CS2105

AY22/23 Sem 2

github.com/securespider

01. Introduction

- **Network Edge** - Hosts (Clients and servers)
- **Access Networks** - Wired and wireless communication links
- **Network Core** - Network of interconnected routers

Network Core

Packet-Switching

- Host breaks messages into packets of L bits
- Transmits packets into access network at transmission rate R (aka Link bandwidth, capacity)

$$\text{Packet Transmission Delay} = \frac{\text{Packet Size (bits)}}{\text{Transmission Rate (bits/sec)}}$$

- **Store and Forward** - Entire packet must arrive at router before being transmitted to next link

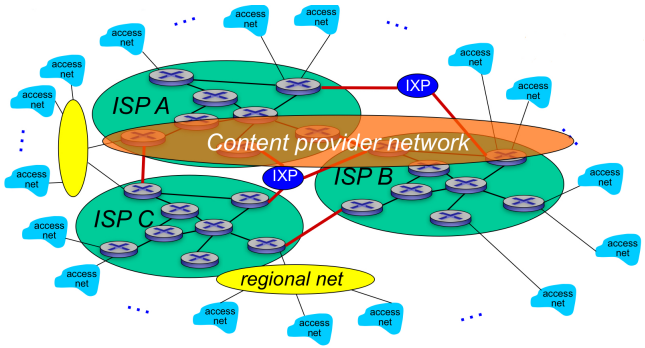
Key Functions of Network Core

- **Routing** - Determines source-destination routes taken by packets (How we get the hashtable)
- **Forwarding** - Move packets from router's input to correct router output

Circuit Switching

- Resources reserved for call between source and destination
- Pros: Better performance
- Cons: More resources

Internet Structure



- End systems connect to Internet via **Access Internet Service Providers (ISPs)**
- ISPs connect to larger global ISPs (usually competitors)
- Large ISPs connect via **peering links** or **internet exchange points (IXP)**
- **IXP** - Physical place with routers from different ISPs
- **Regional Networks** - Smallers ISPs
- **Content Provider Networks** - Provide content close to end users

Loss, Delay, and Throughput

Packet Loss

- If Arrival Rate $>$ Transmission Rate, packets will queue and can be dropped if queue fills up
- Solutions: Lost packets can be retransmitted

Packet Delay

$$d_{\text{nodal}} = d_{\text{proc}} + d_{\text{queue}} + d_{\text{trans}} + d_{\text{prop}}$$

- **Nodal Processing** - (d_{proc}) Check for bit errors and determine output link
- **Queueing Delay** - (d_{queue}) Time at queue waiting for transmission, router congestion
- **Transmission Delay** - (d_{trans}) Time to load packet onto link
  - $d_{\text{trans}} = \frac{L}{R}$ where L is packet length and R is link bandwidth
- **Propagation Delay** - (d_{prop}) Time for 1 bit to reach end of link
  - $d_{\text{prop}} = \frac{d}{s}$ where d is length of link and s is propagation speed

Throughput

- Rate at which bits transferred between hosts
  - Different from transmission rate (Theoretical upper bound)
  - Measures end-to-end, even through intermediaries
  - Irregardless of the size of packets
- Average: Rate over long period of time
- Instantaneous: Rate at given point in time

Protocol Layers and Service Models

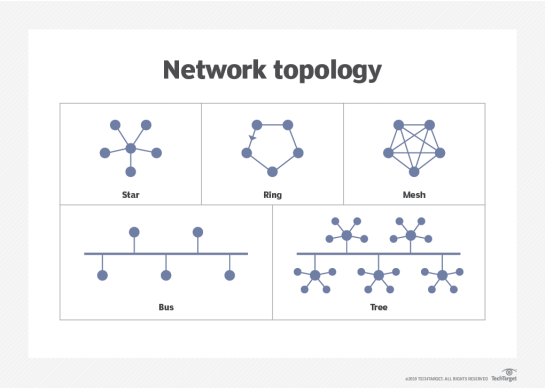
- **Protocol** - Defines format, order of messages sent and received, and actions taken on message transmission
- **Layering** - Each layer implements a service by doing something within layer and relying on services provided by layer below it
  - Explicit structure allows us to make sense of complex components
  - Easy maintenance (Like OOP, change in 1 layer should not affect others)

