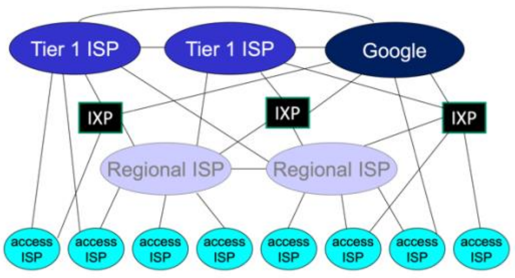
Network Edge (Access Network)
- Hosts access the Internet through access network (local networks)
  - Range from home network to ethernet
- Internet is a network of networks

Wireless Access Network
- Connect hosts to router via access point
- Alternative ways to connect to router is with ethernet cable
  - Device <> [Wireless Access Point] <> Router <> [Modem] <> ISP <> IXP <> ISP <> Server



Circuit Switching
- End-end resources allocated to and reserved for "call" between source and destination
  - Requires call setup
  - Circuit-like performance guaranteed

Packet Switching
- Host breaks application message into smaller chunks (packets) of length L bits, transmitting packets onto the link at transmission rate R bits/sec (link capacity/bandwidth)

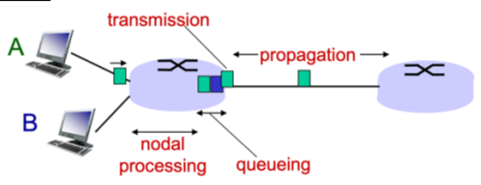
$$transmission\ delay = \frac{L}{R} s$$

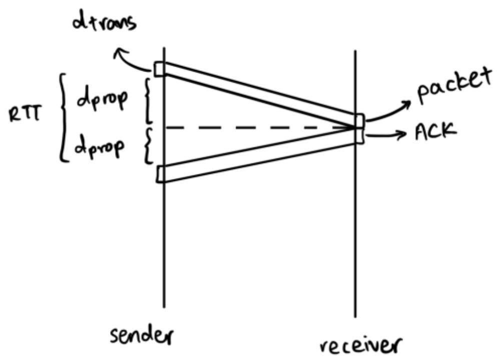
Store and forward: entire packet must arrive at a router before it can be transmitted on the next link

$$delay = n * \frac{L}{R}$$

where n is the number of links between source and destination

Delay and Loss





Packet loss: packet drops when arriving at full queue (buffer)

Types of delay:
1. Nodal processing: time taken to check bit errors and determine output link, time waiting before getting into queue
2. Queuing delay: time waiting in the queue for transmission, depending on the congestion of the router
   - Total time taken to transmit all other packets queued (n) and in-transmission (x)

$$\frac{nL + (L - x)}{R}$$

3. Transmission delay: time taken to push the packet bits onto the physical layer (i.e. go from code to hardware, also thought of as the time difference between the first bit leaving to the last bit leaving)
   - Once per object/packet

$$\frac{packet\ length\ (L)}{link\ bandwidth\ (R)}$$

4. Propagation delay: time taken for the physical bits to be sent to the target; not dependent on size of packet

$$\frac{length\ of\ physical\ link\ (d)}{propagation\ speed\ in\ medium\ (s)}$$

   - Incurred once for each packet sent (if not sent continuously)
   - Response packets will incur same delay so propagation delay * 2 is the RTT
- Think of propagation as 4 step process:
  - Receive packet: perform checks, incur nodal processing
  - Queue packet: queuing delay
  - Move packet from network to link layer: transmission delay
  - Move across physical link: propagation delay

Message segmentation
- Calculated by accounting for time taken for each packet to be sent through the server

$$(n + m - 1) \times \frac{L}{R}$$

where n is the number of links, m is the number of packets

	Time				
Packet	1	2	3	4	4
1	🟨	🟨	🟨		
2		🟨	🟨	🟨	
3			🟨	🟨	🟨

Reasons to use:
1. Reduce delay of sending large files
2. Bit errors that trigger a retransmission only need to retransmit a smaller packet than the entire file
3. Large files can hog the queue and prevent smaller packets from being sent

Drawbacks:
1. Packets must be put in sequence at the destination
2. Smaller packets each have a packet header, introducing header overhead

Topology
- Given N devices with at most 1 link between 2 devices
- Minimum number of links: tree (N - 1) (tree/chain/star)
- Maximum number of links: fully connected graph (N*(N-1)/2) (mesh)

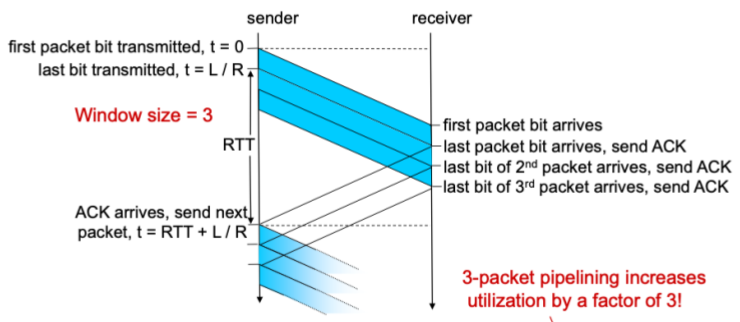
	Pro	Con
Sparse	Simple topology	• Failure point of failure • Longer path between 2 nodes
Dense	More robust with 1 hop between all nodes	More expensive

Throughput
- Number of bits that can be transmitted per unit time
- Measured for end-to-end communication
- Core idea is to take the total amount of bits transmitted over the total time spent transmitting **
- N is the number of packets sent (formulating the total size + transmission delay)
- K is the number of propagation delays incurred (link throughput has K = 1, connection throughput has K = 2)

$$\frac{n \times L}{k \times d_{prop} + n \times d_{trans}}$$

Utilization
- Fraction of time sender is busy sending/not idling
- If sending using sliding window, must include c where c is the total number of packets per window, otherwise c = 1 (same for pipelining)

$$\frac{d_{trans} \times c}{RTT + d_{trans}}$$


sender
receiver
first packet bit transmitted, t = 0
last bit transmitted, t = L / R
Window size = 3
RTT
first packet bit arrives
last packet bit arrives, send ACK
last bit of 2nd packet arrives, send ACK
last bit of 3rd packet arrives, send ACK
ACK arrives, send next packet, t = RTT + L / R
3-packet pipelining increases utilization by a factor of 3!

$$U_{sender} = \frac{3 * L / R}{RTT + L/R} = \frac{0.024}{30 + 0.008} = 0.00081$$

Metric Units

- 1 byte = 8 bits

Prefix	Exp.	Prefix	Exp.
mili	-3	Kilo	3
micro	-6	Mega	6
nano	-9	Giga	9
pico	-12	Tera	12

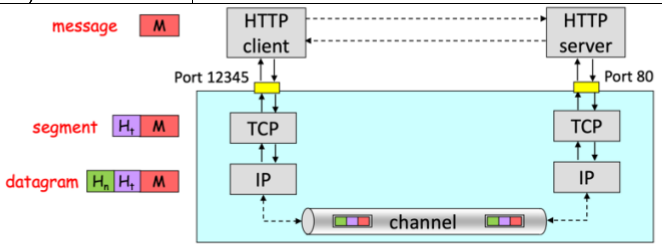
Network Protocols

- Defines format and order of messages exchanged and actions taken per message

Network Layers

- More examples found in chapter 2/3 slides 13/16

[HOST] **Application** (HTTP/DNS/DHCP /SMTP/FTP)	Supporting network applications
[HOST] **Transport** (TCP/UDP)	Process to process data transfer***
[HOST] **Network** (IP/routing protocols/ICMP)	Routing of datagrams from source to destination; host to host
[HOST/PHYSICAL] **Link** (Ethernet/Wi-Fi)	Data transfer between neighboring network elements
[PHYSICAL] **Physical** (Cables/Optical Fibre)	Bits "on the wire"



Client-Server Architecture

Client	Server
Initiates contact with server and requests service from server	Waits for incoming requests and provides requested service to client

RTT: time for a packet to travel from client to server and go back (round trip)

Peer-to-Peer Architecture

- Peers request service from other peers and provide services in return
- Peers intermittently connected and change IP address

Self-scalable: more peers mean more service capacity

Transport Service Considerations

1. Data integrity (acceptable loss rate)
2. Throughput (minimum bandwidth)
3. Timing (delay)
4. Security (encryption, data integrity)

Application Layer Protocol

1. Types of messages exchanged: request/response
2. Message syntax: fields/delineation
3. Message semantics: meaning of information
4. Rules: when/how applications send/respond to messages

Open protocol: defined in RFC and allows for interoperability
Proprietary protocol: Skype

Transport Layer Protocol

1. TCP: reliable data transfer, flow control, congestion control; lacking timing, minimum throughput guarantee, security
2. UDP: unreliable data transfer, no flow control, no congestion control; lacking same as TCP

Sender: breaks application message into segments and passes to network layer
Receiver: reassembles segments into message and passes to application layer
Routers: check destination IP address
Segment: contains source and destination port number

Network Layer

- IP datagram contains source and destination IP addresses
- Receiver identified by destination IP address
- Data added onto transport layer segment

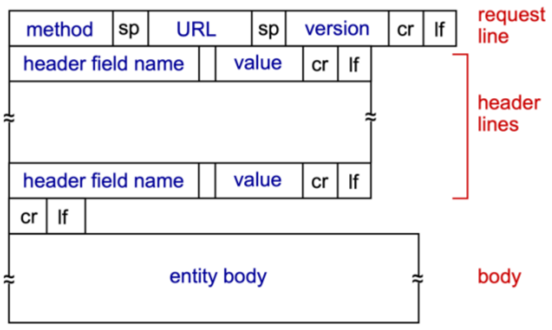
HyperText Transfer Protocol (HTTP)

- Application layer protocol for the web
- Rely on client-server model
- Using TCP for transport layer protocol

Persistence: non-persistent HTTP closes connection after sending one object; persistent HTTP requires just one connection to send all objects

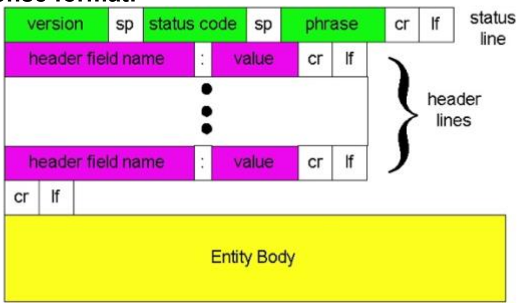
- Persistent HTTP enabled in HTTP 1.1
- Client can reuse the connection to retrieve all referenced objects and only require 1 RTT per web page
- Persistence through the same server (not resource)
- Indicated by header field: **Connection: keep-alive**
  - Response will acknowledge with same if persistent

