

The Transport Layer: TCP & Congestion Control

Smith College, CSC 249
February 28, 2008

slides mostly from J.F Kurose and K.W. Ross,
copyright 1996-2007

Discussion Questions Reviewing Tuesday & Reliable Data Transport

2

Discussion Question 1

- Suppose Host A sends two TCP segments back to back to Host B over a TCP connection.
 - ❖ What might be the first sequence number?
 - ❖ If 20 bytes are sent, what is the second has sequence?
 - ❖ Suppose that the first segment is lost but the second segment arrives at B. In the acknowledgment that Host B sends to Host A, what will be the acknowledgment number?
- a) A random initial number, plus 20 bytes
- b) ACK number = initial sequence number - *i.e.*, "still waiting for the first segment"

3

Discussion Question 2

- Consider an rdt protocol that uses only NAKs. Suppose the sender sends data infrequently. Would a NAK-only protocol be preferable to a protocol that uses ACKs? Why?
- In a NAK only protocol, the loss of packet x is only detected by the receiver when packet $x+1$ is received. That is, the receiver receives $x-1$ and then $x+1$, only when $x+1$ is received does the receiver realize that x was missed. If there is a long delay between the transmission of x and the transmission of $x+1$, then it will be a long time until x can be recovered, under a NAK-only protocol.

4

Discussion Question 3

- ❑ Now suppose the sender has a lot of data to send and the end-to-end connection experiences few losses. In this second case, would a NAK-only protocol be preferable to a protocol that uses ACKs? Why?
- ❑ If data is being sent often, then recovery under a NAK-only scheme could happen quickly. Moreover, if errors are infrequent, then NAKs are only occasionally sent (when needed), and ACK are never sent - a significant reduction in feedback in the NAK-only case over the ACK-only case.

5

Chapter 3 outline

- ❑ 3.1 Transport-layer services
- ❑ 3.2 Multiplexing and demultiplexing
- ❑ 3.3 Connectionless transport: UDP
- ❑ 3.4 Principles of reliable data transfer
- ❑ 3.5 Connection-oriented transport: TCP
 - ❖ segment structure
 - ❖ reliable data transfer
 - ❖ flow control
 - ❖ connection management
- ❑ 3.6 Principles of congestion control
- ❑ 3.7 TCP congestion control

6

TCP Connection Management: Set up

Recall: TCP sender, receiver establish "connection" before exchanging data segments

Three way handshake:

- Step 1:** client host sends TCP SYN segment to server
 - ❖ "SYN" for "synchronize"
 - ❖ specifies initial sequence #
 - ❖ no data is sent
- Step 2:** server host receives SYN, replies with SYNACK segment
 - ❖ server allocates buffers and variables
 - ❖ specifies its own, server initial sequence #
- Step 3:** client receives SYNACK, replies with ACK segment, which may contain data
 - ❖ client allocates buffers and variables

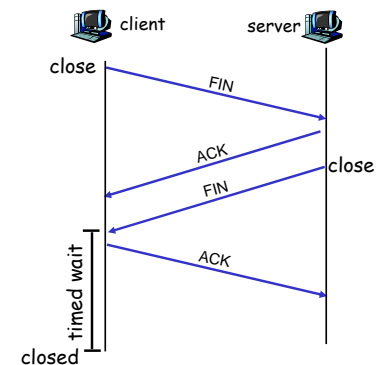
7

TCP Connection Management: Close

Closing a connection:

Step 1: client end system sends TCP FIN control segment to server

Step 2: server receives FIN, replies with ACK. Closes connection, sends FIN.



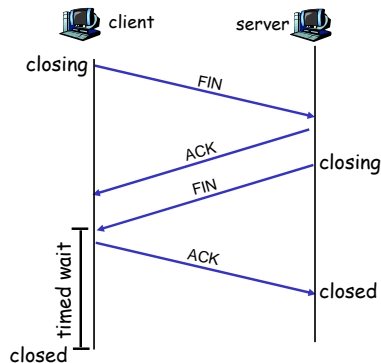
8

TCP Connection Management (cont.)

