

Computer Networks CS3611

Transport Layer-Part 3

Haiming Jin

The slides are adapted from those provided by Prof. J.F Kurose and K.W. Ross.

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**

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

- point-to-point:
 - one sender, one receiver
- reliable, in-order byte stream
- pipelined:
 - TCP congestion and flow control set window size

- full duplex data:
 - bi-directional data flow in same connection
 - MSS (maximum segment size): the largest amount of data that can be placed in a segment
- connection-oriented:
 - handshaking (exchange of control msgs) inits sender, receiver state before data exchange
- flow controlled:
 - sender will not overwhelm receiver

Transport Layer 3-4

TCP: Overview



TCP segment structure



TCP seq. numbers, ACKs

sequence numbers:

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

acknowledgement numbers:

- seq # of next byte expected from other side
- cumulative ACK : TPC only acknowledges bytes up to the first missing byte



TCP segment structure



TCP round trip time, timeout

- Q: how to set TCP timeout value?
- Ionger 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 (why?)
- SampleRTT will vary, want estimated RTT "smoother"
 - average several recent measurements, not just current SampleRTT

TCP round trip time, timeout

EstimatedRTT = $(1 - \alpha)$ *EstimatedRTT + α *SampleRTT

- exponential weighted moving average
- influence of past sample decreases exponentially fast
- typical value: $\alpha = 0.125$



TCP round trip time, timeout

- timeout interval: EstimatedRTT plus "safety margin"
 - large variation in EstimatedRTT -> larger safety margin
- estimate SampleRTT deviation from EstimatedRTT:

```
DevRTT = (1-\beta) *DevRTT +
\beta*|SampleRTT-EstimatedRTT|
(typically, \beta = 0.25)
```

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

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

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
- flow control, congestion control

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

TCP sender (simplified)



TCP: retransmission scenarios



Transport Layer 3-21

TCP: retransmission scenarios



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 seq #. Gap detected	immediately send <i>duplicate ACK</i> , indicating seq. # 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

TCP fast retransmit

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

TCP fast retransmit

if sender receives triple duplicate ACKs", resend unacked segment with smallest seq #

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

TCP fast retransmit



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

TCP flow control



TCP flow control

- 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



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

Connection Management

before exchanging data, sender/receiver "handshake":

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





Transport Layer 3-30

TCP 3-way handshake



TCP: closing a connection

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

TCP: closing a connection



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

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!

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

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



additively increase window size ...

TCP Congestion Control: details



sender limits transmission:

LastByteSent-LastByteAcked ≤ cwnd

 cwnd is dynamic, function of perceived network congestion TCP sending rate:

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



Summary: TCP Congestion Control



Transport Layer 3-53

TCP Slow Start

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



TCP: detecting, reacting to loss

Ioss indicated by timeout:

- cwnd set to I MSS;
- begins slow start again until **cwnd** reaches threshold, then grows linearly (i.e., enters congestion avoidance)

Ioss indicated by 3 duplicate ACKs: TCP RENO

- dup ACKs indicate network capable of delivering some segments
- cwnd is cut in half window and added in 3 MSS, then enters the fast recovery stage

TCP Tahoe always sets cwnd to 1 (timeout or 3 duplicate acks) and begins slow start again

TCP: switching from slow start to Congestion Avoidance

- Q: when should the exponential increase switch to linear?
- A: when cwnd gets to 1/2 of its value before timeout.

Implementation:

- variable ssthresh
- on loss event, ssthresh is set to 1/2 of cwnd just before loss event



Summary: TCP Congestion Control



Transport Layer 3-58

TCP throughput

- avg. 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
 - avg. window size (# in-flight bytes) is 3/4 W
 - avg. thruput is 3/4W per RTT

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



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



Why is TCP fair?

two competing sessions:

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



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
- e.g., link of rate R with 9 existing connections:
 - new app asks for I TCP, gets rate R/10
 - new app asks for 11 TCPs, gets R/2

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



Chapter 3: 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