Transport services and protocols

- provide *communication* between app processes running on different hosts
- transport protocols run in end systems
  - Sender side: breaks app messages into *segments*, passes to Internet layer
  - Receiver side: reassembles segments into messages, passes to Application layer
- More than one transport protocol available to apps
  - Internet: TCP and UDP

Transport vs. Internet layer

- *Internet layer:* logical communication between hosts
- *Transport layer:* logical communication between processes

household analogy:

12 kids in Ann's house sending letters to 12 kids in Bill's house:

- hosts = houses
- processes = kids
- app messages = letters in envelopes
- transport protocol = Ann and Bill who distribute to in-house siblings
- internet-layer protocol = postal service

Transport-layer protocols

- reliable, in-order delivery (TCP)
  - congestion control
  - flow control
  - connection setup
- unreliable, unordered delivery: UDP
  - no-frills extension of “best-effort” IP
- services not available:
  - delay guarantees
  - bandwidth guarantees

UDP: User Datagram Protocol

- "no frills," "bare bones" transport protocol
- "best effort" service, UDP segments may be:
  - lost
  - delivered out-of-order to app
- *connectionless:*
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

- UDP use:
  - streaming multimedia apps (loss tolerant, rate sensitive)
- reliable transfer over UDP:
  - add reliability at application layer
  - application-specific error recovery!

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: UDP can blast away as fast as desired

TCP: Overview

- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte steam:*
  - no "message boundaries"
- pipelined:
  - TCP congestion and flow control set window size
- full duplex data:
  - bi-directional data flow in same connection
- connection-oriented:
  - handshaking (exchange of control messages) in its sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver

TCP reliable data transfer

- TCP creates reliable data transfer service
  - pipelined segments
  - cumulative acknowledgements
  - single retransmission timer

TCP sender events:

data received from app:

- create segment with seq #
- start timer if not already running
  - think of timer as for oldest unacknowledged segment
  - expiration interval: **TimeOutInterval**

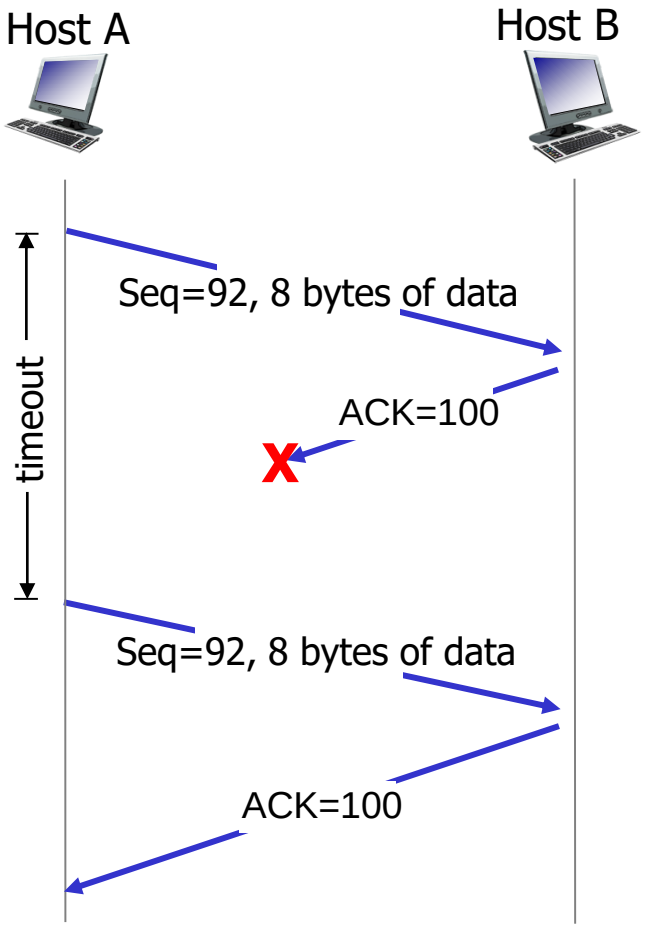
timeout:

- retransmit segment that caused timeout
- restart timer

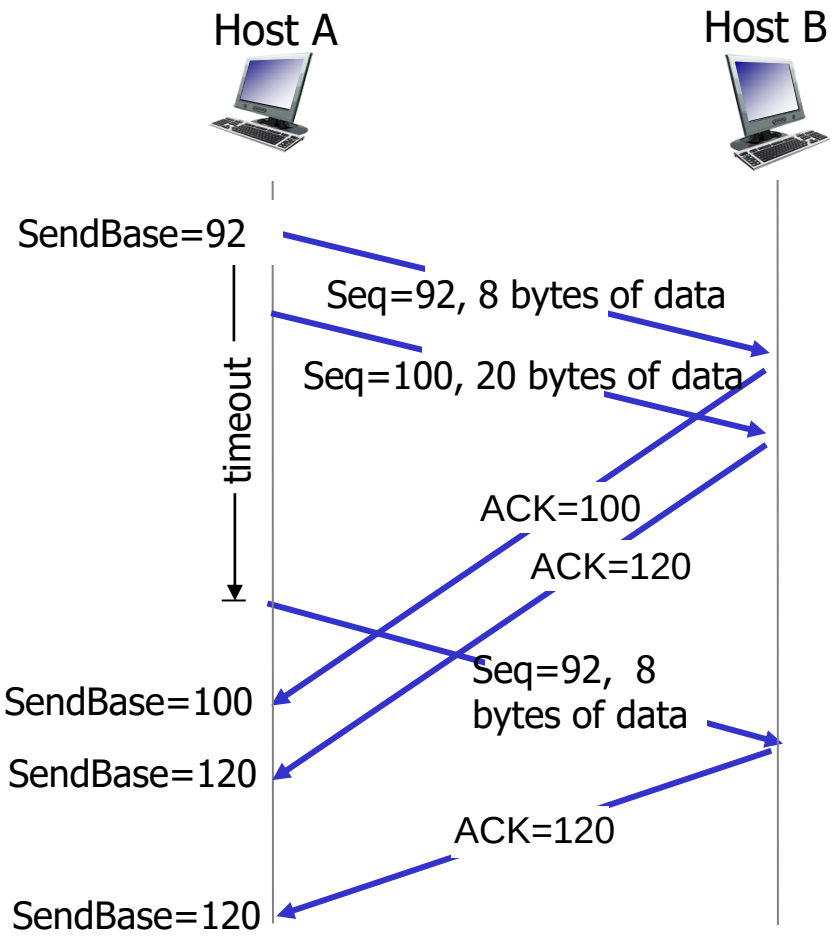
ack received:

- if ack acknowledges previously unacknowledged segments
  - update what is known to be ACKed
  - start timer if there are still unacknowledged segments

