Network Transport Layer: Congestion Control

Y. Richard Yang

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

11/2/2017
Admin.

- Programming assignment 4 Part 1 discussion slots:
  - Nov. 3: 11:15-12:15 pm
  - Nov. 6: 11-12; 2:30-5:30 pm
  - Nov. 7: 2:30-3:00; 4-6 pm
Recap: Protocol Analysis

General technique: write down joint state and analyze correctness.
Recap: Reliable Transport

- Basic structure: sliding window protocols
  - Realization: GBN or SR

<table>
<thead>
<tr>
<th></th>
<th>Go-back-n</th>
<th>Selective Repeat</th>
</tr>
</thead>
<tbody>
<tr>
<td>data bandwidth: sender</td>
<td>Less efficient</td>
<td>More efficient</td>
</tr>
<tr>
<td>to receiver (avg. number</td>
<td>$\frac{1-p + pw}{1-p}$</td>
<td>$\frac{1}{1-p}$</td>
</tr>
<tr>
<td>of times a pkt is</td>
<td></td>
<td></td>
</tr>
<tr>
<td>transmitted)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACK bandwidth</td>
<td>More efficient</td>
<td>Less efficient</td>
</tr>
<tr>
<td>(receiver to sender)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Relationship between M</td>
<td>$M &gt; W$</td>
<td>$M \geq 2W$</td>
</tr>
<tr>
<td>(the number of seq#) and</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W (window size)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Buffer size at receiver</td>
<td>1</td>
<td>W</td>
</tr>
<tr>
<td>Complexity</td>
<td>Simpler</td>
<td>More complex</td>
</tr>
</tbody>
</table>

General technique: pipelining.
Recap: Connection Management

Distributed synchronization/agreement is a basic, hard problem.
TCP: reliable data transfer

Hybrid GBN/SR

Key Parameters?
Summary: Reliable Transport

- Basic structure: sliding window protocol
- Remaining issue: How to determine the “right” parameters?
  - timeout value?
  - sliding window size?
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!

- The intuition was that the collapse was caused by wrong parameters...
Outline

- Admin and recap
- Setting timeout (RTO)
Timeout: Cost of Timeout Param

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

- too long
  - slow reaction to segment loss

Q: Is it possible to set Timeout as a constant?

Q: Any problem w/ the early approach: Timeout = 2 RTT
Setting Timeout

**Problem:**
- Ideally, we set timeout = RTT, but RTT is not a fixed value
  => using the average of RTT will generate many timeouts due to network variations

**Solution:**
- Set Timeout RTO = avg + “safety margin” based on variation

**Current approach:**

\[
\text{Timeout} = \text{EstRTT} + 4 \times \text{DevRTT}
\]
Compute EstRTT and DevRTT

- Exponential weighted moving average
  - influence of past sample decreases exponentially fast

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

- **SampleRTT**: measured time from segment transmission until ACK receipt
  - typical value: \( \alpha = 0.125 \)

\[ \text{DevRTT} = (1-\beta) \times \text{DevRTT} + \beta \times |\text{SampleRTT} - \text{EstRTT}| \]

  (typically, \( \beta = 0.25 \))
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?
**Sliding Window Size Function: Rate Control**

- Transmission rate determined by congestion window size, `cwnd`, over segments:

$$\text{Rate} = \frac{\text{cwnd} \times \text{MSS}}{\text{RTT}} \text{ Bytes/sec}$$

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

Assume W is small enough. Ignore small details. MSS: Minimum Segment Size
Some General Questions

Big picture question:

- How to determine a flow’s sending rate?

