EE 122: Congestion Control and Avoidance

Kevin Lai
October 23, 2002

Problem

- At what rate do you send data?
- Flow control
  - make sure that the receiver can receive
  - sliding-window based flow control:
    - receiver reports window size to sender
    - higher window → higher throughput
    - throughput = wnd/RTT
- Congestion control
  - make sure that the network can deliver

Solutions

- Everyone sends as fast as they can
  - excess gets dropped
  - very bad idea: leads to congestion collapse
- Reserve bandwidth
  - low utilization
  - never have packet drops, queueing delays
- Congestion control and avoidance
  - high utilization
  - occasionally have packet drops, queueing delays

Non-decreasing Efficiency under Load

- Efficiency = useful_work/time
- Critical property of system design
  - network technology, protocol or application
- otherwise, system collapses exactly when most demand for its operation
- trade lower overall efficiency for this?
Congestion Collapse

- Decrease of network efficiency under load
  
  - efficiency = utilization of bandwidth

- Waste resources on useless or undelivered data
  
- Network layer
    
    - load → drops, ≥1 fragment dropped

- Transport layer
  
  - retransmit too many times
  
  - no congestion control / avoidance

Transport Layer

Congestion Collapse 1

- network is congested (a router drops packets)
  
  - the receiver didn’t get a packet
  
  - sender waits, but not long enough
    
    - timeout too short,

  - retransmit again
    
    - adds to congestion, increases likelihood that retransmission will be dropped, or delayed

  - rinse, repeat

  - assume that everyone is doing the same

  - network will become more and more congested

  - with duplicate packets rather than new packets

  - Solution: fix timeout (previous lecture)

Congestion Collapse and Efficiency

- knee – point after which
  
  - throughput increases slowly

  - delay increases quickly

- cliff – point after which
  
  - throughput decreases quickly to zero (congestion collapse)

  - delay goes to infinity

- Congestion avoidance
  
  - stay at knee

- Congestion control
  
  - stay left of (but usually close to) cliff
TCP Congestion Control

- Operate to the left of the cliff
  - less efficient than operating at the knee, but requires less support from network
- Solution: Carefully manage number of packets in network
  - starting up: increase number of packets allowed
  - equilibrium: maintain constant number of packets
  - must probe periodically for more available bandwidth
  - congestion: decrease number of packets allowed

TCP Congestion Control Components

- Measure available bandwidth
  - slow start
    - fast, hard on network
    - additive increase/multiplicative decrease
    - slow, gentle on network
- Detecting congestion
  - timeout based on RTT (last lecture)
    - robust, causes low throughput
    - duplicate acks (Fast Retransmit)
    - can be fooled, maintains high throughput
- Recovering from loss
  - Fast recovery: avoid slow start when few packets lost

Congestion Window (cwnd)

- Limits how much data can be in transit
- Integrate with flow control
- implemented as # of bytes, describe as # packets

\[
\text{MaxWindow} = \min(\text{cwnd}, \text{AdvertisedWindow})
\]

\[
\text{EffectiveWindow} = \text{MaxWindow} - (\text{LastByteSent} - \text{LastByteAcked})
\]

Slow Start Goal

- Goal: discover congestion quickly
  - used to get rough initial estimate of cwnd
- Slow Start used when
  - a new connection, or
  - after timeout, or
  - after connection has been idle for a while
Slow Start Algorithm

- Algorithm:
  - Set $cwnd = 1$
  - Each time a segment is acknowledged increment $cwnd$ by one ($cwnd++$)
- Slow Start increases $cwnd$ exponentially
  - called “Slow” because it gradually increases sending rate according to the RTT
  - predecessor just increased to advWin immediately

Slow Start Example

- $cwnd$ doubles every RTT

Slow Start Problem

- Slow Start can cause serious loss
  - loss = bottleneck bandwidth * RTT
- Example:
  - bottleneck link is 8 Mb/s
  - RTT is 100ms
  - at some point $cwnd = 800,000$ bits
    - exactly right for bottleneck
    - 100 ms late, $cwnd = 1,600,000$ bits
    - twice the bottleneck, 800,000 bits lost (oops)
- Need to switch to something else when throughput is close to bottleneck bandwidth

