency of Google Search response by 8.0% for desktop users and 3.6% for mobile users. It reduced rebuffer rates of YouTube playbacks by 18.0% for desktop users and 15.3% for mobile users. It accounts for over 30% of Google’s total egress traffic in bytes and consequently an estimated 7% of global Internet traffic.

**Need for New Protocol**

Development of new protocols is unable to be deployed over the internet mainly because of the Middleboxes. Middleboxes like firewall usually tend to block anything unfamiliar and NAT’s rewrite transport header and thus making them undeployable without adding explicit support. Moreover, even modifying TCP has become a bottleneck as any packet content which is not encrypted such as TCP packet header can be accessed by middleboxes to inspect and modify the contents.

TCP is implemented at the kernel level and hence even if the updates or modifications were deployable, pushing changes to TCP stack requires OS upgrades which take a long time for reaching the users.

Head-Of-Line blocking delay can be of two types. First due to HTTP where it limits the number of connections with the server. Once a file is requested the connection is unused till the server responds. This is called HTTP HOL blocking. The second type is called TCP HOL blocking. TCP’s byte stream abstraction makes it unable to identify the stream in which the packet is lost and hence all the streams have to pause until the lost packet is retransmitted even though some of the packets in the stream are independent of the lost packet.

Latency plays an important role in the ever-growing traffic of the internet. The TCP takes a one-round trip and TLS takes another two-roundtrips to ensure a secure connection. This seems to be a burden for short-lived transactions. TCP Fast Open and TLS 1.3 tries to reduce this delay.

QUIC tries to resolve all the above-stated problems. The middleboxes are mitigated by completely encrypting the packets so that the protocol is protected from ossification. QUIC is implemented at the user level so that any future updates need not face the issue of OS upgrades to make it practically implementable. QUIC uses multiplexing of streams and hence a single connection between server and client is sufficient and each stream is independent of each other. Hence this solves both HTTP and TCP HOL blocking issue. QUIC usually takes 1-RTT to establish a new connection and 0-RTT for a connection with an already known server.

**Differences Between QUIC and TCP**

QUIC uses a slightly modified version of loss detection and congestion control of TCP.

* **Separate Packet Number Spaces:** Packet number ranges from 0 to 262 – 1. This number is used in the cryptographic nonce for packet protection. Each endpoint maintains a separate packet number for sending and receiving. Packet numbers are divided into 3 spaces:
  + - **Initial space:** All initial packets are in this space. Initial packets use long header, includes first CRYPTO frame sent by server and client for key exchange and carries ACKs in either direction.
    - **Handshake space:** All handshake packets are in this space. These packets are used after the client has received a Handshake packet from a server. These are used to send subsequent cryptographic handshake messages and acknowledgments to the server. It may even contain CONNECTION\_CLOSE frames. The payload may contain PADDING or ACK frames.
    - **Application space:** All 0-RTT and 1-RTT encrypted packets are in this space.

This method makes sure that the acknowledgment packet sent with one level of encryption does not cause spurious retransmission of packets in another encryption level.

* **Monotonically Increasing Packet Numbers:** TCP uses same packet numbers at both sender and receiver and retransmits same data carrying the same packet number and results in “retransmission ambiguity” (Suppose a time out occurs for a packet and is retransmitted and later an acknowledgment is received for that packet. In estimating RTT there arises an ambiguity as we do not know whether it is for the initial packet or the retransmitted packet.) QUIC separates the two. When an ack-eliciting frame is lost, QUIC bundles the necessary frames in a new packet with new packet number and hence more accurate measurements of RTT can be estimated.
* **No Reneging:**
* **More ACK Ranges:** QUIC supports many ACK ranges which is more favorable in lossy networks and speeds recovery, reduces spurious retransmissions and ensures forward progress without relying on timeouts.
* **Explicit Correction For Delayed Acknowledgments:** QUIC takes into account for the delay incurred between when a packet is received and when the corresponding acknowledgment is sent for a more accurate estimate of RTT.

**TCP Congestion Control**

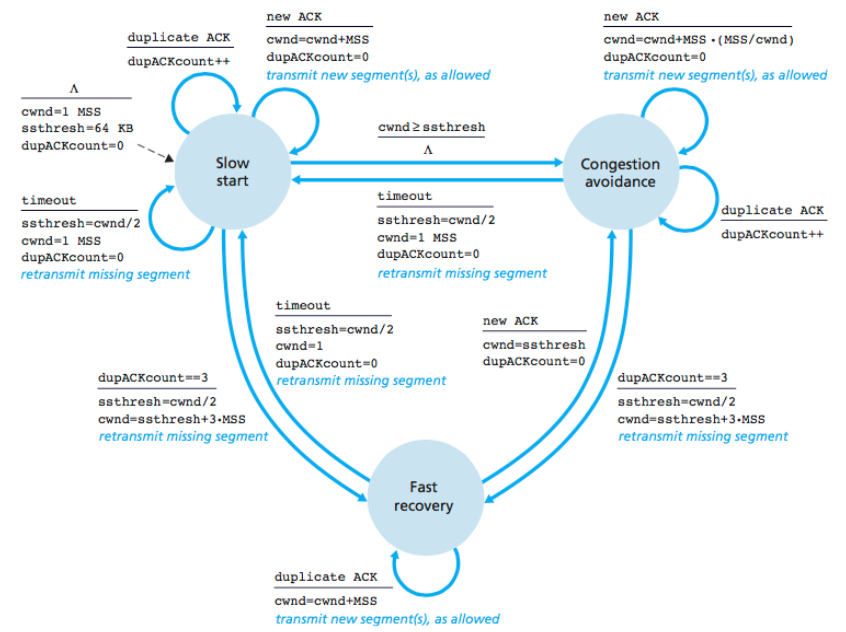
**Congestion Window:** It is a window maintained by the sender which acts as a factor to restrict the number of outstanding packets. It is updated dynamically with the network changes due to congestion.

