Data **link layer** services:

- Flow control
- Link Access: A MAC protocol specifies the rules by which a frame is transferred onto the link.
- Reliable Delivery
- Error Detection and Correction - Checksum

Functions of the **network layer**:

Routing : Determine route taken by packets from source to destination.
Forwarding : Move packets from routers input to routers output.

A **routing protocol** defines how routers communicate with each other about the topology of the network, which will be used by a routing algorithm. A **routing algorithm** uses information about the topology of the network to compute the least-cost path from a source to a destination.

Difference between **broadcasting and flooding**: broadcasting is sending out a packet to the broadcast address (host action), flooding is the act of forwarding a packet (router/switch action).
Anycast: similar to multicast, except that only the closest node receives the packet.

Guided Media:

- Fibre Optic (Fast/Expensive)
- Twisted Pair Copper Wire(Cheap/Faster options available)

Unguided:

- WiFi (No physical connection needed/Collisions)
- Satellite (connection range is large/expensive)

Access networks connect an end system (host) to the first (edge) router on a path to any other end system.

DSL : Uses telephone line (twisted-pair copper wire) to communicate with a DSL access multiplexer in the telco's local central office. Modem converts digital to analogue and vice versa. Uses FDM to support internet and phone traffic.
Cable : Uses cable TV line. Fibre to the neighbourhood junction, then coaxial to homes (hybrid fibre coax, HFC). Shared broadcast medium, as each packet sent from the head travels on every link to every home.
Fibre : *Direct fibre* allocates one fibre from the central office to the home. Others split fiber closer to the home, using an *active optical network* or *passive optical network*.

MAC: Distributed algorithm that determines how nodes share the channel. Types:

Channel partitioning : divide channel based on time/frequency/code, or allocate portion to node for exclusive use.
Random access : Channel not divided, which allows collisions. Need to be able to recover from collisions.
Ordered access : Nodes take turns. Nodes with more to send can take longer turns, or nodes with higher priority get more turns.

Throughput: number of frames successfully transmitted through the channel per frame time.

"Pure" ALOHA:

- If you have data to send, send the data
- If, while you are transmitting data, you receive any data from another station, there has been a message collision. All transmitting stations will need to try resending "later".
- Disadvantage: time is wasted, data is lost.
- Probability that k frames are generated during a frame transmission time: $Pr[k] = \frac{S^k e^{-S}}{k!}$ (Poisson), where S is the mean number of frames generated per frame transmission time. If we include retransmissions with G instead of S: $Pr[k] = \frac{G^k e^{-G}}{k!}$.
- Under all loads: $S = GP_0$, where P_0 is the probability that a frame doesn't suffer a collision.

"Slotted" ALOHA, transmitting with probability p:

- Probability that a node successfully transmits: $p(1-p)^{N-1}$
- Probability that any node is successful: $Np(1-p)^{N-1}$

Why can't CSMA/CD be used on wireless networks?

- Wireless transceivers can't send and receive on the same channel at the same time, so they can't detect collisions.
- Instead they incorporate a CSMA/CA (Collision Avoidance) to sense the channel hear if anyone is sending, and if not they send, otherwise, they back off for a random amount of time.

CSMA/CD efficiency:

$$\eta = \frac{1}{1 + 5\frac{d_{prop}}{d_{trans}}}$$

Vulnerable period for pure CSMA is d_{prop}, versus $2d_{prop}$ for CSMA/CD. CSMA variants:

1-persistent : If link is busy, try again. If idle, send immediately, and if a collision is detected, wait a random period and try again.
Non-persistent : If busy, wait a random period, then try again. If idle, try to send, and do the same as 1-persistent if a collision occurs.
***p*-persistent** : If busy, try again. If idle, transmit with probability p, otherwise try again. If a collision occurs, do the same as 1-persistent.

Network core:

Circuit Switching : Dedicated resources; but inefficient, with setup overhead (signalling protocol). Usually less efficient than packet switching.
Packet switching : Good for bursty data, no call setup overhead; but congestion leads to packet delay and loss. Uses *store-and-forward transmission* at links, i.e., the entire packet must be received before doing anything with it. Uses FDM and TDM. Allows more users to use the network. Resource sharing.

CRC - Checked by receiver, if packet has error it is dropped.
4 Types of **Delay** - Processing, Queueing, Transmission (L/R), Propagation.
Nodal delay: $d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop}$.
Traffic intensity: $\Lambda = \frac{La}{R}$.
Bandwidth delay product: $BD = B \times d_{prop}$, where B is the bandwidth.
Network Edge - Hosts - Network Core - Interconnected Routes - Access Network - Wired/Wireless Links

- TCP - Connection oriented
- UDP - Connectionless, best effort, used for streaming and DNS.
- **UDP services** - process-to-process data delivery and error checking (checksum).
- **TCP services** - Same as UDP, plus: reliable data transfer, congestion control. UDP has smaller packet header overhead.

