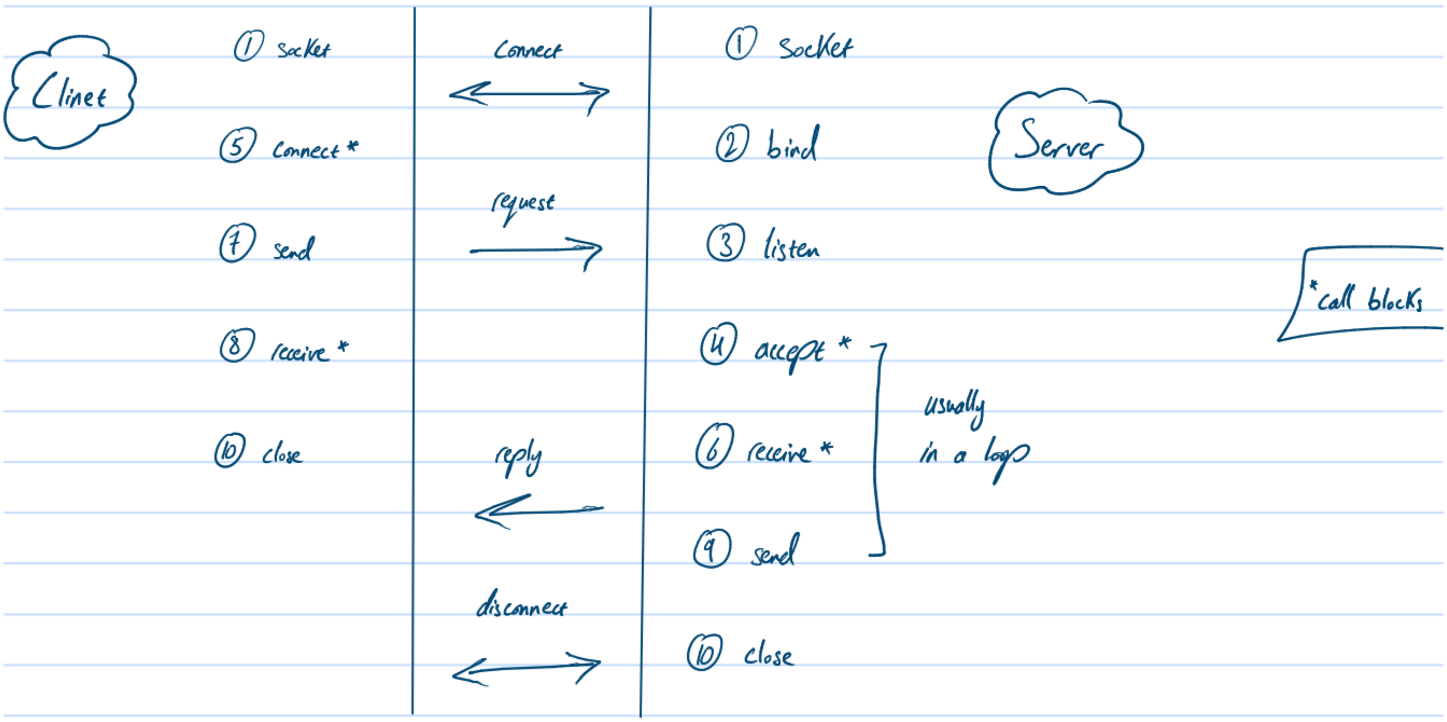
Week 1

---

1. Statistical Multiplexing: sharing of network bandwidth between users according to the statistics of their demand.
1. Multiplexing == Sharing
2. Can serve more users with the same size network
3. Users may have degraded service
2. Metcalfe's Law: the value of a network of N nodes is proportional to N^2
4. Think of it as a graph (number of edges for a fully connected graph = $\frac{n\,(n-1)}{2}$)
5. Robert Metcalfe is the inventor of the Ethernet (he also founded 3Com)

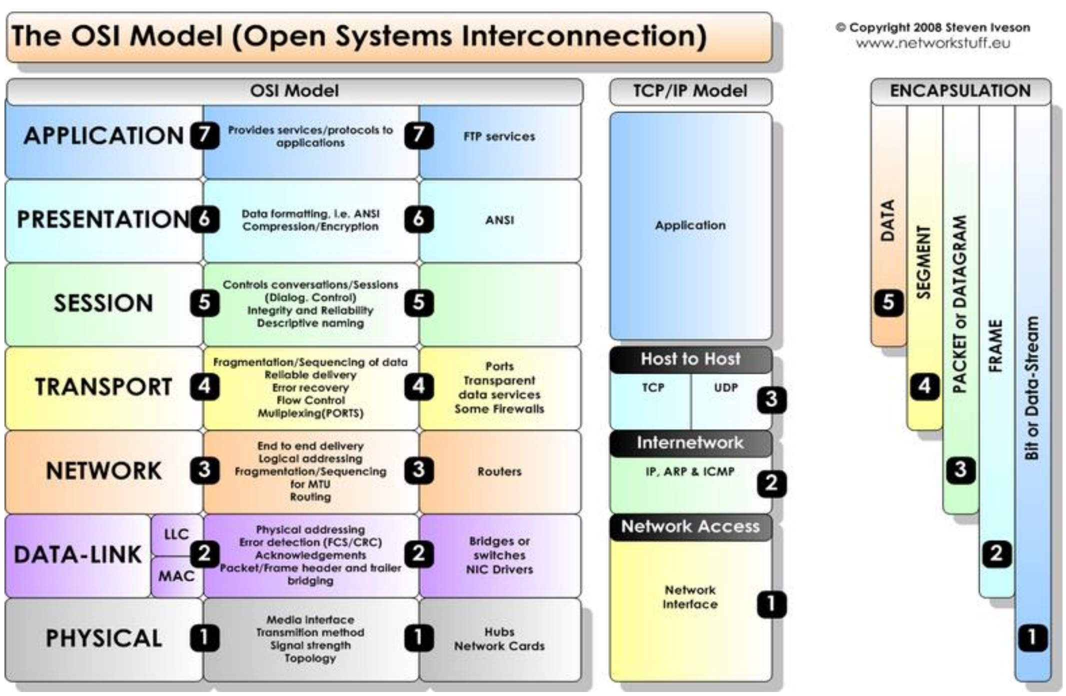
3. A node on the network can be a
    3.1. Host
    3.2. Router, switch
4. Types of links
    4.1. Full duplex (e.g., wired connection)
    4.2. Half duplex (e.g., wireless connection)
    4.3. Simplex (i.e., in one direction only)
5. In wireless links, messages are broadcasted for all to hear (both for the access point and the host)
6. Examples of Networks
    6.1. Wi-Fi
    6.2. Enterprise / Ethernet
    6.3. ISP
    6.4. Cable / DSL
    6.5. 2G, 3G, …
    6.6. Bluetooth
    6.7. Satellite
    6.8. Telephone
6. Internet == internetwork
7. Traceroute == display path to IP address
8. Programs communicate on a network by using the socket API
7. Network names by scale
    7.1. PAN → personal area network
    7.2. LAN → local area network
    7.3. MAN → metropolitan area network
    7.4. WAN → wide area network
    7.5. The Internet

8. Client / Server connection timeline

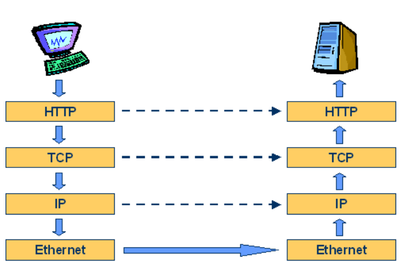


9. Protocols are horizontal. While layers are vertical.

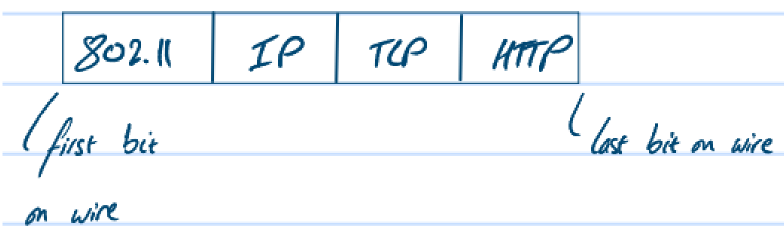
9.1. Layers



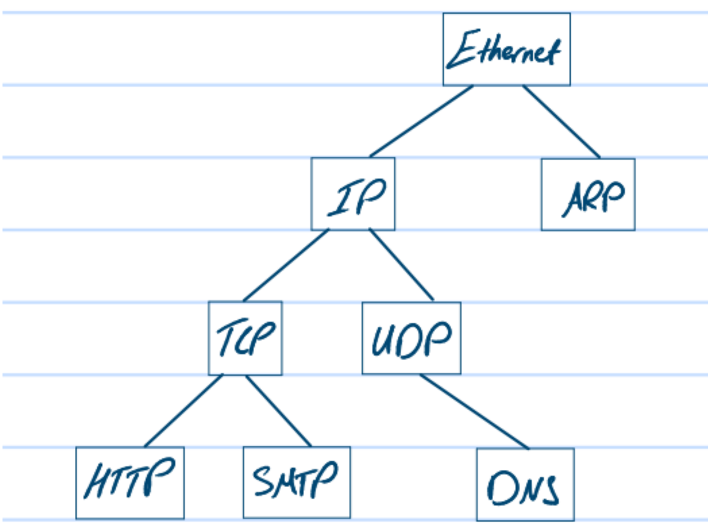
9.2. Protocols



10. The set of protocols in use is called a protocol stack
11. Example of protocols
    11.1. TCP
    11.2. IP
    11.3. 802.11 (i.e., wireless)
    11.4. Ethernet
    11.5. HTTP
    11.6. SSL
    11.7. DNS
12. Encapsulation: each lower layer encapsulates the higher layer
13. Each layer adds its own header



14. Protocol De-Multiplexing
15. Uses De-Multiplexing keys in the headers



16. Layering
    16.1. Advantages
        16.1.1. Information hiding and reuse
        16.1.2. Using information hiding to connect to different systems (protocols)
    16.2. Disadvantages
        16.2.1. Adds overhead (minor for long messages)
        16.2.2. Hides information (e.g. wired or wireless?)
17. Reference Models
    17.1. OSI (7 layers, previously mentioned)
    17.2. TCP/IP model (4 layers, previously mentioned)

17.3. Internet reference model (roughly speaking, what is commonly used in practice)

application layer
transport layer
Internet layer
link layer

18. Naming convention of information across different layers
    18.1. Physical layer → bits
    18.2. Link layer → frame
    18.3. Network layer → packet
    18.4. Transport layer → segment
    18.5. Application layer → message

Week 2

---

1. Course reference model

Application
Transport
Network
Link
Physical

2. Simple link properties
   2.1. Rate (bandwidth = bits / seconds)
   2.2. Delay (related to length = seconds)
3. Data propagates at around $\frac{2}{3}$ speed of light in a wire
4. Latency: the delay to send a message over a link

$$\text{Latency} = \frac{M}{R} + D$$

message bits (M), rate (R), $D = \frac{\text{length}}{\frac{2}{3}c}$

   4.1. Transmission delay: time to put a message “on a wire”
   4.2. Propagation delay: time for bits to propagate “across a wire”

5. Message bits take up space on the wire

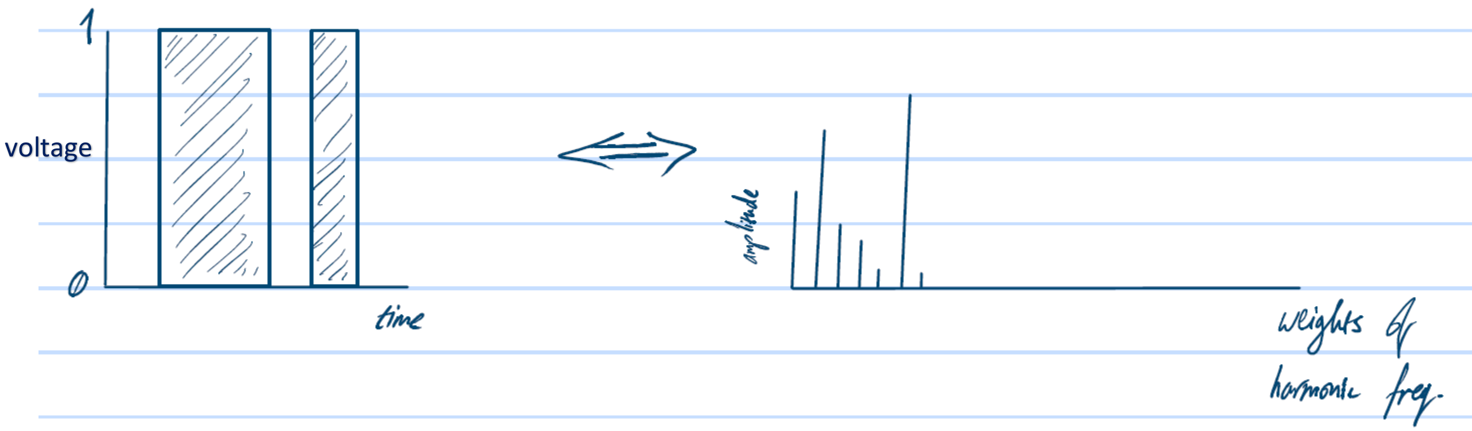
bandwidth – delay product (bits)

$$BD = R \cdot D$$

6. Types of signals
   6.1. Analog == continuous
   6.2. Digital == discrete

7. Types of media which propagate signals that carry bits of information
    7.1. Wire
    7.2. Fiber
    7.3. Wireless
8. A signal over time can be represented by its frequency components

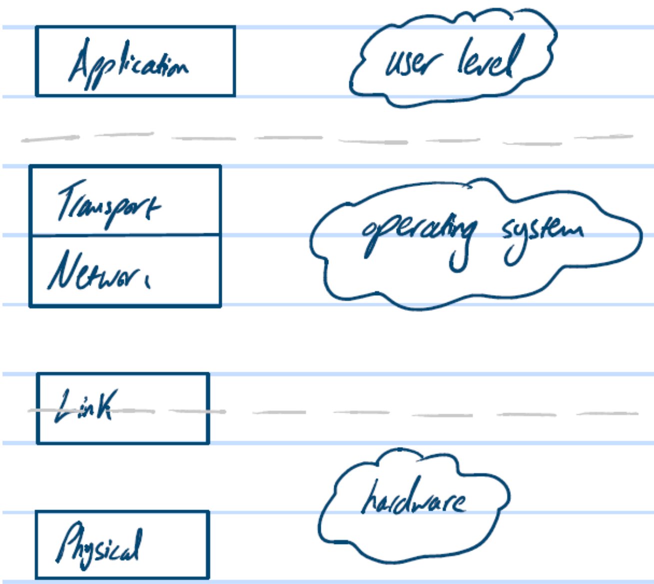
$$g(t) = \frac{1}{2} c + \sum_{n=1}^{\infty} a_n \sin(2\pi n f t) + \sum_{n=1}^{\infty} b_n \cos(2\pi n f t)$$



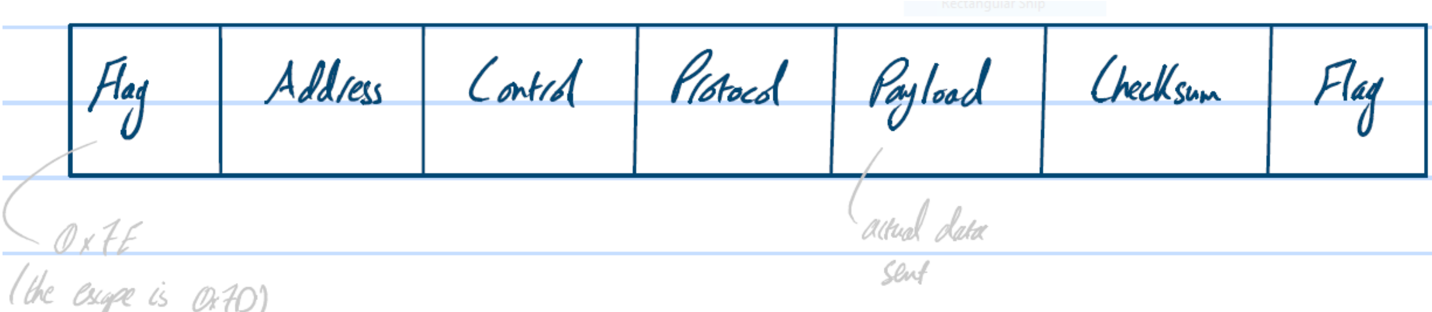
9. The perfect channel has just about enough bandwidth to transmit signals, since a higher bandwidth will be wasted
    9.1. EE bandwidth → width of frequency band (in Hz)
    9.2. CS bandwidth → information carrying capacity ($bits$ / sec)
10. Signal transmission across
    10.1. Wire
        10.1.1. Signal is delayed ($\frac{2}{3}$ $speed\ of\ light$)
        10.1.2. Signal is attenuated over distance
        10.1.3. Signal is attenuated if a frequency is above bandwidth (on Hz)
        10.1.4. Noise is added to the signal
    10.2. Fiber
        10.2.1. Very long delay
        10.2.2. Very low loss
        10.2.3. Travels in very wide frequency bands
    10.3. Wireless
        10.3.1. Signal travels at the speed of light
        10.3.2. Signal spreads out
        10.3.3. Signal attenuates faster than $\frac{1}{distance^2}$
        10.3.4. Signal interference (spatial reuse)
        10.3.5. Propagation depends on the environment
        10.3.6. Same signal can be received from multiple paths (i.e., wireless multipath)
11. Modulation: how signals represent bits of information across a media
12. Simple modulation (NRZ, non-return to zero):
    12.1. 1 → +v
    12.2. 0 → -v
13. Note: we can also use more signal levels (e.g., 4 levels is 2 bits per symbol)

