Computer Networking: A Top Down Approach

Seventh Edition

Chapter 3

Transport Layer

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Transport Layer

our goals:

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

Learning Objectives (1 of 10)

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

Transport Services and Protocols

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

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Transport vs. Network Layer

- **network layer***:* logical communication between hosts
- **transport layer***:* logical communication between processes
  - relies on, enhances, network layer services

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 demux to in-house siblings
- network-layer protocol = postal service

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Internet 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

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Learning Objectives (2 of 10)

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

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Multiplexing/Demultiplexing

multiplexing at sender:

handle data from multiple sockets, add transport header (later used for demultiplexing)

demultiplexing at receiver:

use header info to deliver received segments to correct socket

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

How Demultiplexing Works

- host receives IP datagrams
  - each datagram has source IP address, destination IP address
  - each datagram carries one transport-layer segment
  - each segment has source, destination port number
- host uses **IP addresses & port numbers** to direct segment to appropriate socket

TCP/UDP segment format

Pearson
Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Connectionless Demultiplexing

- **recall:** created socket has host-local port #:

```
DatagramSocket mySocket1
= new DatagramSocket(12534);
```

- when host receives UDP segment:
  - checks destination port # in segment
  - directs UDP segment to socket with that port #

- **recall:** when creating datagram to send into UDP socket, must specify
  - destination IP address
  - destination port #

IP datagrams with **same dest. port #,** but different source IP addresses and/or source port numbers will be directed to **same socket** at dest

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Connectionless Demux: Example

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Connection-Oriented Demux

- TCP socket identified by 4-tuple:
  - **source IP address**
  - **source port number**
  - **dest IP address**
  - **dest port number**
- demux: receiver uses all four values to direct segment to appropriate socket
- server host may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
- web servers have different sockets for each connecting client
  - non-persistent HTTP will have different socket for each request

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Connection-Oriented Demux: Example (1 of 2)

three segments, all destined to IP address: B, dest port: 80 are demultiplexed to **different** sockets

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Connection-Oriented Demux: Example (2 of 2)

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Learning Objectives (3 of 10)

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

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

UDP: User Datagram Protocol [RFC 768]

- "no frills," "bare bones" Internet 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)
  - DNS
  - SNMP
- reliable transfer over UDP:
  - add reliability at application layer
  - application-specific error recovery!

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

UDP: Segment Header (1 of 2)

UDP segment format

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

UDP: Segment Header (2 of 2)

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

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

UDP Checksum

Goal*:* detect “errors” (example, flipped bits) in transmitted segment

sender:

- treat segment contents, including header fields, as sequence of 16-bit integers
- checksum: addition (one’s complement sum) of segment contents
- sender puts checksum value into UDP checksum field

receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected. **But maybe errors nonetheless***?* More later ….

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Internet Checksum: Example

example: add two 16-bit integers

	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
wraparound	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
sum	1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

Note*:* when adding numbers, a carryout from the most significant bit needs to be added to the result

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Learning Objectives (4 of 10)

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

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Principles of Reliable Data Transfer

- important in application, transport, link layers
  - top-10 list of important networking topics!

- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

Pearson
Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Reliable Data Transfer: Getting Started (1 of 2)

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Reliable Data Transfer: Getting Started (2 of 2)

we'll:

- incrementally develop sender, receiver sides of **r**eliable **d**ata **t**ransfer protocol (rdt)
- consider only unidirectional data transfer
  - but control info will flow on both directions!
- use finite state machines (FSM) to specify sender, receiver

Pearson
Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

rdt1.0: Reliable Transfer over a Reliable Channel

- underlying channel perfectly reliable
  - no bit errors
  - no loss of packets
- separate FSMs for sender, receiver:
  - sender sends data into underlying channel
  - receiver reads data from underlying channel

sender

receiver

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

rdt2.0: Channel with Bit Errors (1 of 2)

- underlying channel may flip bits in packet
  - checksum to detect bit errors
- **the** question: how to recover from errors:

How do humans recover from "errors" during conversation?

