Network Transport Layer: Congestion Control

Y. Richard Yang

http://zoo.cs.yale.edu/classes/cs433/

3/28/2016
Admin.

- Programming assignment 3
  - Check w/ TF on Monday/Wednesday on any eval setting issues
  - Results to be announced after the checking

- Programming assignment 4 update
  - Part 1 design review: Apr. 3

- Exam 1 (Monday next week)
  - See class page for samples from past
Recap: Reliable Transport

- Basic structure: sliding window protocols
- Remaining issues:
  - How to determine the “right” parameters?
History

- Key parameters for TCP in mid-1980s
  - fixed window size $W$
  - timeout value = 2 RTT

- Network collapse in the mid-1980s
  - UCB $\leftrightarrow$ LBL throughput dropped by 1000X!
Timeout

Impacts of timeout value:

- too short
  - premature timeout
  - unnecessary retransmissions; many duplicates

- too long
  - slow reaction to segment loss

Q: What is an ideal value for the timeout of a transmitted packet?
Estimating Round Trip Time

- Ideally, we set timeout = RTT, but RTT is not a fixed value
- SampleRTT: measured time from segment transmission until ACK receipt
- SampleRTT will vary, current approach: a “smoother” RTT est. use several recent measurements, not just current SampleRTT

\[
\text{EstimatedRTT} = (1-\alpha) \times \text{EstimatedRTT} + \alpha \times \text{SampleRTT}
\]

- Exponential weighted moving average
- influence of past sample decreases exponentially fast
- typical value: \( \alpha = 0.125 \)
**Setting Timeout**

**Problem:**
- Using the average of `SampleRTT` will generate many timeouts due to network variations.

**Solution:**
- Estimated `RTT` plus “safety margin” based on variation
  - Large variation in `EstimatedRTT` → larger safety margin

\[
DevRTT = (1 - \beta) \times DevRTT + \beta |SampleRTT - EstimatedRTT| 
\]

(typically, \( \beta = 0.25 \))

Then set timeout interval:

\[
TimeoutInterval = EstimatedRTT + 4 \times DevRTT
\]
An Example TCP Session
A Summary of Questions

- Basic structure: sliding window protocols
- How to determine the “right” parameters?
  - timeout: mean + variation
  - sliding window size
Principles of Congestion Control

Big picture:
- How to determine a flow’s sending rate?

Congestion:
- Informally: “too many sources sending too much data too fast for the network to handle”
- Different from flow control!
- Manifestations:
  - Lost packets (buffer overflow at routers)
  - Wasted bandwidth
  - Long delays (queueing in router buffers)
Some General Questions

- How can congestion happen?
- What is congestion control?
- Why is congestion control difficult?
Cause/Cost of Congestion: Single Bottleneck

- Flow 2 has a fixed sending rate of 5 Mbps
- We vary the sending rate of flow 1 from 0 to 20 Mbps
- Assume
  - no retransmission; link from router 1 to router 2 has infinite buffer

throughput: e2e packets delivered in unit time

<table>
<thead>
<tr>
<th>sending rate by flow 1 (Mbps)</th>
<th>throughput of flow 1 &amp; 2 (Mbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
</tr>
</tbody>
</table>

Delay?

delay due to randomness

delay at central link

<table>
<thead>
<tr>
<th>sending rate by flow 1 (Mbps)</th>
<th>delay at central link</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
**Cause/Cost of Congestion: Single Bottleneck**

- **flow 1**
- **flow 2 (5 Mbps)**

Assume

- **no retransmission**
- the link from router 1 to router 2 has *finite* buffer
- throughput: e2e packets delivered in unit time

![Diagram](image)

- **Zombie packet**: a packet dropped at the link from router 2 to router 5; the upstream transmission from router 1 to router 2 used for that packet was wasted!

$$\text{throughput of flow 1 & 2 (Mbps)} = \min(\frac{x}{x+5}10,5) + \frac{5}{x+5}10$$
Summary: The Cost of Congestion

Cost

- Packet loss
  - wasted upstream bandwidth when a pkt is discarded at downstream
  - wasted bandwidth due to retransmission (a pkt goes through a link multiple times)

- High delay
Outline

- Recap
  - *Transport congestion control*
    - what is congestion
    - *congestion control approaches*
Rate-based vs. Window-based

Rate-based:
- Congestion control by explicitly controlling the sending rate of a flow, e.g., set sending rate to 128Kbps
- Example: ATM

Window-based:
- Congestion control by controlling the window size of a transport scheme, e.g., set window size to 64KBytes
- Example: TCP

Discussion: rate-based vs. window-based
Window-based congestion control is **self-clocking**: considers flow conservation, and adjusts to RTT variation automatically.
Sliding Window Congestion Control

- Transmission rate limited by congestion window size, $cwnd$, over segments:

\[ \text{Throughput} = \frac{cwnd \times MSS}{RTT} \text{ Bytes/sec} \]

- $cwnd$ segments, each with MSS bytes sent in one RTT:

Assume $W$ is small enough. Ignore small details. MSS: Minimum Segment Size
The Desired Properties of a Congestion Control Scheme

- **Efficiency**: close to full utilization but low delay
  - fast convergence after disturbance

- **Fairness (resource sharing)**

- **Distributedness (no central knowledge for scalability)**
A Simple Model

Flows observe congestion signal $d$, and locally take actions to adjust rates.
Linear Control

- Proposed by Chiu and Jain (1988)
- The simplest control strategy

\[ x_i(t + 1) = \begin{cases} 
  a_I + b_I x_i(t) & \text{if } d(t) = \text{no cong.} \\
  a_D + b_D x_i(t) & \text{if } d(t) = \text{cong.} 
\end{cases} \]

Discussion: values of the parameters?
State Space of Two Flows

\[
x_i(t + 1) = \begin{cases} 
  a_I + b_I x_i(t) & \text{if } d(t) = \text{no cong.} \\
  a_D + b_D x_i(t) & \text{if } d(t) = \text{cong.}
\end{cases}
\]

fairness line: \( x_1 = x_2 \)

overload

underload

efficiency line: \( x_1 + x_2 = C \)
\[ x_i(t+1) = \begin{cases} 
    a_i + b_i x_i(t) & \text{if } d(t) = \text{no cong.} \\
    a_D + b_D x_i(t) & \text{if } d(t) = \text{cong.} 
\end{cases} \]
Implication: Congestion (overload) Case

- In order to get closer to efficiency and fairness after each update, decreasing of rate must be **multiplicative decrease** (MD)
  - $a_D = 0$
  - $b_D < 1$

\[
x_i(t + 1) = \begin{cases} 
  a_I + b_I x_i(t) & \text{if } d(t) = \text{no cong.} \\
  b_D x_i(t) & \text{if } d(t) = \text{cong.}
\end{cases}
\]
no-congestion

\[
x_i(t + 1) = \begin{cases} 
    a_l + b_l x_i(t) & \text{if } d(t) = \text{no cong.} \\
    a_D + b_D x_i(t) & \text{if } d(t) = \text{cong.}
\end{cases}
\]
Implication: No Congestion Case

- In order to get closer to efficiency and fairness after each update, additive and multiplicative increasing (AMI), i.e.,
  - $a_I > 0, b_I > 1$

$$x_i(t+1) = \begin{cases} 
  a_I + b_I x_i(t) & \text{if } d(t) = \text{no cong.} \\
  b_D x_i(t) & \text{if } d(t) = \text{cong.}
\end{cases}$$

- Simply additive increase gives better improvement in fairness (i.e., getting closer to the fairness line)
- Multiplicative increase may grow faster
# Four Special Cases

<table>
<thead>
<tr>
<th></th>
<th>Additive Decrease</th>
<th>Multiplicative Decrease</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Additive Increase</strong></td>
<td>AIAD (b_I=b_D=1)</td>
<td>AIMD (b_I=1, a_D=0)</td>
</tr>
<tr>
<td><strong>Multiplicative Increase</strong></td>
<td>MIAD (a_I=0, b_I&gt;1, b_D=1)</td>
<td>MIMD (a_I=a_D=0)</td>
</tr>
</tbody>
</table>

\[
x_i(t + 1) = \begin{cases} 
  a_I + b_I x_i(t) & \text{if } d(t) = \text{no cong.} \\
  a_D + b_D x_i(t) & \text{if } d(t) = \text{cong.}
\end{cases}
\]

Discussion: state transition trace.
AIMD: State Transition Trace

- fairness line: $x_1 = x_2$
- efficiency line: $x_1 + x_2 = C$

- overload
- underload

$\begin{align*} & \text{fairness line: } \quad x_1 = x_2 \\ & \text{efficiency line: } \quad x_1 + x_2 = C \end{align*}$
Another Look

- Consider the difference or ratio of the rates of two flows
  - AIAD
  - MIMD
  - MIAD
  - AIMD
Mapping A(M)I-MD to Protocol

What do we need to apply the A(M)I-MD algorithm to a sliding window protocol?

\[ x_i(t+1) = \begin{cases} 
  a_I + x_i(t) & \text{if } d(t) = \text{no cong.} \\
  b_D x_i(t) & \text{if } d(t) = \text{cong.} 
\end{cases} \]
Outline

Recap

- *Transport congestion control*
  - what is congestion
  - congestion control using the AIMD alg
  - *TCP congestion control*
TCP Congestion Control

- Closed-loop, end-to-end, window-based congestion control
- Designed by Van Jacobson in late 1980s, based on the AIMD alg. of Dah-Ming Chu and Raj Jain
- Worked in a large range of bandwidth values: the bandwidth of the Internet has increased by more than 200,000 times

- Many versions
  - TCP/Tahoe: this is a less optimized version
  - TCP/Reno: many OSs today implement Reno type congestion control
  - TCP/Vegas: not currently used

For more details: see TCP/IP illustrated; or read http://lxr.linux.no/source/net/ipv4/tcp_input.c for linux implementation
Basic Structure

- Two “phases”
  - slow-start: MI
  - congestion avoidance: AIMD

- Important variables:
  - cwnd: congestion window size
  - ssthresh: threshold between the slow-start phase and the congestion avoidance phase
Visualization of the Two Phases

- ssthresh
- cwnd (segments)
- round-trip times
Slow Start: MI

- What is the goal?
  - getting to equilibrium gradually but quickly

- Implements the MI algorithm
  - double cwnd every RTT until network congested
    \( \Rightarrow \) get a rough estimate of the optimal of cwnd
Slow-start

Initially:
   cwnd = 1;
   ssthresh = infinite (e.g., 64K);

For each newly ACKed segment:
   if (cwnd < ssthresh)
     /* slow start*/
     cwnd = cwnd + 1;
Startup Behavior with Slow-start

See [Jac89]
TCP/Reno Congestion Avoidance

- Maintains equilibrium and reacts around equilibrium

- Implements the AIMD algorithm
  - increases window by 1 per round-trip time (how?)
  - cuts window size
    - to half when detecting congestion by 3DUP
    - to 1 if timeout
    - if already timeout, doubles timeout
TCP/Reno Congestion Detection

- Detect congestion (d) in two cases and react differently:
  - 3 dup ACKs
  - timeout event

Philosophy:
- 3 dup ACKs indicates network capable of delivering some segments
- timeout is “more alarming”
Initially:
   cwnd = 1;
   ssthresh = infinite (e.g., 64K);
For each newly ACKed segment:
   if (cwnd < ssthresh)
      /* slow start*/
      cwnd = cwnd + 1;
   else
      /* congestion avoidance; cwnd increases (approx.)
       by 1 per RTT*/
      cwnd += 1/cwnd;
Triple-duplicate ACKs:
   /* multiplicative decrease */
   cwnd = ssthresh = cwnd/2;
Timeout:
   ssthresh = cwnd/2;
   cwnd = 1;
   (if already timed out, double timeout value; this is called exponential backoff)
TCP/Reno: Big Picture

TD: Triple duplicate acknowledgements
TO: Timeout
A Session

Question: when cwnd is cut to half, why sending rate is not?
Backup Slides
Implicit vs. Explicit

Explicit:
- routers provide feedback to end systems
  - explicit rate sender should send at
  - single bit indicating congestion (SNA, DECbit, TCP ECN, ATM)

Implicit:
- congestion inferred by end systems through observed loss, delay