14. Clock recovery problem: data may exhibit long sequences of 1's and 0's, therefore the receiver may be *confused* about the actual number of 1's and 0's received.
15. Solution: receiver needs frequent signal transitions to decode bits
16. Clock recovery (4B / 5B): map every 4 data bits into 5 code bits without long runs of zeros (solves long sequences of zeros)
17. NRZI: invert signal level on a 1 to break up long runs of 1's (solves long sequences of 1's)
18. In passband modulation, a carrier is a signal oscillating at a desired frequency. It can be modulated by changing:
    18.1. Amplitude
    18.2. Frequency
    18.3. Phase
19. There are two main limits for a channel:
    19.1. Nyquist limit
    19.2. Shannon capacity
20. Key channel properties:
    20.1. Bandwidth → measured at source
    20.2. Signal strength → measured at destination
    20.3. Noise strength → measured at destination
21. SNR == signal to noise ration
22. Capacity == the maximum information carrying rate of the channel
23. Nyquist limit (does not consider noise)
    23.1. R == rate
    23.2. B == bandwidth
    23.3. V == signal levels
    23.4. $R = 2\,B\,\log_2 V$ *(bits/sec)*
24. Shannon capacity
    24.1. C == capacity
    24.2. B == bandwidth
    24.3. S == signal
    24.4. N == noise
    24.5. $C = B\,\log_2(\frac{S+N}{N})$ *(bits/sec)*
25. Wired vs. Wireless perspective
    25.1. Wires and fiber → engineer SNR for data rate
    25.2. Wireless → adapt data rate to SNR
26. DSL == digital subscriber line
27. ADSL == asymmetric DSL
28. DSL properties:
    28.1. Reuses twisted pair telephone lines
    28.2. If you're close to local exchange → high SNR
    28.3. If you're far from local exchange → low SNR
    28.4. Uses passband modulation
    28.5. Has separate bands for upstream (small) and downstream (large)
    28.6. Modulation varies both amplitude and phase

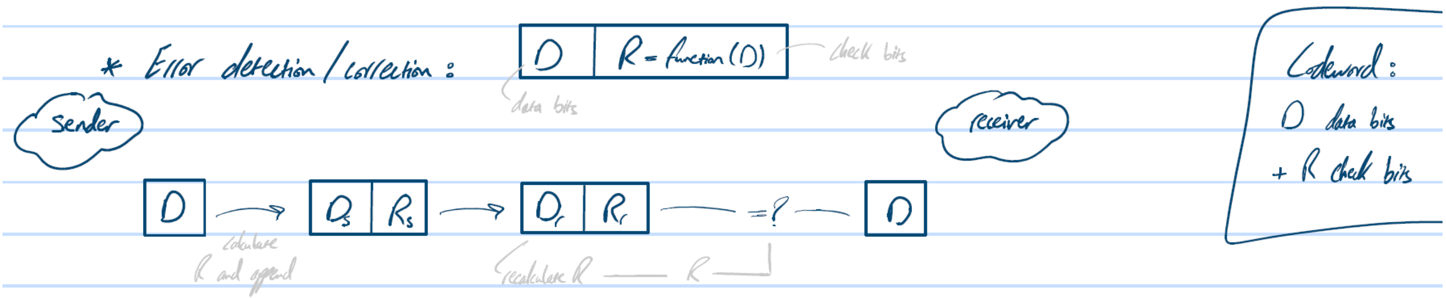
29. Typical implementation of layers



30. Framing: used to interpret the stream of bits that come out of the physical layer
31. In practice, physical layer helps to identify frame boundaries
32. Framing methods:
    32.1. Byte count
    32.2. Byte stuffing
    32.3. Bit stuffing
33. Byte count:
    33.1. Start each frame with a count field
    33.2. Difficult to re-synchronize after framing error
34. Byte stuffing:
    34.1. Adds head and tail flag bytes
    34.2. Escape flag if it occurs inside frame
    34.3. Needs to escape the escape
35. Bit stuffing:
    35.1. On transmit, after five 1’s in data, insert a zero
    35.2. On receive, a zero after five 1’s is deleted
    35.3. Has better performance than “byte stuffing” but not practical
36. Link example: PPP over SONET
37. PPP: a point to point protocol that is used to frame IP packets that are sent over SONET optical links
38. Uses a byte stuffing method
    38.1. To stuff a byte, prepend esc an XOR byte with 0x20
    38.2. To un-stuff a byte, remove esc an XOR byte with 0x20

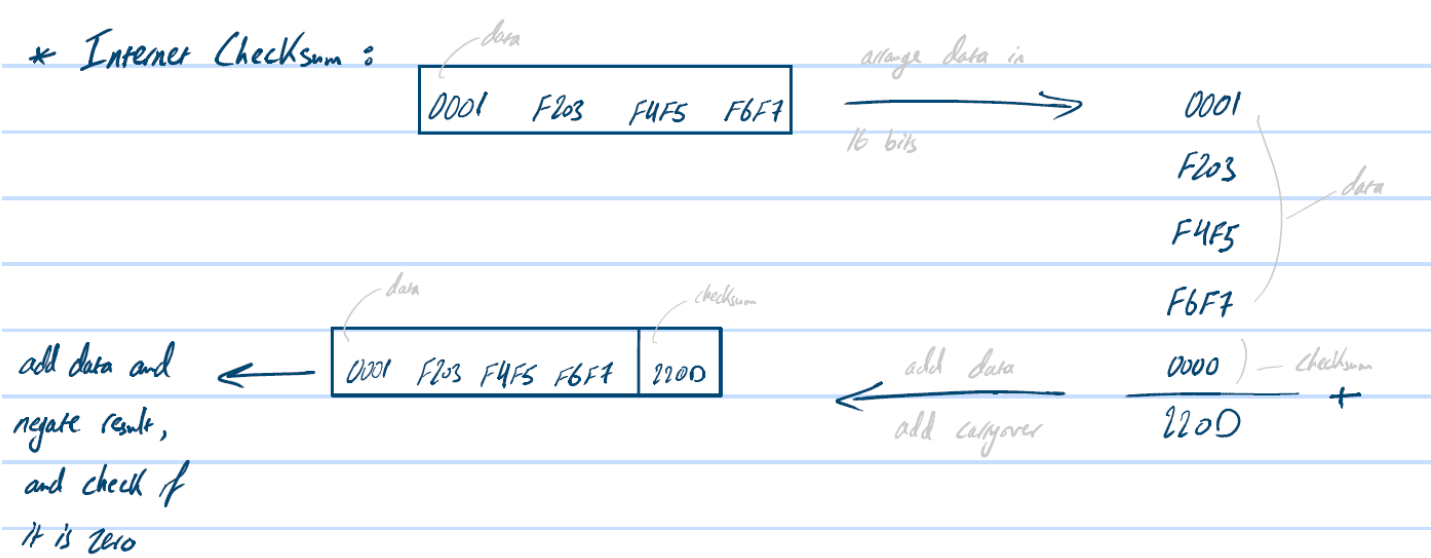


39. Error detection / correction:

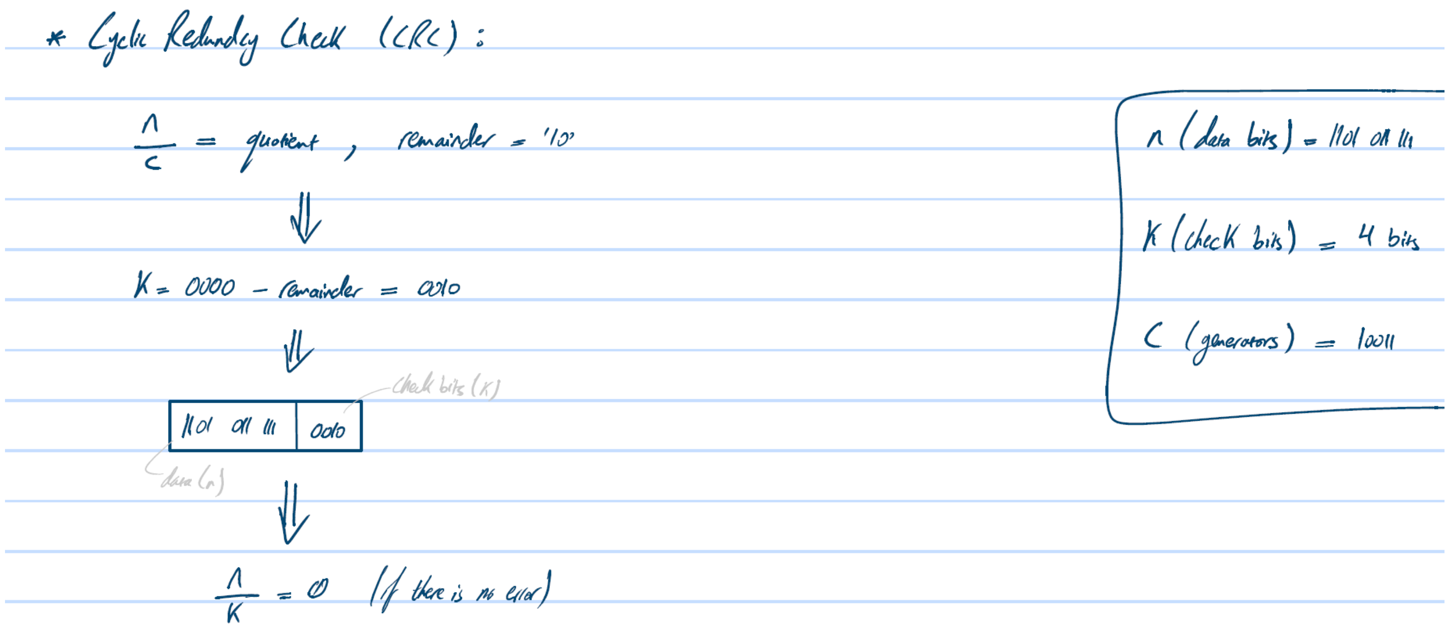


40. Distance: the number of bit flips needed to change D_1 to D_2
41. Hamming distance: minimum distance between D_1 and D_2
42. Hamming error detection: for a code of distance (d+1), up to (d) errors will always be detected
43. Hamming error correction: for a code of distance (2d+1), up to d errors can always be corrected by mapping to the closest code word
44. Error detection methods:
    44.1. Parity
    44.2. Checksum
    44.3. CRC
45. Parity bit:

46. Internet checksum:



47. Cyclic redundancy check (CRC):

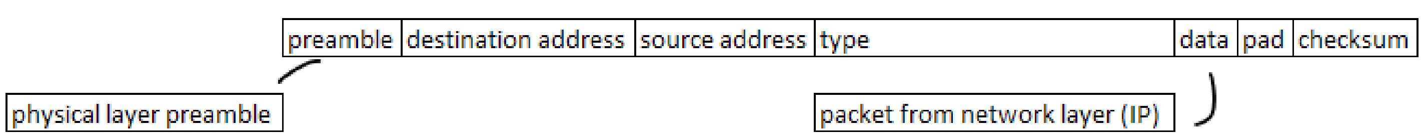


48. Usage:
   48.1. Parity → not widely used
   48.2. Checksum → IP, TCP, UDP, …
   48.3. CRC → Ethernet, 802.11, ADSL, Cable, …

Week 3

---

1. Reliability should exist in all layers of a network
2. Automatic repeat request (ARQ): used when errors are common or must be corrected (e.g., Wi-Fi, TCP)
    - 2.1. Rules:
        - 2.1.1. Receiver automatically acknowledges correct frames with an ACK (acknowledgment)
        - 2.1.2. Sender automatically resends after a timeout, until an ACK is received
    - 2.2. Issues:
        - 2.2.1. How long to set the timeout? (easy on LAN, difficult over the Internet)
        - 2.2.2. How to avoid accepting duplicate frames as new frames?
    - 2.3. Goal:
        - 2.3.1. Performance in the common case, correctness always
    - 2.4. Duplicate cases:
        - 2.4.1. ACK is lost
        - 2.4.2. Timeout is early
    - 2.5. Duplicates Solution:
        - 2.5.1. Stop-and-wait ARQ → single bit, good for LAN
        - 2.5.2. Sliding window → generalization of stop-and-wait
3. Types of multiplexing:
    - 3.1. Time division multiplexing (TDM): users take turns on a fixed schedule
    - 3.2. Frequency division multiplexing (FDM): put different users on a different frequency bands
    - 3.3. TDM / FDM are suited for fixed traffic, fixed number of users (e.g., TV, radio, GSM)
    - 3.4. Multiple access schemes: multiple users according to their demands
        - 3.4.1. Random multiple access
            - 3.4.1.1. ALOHA protocol:
                - 3.4.1.1.1. node just sends when it has traffic, if there was a collision (no ACK received) then wait a random time and resend
                - 3.4.1.1.2. not efficient under high load (too many collisions)
            - 3.4.1.2. Classic Ethernet has:
                - 3.4.1.2.1. CSMA (carrier sense multiple access)
                - 3.4.1.2.2. CSMA / CD (with collision detection)
                - 3.4.1.2.3. BEB (binary exponential back off)
                - 3.4.1.2.4. CRC-32 is used for error-detection; no ACKs or retransmission
                - 3.4.1.2.5. Modern Ethernet is based on switches, not multiple access



            - 3.4.1.3. Wireless multiple access (Wi-Fi)
                - 3.4.1.3.1. Wireless complications:
                    - 3.4.1.3.1.1. Nodes may have different areas of convergence
                    - 3.4.1.3.1.2. Nodes can't receive while sending (can't collision detect)
                - 3.4.1.3.2. MACA (MAC, multiple access control protocol, with collision avoidance): solves the problems of "hidden terminals" and exposed terminals. Steps:

3.4.1.3.2.1. A sender transmits a RTS
3.4.1.3.2.2. A receiver replies with a CTS
3.4.1.3.2.3. Sender transmits the frame while nodes leaving the CTS stay silent

3.4.1.3.3. 802.11 physical layer
3.4.1.3.3.1. Uses 20/40 MHz channels on ISM bands
3.4.1.3.3.2. Uses OFDM modulation (except 802.11 b)
3.4.1.3.3.2.1. Different amplitudes / phases for varying SNR's
3.4.1.3.3.2.2. Rates from 6 to 54 Mbps plus error correction
3.4.1.3.3.2.3. 802.11 n uses multiple antennas

3.4.1.3.4. 802.11 link layer:
3.4.1.3.4.1. Multiple access uses CSMA / CA; RTS/CTS is optional (by inserting small random gaps)
3.4.1.3.4.2. Frames are ACKed and retransmitted with ARQ
3.4.1.3.4.3. Errors are detected with a 32-bit CRC
3.4.1.3.4.4. Has many features (e.g., encryption, power save, …)

frame control	duration	address (1) (recepient)	address (2) (transmitter)	address (3) (next host)	sequence	data	check sequence

packet from network layer (IP)

3.4.2. Contention-Free multiple access (based on turns, not randomization)

3.4.2.1. Issues with random multiple access:
3.4.2.1.1. Under low load:
3.4.2.1.1.1. Grant immediate access
3.4.2.1.1.2. Few collisions
3.4.2.1.2. Under high load:
3.4.2.1.2.1. Access time varies
3.4.2.1.2.2. High collisions

3.4.2.2. Token ring: arrange nodes in a ring; token rotates "permission to send" to each node in turn. If n node has nothing to send, it just passes the token to the next node in the ring.
3.4.2.2.1. Turn-taking advantages:
3.4.2.2.1.1. Fixed overhead with no collisions
3.4.2.2.1.2. Regular chance to send with no unlucky nodes
3.4.2.2.2. Turn-taking disadvantages:
3.4.2.2.2.1. Complexity (more things can go wrong than random access protocols)
3.4.2.2.2.2. Higher overhead at low load

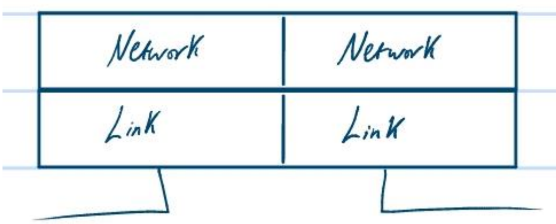
4. Hub or repeater

4.1. Inside a hub, all ports are wired together; more convenient and reliable than a single shared wire (a message is a broadcast)

5. Switch

5.1. Inside a switch, use frame addresses to connect input to correct output port; multiple frames may be switched in parallel
    5.1.1. Full duplex
    5.1.2. Need buffers for multiple inputs to send to one output
    5.1.3. Overload will fill buffer and lead to frame loss
5.2. Advantages of switches:
    5.2.1. Switches and hubs have replaced the shared cable of classic Ethernet
    5.2.2. Switches offer scalable performance
5.3. Switch forwarding is done using “backward learning”
    5.3.1. Backward learning: switch forwards frames with a port/address table as follows
        5.3.1.1. To fill table, look at the source address of input frames.
        5.3.1.2. To forward, send to the port of matching address in table. Else, broadcast to all ports.
        5.3.1.3. It also works with multiple switches and a mix of hubs assuming no loops.

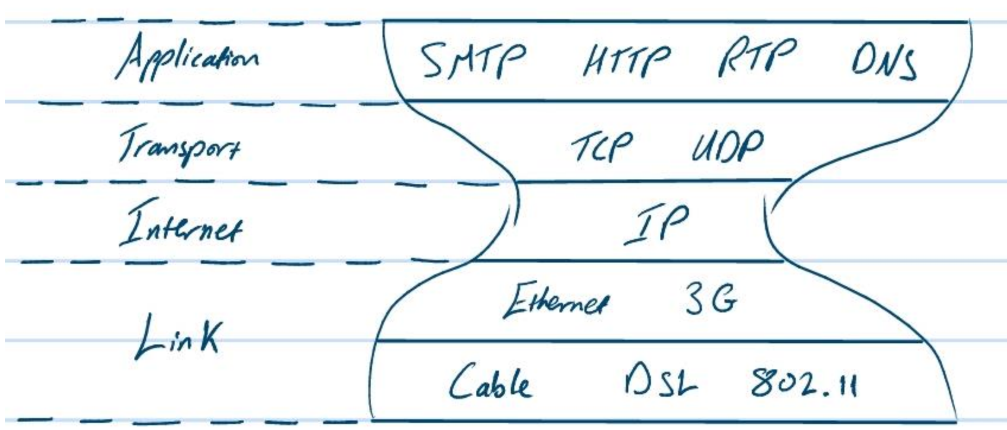
6. Router



Week 4

---

1. Shortcomings of switches:
   1.1. Don't scale to large networks (large tables)
   1.2. Don't work across more than one link layer technology
   1.3. Don't give much of traffic control
2. Routing: the process of deciding in which direction to send traffic
3. Forwarding: the process of sending packets on its way
4. Network service models:
   4.1. Datagrams / connectionless service (like postal letters, e.g., IP)
   4.2. Virtual circuits / connection oriented service (like telephone calls)
5. Datagram model:
   5.1. Packets contain a destination address (each router uses it to forward each packet, possibly on different paths)
   5.2. Each router has a forwarding table keyed by address
6. Virtual circuit model:
   6.1. Connection establishment (path is chosen, circuit information is stored in routers)
   6.2. Data transfer (packets are forwarded along the path)
   6.3. Connection teardown (circuit information is removed from routers)
   6.4. Notes:
      6.4.1. Packets only contain a short label to identify the circuit (labels don't have any global meaning, only unique for a link)
      6.4.2. Each router has a forwarding table keyed by circuit (gives output line and next label to place on packet)
      6.4.3. MPLS (multi-protocol label switching) is a virtual-circuit like technology widely used by ISP's
7. Both datagrams and virtual circuits use store-and-forward packet switching and they use statistical sharing of links
8. IP is used for internetworking (connecting different networks together)
9. Networks may differ in:
   9.1. Service models (datagrams, VC's)
   9.2. Addressing
   9.3. QOS
   9.4. Packet sizes
   9.5. Security
10. Internet reference model:



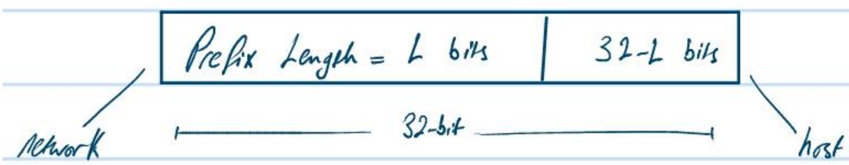
11. IPv4

IPv4 Header Format

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				IHL				DSCP						ECN		Total Length															
4	32	Identification																Flags			Fragment Offset												
8	64	Time To Live								Protocol								Header Checksum															
12	96	Source IP Address																															
16	128	Destination IP Address																															
20	160	Options (if IHL > 5)																															
24	192																																
28	224																																
32	256																																

11.1. Written in "dotted quad" notation (four 8-bit numbers separated by dots, e.g., A.B.C.D)

11.2. IP prefixes



11.2.1. They are written in "IP address / length" notation (e.g., 128.13.0.0/16 is from 128.13.0.0 to 138.13.255.255)

11.2.2. All addresses on one network belongs to the same prefix

12. IP forwarding:

12.1. A node uses a table that lists the next hop for prefixes

12.2. Prefixes in the table might overlap

12.3. Longer matching prefix rule:

12.3.1. For each packet, find the longest prefix that contains the destination address

12.3.2. Forward packet to the next hop router for that prefix

13. Host / router distinction:

13.1. Routers do the routing, know which way to all destinations

13.2. Hosts send remote traffic (out of prefix) to nearest router

14. Dynamic host control protocol (DHCP):

14.1. leases IP addresses to nodes

14.2. provides other parameters such as:

14.2.1. network prefix

14.2.2. address of local router

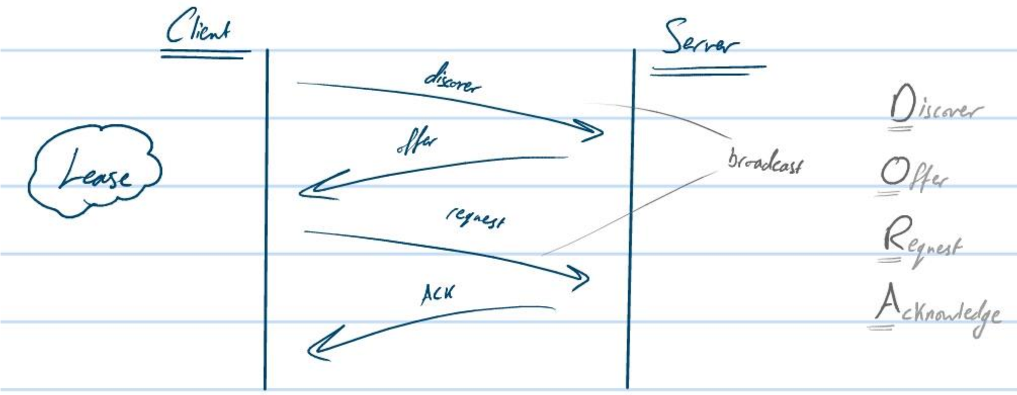
14.2.3. DNS server, Time server, …

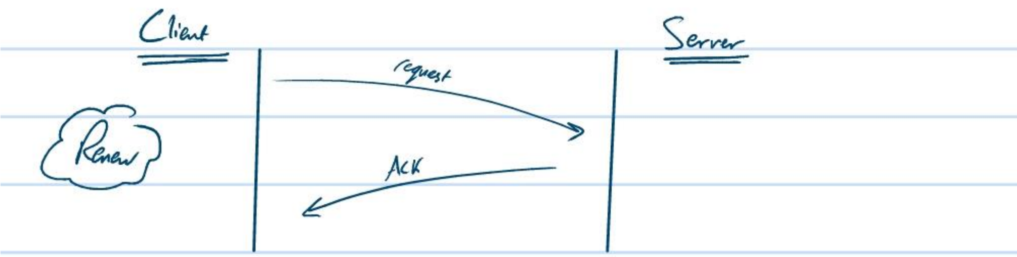
14.3. Is a client / server application (uses UDP ports 67, 68)

14.4. To get an IP, a node sends a broadcast message to all nodes on network:

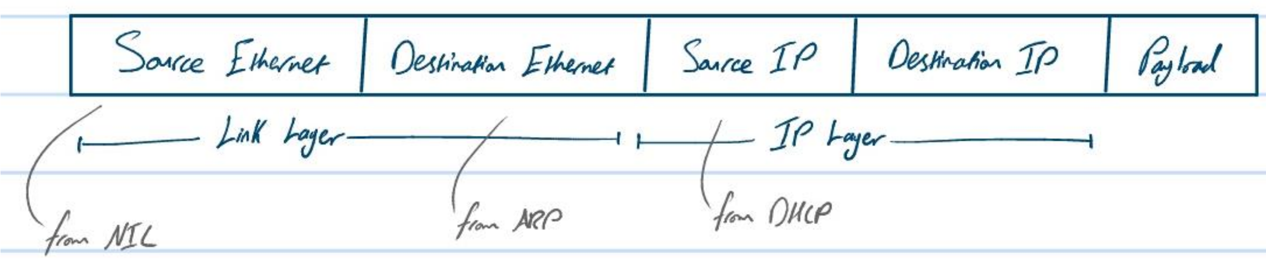
14.4.1. IP (32-bit) is 255.255.255.255

14.4.2. Ethernet (48-bit) is FF:FF:FF:FF:FF:FF

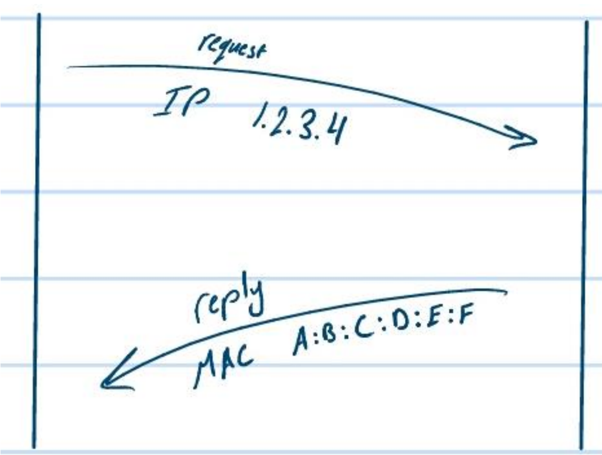




15. Address resolution protocol (ARP): used to map local IP address to its link layer address



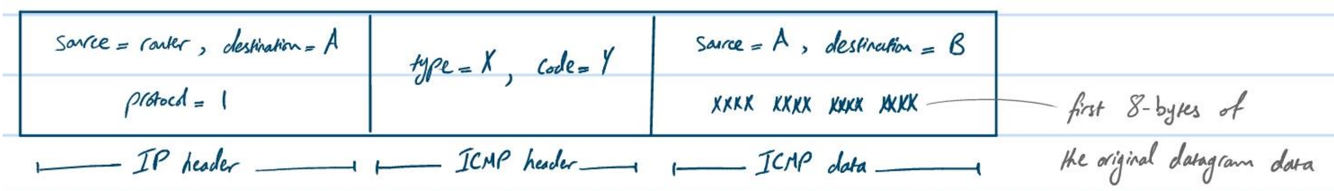
    15.1. Sits on top of link layer (no servers)
    15.2. Users broadcast to reach all nodes



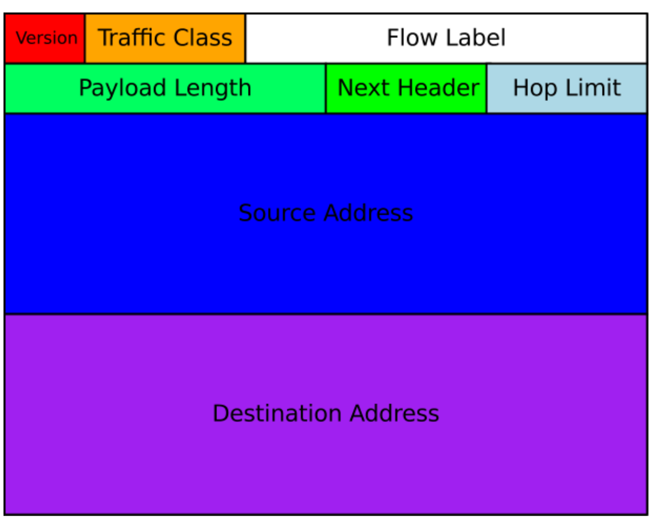
16. Different networks have different maximum packet sizes, or MTU. There are two solutions:
    16.1. Fragmentation
    16.2. Discovery
17. IPv4 fragmentation fields:

17.1. Identification
17.2. Fragment offset
17.3. MF (more fragments) / DF (don't fragment) control bits

18. IPv4 fragmentation procedure:
    18.1. Break into large pieces
    18.2. Copy IP header to pieces
    18.3. Adjust length or pieces
    18.4. Set offset to indicate position
    18.5. Set MF on all pieces except last
    18.6. Notes:
        18.6.1. Receiving hosts reassemble the pieces: identification field links pieces together, MF tells receiver when it has all pieces
        18.6.2. Fragmentation is undesirable due to:
            18.6.2.1. More work for routers, hosts
            18.6.2.2. Tends to magnify loss rate
19. Path MTU discovery: hosts test path with large packets and routers provide feedback if too large (e.g., traceroute)
    19.1. Depends on path so can change overtime
    19.2. Implemented with ICMP (set DF bit in IP header to get feedback messages)
20. Internet control message protocol (ICMP): a companion protocol to IP which provides error report and testing



    20.1. Sends an ICMP error report back to the IP source address upon error
    20.2. IP header contains a TTL field that is decremented on every router hop, with ICMP error if it hits zero. It also protects against forwarding loops
21. IPv6



    21.1. Features large addresses (128-bits)
    21.2. Has a new notation (8 groups of 4 hex digits)
    21.3. Example: 8329:2001:ODB8:000:000:000:FF00:0042
22. IPv6 transition methods:
    22.1. Dual stack (speak IPv4 and IPv6)

22.2. Translators (convert packets)
22.3. Tunnels (carry IPv6 inside / using IPv4)

23. Tunneling: carry IPv4 packets across IPv4 network

24. Middlebox: a computer networking device that transforms, inspects, filters, or otherwise manipulates traffic for purposes other than packet forwarding.
    24.1. Advantages:
        24.1.1. Possible rapid deployment path when there is no other option
        24.1.2. Control over many hosts
    24.2. Disadvantages:
        24.2.1. Breaking layering interferes with connectivity (strange side effects)
        24.2.2. Poor vantage point for many tasks
25. Network address translation (NAT) box: a middlebox that connects an internal network to an external network by translating addresses.
    25.1. How does it work?
        25.1.1. Keeps internal / external table
        25.1.2. Typically uses IP:Port
        25.1.3. Translates an internal IP:Port to an external IP:Port (and vise-versa) by looking up table and rewriting source / destination IP:Port
    25.2. Disadvantages?
        25.2.1. Can only send an incoming packet after an outgoing connection is set up
        25.2.2. Difficult to run servers or P2P applications
        25.2.3. Doesn't work well when there are two connections (UDP applications)
        25.2.4. Breaks applications that unwisely expose their IP addresses (such as FTP)
    25.3. Advantages?
        25.3.1. Relieves much IP address pressure
        25.3.2. Easy to deploy
        25.3.3. Useful functionality (firewalls, helps with privacy)

Week 5

---

1. A spanning tree provides basic connectivity, while routing uses all links to find "best" path.
2. Bandwidth allocation perspective:

Mechanism	Timescale / Adoption
load-sensitive routing	seconds / traffic hotspots
routing	minutes / equipment failures
traffic engineering	hours / network load
provisioning	months / network customers

3. Delivery models:
   3.1. Unicast
   3.2. Broadcast
   3.3. Multicast
   3.4. Anycast

   ➔ different routing is used for different delivery models

4. Routing algorithms goals:
   4.1. Correctness
   4.2. Efficient paths
   4.3. Fair paths
   4.4. Fast convergence
   4.5. Scalability
5. Routing algorithm rule: decentralized, distributed setting.
6. Possibilities of a "best" path:
   6.1. Latency
   6.2. Bandwidth
   6.3. Money
   6.4. Hops
   6.5. Hotspots

   ➔approximate "best" by a cost function that captures the needed factors

7. Shortest paths properties:
   7.1. Optimality property: sub paths of shortest paths are also shortest paths
   7.2. Sink tree: a sink tree for a destination is the union of all shortest paths towards the destination (similarly, source tree)

   ➔ only need to use destination to follow shortest paths

   ➔ each node only need to send to next hop

8. A forwarding table at a node lists next hop for each destination (routing table may know more)
9. Dijkstra's algorithm can be used for shortest path (but requires a complete topology)
   ➔Note: link-state algorithms are now typically used in practice
10. Distance vector routing:
    10.1. Simple early routing approach
    10.2. A distributed version of Bellman-Ford
    10.3. Has very slow convergence after failures
    10.4. Setting:
        10.4.1. Nodes know only the cost to their neighbors, not the entire topology
        10.4.2. Nodes can talk only to their neighbors using messages
        10.4.3. All nodes run the same algorithm concurrently
        10.4.4. Nodes and links may fail, messages may be lost
    10.5. Algorithm: each node maintains a vector of distances and next hops to all destinations
        10.5.1. Initialize vector with zero cost to self, infinity to other destinations
        10.5.2. Periodically send vector to neighbors
        10.5.3. Update vector for each destination by selecting the shortest distance heard, after adding cost of neighbor link
        10.5.4. Use best neighbor for forwarding
    10.6. Dynamics:
        10.6.1. Adding routes
        10.6.2. Removing routes
        10.6.3. Partitions are a problem ("count to infinity" scenario, can be solved using "split horizon", "poison reverse")
11. Routing information protocol (RIP):
    11.1. a distance vector protocol with hop count as a metric
    11.2. Includes split horizon, poison reverse
    11.3. Runs on top of UDP
12. Flooding: a simple mechanism that is used to broadcast a message to all nodes in the network
    12.1. Rules:
        12.1.1. Send an incoming message to all other neighbors
        12.1.2. Remember the message so that it is only flood once
    12.2. Remarks:
        12.2.1. Inefficient because a node may receive multiple copies
        12.2.2. Remembering a message is done using source and sequence numbers
        12.2.3. To make flooding reliable, use ARQ
13. Link state routing:
    13.1. Setting:
        13.1.1. Nodes know the cost to their neighbors, not the topology
        13.1.2. Nodes can talk to their neighbors using messages
        13.1.3. All nodes run the same algorithm concurrently
        13.1.4. Nodes / links may fail, messages may be lost
    13.2. Algorithm:
        13.2.1. Nodes flood their portion of the topology in the form of link state packets (LSP)
        13.2.2. Each node computes its own forwarding table (Dijkstra, or equivalent) since it has full topology by combining all LSP's
    13.3. Handling changes: on change, flood updated LSP's and re-compute routes
        13.3.1. Link failures:
            13.3.1.1. Both nodes notice, send updated LSP's
            13.3.1.2. Link is removed from topology

13.3.2. Node failure:
13.3.2.1. All neighbors notice a link has failed
13.3.2.2. Failed node can't update its own LSP (but all links to node are removed)
13.3.3. Addition of link or node:
13.3.3.1. Add LSP of new node to topology
13.3.3.2. Old LSP's are updated with new link
13.4. Complications:
13.4.1. Flooding sequence number reaches max, or is corrupted
13.4.2. Network partitions then heals
➔ Solution: include age on LSP's and forget old information that is not refreshed

14. Distance vector vs. Link state:

Goal	Distance Vector	Link State
fast convergence	slow, many exchanges	fast, flood and compute
scalability	excellent, storage / compute	moderate, storage / compute

15. Link-State protocols:
15.1. Intermediate system to intermediate system (IS-IS)
15.2. Open shortest path first (OSPF)

16. Multipath routing: allow multiple routing paths from node to destination be used at once
16.1. Topology has them for redundancy
16.2. Using them can improve performance

