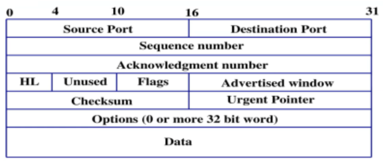
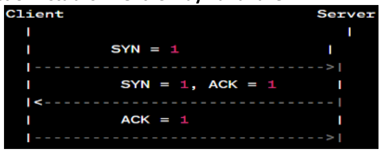
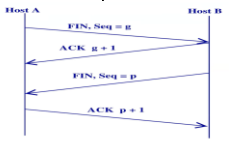
TRANSPORT LAYER
Ephemeral ports – 1024-65535
Quadruple (SIP, STCPPORT, DIP, DTCPPORT) – uniquely identifies TCP
TCP Header format:



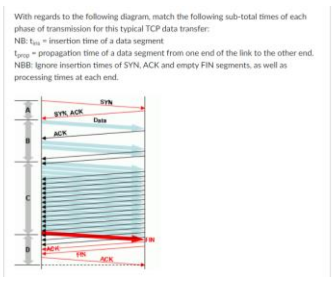
TCP Header Checksum – checksums are computed over the TCP header, TCP data, and the Pseudoheader. Pseudoheader consists of IP source and dest addresses, TCP seg length, and protocol field from IP header.
Transport Layer Header includes source and destination port numbers.
TCP Connection Establishment: 3 way handhake



TCP Connection Termination: 4 way Handshake



TCP Flow Control: Regulates data flow between sender and receiver. Receiver advertises its available buffer space through a receive window. Sender adjusts transmission rate based on receiver's window size. Prevents overwhelming the receiver and network congestion. **Data Acknowledgement:** The receiver sends ACK packets to confirm successful receipt of data. ACKs indicate the next expected sequence number. If sender doesn't receive ACK within timeout, it retransmits the data. **ACK Piggybacking:** Technique to combine ACK packets with data packets. Reduces the number of transmitted packets on the network. Improves network efficiency and conserves resources. ACK information is included in the data packet's header or payload. **Error Control:** Ensures reliable delivery of data packets. Sequence numbers and acknowledgments detect missing or duplicate segments. Retransmission of missing segments if ACK not received or errors detected. Checksum verifies integrity of received segments. Selective Repeat or Go-Back-N strategies handle retransmitted segments.
Congestion occurs when the load to the link is persistently higher than its capacity. The effects of congestion include an increase in packet delay and sustained packet loss. This can be alleviated by sources reducing load in the short term, or increasing link capacity in the long term. It's important to note that increasing router buffer size only delays the inevitable, leading to increased latency, an issue commonly known as buffer bloat.
TCP uses various strategies to control congestion:
AIMD Rule: TCP sources adjust their sending rates according to the Additive Increase Multiplicative Decrease (AIMD) rule. The rate is increased linearly if there is no congestion and decreased exponentially if there is congestion.
Slow Start: This technique is used at the beginning of a TCP stream to increase the congestion window quickly. The congestion window is incremented by 1 every time an acknowledgment (ACK) is received, allowing it to double every round-trip time (RTT) if there are no losses.
Congestion Avoidance: This strategy is entirely the sender's responsibility and is performed independently in each direction.
Fast Retransmit: If the sender receives three duplicate acknowledgments, it retransmits the packet without waiting for a timeout.
Tahoe introduced AIMD, slow start, and fast retransmit, while Reno added fast recovery, which sets the congestion window to half of its previous value after fast retransmitting a packet.
User Datagram Protocol (UDP) is another transport layer protocol, which is connectionless, message-oriented, and does not provide delivery guarantees, flow control, or congestion control.
Since UDP doesn't perform congestion control, it can lead to unfair bandwidth sharing when TCP and UDP share a network link. Additionally, TCP can incorrectly perceive packet loss due to interference (for example, in a WiFi network) as congestion, leading to unnecessary slowing of transmission and reduced throughput.