TCP: retransmission scenarios

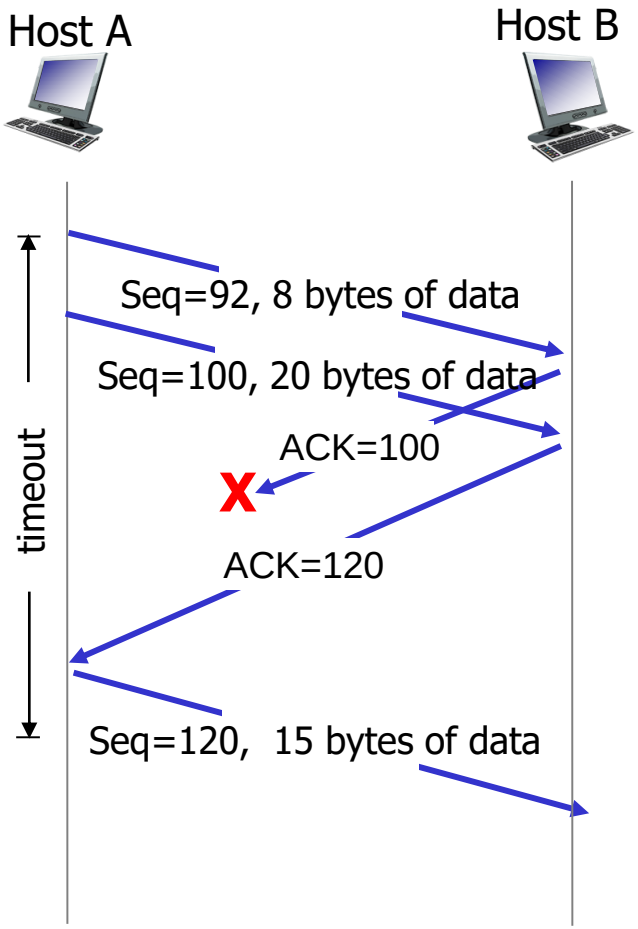


lost ACK scenario



premature timeout

TCP: retransmission scenarios



cumulative ACK

Connection Management

before exchanging data, sender/receiver "handshake":

- agree to establish connection (each knowing the other willing to establish connection)
- agree on connection parameters

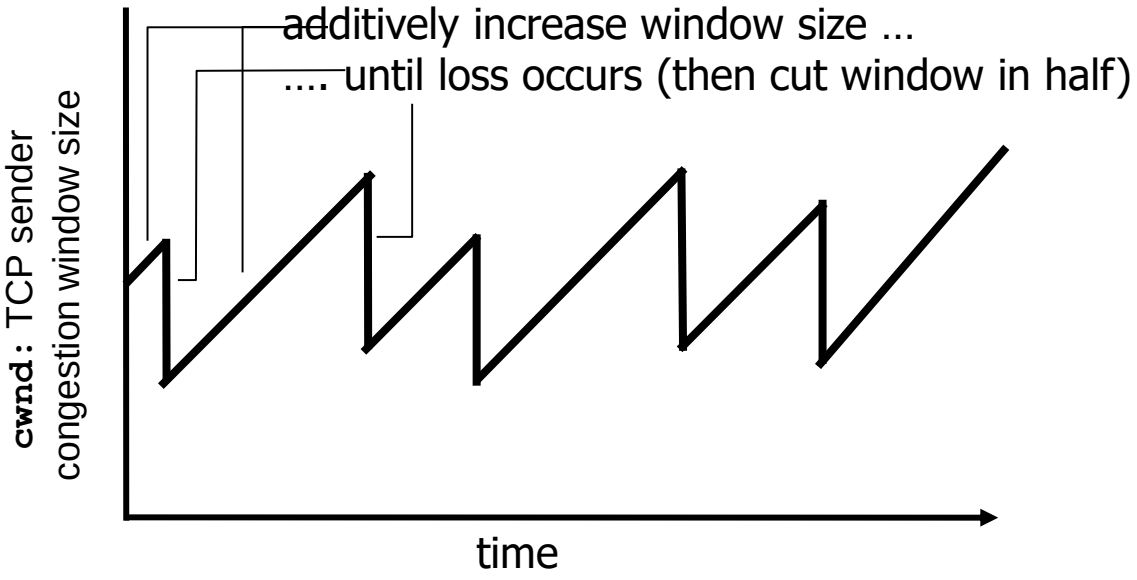
Principles of congestion control

congestion:

- informally: "too many sources sending too much data too fast for *network* to handle"
- different from flow control!
- manifestations:
  - lost packets (buffer overflow at routers)
  - long delays (queueing in router buffers)
- a top-10 problem!

TCP congestion control: additive increase multiplicative decrease

- *approach:* sender increases transmission rate (window size), probing for usable bandwidth, until loss occurs
  - *additive increase:* increase **cwnd** by 1 unit until loss detected
  - *multiplicative decrease*: cut **cwnd** in half after loss



TCP Fairness

fairness goal: if K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of R/K