Step 3: client receives FIN, replies with ACK.

- ❖ Enters "timed wait" - client able to resend final ACK in case it is lost

Step 4: server, receives ACK. Connection closed.



9

Recap

- ❑ Overview of transport layer services
- ❑ Multiplexing and demultiplexing
- ❑ UDP
 - ❖ Characteristics and uses
- ❑ Generic reliable data transport
 - ❖ Incremental development up to a realistic and useful protocol
- ❑ Segment formats
- ❑ Specific implementation of TCP
 - ❖ with congestion control to come...

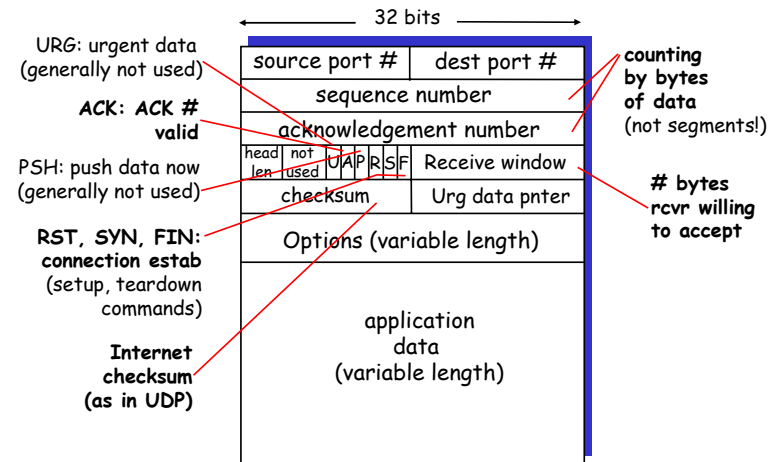
10

Chapter 3 outline

- ❑ 3.1 Transport-layer services
- ❑ 3.2 Multiplexing and demultiplexing
- ❑ 3.3 Connectionless transport: UDP
- ❑ 3.4 Principles of reliable data transfer
- ❑ 3.5 Connection-oriented transport: TCP
 - ❖ segment structure
 - ❖ reliable data transfer
 - ❖ **flow control**
 - ❖ connection management
- ❑ 3.6 Principles of congestion control
- ❑ 3.7 TCP congestion control

11

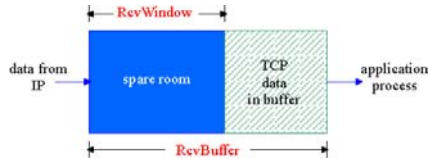
TCP segment structure



12

TCP Flow Control

- receive side of TCP connection has a receive buffer:



- application process may be slow at reading from buffer

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

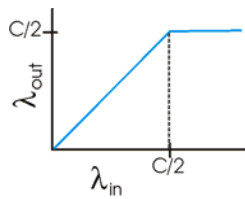
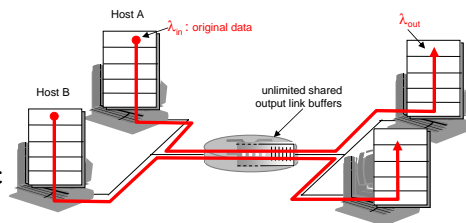
- speed-matching service: matching the send rate to the receiving application's drain rate
- Sender ensures that $(LastByteSent - LastByteAked) \leq RcvWindow$

Chapter 3 outline

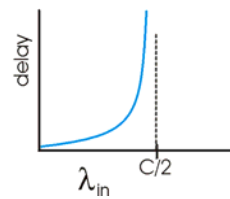
- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
 - segment structure
 - reliable data transfer
 - flow control
 - connection management
- 3.6 & 3.7 Congestion control