17. Equal cost multipath routes: a form of multipath routing that extends shortest path model by keeping a set if there are ties
17.1. With ECMP, source / sink "tree" is a DAG
17.2. Forwarding:
17.2.1. Randomly pick a next hop for each packet based on destination (balances load, but adds jitter)
17.2.2. Try to send packets from a given (source, destination) pair on the same path
17.2.2.1. (source, destination) pair is called a flow
17.2.2.2. Map flow identifier to next hop
17.2.2.3. No jitter within flow, but less balanced

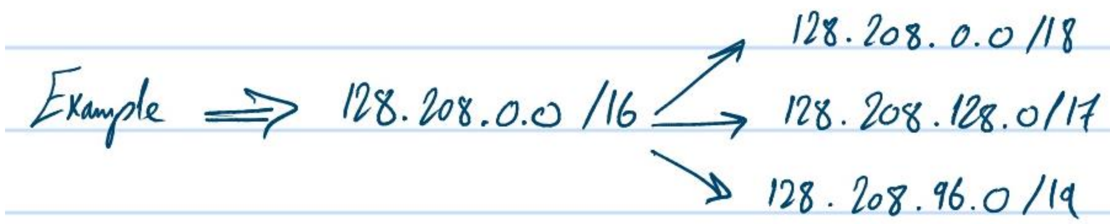
18. Impact of routing growth:
18.1. Forwarding table grow
18.2. Routing messages grow
18.3. Routing computations grow

19. Techniques to scale routing:
19.1. IP prefixes
19.2. Network hierarchy
19.3. IP prefix aggregation

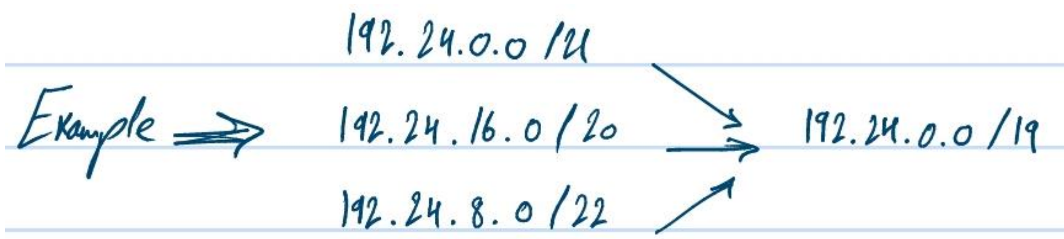
20. Hierarchical routing: route first to the region, then to the IP prefix within the region
20.1. Observations:
20.1.1. Possibility of longer paths
20.1.2. Outside a region, nodes have one route to all hosts within a region
20.1.3. Each node may have a different route to an outside region (routing decisions are still made by individual nodes, there's no single decision made by a region)

21. Subnets: internally split one large prefix into multiple smaller ones

➔ Note: routes can change prefix length without affecting hosts



22. Aggregation: externally join multiple smaller prefixes into one large prefix

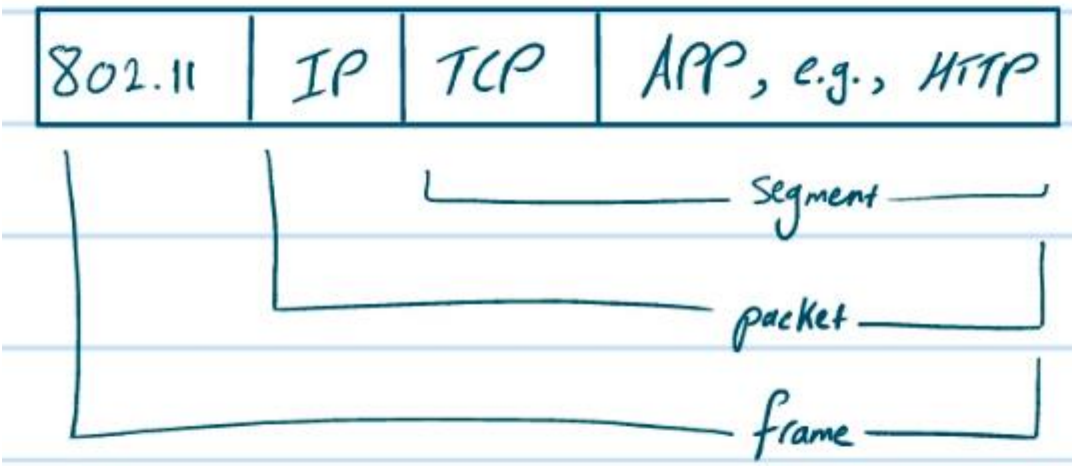


23. Structure of the Internet:
    23.1. Network group hosts as IP prefixes
    23.2. Networks are richly interconnected, often using IXP's
24. Internet-wide routing issues:
    24.1. Scaling to very large networks
    24.2. Incorporating policy decisions (letting different parties choose their routes to suit their own needs)
25. Effects of independent parties: selected paths are normally longer than overall shortest paths (and asymmetric too)
    ➔ Note: a sequence of independent goals and decisions, not hierarchy
26. Routing policies capture the goals of different parties. Common policies are:
    26.1. Transit
    26.2. Peer
27. Transit: An AS (autonomous system, e.g., ISP) gets transit service from another AS.
    27.1. AS accepts traffic for costumer from the rest of the Internet
    27.2. AS sends traffic from costumer to the rest of Internet
    27.3. Customer pays AS for the privilege
28. Peer: both AS's get peer service from each other
    28.1. Each AS accepts traffic from the other AS only for their customers
    28.2. AS do not carry traffic to the rest of the Internet for each other
    28.3. AS's don't pay each other
29. Border Gateway Protocol (BGP): the interdomain routing protocol used in the Internet
    29.1. Border routes of AS's announce BGP routes to each other
    29.2. Router announcements contain an IP prefix, path vector and next hop (path vector is a list of AS's on the way to the prefix; mainly to prevent loops)
    29.3. Implemented in two ways:
        29.3.1. Border routes of AS announce paths only to other parties who may use those paths
        29.3.2. Border routes of AS select the best path of the ones they hear in any, non-shortest way

Week 7

---

1. Segments carry application data across the network:



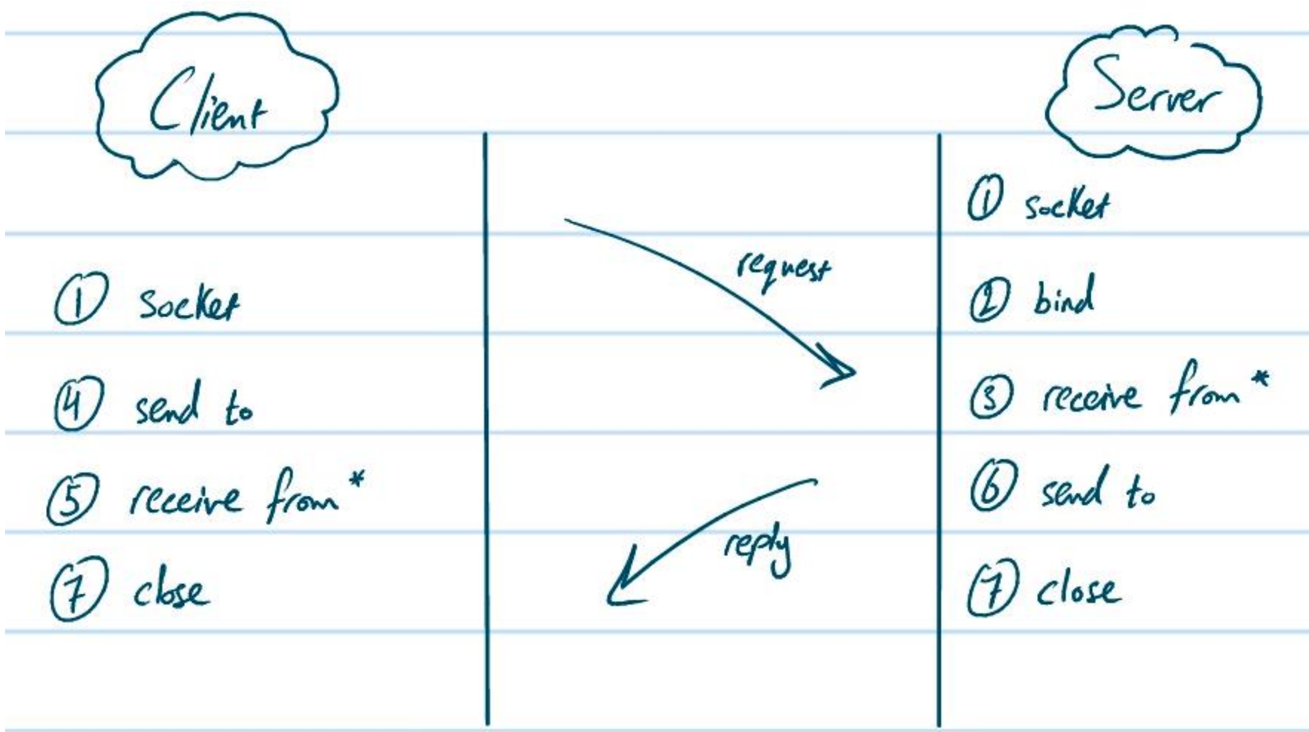
2. Comparison of Internet transports:

TCP	UDP
reliable	unreliable
bytestream	datagram
arbitrary length content	limited message size

3. The socket API can be used for TCP or UDP:
   3.1. Socket
   3.2. Bind
   3.3. Listen // streams
   3.4. Accept // streams
   3.5. Connect // streams
   3.6. Send (to) // datagrams
   3.7. Receive (from) // datagrams
   3.8. Close
4. An application process is identified by the IP, Protocol and Port
5. Port:
   5.1. They are 16-bit integers
   5.2. Servers often bind to “well known” ports (< 1024, which require admin privileges)

5.3. Clients often assigned "ephemeral" ports

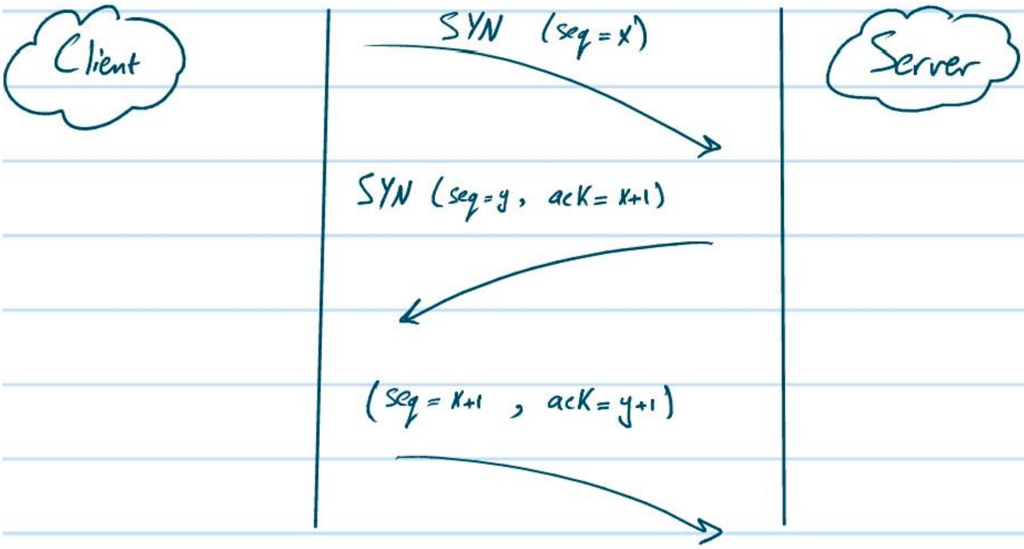
6. Examples of well-known ports:
    6.1. 20, 21 → FTP
    6.2. 22 → SSH
    6.3. 25 → SMTP
    6.4. 80 → HTTP
    6.5. 443 → HTTPS
7. UDP:
    7.1. Used by applications that don't want reliability or byte streams (VOIP, DNS, DHCP)
    7.2. Uses buffering with port (Mux / de-Mux)
    7.3. Uses ports to identify sending and receiving application process
    7.4. Has an optional checksum (zero means no checksum)



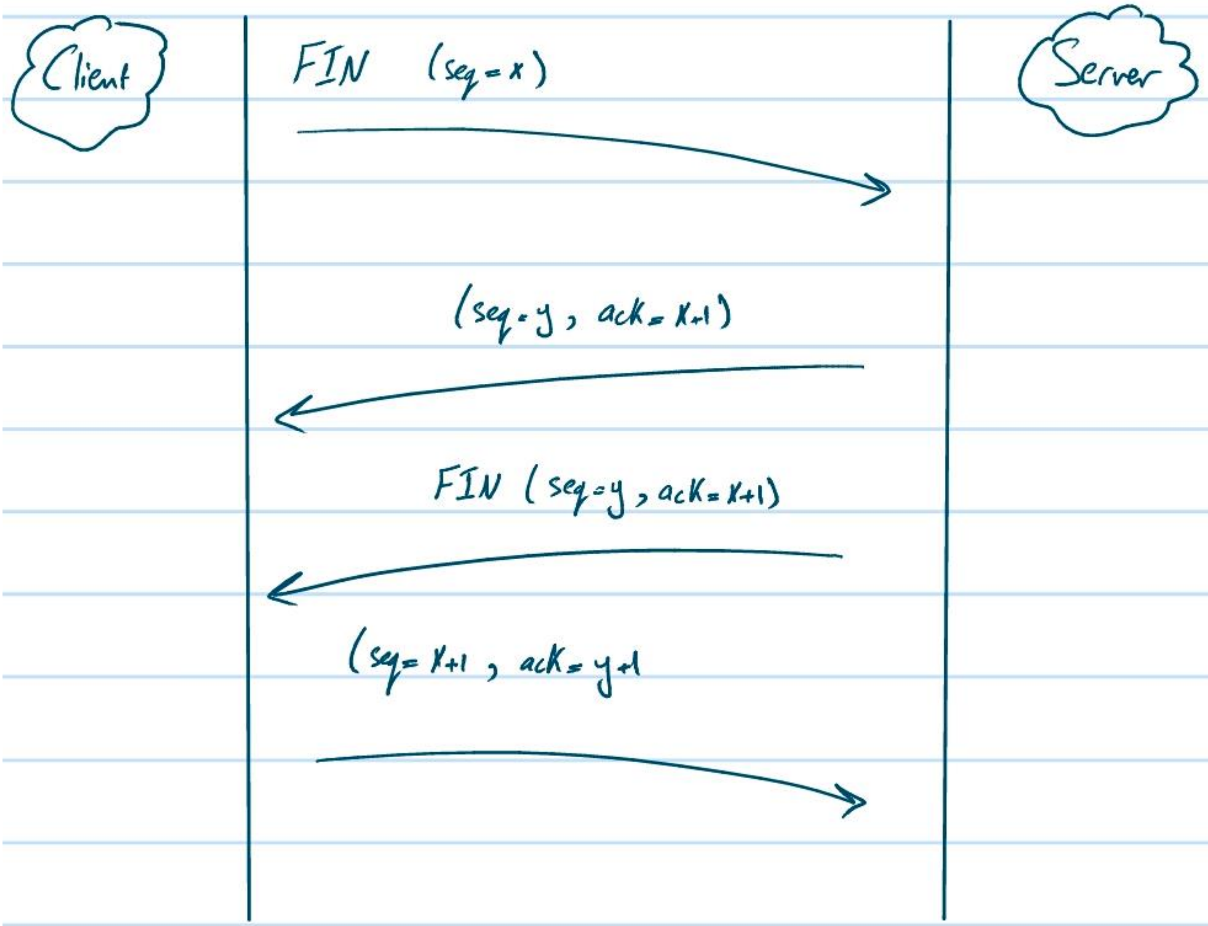
IPv6 Pseudo Header Format

<table>
<tr><th>Offsets</th><th>Octet</th><th colspan="8">0</th><th colspan="8">1</th><th colspan="8">2</th><th colspan="8">3</th></tr>
<tr><th>Octet</th><th>Bit</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th>16</th><th>17</th><th>18</th><th>19</th><th>20</th><th>21</th><th>22</th><th>23</th><th>24</th><th>25</th><th>26</th><th>27</th><th>28</th><th>29</th><th>30</th><th>31</th></tr>
<tr><td>0</td><td>0</td><td colspan="32" rowspan="4">Source IPv6 Address</td></tr>
<tr><td>4</td><td>32</td></tr>
<tr><td>8</td><td>64</td></tr>
<tr><td>12</td><td>96</td></tr>
<tr><td>16</td><td>128</td><td colspan="32" rowspan="4">Destination IPv6 Address</td></tr>
<tr><td>20</td><td>160</td></tr>
<tr><td>24</td><td>192</td></tr>
<tr><td>28</td><td>224</td></tr>
<tr><td>32</td><td>256</td><td colspan="32">UDP Length</td></tr>
<tr><td>36</td><td>288</td><td colspan="24">Zeroes</td><td colspan="8">Next Header</td></tr>
<tr><td>40</td><td>320</td><td colspan="16">Source Port</td><td colspan="16">Destination Port</td></tr>
<tr><td>44</td><td>352</td><td colspan="16">Length</td><td colspan="16">Checksum</td></tr>
<tr><td>48</td><td>384+</td><td colspan="32">Data</td></tr>
</table>

8. Transmission Control Protocol (TCP):
   8.1. 3-way handshake: opens connection for data in both directions



   8.2. Connection release:
      8.2.1. Delivers all pending data and "hangs up"
      8.2.2. Cleans up state in sender and receiver
      8.2.3. Both sides shutdown independently



9. Stop-and-Wait: ARQ with one message at a time

➔ Limitations: allows only one message to be outstanding from the sender (fine for LAN, not efficient for network paths with BD >> 1 packet)

➔ Example:

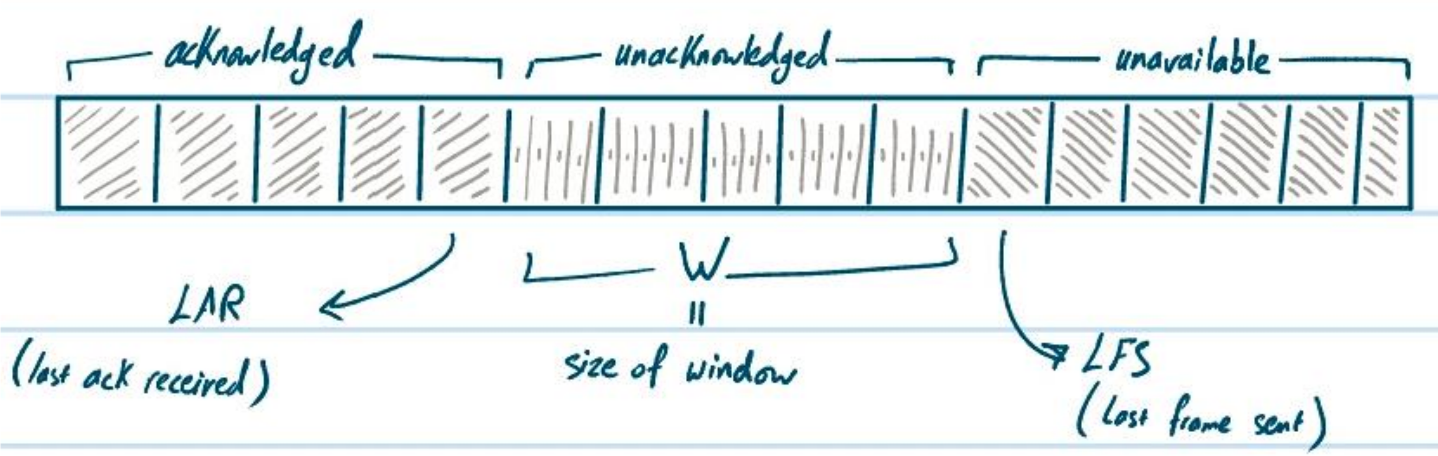
- R = 1 Mbps
- Packet size = 10 Kb
- D = 50 ms
- RTT = 100 ms
- Packets / sec = 10 packets / sec = 100 Kbps
- If R = 100 Mbps → 10 packets / sec = 100 Kbps

10. Sliding window: a generalization of stop-and-wait that allows to send W packets per RTT (W = min(2BD, recv_buf_size)

➔ Example:

- R = 1 Mbps
- Packet size = 10 Kb
- D = 50 ms
- W = 2BD = 100 Kbps
- If R = 10 Mbps → 1000 Kbps
- Note: send while LFS-LAR <= W

➔ Sender:

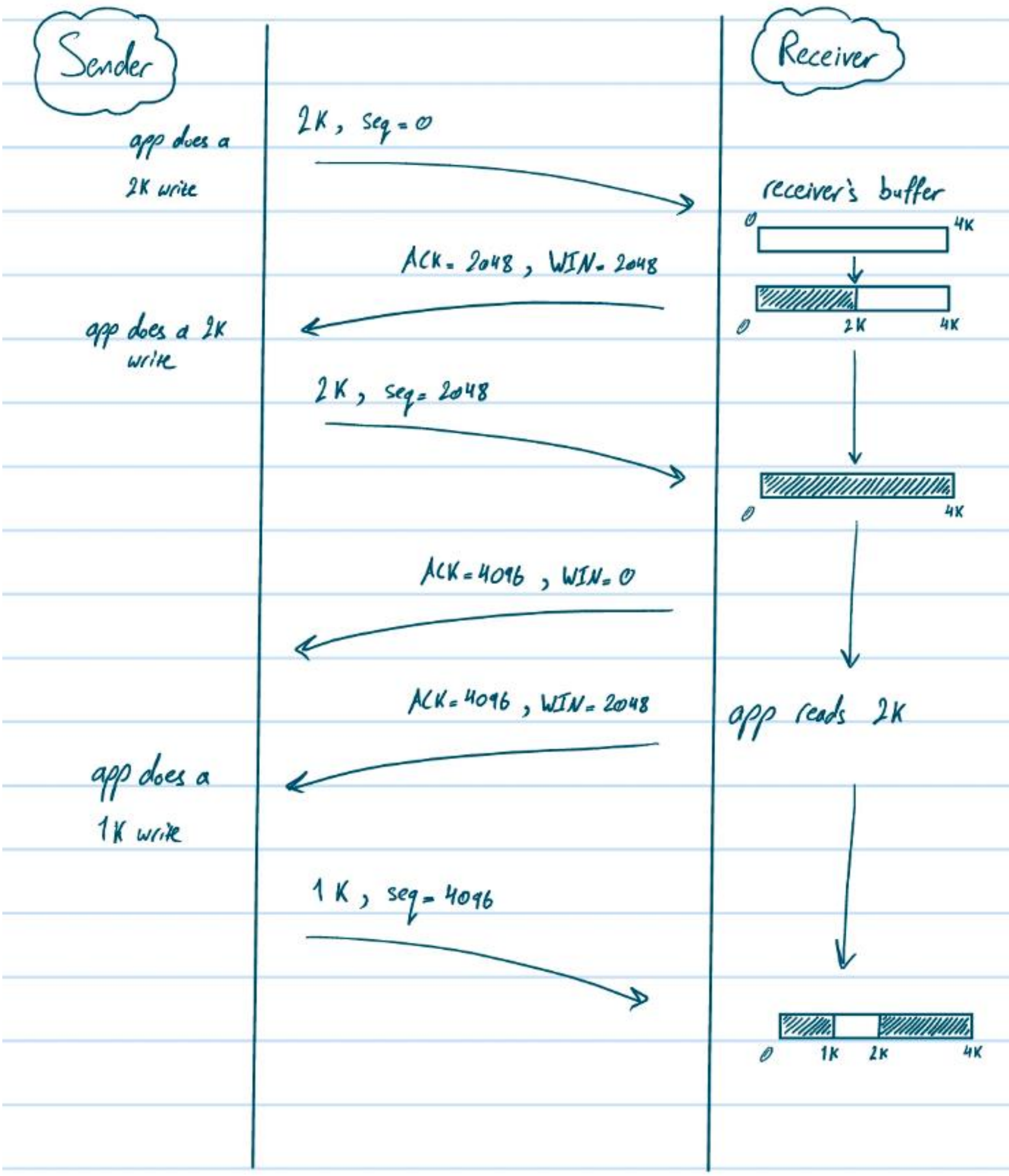


➔ Protocols:

10.1. Go-back-N: receiver only keeps a single packet, buffer for the next segment.
  10.1.1. On receive:
    10.1.1.1. If sequence number is LAS+1, accept and pass it to app, update LAS, send ACK
    10.1.1.2. Else, discard
10.2. Selective repeat:
  10.2.1. Receiver passes data to app in order, and buffers out of order segments to reduce retransmissions
  10.2.2. ACK conveys highest in order segment, plus hints about out of order segment
  10.2.3. TCP uses a selective repeat design
  10.2.4. On receive:
    10.2.4.1. Buffer segments [LAS+1, LAS+W]

10.2.4.2. Pass up to app in order segments from LAS+1 and update LAS
10.2.4.3. Send ACK for LAS regardless
10.2.5. Retransmission:
10.2.5.1. Go-back-N sender uses a single timer to detect losses
10.2.5.2. Selective repeat sender uses a timer per unACKed segment to detect losses
10.2.6. Sequence numbers:
10.2.6.1. For selective repeat, need W numbers for packets plus W ACKs of earlier packets
10.2.6.2. Typically implemented with an N-bit number that wraps around at $2^n - 1$

11. Flow control:
11.1. Sliding window uses pipelining to keep the network busy
11.2. Avoid loss at receiver by telling sender the available buffer space (WIN = number of acceptable segments)
11.3. Sender uses the lower of the sliding window and flow control window (WIN) as the effective window size
11.4. Note: the window only slides when the app calls recv(), not when the segment is ACKed



12. Retransmission timeouts: in a sliding window, the strategy for detecting loss is the timeout
    12.1. Set timer when a segment is sent
    12.2. Cancel timer when ACK is received
    12.3. If timer fires, retransmit data as if lost

➔ Problems:

- Too long wastes network capacity
- Too short leads to spurious resends
- Easy to set on LAN (short, fixed, predictable RTT)
- Difficult over Internet (wide range, variable RTT)

➔ Solution: adaptive timeout → keep smoothed estimates of the RTT and variance in RTT

➔ Laws:

- $SRTT_{N+1} = (0.9 * SRTT_N) + (0.1 * RTT_{N+1})$
- $SVar_{N+1} = (0.9 * SVar_N) + (0.1\ * |RTT_{N+1} - SRTT_{N+1})$
- $TCP\ Timeout_N = SRTT_N + (4 * SVar_N)$

13. TCP features:
    13.1. Message boundaries are not preserved from send() to recv()
    13.2. Bidirectional data transfer
    13.3. Control information (e.g., ACK) piggybacks on data segments in reverse direction
    13.4. “cumulative ACK” tells next expected byte sequence number (LAS+1)
    13.5. Selective ACK’s (SACK) give hints for receiver buffer state
    13.6. Sender uses an adaptive retransmission timeout to resend data from LAS+1
    13.7. Sender uses heuristics to infer loss quickly and resends to avoid timeouts (3 duplicate ACKs treated as a loss)
    13.8. Note: a single recv() call may receive several segments that were split to be sent
14. TCP header:

TCP Header

<table>
<tr><th>Offsets</th><th>Octet</th><th colspan="8">0</th><th colspan="8">1</th><th colspan="8">2</th><th colspan="8">3</th></tr>
<tr><th>Octet</th><th>Bit</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th>16</th><th>17</th><th>18</th><th>19</th><th>20</th><th>21</th><th>22</th><th>23</th><th>24</th><th>25</th><th>26</th><th>27</th><th>28</th><th>29</th><th>30</th><th>31</th></tr>
<tr><td>0</td><td>0</td><td colspan="16">Source port</td><td colspan="16">Destination port</td></tr>
<tr><td>4</td><td>32</td><td colspan="32">Sequence number</td></tr>
<tr><td>8</td><td>64</td><td colspan="32">Acknowledgment number (if ACK set)</td></tr>
<tr><td>12</td><td>96</td><td colspan="4">Data offset</td><td colspan="3">Reserved
0 0 0</td><td>N
S</td><td>C
W
R</td><td>E
C
E</td><td>U
R
G</td><td>A
C
K</td><td>P
S
H</td><td>R
S
T</td><td>S
Y
N</td><td>F
I
N</td><td colspan="16">Window Size</td></tr>
<tr><td>16</td><td>128</td><td colspan="16">Checksum</td><td colspan="16">Urgent pointer (if URG set)</td></tr>
<tr><td>20
...</td><td>160
...</td><td colspan="32">Options (if data offset > 5. Padded at the end with "0" bytes if necessary.)
...</td></tr>
</table>

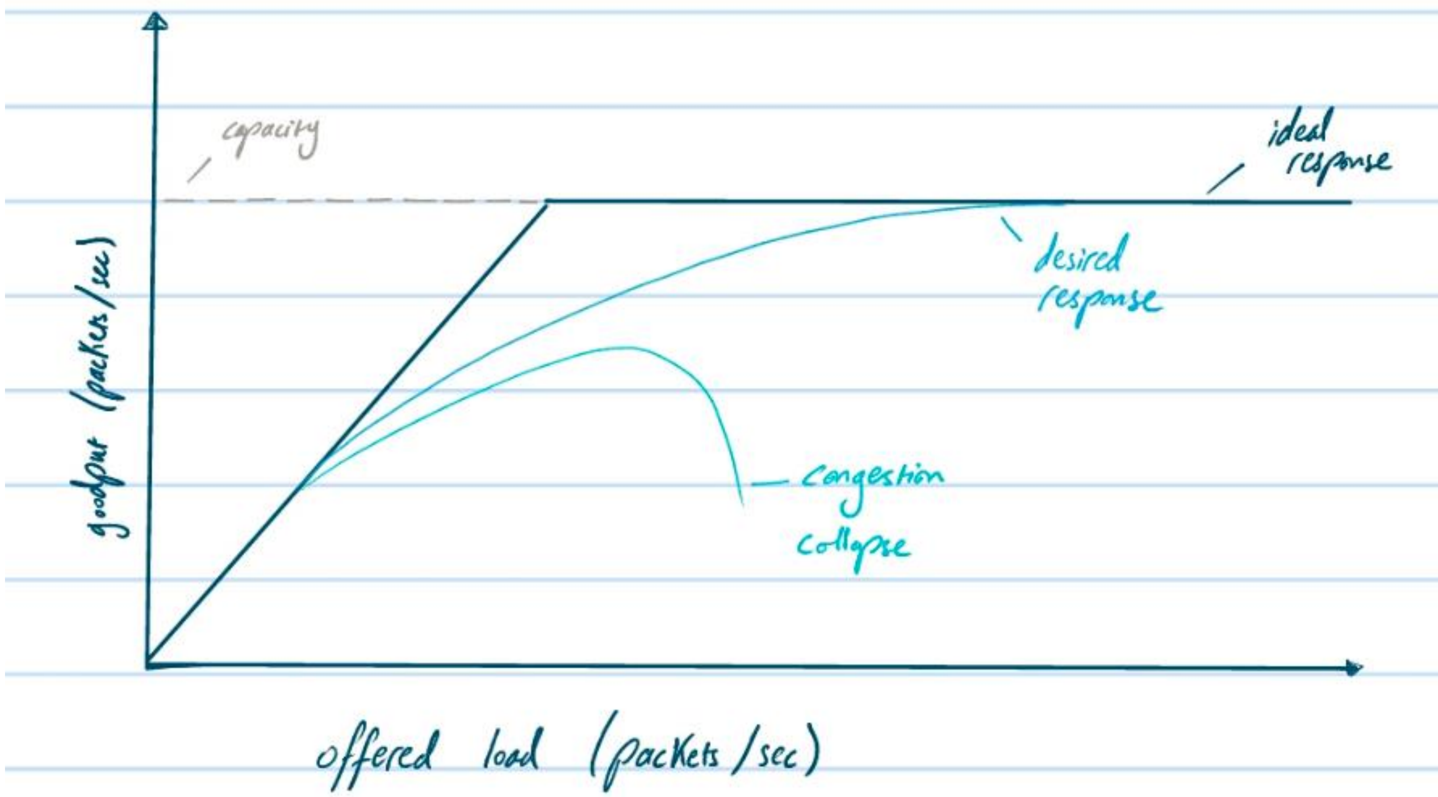
Week 8

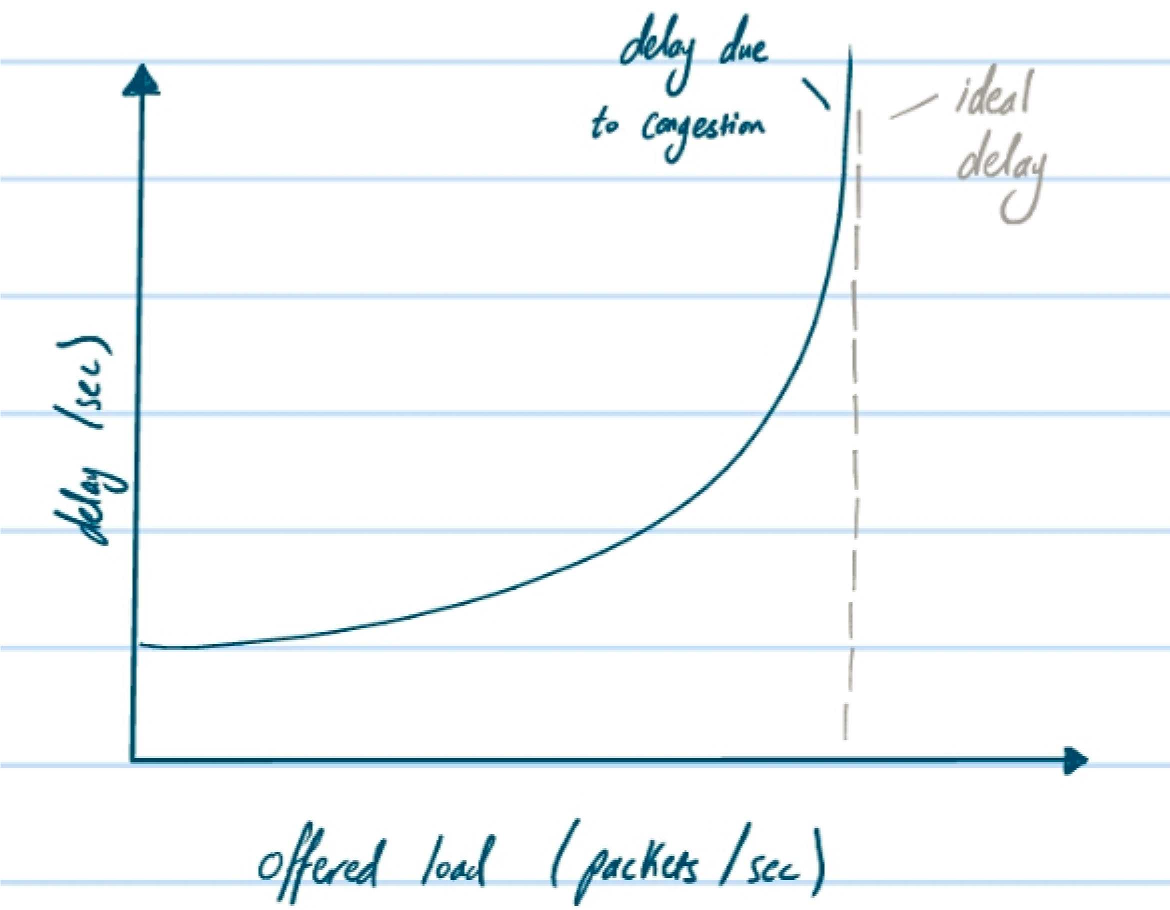
---

1. Congestion: a function of the traffic patterns which can occur even if every link have the same capacity

➔ Description of problem: congestion occurs when the internal buffers of routers / switches are full (buffer implemented using FIFO, discard when full)

→ Congestion collapse: due to spurious retransmissions





➔ Solution properties:

- Efficient → most network capacity is used, no congestion
- Fair → every sender gets a reasonable share of the network

➔ Key observation: in an effective solution, the network layer (can provide direct feedback) and the transport layer (can reduce offered load) must work together

2. Max-min fair allocation: increasing the rate of one flow will decrease the rate of another flow.

➔ Steps:

2.1. Start with all flows at rate zero
2.2. Increase the flow until there is a bottleneck in the network
2.3. Hold fixed the rate of the flows that are bottlenecked
2.4. Go to step 2 for any remaining flows

➔ Remark: allocation changes as flows start and stop

3. Bandwidth allocation models:
    3.1. Open loop / closed loop
        3.1.1. Open loop → reserve bandwidth before use
        3.1.2. Closed loop → use feedback to adjust rates
    3.2. Host / network support
        3.2.1. Who enforces allocation?