Exiting Slow Start

- $cwnd >= advWin$
  - limited by receiver’s advertised window → if $cwnd$ allowed to increase, would grow unbounded
- congestion
  - reached available bandwidth → switch to AIMD
- $cwnd >= ssthresh$
  - $ssthresh$: slow start threshold
  - $ssthresh$ is a hint about when to slow down, calculated from previous congestion on same connection
  - “we experienced congestion beyond this point before, slow down”
Additive Increase/ Multiplicative Decrease

- Used Slow Start to get rough estimate of cwnd
- Goal: maintain operating point at the left of the cliff
  - Additive increase: starting from the rough estimate, slowly increase cwnd to probe for additional available bandwidth
  - Multiplicative decrease: cut congestion window size aggressively if congestion occurs
- Why not AI/AD, MI/MD, MI/AD:
  - mathematical models show A/M is only choice that converges to fairness and efficiency

AI/MD Algorithm

- ssthresh = \infty
- a segment is acknowledged
  - increment cwnd by 1/cwnd (cwnd += 1/cwnd)
  - as a result, cwnd is increased by one only if all segments in a cwnd have been acknowledged.
- congestion detected
  - timeout: ssthresh = cwnd/2; cwnd = 1; begin slow start
  - duplicate acks (Fast Retransmit): later

Slow Start/Al/MD Example

- Assume that ssthresh = 8

The big picture
Slow Start/Al/MD 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 = win/2;
  cwnd = 1;

AI/MD Problem

- If available bandwidth increases suddenly
  - e.g., other flows finish, route changes
  - then AI/MD will be slow to use it
- Without help from network, there is a fundamental tradeoff between
  - being able to take available bandwidth and
  - causing loss

Congestion Detection

- Wait for Retransmission Time Out (RTO)
  - RTO kills throughput
- In BSD TCP implementations, RTO is usually more than 500ms
  - the granularity of RTT estimate is 500 ms
  - retransmission timeout is RTT + 4 * mean_deviation
- Solution: Don’t wait for RTO to expire

Fast Retransmit

- Resend a segment after 3 duplicate ACKs
  - a duplicate ACK means that an out-of-sequence segment was received
- Notes:
  - packet reordering can cause duplicate ACKs
  - window may be too small to get enough duplicate ACKs
Fast Recovery:
After a Fast Retransmit

- $ssthresh = cwnd / 2$
- $cwnd = ssthresh$
- Instead of setting $cwnd$ to 1, cut $cwnd$ in half (multiplicative decrease)
- For each dup ack arrival
  - $dupack++$
  - $MaxWindow = min(cwnd + dupack, AdvWin)$
  - Indicates packet left network, so we may be able to send more
- Receive ack for new data (beyond initial dup ack)
  - $dupack = 0$
  - Exit fast recovery
- But when RTO expires still do $cwnd = 1$

Fast Recovery Problem

- Can’t recover when more than half the window size is lost
  - $cwnd$ is cut in half
  - $MaxWindow$ is only increased when dup ack is received
  - If more than half the window is lost, less than $cwnd/2$ dup acks will be received by sender
  - Sender will not be able to send anything, must wait for timeout
- More than half the window lost indicates serious congestion anyway

Fast Retransmit and Fast Recovery

- Retransmit after 3 duplicated acks
- Prevent expensive timeouts
- Reduce slow starts
- At steady state, $cwnd$ oscillates around the optimal window size

Other TCP-related techniques

- TCP SACK
  - Instead of cumulative acknowledgements, receiver tells sender exactly what was lost
  - Useful when more than one packet lost in a window
- Router support
  - Have higher utilization, more fairness with router support
## TCP Congestion Control Summary

- **Measure available bandwidth**
  - slow start
    - fast, hard on network
    - additive increase/multiplicative decrease
  - slow, gentle on network

- **Detecting congestion**
  - timeout based on RTT
    - robust, causes low throughput
    - Fast Retransmit
      - can be fooled, maintains high throughput

- **Recovering from loss**
  - Fast recovery: avoid timeout when few packets lost