Queueing: When a shared facility needs to be accessed for service by a large number of jobs or customers. There is an exponential relationship between the queueing delay and the traffic intensity.
Little's law:

$$E[N] = \lambda E[T]$$

Where $E[N]$ is the expected number of packets in the system, λ is the arrival rate, and $E[T]$ is the expected time spent in the system.
Kendall notation:

$$1/2/3(4/5/6)$$

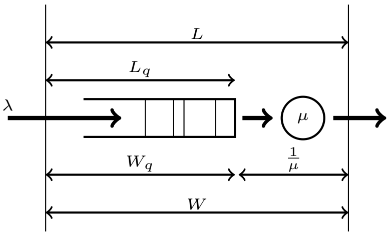
1. Arrival distribution
2. Service distribution
3. Number of servers
4. Total storage (including servers, often infinite)
5. Population size (often infinite)
6. Service discipline (FCFS/FIFO)

- Probability of inter-arrival time t in a Poisson process: $Pr(t) = \lambda e^{-\lambda t}$
- Expected (average) inter-arrival time: $E(t) = \frac{1}{\lambda}$

Markovian systems map to continuous-time Markov chains, which can be solved. For Markovian systems, we assume Poisson distributed arrivals and exponential service.

M/M/1 : 1 server, infinite capacity/population, FIFO.
M/M/m/m : As above, but m servers, m storage (e.g. telephone switching network).

With L being the average number of packets in the system, L_q the average number in the queue (~ 1), W the average waiting time in the system, and W_q the average waiting time in the queue ($\sim 1/\mu$):



Solutions:

- Utility factor: $\rho = \frac{\lambda}{\mu}$
- $L_q = \frac{\rho^2}{1-\rho}, L = \frac{\rho}{1-\rho}$
- $W_q = \frac{\rho}{\mu(1-\rho)}$
- $W = \frac{1}{\mu(1-\rho)} = \frac{1}{\mu-\lambda}$
- Fraction of time spent with n packets in system: $P_n = \rho^n(1-\rho)$
- Utilization factor: $\rho = 1 - P_0$
- $E[L] = \frac{\rho}{1-\rho}$
- Little's law: $E[W] = \frac{E[L]}{\lambda} = \frac{\rho}{\lambda(1-\rho)} = \frac{1}{\mu-\lambda}$
- $E[L_q] = \frac{\rho^2}{1-\rho}$
- $E[W_q] = E[W] - \frac{1}{\mu} = \frac{\rho}{\mu-\lambda}$
- Balance equation for M/M/1/K queue: $P_j = \left(\frac{\lambda}{\mu}\right)^j P_0$
- Balance equation for the Erlang loss system: $P_j = \frac{1}{j!}\left(\frac{\lambda}{\mu}\right)^j P_0$

Scheduling policies:

Priority queuing : Classes have different priorities, choose the highest priority packet from non-empty queue.
Round robin : Each flow gets its own queue, circulate over the queues, processing the head of the queue one queue at a time.
Weighted Fair Queueing Generalized round robin, provide classes with different service over time. It works by keeping virtual timers for each packet, and each time the output link is idle it will transmit the packet with the smallest time.

Derivation of **buffer overflow** with 13 slots: intuitively, this is the probability of 1 minus the buffer holding any from 0 to 13 packets (or alternatively, the probability of it holding from 14 to infinity packets).

$$P_{ovflw} = 1 - (P_0 + P_1 + \cdots + P_{12})$$
$$= 1 - \sum_{n=0}^{12} P_n$$

Factorize $P_n = \rho^n(1-\rho)$ into $P_0 = 1 - \rho$ and ρ^n:

$$= 1 - P_0 \sum_{n=0}^{12} \rho^n$$

Using a geometric sum:

$$= 1 - P_0 \frac{1-\rho^{13}}{1-\rho}$$
$$= \rho^{13}$$

Limiting the **probability of loss** to less than 1×10^{-6}: we know that loss will occur when the buffer overflows, and we know that the probability of it holding n packets is ρ^n. So we need to solve an inequality for n:

$$\rho^n \leq 10^{-6}$$
$$\log \rho^n \leq \log(10^{-6})$$
$$n \log \rho \leq \log(10^{-6})$$
$$n \cdot -0.602 \leq -6$$
$$n > \frac{-6}{-0.602} \approx 10$$

Buffering recommendation: $\frac{RTT \times C}{\sqrt{N}}$, where C is the link capacity, and N is the number of flows connected to the node.

With an **Erlang loss system**, we assume exponential inter-arrival time and exponential service time, with s servers and no queueing. An Erlang delay system is the same, except it has an infinite queue.