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

rdt2.0: Channel with Bit Errors (2 of 2)

- underlying channel may flip bits in packet
  - checksum to detect bit errors
- **the** question: how to recover from errors:
  - **acknowledgements (ACKs*)*:** receiver explicitly tells sender that pkt received OK
  - **negative acknowledgements (NAKs*)*:** receiver explicitly tells sender that pkt had errors
  - sender retransmits pkt on receipt of NAK
- new mechanisms in **`rdt2.0`** (beyond **`rdt1.0`**):
  - error detection
  - feedback: control msgs (ACK, NAK) from receiver to sender

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

rdt2.0: FSM Specification

sender

receiver

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

rdt2.0: Operation with No Errors

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

rdt2.0: Error Scenario

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

rdt2.0 Has a Fatal Flaw!

what happens if ACK / NAK corrupted?

- sender doesn't know what happened at receiver!
- Can't just retransmit: possible duplicate

handling duplicates:

- sender retransmits current pkt if ACK / NAK corrupted
- sender adds **sequence number** to each pkt
- receiver discards (doesn't deliver up) duplicate pkt

stop and wait

sender sends one packet, then waits for receiver response

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

rdt2.1: Sender, Handles Garbled ACK/NAKs

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

rdt2.1: Receiver, Handles Garbled ACK/NAKs

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

rdt2.1: Discussion

sender:

- seq # added to pkt
- two Sequence #'s (0,1) will suffice. Why?
- must check if received ACK/NAK corrupted
- twice as many states
  - state must "remember" whether "expected" pkt should have seq # of 0 or 1

receiver:

- must check if received packet is duplicate
  - state indicates whether 0 or 1 is expected pkt seq #
- note: receiver can **not** know if its last ACK/NAK received OK at sender

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

rdt2.2: A NAK-free Protocol

- same functionality as rdt2.1, using ACKs only
- instead of NAK, receiver sends ACK for last pkt received OK
  - receiver must **explicitly** include seq # of pkt being ACKed
- duplicate ACK at sender results in same action as NAK: **retransmit current pkt**

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

rdt2.2: Sender, Receiver Fragments

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

rdt3.0: Channels with Errors and Loss

new assumption: underlying channel can also lose packets (data, ACKs)

- checksum, Sequence #, ACKs, retransmissions will be of help … but not enough

approach: sender waits "reasonable" amount of time for ACK

- retransmits if no ACK received in this time
- if pkt (or ACK) just delayed (not lost):
  - retransmission will be duplicate, but Sequence #'s already handles this
  - receiver must specify seq # of pkt being ACKed
- requires countdown timer

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

rdt3.0 Sender

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

rdt3.0 in Action (1 of 2)

(a) no loss

(b) packet loss

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

rdt3.0 in Action (2 of 2)

(c) ACK loss

(d) premature timeout/ delayed ACK

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Performance of rdt3.0 (1 of 2)

- rdt3.0 is correct, but performance stinks
- example: 1 Gbps link, 15 ms prop. delay, 8000 bit packet:

$$d_{trans} = \frac{L}{R} = \frac{8000 \text{ bits / packet}}{10^9 \text{ bits / sec}} = 8 \text{ microseconds}$$

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Performance of rdt3.0 (2 of 2)

- U_{sender}: **utilization** – fraction of time sender busy sending

$$U_{sender} = \frac{\frac{L}{R}}{RTT + \frac{L}{R}} = \frac{.008}{30.008} = 0.00027$$

- if RTT = 30 msec, 1KB pkt every 30 msec: 33kB/sec thruput over 1 Gbps link

- network protocol limits use of physical resources!