Internet Protocol Stack

1. **Application** - eg. FTP, SMTP
2. **Transport** - TCP, UDP
3. **Network** - IP, Routing protocols
4. **Link** - Ethernet, WiFi, PPP
5. **Physical** - Wires, bits

Encapsulation - Take information from a higher layer and adds a header to it, treating the higher layer information as data

Topology



- **Star** - Lowest cost and simplest
- **Mesh** - High redundancy, increased cost but highest transmission speeds

02. Application Layer

- Programs that run on end systems, and not on network-core devices

Client-server Architecture

- Server: Always-on host, Permanent IP address
- Clients: Communicates with server, Intermittently connected, Dynamic IP addresses, Do not communicate with each other directly

P2P Architecture

- Peers request service from other peers and provide service in return
- No always-on server, Intermittently connected, Dynamic IP addresses
- **Self Scalability** - New peers offer new services and demands

Process

- **Process** - Program running in host
- **Inter-process Communication** - How 2 processes in 1 host communicate
- **Client Process** - Process that initiates communication
- **Server Process** - Process that waits to be contacted
- **Socket** - Process sends/receives messages to/from its socket (like a door)
  - Outside of socket, transport layer delivers message

Addressing Processes

- Motivation: IP address is not enough to address process, since many processes can be running on same host
  - **Identifier** - IP address and port number
  - **IP** - 32-bit address for identifying host
  - **Port Number** - 16-bit to identify specific process on host

Services

- **Data Integrity** - Reliable data transfer
- **Timing** - Low delay/latency
- **Throughput** - Minimum amount of throughput for effectiveness
- **Security** - Encryption, data integrity

Transport Protocol Services

1. **TCP** - Transmission Control Protocol
   - Reliable transport
   - Flow control: Sender does not overwhelm receiver
   - Congestion control
   - Connection-oriented: Setup required between client and server
2. **UDP** - User Datagram Protocol
   - Unreliable data transfer
   - Fast

App-layer Protocol

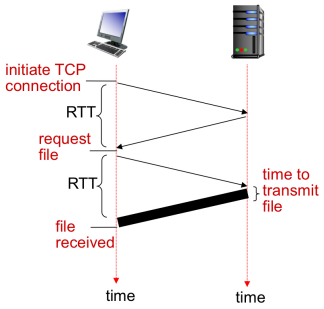
- Types of messages exchanged (e.g. Request or response)
- Message syntax: How fields are delineated
- Messages semantics: Meaning of information in fields

HTTP - 80

- **Hypertext Transfer Protocol** - Web's application layer protocol
- Motivation: Web page consists of objects (HTML, images). Need method to request/send web objects.
- Follows client/server model
- Uses TCP
- **Stateless** - Server maintains no information about past requests

Non-persistent HTTP

- At most 1 object sent over TCP connection
- Downloading multiple objects requires multiple TCP connections

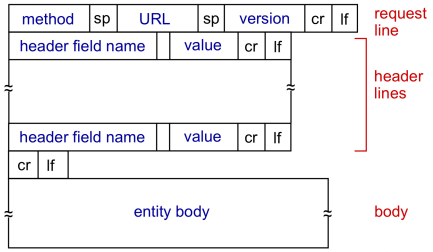


- Server closes TCP connection after sending file
- **Return Trip Time** - (RTT) Time for small packet to travel from client to server and back
- Response Time: 2 RTT + File transmission time

