

TCP

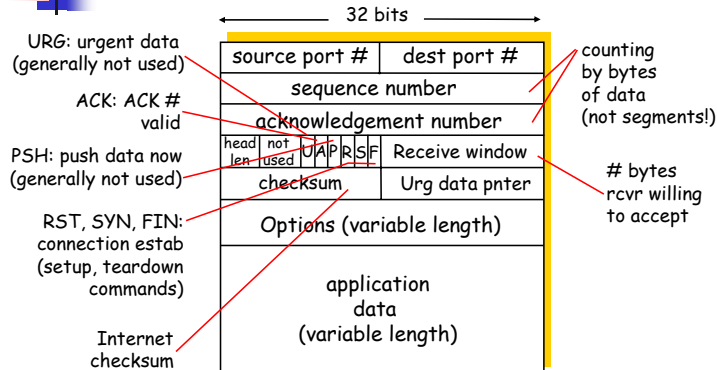
Kai Shen

Dept. of Computer Science, University of Rochester

TCP: Overview

- **connection-oriented:**
 - handshaking (exchange of control msgs) to initialize sender, receiver state before data exchange
- **pipelined:**
 - multiple in-flight segments
- **full duplex data:**
 - bi-directional data flow in same connection
- **reliable data transfer:**
 - guaranteed arrival, no error, in order
- **flow controlled:**
 - sender does not overwhelm receiver
- **congestion controlled:**
 - sender does not overwhelm the network
- **no delay or bandwidth guarantee.**

TCP Segment Structure



TCP Reliable Data Transfer

- TCP provides reliable data transfer service on top of IP's unreliable service
 - Pipelined transmissions
 - Cumulative ACKs
 - When the receiver receives out-of-order segments, it buffers them and re-ACKs the last in-order data
 - Retransmit a single segment at each timeout
 - The sender retransmits at timeout or receiving duplicate ACKs
- Somewhere between Go-back-N and Selective Repeat, with some additional twists.

TCP Timeout

Q: principles for setting transmission timeout value?

- longer than normal RTT (round trip time)
 - but RTT varies
- too short: premature timeout and unnecessary retransmissions
- too long: slow reaction to segment loss

10/17/2007

CSC 257/457 - Fall 2007

5

Estimating Round Trip Time

Q: how to estimate RTT?

- SampleRTT:** measured time from segment transmission until ACK receipt
- SampleRTT** fluctuates, we want estimated RTT "smoother" to avoid instability (pre-mature reaction to short-term spikes)
 - average several recent measurements, not just current **SampleRTT**
- we also want to give more recent measurements higher weight in case things do change

10/17/2007

CSC 257/457 - Fall 2007

6

EWMA - Exponentially Weighted Moving Average

- influence of past sample decreases exponentially fast

$$\text{EstimatedRTT} = \frac{\text{SampleRTT}_1 + \alpha \cdot \text{SampleRTT}_2 + \alpha^2 \cdot \text{SampleRTT}_3 + \dots}{1 + \alpha + \alpha^2 + \dots}$$

SampleRTT₁ is RTT for the most recent data segment,
SampleRTT₂ is RTT for the next recent data segment, etc.

$$\Rightarrow \text{EstimatedRTT} = \alpha \cdot \text{EstimatedRTT}_{\text{last}} + (1 - \alpha) \cdot \text{SampleRTT}_1$$

- typical value: $\alpha = 0.875$

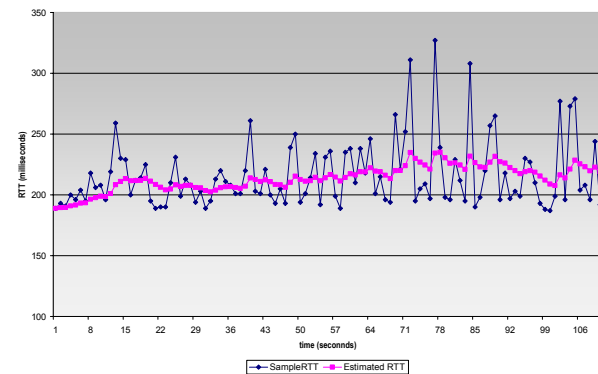
10/17/2007

CSC 257/457 - Fall 2007

7

Example RTT Estimation

RTT: gaia.cs.umass.edu to fantasia.eurecom.fr



10/17/2007

CSC 257/457 - Fall 2007

8

TCP Timeout

Setting the timeout:

- EstimatedRTT plus "safety margin"
 - large variation in EstimatedRTT → larger safety margin
- we need to estimate of how much SampleRTT deviates from EstimatedRTT (EWMA):

$$\text{DevRTT} = \beta * \text{DevRTT}_{\text{last}} + (1 - \beta) * |\text{SampleRTT} - \text{EstimatedRTT}|$$

(typically, $\beta = 0.75$)

Then set timeout interval:

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 * \text{DevRTT}$$

10/17/2007

CSC 257/457 - Fall 2007

9

TCP Sender Events and Processing

Data ready to send:

- create segment with seq #
- seq # is byte-stream number of first data byte in segment
- start timer
- timeout value: we just decided it!!

Timeout:

- retransmit segment that caused timeout
- restart timer

ACK rcvd:

- slide sender window if acknowledges previously unacked segments
- retransmit if 3 duplicate ACKs

10/17/2007

CSC 257/457 - Fall 2007

10

TCP byte-oriented seq. #'s and ACKs

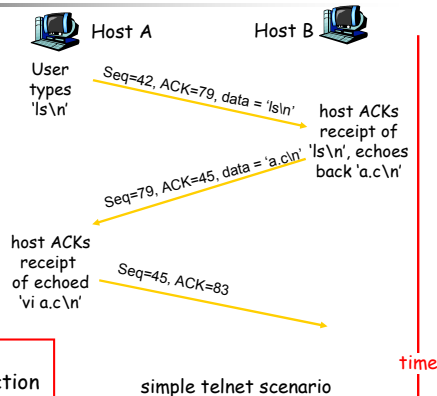
Seq. #'s:

- byte stream "number" of first byte in segment's data

ACKs:

- seq # of next byte expected from other side
- cumulative ACK

ACKs piggybacked in data segments in the other direction

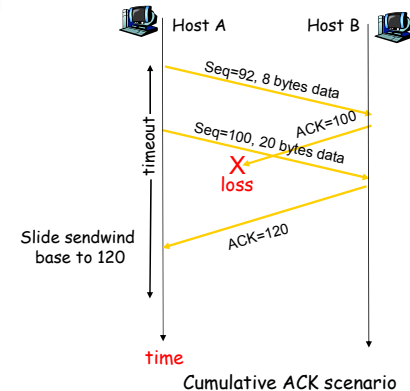


10/17/2007

CSC 257/457 - Fall 2007

11

TCP in Action: Cumulative ACK



10/17/2007

CSC 257/457 - Fall 2007

12

Fast Retransmission

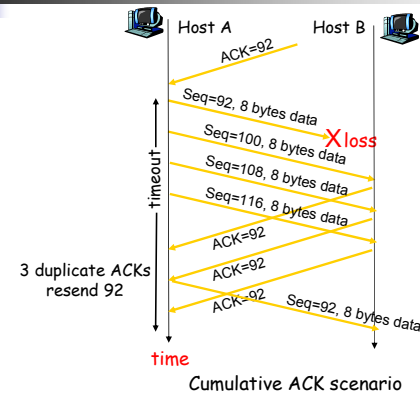
- Time-out period often relatively long:
 - long delay before resending lost packet
- When receiver receives out-of-order segments, it re-ACKs the last in-order byte
- If sender receives 3 ACKs for the same data, it supposes that segment after ACKed data was lost:
 - fast retransmission:** resend segment before timer expires, restart timer

10/17/2007

CSC 257/457 - Fall 2007

13

TCP in Action: Duplicate ACKs and Fast Retransmission



10/17/2007

CSC 257/457 - Fall 2007

14

Outline

- segment structure
- reliable data transfer
- flow control**
- connection management

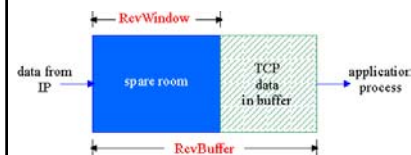
10/17/2007

CSC 257/457 - Fall 2007

15

TCP Flow Control

- receive side of TCP connection has a receive buffer:



- app process may be slow at reading from buffer

flow control —
sender does not overflow receiver's buffer by transmitting too much, too fast

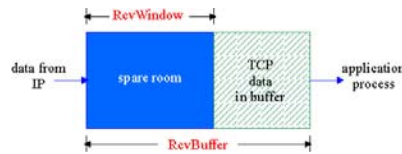
- speed-matching service: matching the send rate to the receiving app's drain rate

10/17/2007

CSC 257/457 - Fall 2007

16

TCP Flow Control: how it works?



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

10/17/2007

CSC 257/457 - Fall 2007

17

TCP Connection Management

- **Establishment:**
 - TCP sender, receiver establish "connection" before exchanging data segments
 - initialize TCP variables: starting seq. #s, MSS, buffers, flow control info (e.g. `RevWindow`)
- MSS is the maximum TCP segment size each side is willing to accept
 - typically the largest segment size fit into a link-layer frame
- **Teardown:**
 - freeing up resources after mutually close

10/17/2007

CSC 257/457 - Fall 2007

18

TCP Connection Establishment

Three way handshake:

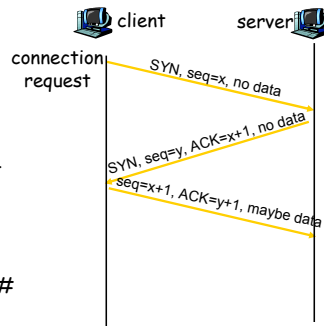
Step 1: client (active open) sends TCP SYN segment to server

- specifies initial seq #
- no data

Step 2: server (passive open) host receives SYN, replies with SYNACK segment

- server allocates buffers
- specifies server initial seq. #

Step 3: client receives SYNACK, replies with ACK segment, which may contain data



10/17/2007

CSC 257/457 - Fall 2007

19

TCP Connection Teardown

Closing a connection:

close socket: `close(sockfd)`;

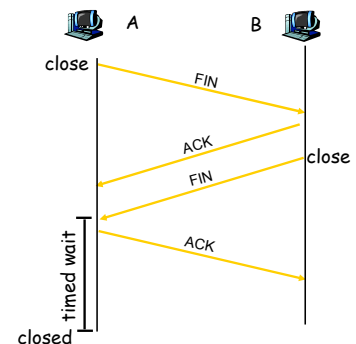
Step 1: A (active closing host) sends TCP FIN control segment to server

Step 2: B (passive closing host) receives FIN, replies with ACK. Closes connection, sends FIN.

Step 3: A receives FIN, replies with ACK.

- Enters "timed wait" - resend ACK in case it is lost

Step 4: B receives ACK. Connection closed.

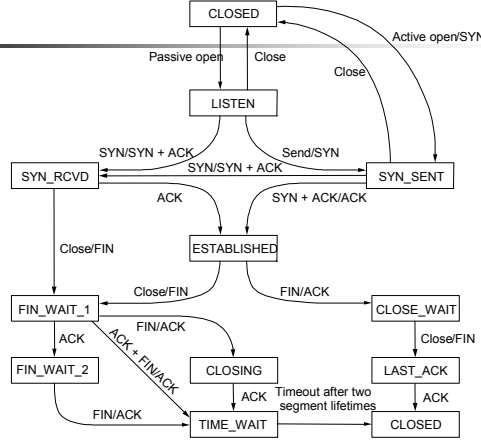


10/17/2007

CSC 257/457 - Fall 2007

20

TCP State Transition Diagram



Disclaimer

- Parts of the lecture slides contain original work of James Kurose, Larry Peterson, and Keith Ross. The slides are intended for the sole purpose of instruction of computer networks at the University of Rochester. All copyrighted materials belong to their original owner(s).