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

rdt3.0: Stop-and-wait Operation

$$U_{sender} = \frac{\frac{L}{R}}{RTT + \frac{L}{R}} = \frac{.008}{30.008} = 0.00027$$

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Pipelined Protocols

- **pipelining:** sender allows multiple, “in-flight”, yet-to-be acknowledged pkts
  - range of sequence numbers must be increased
  - buffering at sender and/or receiver

a. A stop-and-wait protocol in operation

b. A pipelined protocol in operation

- two generic forms of pipelined protocols: **go-Back-N, selective repeat**

Pearson
Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Pipelining: Increased Utilization

$$U_{\text{sender}} = \frac{\frac{3L}{R}}{RTT + \frac{L}{R}} = \frac{.0024}{30.008} = 0.00081$$

3-packet pipelining increases utilization by a factor of 3!

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Pipelined Protocols: Overview

Go-back-N:

- sender can have up to N unacked packets in pipeline
- receiver only sends **cumulative ack**
  - doesn't ack packet if there's a gap
- sender has timer for oldest unacked packet
  - when timer expires, retransmit **all** unacked packets

Selective Repeat:

- sender can have up to N unack'ed packets in pipeline
- rcvr sends **individual ack** for each packet
- sender maintains timer for each unacked packet
  - when timer expires, retransmit only that unacked packet

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Go-Back-N: Sender

- k-bit seq # in pkt header
- "window" of up to N, consecutive unack'ed pkts allowed

- ACK (n): ACKs all pkts up to, including seq # n – **"cumulative ACK"**
  - may receive duplicate ACKs (see receiver)
- timer for oldest in-flight pkt
- **timeout(n):** retransmit packet n and all higher seq # pkts in window

Pearson
Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

GBN: Sender Extended FSM

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

GBN: Receiver Extended FSM (1 of 2)

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

GBN: Receiver Extended FSM (2 of 2)

- ACK-only: always send ACK for correctly-received pkt with highest **in-order** seq #
  - may generate duplicate ACKs
  - need only remember **`expectedseqnum`**
- out-of-order pkt:
  - discard (don't buffer): **no receiver buffering!**
  - re-ACK pkt with highest in-order seq #

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

GBN in Action

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Selective Repeat (1 of 3)

- receiver **individually** acknowledges all correctly received pkts
  - buffers pkts, as needed, for eventual in-order delivery to upper layer
- sender only resends pkts for which ACK not received
  - sender timer for each unACKed pkt
- sender window
  - **N** consecutive seq #'s
  - limits seq #s of sent, unACKed pkts

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Selective Repeat: Sender, Receiver Windows

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Selective Repeat (2 of 3)

sender

data from above:

- if next available seq # in window, send pkt

timeout(n):

- resend pkt n, restart timer

ACK(n) in [sendbase,sendbase+N]:

- mark pkt n as received
- if n smallest unACKed pkt, advance window base to next unACKed seq #

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Selective Repeat (3 of 3)

receiver

pkt n in [rcvbase, rcvbase+N-1]

- send ACK(n)
- out-of-order: buffer
- in-order: deliver (also deliver buffered, in-order pkts), advance window to next not-yet-received pkt

pkt n in [rcvbase-N,rcvbase-1]

- ACK(n)

otherwise:

- ignore

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Selective Repeat in Action

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Selective Repeat: Dilemma (1 of 2)

- example:
- seq #'s: 0, 1, 2, 3
- window size=3
- receiver sees no difference in two scenarios!
- duplicate data accepted as new in (b)

(a) no problem

receiver can't see sender side. receiver behavior identical in both cases!
something's (very) wrong!

Pearson
Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Selective Repeat: Dilemma (2 of 2)

Q: what relationship between seq # size and window size to avoid problem in (b)?

(b) oops!

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Learning Objectives (5 of 10)

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

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

TCP: Overview RFCs: 793, 1122, 1323, 2018, 2581

- **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
  - MSS: maximum segment size
- **connection-oriented:**
  - handshaking (exchange of control msgs) inits sender, receiver state before data exchange
- **flow controlled:**
  - sender will not overwhelm receiver

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

TCP Segment Structure

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

TCP Sequence Numbers, ACKs (1 of 2)

sequence numbers:

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

acknowledgements:

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

Q: how receiver handles out-of-order segments

- A: TCP spec doesn't say, - up to implementor

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

TCP Sequence Numbers, ACKs (2 of 2)

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

TCP Round Trip Time, Timeout (1 of 3)

Q: how to set TCP timeout value?

- longer than RTT
  - but RTT varies
- **too short***:* premature timeout, unnecessary retransmissions
- **too long***:* slow reaction to segment loss

Q: how to estimate RTT?

- **`SampleRTT`:** measured time from segment transmission until ACK receipt
  - ignore retransmissions
- **`SampleRTT`** will vary, want estimated RTT "smoother"
  - average several **recent** measurements, not just current **`SampleRTT`**

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

TCP Round Trip Time, Timeout (2 of 3)

EstimatedRTT = (1 – α) * EstimatedRTT + α * SampleRTT

- exponential weighted moving average
- influence of past sample decreases exponentially fast
- typical value: **α =** 0.125

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

TCP Round Trip Time, Timeout (3 of 3)

- **timeout interval: `EstimatedRTT`** plus "safety margin"
  - large variation in **`EstimatedRTT`** → larger safety margin.
- estimate SampleRTT deviation from EstimatedRTT:

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

(typically, $\beta = .25$)

```
TimeoutInterval = EstimatedRTT + 4*DevRTT
```

estimated RTT

"safety margin"

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Learning Objectives (6 of 10)

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

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

TCP Reliable Data Transfer

- TCP creates rdt service on top of IP's unreliable service
  - pipelined segments
  - cumulative acks
  - single retransmission timer
- retransmissions triggered by:
  - timeout events
  - duplicate acks
- let's initially consider simplified TCP sender:
  - ignore duplicate acks
  - ignore flow control, congestion control

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

TCP Sender Events:

data rcvd from app:

- create segment with seq #
- seq # is byte-stream number of first data byte in segment
- start timer if not already running
  - think of timer as for oldest unacked segment
  - expiration interval: **`TimeOutInterval`**

timeout:

- retransmit segment that caused timeout
- restart timer

ack rcvd:

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

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

TCP Sender (Simplified)

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

TCP: Retransmission Scenarios (1 of 2)

lost ACK scenario

premature timeout

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

TCP: Retransmission Scenarios (2 of 2)

cumulative ACK

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

TCP ACK Generation [RFC 1122, RFC 2581]

event at receiver	TCP receiver action
arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
arrival of in-order segment with expected seq #. One other segment has ACK pending	immediately send single cumulative ACK, ACKing both in-order segments
arrival of out-of-order segment higher-than-expect Sequence # . Gap detected	immediately send **duplicate ACK,** indicating Sequence # of next expected byte
arrival of segment that partially or completely fills gap	immediate send ACK, provided that segment starts at lower end of gap

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

TCP Fast Retransmit (1 of 2)

- time-out period often relatively long:
  - long delay before resending lost packet
- detect lost segments via duplicate ACKs.
  - sender often sends many segments back-to-back
  - if segment is lost, there will likely be many duplicate ACKs.

TCP fast retransmit

if sender receives 3 ACKs for same data ("triple duplicate ACKs"), resend unacked segment with smallest seq #

  - likely that unacked segment lost, so don't wait for timeout

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

TCP Fast Retransmit (2 of 2)

fast retransmit after sender receipt of triple duplicate ACK

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Learning Objectives (7 of 10)

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

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

TCP Flow Control (1 of 2)

flow control

receiver controls sender, so sender won't overflow receiver's buffer by transmitting too much, too fast

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

TCP Flow Control (2 of 2)

- receiver "advertises" free buffer space by including **`rwnd`** value in TCP header of receiver-to-sender segments
  - **`RcvBuffer`** size set via socket options (typical default is 4096 bytes)
  - many operating systems autoadjust **`RcvBuffer`**
- sender limits amount of unacked ("in-flight") data to receiver's **`rwnd`** value
- guarantees receive buffer will not overflow

receiver-side buffering

Pearson
Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Learning Objectives (8 of 10)

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

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Connection Management

before exchanging data, sender/receiver “handshake”:

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

```
Socket clientSocket =
  newSocket("hostname","port
   number");
```

```
Socket connectionSocket =
   welcomeSocket.accept();
```

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Agreeing to Establish a Connection (1 of 2)

2-way handshake:

Q: will 2-way handshake always work in network?

- variable delays
- retransmitted messages (example. req_conn(x)) due to message loss
- message reordering
- can't "see" other side

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Agreeing to Establish a Connection (2 of 2)

2-way handshake failure scenarios:

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

TCP3-Way Handshake

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

TCP3-Way Handshake: FSM

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

TCP: Closing a Connection (1 of 2)

- client, server each close their side of connection
  - send TCP segment with FIN bit = 1
- respond to received FIN with ACK
  - on receiving FIN, ACK can be combined with own FIN
- simultaneous FIN exchanges can be handled

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

TCP: Closing a Connection (2 of 2)

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Learning Objectives (9 of 10)

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

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

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!

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Causes/Costs of Congestion: Scenario 1 (1 of 2)

- two senders, two receivers
- one router, infinite buffers
- output link capacity: R
- no retransmission

Pearson
Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Causes/Costs of Congestion: Scenario 1 (2 of 2)

- maximum per-connection throughput: $\frac{R}{2}$

- large delays as arrival rate, λ_{in}, approaches capacity

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Causes/Costs of Congestion: Scenario 2 (1 of 6)

- one router, **finite** buffers
- sender retransmission of timed-out packet
  - application-layer input = application-layer output: $\lambda_{in} = \lambda_{out}$
  - transport-layer input includes **retransmissions** : $\lambda'_{in} \geq \lambda_{in}$

Pearson
Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Causes/Costs of Congestion: Scenario 2 (2 of 6)

idealization: perfect knowledge

- sender sends only when router buffers available

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Causes/Costs of Congestion: Scenario 2 (3 of 6)

- **Idealization: known loss** packets can be lost, dropped at router due to full buffers
- sender only resends if packet **known** to be lost

Pearson
Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Causes/Costs of Congestion: Scenario 2 (4 of 6)

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Causes/Costs of Congestion: Scenario 2 (5 of 6)

Realistic: duplicates

- packets can be lost, dropped at router due to full buffers
- sender times out prematurely, sending **two** copies, both of which are delivered

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Causes/Costs of Congestion: Scenario 2 (6 of 6)

"costs" of congestion:

- more work (retrans) for given "goodput"
- unneeded retransmissions: link carries multiple copies of pkt
  - decreasing goodput

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Causes/Costs of Congestion: Scenario 3 (1 of 2)

- four senders
- multihop paths
- timeout/retransmit

Q: what happens as λ_{in} and λ_{in}' increase **?**

A: as red λ_{in}' increases, all arriving blue pkts at upper queue are dropped, blue throughput → 0

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Causes/Costs of Congestion: Scenario 3 (2 of 2)

another "cost" of congestion:

- when packet dropped, any "upstream transmission capacity used for that packet was wasted!