Causes congestion 1: Limited Bandwidth

- two senders, two receivers
- one router, infinite buffers
- Rate = λ bytes/sec



Throughput

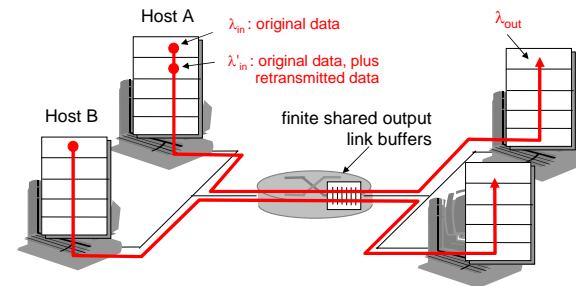


Delay

- large delays when the packet arrival rate approaches the link capacity

Causes of congestion 2: Limited Buffers

- one router, *finite* buffers
- sender retransmission of lost packet



Principles of Congestion Control

- ❑ **Packet loss** is caused by overflowing router buffers
- ❑ **Retransmission** treats the symptom
- ❑ **Congestion control** treats the cause
 - ❖ Too many packets being sent

17

Costs of congestion

- 1) **Large queuing delays** are experienced as the packet-arrival rate nears the link capacity
- 2) **Sender must retransmit** in order to compensate for dropped (lost) packets due to buffer overflow
- 3) Unneeded retransmission (from long delays) cause routers to **waste link bandwidth**
- 4) If a packet is dropped, then the **upstream transmission capacity** used for the packet up to that point was **wasted**

19

Principles of Congestion Control

Congestion:

- ❑ 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

18

TCP Congestion Control

- ❑ Congestion v. Flow control?
- ❑ Each sender limits its sending rate as a function of perceived network congestion
 - ❖ If it perceives little congestion, the rate is increased
 - ❖ If it senses congestion, the rate is decreased
- ❑ Three questions
 1. **How** does a sender **sense** congestion between itself and its destination?
 2. **How** does a sender **limit** its sending rate?
 3. **What algorithm** should be used to change the send-rate as a function of perceived congestion?

20

TCP Congestion Control: details

How does sender perceive congestion?

□ A loss event is ?

- ❖ A timeout *or*
- ❖ 3 duplicate ACKs

How does sender limit its send rate?

□ TCP sender reduces the send rate via changing the "CongWin" after a loss event

$$\text{LastByteSent} - \text{LastByteAcked} \leq \text{CongWin}$$

21

TCP Congestion Control Window

- CongWin is dynamic, function of perceived network congestion
- Sender limits transmission:
 $\text{LastByteSent} - \text{LastByteAcked} \leq \text{CongWin}$



$$\text{send rate} = \frac{\text{CongWin}}{\text{RTT}} \text{ Bytes/sec}$$

22

TCP Congestion Control Algorithm

Three major mechanisms:

- 1) Slow start
- 2) AIMD = additive increase, multiplicative decrease
- 3) Reaction to timeout events vs. 3 duplicate ACKs

Are used to... ?

- ❖ Adjust CongWin

23

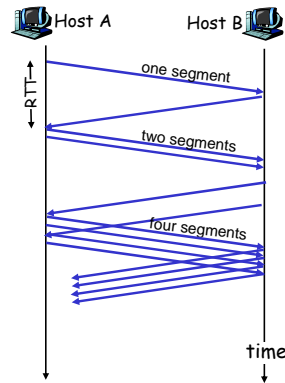
TCP Slow Start

- When connection begins, CongWin = 1 MSS
 - ❖ Sender rate thus = MSS/RTT
 - Example: MSS = 500 bytes & RTT = 200 msec
 - So initial send rate = 20 kbps
 - ❖ But, available bandwidth probably \gg MSS/RTT
 - ❖ Desirable to quickly ramp up to respectable rate
- Thus when connection begins, increase rate exponentially fast until first loss event
 - ❖ Grow window 1MSS for *each* ACK received
- Summary: initial rate is slow (MSS/RTT) but ramps up exponentially fast

24

TCP Slow Start with Exp. Increase

- When connection begins, increase rate exponentially until first loss event:
 - ❖ double CongWin every RTT
 - ❖ done by incrementing CongWin for every ACK received
- **Summary:** initial rate is slow but ramps up exponentially fast



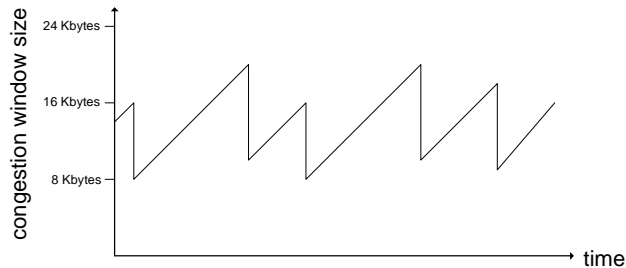
25

TCP congestion control - AIMD: additive increase, multiplicative decrease

- **Approach:** increase transmission rate (window size), probing for usable bandwidth, until loss occurs
 - ❖ **additive increase:** increase **CongWin** by 1 MSS (max segment size) every RTT until loss detected
 - Vs. "Slow Start" with increase of 1 MSS for every ACK received
 - ❖ **multiplicative decrease:** cut **CongWin** in half after loss

26

TCP congestion control - AIMD: additive increase, multiplicative decrease



Saw tooth behavior:
probing for bandwidth

27

Reaction to Loss Events

- After 3 duplicate ACKs:
 - ❖ **CongWin is cut in half**
 - ❖ window then grows linearly
- After timeout event:
 - ❖ **CongWin instead set to 1 MSS**
 - ❖ window then grows exponentially (doubling) ...
 - ❖ to a threshold, then grows linearly

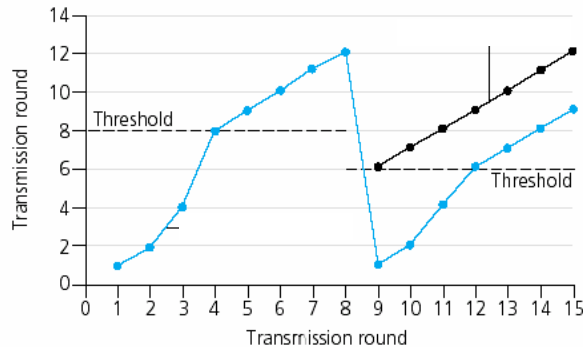
Philosophy:

- 3 dup ACKs indicates network capable of delivering some segments
- timeout indicates a "more alarming" congestion scenario

28

Reaction to Loss Events Refinement

- Exponential increase switches to linear increase when CongWin gets to the 'threshold' value (size)



29

Summary: TCP Congestion Control

Increase Sending Rate Phase:

- When CongWin is below Threshold, sender in **slow-start** phase, window grows exponentially.
- When CongWin is above Threshold, sender is in **congestion-avoidance** phase, window grows linearly.

Decrease Sending Rate Phase:

- When a **triple duplicate ACK** occurs, Threshold set to CongWin/2 and CongWin set to Threshold.
- When **timeout** occurs, Threshold set to CongWin/2 and CongWin is set to 1 MSS.

30

TCP Fairness

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

Fairness and UDP

- Multimedia applications often do not use TCP
 - do not want rate throttled by congestion control
- Instead use UDP:
 - pump audio/video at constant rate, tolerate packet loss

Fairness and parallel TCP connections

- Nothing prevents an application from opening parallel connections between 2 hosts.
- Web browsers do this

31

Chapter 3 Summary

Our goals:

- understand principles behind transport layer services:
 - multiplexing/de-multiplexing
 - reliable data transfer
 - flow control
 - congestion control
- learn about transport layer protocols in the Internet:
 - UDP: connectionless transport
 - TCP: connection-oriented transport
 - TCP congestion control

32