CS 268: Lecture 6
(TCP Congestion Control)

Ion Stoica

February 6, 2006

Today's Lecture

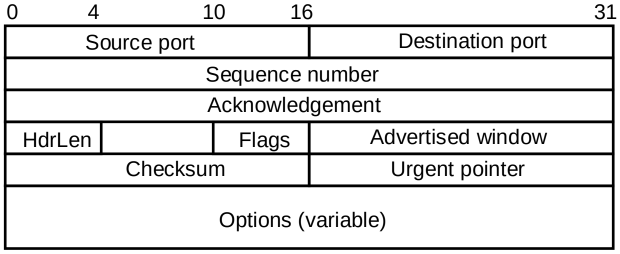
- Basics of Transport
- Basics of Congestion Control
- Comments on Congestion Control

Duties of Transport

- Demultiplexing:
  - IP header points to protocol
  - Transport header needs demultiplex further
    - UDP: port
    - TCP: source and destination address/port
  - Well known ports and ephemeral ports

- Data reliability (if desired):
  - UDP: checksum, but no data recovery
  - TCP: checksum and data recovery

TCP Header



- Sequence number, acknowledgement, and advertised window – used by sliding-window based flow control
- Flags:
  - SYN, FIN – establishing/terminating a TCP connection
  - ACK – set when Acknowledgement field is valid
  - URG – urgent data; Urgent Pointer says where non-urgent data starts
  - PUSH – don’t wait to fill segment
  - RESET – abort connection

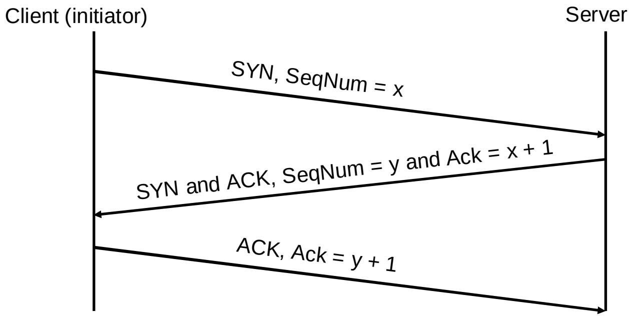
TCP Header (Cont)

- Checksum – 1’s complement and is computed over
  - TCP header
  - TCP data
  - Pseudo-header (from IP header)
    - Note: breaks the layering!

<table>
<tr><td colspan="3">Source address</td></tr>
<tr><td colspan="3">Destination address</td></tr>
<tr><td>0</td><td>Protocol (TCP)</td><td>TCP Segment length</td></tr>
</table>

TCP Connection Establishment

- Three-way handshake
  - Goal: agree on a set of parameters: the start sequence number for each side



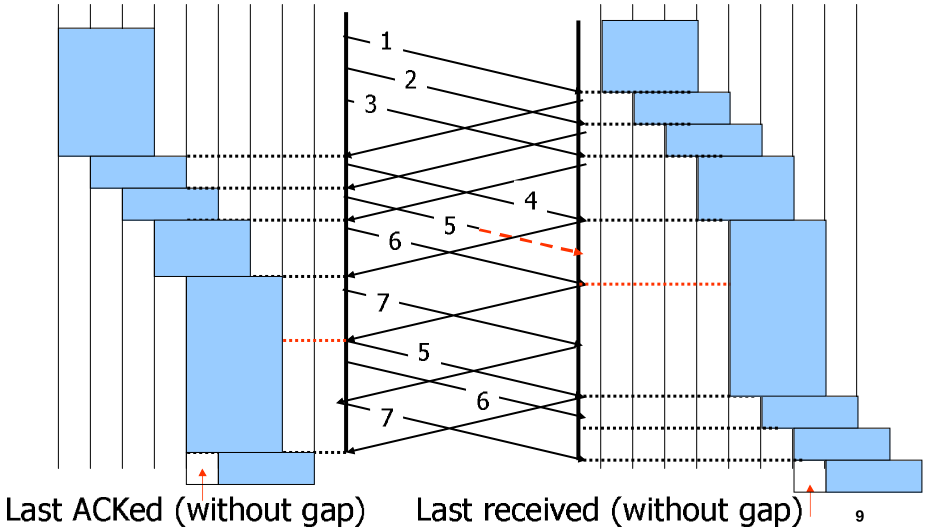
TCP Issues

- Connection confusion:
  - ISNs can't always be the same
- Source spoofing:
  - Need to make sure ISNs are random
- SYN floods:
  - SYN cookies
- State management with many connections
  - Server-stateless TCP (NSDI 05)

TCP Flow Control

- Make sure receiving end can handle data
- Negotiated end-to-end, with no regard to network
- Ends must ensure that no more than W packets are in flight
  - Receiver ACKs packets
  - When sender gets an ACK, it knows packet has arrived

Sliding Window



Observations

- Throughput is ~ (w/RTT)
- Sender has to buffer all unacknowledged packets, because they may require retransmission
- Receiver may be able to accept out-of-order packets, but only up to its buffer limits

What Should the Receiver ACK?