With regards to the following diagram, match the following sub-total times of each phase of transmission for this typical TCP data transfer:
NB: t_{ins} = insertion time of a data segment
t_{prop} = propagation time of a data segment from one end of the link to the other end.
NBB: Ignore insertion times of SYN, ACK and empty FIN segments, as well as processing times at each end.

Insertion time is the time taken to send one segment (proportional to the size of the packet and inversely proportional to the speed of the network). The propagation time is how long it takes for the packet to get from the source to the destination (e.g. from the time the first bit 'leaves' the sender to when it arrives at the receiver. This is a function of the distance between the two endpoints, delays in the network, speed of signal propagation (e.g. speed of light or speed of electromagnetic signal transfer in the medium).

So for phase A, for example, we aren't sending any data - the TCP segments are empty, only containing a header with SYN/ACK flags set as appropriate for connection establishment. t_ins is negliglble here, so the time to get the ACK back is the time it takes for the SYN packet to get from source to destination plus the time taken for the ACK to return (we assume the t_prop is the same in both directions here). Therefore Phase A is 2 t_prop. For Phase B, we send one data frame - this takes a full t_ins to leave the sending side - and then all the bits take t_prop to get to the receiver. So the receiver has received the last bit in the segment after t_ins + t_prop. Only then can it send an ACK, which takes another t_prop to get back to the sender (so phase B has lasted t_ins + 2 t_prop so far). Now we can increase the window size to 2. We send data segment 2 (this again takes t_ins) and off it goes, while it is travelling we are transmitting data segment 3. When data segment 2 fully arrives (t_ins + t_prop after we sent segment 2) we can ACK it, which arrives at the sender another t_prop later (so total duration of Phase B is now 2 * (t_ins + 2 t_prop)). As soon as that happens we can send another segment - and in fact we will get ACKs back before our window becomes empty, so we can now transmit continuously - we have entered Phase C, the steady-state phase. So Phase B lasted 2*(t_ins + 2 t_prop). Phase C is steady-state transmission - it continues until we run out of segments, and it is just the number of segments to be transmitted multiplied by t_ins (16 * t_ins). Finally we have the teardown; we apparently did a piggyback of the FIN flag on the last data segment (red), so teardown lasts from the end of the transmission of the last packet to the final stage of the handshake, which you should be able to see is 3 t_prop (no t_ins because the termination handshake uses empty segments - just with FIN/ACK flags set as appropriate).

APPLICATION LAYER
Based on their service requirements - for example:
- Delay-tolerant, loss-sensitive (web traffic, file distribution, one-way video streaming)
- Loss-tolerant, delay-sensitive (real-time two-way voice/video calls, games)

Based on their architecture:
- Client-server (the traditional architecture) - e.g. a web browser and web server. This is a star topology.
- Peer-to-peer - distributed protocols in which all hosts can play a similar role - e.g. BitTorrent, TOR. This is a mesh topology.

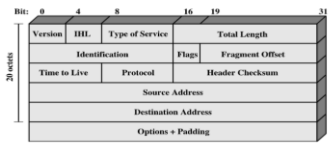
Many combinations of these classifications exist, and many applications utilize more than one mode of operation for different aspects of their functionality.
Connection-Oriented protocols at the Application Layer establish a connection before data transmission, ensuring reliable and ordered delivery with features like flow control and error control. They guarantee delivery, maintain the connection state, and are exemplified by protocols like TCP and HTTP.
Connectionless protocols operate without establishing a connection beforehand. They send data independently and lack guarantees of delivery or connection state maintenance. Examples include UDP and DNS.
Network socket buffers- A buffer is just a region of memory used to store data**.** It is important to realize that when data is received, it is stored by the operating system in a buffer. The application which is to receive this data is not obliged to read it at that moment - it can wait. In this case, more data may arrive at the buffer Even if the application tries to read the buffer, and there is some data in it, the operating system can make it wait The idea here is that if a very small amount of data has been received,

reading_this will be inefficient. So allow a bit more data to accumulate before allowing the_read operation to succeed._This behavior can be overruled when you create the socket. Think about why you may want to do this!