IPv4 **address classes**:

A : 1 - 126
B : 128 - 191
C : 192 - 223
D : 224 - 239
E : 240 - 254

Devices on the same **subnet** interface have the same subnet part of the IP address. Can physically reach each other without an intervening router.
Types of **routing table** entries:

- Network route
- Host route
- Default route
- Loopback address

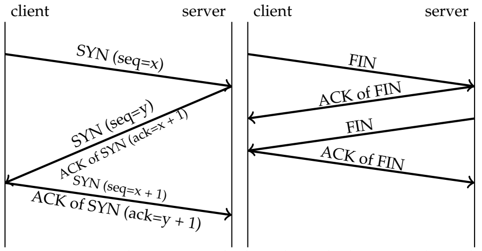
Multicast trees:

Source-based : One tree per source (shortest path, reverse path forwarding).
Group-shared : Group uses one tree (minimal spanning [Steiner] tree, centre-based).

Reverse path forwarding relies on the routers knowledge of the unicast shortest path from it to the sender, if packet comes from shortest path to spanning tree center, then flood. **Distance vector**: router knows physically-connected neighbours, iterative, distributed, asynchronous. **Link state**: All routers have complete topology/link costs.

Transport layer: logical communication between processes on different hosts. They run on end systems, sender breaks messages into segments and receiver reassembles segments into messages.
Multiplexing: Handle data from multiple sockets, add transport header.
Demultiplexing: Use header information to deliver received segments to correct socket.

TCP : Reliable, in-order delivery, congestion control, flow control, connection setup.
UDP : Unreliable, unordered delivery, thin wrapper over IP.



Congestion: sending too much data through the network, such that routers/switches start dropping packets (packet loss). Two approaches to congestion control:

Network-assisted : Routers provide feedback to end systems (e.g., a single bit indicating congestion [a la TCP/IP ECN], or an explicit transmission rate for sender).
End-end : No explicit feedback from network, so congestion is inferred from end system packet loss and delay (this is TCP's approach).

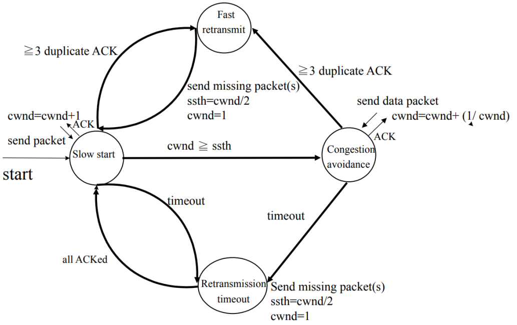
Difference between **flow control and congestion control**: flow control is about not overloading the *hosts*, whereas congestion control is about not overloading the *network*. TCP uses *window-based* flow control and congestion control.

TCP congestion control:

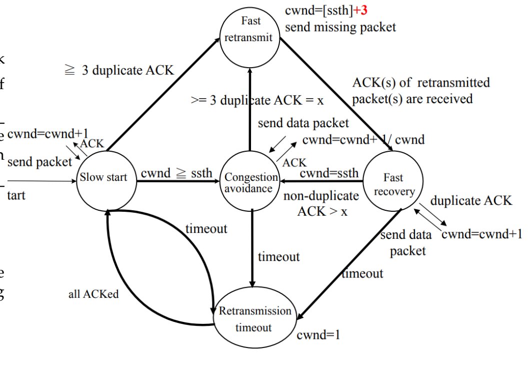
- Uses a congestion window (cwnd) to control the amount of transmitted data.
- Estimates the amount of outstanding data (awnd) as `snd.nxt` − `snd.una` (next unsent minus unacknowledged data). While awnd < cwnd, sender continues sending.
- When connection starts, the ssthresh is set to the maximum value of ssthresh (which depends on its data type).
- Doubles the congestion window every RTT (slow start).
- **Congestion avoidance** is about *slowly probing* available bandwidth, but *rapidly responding* to congestion events (*additive increase, multiplicative decrease*).
- When cwnd exceeds ssthresh, congestion avoidance begins, and increases cwnd by 1 MSS every RTT.
- **Fast retransmit**: When 3 duplicate ACK's (so 4 in total) are received after sending different packets, TCP knows that a packet was dropped, so it will retransmit it immediately instead of waiting for it to timeout.

After either a fast retransmit occurs, or an actual timeout, ssthresh is set to half the current congestion window size. In TCP Tahoe, slow start begins from the initial cwnd of 1 MSS; but TCP Reno skips slow start by halving cwnd, and enters **fast recovery**.

TCP Tahoe congestion control FSM:



TCP Reno congestion control FSM:


ssth=cwnd/2
cwnd=[ssth]+3
send missing packet
Fast retransmit
≧ 3 duplicate ACK
>= 3 duplicate ACK = x
ACK(s) of retransmitted packet(s) are received
send data packet
cwnd=cwnd+ 1/ cwnd
cwnd=cwnd+1
ACK
send packet
tart
Slow start
cwnd ≧ ssth
Congestion avoidance
ACK
cwnd=ssth
Fast recovery
non-duplicate ACK > x
duplicate ACK
cwnd=cwnd+1
send data packet
timeout
timeout
timeout
all ACKed
Retransmission timeout
cwnd=1