1. ACK every packet, giving its sequence number
2. Use *negative ACKs* (NACKs), indicating which packet did not arrive
3. Use *cumulative ACK*, where an ACK for number n implies ACKS for all $k < n$
4. Use *selective ACKs* (SACKs), indicating those that did arrive, even if not in order

Error Recovery

- Must retransmit packets that were dropped
- To do this efficiently
  - Keep transmitting whenever possible
  - Detect dropped packets and retransmit quickly
- Requires:
  - Timeouts (with good timers)
  - Other hints that packet were dropped

Timer Algorithm

- Use exponential averaging:

A(n) = b*A(n- 1) + (1 – b)T(n)
D(n) = b*D(n-1) + (1 – b)*(T(n) – A(n))
Timeout(n) = A(n) +4D(n)

Question: Why not set timeout to average delay?

Notes:

1. Measure T(n) only for original transmissions
2. Double Timeout after timeout …
3. Reset Timeout for new packet and when receive ACK

Hints

- When should I suspect a packet was dropped?

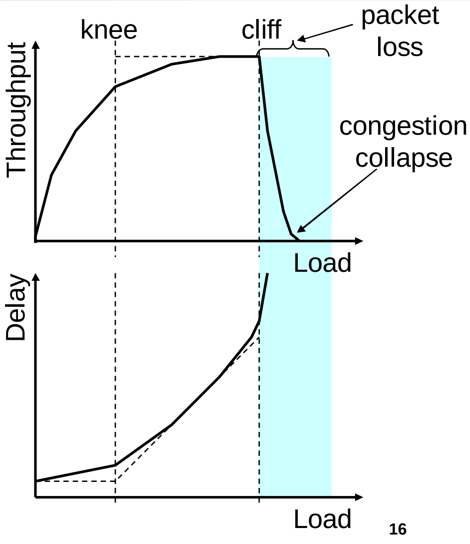
- When I receive several duplicate ACKs
  - Receiver sends an ACK whenever a packet arrives
  - ACK indicates seq. no. of last received *consecutively* received packet
  - Duplicate ACKs indicates missing packet

TCP Congestion Control

- Can the network handle the rate of data?
- Determined end-to-end, but TCP is making guesses about the state of the network
- Two papers:
  - Good science vs great engineering

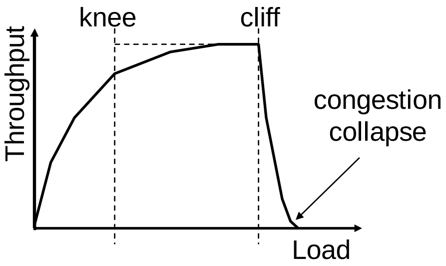
Dangers of Increasing Load

- Knee – point after which
  - Throughput increases very slow
  - Delay increases fast
- Cliff – point after which
  - Throughput starts to decrease very fast to zero (congestion collapse)
  - Delay approaches infinity
- In an M/M/1 queue
  - Delay = 1/(1 – utilization)

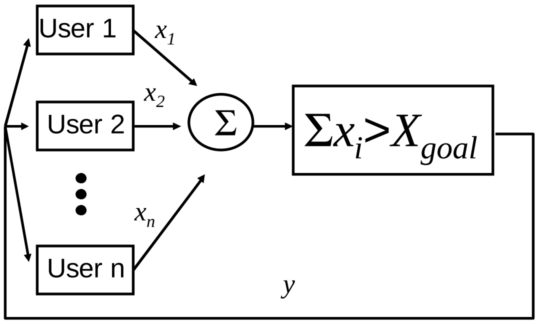


Cong. Control vs. Cong. Avoidance

- Congestion control goal
  - Stay left of cliff
- Congestion avoidance goal
  - Stay left of knee



Control System Model [CJ89]



- Simple, yet powerful model
- Explicit binary signal of congestion

Possible Choices

$$x_i(t+1) = \begin{Bmatrix} a_I + b_I x_i(t) \ \ increase \\ a_D + b_D x_i(t) \, decrease \end{Bmatrix}$$