Response time: 1 RTT to establish TCP connection + 1 RTT for HTTP request and HTTP response + file transmission time
Problems with non-persistent: requires 2 RTT per object, each TCP connection contains overhead, and browser must open parallel connections to retrieve all objects
Request format:



- cr and lf is equivalent to \r\n
- Methods: HTTP/1.0 includes GET, POST, HEAD, HTTP/1.1 added HEAD, PUT, and DELETE

Response format:



- Common status codes:
  - 200 OK
  - 301 Moved Permanently
  - 403 Forbidden
  - 404 Not Found
  - 400 Bad Request
- Content-Length only includes length of body, not headers

Cookies

- Carry state in HTTP messages

- Cookie file stored on user's host and managed by user's browsers
- Sent to/from user/server

<u>Conditional GET</u>

- Don't send object if client cache has up-to-date cached version
- Determined by If-modified-since header in HTTP request containing the date of cached copy in the request
- Server checks if object was modified since time indicated and if not updated, responds with 304 Not Modified

<u>Domain Name System (DNS)</u>

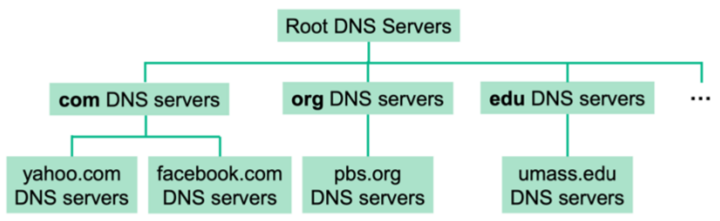
- Translates between hostname and IP address of host
- Client must perform DNS query to determine the IP address corresponding to the server name prior to the connection
- Runs on UDP
- Hostname can map to multiple IP addresses (see telnet)
- Defaults to port 53

Resource Records (RR): mapping between host names and IP addresses

(name, value, type, ttl)

- Types of record:
  - A: default address; name is hostname, value is IP address
  - CNAME: name is alias for "canonical" name, value is canonical name
  - NS: name server; name is domain, value is hostname of authoritative name server for domain
  - MX: mail server; value is name of mail server associated with name

DNS hierarchy:



- Go from root server down to individual servers

Root server: answers requests for records in the root zone by returning list of authoritative name servers for top-level domain (TLD); <u>not always needed if local DNS can already find the necessary resource</u>

TLD server: responsible for .com, .org, .net, .edu, …

Authoritative server: organization specific DNS server providing authoritative hostname to IP mapping for organization's named hosts, maintained by organization or service provider

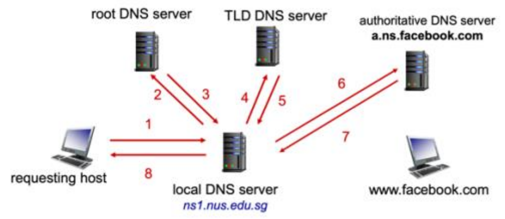
Local DNS server: does not belong strictly in hierarchy

- Every ISP has one local DNS server, i.e. default name server
- Client <> Local DNS server <> Hierarchy
- Only forwards if not found in local DNS server

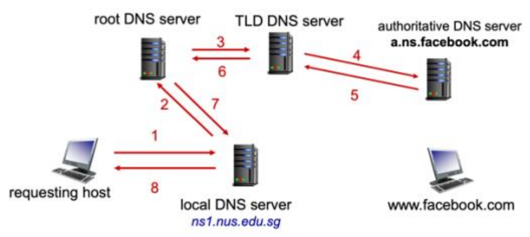
DNS caching: name server caches mapping; cached entries expire after a while (TTL) and may be out of date

- Update/notify mechanisms to update when name host changes IP address
- Can be used to determine if an external website was previously accessed; response will be faster
  - Not a guaranteed way to check

Iterative query: local DNS server performs all requests to and from other DNS servers



Recursive query: every DNS server is responsible for querying the next DNS server (supposedly used more for local DNS)



<u>DNS cache poisoning</u>

- Computer hacking attack where rogue DNS records are introduced into a DNS resolver's cache
- Causes the name server to return an incorrect IP address, diverting traffic to the attacker's computer

<u>Processes</u>

- Program running within a host
- Use Inter-Process Communication (IPC) to communicate between same host processes
- Use messages to communicate between different host processes (according to protocols)

Process address: identifier for host and what process

(IP address, port number)

- Port number: 16 bit integer with 1-1023 reserved for standard use

<u>Sockets</u>

- Software interface between app processes and transport layer protocols
- Process sends/receives messages to/from its socket via set of APIs
- Don't need to flush socket after every send, only flush before closing the socket

<u>UDP</u>

- Unreliable datagram socket
  - Can implement reliability checking on application layer
- Every request includes the server IP and port number of the process
- Client must form datagram explicitly with destination IP and port for every packet
- Used by streaming multimedia applications that are loss-tolerant and rate sensitive

Pros:

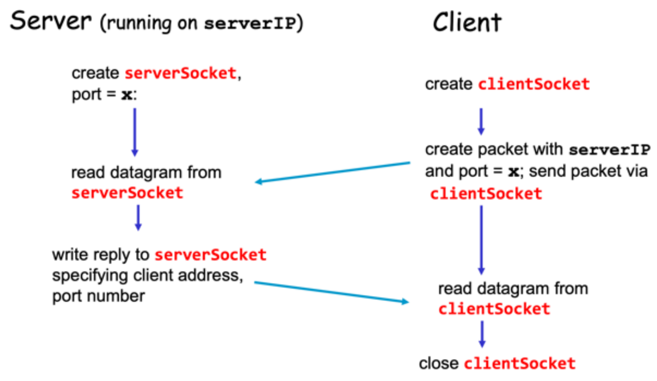
- Connectionless (can add delay)
- Simple
- Small header size
- Client does not need the server to run
- No congestion control, can go as fast as it wants

Programming:

- Uses SOCK_DGRAM

Server: application program <u>explicitly attaches destination IP address and port number to each packet</u>

- Contains exactly 1 process running the UDP program with 1 port

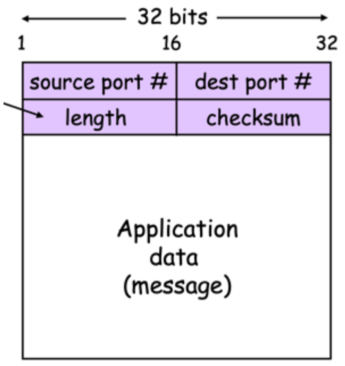

Server (running on serverIP)
Client
create serverSocket, port = x:
create clientSocket
create packet with serverIP and port = x; send packet via clientSocket
read datagram from serverSocket
write reply to serverSocket specifying client address, port number
read datagram from clientSocket
close clientSocket

Added features (on top of IP):

1. Multiplexing at sender: gather data from process, forms packets, and passes to IP
2. De-multiplexing at receiver: receives packets from lower layer and dispatches them to the right processes
   - Checks destination port number in segment
   - Directs UDP segment to the socket with that port number
   - IP datagrams with the same destination port number is directed to the same UDP socket

- Server sends response to the source IP in the header to differentiate

3. Checksum



- Header length is length of header (64 bits) + data

TCP

- Reliable, byte stream-oriented socket
- Point-to-point, full duplex (bi-directional)
- Establish TCP connection with server first (through welcome/listening socket)
  - Once established, new socket is created for server process to communicate with client (exclusive socket)
  - Identified with (source IP address, source port, destination IP address, destination port)
  - Always +1 to the number of sockets for a TCP server
- All future requests do not need the port number
- Client requires server to run; asymmetric
- Exclusive socket is a persistent pipe between processes until one of the processes close it
- Uses SOCK_STREAM
- Server must use bind() to set port, client does not and will rely on OS to assign if not provided

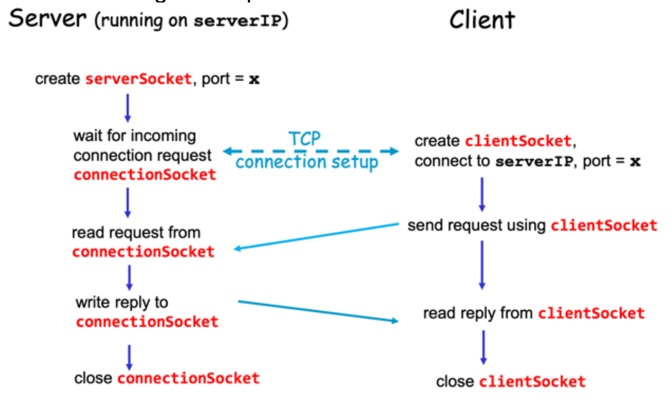

Server (running on serverIP)
Client
create serverSocket, port = x
wait for incoming connection request connectionSocket
TCP connection setup
create clientSocket, connect to serverIP, port = x
read request from connectionSocket
send request using clientSocket
write reply to connectionSocket
read reply from clientSocket
close connectionSocket
close clientSocket

Figure 1: TCP Interaction

Total time to make query:

$$n \times D_{DNS} + (m+1) \times 2 \times D_{Web}$$

Segment: maximum segment size (MSS) is how much the segment can carry (excluding the header)

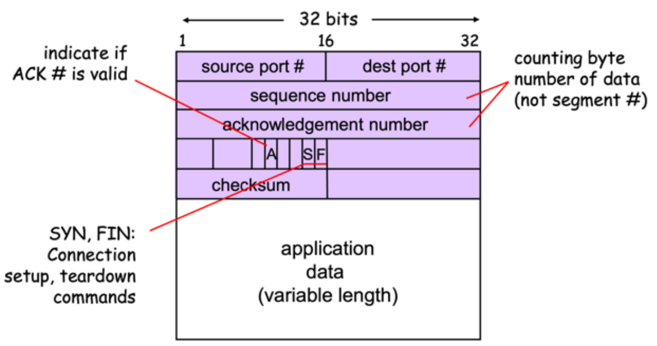
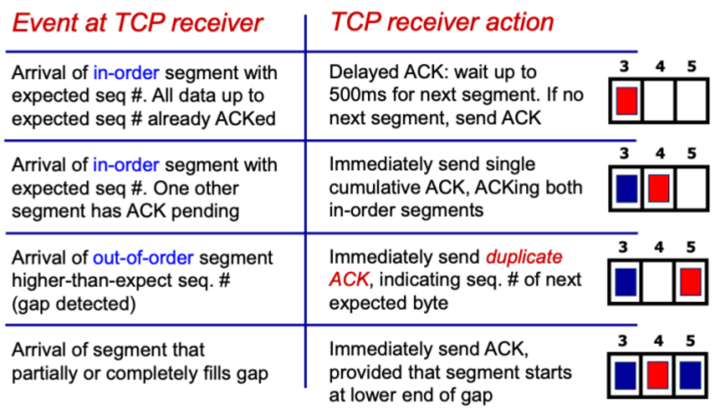

32 bits
1
16
32
source port #
dest port #
sequence number
acknowledgement number
A
S
F
checksum
application data (variable length)
indicate if ACK # is valid
counting byte number of data (not segment #)
SYN, FIN: Connection setup, teardown commands

Figure 2: TCP Header

Sequence number: byte number of the first byte of data in a segment; usually random offset for initial value from [0, 2^{32}-1]

ACK number: sequence number of next byte of data expected by the receiver (sequence number + MSS)

- TCP ACKs up to first missing byte in the stream (repeats the missing byte until it is found)



Event at TCP receiver	TCP receiver action	3	4	5
Arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	Delayed ACK: wait up to 500ms for next segment. If no next segment, send ACK	🟥	⬜	⬜
Arrival of in-order segment with expected seq #. One other segment has ACK pending	Immediately send single cumulative ACK, ACKing both in-order segments	🟦	🟥	⬜
Arrival of out-of-order segment higher-than-expect seq. # (gap detected)	Immediately send *duplicate ACK*, indicating seq. # of next expected byte	🟦	⬜	🟥
Arrival of segment that partially or completely fills gap	Immediately send ACK, provided that segment starts at lower end of gap	🟦	🟥	🟦

Timeout: too short causes premature timeout and unnecessary retransmissions; too long causes slow reaction to segment loss; should be longer than RTT

- Only one timer and retransmits one packet at most

$$est\ RTT = (1-\alpha) \times est\ RTT + \alpha \times sample\ RTT$$
$$dev\ RTT = (1-\beta) \times dev\ RTT + \beta \times |sample\ RTT - est\ RTT|$$
$$timeout = est\ RTT + 4 \times dev\ RTT$$

Fast retransmission: if sender receives 4 ACKs for the same segment, it supposes that segment is lost so it resends the segment (even before the timer expires)

Handshake:

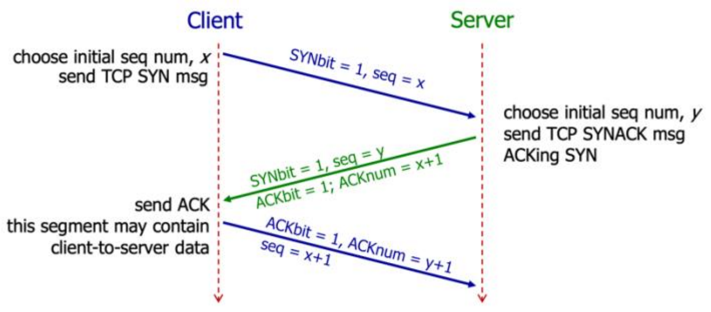

Client
Server
choose initial seq num, x send TCP SYN msg
SYNbit = 1, seq = x
choose initial seq num, y send TCP SYNACK msg ACKing SYN
SYNbit = 1, seq = y
ACKbit = 1; ACKnum = x+1
send ACK this segment may contain client-to-server data
ACKbit = 1, ACKnum = y+1
seq = x+1

Figure 3: TCP Handshake

- The third packet can contain 0 data as it's a regular ACK
- SYN must use at least 1 byte
- Adds 1 RTT to every request (handshake + request for object)

Closing connection:

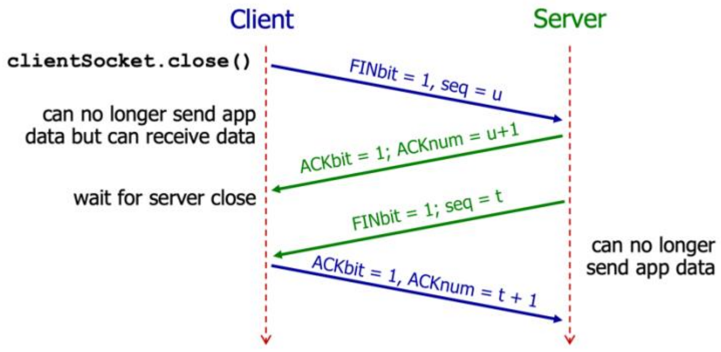
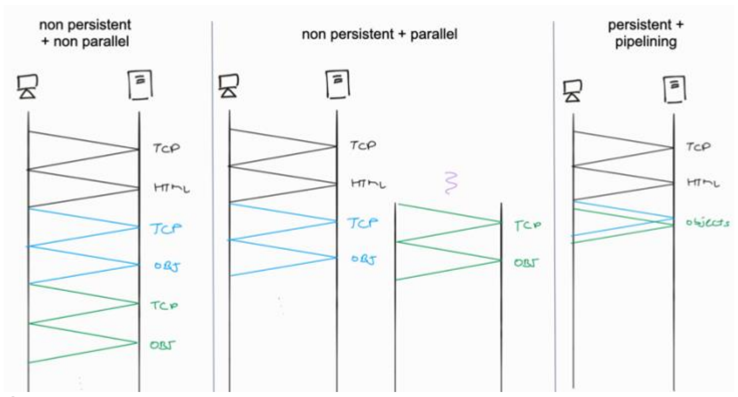


Figure 4: TCP Close Connection

- FIN uses at least 1 byte

Parallel TCP connections

- Parallel creates a new TCP connection per object at the same time
- Pipelining uses the same connection and sends multiple requests at once; different from parallel



Guards against duplicate packets

- Ensure that a sequence number is not reused until the sender is "sure" that the packets with the same sequence number is not in the network
- Large sequence number field (32-bit) lowers chance of reuse (random)
  - Maximum file size without using all sequence numbers is 2^{32} bytes
- TTL provided by IP protocol to avoid circulating in network infinitely (decreases for each hop)
  - Each packet's lifetime is about 3 minutes

Reliable Data Transfer (rdt)

- Sits between transport and network layer to ensure that unreliable data stream from network becomes reliable in the transport layer

Unreliability:

1. Corrupt packets
2. Dropped packets
3. Re-ordered packets
4. Delivered packets after long delay

rdt version	Scenario	Features used
1.0	No error	Nothing
2.0	Data bit error	Checksum, ACK/NAK
2.1	Data bit error + ACK/NAK bit error	Checksum, ACK/NAK, sequence number
2.2	Same as 2.1	NAK free
3.0	Data bit error + ACK bit error + packet loss	Checksum, ACK, sequence number, timeout/re-trasmission

rdt 1.0

- Assume underlying channel is perfectly reliable
- Sender sends data into underlying channel and receiver receives without any issues

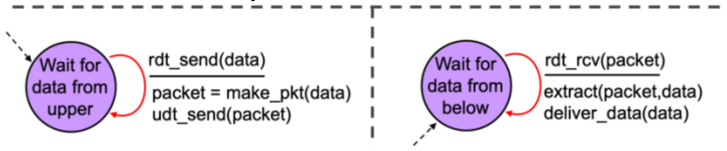


Figure 5: rdt 1.0

rdt 2.0

- Assume underlying channel may flip bits in packet
- Checksum used to detect bit errors and recover using acknowledgements from the receiver to the sender on success and negative acknowledgements on failure
  - NAKs cause sender to retransmit packet

Stop and wait protocol: sender sends one packet at a time, waiting for receiver response

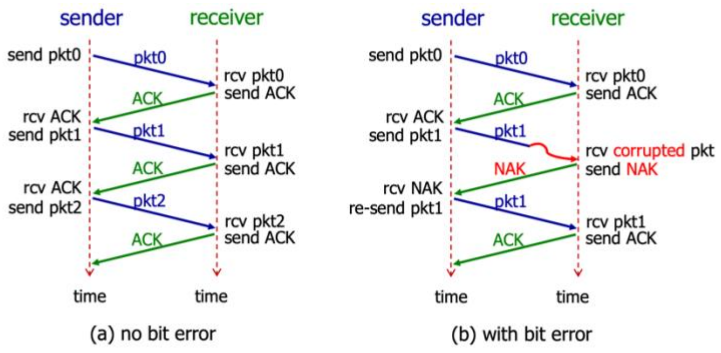


Figure 6: rdt 2.0 Simulation

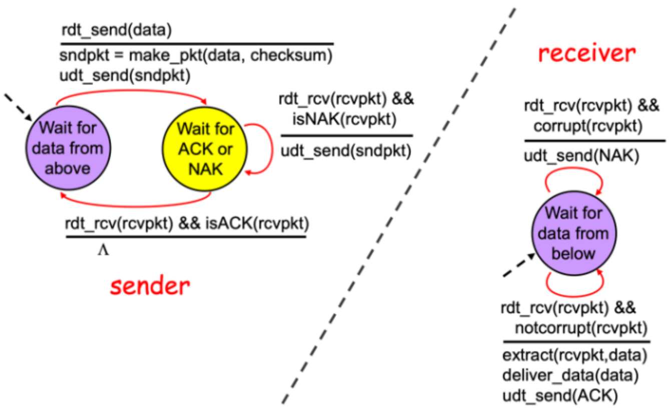


Figure 7: rdt 2.0 FSM

rdt 2.1

- Adding packet sequence numbers
  - Used for duplicate detection
- Once packet received, send acknowledgement for that packet number and tracks which packets have been received
- If receiver receives duplicate packet, discard it
- If sender receives any garbled ACK/NAK, resend packet

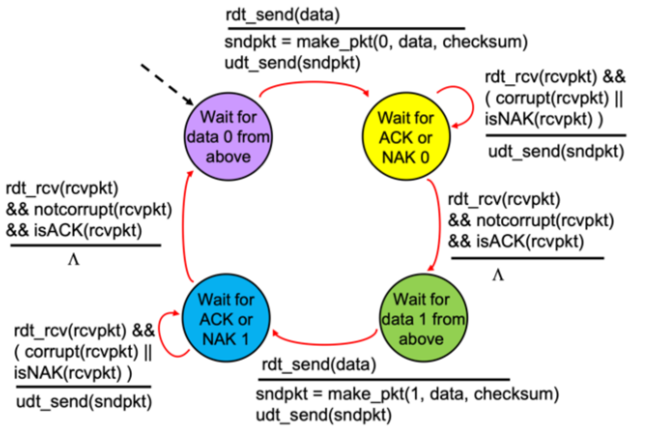

rdt_send(data)
sndpkt = make_pkt(0, data, checksum)
udt_send(sndpkt)
Wait for data 0 from above
Wait for ACK or NAK 0
rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isNAK(rcvpkt))
udt_send(sndpkt)
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt)
Λ
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt)
Λ
Wait for ACK or NAK 1
Wait for data 1 from above
rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isNAK(rcvpkt))
udt_send(sndpkt)
rdt_send(data)
sndpkt = make_pkt(1, data, checksum)
udt_send(sndpkt)

Figure 8: rdt 2.1 Sender FSM

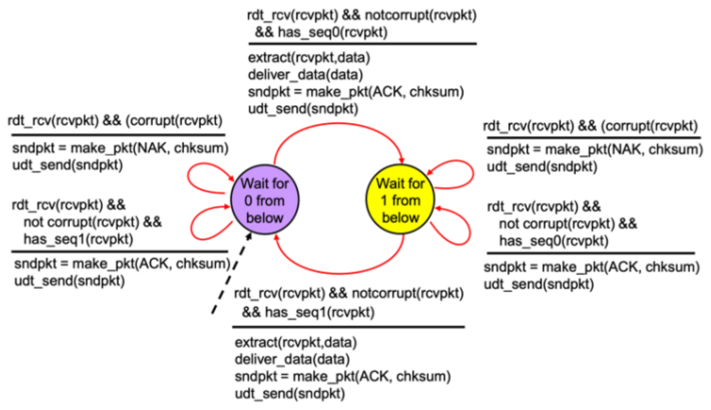

rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && has_seq0(rcvpkt)
extract(rcvpkt,data)
deliver_data(data)
sndpkt = make_pkt(ACK, chksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt) && (corrupt(rcvpkt)
sndpkt = make_pkt(NAK, chksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt) && not corrupt(rcvpkt) && has_seq1(rcvpkt)
sndpkt = make_pkt(ACK, chksum)
udt_send(sndpkt)
Wait for 0 from below
Wait for 1 from below
rdt_rcv(rcvpkt) && (corrupt(rcvpkt)
sndpkt = make_pkt(NAK, chksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt) && not corrupt(rcvpkt) && has_seq0(rcvpkt)
sndpkt = make_pkt(ACK, chksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && has_seq1(rcvpkt)
extract(rcvpkt,data)
deliver_data(data)
sndpkt = make_pkt(ACK, chksum)
udt_send(sndpkt)

Figure 9: rdt 2.1 Receiver FSM

rdt 2.2

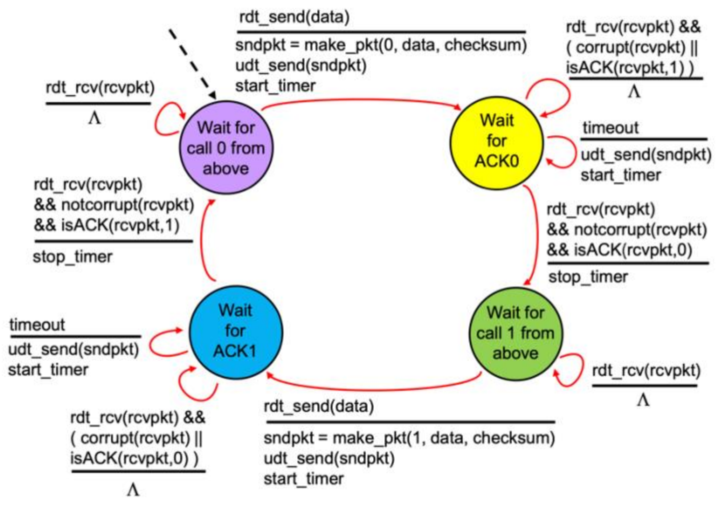
- Use only ACKs

- Receiver sends ACK of last packet received OK
- If same ACK received twice, sender transmit packet again

rdt 3.0

- Assume underlying channel may flip bits, lose packets, and incur arbitrarily long packet delay, but won't re-order packets
- Packet loss detected using timeouts; if no ACK within time limit, will resend packet
    - Delays will create duplicate packets that are discarded
- Receiver ACKs with sequence number of packet
- Poor performance

Scenario	**Sender**	**Receiver**
Duplicate	Does nothing	ACK for previous packet
Corrupted/Lost	Does nothing	Does nothing



rdt_send(data)
sndpkt = make_pkt(0, data, checksum)
udt_send(sndpkt)
start_timer
rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isACK(rcvpkt,1))
Λ
rdt_rcv(rcvpkt)
Λ
Wait for call 0 from above
Wait for ACK0
timeout
udt_send(sndpkt)
start_timer
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt,1)
stop_timer
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt,0)
stop_timer
timeout
udt_send(sndpkt)
start_timer
Wait for ACK1
Wait for call 1 from above
rdt_rcv(rcvpkt)
Λ
rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isACK(rcvpkt,0))
Λ
rdt_send(data)
sndpkt = make_pkt(1, data, checksum)
udt_send(sndpkt)
start_timer

Figure 10: rdt 3.0 Sender FSM

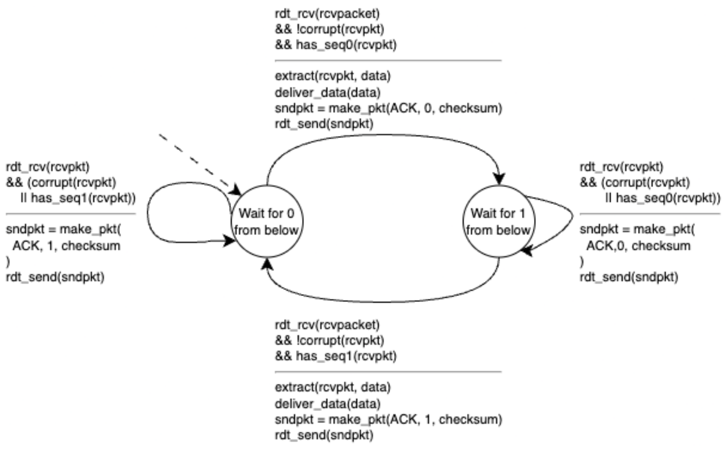
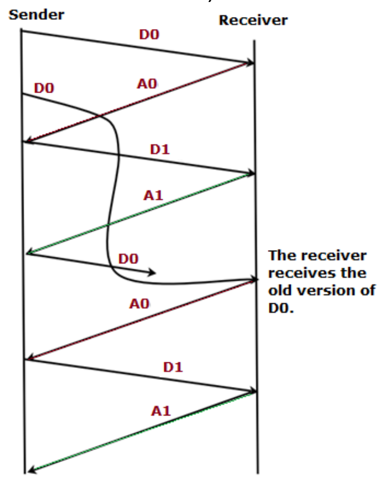


Figure 11: rdt 3.0 Receiver FSM

Reordering allowed: can cause new packets to be lost (if timeout is less than time taken to receive ACK)



Pipelining

- Increases utilization
- Send multiple, in-flight, yet-to-be-acknowledged packets
- Same assumptions as rdt 3.0

$$U_{sender} = \frac{n \times d_{trans}}{RTT + d_{trans}}$$

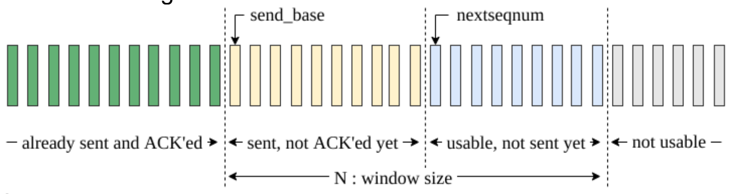
Go-Back-N (GBN)

Sender: contains N unACKed packets in pipeline; packet header contains k-bits sequence number

- Use N-sized sliding window to keep track of unACKed packets
- Timer set on oldest unACKed packet
- When timeout, retransmit packets i till i + N
- Sliding window slides to the right once the left most packet in the window is ACKed
- Each packet sends ACKi
- If packet 1 arrives but packet 0 hasn't, discard packet 1
- Sequence numbers can wrap around to the front

Receiver: only ACK packets that arrive in order, discarding all out-of-order packets

- ACK the last in-order sequence number (i.e. cumulative ACK)
- If ACK3 received on sender, ACK0-3 have all been acknowledged



Selective Repeat

Sender: maintain timer for each unACKed packet and when expires, retransmit only that packet

- Sends packet if window slides
- Sliding window of size N contains exactly N elements, even if it's all but one unACK
- When leftmost packet ACKed, slide window to the right by one

Receiver: individually acknowledge all correctly received packet, buffering out-of-order packets

- If received packet is in sliding window, send ACK and buffer if out-of-order, else slide window to the right
- If received packet is before sliding window, send ACK and do nothing
- Packets outside of window are ignored

IP Address

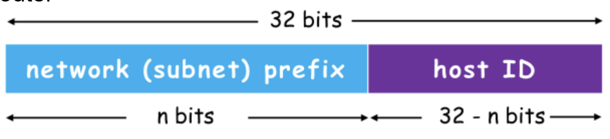
- 32-bit integer expressed in binary or decimal used to identify host or router
- Assigned manually (static IP) or dynamically using Dynamic Host Configuration Protocol (DHCP) server (dynamic IP)
- Associated with a network interface (host has 1 or 2, router has multiple)
    - Routers typically have an IP address per subnet they are connected to

Special IP address:

0.0.0.0/8	Non-routable meta-address for special use (identifies machine before IP address is provided)
127.0.0.0/8	Loopback address; datagram sent to address within this block loops back inside the host
10.0.0.0/8 172.16.0.0/12 192.168.0.0/16	Private address, can be used without any coordination with IANA or an Internet registry
255.255.255.255/32	Broadcast address where all hosts on the same subnet receive a datagram
x.x.x.255	Broadcast address like 255.255.255.255 but can be used for other subnets to broadcast to the current subnet
x.x.x.0	Similar to the meta-address

Subnet: network formed by a group of "directly" interconnected hosts; hosts in the same subnet share the same network prefix

- Hosts in the same subnet can reach each other without using a router



Classless Inter-domain Routing (CIDR): IP address assignment strategy

- Subnet prefix of length x bits where a.b.c.d/x

Subnet mask: determine which subnet an IP address belongs to; first x bits are 1 and the rest are 0

- (IP & mask) to check which subnet it belongs to
- To split a network further, figure out how many addresses per block and how many new control bits are needed
- Amount of available IP addresses in subnet is $2^{\text{size of host ID}}$

Address allocation (hierarchical addressing): ISPs own blocks of IP address (from ICANN) and organisations buy bits within the address space

- E.g. ISP owns 200.23.16.0/20 and it allocates 200.23.18.0/23 to an organisation

Longest prefix match: router uses longest prefix match against forwarding table to decide what's the next router

Prefix	Interface
01	A
Otherwise	B

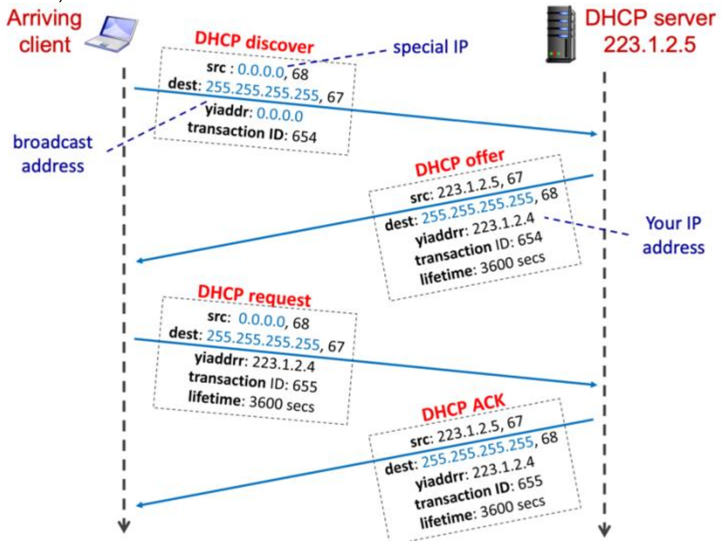
<u>Dynamic Host Configuration Protocol (DHCP)</u>

- Allows host to dynamically obtain its IP address from DHCP server when it joins the network
- IP address is renewable and can be reused
- Run on UDP, server is 67, client is 68
- Allocates both private and public IP addresses

Process:

1. Host broadcasts "DHCP discover" message
2. DHCP server respond with "DHCP offer" message
3. Host requests IP address with "DHCP request" message
4. DHCP server sends address under "DHCP ACK" message

Additional information: IP address of first-hop router, local DNS server, and network mask



<u>Routing</u>

- AS: Autonomous Systems
- Think of as graph with edge costs with nodes as routers and edges as physical links between routers
  - c(x, y): cost of link between routers x and y, inf if not direct neighbors
  - dx(y): cost of the least-cost path from x to y starting from x
    - Calculated using Bellman-Ford
      $$d_x(y) = \min\{\mathrm{c}(\mathrm{x},\mathrm{v}) + \mathrm{d_v}(\mathrm{y})\}$$
    - dv(y) is sent from every direct neighbor to x

1. Intra-AS routing: finds a good path between 2 routers within an autonomous system (RIP and OSPF protocols)
   - Single admin (no policy decisions needed) and focused on performance
2. Inter-AS routing: handles the interface between ASs
   - Admin controls how traffic is routed and policy dominates performance

- Cost of each link that could be 1 or inversely related to bandwidth or related to congestion

<u>Routing algorithm</u>

Link state: all routers have complete knowledge of network topology and link cost; routers periodically broadcast link costs to each other

- Use Dijkstra to compute least cost path locally

Distance vector: routers know physically-connected neighbors and link costs to neighbors; exchange "local views" with neighbors and update own local view

- Iterative process of computation (Bellman-Ford)
  - <u>Swap local view with direct neighbors (i.e. current lowest cost with neighbors)</u>
  - Update own local view
  - Repeat till no more changes to local view
- Every router sends its distance vector to its directly connected neighbors
  - When router finds that y is advertising a path to z that is cheaper than x currently knows, x updates its distance vector to z accordingly and note down that all packets for z should be sent to y (stored in forwarding table)
    $$\exists k \in N(m), \min\{d_m(k), \exists o \in N(k), d_m(k) + c(k,o)\}$$
- Where m is the node in question and N(x) is the direct neighboring nodes of x
- Above formula is how each node processes the shared view of its neighbors

Routing Information Protocol (RIP): uses hop count as cost metric for <u>Distance Vector algorithm</u>

1. Exchange routing table every 30 seconds over UDP port 520
2. Self-repair: if no update from a neighbor for 3 minutes, assume neighbor has failed

<u>Network Address Translation (NAT)</u>

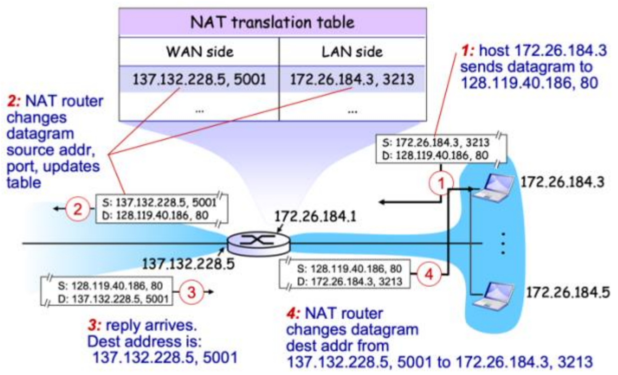
- Translates local network IP address to public IP address
- All datagrams leaving the same local network has the same source NAT IP address

NAT router: replace (source IP, port #) to (NAT IP, new port #)

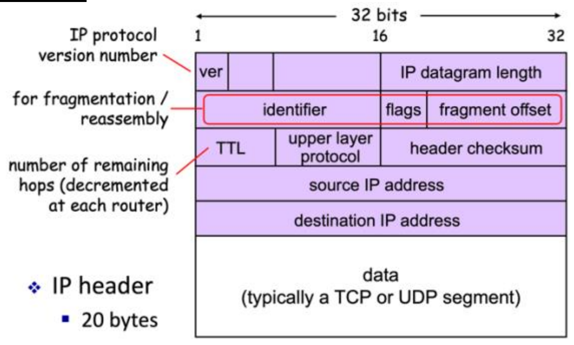
- Contain **NAT translation table** for every mapping
- Replace (NAT IP, new port #) in incoming datagrams to (source IP, port #)
- Replace (source IP, port #) with (NAT IP, new port #) for outgoing datagrams
- LAN <> NAT <> WAN

Motivations:

1. No need to rent range of public IP addresses (1 public IP per NAT router)
2. All hosts use private IP address, can change address of host in local network without affecting outside
3. Can change ISP without affecting hosts
4. Hosts inside local network are not directly addressable and visible to the outside world

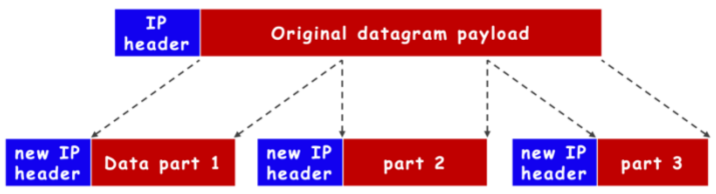


<u>IPv4 Datagram</u>

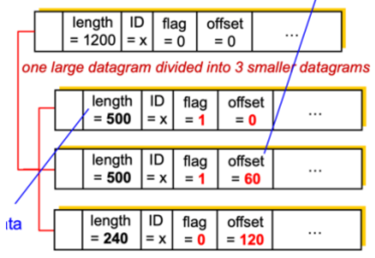


<u>IP Fragmentation</u>

- Large IP datagrams may be fragmented by routers depending on Max Transfer Unit (MTU) of links



- Destination host reassembles packet with IP header fields used to identify fragments and their relative order
- Frag flag set to
  - 1 if there is next fragment from the same segment
  - 0 if it is the last fragment
- Fragment offset expressed in units of 8-bytes
  - Divide the current datagram data size (in bytes) by 8 to derive offset (in bytes)
- Only ID stays the same (identifier), the rest adjust



- Transport layer header is not duplicated

Internet Control Message Protocol (ICMP)

- Used by hosts and routers to communicate network-level information
- Error reporting like unreachable host/network/port/protocol
- Echo request/reply (ping)
- Does not use TCP
- Carried in IP datagrams, ICMP header after IP header
  - Type + code + checksum + others

Type	Code	Description
8	0	Echo request (ping)
0	0	Echo reply (ping)
3	1	Destination host unreachable
3	3	Destination port unreachable
11	0	TTL expired
12	0	Bad IP header

Link Layer

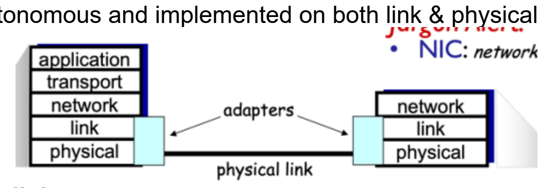
- Sends datagram between adjacent nodes (hosts or routers) over a single link
- IP datagrams are encapsulated into frames for transmission
- Each link can have different protocols

Data-link Layer: responsibility of transferring datagram from one node to physically adjacent node over a link

Services:

1. Framing: encapsulate datagram into frame by adding header and trailer
2. Link access control: coordinating which nodes can send frames at a certain point of time over a shared medium
3. Error detection: detect presence of errors caused by signal attenuation or noise; can signal sender for retransmission or drop frame
4. Error correction: receiver identifies and corrects bit errors without retransmission
5. Reliable delivery: used more often on error-prone links like wireless

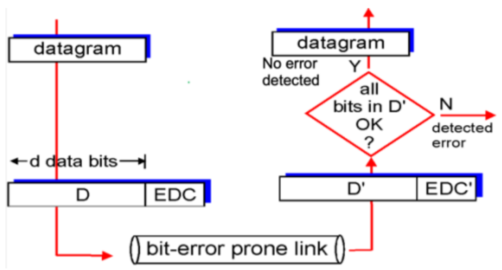
Network Interface Card (Adapter): implementation of link layer; semi-autonomous and implemented on both link & physical layers



Network links:

- Point-to-point: dedicated link
- Broadcast link: shared medium and connected via shared broadcast channel; every node receives a copy

Error Detection



- Larger EDC fields yield better detection & correction
- Includes checksum, parity checking, CRC

Checksum

- Found in both TCP, UDP, and IP (only for IP header)
- Detects “errors” in transmitted segment
- Not guaranteed that there are no errors, but can detect when there are since the sum could be interchanged
- TCP and UDP include the headers for checksum generation

Sender: compute checksum value and put checksum value into transport layer header

Receiver: compute checksum of received segment and check if computed checksum is the same as checksum field value

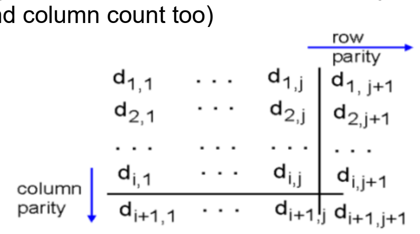
Computation:

1. Break segment into sequence of 16-bit integers
2. Apply binary addition on every 16-bit integer
3. Add any excess carry to the result
4. Compute the 1’s complement (i.e. invert)

Parity Checking

- Given even parity scheme and d bits, parity bit set to 1 if d + 1 contains even bits, else 0 (set to 1 if d bits has odd number of 1s)
- Odd parity scheme inverts this
- Detects single bit errors (for even parity, detects odd number of single bit errors only)
- Works well mathematically as probability of multiple bit errors is low and errors are independent
  - Clusters of errors (bursts) can increase probability of undetected errors to 50%

2D Parity Checking: d bits divided into i rows and j columns; calculate parity value per row and column with i + j + 1 parity bits (added row and column count too)



$$\begin{array}{cccc|c}
d_{1,1} & \cdots & d_{1,j} & & d_{1,j+1} \\
d_{2,1} & \cdots & d_{2,j} & & d_{2,j+1} \\
\cdots & \cdots & \cdots & & \cdots \\
d_{i,1} & \cdots & d_{i,j} & & d_{i,j+1} \\
d_{i+1,1} & \cdots & d_{i+1,j} & & d_{i+1,j+1}
\end{array}$$

- Can detect and correct single bit errors (mismatch of row/column parity bits)
- Can detect any two-bit error
- May not detect four-bit errors

Cyclic Redundancy Check

- Generate R (redundancy code)
- D is data to be sent with d bits, G is generator (pre-determined value) with r + 1 bits, R is the redundancy code with r bits

Sender: perform division with modulo 2 (XOR) of D over G, append r 0s to D beforehand

- Remainder is R
- Keep bringing down bits until length is >= r + 1 bits
- Sends (D, R)

Receiver: divides (D, R) by G; if remainder is 0, no errors

- Don’t care about quotient, just bring down all remaining digits after XOR and verify

Implemented on Link Layer: easy to implement on hardware; detects all odd number of single bit errors

- r bit R detects all burst errors of less than r + 1 bits and all burst errors of greater than r bits with probability 1 – 0.5^r

Polynomial code: k-bit frame is coefficient list for polynomial with k terms from x^{k-1} to x^0

Multiple Access Protocols

- Only prevalent in shared broadcast channels (rate R bps)
- Coordination about channel sharing must use the channel itself (no out-of-band channel signalling)

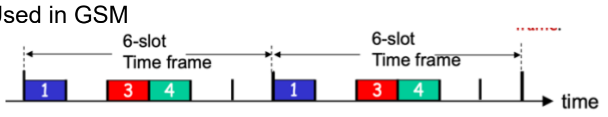
Desired properties: given broadcast channel of rate R bps
1. Collision free
2. Efficient: when only one node wants to transmit, it can send at rate R
3. Fairness: when M nodes want to transmit, each can send at an average of R/M
4. Fully decentralized: no special node to coordinate
5. [Mandatory] No out-of-band channel signalling: coordination about channel sharing must use the channel itself

Categories of protocols: in increasing complexity **
1. Channel partitioning: dividing the channel into fixed smaller pieces and allocate each piece to a node for exclusive use
2. Taking turns: each node takes turns to transmit
3. Random access: collisions are possible and must be able to recover from collisions

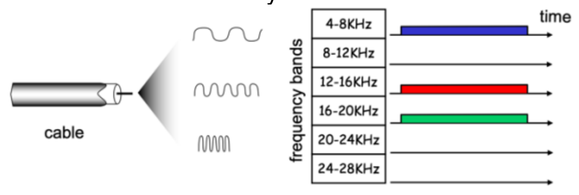
	Collision?	Efficiency	Fairness	Decentralized
TDMA	Yes	Unused slots go idle, maximum throughput is R/N	Perfectly fair	Yes
FDMA	Yes	Unused slots go idle, maximum throughput is R/N	Perfectly fair	Yes
Polling	Yes	Highly efficient but polling has overhead	Perfectly fair	No as master node is a single point of failure
Token Passing	Yes	Highly efficient but token passing has overhead	Perfectly fair	Yes
Slotted ALOHA	No	Yes only if one node is active, multiple nodes have maximum efficiency of 37%; slots wasted due to collision and empty	Perfectly fair	Yes
ALOHA	No	Yes only if one node is active, multiple nodes have maximum efficiency of 18%; slots wasted due to collision and empty	Perfectly fair	Yes
CSMA/CD	No	Yes	Yes	Yes

Channel Partitioning

Time Division Multiple Access (TDMA): access to channel in rounds and each node gets fixed length time slots in each round
- Length of timeslot is the data frame transmission time
- N-time slots form a frame
- Used in GSM



Frequency Division Multiple Access (FDMA): channel spectrum divided into frequency bands with each node assigned a fixed frequency bad
- Unused transmission time in frequency bands go idle
- Used in radio and satellite systems



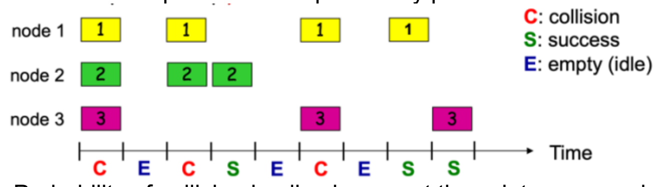
Taking Turns

Polling: master node polls each node in round-robin fashion informing them to transmit some maximum number of frames and repeats this for all nodes; master node can be single point of failure
- Used in Bluetooth

Token passing: special frame (token) passed from one node to another
- Each node holds onto the token only if it has frames to transmit and sends a maximum number of frames
- Forwards token to the next node
- Token loss can be disruptive and node failure could break the ring
- Used in FDDI and token ring

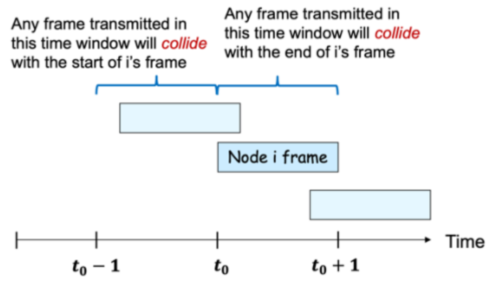
Random Access
- Each node transmits at rate R as soon as it has data to send without prior coordination
- Collisions occur when two or more nodes transmit at the same time
- Random access protocols detect and recover from collisions

Slotted ALOHA: all frames are of equal size L bits and time is divided into slots of equal length
- Length of slot is time to transmit one frame: L/R
- Nodes start transmitting at the beginning of a slot (synchronised timing)
- When node has a fresh frame to send, it waits for the beginning of next slot and transmits the entire frame in the slot
  - No collisions: data transmission is successful
  - Collision: data transmission failed; retransmit the frame in each subsequent slot with probability p until success

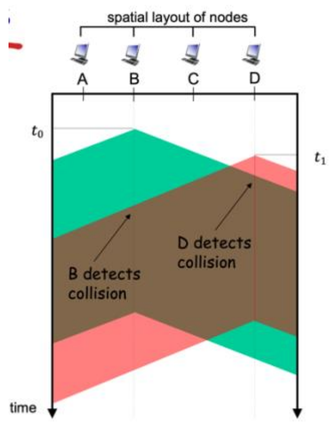


- Probability of collision in all subsequent time slots are equal or worse if more nodes transmit
- Used in wireless packet switched network

ALOHA: slotted ALOHA without time slots and synchronisation
- When node has a fresh frame to send, it transmits the frame immediately
  - No collisions: successful
  - Collision: failed; wait for 1 frame transmission time to retry with probability p until success
- Chance of collision increases as frame sent at t0 collides with other frames sent in t0 – 1 and t0 + 1


Any frame transmitted in this time window will collide with the start of i's frame
Any frame transmitted in this time window will collide with the end of i's frame
Node i frame
Time
$t_0 - 1$
t_0
$t_0 + 1$

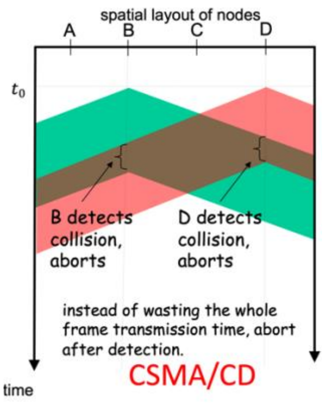
Carrier Sense Multiple Access (CSMA): listen before transmitting (opposite of ALOHA's send when free)



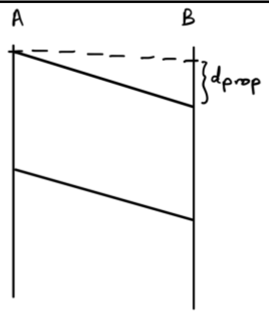
- Collisions occur as nodes may not hear of each other's transmissions immediately due to **propagation delay**

- Collisions detected when the colliding packet reaches the other side
- Nodes do not stop transmitting even with collision

CSMA/CD: aborts transmission if collision detected and retransmission occurs with random delay


spatial layout of nodes
A
B
C
D
t_0
B detects collision, aborts
D detects collision, aborts
instead of wasting the whole frame transmission time, abort after detection.
CSMA/CD
time

- Retransmission attempts adapt to estimated current load where more collisions imply heavier loads so longer back-off interval required
- **Backoff algorithm:**
  - After collision 1: choose K at random from {0, 1} and wait K * 512 bit time before retransmission
  - After collision 2: choose K from {0, 1, 2, 3}
  - After collision m: choose K from {0, 1, …, 2^m - 1}
- Frame too small can cause collisions to go undetected
- Used in Ethernet
- Bit time: time it takes to transmit 1 bit of data
- **<u>This module uses 512 bit time for Ethernet</u>**



- Any transmissions from B during the propagation delay will result in a collision on A provided that the frame is large enough

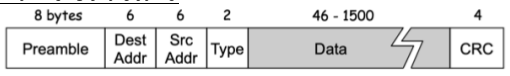
<u>MAC Address</u>
- Every adapter (NIC) has a MAC address
- Used to send and receive link layer frames
- When NIC receives frame, check destination MAC address = own MAC address
  - Extracts enclosed datagram if match else discard
- 48 bits burned in NIC ROM
- Administered by IEEE (first 3 bytes is vendor)

Broadcast address: FF-FF-FF-FF-FF-FF

Address Resolution Protocol (ARP): routing IP address to MAC
- <IP, MAC, TTL>
- Stored on each IP node
- If IP <> MAC not known, sender sends broadcast frame and receiver responds, sender saves IP <> MAC to ARP table
- ARP frame sets target MAC to 00:00… and receiver uses IP to detect if it's the right device to respond
- TTL is the time after which the mapping is forgotten (few minutes)

<u>Local Area Network (LAN)</u>
- Computer network that interconnects computers within a geographical area like office building or university campus
- Ethernet is the dominant wired LAN technology

<u>Ethernet Frame Structure</u>



- Source/Destination MAC address: check if destination = own
- Data: maximum size of 1500 bytes (MTU), minimum of 46 bytes
- CRC: corrupted frame will be dropped
- Type: higher level protocol like IP
- Preamble: 7 bytes with pattern of AA hexadecimal
  - Next byte is AB hexadecimal (start of frame)
  - Used to synchronize receiver and sender clock rates
  - Provides "square wave" pattern to tell the receiver the sender's clock rate

<u>Ethernet Data Delivery Service</u>
- Unreliable as receiving NIC does not send ACK or NAK to sending NIC
- Use CSMA/CD as multiple access protocol

<u>Ethernet Topologies</u>

Bus	Broadcast LAN where all transmitted frames received by all adapters connected to bus so all nodes can collide with each other • Requires backbone cable so single point of failure; difficult to troubleshoot problems; slow and not ideal for larger networks
Star	Nodes directly connected to central device that routes the requests • Hub: physical-layer device that acts on individual bits, not frames ○ Re-creates incoming bit, boosts energy strength, or transmits bit onto all other interfaces ○ Cheap and easy to maintain but very slow and unideal for larger networks • Switch: layer-2 device that works on frames, not bits ○ No collisions, works as store-and-forward packet switch

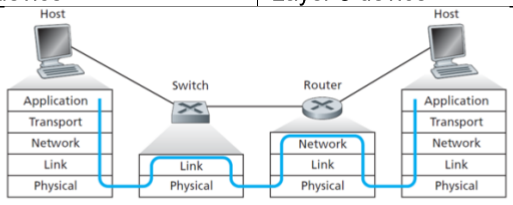
<u>Switch</u>
- Link-layer device
- Examines incoming frame's MAC address and selectively forwards frame to one-or-more outgoing link
- Stores and forwards Ethernet frames
- Uses CSMA/CD to access link
- Transparent as hosts are unaware of presence of switch
- Plug-and-play that does not need to be configured
- All nodes have a dedicated and direct connection to switch
- Buffers packets
- Can be interconnected with other switches
- Allows simultaneous transmissions

Switch forwarding table: <MAC address, interface to reach host, TTL>
- Self-learning every time a frame is received, switch learns location of sender
- Records sender/location pair in switch table
- If Wi-Fi access point connected, each connected device on Wi-Fi access point has a separate entry but same interface
- MAC address can also point to router if needs to go through router

Frame filtering/forwarding: when frame received at switch
1. Record incoming link, MAC address of sending host
2. Index switch table using MAC destination address
3. If entry found for destination,
   a. If destination on segment from same device, drop frame
   b. Else, forward frame on interface indicated by entry
4. Else, flood by forwarding on all interfaces except arriving interface

<u>Switch vs Router</u>

Switch	Router
Check MAC address	Check IP address
Store and forward	Store and forward
Forward frame to outgoing link or broadcast	Compute routes to destination
Layer 2 device	Layer 3 device



- Routers can have multiple IP/MAC addresses
- Switches don't have an IP address

Sending data across subnets

1. Sender creates IP datagram to receiver
2. Sender attempts to create frame to receiver
3. Sender searches ARP table for IP <> MAC
   a. Has mapping: use mapping MAC
   b. No mapping: broadcast and cache mapping; if no responses, then need to forward to router, else forward internally
4. Sender sends frame to router
5. Router reads IP of destination and creates frame to receiver's MAC address (with same ARP search)
6. Router searches routing table for mapping
   a. Has mapping: use mapping
   b. No mapping: send broadcast frame and discover; if no responses, then need to forward to next hop router
7. Receiver receives fra)me
   a. Updates ARP if it is the destination for the sender

ARP & Routing

- Both update when receiving a request

Network Attacks

1. Eavesdrop: intercept messages
2. Insert messages into connection
3. Impersonation: fake/spoof source address in packet
4. Hijacking: "take over" ongoing connection by removing sender/receiver; i.e. man in the middle
5. Denial of service: prevent service from being used by others

Network Security

1. Confidentiality (cipher/symmetric/public keys): only sender, intended receiver should understand message contents
2. Authentication (signature): sender and receiver can confirm identity of the other
3. Message integrity (hashing): sender and receiver can ensure that message is not altered without detection
4. Access and availability (firewall): services must be accessible and available to users

Fixed Ciphers

1. Substitution cipher: substitute one thing for another like fixed shift of alphabet (Caesar's Cipher); easy to bruteforce search
2. Monoalphabetic cipher: substitute one letter for another with 26! mappings
   - Can be broken with statistical analysis: most frequently used letters, occurrences, knowledge of possible contents
3. Polyalphabetic encryption: cyclic pattern of monoalphabetic substitutions
4. Block cipher: message to be encrypted is processed in blocks of K bits and each block is independently encrypted using one-to-one mapping
   - 2^K! keys
   - Used for Data Encryption Standard (DES) and Advanced Encryption Standard (AES)

Breaking Encryption Schemes

1. Ciphertext only attack: analysing existing ciphertext
2. Known-plaintext attack: has both mapping of plaintext and corresponding ciphertext
3. Chosen-plaintext attack: retrieving the ciphertext for a chosen plaintext

Types of Cryptography

1. Symmetric Key: sender and receiver use the same key KA = KB
2. Asymmetric Key: sender and receiver use different keys KA != KB; aka public key cryptography

Public Key Cryptography

- Sender and receiver do not share a secret key
- Sender uses receiver's public key and receiver uses private key to decipher

Requirements:

1. Public and private key must form an identity function
$$m = K_B^-(K_B^+(m))$$
2. Given public key, private key should not be able to be computed

Modular Arithmetic: x mod n = remainder of x when divided by n
$$[(a\,mod\,n) + (b\,mod\,n)]mod\,n = (a + b)mod\,n$$
$$[(a\,mod\,n) - (b\,mod\,n)]mod\,n = (a - b)mod\,n$$
$$[(a\,mod\,n) \times (b\,mod\,n)]mod\,n = (a \times b)mod\,n$$
$$(a\,mod\,n)^d mod\,n = a^d mod\,n$$

Rivest, Shamir, Adleman (RSA) Algorithm

Message: bit pattern that is uniquely represented by an integer number

Creating public/private key pair:

1. Choose two large prime numbers, p and q
2. Compute n = pq and z = (p – 1)(q – 1)
3. Choose e, where e < n, that has no common factors with z
4. Choose d such that ed – 1 is exactly divisible by z
5. Public key is (n, e)
6. Private key is (n, d)

Encryption: $c = m^e mod\,n$

Decryption: $c^d mod\,n$

Order of application: order does not matter

Secure: to break RSA, you require the factors of n without knowing the factors p and q which is hard

Session keys: used to transfer symmetric session key for DES

Cryptographic Hash Function

- Given message m, generate hash H(m)
- Computationally infeasible to find any x and y such that x != y and H(x) == H(y)
- Provides message integrity (must ensure detection of m and m')

Examples: MD5, SHA-1, SHA-2, SHA-3

MD5: fixed 128-bits; small change to m results in big change to H(m)

Message Authentication Code (MAC)

- Sender and receiver share authentication key, s
- Send(m, H(m + s)) where H(m + s) is the MAC
- Ensures message integrity as attacker cannot send (m', H(m')) and go unnoticed
- Similar to salting in password hashing

Password hashing: passwords cannot be recovered, only reset

Digital Signature

- Verifiable and unforgeable proof that sender generated the message
- Provides message authentication
- Sender signs message using private key and receiver uses sender's public key to decrypt and if message the same, then the message is signed by sender
  - (m, K^-_B (m)) is sent and decrypt to check that m = m' (encrypted)

Optimizations: generate fixed-length and easy-to-compute digital fingerprint so any message length takes the same time to generate (hash function)

- Sender: message -> hash function -> encrypt
- Receiver: decrypt -> generate hash from message -> compare

Comparison to MAC: digital signature requires a private key and hash function while MAC requires a shared key and hash function

- Digital signature: K(H(m))
- MAC: H(m + s)

Certification Authorities

- Public database to map person to public key
- Signs messages to verify that it hasn't been tampered with
- Maintained by OS list of "Trusted Root Certification Authorities"

Registration: provide proof of identity to CA and CA creates certificate binding entity to public key (digitally signed by CA)

- Receiver: get public key -> CA public key to get sender key -> sender public key to get message

Firewalls

- Isolate organization's internal net from larger Internet, allowing some packets to pass, blocking others
- Prevents DoS, illegal modification/access to internal data, and allows only authorized access to inside network

Types:

- Stateless packet filters
- Stateful packet filters
- Application gateways

Limitations:

1. IP spoofing: router does not know if data comes from claimed source
2. Bottleneck
3. Trading degree of communication with outside world for security
4. Not full proof

Stateless packet filtering: internal network connected to Internet via router firewall and router filters packet-by-packet, decision to forward/drop packet based on:

- Source/Destination IP
- TCP/UDP source/destination port
- ICMP message type

- TCP SYN/ACK bits

Policy	Setting
All incoming/outgoing UDP flows blocked	Block incoming and outgoing datagrams with IP protocol field = 17
Prevent external clients from making TCP connections with internal clients but allow the opposite	Block inbound TCP segments with ACK=0 (handshake)
No outside web access	Drop all outgoing packets to any IP address, port 80
No incoming TCP connections except those for institutions public Web server only	Drop all incoming TCP SYN packets to any IP except web server IP + port 80
Prevent web-radios from eating up available bandwidth	Drop all incoming UDP packets except DNS and router broadcasts
Prevent network from being used for smurf DoS attack	Drop all ICMP packets going to "broadcast" address
Prevent network from being tracerouted	Drop all outgoing ICMP TTL expired traffic

Access Control Lists

- Table of rules applied top to bottom to incoming packets, <action, condition>

action	source address	dest address	protocol	source port	dest port	flag bit
allow	222.22/16	outside of 222.22/16	TCP	> 1023	80	any
allow	outside of 222.22/16	222.22/16	TCP	80	> 1023	ACK
allow	222.22/16	outside of 222.22/16	UDP	> 1023	53	---
allow	outside of 222.22/16	222.22/16	UDP	53	> 1023	----
deny	all	all	all	all	all	all

Secure E-mail

AInitial:

1. Given email m, sender generates random symmetric private key S and encrypts m with S
2. Sender encrypts S with receiver's public key R
3. Sender sends S(m) and R(S) to receiver
4. Receiver uses private key to decrypt S
5. Receiver uses S to decrypt m

Providing authentication message integrity: sender digitally signs message and sends to receiver to verify

- Sender uses own private key, receiver's public key, and symmetric private key

Multimedia Networking

- Any network application that employs audio or video

Streaming: can being playout before downloading entire file

Stored: on server/Content Distribution Network (CDN); implies transmission faster than audio/video can be rendered, i.e. storing/buffering at the client

Streaming stored audio/video: YouTube, Netflix, Hulu

Conversational ("two-way live") voice/video over IP: interactive nature of human-to-human conversation limits delay tolerance (any delay > 400ms is intolerable); Skype, Zoom, WhatsApp

Streaming live ("one-way live") audio/voice: done via CDNs; live sporting events (soccer, football)

Video: sequence of images displayed at a constant rate

- High bit rate

Constant Bit Rate	Variable Bit Rate
Video encoding rate fixed • Not responsive to complexity of video • Bit rate set relatively high to handle more complex segments of video • Well-suited for real-time encoding	Video encoding rate changes as amount of spatial and temporal coding changes • Best suited for on-demand video due to longer time to process data

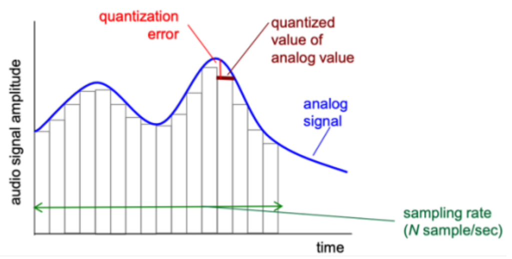
Digital image: array of pixels (each pixel represented by bits)

Video compression: reduce data usage

- Use redundancy within and between images to decrease number of bits to encode image
- Spatial coding (within): instead of sending N values of the same color, send the color + N times
- Temporal coding (between): instead of sending the complete frame i + 1, send differences from frame i

Audio: analog audio signal sampled at a constant rate (continuous function)

- Each sampled quantized (rounded) with quantized value represented by bits
- Quantized values/levels are preset amounts assigned to a range of analog value (i.e. convert continuous value to discrete value)
  - Think of it as bits per sample

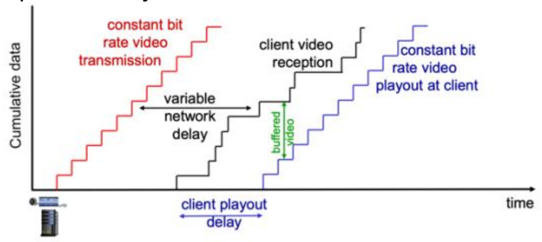


- Calculating bit rate: sample rate * quantized value bit count

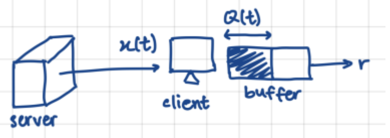
Streaming Stored Video

Challenges:

- Continuous playout constrained by variable network delays (jitter), client playback cannot match original timing
- Client interactivity
- Video packets may be lost and retransmitted



Client-side buffering: stores buffer of server-sent bits; playout reads from this buffer instead of directly from server



- x(t) is the variable fill rate, Q(t) is the buffer fill level, buffer size of B, and playout rate of r (constant bit rate)
- Fill buffer till playout at t_p
- If average fill rate < r then buffer eventually empties and causes freezing of video playout, otherwise the buffer will never empty
  - If the buffer is full, it creates "back pressure" that causes the server to slowdown transmission
- Measure in chunks, obtained by dividing size by playout bit rate

Streaming Multimedia: UDP

- Server sends at rate appropriate for client with send rate = encoding rate = constant rate (push-based streaming)
- UDP has no congestion control so no rate control restrictions
- Short playout delay (~2-5 seconds) to remove network jitter
- Error-recovery at application level and time permitting

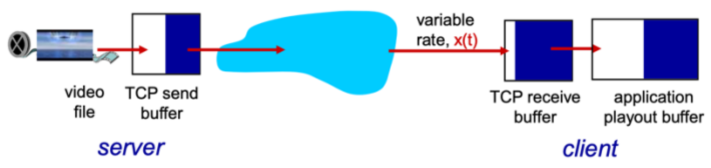
Implementation:

- Real time Transport Protocol (RTP): encapsulates video chunks
  - Sequence number, timestamp, and video encoding
- Real Time Streaming Protocol (RTSP): maintain control connection
  - Establish & control media sessions between endpoints; allow clients to issue commands like play, record, and pause

Drawbacks: need for separate media control server like RTSP which increases cost and complexity; UDP may not go through firewalls

Streaming Multimedia: HTTP

- File retrieved using HTTP GET (pull-based streaming)
- File sent at maximum possible rate under TCP
- TCP receive buffer != client application buffer



Advantages: HTTP/TCP passes more easily through firewalls; network infrastructure like CDNs and routers work well for HTTP/TCP
Drawbacks: fill rate fluctuates due to TCP congestion control and retransmission; larger playout delay due to TCP delivery rate

<u>**Conversational Multimedia: Voice-over-IP (VoIP)**</u>

End-end-delay requirement: maintain "conversational" aspect

- Higher delays are noticeable and impair interactivity; includes application-level network delays
- < 150ms is good while > 400ms is bad
- Data loss over 10% makes the conversation unintelligible
- Challenge: IP is best-effort service; no upper bound on delay (when to play) or % of packet loss (what to do with missing chunk)

Characteristics:

- Speaker audio: alternating talk spurts and silent periods, packets generated only during talk spurts; 20ms chunk at 8Kbps = 160 bytes of data
- Application-layer header: added to each chunk
- Chunk + header encapsulated in UDP/TCP segment: sent during talk spurt

Loss tolerance: 1 – 10%

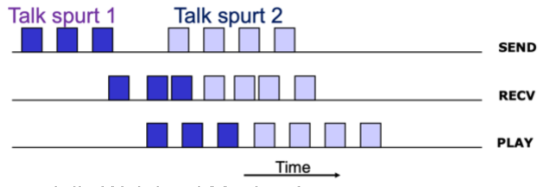
- Network loss: due to network congestion
- Delay loss: arrive too late for playout
  - Caused by processing + queuing delays
  - Maximum tolerable delay: 400ms
  - Use UDP to avoid congestion control

Fixed playout delay: receiver attempts to playout each chunk exactly q ms after chunk is generated (received)

- Chunks contain sequence number and timestamp
- Any arrival > t + q is lost (otherwise will mess up other chunks)
- No optimal q: large q imply less packet loss but small q imply better interactive experience but easier to lose the chunk

Adaptive playout delay: provide low playout delay and low late loss rate

- Estimate network delay and adjust playout delay at the beginning of each talk spurt
- Silent periods compressed and elongated with chunks still played out every 20ms during talk spurts
- Forcibly spacing out the playout based on receive



- Exponentially Weighted Moving Average:

$$d_i = (1-\alpha)d_{i-1} + \alpha(r_i - t_i)$$

  - d_i = delay estimate after ith packet
  - α = small constant
  - r_i = time received
  - t_i = time sent

$$v_i = (1-\beta)v_{i-1} + \beta|r_i - t_i - d_i|$$

  - v_i = estimate of average deviation of delay after ith packet
  - β = small constant
- d_i and v_i calculated every packet but used at start of talk spurt
- Playout time = $t_i + d_i + 4v_i$ (first packet of talk spurt)
  - The rest of packets to play periodically

Packet recovery: ACK/NAK takes about 1 RTT which is too long

- Forward Error Correction: send enough bits to allow recovery without retransmission

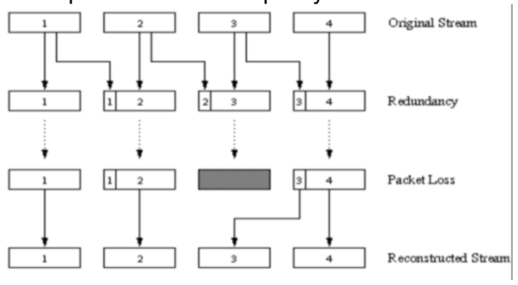
<u>**VoIP Packet Recovery Strategies**</u>

Simple FEC: for every group of n chunks, create redundant chunk by XOR-ing the other n chunks and send this as n + 1 chunks

- Reconstruct original n chunks if at most 1 lost chunk with playout delay
- Increases bandwidth by factor of 1/n and playout delay increases during packet loss (receiver waits for n + 1 chunks before playout)
- Does not work if n <= 2

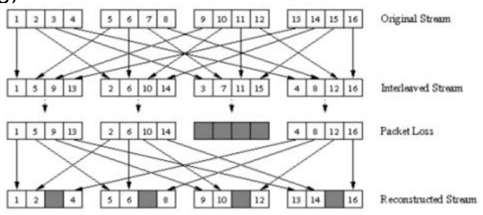
Piggyback lower quality stream FEC: send lower resolution audio stream as redundant information

- Receiver can conceal loss and can be generalized (append n-1 and n-2 low-bit rate chunk)
- Can cause spots of low audio quality



Interleaving to conceal loss: audio chunks divided into smaller units and each packet contains small units from different chunks

- If packet lost, still have most of every original chunk, concealed by packet repetition or interpolation
- No redundancy overhead but increases playout delay (re-ordering)



<u>**HTTP Streaming**</u>

Video-on-Demand: streaming over HTTP using GET on whole video

- Can be wasteful and requires large client buffer
- All clients receive the same encoding regardless of device/network bandwidth

Dynamic Adaptive Streaming over HTTP (DASH):

- Server
  - Divides video file into multiple chunks and stores each chunk with different encoding rates
  - Contains manifest file, providing URLs for different encoding
- Client
  - Downloads chunk before requesting for data
  - Periodically measures server-to-client bandwidth
  - Consults manifest file for chunk that is maximally sustainable given bandwidth (adaptive bitrate algorithm ABR)
  - Determines when to request chunk (buffer starvation/overflow avoided), what encoding rate to request (higher quality when more bandwidth available), and where to request chunk (can request from URL server closest to client)
- Advantages:
  - Simple server
  - No firewall problems (port 80 for HTTP)
  - Standard image web caching works
- Disadvantages:
  - DASH is based on media segment transmissions, typically 2-10 seconds in length
  - Does not provide low latency for interactive, two-way applications (buffers on client side)

<u>**Content Distribution Networks (CDNs)**</u>

- Store/serve multiple copies of videos at multiple geographically distributed sites (CDN)
- Client requests content and service provider returns manifest
  - Client uses manifest to retrieve content at highest supportable rate and may choose different rate or copy if network path is congested

Server placement philosophies:

- Enter deep: push CDN servers deep into access networks (closer to users)
- Bring home: smaller number of large clusters in Internet Exchange Points but not within access networks

Extra Knowledge

Commands

telnet: establish TCP connection to a remote server; requires DNS lookup and TCP handshake but no other requests made
netcat: similar to telnet but works with UDP
nslookup: determine the DNS records of a given domain name (provides the host <> IP mapping)
dig: gathers DNS information

- Focus on the "ANSWER" section
- DNS server that responded found under final section "SERVER"

Packet Length Information

UDP: length includes body and header
TCP MSS: does not include header
IP MTU: includes both IP header and body (which in turn contains TCP/UDP headers)

Checksum

UDP/TCP: includes header, body, and pseudo IP header
IP: only IP header