Overview

• Congestion sources and collapse
• Congestion control basics
• TCP congestion control
• TCP interactions

Congestion Control

• Congestion control basics
• TCP congestion control
• Assigned reading
  • [JK88] Congestion Avoidance and Control
  • [CJ89] Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks

Congestion

• Different sources compete for resources inside network
• Why is it a problem?
  • Sources are unaware of current state of resource
  • Sources are unaware of each other
  • In many situations will result in < 1.5 Mbps of throughput (congestion collapse)
Congestion Collapse

- Definition: *Increase in network load results in decrease of useful work done*
- Many possible causes
  - Spurious retransmissions of packets still in flight
    - Classical congestion collapse
    - How can this happen with packet conservation
    - Solution: better timers and TCP congestion control
  - Undelivered packets
    - Packets consume resources and are dropped elsewhere in network
    - Solution: congestion control for ALL traffic

Other Congestion Collapse Causes

- Fragments
  - Mismatch of transmission and retransmission units
  - Solutions
    - Make network drop all fragments of a packet (early packet discard in ATM)
    - Do path MTU discovery
- Control traffic
  - Large percentage of traffic is for control
    - Headers, routing messages, DNS, etc.
- Stale or unwanted packets
  - Packets that are delayed on long queues
  - "Push" data that is never used

Where to Prevent Collapse?

- Can end hosts prevent problem?
  - Yes, but must trust end hosts to do right thing
    - E.g., sending host must adjust amount of data it puts in the network based on detected congestion
- Can routers prevent collapse?
  - No, not all forms of collapse
    - Doesn’t mean they can’t help
    - Sending accurate congestion signals
    - Isolating well-behaved from ill-behaved sources

Congestion Control and Avoidance

- A mechanism which:
  - Uses network resources efficiently
  - Preserves fair network resource allocation
  - Prevents or avoids collapse
- Congestion collapse is not just a theory
  - Has been frequently observed in many networks
Congestion Control vs. Avoidance

- Avoidance keeps the system performing at the knee
- Control kicks in once the system has reached a congested state

Load

Throughput

Delay

Load

Overview

- Congestion sources and collapse
- Congestion control basics
- TCP congestion control
- TCP interactions

Objectives

- Simple router behavior
- Distributedness
- Efficiency: $X_{\text{knee}} = \sum x_i(t)$
- Fairness: $(\Sigma x_i)^2/n(\Sigma x^2)$
- Power: (throughput$^\alpha$/delay)
- Convergence: control system must be stable

Basic Control Model

- Let’s assume window-based control
- Reduce window when congestion is perceived
  - How is congestion signaled?
    - Either mark or drop packets
  - When is a router congested?
    - Drop tail queues – when queue is full
    - Average queue length – at some threshold
- Increase window otherwise
  - Probe for available bandwidth – how?
Linear Control

- Many different possibilities for reaction to congestion and probing
  - Examine simple linear controls
  - $\text{Window}(t + 1) = a + b \text{Window}(t)$
  - Different $a_i/b_i$ for increase and $a_d/b_d$ for decrease
- Supports various reaction to signals
  - Increase/decrease additively
  - Increased/decrease multiplicatively
- Which of the four combinations is optimal?

Phase plots

- Simple way to visualize behavior of competing connections over time

Phase plots

- What are desirable properties?
- What if flows are not equal?

Additive Increase/Decrease

- Both $X_1$ and $X_2$ increase/decrease by the same amount over time
  - Additive increase improves fairness and additive decrease reduces fairness
Multiplicative Increase/Decrease

- Both $X_1$ and $X_2$ increase by the same factor over time
- Extension from origin – constant fairness

Convergence to Efficiency

Distributed Convergence to Efficiency

Convergence to Fairness
Convergence to Efficiency & Fairness

Increase

Constraints

• Distributed efficiency
  • I.e., $\Sigma \text{Window}(t+1) > \Sigma \text{Window}(t)$ during increase
    • $a_i > 0$ & $b_i = 1$
    • Similarly, $a_d < 0$ & $b_d = 1$
  • Must never decrease fairness
    • $a$ & $b$'s must be $= 0$
    • $a_i/b_i > 0$ and $a_d/b_d = 0$
• Full constraints
  • $a_d = 0$, $0 = b_d < 1$, $a_i > 0$ and $b_i = 1$

What is the Right Choice?