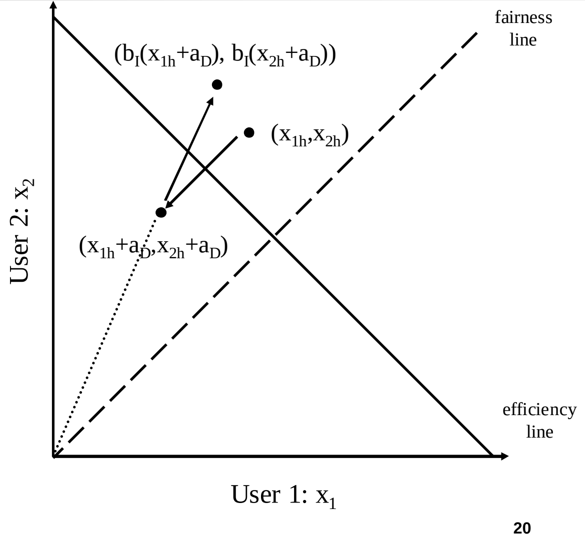
- Multiplicative increase, additive decrease
  - $a_I=0$, $b_I>1$, $a_D<0$, $b_D=1$
- Additive increase, additive decrease
  - $a_I>0$, $b_I=1$, $a_D<0$, $b_D=1$
- Multiplicative increase, multiplicative decrease
  - $a_I=0$, $b_I>1$, $a_D=0$, $0<b_D<1$
- Additive increase, multiplicative decrease
  - $a_I>0$, $b_I=1$, $a_D=0$, $0<b_D<1$
- Which one?

Multiplicative Increase, Additive Decrease

- Fixed point at

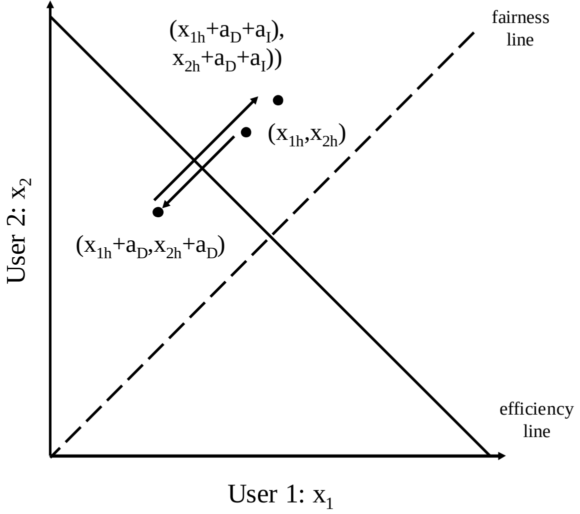
$$x_{1h} = x_{2h} = \frac{b_I a_D}{1-b_I}$$

Fixed point is unstable!



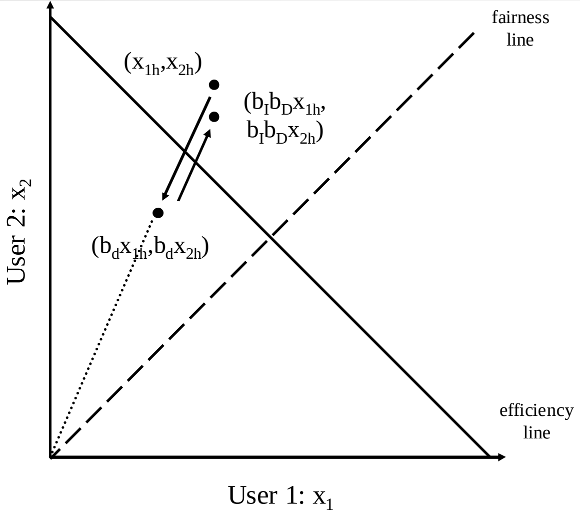
Additive Increase, Additive Decrease

- Reaches stable cycle, but does not converge to fairness



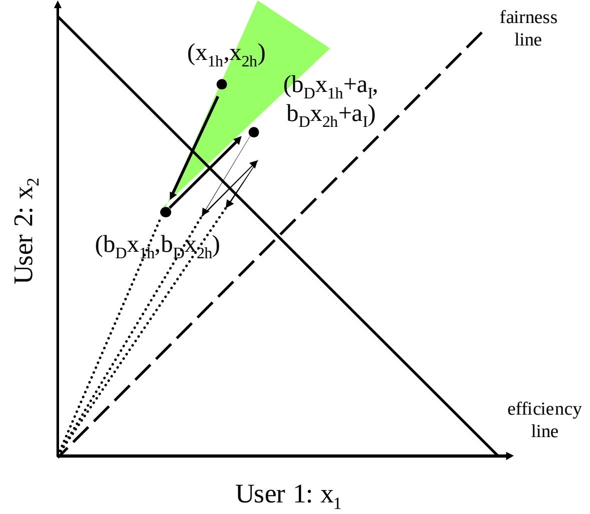
Multiplicative Increase, Multiplicative Decrease

