Transport Level Congestion Control

Orientation
Definitions

- **Flow Control:**
  Protocol mechanisms to prevent that the sender overruns the receiver with information

- **Congestion Control:**
  Protocol mechanisms that prevent an overload of the network (by controlling the transmission rates of senders)

- **Error Control:**
  Algorithms to recover or conceal the effects from packet losses

How does a network become congested?

- Suppose a router is overloaded
- Transmission queue grows (overflows)....
- …until packets get delayed very long or get lost
- This causes retransmissions due to timeouts or loss detections
- Retransmissions increase traffic …
  …. so delays and losses at overloaded router increase
How congestion can spread

- If routing algorithms direct traffic around congested area…

… congestion can spread throughout the network.

Delay at a single buffer (M/M/1 queue)

- **Assumptions:**
  - Poisson traffic assumption: $\lambda$ packets/µs
  - Packet length is exponentially distributed with mean $1/\mu$
  - **Load** (=fraction of time that the queue or server are not empty):
    \[
    \rho = \frac{\lambda}{\mu}
    \]
  - **Average delay:**
    \[
    W = \frac{1}{\mu - \lambda}
    \]
What happens during congestion?

- **Knee** – Point after which throughput increases very slow, but delay increases fast

- **Cliff** – Point after which Throughput starts to decrease to zero (congestion collapse) and delay grow to infinity

---

**Congestion Control vs. Congestion Avoidance**

- **Congestion control** goal
  - Stay left of cliff

- **Congestion avoidance** goal
  - Stay left of knee

Good goal: Operate network near the “Knee”
Congestion control as a feedback system

- Congestion control can be seen as a feedback system

Sources adjust traffic rate

Network switches sense state (load) and feeds state back to traffic sources

• Reduce traffic if load is high
• Increase traffic if load is low

Issues to be addressed:
1. When to send feedback?
2. How to send feedback?
3. How to adjust rate?
How to detect congestion?

- **Explicit network signal**
  - Set bit in header of a packet when it encounters congestion
  - Receiver returns feedback signal
  - Used in: DEC DNA, ATM ABR, TCP ECN

- **Implicit network signal**
  - Acknowledgement for new data is interpreted as no congestion
  - Packet loss is seen as sign of congestion
  - Used in: TCP Tahoe, Reno, New Reno, SACK

---

Binary congestion control

**Network Model:**

- Discrete time: \( t=1,2,3,\ldots \) (feedback takes one time unit)
- \( x_i(t) \) : load from source \( i \) at time \( t \)
- Network is represented as a single resource ("bottleneck resource").
- \( X_{\text{goal}} \) is desired load level at "Knee"

- \( y(t) \) : binary feedback at time \( t \)
  - \( y(t)=0 \): No congestion (increase load)
  - \( y(t)=1 \): Congestion (decrease load)

Binary congestion control is widely used: TCP (Tahoe, Reno) and ATM ABR!
Objectives of binary feedback scheme

- **Fairness:**
  - All sources should be treated “fairly”

- **Efficiency:**
  - Network resources should be well utilized

- **Convergence:**
  - Network should quickly converge to desired load level
  - Load should not oscillate

- **Distributedness:**
  - No entity has complete knowledge
  - Sources do not communicate with each other

---

Efficiency

- **Efficiency:**
  - Network resources should be used well
  - Network should operate around the “Knee”

- $X_{\text{goal}}$ is desired load level at “Knee”

- Optimal allocation: $\sum x_i = X_{\text{goal}}$
Fairness

- **Maxmin fairness**
  - Flows which share the same bottleneck get the same amount of bandwidth

- Fairness is evaluated with a fairness index:
  \[ F(x) = \frac{\left(\sum x_i\right)^2}{n\left(\sum x_i^2\right)} \]

- Fairness index = 1: totally fair
- Fairness index = 0: totally unfair

Possible Control functions

\[ x_i(t + 1) = \begin{cases} 
  a_i + b_i x_i(t) & \text{if } y(t) = 0 \Rightarrow \text{increase} \\
  a_D + b_D x_i(t) & \text{if } y(t) = 1 \Rightarrow \text{decrease} 
\end{cases} \]

- 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 (AIMD)
  - \( a_i > 0, b_i = 1, a_D = 0, 0 < b_D < 1 \)
- Which one?
Operating Point

- Operating point is at the intersection of fairness and efficiency lines.