NETWORK LAYER

Routing is finding a path through the network from source to destination – is the main responsibility of the network layer.

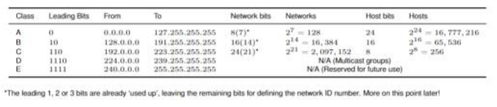
Ipv4 datagram header:



The minimum (and default) IP header length is 20 bytes; this would be equal to 5 (01012). Maximum IP header length is 11112 = 1510 32-bit words or 60 bytes.

The IPv4 header carries important information for routing and delivering data packets. Here's a summary of the IPv4 header details and their values:
Version: The value is typically 4, indicating IPv4.
Header Length: Varies from 5 to 15, representing the length in 32-bit words.
Differentiated Services: 8 bits are used for prioritization.
Total Length: a 16-bit value indicating the total length of the packet.
Identification: Unique 16-bit value for packet fragments.
Flags: 3 bits with the "Don't Fragment" (DF) and "More Fragments" (MF) flags.
Fragment Offset: 13 bits denoting position within the original packet.
Time to Live (TTL): 8-bit maximum hop count before discarding.
Protocol: 8 bits specifying the protocol used (e.g., TCP, UDP, ICMP).
Header Checksum: 16-bit checksum for error detection.
Source IP Address: 32-bit sender's IP address.
Destination IP Address: 32-bit receiver's IP address.
Options: Variable-length field for optional information.
These values within the IPv4 header ensure efficient routing and delivery of IP packets across networks.

<u>Ipv4 Address Classess</u>



Class	Leading Bits	From	To	Network bits	Networks	Host bits	Hosts
A	0	0.0.0.0	127.255.255.255	8(7)*	$2^7 = 128$	24	$2^{24} = 16,777,216$
B	10	128.0.0.0	191.255.255.255	16(14)*	$2^{14} = 16,384$	16	$2^{16} = 65,536$
C	110	192.0.0.0	223.255.255.255	24(21)*	$2^{21} = 2,097,152$	8	$2^8 = 256$
D	1110	224.0.0.0	239.255.255.255		N/A (Multicast groups)		
E	1111	240.0.0.0	255.255.255.255		N/A (Reserved for future use)		

*The leading 1, 2 or 3 bits are already 'used up', leaving the remaining bits for defining the network ID number. More on this point later!

Routing Protocols

Static routing involves manually configuring network routes by a network administrator. Routes are manually entered into the routing table, specifying the destination networks and the corresponding next-hop routers. It is commonly used in small networks or situations where network topology changes infrequently. However, static routing does not adapt to changes in the network, necessitating manual updates when modifications occur. On the other hand, **dynamic routing** utilizes routing protocols to automatically exchange routing information between routers. Routers dynamically update and share details about the network's topology and reachable destinations. Dynamic routing protocols like RIP, OSPF, and BGP enable routers to adjust to network changes, facilitating efficient routing and faster convergence. Dynamic routing offers scalability, flexibility, and automated handling of network modifications**. Routing stability**, regardless of the routing method used, is essential. It ensures consistent and reliable functioning of the routing infrastructure. Stable routing helps maintain uninterrupted connectivity, prevents routing loops, and ensures correct forwarding decisions. Achieving routing stability requires proper network design, reliable routing protocols, and effective network management.

Interior routing protocols are used within an autonomous system (AS), which refers to a single organization's network. They focus on exchanging routing information between routers within the same AS to determine the best paths for internal routing. Examples of interior routing protocols include *Routing Information Protocol (RIP)*, *Open Shortest Path First (OSPF)*, and *Interior Gateway Routing Protocol (IGRP)*. **Exterior routing protocols**, on the other hand, are designed for inter-domain routing between multiple autonomous systems. The *Border Gateway Protocol (BGP)* is the primary exterior routing protocol used on the internet. Exterior routing protocols consider factors such as policies, inter-AS relationships, and preferences to determine the optimal paths for routing between different ASes. **Least cost routing algorithms** play a role in determining the best paths for packet forwarding by considering metrics or costs associated with network links, such as hop count, latency, or bandwidth. **Distance vector routing algorithms**, like RIP, make routing decisions based on hop count, while link-state routing protocols, such as OSPF and IS-IS, exchange detailed information about the network's state and topology to enable faster convergence and more efficient routing.

OSPF (Open Shortest Path First) is a widely used interior gateway routing protocol that operates within autonomous systems (ASes) to facilitate efficient and scalable routing. Based on the link-state routing algorithm, OSPF routers exchange information about their connections and the network topology, creating a synchronized view of the network. This information is stored in a link-state database, allowing OSPF to calculate the shortest path to each network destination using *Dijkstra's algorithm*. OSPF utilizes areas to organize the network, reducing the amount of routing information exchanged and improving scalability. It supports multiple metrics, including bandwidth, delay, reliability, and cost, enabling network administrators to prioritize routing decisions. OSPF provides fast convergence by quickly detecting network changes and adjusting routing tables. It is commonly used in enterprise networks, data centers, and internet service provider (ISP) networks due to its scalability, flexibility, and reliability. Additionally, OSPF supports authentication mechanisms to secure the exchange of routing information between routers, ensuring network integrity.

Dijkstra's algorithm is a well-known and widely used graph traversal algorithm that solves the single-source shortest path problem. It finds the shortest path from a given source node to all other nodes in a weighted graph. The algorithm begins by assigning a tentative distance value to all nodes, setting the source node's distance to 0 and the distances of all other nodes to infinity. It then iteratively selects the node with the smallest tentative distance and explores its neighboring nodes. For each neighboring node, it updates the tentative distance if a shorter path is discovered. This process continues until all nodes have been visited or until the destination node is reached. Dijkstra's algorithm employs a priority queue or min-heap data structure to efficiently select the node with the smallest tentative distance. By the end of the algorithm, the final distance assigned to each node represents the shortest path from the source node. Dijkstra's algorithm can be applied to both directed and undirected graphs with non-negative edge weights. It has widespread applications in various domains, including routing protocols, network analysis, pathfinding in maps or GPS systems, and optimizing transportation or logistics routes.

THE DATA LINK LAYER

The data link layer is responsible for the delivery of data within a single network. Key functions include; Framing, Addressing, Error detection/Correction/Recovery. Importantly - Layer 2 protocols are generally closely linked to an associated Layer 1 protocol.

Framing involves dividing the data stream into manageable units called frames, which are encapsulated with a header and a trailer. The header contains control information like source and destination addresses, frame type, and error detection bits, while the trailer often includes error-checking codes like *cyclic redundancy check (CRC)* for detecting transmission errors. Framing ensures that data can be transmitted and received as discrete units, enabling synchronization and proper handling by the receiving device. On the other hand, **addressing** allows for the identification of specific devices within a network. Each device is assigned a unique address, such as a *Media Access Control (MAC) address*, which is globally unique and typically assigned by the manufacturer. The source and destination MAC addresses are used in the framing process to indicate the sender and intended recipient of the frame. Addressing in the Data Link Layer enables direct communication between devices on the same network segment, facilitating reliable data transmission and efficient communication within the network.

The Layer 2 preamble is a sequence of bits located at the beginning of a data frame within Layer 2 of the OSI model. Its primary purpose is to establish synchronization and signaling between the sender and receiver devices. Typically consisting of a specific bit pattern, the Layer 2 preamble is inserted at the start of the frame. Upon receiving a frame, the receiver utilizes the bit pattern in the preamble to identify the beginning of the frame and initiate synchronization with the incoming data. This synchronization process allows the receiver's internal clock to align with the sender's clock, ensuring accurate data reception. Once synchronization is achieved, the subsequent data in the frame can be reliably processed. It is important to note that the Layer 2 preamble is not part of the actual payload or data within the frame and is typically discarded or disregarded after the synchronization process is complete.