**Additive Increase Multiplicative Decrease (AIMD):** It is a method adopted by TCP congestion control to vary the congestion window.

w(t) = w(t) + a ; if congestion is not detected.  
 w(t) \* b ; if congestion is detected.

Where w(t) is sending rate (eg: Congestion Window), a>0, 0<b<1.

**State Machine Diagram for TCP Congestion Control**

****

**Slow Start:   
For each “*ack”,*  cwnd = cwnd + MSS. (Exponential Increase)**

* In this state, the congestion window (cwnd) is increased by one packet for each “ack” packet.
* The congestion window is initially set to 1MSS.
* If 3 duplicate ack is received by the server it enters Fast Recovery State by performing the following actions
  + slow start threshold value set to half of the congestion window.
  + the congestion window is set to the new ssthresh + 3MSS.
  + retransmit missing segment
* If congestion window hits ssthresh it enters Congestion Avoidance State.
* If a timeout occurs
  + slow start threshold value will be set to half of the congestion window.
  + duplicate ack count is reset to zero.

**Congestion Avoidance:**

**For each “ack” cwnd = cwnd + MSS(MSS/CWND) (Linear Increase)**

* In this state congestion window is updated by 1MSS.
* If 3 duplicate ACKs are received by the server it enters Fast Recovery State by performing the following actions
  + slow start threshold value will be set to half of the congestion window.
  + the congestion window is set to the new ssthresh + 3MSS.
  + retransmit missing segment
* If a timeout occurs it enters Slow Start State
  + slow start threshold value will be set to half of the congestion window.
  + duplicate ACK count is reset to zero.

**Fast Recovery:**

**For each “duplicate ack” cwnd = cwnd + MSS (Linear Increase)**

* The increase is linear because the congestion window is updated by 1MSS only for duplicate ack.
* If a new ack is received it enters Congestion Avoidance State by performing the following actions
  + congestion window set to slow start threshold value.
  + duplicate ack count is reset to zero.
* If a timeout occurs it enters Slow Start State
  + slow start threshold value will be set to half of the congestion window.
  + duplicate ack count is reset to zero.

**TCP Tahoe**

* Initially, the congestion window is set to 1 MSS and enters the slow-start state.
* When a loss occurs
  + Retransmits the lost packet.
  + Sets ssthresh to half of the congestion window
  + Enters slow-start with congestion window again set to 1 MSS
* When cwnd reaches ssthresh it enters the congestion-avoidance state.

Tahoe waits for a timeout interval to detect a packet loss which appears to be very poor in performance for high bandwidth-delay product links.

**TCP Reno**

TCP Reno handles the problem of a lost packet by introducing fast retransmit state instead of waiting for a time-out to detect a packet loss.

* Initially, congestion window is set to 1 MSS and enters the slow-start state.
* When a loss is detected (3 dup ACKs)
  + Enters fast retransmits state
  + Sets ssthresh to half of the cwnd.
  + Sets the cwnd to the ssthresh.
* When cwnd hits ssthresh it enters the congestion-avoidance state.

TCP Reno can handle a small number of packet losses but in case of multiple packet losses, it suffers the same as TCP Tahoe.

**TCP NewReno**

TCP NewReno is a slight modification to Reno to handle multiple packet losses. It introduces partial ACK. Partial ACK is an acknowledgment received during fast-retransmit which has a packet number less the packet number that was transmitted before the fast-retransmit state.

* Initially, the congestion window is set to 1 MSS and enters the slow-start state.
* When a loss is detected (3 dup ACKs)
  + Enter the fast retransmit state.
  + Sets ssthresh to half of the cwnd.
  + Sets cwnd to the ssthresh.
* When a partial ACK is received it is immediately retransmitted.

This algorithm suffers from the fact that it takes 1-RTT to detect each packet loss.

**TCP CUBIC**

This is suitable for networks with high latency and bandwidth (big fat networks). In such a type of networks clients which are close to server receives ACKs faster than the clients which are far away and hence the congestion window of the clients closer will have faster growth than the one which is far away. CUBIC does not depend on the RTTs to modify the window size. It depends on the time elapsed since the last congestion.

: Multiplicative decrease factor

: Window size just before the last reduction.

T: Time elapsed since the last window reduction.

C: A scaling constant.

cwnd: The congestion window at the current time.