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Learning Objectives (10 of 10)

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

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

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 MSS every RTT until loss detected
  - **multiplicative decrease**: cut **`cwnd`** in half after loss

AIMD saw tooth behavior: probing for bandwidth

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

TCP Congestion Control: Details

- sender limits transmission:

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

- **cwnd** is dynamic, function of perceived network congestion

TCP sending rate:

- **roughly:** send cwnd bytes, wait RTT for ACKS, then send more bytes

$$\text{rate} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/sec}$$

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

TCP Slow Start

- when connection begins, increase rate exponentially until first loss event:
  - initially **`cwnd`** = 1 MSS
  - double **`cwnd`** every RTT
  - done by incrementing **`cwnd`** for every ACK received
- **summary:** initial rate is slow but ramps up exponentially fast

Pearson
Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

TCP: Detecting, Reacting to Loss

- loss indicated by timeout:
  - **`cwnd`** set to 1 MSS;
  - window then grows exponentially (as in slow start) to threshold, then grows linearly
- loss indicated by 3 duplicate ACKs: TCP RENO
  - dup ACKs indicate network capable of delivering some segments
  - **`cwnd`** is cut in half window then grows linearly
- TCP Tahoe always sets **`cwnd`** to 1 (timeout or 3 duplicate acks

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

TCP: Switching from Slow Start to CA

Q: when should the exponential increase switch to linear?

A: when **`cwnd`** gets to $\frac{1}{2}$ of its value before timeout.

Implementation:

- variable **`ssthresh`**
- on loss event, **`ssthresh`** is set to $\frac{1}{2}$ of **`cwnd`** just before loss event

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Summary: TCP Congestion Control

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

TCP Throughput

- average TCP thruput as function of window size, RTT?
  - ignore slow start, assume always data to send
- W: window size (measured in bytes) where loss occurs
  - average window size (# in-flight bytes) is
  - average thruput is $\frac{3}{4}W$ per RTT $\frac{3}{4}W$

$$\text{avg TCP thruput} = \frac{3}{4}\frac{W}{RTT} \text{ bytes / sec}$$

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

TCP Futures: TCP over "Long, Fat Pipes"

- example: 1500 byte segments, 100ms RTT, want 10 Gbps throughput
- requires W = 83,333 in-flight segments
- throughput in terms of segment loss probability, L [Mathis 1997]:

$$\text{TCP throughput} = \frac{1.22 \cdot \text{MSS}}{\text{RTT}\sqrt{L}}$$

→ to achieve 10 Gbps throughput, need a loss rate of

$L = 2 \cdot 10^{-10}$ **– a very small loss rate!**

- new versions of TCP for high-speed

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

TCP Fairness

fairness goal*:* if K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of $\frac{R}{K}$

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Why is TCP Fair?

two competing sessions:

- additive increase gives slope of 1, as throughout increases
- multiplicative decrease decreases throughput proportionally

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Fairness (More)

Fairness and UDP

- multimedia apps often do not use TCP
  - do not want rate throttled by congestion control
- instead use UDP:
  - send audio/video at constant rate, tolerate packet loss

Fairness, parallel TCP connections

- application can open multiple parallel connections between two hosts
- web browsers do this
- example, link of rate R with 9 existing connections:
  - new app asks for 1 TCP, gets rate $\frac{R}{10}$
  - new app asks for 11 TCPs, gets $\frac{R}{2}$

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Explicit Congestion Notification (ECN)

network-assisted congestion control:

- two bits in IP header (ToS field) marked **by network router** to indicate congestion
- congestion indication carried to receiving host
- receiver (seeing congestion indication in IP datagram)) sets ECE bit on receiver-to-sender ACK segment to notify sender of congestion

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Chapter Summary

- principles behind transport layer services:
  - multiplexing, demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- instantiation, implementation in the Internet
  - UDP
  - TCP

next:

- leaving the network "edge" (application, transport layers)
- into the network "core"
- two network layer chapters:
  - data plane
  - control plane

Pearson

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved

Copyright

This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted. The work and materials from it should never be made available to students except by instructors using the accompanying text in their classes. All recipients of this work are expected to abide by these restrictions and to honor the intended pedagogical purposes and the needs of other instructors who rely on these materials.

Copyright © 2017, 2013, 2010 Pearson Education, Inc. All Rights Reserved