Persistent HTTP

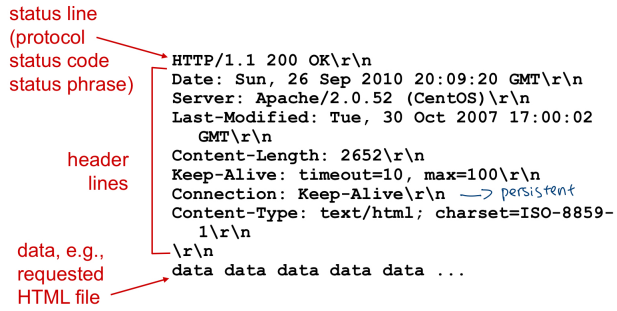
- Multiple objects can be sent over single TCP connection
- Server leaves TCP connection open after sending response
- As little as one RTT for all referenced objects

HTTP Request Message



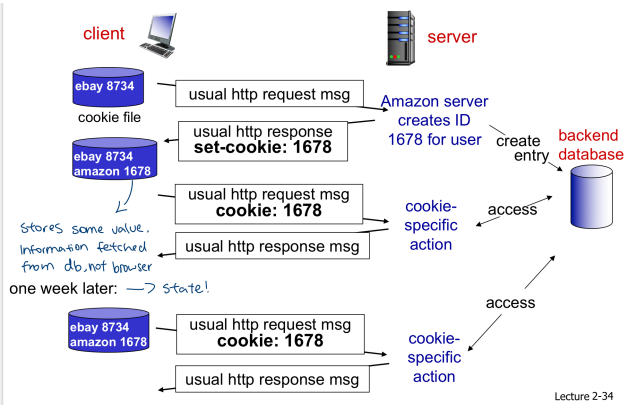
- To upload form input: **POST method** - Input uploaded via entity body and **URL method** - Input uploaded in URL field of GET method
- **HTTP/1.0** - GET, POST, HEAD (Ask server to leave request object out of response)
- **HTTP/1.1** - GET, POST, HEAD, PUT, DELETE

HTTP Response Message



Cookies

- Maintains state on client side
- Components: Cookie header for HTTP response, Cookie header for HTTP, request, Cookie file on user's host (Key-value pair), Database on server



Lecture 2-34

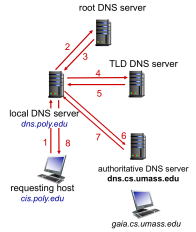
Web Cache (Proxy Server)

- Goal: Fulfill request without involving origin server via caching
- Browser sends all HTTP requests to cache
- Pros: Faster, Reduces traffic to origin server
- Cons: What if origin server updates?
  - **Conditional GET** - Origin server doesn't send object if cache has updated version
  - Cache: Specifies date of cached copy in HTTP request to origin
  - Origin Server: Response contains no object if cached object is updated

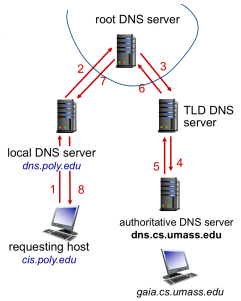
Domain Name System

- Maps between IP address and name (e.g. yahoo.com)
- Implemented using distributed and hierarchical databases
- Application-layer protocol (Why?)

  Build complexity in edges to simplify inner core networks
- Uses UDP
- **Local DNS Name Server** - Local cache of name-to-address mapping. Forwards query into hierarchy.
  - **Time to Live** - (TTL) Cached mappings disappear after some time
- **Root Name Server** - Contacted by local name server that cannot resolve name. Provides IP address of TLD servers.
- **Top-level Domain Server** - (TLD) Provides IP address of authoritative server
- **Authoritative DNS Server** - Organization's own DNS server. Provides mappings for organization's named hosts.
- Iterated query: "Not sure, ask this server"



- Recursive query: "Okay, let me find for you"
  - Heavy load on upper levels of hierarchy



