

## 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

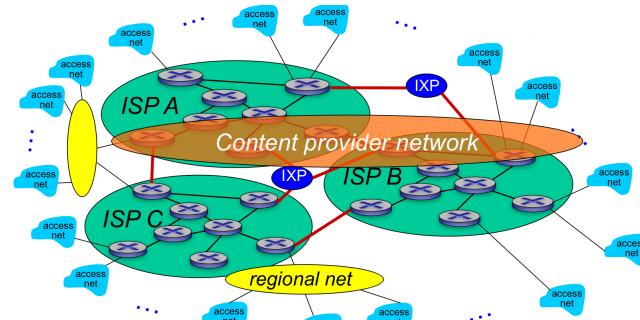
#### Key Functions of Network Core

- **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** - Smaller 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** - ( $d_{\text{proc}}$ ) Check for bit errors and determine output link
- **Queueing Delay** - ( $d_{\text{queue}}$ ) Time at queue waiting for transmission
- **Transmission Delay** - ( $d_{\text{trans}}$ ) Time to load packet onto link  
$$d_{\text{trans}} = \frac{L}{R}$$
 where  $L$  is packet length and  $R$  is link bandwidth
- **Propagation Delay** - ( $d_{\text{prop}}$ ) 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, which is the theoretical upper bound
- 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
- Networks are complex with many components. How can we organize its structure?
- **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
2. Transport
3. Network
4. Link
5. Physical

## 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
- **Messages** - Processes in different hosts communicate by exchanging this
- **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
  - **Port Number**: - Associated with process on host

### 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

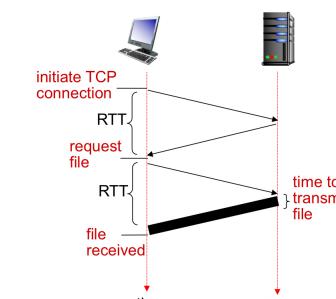
- Types of messages exchanged (e.g. Request or response)
- Message syntax: How fields are delineated
- Messages semantics: Meaning of information in fields

### HTTP

- **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

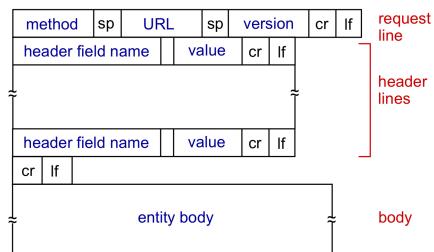
- **Return Trip Time** - (RTT) Time for small packet to travel from client to server and back

- Response Time: 2 RTT + File transmission time

## Persistent HTTP

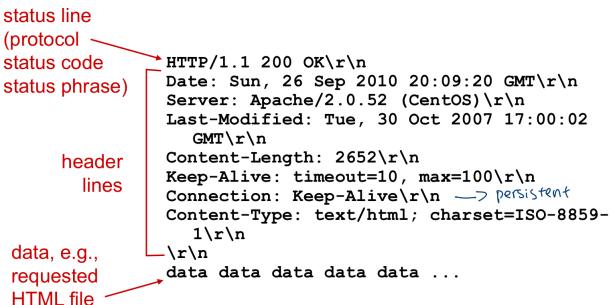
- Multiple objects can be sent over single TCP connection
- Server leaves TCP connection open after sending response
- Subsequent objects can use same TCP connection and be sent using 1 RTT + File transmission time

## HTTP Request Message



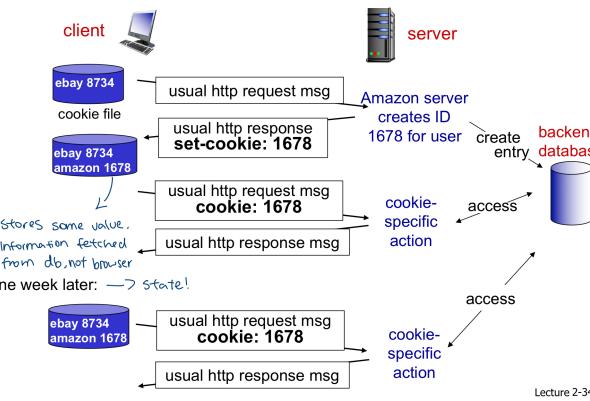
- To upload form input: **POST method** - Input uploaded via entity body
- **URL method** - Input uploaded in URL field of GET method
- **HTTP/1.0** - GET, POST, HEAD (Ask server to leave request object out of response)
- **HTTP/1.1** - GET, POST, HEAD, PUT, DELETE

