CMPE 150/L : Introduction to Computer Networks

Chen Qian

Computer Engineering
UCSC Baskin Engineering

Lecture 10
Midterm exam

- Midterm next Thursday

- Close book but one-side 8.5"x11" note is allowed (must use hand-writing!)

- Let me know by next Monday if you have any problem

- Sample midterm and sample question of Chapter 2&3
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
Agreeing to establish a connection

2-way handshake:

Q: will 2-way handshake always work in network?
   - variable delays
   - retransmitted messages (e.g. req_conn(x)) due to message loss
   - message reordering
   - can’t “see” other side
Agreeing to establish a connection

2-way handshake failure scenarios:

choose x

retransmit req_conn(x)

ESTAB

req_conn(x)

client terminates

connection x completes

half open connection! (no client!)

req_conn(x)

ESTAB

acc_conn(x)

server forgets x

2-way handshake failure scenarios:

choose x

req_conn(x)

ESTAB

req_conn(x)

client terminates

connection x completes

half open connection! (no client!)

req_conn(x)

ESTAB

acc_conn(x)

server forgets x

2-way handshake failure scenarios:

choose x

req_conn(x)

ESTAB

req_conn(x)

client terminates

connection x completes

half open connection! (no client!)

req_conn(x)

ESTAB

accept data(x+1)

accept data(x+1)
TCP 3-way handshake

**client state**

- **LISTEN**
  - choose init seq num, \( x \)
  - send TCP SYN msg

- **SYNSENT**
  - receive SYNACK(\( x \))
  - indicates server is live
  - send ACK for SYNACK
  - this segment may contain client-to-server data

- **ESTAB**

**server state**

- **LISTEN**
  - choose init seq num, \( y \)
  - send TCP SYNACK msg, acking SYN

- **SYN RCVD**
  - SYN bit=1, Seq=\( y \)
  - ACK bit=1; ACK num=\( y+1 \)

- **ESTAB**
  - received ACK(\( y \))
  - indicates client is live
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 (queuing in router buffers)
- a top-10 problem!
Causes/costs of congestion: scenario 1

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

- maximum per-connection throughput: R/2
- large delays as arrival rate, $\lambda_{in}$, approaches capacity

![Diagram](image)
Causes/costs of congestion: scenario 2

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

Diagram:
- Host A
  - $\lambda_{in}$: original data
  - $\lambda_{in}'$: original data, \textit{plus} retransmitted data
- Host B
- Finite shared output link buffers
- Router
  - Finite buffers
- λ in → Router → λ out

Transport Layer 10
Causes/costs of congestion: scenario 2

idealization: perfect knowledge

- sender sends only when router buffers available

\[ \lambda_{\text{in}} : \text{original data} \]
\[ \lambda'_{\text{in}} : \text{original data, plus retransmitted data} \]

\[ \lambda_{\text{out}} \]

free buffer space!

finite shared output link buffers

Host A

Host B
Causes/costs of congestion: scenario 2

Idealization: known loss
packets can be lost, dropped at router due to full buffers

- sender only resends if packet known to be lost
Causes/costs of congestion: scenario 2

**Idealization: known loss**
packets can be lost, dropped at router due to full buffers

- sender only resends if packet known to be lost

\[ \lambda_{\text{in}}: \text{original data} \]
\[ \lambda_{\text{out}}: \text{original data, plus retransmitted data} \]

when sending at R/2, some packets are retransmissions but asymptotic goodput is still R/2 (why?)

\[ \lambda_{\text{in}} \rightarrow R/2 \]
\[ \lambda_{\text{out}} \rightarrow R/2 \]
Causes/costs of congestion: scenario 2

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

When sending at R/2, some packets are retransmissions including duplicated that are delivered!
- **Throughput:**
  - Data rate at the receiver

- **Goodput:**
  - Rate at the receiver for data without duplicate!
Causes/costs of congestion: scenario 2

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

“costs” of congestion:

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

when sending at R/2, some packets are retransmissions including duplicated that are delivered!
Causes/costs of congestion: scenario 3

- four senders
- multihop paths
- timeout/retransmit

Q: what happens as \( \lambda_{in} \) and \( \lambda_{in}' \) increase?
A: as red \( \lambda_{in}' \) increases, all arriving blue pkts at upper queue are dropped, blue throughput \( \to 0 \)
Causes/costs of congestion: scenario 3

another “cost” of congestion:

- when packet dropped, any “upstream transmission capacity used for that packet was wasted!
Approaches towards congestion control

two broad approaches towards congestion control:

end-end congestion control:
- no explicit feedback from network
- congestion inferred from end-system observed loss, delay
- approach taken by TCP

network-assisted congestion control:
- routers provide feedback to end systems
  - single bit indicating congestion (SNA, DECbit, TCP/IP ECN, ATM)
  - explicit rate for sender to send at
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: details

- **sender limits transmission:**
  
  \[
  \text{LastByteSent} - \text{LastByteAcked} \leq \text{cwnd}
  \]

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

**TCP sending rate:**

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

\[
\text{rate} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/sec}
\]
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
TCP: detecting, reacting to loss

- loss indicated by timeout:
  - set a threshold $\text{ssthresh}$ to half of the $\text{cwnd}$;
  - $\text{cwnd}$ set to 1 MSS (by both TCP Tahoe and Reno);
  - window then grows exponentially (as in slow start) to threshold, then grows linearly

- TCP Tahoe always sets $\text{cwnd}$ to 1 (timeout or 3 duplicate acks)

- TCP RENO: loss indicated by 3 duplicate ACKs
  - dup ACKs indicate network capable of delivering some segments
  - $\text{cwnd}$ is cut in half window then grows linearly
After cwnd reaching the threshold

- Congestion avoidance algorithm:

- Additive increase multiplicative decrease (AIMD)
TCP congestion control: AIMD

- **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

![AIMD saw tooth behavior diagram](image-url)
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

- **slow start**
  - $cwnd = 1$ MSS
  - $ssthresh = 64$ KB
  - $dupACKcount = 0$
  - Transmit new segment(s), as allowed

- **congestion avoidance**
  - $cwnd > ssthresh$
  - $\Lambda$
  - $timeout$
  - $ssthresh = cwnd/2$
  - $cwnd = 1$ MSS
  - $dupACKcount = 0$
  - Retransmit missing segment
  - Transmit new segment(s), as allowed

- **fast recovery**
  - $dupACKcount = 3$
  - $ssthresh = cwnd/2$
  - $cwnd = ssthresh + 3$
  - Retransmit missing segment

- **new ACK**
  - $cwnd = cwnd + MSS \cdot (MSS/cwnd)$
  - $dupACKcount = 0$
  - Transmit new segment(s), as allowed

- **new ACK!**
  - $dupACKcount++$

- **timeout**
  - $ssthresh = cwnd/2$
  - $cwnd = 1$ MSS
  - $dupACKcount = 0$
  - Retransmit missing segment

- **retransmit missing segment**
  - $dupACKcount == 3$
  - $ssthresh = cwnd/2$
  - $cwnd = ssthresh + 3$
  - Retransmit missing segment
TCP throughput

- avg. TCP throughput 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 $\frac{3}{4} W$
  - avg. throughput is $\frac{3}{4}W$ per RTT

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

**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 1, as throughout increases
- multiplicative decrease decreases throughput proportionally

![Diagram showing equal bandwidth share and congestion avoidance with additive increase and loss: decrease window by factor of 2.](image)
Van Jacobson

- One of the key designers of TCP congestion control
- [https://www.youtube.com/watch?v=QP4A6L7CEqA](https://www.youtube.com/watch?v=QP4A6L7CEqA)
- 1:40-9:20
**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 1 TCP, gets rate $R/10$
  - new app asks for 11 TCPs, gets $R/2$
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”
Next class

- Midterm covers every slide until here.

- Please read Chapter 4.1-4.2 of your textbook BEFORE Class