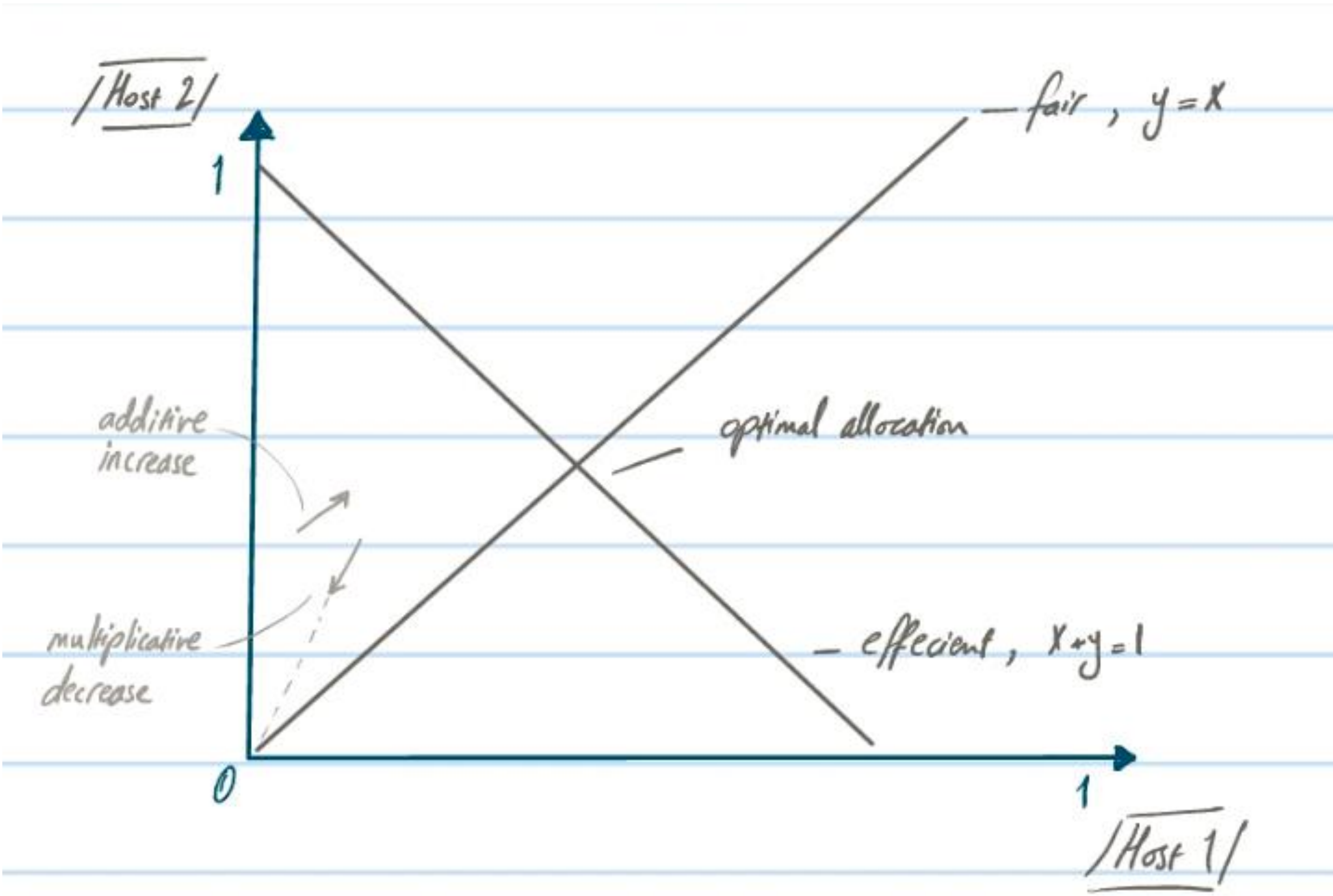
3.3. Window / rate based
    3.3.1. How is allocation is expressed?

➔ Note: TCP is a:

- Closed loop
- Host driven
- Window based

4. Additive increase multiplicative decrease (AIMD): a control law hosts can use to reach a good allocation
    4.1. Hosts additively increase rate while network is not congested
    4.2. Hosts multiplicatively decrease rate when congestion occurs
    4.3. Used by TCP
5. AIMD Rules:
    5.1. Hosts share a bottleneck but do not talk to each other directly
    5.2. Routers provide binary feedback if network is congested or not
6. AIMD game:

➔ produces a "saw tooth" pattern overtime for the rate of each host



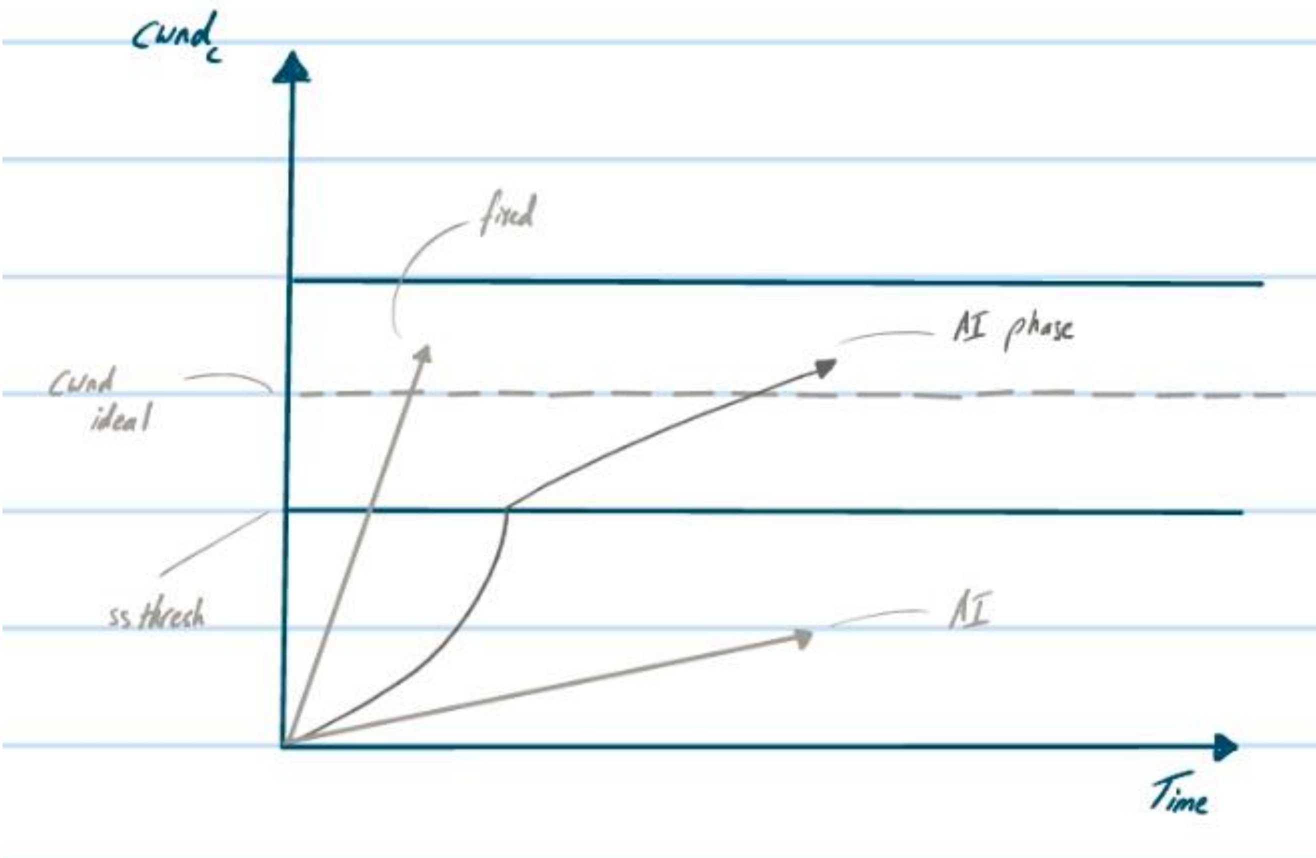
7. AIMD properties:
    7.1. Converges to an allocation that is both efficient and fair when hosts run it (other control laws do not)
    7.2. Requires only binary feedback from the network
8. AIMD feedback signals:

Signal	Example Protocol	Pros / Cons
packet loss	TCP NewReno Cubic TCP (Linux)	* hard to get wrong * hears about congestion late
packet delay	Compound TCP (Windows)	* hears about congestion early * needs to infer congestion
router indication	TCPs with explicit congestion notification	* hears about congestion early * requires router support

9. Van Jacobson:
   9.1. Widely credited with saving the Internet from congestion collapse in the late 1980's
   9.2. Introduced congestion control principles
   9.3. Provided practical solutions (TCP Tahoe / Reno)
10. TCP Tahoe / Reno
    10.1. Avoid congestion collapse without changing routers (or receivers)
    10.2. Fix timeouts and introduce a congestion window (cwnd) over the sliding window to limit queues / loss
    10.3. Implements AIMD by adapting cwnd using packet loss as the network feedback signal
11. ACK clocking:
    11.1. A sender injects a burst of segments into the network
    11.2. A queue forms in front of a slower speed link
    11.3. The slower link causes segments to spread
    11.4. The spread segments result in spread ACKs
    11.5. The speed ACKs end up clocking the source segments at the slower link rate

    ➔ the congestion window controls how many segments are inside the network (rate is roughly cwnd/RTT)

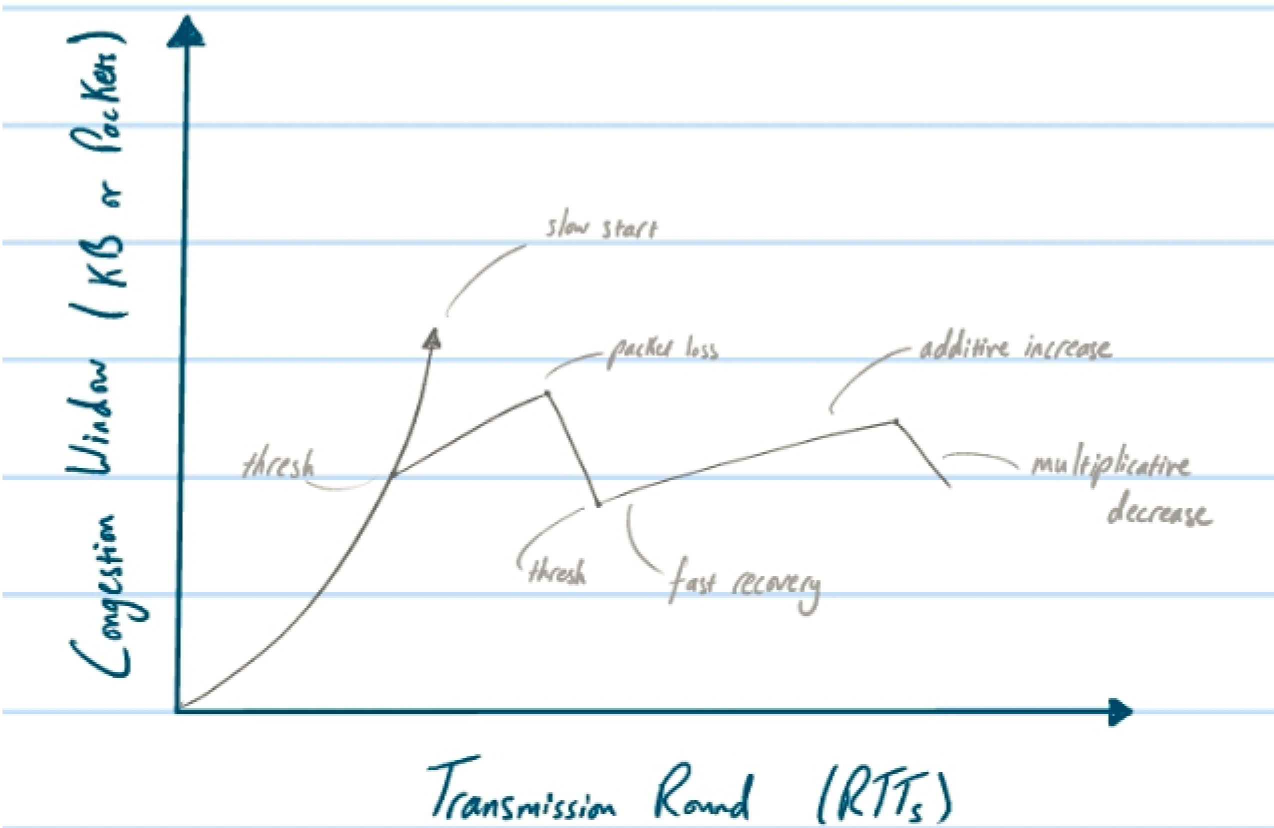
12. Congestion window: one of the factors that determines the number of bytes that can be outstanding at any time (maintained by sender). Not to be confused with the TCP window size (maintained by receiver).
13. Slow start: a component of the additive increase portion of AIMD.
    13.1. Sender uses a cwnd to send its rate (≈ cwnd/RTT)
    13.2. Sender uses packet loss as the network congestion signal
    13.3. Start by doubling cwnd every RTT (1,2,4,…)
    13.4. Expect loss for $cwnd_c \approx 2BD + queue$
    13.5. Use $ssthresh = \frac{cwnd_c}{2}$ to switch to AI



14. TCP Tahoe:
    14.1. Slow start phase
        14.1.1. Start with cwnd = 1 (or a small value)
        14.1.2. Cwnd += 1 packet per ACK
    14.2. AI phase
        14.2.1. Cwnd += 1/cwnd packets per ACK
        14.2.2. Roughly adds one packet per RTT
    14.3. Switching threshold
        14.3.1. Switch to AI when cwnd > ssthresh
        14.3.2. Set ssthresh = cwnd/2 after loss
        14.3.3. Begin with slow start after timeout (back to the beginning)
15. TCP implements the MD part of AIMD by using "fast retransmit" and "fast recovery"
16. Inferring loss from ACKs:
    16.1. TCP uses a cumulative ACK that carries the highest in-order sequence number
    16.2. Duplicate ACKs gives us hints about what data hasn't arrived (new data has arrived, but it wasn't the next segment)
17. Fast retransmit: treat three duplicate ACK's as a loss and retransmit the next expected segment (typically before a timeout)
18. Inferring non-loss from ACKs
    18.1. Each new duplicate ACK means that some new segment has arrived
    18.2. It will be the segments after the loss
    18.3. Thus advancing the sliding window will not increase the number of segments stored in the network
19. Fast recovery:
    19.1. First fast retransmit and MD cwnd
    19.2. Then pretend further duplicate ACKs are the expected ACKs

19.3. Set ssthresh → cwnd = cwnd/2

20. TCP Reno: combines slow-start, fast retransmit and fast recovery (MD is ½)
    20.1. TCP Reno can repair one loss per RTT → multiple losses cause a timeout
    20.2. TCP New Reno further refines ACK heuristics → reoairs multiple losses without timeout
    20.3. SACK → receiver sends ACK ranges so sender can retransmit without guesswork



21. Explicit congestion notification (ECN):
    21.1. Routers detect congestion via its queue
    21.2. When congested, it marks affected packets (IP header)
    21.3. Marked packets arrive at receiver and treated as a loss
    21.4. TCP receiver reliably informs TCP sender of the congestion

→ Advantages:

- Routers deliver clear signal to hosts
- Congestion is detected early
- No extra packets need to be sent

→ Disadvantages:

- Routers and hosts must be upgraded

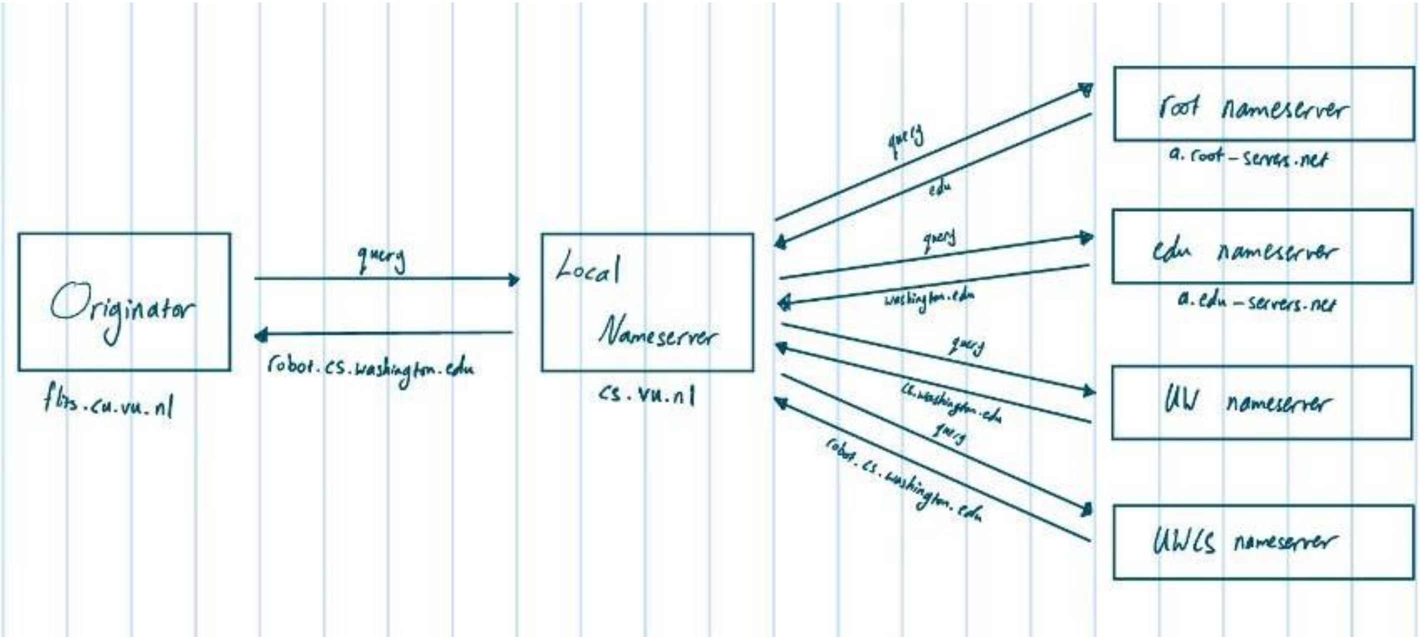
Week 9

---

1. OSI session concept: a series of related network interactions in support of an application task
2. OSI presentation concept: apps need to identify the type of content and encode it for transfer
3. Names: higher level identifiers for resources
4. Addresses: lower level locators for addresses
5. Resolution (lookup): mapping a name to an address
6. Before DNS:
   6.1. A file "hosts.txt" was regularly retrieved for all hosts from a central machine at the "network information center"
   6.2. Names were initially flat, they became hierarchical
7. DNS: a naming service to map between host names and their IP addresses
   7.1. Goals:
      7.1.1. Easy to manage
      7.1.2. Efficient
   7.2. Approach:
      7.2.1. Distributed directory based on a hierarchical namespace
      7.2.2. Automated protocol to tie pieces together
8. TLD's: top level domains that are run by the ICANN
9. DNS zone: a contiguous portion of the namespace
   9.1. They're the basis for distribution (EDU registrar administers .edu)
   9.2. Each zone has a nameserver to contact for information about it (zone must include contacts for delegations)
   9.3. A zone is comprised of DNS resource records that give information for its domain names

Type	**Meaning**
SOA	start of authority, has key zone parameters
A	IPv4 address of a host
AAAA	IPv6 address of a host
CNAME	canonical name for an alias
MX	mail exchanger for the domain
NS	nameserver of domain or delegated subdomain

10. DNS Resolution:



11. Request query:
    11.1. Nameserver completes resolution and returns the final answer
    11.2. Example → flits → local nameserver
    11.3. Lets server offload client burden (simple resolver) for manageability
    11.4. Lets server cache over a pool of clients for better performance
12. Iterative query
    12.1. Nameserver returns the answer or who to contact next for the answer
    12.2. Example → local nameserver → all others
    12.3. Lets server "file and forget"
    12.4. Easy to build high load servers
13. Caching: cache query/responses to answer future queries immediately
    13.1. Including partial (iterative) answers
    13.2. Responses carry a TTL for caching
14. Local nameserver:
    14.1. Typically run by IT
    14.2. May be your local host of AP
    14.3. May be alternatives like Google public DNS
    14.4. Typically configured by DHCP
15. Root nameservers:
    15.1. Root (dot) is served by 13 server names ([a,m].root-servers.net)
    15.2. All nameservers need root IP addresses
    15.3. Handled by a config file
    15.4. More than 250 distributed server instances
    15.5. Most servers are reached by IP anycast
    15.6. Servers are IPv4 and IPv6 reachable
16. DNS protocol:
    16.1. Mechanism
        16.1.1. Built on UDP messages, port 53
        16.1.2. ARQ for reliability; server is stateless
        16.1.3. Messages are linked by 16-bit ID field
    16.2. Reliability
        16.2.1. Multiple nameservers for domain

16.2.2. Return the list; clients use one answer
16.2.3. Helps distribute loads
16.3. Security
16.3.1. Compromises may redirect to wrong website
16.3.2. DNSSEC (DNS security extensions) is partially deployed

17. HTTP (hyper-text transfer protocol): a request / response protocol for fetching web resources
17.1. Runs on TCP, typically port 80
17.2. Part of browser / server app

➔ Steps:

- Resolve the server to its IP address (DNS)
- Set up a TCP connection to the server
- Send HTTP request for the page
- Execute / fetch embedded resources / render
- Clean up any idle TCP connection

➔ Commands used in the request:

Method	**Description**
GET	read a webpage
HEAD	read a webpage's header
POST	append to a webpage
PUT	store a webpage
DELETE	remove the webpage
TRACE	echo the incoming request
CONNECT	connect through a proxy
OPTIONS	query options for a page

➔ Codes returned with the response:

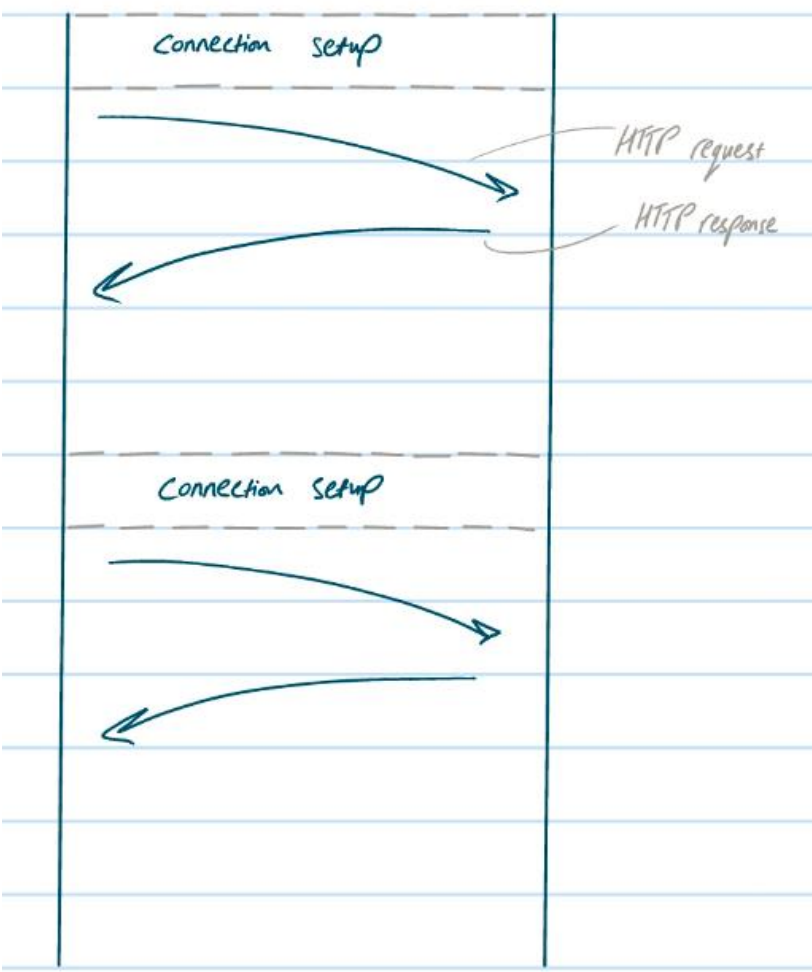
Code	**Meaning**
1xx	info
2xx	success
3xx	redirection
4xx	client error
5xx	server error

➔ Many header fields specify capabilities and content:

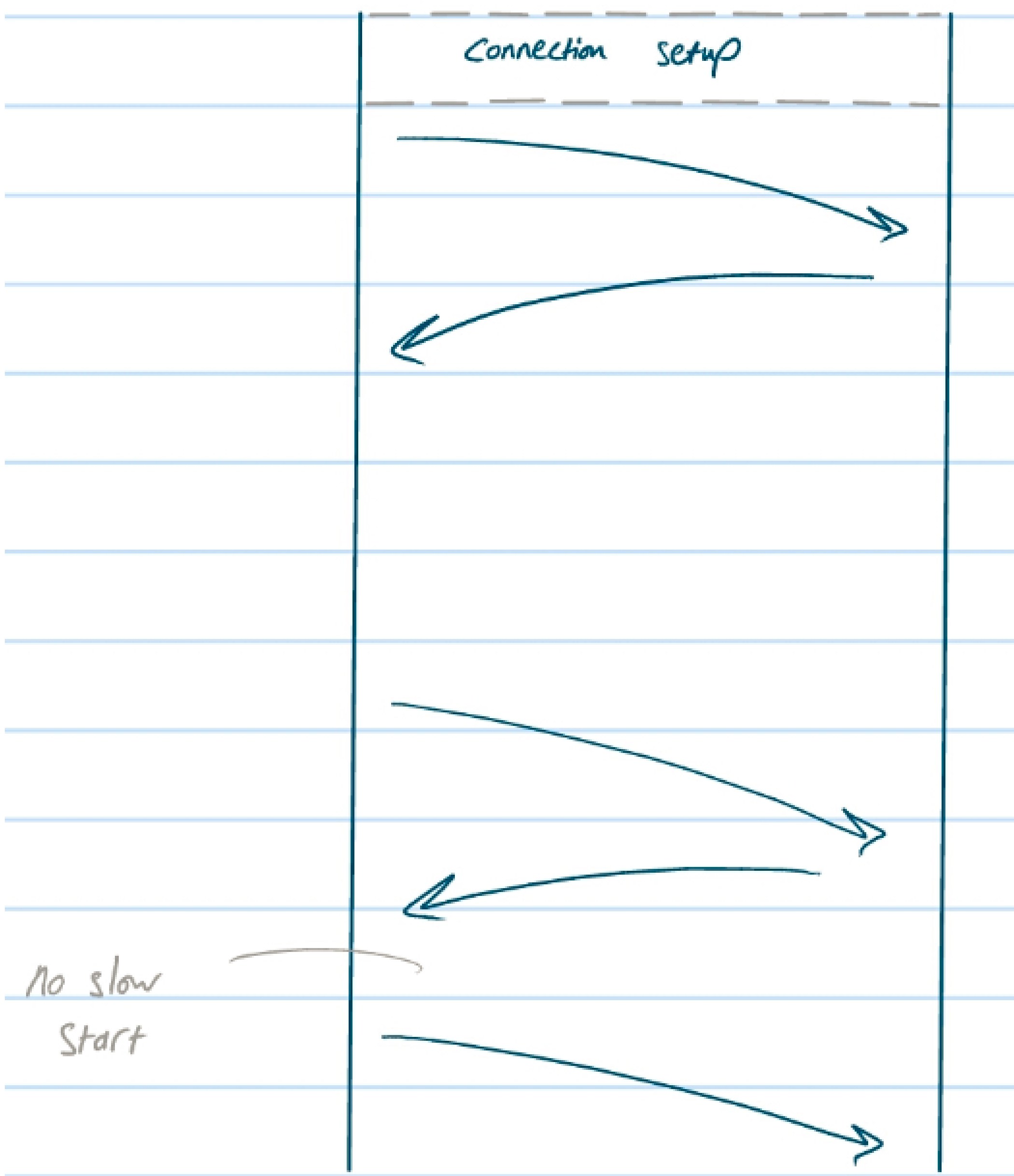
Function	**Example Header**
browser capabilities (client --> server)	user-agent, accept, accept-charset, accept-encoding, accept-language
caching related (mixed directions)	if-modified-since, if-none-match, date, lost-modified, expires, cache-control, Etag
browser context (client --> server)	cookie, referrer, authorization, host
content delivery (server --> client)	content-encoding, content-length, content-type, content-language, set-cookie

18. PLT (page load time):
18.1. The key measure of web performance
18.2. Depends on
18.2.1. Structure of page / content

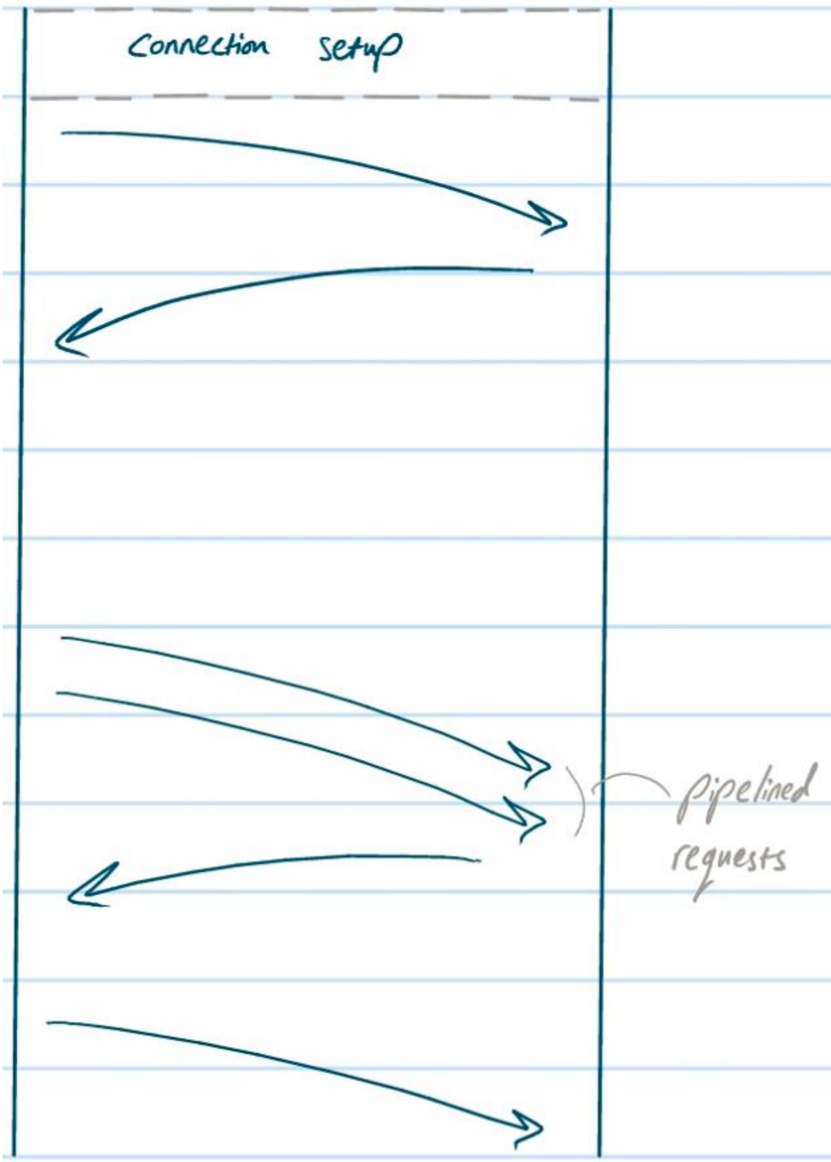
18.2.2. HTTP (and TCP) protocol
18.2.3. Network RTT and bandwidth
19. HTTP 1.0 uses one TCP connection to fetch one web resource
20. Ways to decrease PLT:
    20.1. Reduce content size for transfer
    20.2. Change HTTP to avoid repeated transfers of the same content
    20.3. Move content closer to clients
21. HTTP 1.1:
    21.1. Support optional pipelining
    21.2. PLT benefits depends on page structure, but easy on network
22. Type of connections:
    22.1. One request per connection



22.2. Sequential requests per connection



22.3. Pipelined requests per connection



23. Using local copies of web caching
    23.1. Locally
        23.1.1. Based on expiry info (“expires header from server”)
        23.1.2. Use heuristics (“cacheable”, “freshly valid”, “not modified recently”)
        23.1.3. Content available right away
    23.2. Remotely
        23.2.1. Based on timestamp of copy (“last modified header” from server)
        23.2.2. Based on content of copy (“ETag” header from server)
        23.2.3. Content available after one RTT
24. Web proxies: place an intermediary between a pool of clients and external web servers
    24.1. Clients benefit from larger shared cache
    24.2. Enforce organizational access policies
25. Content distribution network (CDN)
    25.1. Problem:
        25.1.1. Concentrated load on popular servers
        25.1.2. Congested network

25.1.3. Poor PLT (and user experience)
25.2. Solution:
25.2.1. Place popular content near clients
25.2.2. DNS resolution of site gives different answers to clients
25.3. Business model:
25.3.1. Placing site replica at an ISP is a win-win
25.3.2. Improves site experience and reduces bandwidth usage of ISP

26. Waterfall diagrams shows progression of page loads

➔ Waterfall and PLT depends on many factors

- Very different for different browsers
- Very different for repeat page views
- Depends on local computation as well as network

27. Recent work to reduce PLT:
27.1. Better use of network (HTTP/2)
27.2. Better content structure (mod-pagespeed)

28. SPDY: a set of HTTP improvements
28.1. Multiplexed (parallel) HTTP requests on one TCP connection
28.2. Client priorities for parallel requests
28.3. Compressed HTTP headers
28.4. Server push of resources

29. Mod-pagespeed: an open source Apache module
29.1. Observation
29.1.1. The way pages are written affects how quickly they load
29.1.2. Many books on best practices for page authors and developers
29.2. Key idea
29.2.1. Have the server re-write (compile) pages to help them load quickly
29.2.2. Rewrite pages “on the fly” with rules based on best practices

30. CDN Advantages:
30.1. Efficient, scales up for popular content
30.2. Reliable, managed for good service

31. CDN disadvantages:
31.1. Need for dedicated infrastructure
31.2. Centralized control / oversight

32. P2P challenges:
32.1. Limited capabilities
32.2. Participation incentives
32.3. Decentralization

➔ peer can send content to all other peers using a distribution tree

- Done with replicas over time
- Has a self-scaling capacity

➔ peer plays two rules (download / upload)

