Chapter 1

- protocols -> define format, order of msgs sent and received among network entities, and actions taken on msg transmission, receipt
- network structure:
  - network edge
    - ✓ links
    - ✓ access network
    - ✓ end system
  - access networks, physical media
  - network core
- frequency division multiplexing -> different channels transmitted in different frequency bands
- Physical media:
  - Bit: propagates between transmitter/receiver pairs
  - Physical link: what lies between transmitter & receiver
  - Guided media: signals propagate in solid media: copper, fiber, coax
  - Unguided media: signals propagate freely, e.g., radio
- twisted pair (TP) -> two insulated copper wires
- coaxial cable:
  - two concentric copper conductors
  - bidirectional
  - broadband
- fiber optic cable:
  - glass fiber carrying light pulses, each pulse a bit
  - high-speed operation: high-speed point-to-point transmission
  - low error rate: repeaters spaced far apart, immune to electromagnetic noise
- radio:
  - signal carried in electromagnetic spectrum
  - no physical “wire”
  - bidirectional
  - propagation environment effects -> reflection , obstruction by objects , interference
- radio link types -> (terrestrial microwave, WLAN, wide-area, satellite)
- packets -> smaller chunks
- link transmission rate = link capacity = link bandwidth
- packet transmission delay -> time needed to transmit L-bit packet into link (L / R)
- end-end delay = 2L/R
- what is happen If arrival rate (in bits) to link exceeds transmission rate of link for a period of time?
  - packets will queue, wait to be transmitted on link
  - packets can be dropped (lost) if memory (buffer) fills up

- routing -> determines source-destination route taken by packets (routing algorithms)
- forwarding -> move packets from router ’s input to appropriate router output
- circuit switching:
  - end-end resources allocated to, reserved for “call” between source & dest
  - dedicated resources: no sharing
  - circuit segment idle if not used by call
  - Commonly used in traditional telephone networks
- transmission rate of a circuit = frame rate * number of bits in a slot
- number of slots = number of users
- capacity one user = rate / number of users
- packet switching:
  - hosts break application-layer messages into packets
  - each packet transmitted at full link capacity
  - great for non-continuous data (resource sharing , simpler, no call setup)
  - excessive congestion possible (packet delay and loss)
- nodal processing (d_{proc})
  - check bit errors
  - determine output link
- queueing delay ($d_{queue} = La/R$)
  - time waiting at output link for transmission
  - depends on congestion level of router
  - La/R ~ 0: avg. queuing delay small
  - La/R < 1: avg. queuing delay large
  - La/R > 1: more “work” arriving than can be serviced, average delay infinite!
- transmission delay ($d_{trans} = L/R$)
- propagation delay ($d_{prop} = d/s$)
- $d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop}$
- bottleneck link -> link on end-end path that constrains end-end throughput
- Data Encapsulation -> each layer adds its own header (and sometimes trailer) to data supplied by higher layer
- On receiving, data is De-Encapsulated (headers and trailer are removed)
- Packets doesn’t contain user data only but some protocol-related data
- Application layer -> message
- Transport layer -> segment
- Network layer -> datagram
- Link layer -> frame
- It is NOT necessary for each device to run protocols of all layers
- Switches run in Link Layer while Routers run in Network Layer

Chapter 2

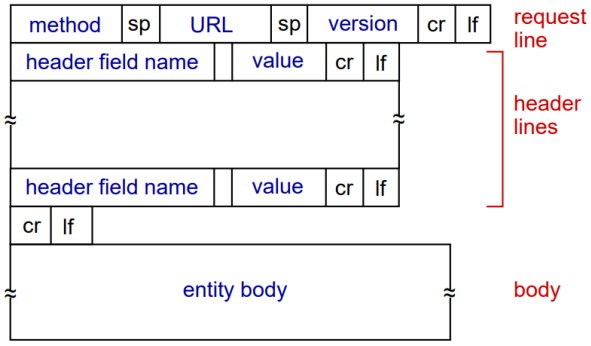
- Application Layer
  - provides an interface between software program and the network itself.
  - Protocol Examples: HTTP, FTP, SMTP
- Application architectures:

Client-server architecture	P2P architecture
Server always-on host	no always-on server
Clients do not communicate directly with each other	arbitrary end systems directly communicate
must sequentially send (upload) N file copies: ✓ time to send one copy: F/u_s ✓ time to send N copies: NF/u_s	must upload at least one copy: ✓ time to send one copy: F/u_s
time to distribute F to N clients using client-server approach $D_{c-s} \geq max\left\{\frac{F}{d_{min}}, \frac{NF}{u_s}\right\}$ increases linearly in N	time to distribute F to N clients using P2P approach $D_{P2P} \geq max\left\{\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{u_s + \sum_{i=1}^{N} u_i}\right\}$ increases linearly in N but so does this, as each peer brings service capacity

- P2P architecture:
  - self scalability – new peers bring new service capacity, as well as new service demands
  - Hybrid systems
- P2P architecture face three major challenges:
  - ISP Friendly
  - Security
  - Incentives
- Process -> program running within a host
  - client process -> process that initiates communication
  - server process -> process that waits to be contacted
- processes in different hosts communicate by exchanging messages
- applications within P2P architectures have client processes & server processes
- TCP service:
  - reliable transport between sending and receiving process
  - flow control: sender won' t overwhelm receiver
  - congestion control: throttle sender when network overloaded
  - does not provide: timing, minimum throughput guarantee, security
  - connection-oriented: setup required between client and server processes
- UDP service:
  - unreliable data transfer between sending and receiving process
  - does not provide: reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup
- HTTP is stateless
- HTTP connections:

non-persistent HTTP	persistent HTTP
downloading multiple objects required multiple connections	multiple objects can be sent over single TCP connection between client, server
requires 2 RTTs per object	as little as one RTT for all the referenced objects
browsers often open parallel TCP connections to fetch referenced objects	server leaves connection open after sending response

- Round Trip Time (RTT) -> time for a small packet to travel from client to server and back
- non-persistent HTTP response time = 2RTT+ file transmission time
- in non-persistent HTTP, OS overhead for each TCP connection
- HTTP request message general format

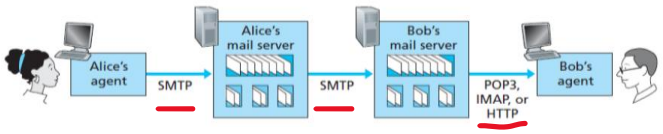


- Method types:
  - HTTP/1.0 (GET, POST, HEAD)
  - HTTP/1.1 (GET, POST, HEAD, PUT, DELETE)
- Uploading form input:

POST method	URL method
web page often includes form input	uses GET method
input is uploaded to server in entity body	input is uploaded in URL field of request line

- HTTP response status codes
  - 200 OK
  - 301 Moved Permanently
  - 400 Bad Request
  - 404 Not Found
  - 505 HTTP Version Not Supported
- cookies can be used for:
  - authorization
  - shopping carts
  - recommendations
- Web caches (proxy server):
  - satisfy client request without involving origin server
  - acts as both client and server
  - is installed by ISP
  - reduce response time for client request
  - reduce traffic on an institution's access link

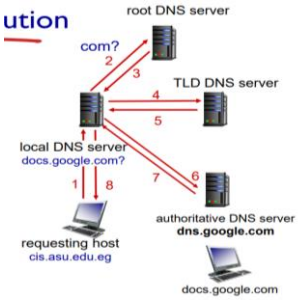
- ftp server: port 21
- TCP control connection on port 21, TCP data connection on port 20
- sample return codes
  - 331 Username OK, password required
  - 125 data connection already open; transfer starting
  - 425 Can't open data connection
  - 452 Error writing file
- Electronic mail major components:
  - user agents
  - mail servers
  - simple mail transfer protocol: SMTP (between mail servers to send email messages)
- mail servers:
  - mailbox -> contains incoming messages for user
  - message queue of outgoing (to be sent) mail messages
- three phases of transfer electronic mail once the TCP connection is established:
  - handshaking (greeting)
  - transfer of messages
  - closure



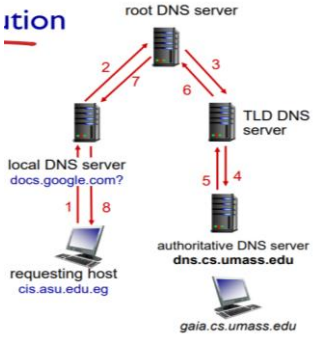
SMPT	HTTP
push	pull
multiple objects sent in multipart msg	each object encapsulated in its own response msg
both have ASCII command/response interaction, status codes	

- POP3:
  - "download and delete" mode (cannot re-read e-mail if he changes client)
  - "download-and-keep" (copies of messages on different clients)
  - is stateless across sessions
- IMAP:
  - Maintain a folder hierarchy on a remote server that can be accessed from any computer. (Create / Organize / Search)
  - IMAP server maintains user state information across IMAP sessions.
  - It has commands that permit a user agent to obtain components of messages.
- Web-Based E-Mail:
  - Users are sending and accessing their e-mail through their Web browsers.
  - User communicates with its remote mailbox via HTTP.
  - However, still sends messages to, and receives messages from, other mail servers using SMTP.

- All DNS query and reply messages are sent within UDP datagrams to port 53
- DNS services:
  - hostname to IP address translation
  - host aliasing (canonical, alias names)
  - mail server aliasing
  - load distribution
- replicated Web servers -> many IP addresses correspond to one name
- Domain Name System (DNS):
  - distributed database implemented in hierarchy of many name servers
  - distributed db storing resource records (RR)
- top-level domain (TLD) servers:
  - Responsible for com, org, e.g.
  - Network Solutions maintains servers for .com TLD
  - Educause for .edu TLD
- Authoritative DNS servers:
  - Organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts
  - can be maintained by organization or service provider
- Local DNS name server -> default name server
- when host makes DNS query, query is sent to its local DNS server
  - has local cache of recent name-to-address translation pairs
  - acts as proxy, forwards query into hierarchy
- DNS name resolution:
  - iterated query:
    - ✓ contacted server replies with name of server to contact



  - recursive query:
    - ✓ puts burden of name resolution on contacted name server

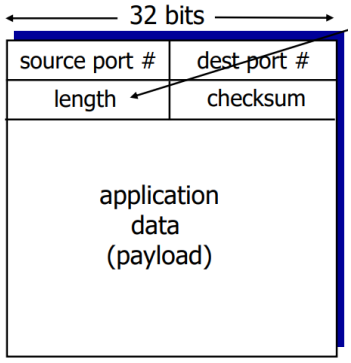


- if name host changes IP address, may not be known Internet-wide until all TTLs expire
- DNS records:
  - Type = A
    - ✓ name is hostname
    - ✓ value is IP address
  - Type = NS
    - ✓ name is domain
    - ✓ value is hostname of authoritative name server
  - Type = CNAME
    - ✓ name is alias name
    - ✓ value is canonical name
  - Type = MX
    - ✓ value is canonical name of mail server that has alias hostname name
- Note that a company can have the same aliased name for its mail server and for one of its other servers (such as its Web server)
- d min = min client download rate
- min client download time: F/d min
- in P2P, All clients must download NF bits (max upload rate (limting max download rate) is us + Sui)
- Torrent -> Collection of all peers participating in the distribution of a particular file
- Peers in a torrent download equal-size chunks of the file from one another, with a typical chunk size of 256 KBytes.
- When a peer first joins a torrent, it has no chunks. Over time it accumulates more and more chunks.
- Churn -> peers may come and go, any peer may leave the torrent at any time with only a subset of chunks, and later rejoin the torrent.
- Once a peer has acquired the entire file, it may leave the torrent, or remain in the torrent and continue to upload chunks to other peers
- When you request, send file chunks, you should choose rarest your neighbors first to request then top 4 uploaders + one probing peer to send requested chunks
- higher upload rate -> find better trading partners, get file faster

Chapter 3

- network layer -> logical communication between hosts
- transport layer -> logical communication between processes
- services not available in transport-layer protocols:
  - delay guarantees
  - bandwidth guarantees

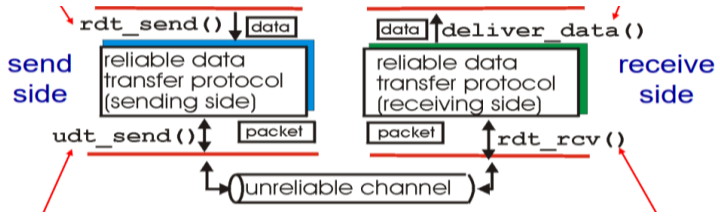
- multiplexing at sender -> handle data from multiple sockets, add transport header
- demultiplexing at receiver -> use header info to deliver received segments to correct socket
- host uses IP addresses & port numbers to direct segment to appropriate socket
- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- demux -> receiver uses all four values to direct segment to appropriate socket
- web servers have different sockets for each connecting client
- non-persistent HTTP will have different socket for each request
- UDP use:
  - streaming multimedia apps (loss tolerant, rate sensitive)
  - DNS
  - SNMP
- reliable transfer over UDP:
  - add reliability at application layer
  - application-specific error recovery



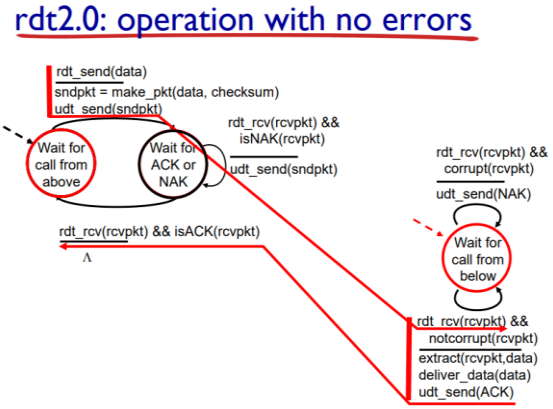
UDP segment format

- why is there a UDP?
  - no connection establishment (which can add delay)
  - simple: no connection state at sender, receiver
  - small header size
  - no congestion control
- rdt_send() -> called from above, (e.g., by app.). Passed data to deliver to receiver upper layer
- udt_send() -> called by rdt, to transfer packet over unreliable channel to receiver
- rdt_rcv() -> called when packet arrives on rcv-side of channel

- deliver_data() -> called by rdt to deliver data to upper

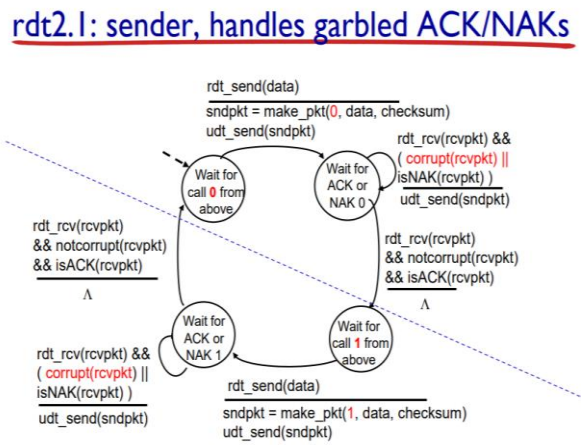


- finite state machines (FSM) -> used to specify sender, receiver
- rdt1.0 (reliable transfer over a reliable channel):
  - underlying channel perfectly reliable
    - ✓ no bit errors
    - ✓ no loss of packets
  - separate FSMs for sender, receiver:
    - ✓ sender sends data into underlying channel
    - ✓ receiver reads data from underlying channel
- rdt2.0 (channel with bit errors):
  - underlying channel may flip bits in packet (checksum to detect bit errors)
  - how to recover from errors?
    - ✓ acknowledgements (ACKs): receiver explicitly tells sender that pkt received OK
    - ✓ negative acknowledgements (NAKs): receiver explicitly tells sender that pkt had errors
    - ✓ sender retransmits pkt on receipt of NAK



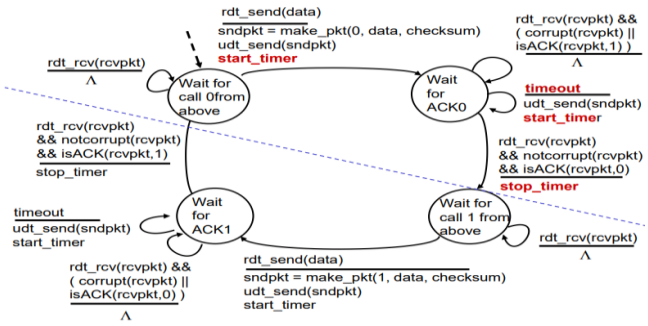
- what happens if ACK/NAK corrupted?
  - sender doesn't know what happened at receiver!
  - possible duplicate
- handling duplicates:
  - sender retransmits current pkt if ACK/NAK corrupted
  - sender adds sequence number to each pkt
  - receiver discards duplicate pkt
- stop and wait sender sends one packet, then waits for receiver response

- rdy2.1 (receiver, handles garbled ACK/NAKs):



  - sender:
    - ✓ seq # added to pkt
    - ✓ two seq. #'s (0,1) will suffice.
    - ✓ must check if received ACK/NAK corrupted
    - ✓ twice as many states (state must "remember" whether "expected" pkt should have seq # of 0 or 1)
  - receiver:
    - ✓ must check if received packet is duplicate (state indicates whether 0 or 1 is expected pkt seq #)
  - note -> receiver can not know if its last ACK/NAK received OK at sender
- rdt2.2 (NAK-free protocol):
  - using ACKs only
  - instead of NAK, receiver sends ACK for last pkt received OK
  - duplicate ACK at sender results in same action as NAK: retransmit current pkt
- rdt3.0(channels with errors and loss):
  - underlying channel can also lose packets (data, ACKs)
  - how to recover from errors?
    - ✓ retransmits if no ACK received in this time
    - ✓ requires countdown timer
    - ✓ if pkt (or ACK) just delayed (not lost):
      - retransmission will be duplicate, but seq. #' s already handles this
      - receiver must specify seq # of pkt being ACKed

rdt3.0 sender



- $U_{sender} = (L / R)/(RTT + L / R)$
- two generic forms of pipelined protocols:

go-Back-N	selective repeat
receiver only sends cumulative ack	receiver sends individual ack for each packet
when timer expires, retransmit all unacked packets	when timer expires, retransmit only that unacked packet
timeout(n) -> retransmit packet n and all higher seq # pkts in window	timeout(n) -> resend pkt n, restart timer

- how to set TCP timeout value?
  - longer than RTT
  - too short: premature timeout, unnecessary retransmissions
  - too long: slow reaction to segment loss
- SampleRTT -> measured time from segment transmission until ACK receipt, ignore retransmissions
- retransmissions triggered by:
  - timeout events
  - duplicate acks

TCP ACK generation [RFC 1122, RFC 2581]

event at receiver	*TCP receiver action*
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 fills gap	**immediate** send **ACK**, provided that segment starts at lower end of gap

- TCP fast retransmit -> if sender receives 3 ACKs for same data (“triple duplicate ACKs”), resend unacked segment with smallest seq #
- Flow control is a speed-matching service—matching the rate at which the sender is sending against the rate at which the receiving application is reading.

- TCP provides a flow-control
  - to eliminate the possibility of the sender overflowing the receiver's buffer (RcvBuffer).
  - having the sender maintain a variable called the receive window (rwnd).
- TCP is full-duplex
- LastByteRead: the number of the last byte in the data stream read from the buffer
- LastByteRcvd: the number of the last byte in the data stream that has arrived from the network and has been placed in the receive buffer
- rwnd = RcvBuffer – [LastByteRcvd – LastByteRead]
- LastByteRcvd – LastByteRead <= RcvBuffer
- LastByteSent – LastByteAcked <= rwnd
- When rwnd =0 and host A is blocked, the TCP specification requires Host A to continue to send segments with one data byte when B's receive window is zero. These segments will be acknowledged by the receiver. Eventually the buffer will begin to empty and the acknowledgments will contain a nonzero rwnd value.
- lost packets -> buffer overflow at routers
- long delays -> queueing in router buffers
- congestion -> too many sources sending too much data too fast for network to handle
- end-end congestion control:
  - no explicit feedback from network
  - congestion inferred from end-system observed loss, delay
  - approach taken by TCP
- network-assisted congestion control:
  - routers provide feedback to end systems (single bit indicating congestion , explicit rate for sender to send at)
- TCP must use end-to-end congestion control
- how does a TCP sender perceive that there is congestion on the path between itself and the destination?
  - Cwnd
- how does a TCP sender limit the rate at which it sends traffic into its connection?
  - sender's send rate is roughly cwnd/RTT bytes/sec
- What algorithm should the sender use to change its send rate as a function of perceived end-to-end congestion?
  - additive increase multiplicative decrease
- additive increase: increase cwnd by 1 MSS every RTT until loss detected
- multiplicative decrease: cut cwnd in half after loss
- cwnd is dynamic, function of perceived network congestion

- seq # is byte-stream number of first data byte in segment
- rate = cwnd / RTT