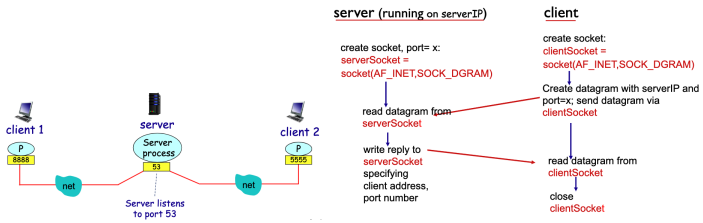
DNS Caching

Cache entry - Cache record of recent name mapping

- Timeout after some time based on TTL
- Best effort name-to-address translation (can be out-of-date)

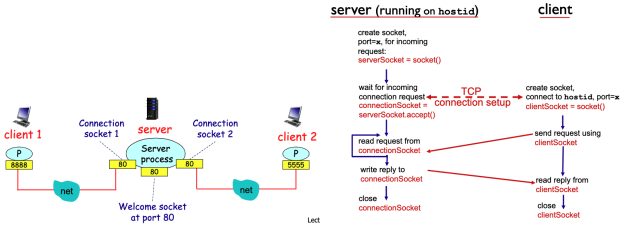
03. Socket Programming with UDP and TCP

UDP Socket



- No connection beforehand. Just send it.
- Server has 1 socket to serve all clients
- Sender attaches destination IP address and port number (**Stateless**)
- Unreliable datagram: Data may be lost or out-of-order
  - **Datagram** - Group of bytes

TCP Socket



- Client establishes connection to server via welcome socket
- Server makes new socket for each client
- Server identifies client via connection (**Stateful**)
- Reliable stream pipe: Data always in order

04. UDP and Reliable Protocol

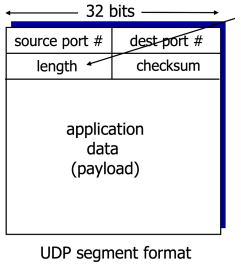
Transport Layer Services

- Provide logical communication between processes in different hosts
- Sender: Breaks app messages into segment
- Receiver: Reassembles segments

UDP

- On top of network layer, UDP adds:
  1. Connectionless multiplexing/de-multiplexing
     - UDP segments contain both source and destination ports
     - Multiplexing: Sent to target processes
  2. Checksum

UDP Segment Header



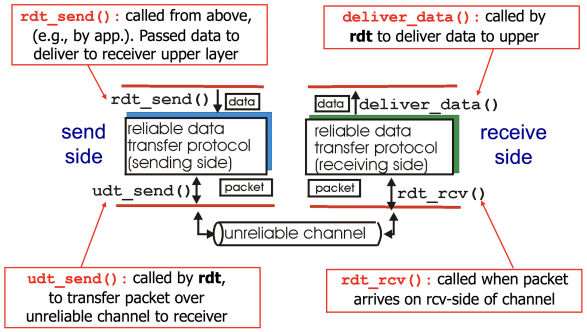
UDP segment format

Checksum

- Goal: Detect errors in received segment
  1. Treat UDP segment as sequence of 16-bit integers
  2. Add every 16-bit integer (Carry added back to result)
  3. Invert to get UDP checksum (1's complement)
  4. When receiving, sum segment again. All 1s if correct.

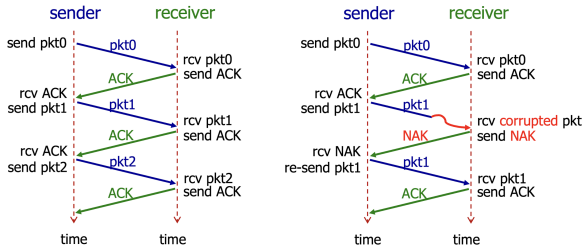
Reliable Data Transfer (rdt)

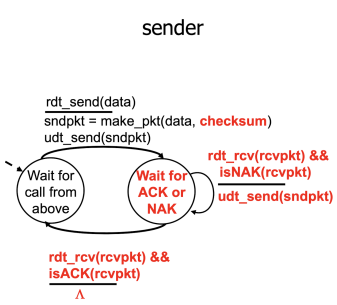
- Characteristics of unreliable channel will determine services provided by rdt