## 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

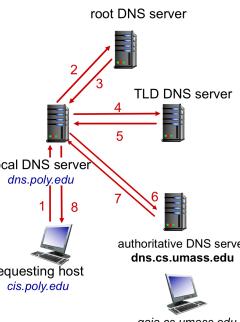


## Web Cache (Proxy Server)

- 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

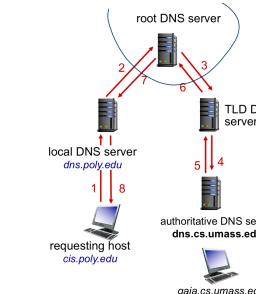
## Domain Name System

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



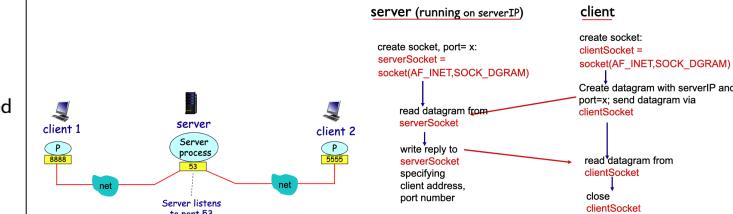
- Recursive query: "Okay, let me find for you"

- Heavy load on upper levels of hierarchy



## 03. Socket Programming with UDP and TCP

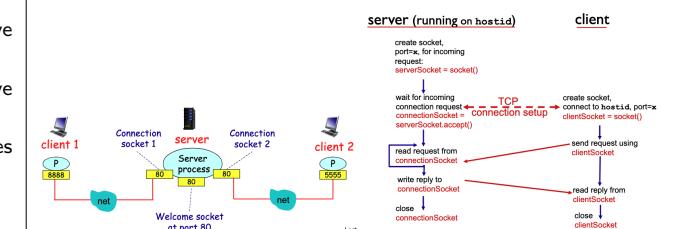
### UDP Socket



- 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

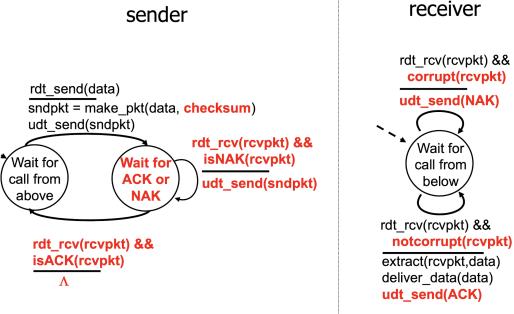
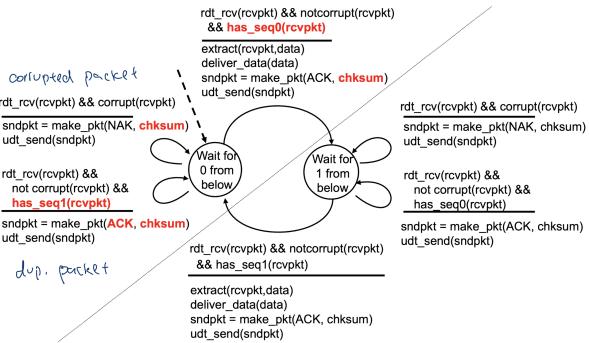
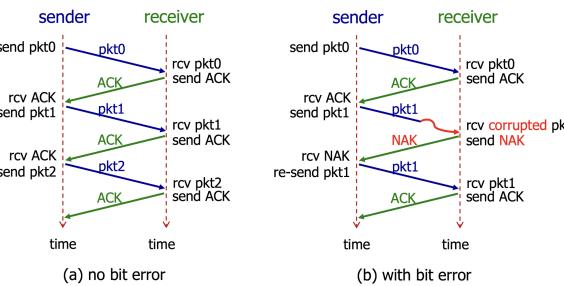
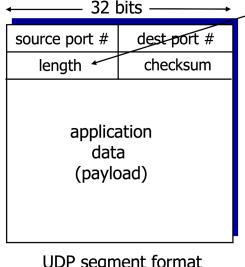
- Transport layer: Logical communication between **processes**
- Network layer: Logical communication between **hosts** (Unreliable)

## UDP

- On top of network layer, UDP adds:

- Connectionless multiplexing/de-multiplexing
  - UDP segments contain both source and destination ports
  - Multiplexing: Sent to target processes
- Checksum

## UDP Segment Header



## rdt 2.2

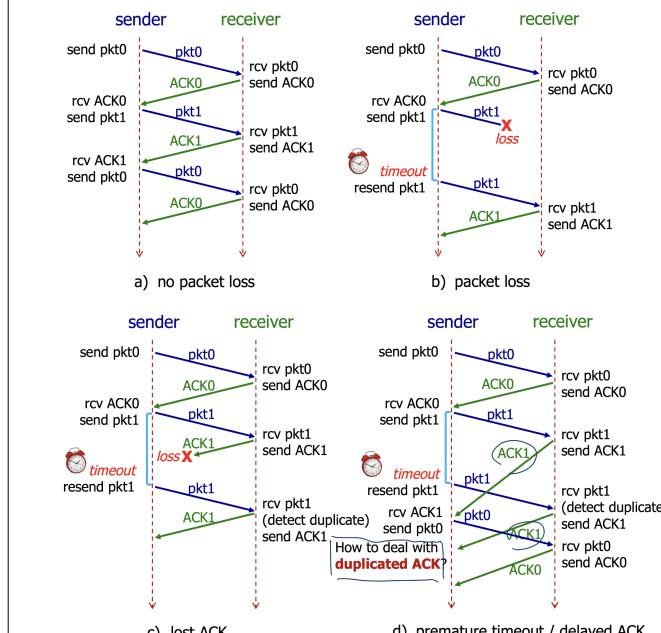
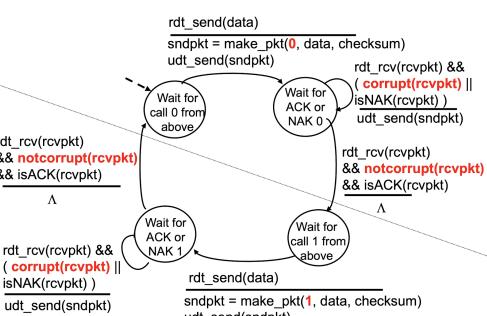
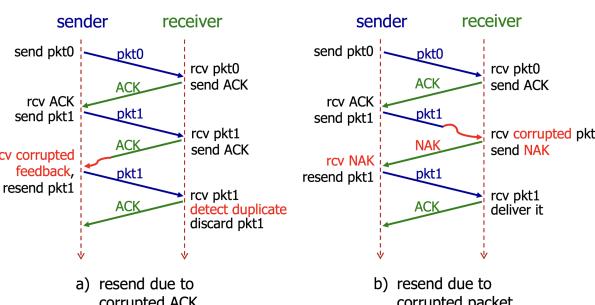
- New problem: Use ACKs only. No NAK.
- Solution: Receiver sends ACK for last packet received. **ACK must include seq number of packet**.

## 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. Only retransmit after timeout.
  - What if packet out of order? Have more sequence numbers.