• Constraints limit us to AIMD
  • Can have multiplicative term in increase (MAIMD)
  • AIMD moves towards optimal point
Overview

• Congestion sources and collapse

• Congestion control basics

• TCP congestion control

• TCP interactions

TCP Congestion Control

• Motivated by ARPANET congestion collapse

• Underlying design principle: packet conservation
  • At equilibrium, inject packet into network only when one is removed
  • Basis for stability of physical systems

• Why was this not working?
  • Connection doesn’t reach equilibrium
  • Spurious retransmissions
  • Resource limitations prevent equilibrium

TCP Congestion Control - Solutions

• Reaching equilibrium
  • Slow start
  • Eliminates spurious retransmissions
    • Accurate RTO estimation
    • Fast retransmit
  • Adapting to resource availability
    • Congestion avoidance

TCP Congestion Control Basics

• Keep a congestion window, cwnd
  • Denotes how much network is able to absorb

• Sender’s maximum window:
  • Min (advertised window, cwnd)

• Sender’s actual window:
  • Max window - unacknowledged segments

• If we have large actual window, should we send data in one shot?
  • No, use acks to clock sending new data
**Self-clocking**

- **Sender** $P_r$  $A_s$
- **Receiver** $P_b$  $A_r$

**Slow Start**

- **How do we get this clocking behavior to start?**
  - Initialize $cwnd = 1$
  - Upon receipt of every ack, $cwnd = cwnd + 1$
- **Implications**
  - Window actually increases to $W$ in $RTT \cdot \log_2(W)$
  - Can overshoot window and cause packet loss

**Slow Start Example**

<table>
<thead>
<tr>
<th>Sequence No</th>
<th>Time</th>
<th>Packet</th>
</tr>
</thead>
<tbody>
<tr>
<td>0R</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1R</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>2R</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>3R</td>
<td>15</td>
<td>16</td>
</tr>
</tbody>
</table>

**Slow Start Sequence Plot**

- **Sequence No**
- **Time**
### Congestion Avoidance

- Loss implies congestion – why?
  - Not necessarily true on all link types
- If loss occurs when \( cwnd = W \)
  - Network can handle \( 0.5W \sim W \) segments
  - Set \( cwnd \) to \( 0.5W \) (multiplicative decrease)
- Upon receiving ACK
  - Increase \( cwnd \) by \( 1/cwnd \)
  - Results in additive increase

### Return to Slow Start

- If packet is lost we may lose our self clocking as well
  - Need to implement slow-start and congestion avoidance together
- When timeout occurs set \( ssthresh \) to \( 0.5w \)
  - If \( cwnd < ssthresh \), use slow start
  - Else use congestion avoidance
Overall TCP Behavior

Overview
- Congestion sources and collapse
- Congestion control basics
- TCP congestion control
- TCP interactions

How to Change Window
- When a loss occurs have $W$ packets outstanding
- New $cwnd = 0.5 \times cwnd$
  - How to get to new state?

Fast Recovery
- Each duplicate ack notifies sender that single packet has cleared network
- When $< cwnd$ packets are outstanding
  - Allow new packets out with each new duplicate acknowledgement
- Behavior
  - Sender is idle for some time – waiting for $\frac{1}{2} cwnd$ worth of dupacks
  - Transmits at original rate after wait
    - Ack clocking rate is same as before loss
Fast Recovery

- Sent for each dupack after \( W/2 \) dupacks arrive

NewReno Changes

- Send a new packet out for each pair of dupacks
  - Adapt more gradually to new window
- Will not halve congestion window again until recovery is completed
  - Identifies congestion events vs. congestion signals
- Initial estimation for ssthresh

Rate Halving Recovery

- Sent after every other dupack

Delayed Ack Impact

- TCP typically only ACKs every other packet
- TCP congestion control triggered by acks
  - If receive half as many acks \( \rightarrow \) window grows half as fast
- Slow start with window = 1
  - Will trigger delayed ack timer
  - First exchange will take at least 200ms
  - Start with > 1 initial window
  - Bug in BSD, now a "feature"/standard
Next Lecture: Transport Alternatives

- TCP Vegas
- Alternative Congestion Control
- Header Compression
- Assigned reading
  - [BP95] TCP Vegas: End to End Congestion Avoidance on a Global Internet
  - [FHPW00] Equation-Based Congestion Control for Unicast Applications