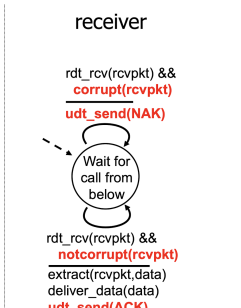


rdt 2.0

- New problem: **Bit error** - May flip bits in packet
- Solution:
  - Perform checksum to detect bit errors
  - **Stop and wait protocol** - Sender sends one packet at a time and wait for response
  - **ACK** - Receiver tells sender that packet received is ok
  - **NAK** - Receiver tells sender that packet received has errors (retransmission)

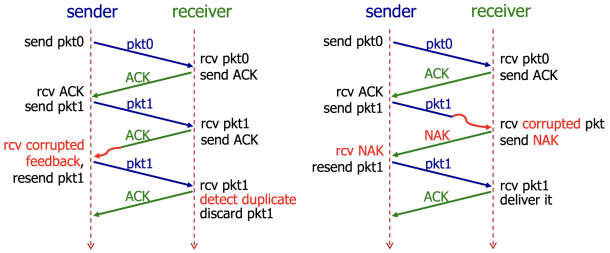


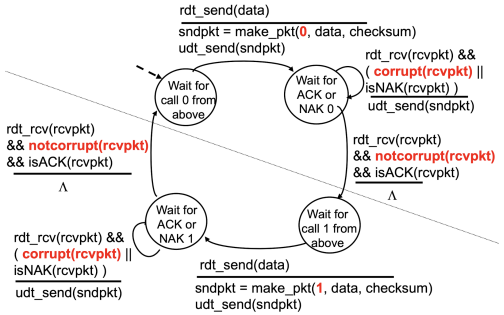


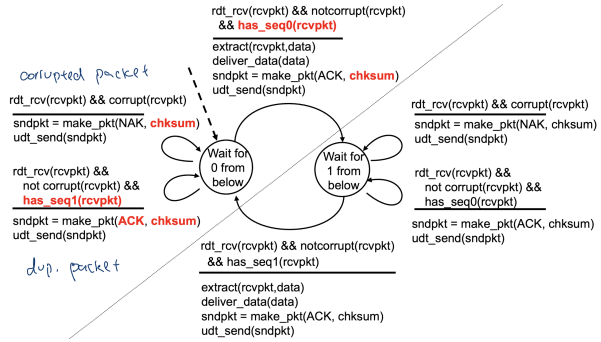


rdt 2.1

- New problem: ACK/NAK may be corrupted
- Solution: Sender retransmits packet after receiving corrupted ACK/NAK
- New problem: Duplicate packets during retransmission
- Solution: Sender adds **sequence number** to each packet and receiver discards duplicate packet (Only 0 and 1 needed)