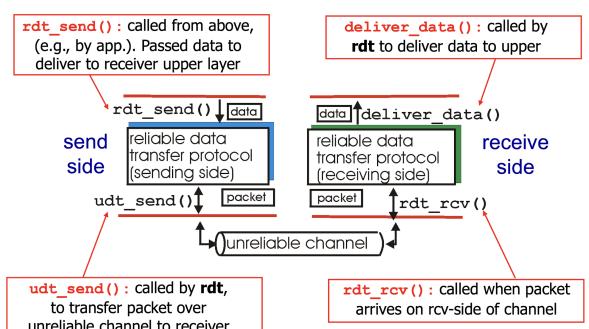
## rdt 2.1

- New problem: ACK/NAK may be corrupted
- Solution: Sender does checksum and retransmits packet if ACK/NAK is corrupted
- New problem: If ACK is corrupted, sender will send a **duplicate packet**
- Solution: Sender adds **sequence number** to each packet and receiver discards duplicate packet (Only 0 and 1 needed)



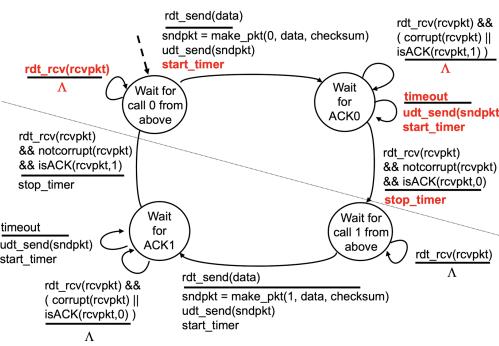
## Reliable Data Transfer (rdt)

- Characteristics of unreliable channel will determine services provided by rdt



## rdt 2.0

- New problem: Bit error in data
- Solution:
  - Recipient performs checksum to detect bit errors
  - ACK** - Receiver tells sender that packet received is ok
  - NAK** - Receiver tells sender that packet received has errors
  - After receiving NAK, sender retransmits packet



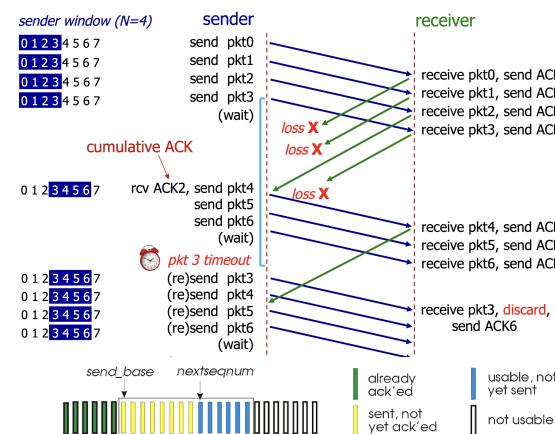
## Performance of rdt 3.0

- 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
  - $D_{trans} = \frac{L}{R} = \frac{8000 \text{ bits}}{10^9 \text{ bits/sec}} = 0.008 \text{ ms}$
  - $U_{sender} = \frac{D_{trans}}{RTT + D_{trans}} = \frac{0.008}{30.008} = 0.027\%$

## Pipelined Protocols

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

## Go-Back-N



## Intuition: Sliding window

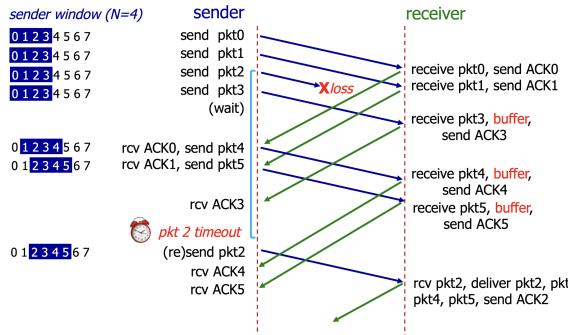
- Sender:**
  - Sends when sliding window reaches packet
  - 1 timer to receive ACK for packet at  $send\_base$
  - If timeout( $n$ ), retransmit packet  $n$  and other pkts in window
  - If receive ACK  $n$ , check if can shift sliding window. Previous un-ACKed packets guaranteed to be received.

- If duplicate ACK, ignore like in rdt 3.0. Retransmit only on timeout.

