CIS 553: Networked Systems

Congestion control

March 22, 2021
Agenda

- Transmission Control Protocol
  - Reliable in-order delivery
  - Connection-oriented
  - Flow control
  - Congestion control
TCP State Diagram
Agenda

- Transmission Control Protocol
  - Reliable in-order delivery
  - Connection-oriented
  - Flow control
  - Congestion control
  - Fairness
  - ACK Clocking
Congestion Collapse

- Easily leads to congestion collapse
  - Senders retransmit the lost packets
  - Leading to even greater load
  - ... and even more packet loss

Increase in load that results in a *decrease* in useful work done.
Van Jacobson’s solution

- Reuse the sliding window
  - Controls number of packets in flight
- Sending rate $\sim \text{Window}/\text{RTT}$
- Vary window size to control sending rate
Windows to keep in mind

- Congestion Window: **CWND**
  - Bytes that can be sent without overflowing routers
  - Computed by sender using congestion control algo.

- Flow control window: **RWND**
  - Bytes that can be sent without overflowing receiver
  - Determined by the receiver and reported to the sender

- Sender-side window = \( \min \{ \text{CWND}, \text{RWND} \} \)
  - Assume for remainder of lecture that \( \text{RWND} \gg \text{CWND} \)
Note

- This lecture talks about CWND in units of **MSS**
  - **MSS** (Maximum Segment Size): the amount of payload data in a TCP packet
  - This is only for the simplicity of presentation

- Real implementations maintain CWND in bytes
A complex problem!

Congestion control is a resource allocation problem involving many flows, many links, and complicated global dynamics.
Two Basic Questions

- How does the end host detect congestion?
Detecting congestion

- **Packet delays**
  - Tricky: noisy signal (delay often varies considerably)

- **Routers tell end hosts when they’re congested**

- **Packet loss**
  - Fail-safe signal that TCP already has to detect
  - Complications:
    - Non-congestive loss (e.g., checksum errors)
    - Different types of loss
Different types of packet loss

- **Duplicate ACKs: isolated loss**
  - Still getting ACKs
  - How to tell difference between loss and reordering?
    - Triple duplicate acks

- **Timeout: much more serious**
  - Must have suffered several losses

- **Will adjust rate differently for each case**
  - Assume duplicate ACKs for now
Two Basic Questions

- How does the end host detect congestion?
- How fast should the end host send?
  - How can it discover available bandwidth?
  - How should it adjust to changes in available bandwidth?
  - How should it share with other flows?
Responding to Congestion

- Upon detecting congestion
  - Decrease the sending rate

- But, what if conditions change?
  - If more bandwidth becomes available,
  - ... unfortunate to keep sending at a low rate

- Upon *not* detecting congestion
  - Increase sending rate, a little at a time
  - See if packets get through
TCP Rate adjustment

- Basic structure
  - Upon receipt of ACK (of new data): increase rate
  - Upon detection of loss: decrease rate

- How we increase/decrease the rate depends on the phase of congestion control we’re in:
  - Discovering available bottleneck bandwidth vs.
  - Adjusting to bandwidth variations
Bandwidth discovery with “Slow Start”

- **Goal**: estimate available bandwidth
  - Start slow (for *safety*)
  - Ramp up quickly (for *efficiency*)

- **Consider**
  - $\text{RTT} = 100\text{ms}$, $\text{MSS} = 1000\text{bytes}$
  - Window size to fill 1Mbps of BW = 12.5 packets
  - Window size to fill 1Gbps = 12,500 packets
  - Either is possible!
**Slow Start phase**

- Sender starts at a slow rate, but **increases exponentially** until first loss
- Start with a small congestion window
  - Initially, CWND = 1
  - So, initial sending rate is MSS/RTT
- Double the CWND for each RTT with no loss
Slow Start in action

- For each RTT: double CWND
  - i.e., for each ACK, CWND += 1

Linear increase per $\text{ACK}(\text{CWND}+1)$ ➔ exponential increase per RTT $(2\times\text{CWND})$
Slow Start in action

- For each RTT: double CWND
  - i.e., for each ACK, CWND += 1
When does Slow Start stop?

- Slow Start gives an estimate of available bandwidth
  - At some point, there will be loss
- Introduce a “slow start threshold” (\textit{ssthresh})
  - Initialized to a large value
- If CWND > ssthresh, stop Slow Start
Adjusting to varying bandwidth

- CWND > ssthresh
  - Stop rapid growth and focus on maintenance
- Now, want to track variations in this available bandwidth, oscillating around its current value
  - Repeated probing (rate increase) and backoff (decrease)
Additive increase
- For each ACK, CWND = CWND + $\frac{1}{\text{CWND}}$
- CWND is increased by one only if all segments in a CWND have been acknowledged

Multiplicativc decrease
- On packet loss, CWND = CWND/2
Leads to the TCP “Sawtooth”

Window

Loss

Exponential “slow start”
Why AIMD?

- Every RTT, we can do
  - Multiplicative increase or decrease: CWND → a*CWND
  - Additive increase or decrease: CWND → CWND + b

- Four alternatives:
  - AIAD: gentle increase, gentle decrease
  - AIMD: gentle increase, drastic decrease
  - MIAD: drastic increase, gentle decrease
  - MIMD: drastic increase and decrease
Simple model of congestion control

- Two users
  - rates $x_1$ and $x_2$

- Congestion when $x_1 + x_2 > 1$

- Unused capacity when $x_1 + x_2 < 1$

- Fair when $x_1 = x_2$

![Graph showing congestion control model]

- Efficiency line ($x_1 + x_2 = 1$)
- Fairness line ($x_1 = x_2$)

User 1’s rate ($x_1$) vs. User 2’s rate ($x_2$) graph with axes labeled from 0 to 1.

University of Pennsylvania
Example

Inefficient: $x_1 + x_2 = 0.7$

Efficient: $x_1 + x_2 = 1$

Fair: $(0.5, 0.5)$

Congested: $x_1 + x_2 = 1.2$

$(0.7, 0.5)$

$(0.7, 0.3)$

Efficient: $x_1 + x_2 = 1$

Not fair: $(0.7, 0.3)$

Efficiency line

Fairness line

University of Pennsylvania
AIAD

- Increase: \( x + a_I \)
- Decrease: \( x - a_D \)
- Does not converge to fairness
AIAD Sharing Dynamics

\[
\begin{align*}
&\text{A} \quad \xrightarrow{x_1} \quad \text{D} \\
&\text{D} \quad \xrightarrow{x_2} \quad \text{B} \\
&\text{B} \quad \xrightarrow{} \quad \text{E}
\end{align*}
\]
MIMD

- Increase: $x^* b_I$
- Decrease: $x^* b_D$
- Does not converge to fairness
AIMD

- Increase: $x + a_I$
- Decrease: $x^* b_D$
- Converges to fairness

User 1: $x_1$
User 2: $x_2$

(b_D x_1 + a_I, b_D x_2 + a_I)

Fairness line

Efficiency line

Inefficient congested
AIMD Sharing Dynamics

A

D

B

E

$X_1$

$X_2$

50 packets/sec

Rates equalize $\rightarrow$ fair share
MIAD

- Increase: $x^*b_I$
- Decrease: $x - a_D$
- Does not converge to fairness
- Does not converge to efficiency

"Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks"
-- Chiu and Jain