- Converges to stable cycle, but is not fair



Additive Increase, Multiplicative Decrease

- Converges to stable and fair cycle



Modeling

- Critical to understanding complex systems
  - [CJ89] model relevant after 15 years, 10^6 increase of bandwidth, 1000x increase in number of users
- Criteria for good models
  - Two conflicting goals: reality and simplicity
  - Realistic, complex model → too hard to understand, too limited in applicability
  - Unrealistic, simple model → can be misleading

TCP Congestion Control

- [CJ89] provides theoretical basis for basic congestion avoidance mechanism
- Must turn this into real protocol

TCP Congestion Control

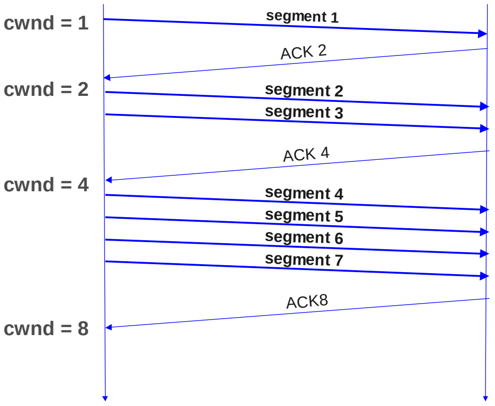
- Maintains three variables:
  - cwnd: congestion window
  - flow_win: flow window; receiver advertised window
  - Ssthresh: threshold size (used to update cwnd)
  -
- For sending, use: win = **min**(flow_win, cwnd)

TCP: Slow Start

- Goal: reach knee quickly
- Upon starting (or restarting):
  - Set *cwnd* =1
  - Each time a segment is acknowledged increment *cwnd* by one (*cwnd*++).
- Slow Start is not actually slow
  - *cwnd* increases exponentially

Slow Start Example

- The congestion window size grows very rapidly
- TCP slows down the increase of *cwnd* when ***cwnd* >= *ssthresh***

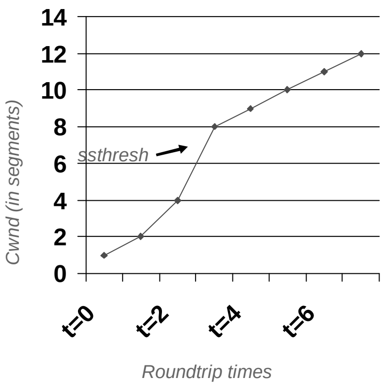


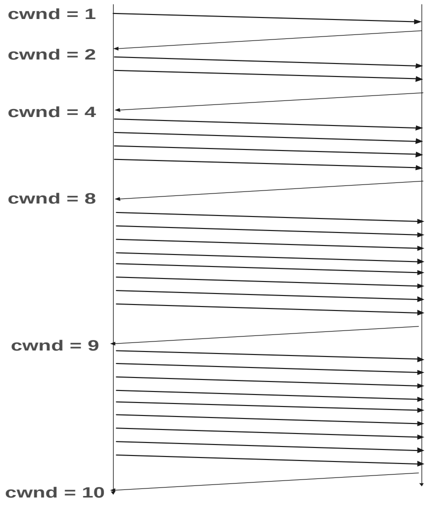
Congestion Avoidance

- Slow down “Slow Start”
- *ssthresh* is lower-bound guess about location of knee
- **If** *cwnd > ssthresh* **then**
  each time a segment is acknowledged
  increment *cwnd* by *1/cwnd* (*cwnd += 1/cwnd*).
- So *cwnd* is increased by one only if all segments have been acknowledged.

