

TCP

Kai Shen

2/27/2013
CSC 257/457 - Spring 2013
1

TCP: Overview

- **Connection-oriented:**
 - handshaking (exchange of control msgs) to initialize sender, receiver state before data exchange
- **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.**

2/27/2013
CSC 257/457 - Spring 2013
2

TCP Segment Structure

The diagram illustrates the structure of a TCP segment, which is 32 bits long. The fields are as follows:

- source port #** and **dest port #**: 16 bits each.
- sequence number**: 32 bits.
- acknowledgement number**: 32 bits.
- header flags**: URG, ACK, PSH, RST, SYN, FIN.
- Receive window**: 16 bits.
- checksum**: 16 bits.
- Urg data pointer**: 16 bits.
- Options**: variable length.
- application data**: variable length.

Annotations on the left side:

- URG: urgent data** (generally not used)
- ACK: ACK #** valid
- PSH: push data now** (generally not used)
- RST, SYN, FIN:** connection estab (setup, teardown commands)
- Internet checksum**

Annotations on the right side:

- counting by bytes of data (not segments!)
- # bytes willing to receive

2/27/2013
CSC 257/457 - Spring 2013
3

Maximum Segment Size (MSS)

- MSS is the maximum TCP segment that'd fit into the link layer frame
- Local MSS
- Path MSS
 - Try and error probing

2/27/2013
CSC 257/457 - Spring 2013
4

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 when receiving duplicate ACKs
- Somewhere between Go-back-N and Selective Repeat, closer to which?

2/27/2013

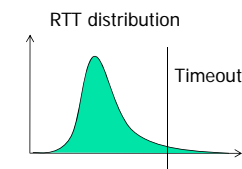
CSC 257/457 - Spring 2013

5

TCP Timeout

Q: principles for setting transmission timeout value?

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



2/27/2013

CSC 257/457 - Spring 2013

6

Expected Round Trip Time

Derive expected RTT from past RTT measurement.

- Basic measurement: measured time from segment transmission until ACK receipt
- **Stability:** RTT fluctuates, we want to avoid instability (pre-mature reaction to short-term spikes)
 - average several recent measurements, not just current RTT
- **Agility:** in case things do change, we want to adjust quickly
 - give more recent measurements higher weight

2/27/2013

CSC 257/457 - Spring 2013

7

EWMA – Exponentially Weighted Moving Average

- influence of past samples decreases exponentially fast

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

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

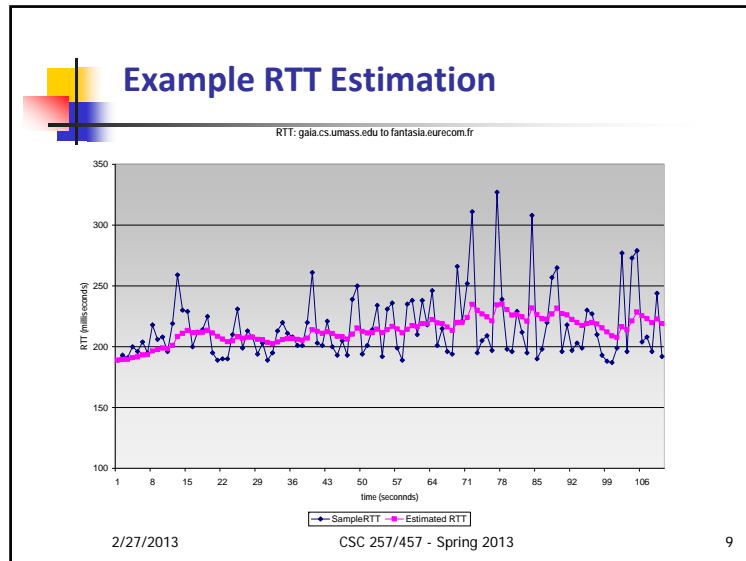
$$\Rightarrow \text{ExpectedRTT} = \alpha \cdot \text{ExpectedRTT}_{\text{last}} + (1 - \alpha) \cdot \text{SampleRTT}_1$$

- typical value in TCP: $\alpha = 0.875$

2/27/2013

CSC 257/457 - Spring 2013

8



TCP Timeout

Setting the timeout:

- ExpectedRTT plus “safety margin”
 - larger variation in RTT → larger safety margin
 - $\text{TimeoutInterval} = \text{ExpectedRTT} + 4 * \text{DevRTT}$
- DevRTT also estimated using EWMA of past measurements

2/27/2013 CSC 257/457 - Spring 2013 10

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!!

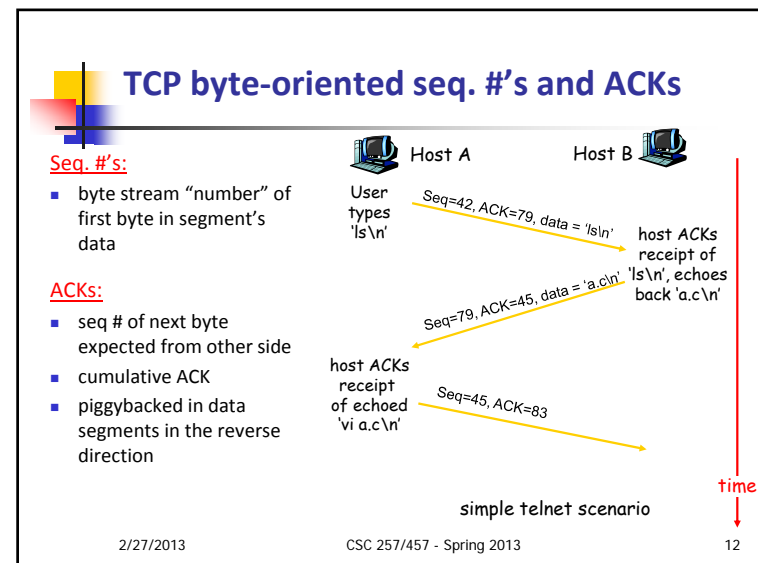
Timeout:

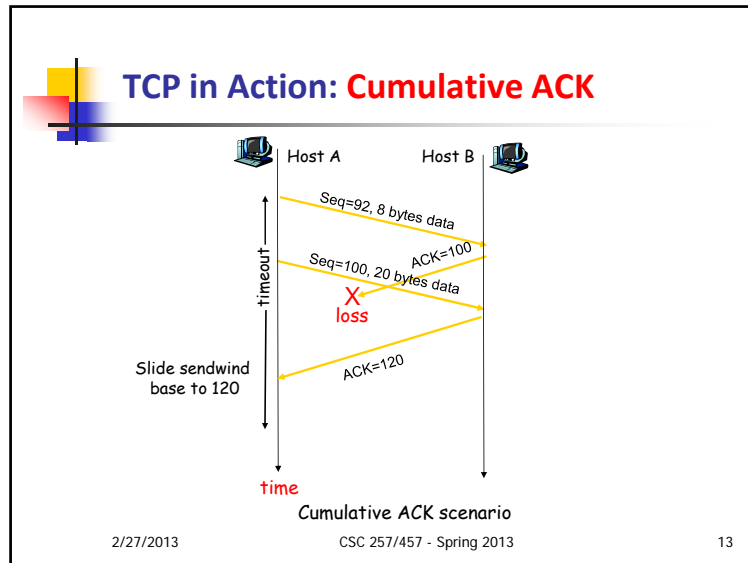
- retransmit segment that caused timeout
- restart timer

ACK rcvd:

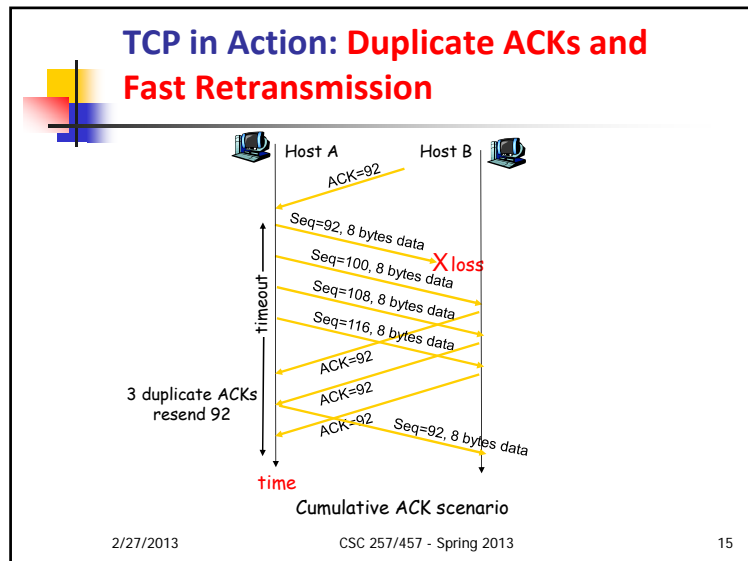
- slide sender window if acknowledges previously unacked segments
- retransmit if 3 duplicate ACKs

2/27/2013 CSC 257/457 - Spring 2013 11





- ### 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
- 2/27/2013 CSC 257/457 - Spring 2013 14



- ### Outline
- Segment structure
 - Reliable data transfer
 - Flow control
 - Connection management
- 2/27/2013 CSC 257/457 - Spring 2013 16

TCP Flow Control

- Receive side of TCP connection has a receive buffer:
- App process may be slow at reading from buffer.

The diagram shows a flow of data from IP into a 'spare room' (blue box) and then into a 'TCP data in buffer' (green hatched box). The 'spare room' is labeled 'RevWindow' and the 'TCP data in buffer' is labeled 'RevBuffer'. An arrow points from the buffer to an 'application process'.

Flow Control:

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

2/27/2013 CSC 257/457 - Spring 2013 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. **RcvWindow**)
- Teardown:**
 - Free up resources after mutually close

2/27/2013 CSC 257/457 - Spring 2013 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

The sequence diagram shows a client and server. The client sends a 'connection request' (SYN, seq=x, no data) to the server. The server responds with 'SYN, seq=y, ACK=x+1, no data'. The client then responds with 'seq=x+1, ACK=y+1, maybe data'.

2/27/2013 CSC 257/457 - Spring 2013 19

TCP Connection Teardown

Closing a connection:

close socket: `close(sockfd);`

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

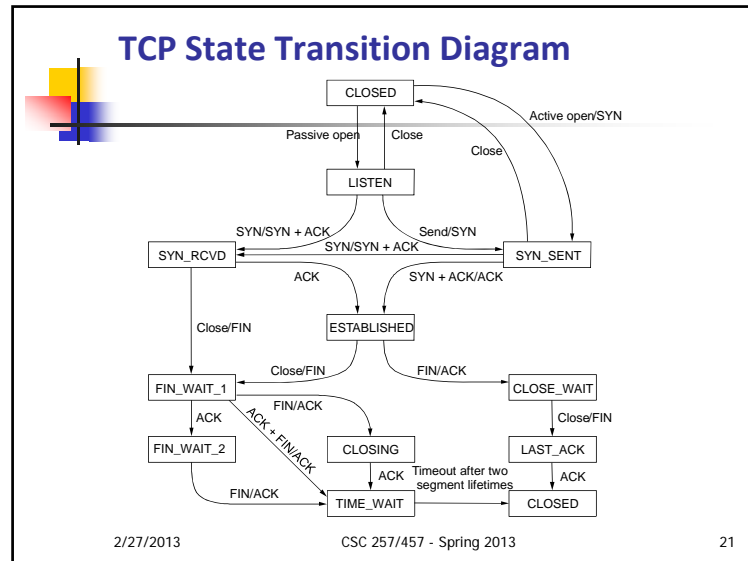
Step 2: B (passive closing host) receives FIN, replies with ACK. Closes connection, sends FIN.

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.

The sequence diagram shows host A and host B. Host A sends 'close' and 'FIN' to host B. Host B sends 'ACK' and 'close' back to host A. Host A sends 'ACK' to host B. A 'timed wait' period is shown on host A's side. Finally, host B sends 'closed' and host A sends 'closed'.

2/27/2013 CSC 257/457 - Spring 2013 20



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).

2/27/2013 CSC 257/457 - Spring 2013 22