### Receiver:

- Cumulative ACK** - ACK for correct pkt with highest in-order sequence
- Discard out-of-order packets
- Not efficient, since future packets discarded if pkt lost and 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( $n$ )

## Summary

rdt	New problems	New features
1.0	n/a	n/a
2.0	Bit error in data	Sender: Checksum, Receiver: ACK/NAK
2.1	Bit error in feedback, Duplicate packets	Receiver: Checksum, Sender: Sequence number
2.2	Remove NAK	Receiver: Sequence number
3.0	Packet loss in data and feedback	Timeout, Re-transmission

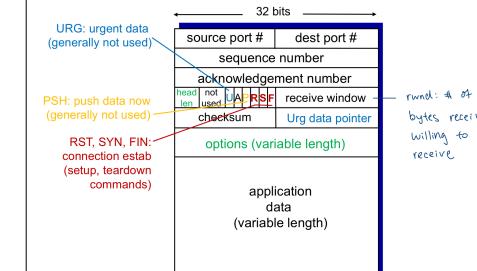
## 05. TCP

- Point-to-point: 1 sender and 1 receiver
- Connection-oriented
- Full duplex data: Data and feedback can be sent together both ways
  - Before in UDP: Data only goes 1 way
- Reliable, in-order **byte stream** (segment)
- Before in UDP: Send packet by packet
- Pipelined: **Dynamic window size** set by flow control

## Buffers and Segments

- Sender and receiver both have 2 buffers to send and receive data
- Maximum Segment Size** - (MSS) Usually 1460 bytes. Limited by max. transmission unit (MTU)

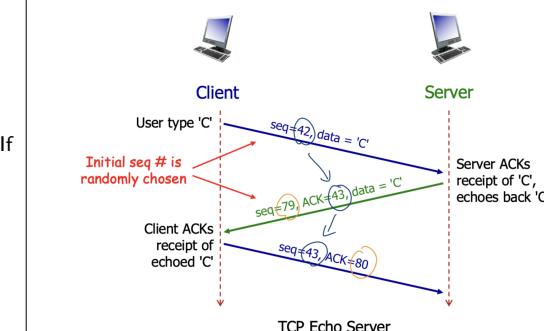
## Segment Structure



- Source/Dest. Port Number: Same as UDP, except ports can be demultiplexed to different sockets

- Sequence Number** - Byte number of first byte of data in segment

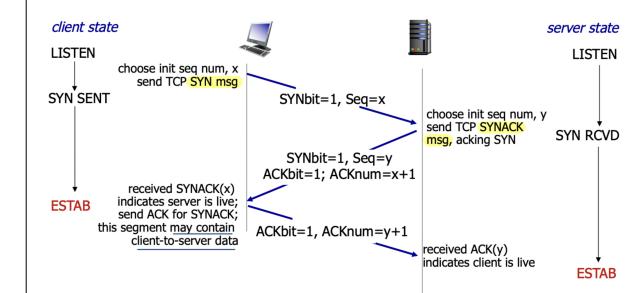
- Acknowledgement Number** - Sequence number of next byte expected from other side (Diff. from UDP)



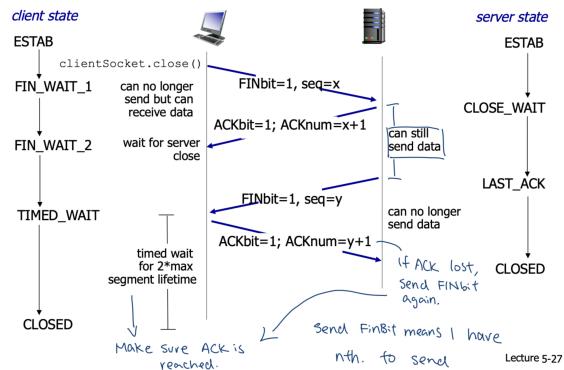
- Receive Window** - For flow control. Receiver controls sender, so sender won't overflow receiver's buffer. Constrains sliding window.