Slow Start/Congestion Avoidance Example

- Assume that
  ssthresh = 8



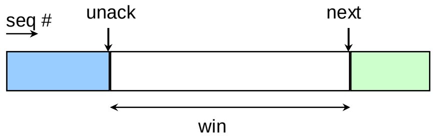


Putting Everything Together: TCP Pseudocode

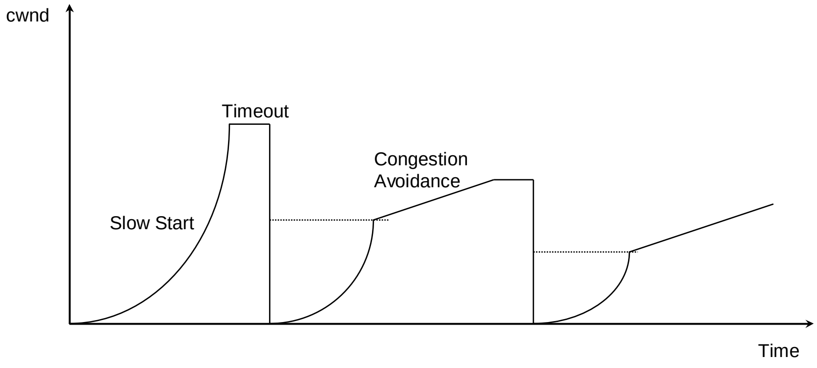
```
Initially:
    cwnd = 1;
    ssthresh = infinite;
New ack received:
    if (cwnd < ssthresh)
        /* Slow Start*/
        cwnd = cwnd + 1;
    else
        /* Congestion Avoidance */
        cwnd = cwnd + 1/cwnd;
Timeout:
    /* Multiplicative decrease */
    ssthresh = cwnd/2;
    cwnd = 1;
```

```
while (next < unack + win)
    transmit next packet;

where   win = min(cwnd,
                  flow_win);
```

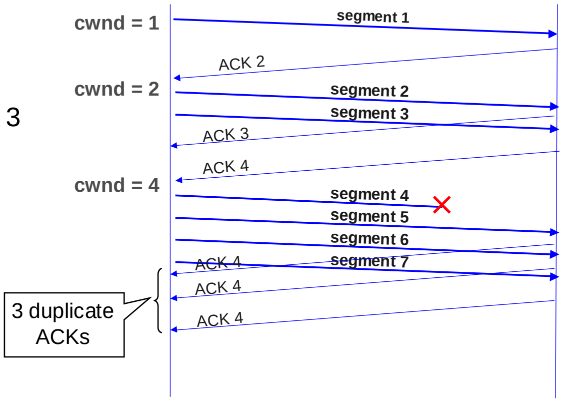


The big picture



Fast Retransmit

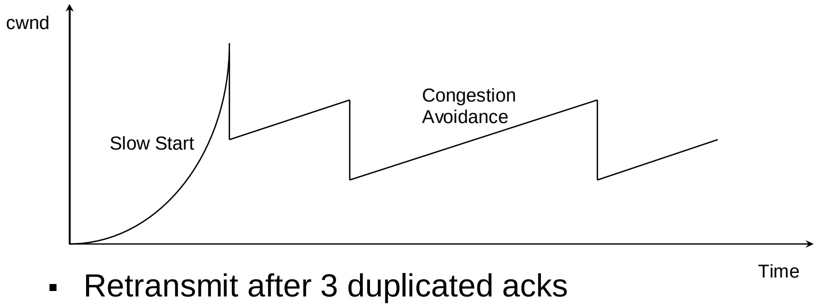
- Don't wait for window to drain
- Resend a segment after 3 duplicate ACKs



Fast Recovery

- After a fast-retransmit set *cwnd* to *ssthresh/2*
  - i.e., don't reset *cwnd* to 1
- But when RTO expires still do *cwnd* = 1
- Fast Retransmit and Fast Recovery
  - Implemented by TCP Reno
  - Most widely used version of TCP today
- Lesson: avoid RTOs at all costs!

Fast Retransmit and Fast Recovery



- Retransmit after 3 duplicated acks
  - prevent expensive timeouts
- No need to slow start again
- At steady state, *cwnd* oscillates around the optimal window size.

Engineering vs Science in CC

- Great engineering built useful protocol:
  - TCP Reno, etc.
- Good science by CJ and others
  - Basis for understanding why it works so well

Behavior of TCP

- Are packets smoothly paced?
  - NO! Ack-compression
- Are long-lived flows nicely interleaved?
  - NO!
- How does throughput depend on drop rate?

Tput ~ 1/sqrt(d)

Extensions to TCP

- Selective acknowledgements: TCP SACK
- Explicit congestion notification: ECN
- Delay-based congestion avoidance: TCP Vegas
- Discriminating between congestion losses and other losses: cross-layer signaling and guesses
- Randomized drops (RED) and other router mechanisms

Issues with TCP

- Fairness:
  - Throughput depends on RTT
- High speeds:
  - to reach 10gbps, packet losses occur every 90 minutes!
- Short flows:
  - How to set initial cwnd properly
- What about flows that want congestion control, but don't want reliable delivery?

TCP: Cooperation and Compatibility

- TCP assumes all flows employ TCP-like congestion control
  - TCP-friendly or TCP-compatible
- Selfish flows: can get all the bandwidth they like
- If new congestion control algorithms are developed, they must be TCP-friendly