33. Distributed hash tables: fully decentralized, efficient algorithms for a distributed index
33.1. Index is spread across all peers
33.2. Index lists peers to contact for content
33.3. Any peer can lookup the index

34. Bit Torrent:

34.1. Delivers data using torrents
34.2. Transfers files in pieces for parallelism
34.3. Notable for its treatment of incentives
34.4. Uses a tracker or a decentralized index (DHT)

35. Bit Torrent protocol:
    35.1. Start with torrent description
    35.2. Contact tracker to join and get list of peers (with at least one seed peer)
    35.3. Or, use DHT index for peers
    35.4. Trade pieces with different peers
    35.5. Favor peers that upload to you rapidly; “choke” peers that don’t by slowing your upload to them

Week 10

---

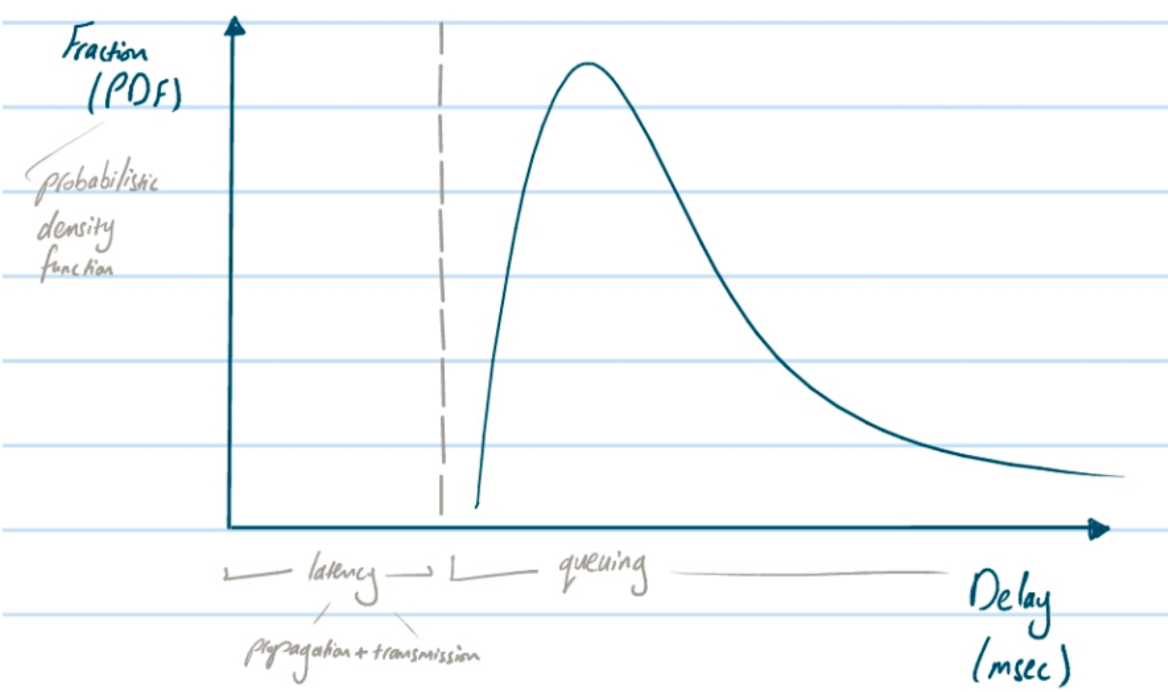
1. "Best Effort" service is what we have on the Internet today
2. Quality Of Service (QOS): allocate bandwidth in a way that improves app / user performance

   ➔ Application requirements:

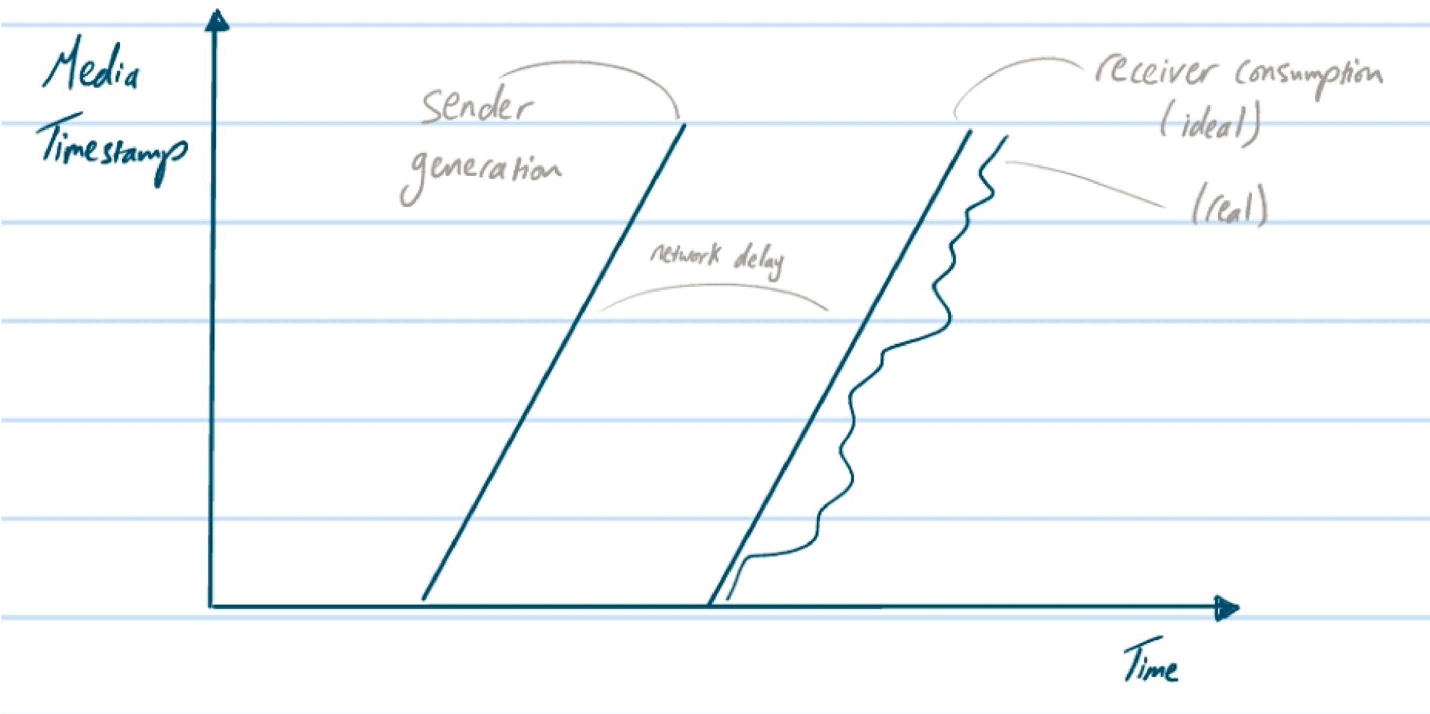
Application	**Bandwidth**	**Delay**	**Jitter**	**Loss**
Email	low	low	low	medium
File Sharing	high	low	low	medium
Web Access	medium	medium	low	medium
Remote Login	low	medium	medium	medium
Audio on Demand	low	low	high	high
Video on Demand	high	low	high	high
Telephony	low	high	high	high
Video Conferencing	high	high	high	high

   ➔ QOS matters only when there is a network bottleneck

3. Real Time Transport: sending interactive real time media over the Internet (e.g., VOIP)
   3.1. Using best effort Internet
   3.2. Using playout buffer technique
   3.3. Constant rate of media is generated at source, later consumed at receiver
4. Network delay



5. Playout

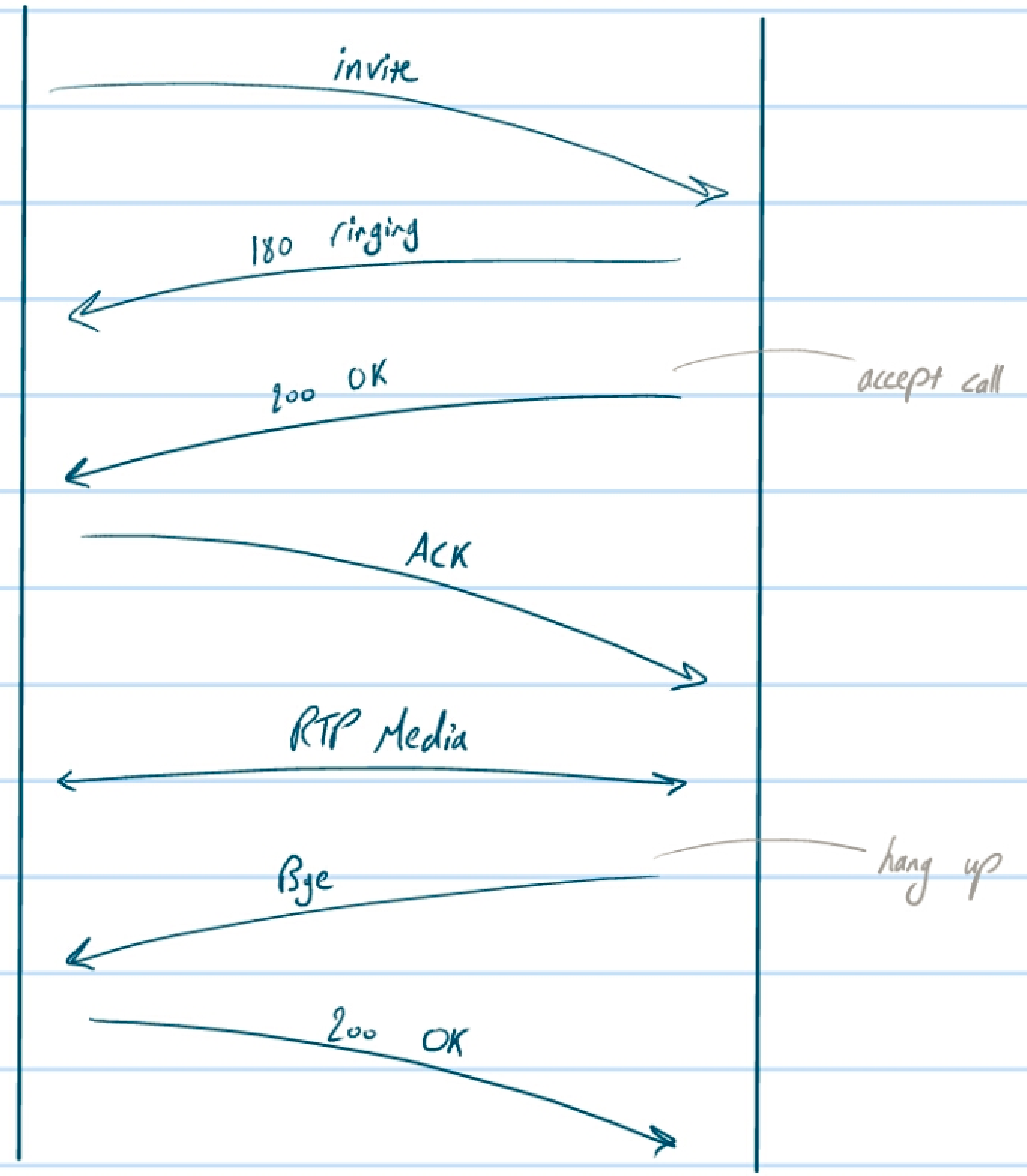


5.1. Pick longest acceptable network delay to set the playout point (packets that arrive after the playout point are ignored)
5.2. Tradeoff:
5.2.1. Larger acceptable network delay leads to larger buffer / delay, less loss
5.2.2. Smaller acceptable network delay leads to smaller buffer / delay, more loss
6. Real-time session components:
6.1. Call setup with SIP
6.2. Session description, with SDP
6.3. Media transport, with RTP
6.4. Media playout, with buffer
7. Real Time Transport Protocol (RTP): used to carry media on top of best effort UDP

RTP packet header

Bit offset[a]	0–1	2	3	4–7	8	9–15	16–31
0	Version	P	X	CC	M	PT	Sequence number
32	Timestamp						
64	SSRC identifier						
96	CSRC identifiers ...						
96+32×CC	Profile-specific extension header ID						Extension header length
128+32×CC	Extension header ...						

8. Session Initiation Protocol (SIP): open protocol for establishing voice and video calls over IP
    8.1. Provides the signaling; media is carried directly with RTP (or other)
    8.2. Like HTTP, uses simple method / response codes
    8.3. Runs on UDP or TCP
    8.4. SIP proxy servers and registrars provide mobility



9. Streaming media:
   9.1. Use high / low watermarks to control source over / underfill
      9.1.1. Start pulling media at low level
      9.1.2. Stop pulling media at high level
   9.2. Send file in one of multiple encodings depending on available bandwidth
   9.3. TCP is typically used
      9.3.1. Low delay is not essential
      9.3.2. Loss recovery simplifies presentation
      9.3.3. HTTP/TCP passes through firewalls
   9.4. Session consists of several parts:
      9.4.1. Signaling (RTSP)
      9.4.2. Media transport (HTTP)
      9.4.3. Media playout (with buffer)
10. Real Time Streaming Protocol (RTSP): streaming with RTSP steps:
   10.1. Video started using HTTP to get metafile
   10.2. Invoke media player → talks RTSP to media server
   10.3. Media sent with, e.g., RTP over TCP/UDP
11. Streaming with HTTP
   11.1. Fetch media description data → gives index clips, rates
   11.2. Fetch small segments → put in playout buffer
   11.3. Adapt selection of encoding → based on buffer occupancy
12. DASH: dynamic adaptive streaming over HTTP
13. Fair queuing:
   13.1. FIFO queue → aggressive user / flow can crowd out others
   13.2. Round-Robin queuing → different packet sizes lead to bandwidth imbalance
   13.3. Fair Queuing
      13.3.1. Approximate by computing virtual finish time
      13.3.2. Send packets in order of their virtual finish times
      13.3.3. Don't preempt packets being transmitted

$$Arrive(j)_F = arrival\ time\ of\ j^{th} packet\ of\ flow\ F$$

$$Length(j)_F = length\ of\ j^{th} packet\ from\ flow\ F$$

$$Finish(j)_F = \max(Arrive(j)_F, Finish(j-1)_F) + Length(j)_F$$

   13.4. Weighted Fair Queuing (WFQ)
      13.4.1. Generalization of fair queuing
      13.4.2. Assign a weight to each flow (higher weight gives more bandwidth)
      13.4.3. Change the computation of the virtual finish time to factor in the weight

$$Arrive(j)_F = arrival\ time\ of\ j^{th} packet\ of\ flow\ F$$

$$Length(j)_F = length\ of\ j^{th} packet\ from\ flow\ F$$

$$Finish(j)_F = \max(Arrive(j)_F, Finish(j-1)_F) + \frac{Length(j)_F}{Weight_F}$$

      13.4.4. Need to determine flows (user? App? TCP connection?)
      13.4.5. Difficult to implement at high speed for many concurrent flows
      13.4.6. Need to assign weights to flows

14. Shaping traffic:
    14.1. Motivation
        14.1.1. Limiting the total traffic enables bandwidth guarantees
        14.1.2. Limiting bursts avoids unnecessary delay and loss
    14.2. Token bucket (R, B)
        14.2.1. Average rate of R bits/sec
        14.2.2. Bursts (over R) of B bits
        14.2.3. Sending removes tokens (or credit) from the bucket; no credit, no send
        14.2.4. Fill rate of R bits/sec
        14.2.5. Bucket capacity of B bits
15. Token bucket implementation methods
    15.1. Shaping
        15.1.1. Run (R,B) token bucket at the source
        15.1.2. Pass sent packets when there are tokens
        15.1.3. Queue packets while more tokens arrive
        15.1.4. Lets user condition their traffic to meet the network contract
    15.2. Policing
        15.2.1. Run (R, B) token bucket at network edge
        15.2.2. Let packets into the network when there are tokens
        15.2.3. Demote or discard packets when there are insufficient tokens
        15.2.4. Lets network check traffic to verify it meets the user contract
16. Differentiated services architecture:
    16.1. User makes packets with desires service
    16.2. Network polices traffic levels at boundary (token bucket)
    16.3. Network provides different forwarding (WFQ at routers)
    16.4. Marking packets → use bits in IPv4 / IPv6 header to mark the kind of service (through the 6-bit DSCP field, by user/OS/Network)

Service Name / Meaning	**DSCP Value**	**Traffic Need (app example)**
default forwarding / best effort	0	elastic (Bit Torrent)
assured forwarding / enhanced effort	Oct-38	average rate (streaming video)
expedited forwarding / real-time	46	low loss / delay (VoIP / gaming)
precedence / network control	48	high priority (routing protocol)

    16.5. Policing policies
        16.5.1. Network (ISP) checks incoming traffic meets service contract
        16.5.2. Not more expedited traffic than agreed
        16.5.3. Only allowed markings (no network control for user)
        16.5.4. Done with token buckets
        16.5.5. Can demote traffic by re-marking or prioritizing for loss
    16.6. Forwarding packets
        16.6.1. Network (ISP) routers use WFQ
        16.6.2. The different kinds of services are the different flows / queues
        16.6.3. DSCP values are used to map packets to the right flow / queue
17. Note: QOS provides value when it is deployed across the network
18. Hard QOS: providing a guaranteed service with minimum rate and maximum delay regardless of how other flows behave
    18.1. Admission control
        18.1.1. Decide whether to admit or reject a flow F that needs rate >= R and delay <= D

18.1.2. For all router (i)

$$\frac{W_F(i)}{W(i)} * L(i) \geq R$$

- $W_F(i)$ → weight of flow at router
- $W(i)$ → weight for all other flows at router
- $L(i)$ → rate of output link
- R → rate required (mbps)

$$Delay \leq Latency + \frac{B}{R}$$

- $Latency$ → transmission + propagation
- B → burst
- R → rate