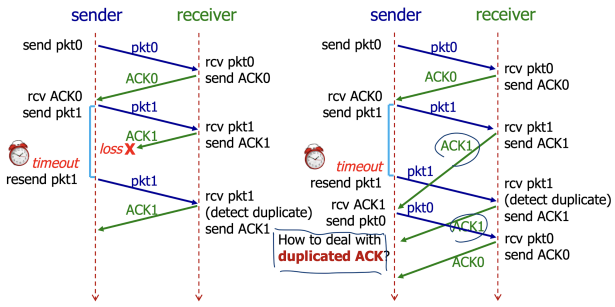


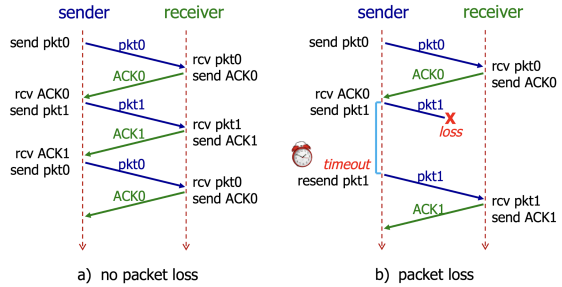
rdt 2.2

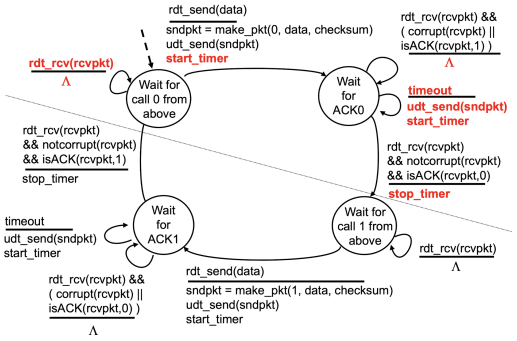
- No NAK, only ACKs
- Receiver sends ACK for last packet received. **ACK must include seq number of packet**.
- Receiver send duplicate ACK to retransmit current packet

rdt 3.0

- New problem: Lost packets
- Solution: Sender waits for some time for ACK and retransmits packet
  - What if duplicate packet? Sequence number handles this.
  - What if duplicate ACK? Do nothing.







Performance of rdt 3.0

- **Stop-and-wait protocol** - Sender sends 1 packet at a time, then waits for receiver response
- Performance is bad. Stop-and-wait protocol limits use of resources
- **Utilization** - Fraction of time sender is busy sending
  - Given: 1 Gbps link, 15 ms prop. delay, 8000 bit packet
  - $D_{trans} = \frac{L}{R} = \frac{8000bits}{10^9 bits/sec} = 8microsecs$
  - $U_{sender} = \frac{D_{trans}}{RTT+D_{trans}} = \frac{0.008}{30.008} = 0.027\%$

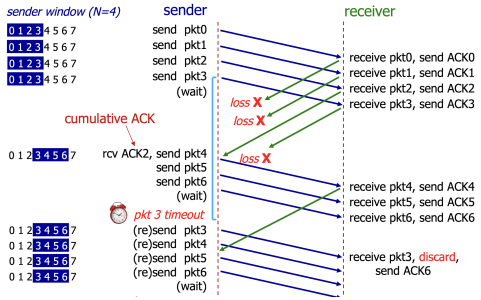
Pipelined Protocols

- **Pipelining** - Sender allows multiple not-yet-ACKed packets
  - Need more sequence numbers
  - Buffering at sender and receiver

Pipeline vs Parallel (HTTP)

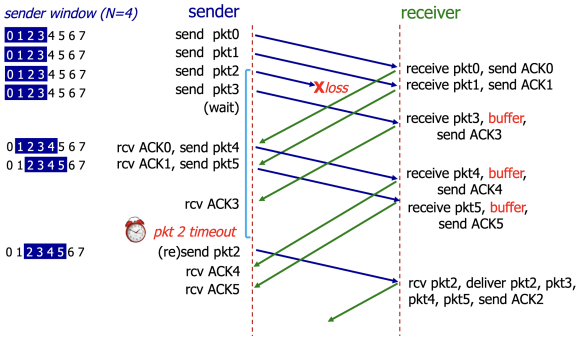
- Parallel connections need to retrieve HTML webpage first before retrieving data elements
- Pipeline can retrieve HTML and data elements all at once

Go-Back-N



- Intuition: Sliding window
- Sender:
  - Timer for oldest in-flight packet
  - If timeout(n), retransmit packet n and other pkts in window
- Receiver:
  - **Cumulative ACK** - ACK for correct pkt with highest in-order sequence
  - Discard out-of-order packets
- Not efficient, since future packets discarded if out-of-order

Selective Repeat



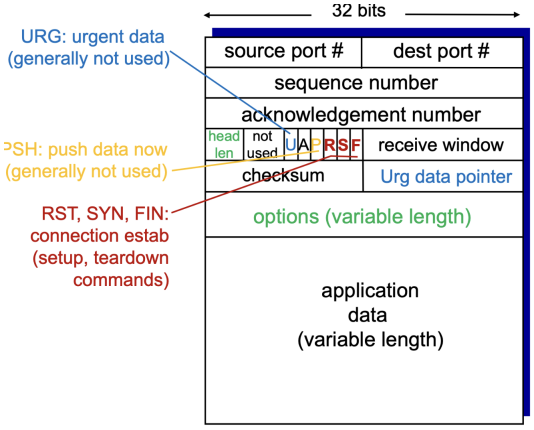
- Receiver individually ACKs correct pkts (Not accumulative) and sender maintains timer for each unACKed pkt
- Sender:
  - If timeout(n), retransmit packet n only
  - If ACK(n) and n is smallest unACKed pkt, slide window
- Receiver:
  - Once receive pkt n in window, send ACK(n). If out-of-order, buffer. If in-order, deliver and slide window
  - Once receive pkt n outside of window, still send ACK(n)

05. TCP

TCP Features

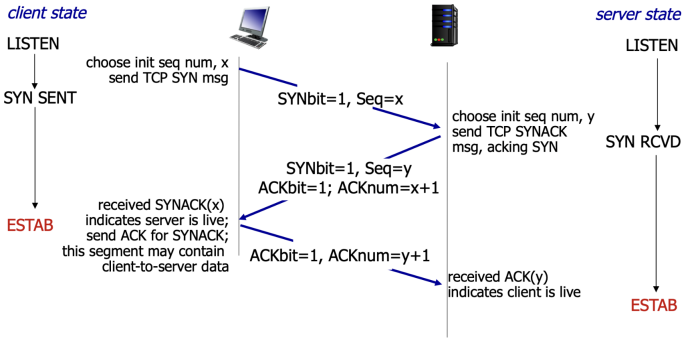
- **Point-to-point** - 1 sender and 1 receiver
- **Connection-oriented** - Handshaking before exchange data
- **Full duplex** - Bi-directional data flow in connection
- Reliable, in order and pipelined
- Buffers on both hosts
- Limited by **Maximum Segment Size**

Structure

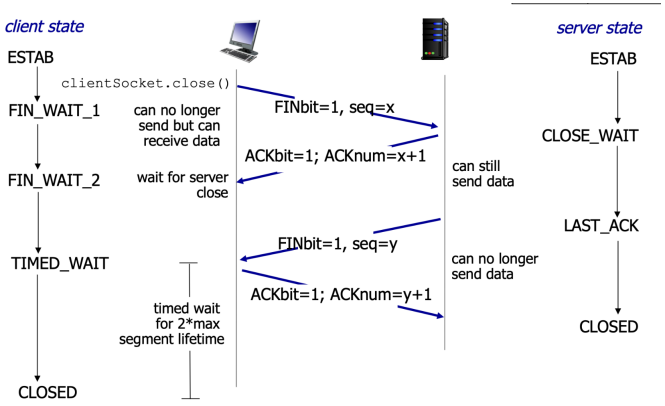


1. Sequence #
   - Let receiver know if packet is retransmission or new data
   - Initial number randomly generated (Why?)
     - Security
     - Prevent duplicate sequence numbers when link is used extensively
   - ACK # = inital seq # + data length in bytes
2. ACK number
3. **rwnd** - Receive window
   - Number of bytes the receiver is willing to accept
   - **Flow control** - Prevent overflow of receiver's buffer
4. Flags - Connection establish or priority

Establish TCP connection



Close TCP connection

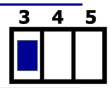
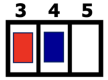
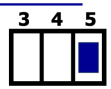
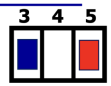


