Internet Protocols
Fall 2004

Lecture 12

TCP

Andreas Terzis

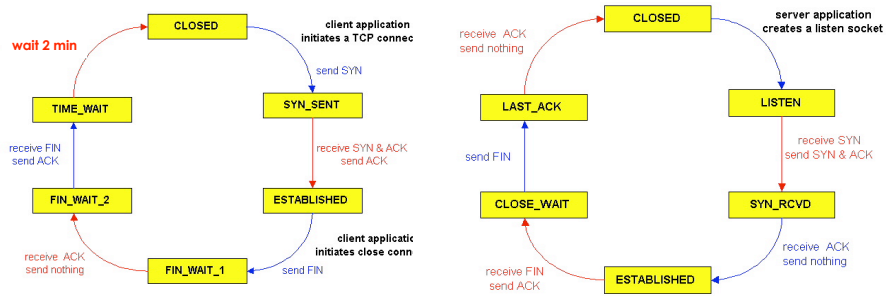
Outline

- TCP Connection Management
- Sliding Window
- ACK Strategy
- Nagle's algorithm
- Timeout estimation
- Flow Control

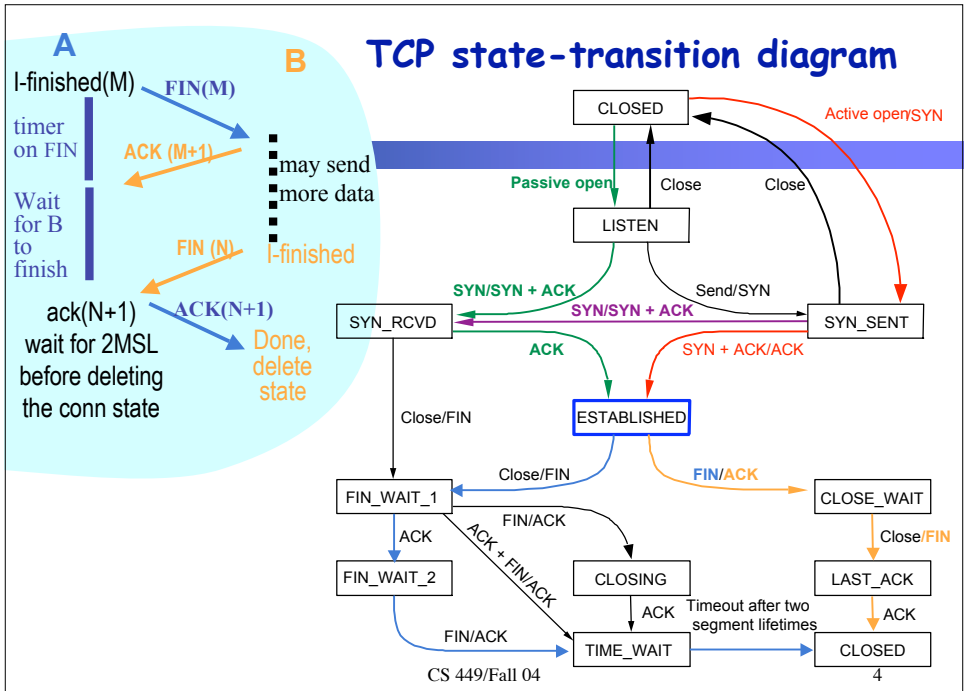
TCP Connection Management

TCP client lifecycle

TCP server lifecycle



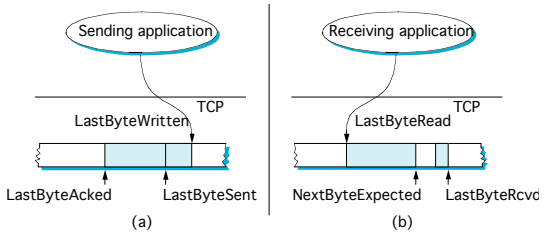
TCP state-transition diagram



Connection Reset

- RST Segments
  - When connection request arrives to non-existing server, OS sends back RST signal
  - Can be used to do *abortive release* of established connection
    - Receiver throws away queued data

Sliding Window Revisited



- Sending side
  - **LastByteAcked < = LastByteSent**
  - **LastByteSent < = LastByteWritten**
  - buffer bytes between **LastByteAcked** and **LastByteWritten**
- Receiving side
  - **LastByteRead < NextByteExpected**
  - **NextByteExpected < = LastByteRcvd +1**
  - buffer bytes between **NextByteRead** and **LastByteRcvd**

Protection Against Wrap Around

- 32-bit **`SequenceNum`**

Bandwidth	Time Until Wrap Around
T1 (1.5 Mbps)	6.4 hours
Ethernet (10 Mbps)	57 minutes
T3 (45 Mbps)	13 minutes
FDDI (100 Mbps)	6 minutes
STS-3 (155 Mbps)	4 minutes
STS-12 (622 Mbps)	55 seconds
STS-24 (1.2 Gbps)	28 seconds

Keeping the Pipe Full

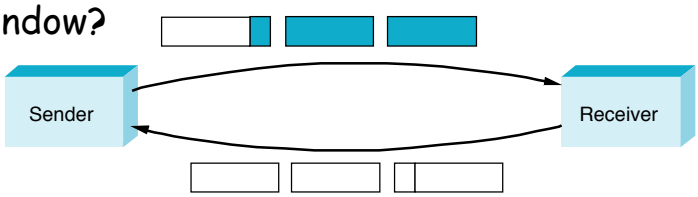
- 16-bit **`AdvertisedWindow`**

Bandwidth	Delay x Bandwidth Product
T1 (1.5 Mbps)	18KB
Ethernet (10 Mbps)	122KB
T3 (45 Mbps)	549KB
FDDI (100 Mbps)	1.2MB
STS-3 (155 Mbps)	1.8MB
STS-12 (622 Mbps)	7.4MB
STS-24 (1.2 Gbps)	14.8MB

assuming 100ms RTT

Silly Window Syndrome

- How aggressively does sender exploit open window?



- Receiver-side solutions
  - after advertising zero window, wait for space equal to a maximum segment size (MSS)
  - delayed acknowledgements

TCP Recvr: when to send ACK?

Event	TCP Receiver action
in-order segment arrival, no gaps, everything earlier already ACKed	delayed ACK: wait up to 200ms, If nothing arrived, send ACK
in-order segment arrival, no gaps, one delayed ACK pending	immediately send one cumulative ACK
out-of-order arrival: higher-than-expect seq. #, gap detected	send duplicate ACK, indicating seq. # of next expected byte
arrival of segment that partially or completely fills a gap	immediate ACK if segment starts at the lower end of gap

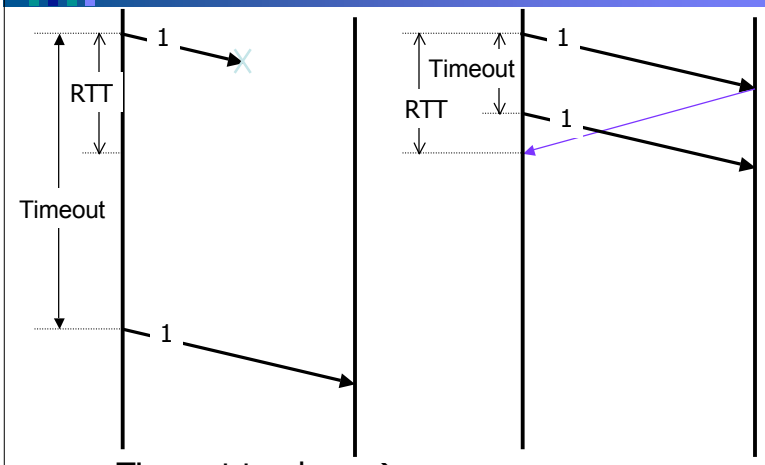
Nagle's Algorithm

- How long does sender delay sending data?
  - too long: hurts interactive applications
  - too short: poor network utilization
  - strategies: timer-based vs self-clocking
- When application generates additional data
  - if fills a max segment (and window open): send it
  - else
    - if there is unack'ed data in transit: buffer it until ACK arrives
    - else: send it

Setting Timers

- The sender needs to set retransmission timers in order to know when to retransmit a packet the may have been lost
- How long to set the timer for?
  - Too short: may retransmit before data or ACK has arrived, creating duplicates
  - Too long: if a packet is lost, will take a long time to recover (inefficient)

Timing Illustration



Timeout too long → inefficiency

Timeout too short → duplicate packets

Adaptive Timers

- The amount of time the sender should wait is about the round-trip time (RTT) between the sender and receiver
  - For link-layer networks (LANs), this value is essentially known
  - For multi-hop WANS, rarely known
- Must work in both environments, so protocol should adapt to the path behavior
- Measure successive ack delays T(n)
  Set timeout = average + 4 deviations

RTT measurement and RTO

SRTT = (1-α) x SRTT + α x SampleRTT
rttvar = rttvar + β x (|diff| - rttvar)
Where diff = SampleRTT - SRTT

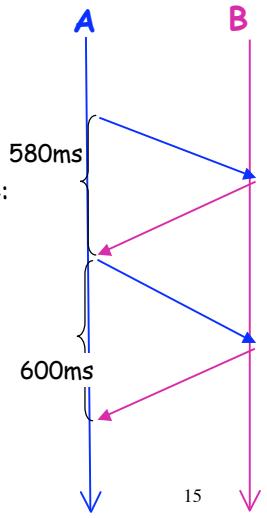
Assume SRTT = 500msec, rttvar = 120, α = 1/8, β = 1/4:

diff = SampleRTT - SRTT = 80ms

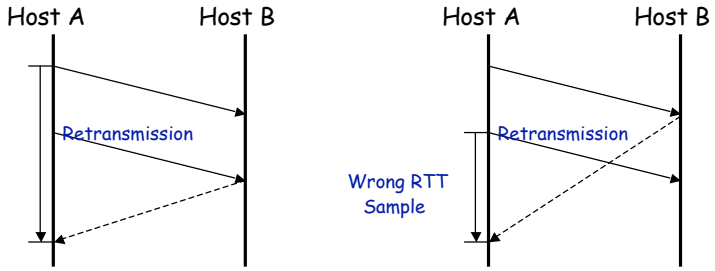
SRTT = SRTT + α x diff = 510ms

rttvar = rttvar + β (|diff| – rttvar) = 110

RTO = SRTT + 4 x rttvar = 510 + 440 = 950



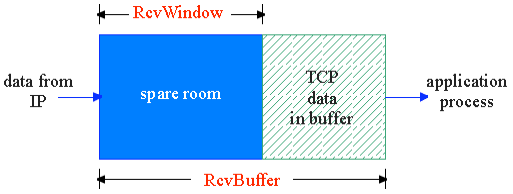
How to measure RTT in case of retransmission?



- Karn's algorithm
  - On retx, don't update estimated RTT (and double RTO)

TCP Flow Control

- receive side of TCP connection has a receive buffer:

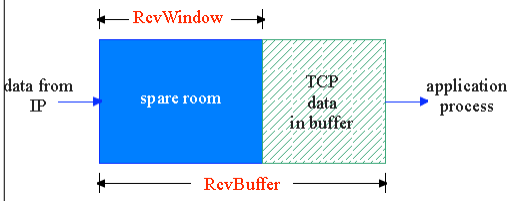


- app process may be slow at reading from buffer

flow control
sender won't overflow receiver's buffer by transmitting too much, too fast

- speed-matching service: matching the send rate to the receiving app's drain rate

TCP Flow control: how it works



(Suppose TCP receiver discards out-of-order segments)

- spare room in buffer

= **RcvWindow**

= **RcvBuffer-[LastByteRcvd - LastByteRead]**

- Rcvr advertises spare room by including value of **RcvWindow** in segments
- Sender limits unACKed data to **RcvWindow**
  - guarantees receive buffer doesn't overflow

Flow Control (cont.)

- If the receiver indicated size of zero in last ACK how can the server send more data?
  - Sender periodically sends probes
  - Receiver sends *window update* ACK