## Connection Management

### 3 Way Handshake



## Closing a Connection

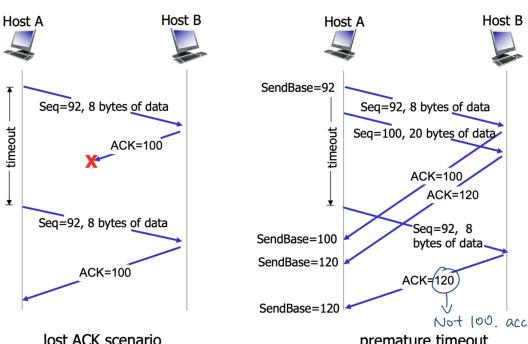


## Reliable Data Transfer

- On top of UDT service, TCP adds:
  - Pipelined segments
  - Cumulative ACKs
  - Single retransmission timer
- Out-of-order packets not specified in TCP. Up to implementer.
- Retransmits on timeout or 4 duplicate ACKs

## Sender Events

- If data received from application:
  - Create segment with sequence number
  - Start timer for **oldest un-ACKed segment**
- If timeout:
  - Retransmit segment causing timeout (Similar to selective repeat)
  - Restart timer
- If ACK received:
  - If ACK acknowledges previously un-ACKed segments, update window and start timer.



## Receiver Events

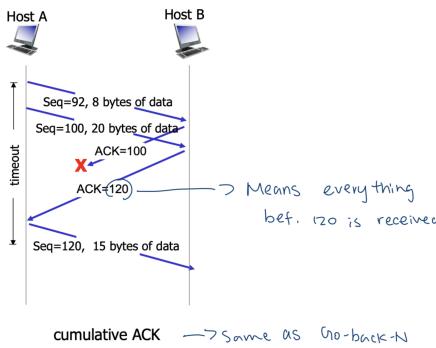
event at receiver	TCP receiver action
(3) arrival of in-order segment with expected seq #. Normal	delayed ACK: Wait up to 500ms for next segment. If no next segment, send ACK(4). Why? optimisation Save 1 ACK
(4) arrival of in-order segment with expected seq #. One other segment has ACK pending (red)	immediately send single cumulative ACK, ACKing (5) both in-order segments
(5) arrival of out-of-order segment higher-than-expect seq #. Gap detected	immediately send <b>duplicate ACK</b> , indicating seq. # of next expected byte (3)
(3) arrival of segment that partially or completely fills gap existing out of order pkt(s)	immediate send ACK, (4) provided that segment starts at lower end of gap Why immediate? might have pkt loss, Lecture 5-35 so trigger fast retransmit

- Delayed ACK for in-order segment
  - If no next segment received in time, send ACK
  - If next segment received in time, send ACK for 2nd segment (Saves 1 ACK. Works due to cumulative ACK.)
- Immediate ACK for out-of-order segment (No matter if it creates or fills gap)
  - Why immediate ACK? Might have packet loss, so can trigger fast retransmit earlier.

## Timeout

- Motivation: What should timeout value be? Too short results in premature timeout. Too long results to slow reaction.
- Sample RTT** - Measured time from segment transmission until ACK receipt. If retransmission, forget it.
  - Sample RTT varies a lot, so not accurate. How to get better estimate?
- Estimated RTT** - Average of recent measurements
  - Uses previous Estimated RTT
  - Usually,  $\alpha = 1/8$

$$\text{Estimated RTT} = (1 - \alpha) * \text{Estimated RTT} + \alpha * \text{Sample RTT}$$



- Timeout Interval** - Estimated RTT + "Safety margin" (Deviation of estimate)

- Usually,  $\beta = 1/4$

$$\text{Dev RTT} = (1 - \beta) * \text{Dev RTT} + \beta * |\text{Sample RTT} - \text{Estimated RTT}|$$

$$\text{Timeout Interval} = \text{Estimated RTT} + 4 * \text{Dev RTT}$$

## Fast Retransmit

- Motivation: Timeout period often quite long. Long delay before resending lost pkt.
- Fast Retransmit** - If sender receives 4 ACKs for same data, resend un-ACKed segment with smallest sequence number