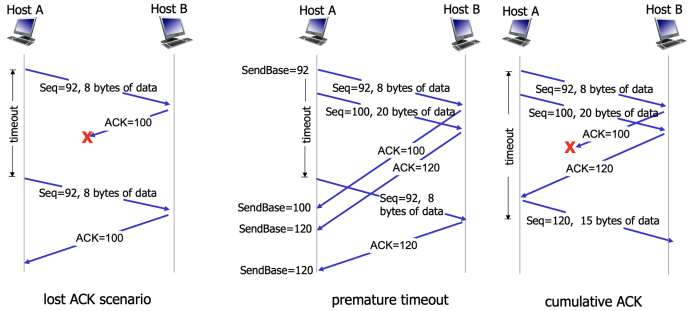
RDT

Features

- Pipelined segments
- Cumulative ACKs (like GBN)
- Single retransmission timer
- Retransmit only missing packets (SR)

Retransmission Scenarios

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 or completely fills gap	immediate send ACK, provided that segment starts at lower end of gap	



RTT

- Problem: Estimate timeout value for retransmission
- Solution: Average recent **SampleRTT** - Measured time from segment transmission until ACK
- EstimatedRTT $= (1-\alpha)*$ EstimatedRTT $+ \alpha*$ SampleRTT
  - Exponential weighted moving average

Fast retransmission

- Send ACKs every other packet OR after timeout of packet
- Retransmit unacked segment with **smallest seq #** after receiving 4 ACKs for same data

06: IP Addressing

Function

1. **Forwarding** - Move packets from router input to output
2. Routing
   - Determine route taken by packets from source to destination
   - Plan trip from source to destination

Planes

1. Data plane
   - Local per router function
   - Determines how datagram forwarded to output port
2. Control plane
   - Network wide logic
   - Routing algorithms

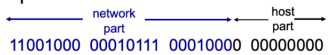
Subnet

- Network formed by directly interconnected host
- Internal host can communicate without router
- Connect to external networks with router
- Same network prefix

CIDR

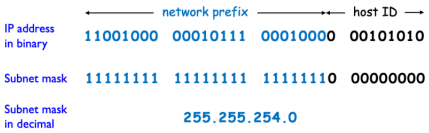
- An IP address logically comprises two parts:

- CIDR: Classless InterDomain Routing
  - arbitrary length for the subnet portion (network prefix)
  - address format: a.b.c.d/x, where x is # bits in subnet portion of address
- Example 200.23.16.0/23



Subnet mask

- Subnet mask is made by setting all network prefix bits to "1"s and host ID bits to "0"s.
  - example: for IP address 200.23.16.42/23:



  - used to determine which network an IP address belongs to (use bitwise AND operation)

Hierachial addressing

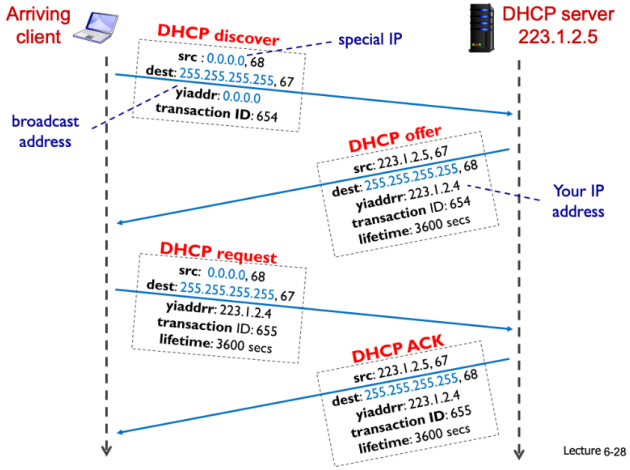
Route aggregation - Advertise routing information by grouping similar IP addresses
Longest Prefix matching

Routes to subnet with the greatest subnet portion of CIDR

DHCP

Dynamic Host Configuration Protocol - Lease IP address from network server when joining network

- Runs over UDP , Client #:68
- Router can act as DHCP
- Server #: 67, sends "DHCP ack" and "DHCP offer"
- Client #: 68, sends "DHCP request" and "DHCP discover"



What happens when multiple servers in network?

- Transaction ID specifies the DHCP connection