For better understanding, we need to look at a few basic questions:

- What is congestion (cost of congestion)?
- Why are desired properties of congestion control?
Roadmap

- What is congestion
- The basic CC alg
- TCP/reno CC
- TCP/Vegas
- A unifying view of TCP/Reno and TCP/Vegas
- Network wide resource allocation
  - Framework
  - Axiom derivation of network-wide objective function
  - Derive distributed algorithm
Outline

- Admin and recap
- Setting timeout
- Transport congestion control
  - what is congestion (cost of congestion)
**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

**Delay?**

- delay due to randomness

**Graphs:**
- Throughput of flow 1 & 2 (Mbps)
  - sending rate by flow 1 (Mbps)
- Delay at central link
  - sending rate by flow 1 (Mbps)
Cause/Cost of Congestion: Single Bottleneck

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

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!
Summary: The Cost of Congestion

When sources sending rate too high for the network to handle”:

- **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

- Admin and recap
- Setting timeout
- Transport congestion control
  - what is congestion (cost of congestion)
  - basic congestion control alg.
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)
Derive CC: A Simple Model

Flows observe congestion signal \( d \), and locally take actions to adjust rates.

\[
d = \text{sum } x_i \quad > \quad X_{\text{goal}}?
\]
Linear Control

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

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

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

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

- Overload
- Underload
- Fairness line: \( x_1 = x_2 \)
- Efficiency line: \( x_1 + x_2 = C \)
congestion

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

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

\[
x_i(t+1) = \begin{cases} 
    a_I + b_I x_i(t) & \text{if } d(t) = \text{no cong.} \\
    b_D x_i(t) & \text{if } d(t) = \text{cong.}
\end{cases}
\]
\[ x_i(t+1) = \begin{cases} \alpha_i + \beta_i x_i(t) & \text{if } d(t) = \text{no cong.} \\
\alpha_D + \beta_D x_i(t) & \text{if } d(t) = \text{cong.} \end{cases} \]
Implication: No Congestion Case

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

- Simply additive increase gives better improvement in fairness (i.e., getting closer to the fairness line)

- Multiplicative increase may grow faster

\[
x_i(t+1) = \begin{cases} 
a_I + b_I x_i(t) & \text{if } d(t) = \text{no cong.} \\
\ b_D x_i(t) & \text{if } d(t) = \text{cong.} 
\end{cases}
\]
Intuition: State Trace Analysis of 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>AIMP (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>MIMP (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: $x_1 + x_2 = C$

Points:
- $x_0$
- $x_1$
- $x_2$

Areas:
- Underload
- Overload
Intuition: Another Look

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

Question to look at: How do we apply the A(M)I-MD algorithm?

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

- Window-based congestion control is self-clocking: considers flow conservation, and adjusts to RTT variation automatically.
- Hence, for better safety, more designs use window-based design.
Mapping A(M)I-MD to Protocol

- Question to look at: How do we apply the A(M)I-MD algorithm?

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

- Admin and recap
- Setting timeout
- Transport congestion control
  - what is congestion (cost of congestion)
  - basic congestion control alg.
  - TCP/reno 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”
  - MI: slow-start
  - AIMD: congestion avoidance

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

- **Algorithm**: MI
  - **double cwnd** every RTT until network congested

- **Goal**: getting to equilibrium gradually but quickly, to get a rough estimate of the optimal of *cwnd*
**MI: Slow-start**

Initially:
\[
cwnd = 1; \\
\text{ssthresh} = \text{infinite (e.g., 64K)};
\]

For each newly ACKed segment:
\[
\text{if (} \text{cwnd} < \text{ssthresh}\) \\
/* \text{MI: slow start}*\/ \\
\text{cwnd} = \text{cwnd} + 1;
\]
Startup Behavior with Slow-start

See [Jac89]
AIMD: Congestion Avoidance

- Algorithm: AIMD
  - 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

- Goal: Maintains equilibrium and reacts around equilibrium
TCP/Reno Full Alg

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

For each newly ACKed segment:
  if (cwnd < ssthresh) // slow start: MI
      cwnd = cwnd + 1;
  else // congestion avoidance: AI
      cwnd += 1/cwnd;

Triple-duplicate ACKs:
    // MD
    cwnd = ssthresh = cwnd/2;

Timeout:
  ssthresh = cwnd/2; // reset
  cwnd = 1;
  (if already timed out, double timeout value; this is called exponential backoff)