Additive changes

- Additive changes: change move in a 45° angle.
Multiplicative changes

- **Multiplicative changes:**
  change move along a line through the current point and the origin

Multiplicative Increase, Additive Decrease

- Does not converge to fairness
- Does not converges to efficiency
- Reaches equilibrium iff
  \[ x_1 = x_2 = \frac{b_1 a_D}{1 - b_I} \]
Additive Increase, Additive Decrease

- Does not converge to fairness
- Does not converge to efficiency
- Reaches equilibrium iff $a_D = a_I$

Multiplicative Increase, Multiplicative Decrease

- Does not converge to fairness
- Stable
- Converges to efficiency
Additive Increase, Multiplicative Decrease (AIMD)

- Converges to fairness
- Converges to efficiency
- Increments smaller as fairness increases

Importance of AIMD

- Characteristics
  - Only needs binary feedback information
  - Converges to efficient and fairness
    - Note: Previous discussion assumes constant delays and a single resource
    - Empirical evidence shows very good performance
      - Performance degrades if delays are very long
      - “Proportional fairness” in a general network

- AIMD-style congestion control used in most transport protocol
  - Implementation of TCP Tahoe congestion control introduced AIMD to Internet protocols
  - Still basis for TCP
TCP Congestion Control

- TCP has a mechanism for congestion control. The mechanism is implemented at the sender.

- The sender has two parameters:
  - **Congestion Window** \( (cwnd) \)
  - **Slow-start threshold Value** \( (ssthresh) \)
    - Initial value is the advertised window size.

- Congestion control works in **two modes**:
  - **slow start** \( (cwnd < ssthresh) \)
  - **congestion avoidance** \( (cwnd >= ssthresh) \)

### Slow Start

- **Initial value:**
  - Set \( cwnd = 1 \)
    - Note: Unit is a segment size. TCP actually is based on bytes and increments by 1 MSS (maximum segment size).

- The receiver sends an acknowledgement (ACK) for each packet.
  - Note: Generally, a TCP receiver sends an ACK for every other segment.

- Each time an ACK is received by the sender, the congestion window is increased by 1 segment:
  - \( cwnd = cwnd + 1 \)
    - If an ACK acknowledges two segments, \( cwnd \) is still increased by only 1 segment.
    - Even if an ACK acknowledges a segment that is smaller than MSS bytes long, \( cwnd \) is increased by 1.

- Does Slow Start increment slowly? Not really. In fact, the increase of \( cwnd \) is exponential.
**Slow Start Example**

- The congestion window size grows very rapidly
  - For every ACK, we increase cwnd by 1 irrespective of the number of segments ACK’ed
- TCP slows down the increase of cwnd when \( cwnd > ssthresh \)

<table>
<thead>
<tr>
<th>cwnd</th>
<th>ACKs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Segment 1</td>
</tr>
<tr>
<td>2</td>
<td>Segment 2</td>
</tr>
<tr>
<td>4</td>
<td>Segment 3</td>
</tr>
<tr>
<td>7</td>
<td>Segment 4</td>
</tr>
</tbody>
</table>

**Congestion Avoidance**

- Congestion avoidance phase is started if cwnd has reached the slow-start threshold value
- If \( cwnd >= ssthresh \) then each time an ACK is received, increment cwnd as follows:
  - \( cwnd = \lceil cwnd \rceil + \frac{1}{cwnd} \)
  
Where \( \lceil cwnd \rceil \) is the largest integer smaller than cwnd
- So cwnd is increased by one only if all cwnd segments have been acknowledged.
Example of Slow Start/Congestion Avoidance

Assume that \( ssthresh = 8 \)

Responses to Congestion

- So, TCP assumes there is congestion if it detects a packet loss
- A TCP sender can detect lost packets via:
  - Timeout of a retransmission timer
  - Receipt of a duplicate ACK
- TCP interprets a Timeout as a binary congestion signal. When a timeout occurs, the sender performs:
  - \( cwnd \) is reset to one:
    \[
    cwnd = 1
    \]
  - \( ssthresh \) is set to half the current size of the congestion window:
    \[
    ssthresh = \frac{cwnd}{2}
    \]
  - and slow-start is entered
Summary of TCP congestion control

Initially:
\[
cwnd = 1;
\]
\[
ssthresh = \text{advertised window size};
\]

New Ack received:
\[
\text{if (cwnd < ssthresh)}
\]
\[
// Slow Start*/
\]
\[
cwnd = cwnd + 1;
\]
\[
\text{else}
\]
\[
// Congestion Avoidance */
\]
\[
cwnd = cwnd + 1/cwnd;
\]

Timeout:
\[
// Multiplicative decrease */
\]
\[
ssthresh = cwnd/2;
\]
\[
cwnd = 1;
\]

Slow Start / Congestion Avoidance

- A typical plot of cwnd for a TCP connection: