Sliding Window Protocol and TCP Congestion Control

Simon S. Lam
Department of Computer Science
The University of Texas at Austin
Sliding Window Protocol

- Consider an infinite array, Source, at the sender, and an infinite array, Sink, at the receiver.

Source:

P1
Sender

| 0 | 1 | 2 | a-1 | a | s-1 | s |

Send window

acknowledged

unacknowledged

Sink:

P2
Receiver

| 0 | 1 | 2 | r |

Next expected

Received

Delivered

Receive window

\( RW \) receive window size

\( SW \) send window size \((s - a \leq SW)\)

TCP Congestion Control (Simon S. Lam)
**Sliding Windows in Action**

- Data unit $r$ has just been received by P2
  - Receive window slides forward
- P2 sends **cumulative ack** with sequence number it expects to receive next ($r+3$)

**TCP Congestion Control (Simon S. Lam)**
**Sliding Windows in Action**

- P1 has just received cumulative ack with \( r+3 \) as next expected sequence number
  - Send window slides forward

**Source:**

- P1 Sender
- Next expected sequence number: \( r+3 \)
- Next expected sequence number: \( r \)
- Sequence numbers: 0, 1, 2, a-1, a, s-1, s
- Send window: shaded in green
- Acknowledged: shaded in blue
- Delivered: shaded in yellow

**Sink:**

- P2 Receiver
- Sequence numbers: 0, 1, 2, r
- Receive window: shaded in yellow

*TCP Congestion Control (Simon S. Lam)*
Sliding Window protocol

- Functions provided
  - error control (reliable delivery)
  - in-order delivery
  - flow and congestion control (by varying send window size)

- TCP uses only cumulative acks

- Other kinds of acks
  - selective nack
  - selective ack (TCP SACK)
  - bit-vector representing entire state of receive window (in addition to first sequence number of window)
Sliding Windows for Lossy FIFO Channels

- A small number of bits in packet header for sequence number
- Necessary and sufficient condition for correct operation: \( SW + RW \leq \text{MaxSeqNum} \)
- Necessity:

Source:

\[
\begin{array}{cccccccc}
0 & 1 & 2 & a-1 & a & & & \\
\end{array}
\]

\( \text{send window} \)

\( \text{acknowledged} \)

\( \text{unacknowledged} \)

Sink:

\[
\begin{array}{cccccccc}
0 & 1 & 2 & & & & & \\
\end{array}
\]

\( \text{delivered} \)

\( \text{next expected} \)

\( \text{receive window} \)

\( RW \) receive window size

\( SW \) send window size

TCP Congestion Control (Simon S. Lam)
Sliding Windows for Lossy FIFO Channels


- Interesting special cases
  - $SW = RW = 1$ alternating-bit protocol
  - $SW = 7, RW = 1$ out-of-order arrivals not accepted, e.g., HDLC
  - $SW = RW$
Sliding Windows for LRD Channels

(LRD stands for Lossy Duplicative Reordering)

- **Assumption**: Packets have bounded lifetime $L$
- Be careful how fast sequence numbers are consumed (i.e., data arrive for sending into network)
  
  \[(\text{send rate}) \times L < \text{MaxSeqNum}\]

- **TCP**
  - 32-bit sequence numbers
  - counts bytes
  - assumes that datagrams will be discarded by IP if too old
Window Size Controls Sending Rate

- \( \approx W \) packets per RTT when no loss

TCP Congestion Control (Simon S. Lam)
Throughput

- Limit the number of unacked transmitted packets in the network to window size $W$

- Max. throughput $\approx \frac{W}{RTT}$ packets/sec

$$\frac{W \times MSS}{RTT} \text{ bytes/sec}$$

(assuming no loss, $MSS$ denotes maximum segment size)

- Where did we apply Little’s Law?
  
  *Answer*: Consider the TCP send buffer
Throughput or send rate?

- Previous formula provides an upper bound
  - Average number in the send buffer is less than $W$ unless packet arrival rate to send buffer is infinite
  - If a packet is lost in the network with probability $p$, then the average time in send buffer is $(1 - p) \times RTT + p \times T_o$
    Since $T_o > RTT$, actual throughput is smaller.

- The throughput of a host’s TCP send buffer is the host’s send rate into the network (including original transmissions and retransmissions)
- As a result of loss, the end-to-end goodput is $(1 - p) \times \text{throughput}$
  or $(1 - p) \times \text{sendrate}$
TCP Window Control

- **Receiver flow control**
  - Avoid overloading receiver
  - rwnd: receiver (advertised) window
  - Receiver sends rwnd to sender

- **Network congestion control**
  - Sender tries to avoid overloading network
  - It infers network congestion from “loss indications”
  - cwnd: congestion window

- **Sender sets** $W = \min (cwnd, rwnd)$
Receiver Flow Control

- Size of `rwnd` indicates available space in receive buffer
  - decreased when data is received from IP layer and ack’d
  - increased when data is consumed by application process
- Receiver advertises `rwnd` in each packet it sends
Effect of Congestion

- $W$ too big for many flows $\rightarrow$ congestion
- Packet loss $\rightarrow$ transmissions on links a packet has traversed prior to its loss are wasted
- Congestion collapse due to too many retransmissions and too much wasted transmission capacity
- October 1986, Internet had its first congestion collapse

TCP Congestion Control (Simon S. Lam)
Network Congestion Control

- Sender calculates cwnd from indications of network congestion

- Congestion indications
  - timeout (loss)
  - 3 dupACKs (loss likely)
  - queueing delay
  - mark (needs ECN)
TCP Congestion Control

- Tahoe (Jacobson 1988)
  - Slow Start
  - Congestion Avoidance (CA)
  - Fast Retransmit

- Reno (Jacobson 1990)
  - Fast Recovery
  - Variants: NewReno, SACK

- Vegas (Brakmo & Peterson 1994)
  - New Congestion Avoidance

- AQM
  - RED (Floyd & Jacobson 1993)
  - REM (Athuraliya & Low 2000)

- Others...
Slow Start

- Start with $cwnd = 1$
- On each successful ACK, increment $cwnd$
  $$cwnd \leftarrow cwnd + 1$$
- Exponential growth of $cwnd$
  each RTT: $$cwnd \leftarrow 2 \times cwnd$$
- Enter *Congestion Avoidance* when $cwnd \geq ssthresh$
- For initial slow start, $ssthresh$ is set to a very large value (e.g., 65 Kbytes)

Note: for clarity in these slides, $cwnd$, $rwnd$, and $ssthresh$ are counted in packets (segments) rather than in bytes
**Slow Start**

- **sender**
  - `cwnd` increases by 1 for each ACK
  - 1 RTT
  - Data packet

- **receiver**
  - ACK

TCP Congestion Control (Simon S. Lam)
Congestion Avoidance (CA)

- CA starts when $cwnd \geq ssthresh$
- On each successful ACK:
  
  $cwnd \leftarrow cwnd + \frac{1}{cwnd}$

- Linear growth of $cwnd$
  
  each RTT:
  
  $cwnd \leftarrow cwnd + 1$
Packet Loss

- Assumption: loss indicates congestion
- Packet loss detected by
  - Retransmission timeout (RTO timer)
  - Duplicate ACKs (at least 3)

Packets

1 2 3 4 5 6 7

Acknowledgements

2 3 4 4 4 4 4

time →

TCP Congestion Control (Simon S. Lam)
Fast Retransmit

- A timeout is quite long (> RTT)
- Upon receiving 3 dupACKs, sender immediately retransmits without waiting for timeout

- Adjusts ssthresh

\[ \text{ssthresh} \leftarrow \max(\text{flightsize}/2, 2) \]

where flightsize is number of outstanding packets, which may be less than \( W = \min(\text{rwnd}, \text{cwnd}) \)

- Enter Slow Start (cwnd = 1) [TCP Tahoe]

TCP Congestion Control (Simon S. Lam)
TCP Tahoe (Jacobson 1988)

SS: Slow Start
CA: Congestion Avoidance

- Decrease to 1 for either timeout or 3 dupACKs
- Fast retransmit on 3 dupACKs
Successive Timeouts

- When there is another timeout, double the timeout value
- Keep doing so for each additional loss-retransmission
  - Exponential backoff up to max timeout value equal to 64 times initial timeout value

Note: red line in figure denotes first timeout
Summary: Tahoe

- Probe network for spare capacity during SS and CA and increase send rate
  - Drastically reduce rate on loss indication

```plaintext
for every ACK {
  if (W < ssthresh) then W ← W + 1 (SS)
  else W ← W + 1/W (CA)
}
for every loss indication {
  ssthresh ← W/2
  W ← 1
}
```

- Self-clocking
- Error recovery by retransmission
  - fast retransmit upon 3 duplicate acks
- Need to estimate round trip time (to get $T_O$ value)

TCP Congestion Control (Simon S. Lam)
**TCP Reno** (Jacobson 1990)

- **SS**: Slow Start
- **CA**: Congestion Avoidance

**Fast retransmit + fast recovery on 3 dupACKs**

TCP Congestion Control (Simon S. Lam)
TCP Reno (another scenario)

- **3 dupACKs**
- **Initial slow start**
- **halved**
- **Slow start until cwnd reaches ssthresh**

3 dupACKs during initial slow start

TCP Congestion Control (Simon S. Lam)
Fast recovery (in more detail)

- Idea: each dupACK represents a packet successfully received. Therefore, no need for very drastic action
- Enter FR/FR after 3 dupACKs
  - Set ssthresh ← max(flightsize/2, 2)
  - Retransmit lost packet
  - Set cwnd ← ssthresh + #dupACKs (window inflation)
  - Wait till W=min(rwnd, cwnd) is large enough; transmit new packet(s)
  - On non-dup ACK (1 RTT later), set cwnd ← ssthresh (window deflation)
- Enter CA
**Example: FR/FR entry and exit**

- **Above scenario:** Packet 1 is lost, packets 2, 3, and 4 are received; **3 dupACKs** with seq. no. 1 returned
- **Fast retransmit**
  - Retransmit packet 1 upon 3 dupACKs
- **Fast recovery**
  - Inflate window with #dupACKs such that new packets 9, 10, and 11 can be sent while repairing loss

TCP Congestion Control (Simon S. Lam)
**AIMD in steady state**

**additive increase:**
- increase \( \text{cwnd} \) by 1 MSS every RTT in the absence of any loss event

**multiplicative decrease:**
- cut \( \text{cwnd} \) in half after 3 dupACKs

![Graph showing the congestion window over time for a long-lived TCP connection.](image-url)
TCP throughput (send rate)

- Approximate formula from Little’s Law (assuming no loss)

\[
\text{max. send rate} \approx \frac{W}{RTT} \text{ packets/sec}
\]

- \( W \) changes with the arrival of each congestion indication
- To calculate (average) send rate, we need the average value of \( W \)

Q: \( W \) is a function of what parameter?
First approximation


- No slow-start, no timeout, long-lived TCP connection
- Independent identically distributed “periods”
- Three dupACKs are received in a round with probability \( p \)

Ave. congestion window (packets)

\[
\begin{array}{c|c|c|c|c}
\text{Time (RTT)} & 0 & W/2 & W & 3W/2 & 2W \\
\hline
\text{# of RTTs} & 0 & W/4 & W/2 & 3W/4 & W
\end{array}
\]
Geometric Distribution
Ave. no. of transmissions to get first “triple dupACKs”

\[ n = \sum_{i=1}^{\infty} i b_i = \sum_{i=1}^{\infty} i (1 - p)^{i-1} p \]

\[ = p \sum_{i=1}^{\infty} i (1 - p)^{i-1} \]

\[ = -p \frac{d}{dp} \sum_{i=1}^{\infty} (1 - p)^i = -p \frac{d}{dp} \sum_{i=0}^{\infty} (1 - p)^i \]

\[ = -p \frac{d}{dp} \frac{1}{1 - 1 + p} = p \frac{1}{p^2} \]

\[ = 1 / p \]

Aside: ave. no. of transmissions to get first success is \( 1/(1-p) \)
First approximation (cont.)

- Average number of packets delivered in one period (area under one saw-tooth):
  \[
  \left(\frac{W}{2}\right)^2 + \frac{1}{2}\left(\frac{W}{2}\right)^2 = \frac{3}{8}W^2
  \]

- Average number of packets sent per period is \(1/p\)

- Equate the two and solve for \(W\), we get:
  \[
  W = \sqrt{\frac{8}{3p}}
  \]

Send rate (in packets/sec):

\[
\text{send rate} = \frac{\text{no. of packets/period}}{\text{time per period}}
\]

\[
= \frac{\frac{3}{8}W^2}{\text{RTT}\left(\frac{W}{2}\right)}
\]

\[
= \frac{1/p}{\text{RTT}\left(\sqrt{\frac{2}{3p}}\right)} = \frac{1}{\text{RTT} \sqrt{\frac{3}{2p}}}
\]

TCP Congestion Control (Simon S. Lam)
# TCP ACK generation [RFC 1122, RFC 2581]

<table>
<thead>
<tr>
<th>Event at Receiver</th>
<th>TCP Receiver action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed</td>
<td>Delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK</td>
</tr>
<tr>
<td>Arrival of in-order segment with expected seq #. One other segment has ACK pending</td>
<td>Immediately send single cumulative ACK, ACKing both in-order segments</td>
</tr>
<tr>
<td>Arrival of out-of-order segment higher-than-expect seq. #. Gap detected</td>
<td>Immediately send <em>duplicate ACK</em>, indicating seq. # of next expected byte</td>
</tr>
<tr>
<td>Arrival of segment that partially or completely fills gap</td>
<td>Immediate send ACK, provided that segment starts at lower end of gap</td>
</tr>
</tbody>
</table>

TCP Congestion Control (Simon S. Lam)
Receiver implements Delayed ACKs

- Receiver sends one ACK for every two packets received -> each saw-tooth is \( W \times \text{RTT} \) wide
  - area under a saw-tooth is \( \frac{3W^2}{4} = \frac{1}{p} \)

- Send rate is
  \[
  \frac{1/p}{RTT \cdot W} = \frac{1/p}{RTT \cdot \sqrt{4/(3p)}} = \frac{1}{RTT \sqrt{4p}}
  \]

- One ACK for every \( b \) packets received -> send rate is
  \[
  \frac{1}{RTT \sqrt{2bp}}
  \]
A more detailed model

Reference:
Motivation

- Previous formulas not so accurate when loss rates are high
- TCP traces show that there are more loss indications due to timeouts (TO) than due to triple dupACKs (TD)
Objectives

- More accurate steady-state throughput formula as a function of loss (indication) rate and RTT by also accounting for TO behavior of a TCP connection
- Formula applicable over a wider range of loss rates
- Explicit statements of assumptions and approximations used in derivation of throughput formula
- Formula to include the impact of a small rwnd
**Many assumptions and approximations**

- **A1.** TCP sender is saturated, i.e., source application process always has a packet to send when send window has space available
  - bulk transfer application

- **A2.** Slow Start not modeled

- **A3.** Time to send all packets in a window is smaller than RTT (as shown in slide 9)
  - transmission rate is not too low
AIMD evolution of Window Size over time

- **A4.** Each TD period is ended by a TD loss indication.
  - TDP$_i$ period has duration $A_i$ RTTs

- **A5.** Duration of a round (RTT) is independent of window size
  - poor assumption for a slow line

- **A6.** Fast Recovery not modeled

TCP Congestion Control (Simon S. Lam)
Loss assumptions

- **A7.** Losses in different rounds are independent

- **A8.** Losses within the same round are correlated as follows: If a packet is lost, all remaining packets transmitted until the end of that round are also lost
  - all lost packets in the same round are counted as a single loss indication when estimating $p$

- **A9.** Assume that $\{A_{ij}\}$ and $\{W_{ij}\}$ are mutually independent i.i.d. sequences of random variables
**AIMD throughput (send rate)**

**A10.** Assume \{W_i\} to be a Markov regenerative process with rewards \{Y_i\}, where \(Y_i\) is the number of packets sent in TDP_i.

\[
\text{send rate } B(p) = \frac{E[Y]}{E[A]} = \frac{1 - p + E[W]}{\frac{p}{E[A]}} = \frac{1}{p} \frac{1}{RTT} \left( \frac{2b}{\sqrt[3]{3p}} \right) + o(1/\sqrt{p})
\]

\[
\approx \frac{1}{RTT} \sqrt{\frac{3}{2bp}} + o(1/\sqrt{p})
\]
AIMD with Timeouts

Let $Y_{ij}$ denote number of packets sent in jth period of $Z_{i}^{TD}$
AIMD with Timeouts (cont.)

- Let $n_i$ denote the number of TD periods within a cycle which ends in $i$-th TO period, $R_i$ denote no. of retransmissions in $i$-th TO period.
- A11. $\{n_i\}$ form an i.i.d. sequence, independent of $\{Y_{ij}\}$ and $\{A_{ij}\}$.

\[
M_i = \sum_{j=1}^{n_i} Y_{ij} + R_i, \quad S_i = \sum_{j=1}^{n_i} A_{ij} + Z_i^{TO}
\]
Throughput of AIMD with TO


\[ E[S] = E[n]E[A] + E[Z^{TO}] \]

Send rate \( B = \frac{E[M]}{E[S]} = \frac{E[n]E[Y] + E[R]}{E[n]E[A] + E[Z^{TO}]} \)

\[ B = \frac{E[Y] + Q \times E[R]}{E[A] + Q \times E[Z^{TO}]} \]

where \( Q \equiv \frac{1}{E[n]} \)

\[ E[R] = \frac{1}{1 - p} \]

with \( Q \) and \( E[Z^{TO}] \) to be determined

Assumption of Markov regenerative process again.

<- Probability that a given loss indication is a TO

TCP Congestion Control (Simon S. Lam)
Throughput of AIMD with TO (cont.)

Prob[ a loss indication is a TO ]

\[ B(p) \approx \frac{1-p}{p} + E[W] + \frac{1}{1-p} \frac{E[A] + \hat{Q}(E[W])T_o f(p)}{1-p} \]

\[ \approx \frac{1}{p} \left( \frac{2b}{3p} \right) + \min \left( 1, 3 \sqrt{\frac{3bp}{8}} \right) (1 + 32p^2)T_o \]

\[ = \frac{1}{RTT \left( \sqrt{\frac{2bp}{3}} \right) + \min \left( 1, 3 \sqrt{\frac{3bp}{8}} \right) p(1 + 32p^2)T_o} \]

<- Eq. (27) more accurate version of throughput formula

Extra term added to account for Timeout

<- Eq. (29) most well-known version of throughput formula

TCP Congestion Control (Simon S. Lam)
Impact of receiver’s flow control limitation—approximate model

Using the well-known Eq. (29) from before,

\[ B(p) = \min\left( \frac{W_{\text{max}}}{RTT}, \frac{1}{RTT \left( \sqrt{\frac{2bp}{3}} \right) + \min \left( 1, 3\sqrt{\frac{3bp}{8}} \right) p(1 + 32p^2)T_0} \right) \]

where \( W_{\text{max}} \) is the maximum window size allowed by receiver.

The above is Eq. (32) referred to as the approximate model. The full model is Eq. (31) in the paper.
### Summary data from traces (1 hour)

- **Saturated TCP sender**
- \( p \) computed from dividing total no. of loss indications by total number of packets sent
- \( RTT \) and \( T_0 \) values are averaged over entire 1-hour trace

<table>
<thead>
<tr>
<th>Sender</th>
<th>Receiver</th>
<th>Packets Sent</th>
<th>Loss Indic.</th>
<th>TD</th>
<th>( T_0 )</th>
<th>( T_1 )</th>
<th>( T_2 )</th>
<th>( T_3 )</th>
<th>( T_4 ) or more</th>
<th>RTT</th>
<th>Time Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>manic</td>
<td>alps</td>
<td>54402</td>
<td>722</td>
<td>19</td>
<td>611</td>
<td>67</td>
<td>15</td>
<td>6</td>
<td>2</td>
<td>2</td>
<td>0.207</td>
</tr>
<tr>
<td>manic</td>
<td>baskerville</td>
<td>58120</td>
<td>735</td>
<td>306</td>
<td>411</td>
<td>17</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.243</td>
</tr>
<tr>
<td>manic</td>
<td>ganef</td>
<td>58924</td>
<td>743</td>
<td>272</td>
<td>444</td>
<td>22</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.226</td>
</tr>
<tr>
<td>manic</td>
<td>mafalda</td>
<td>56283</td>
<td>494</td>
<td>2</td>
<td>474</td>
<td>17</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.233</td>
</tr>
<tr>
<td>manic</td>
<td>maria</td>
<td>68752</td>
<td>649</td>
<td>1</td>
<td>604</td>
<td>35</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.180</td>
</tr>
<tr>
<td>manic</td>
<td>spiff</td>
<td>117992</td>
<td>784</td>
<td>47</td>
<td>702</td>
<td>34</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.211</td>
</tr>
<tr>
<td>manic</td>
<td>sutton</td>
<td>81123</td>
<td>1638</td>
<td>988</td>
<td>597</td>
<td>41</td>
<td>7</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0.204</td>
</tr>
<tr>
<td>manic</td>
<td>tove</td>
<td>7938</td>
<td>264</td>
<td>1</td>
<td>190</td>
<td>37</td>
<td>18</td>
<td>8</td>
<td>3</td>
<td>7</td>
<td>0.275</td>
</tr>
<tr>
<td>void</td>
<td>alps</td>
<td>37137</td>
<td>838</td>
<td>7</td>
<td>588</td>
<td>164</td>
<td>56</td>
<td>17</td>
<td>4</td>
<td>2</td>
<td>0.162</td>
</tr>
<tr>
<td>void</td>
<td>baskerville</td>
<td>32042</td>
<td>853</td>
<td>339</td>
<td>430</td>
<td>67</td>
<td>12</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0.482</td>
</tr>
<tr>
<td>void</td>
<td>ganef</td>
<td>60770</td>
<td>1112</td>
<td>414</td>
<td>582</td>
<td>79</td>
<td>20</td>
<td>9</td>
<td>4</td>
<td>2</td>
<td>0.254</td>
</tr>
<tr>
<td>void</td>
<td>maria</td>
<td>93005</td>
<td>1651</td>
<td>33</td>
<td>1344</td>
<td>197</td>
<td>54</td>
<td>15</td>
<td>5</td>
<td>3</td>
<td>0.152</td>
</tr>
<tr>
<td>void</td>
<td>spiff</td>
<td>65536</td>
<td>671</td>
<td>72</td>
<td>539</td>
<td>56</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.415</td>
</tr>
<tr>
<td>void</td>
<td>sutton</td>
<td>78246</td>
<td>1928</td>
<td>840</td>
<td>863</td>
<td>152</td>
<td>45</td>
<td>18</td>
<td>9</td>
<td>1</td>
<td>0.211</td>
</tr>
<tr>
<td>void</td>
<td>tove</td>
<td>8265</td>
<td>856</td>
<td>5</td>
<td>444</td>
<td>209</td>
<td>100</td>
<td>51</td>
<td>27</td>
<td>12</td>
<td>0.272</td>
</tr>
<tr>
<td>babel</td>
<td>alps</td>
<td>13460</td>
<td>1466</td>
<td>0</td>
<td>1068</td>
<td>247</td>
<td>87</td>
<td>33</td>
<td>18</td>
<td>8</td>
<td>0.194</td>
</tr>
<tr>
<td>babel</td>
<td>baskerville</td>
<td>62237</td>
<td>1753</td>
<td>197</td>
<td>1467</td>
<td>76</td>
<td>10</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0.253</td>
</tr>
<tr>
<td>babel</td>
<td>ganef</td>
<td>86675</td>
<td>2125</td>
<td>398</td>
<td>1686</td>
<td>38</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.201</td>
</tr>
<tr>
<td>babel</td>
<td>spiff</td>
<td>57687</td>
<td>1120</td>
<td>0</td>
<td>939</td>
<td>137</td>
<td>36</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>0.331</td>
</tr>
<tr>
<td>babel</td>
<td>sutton</td>
<td>83486</td>
<td>2320</td>
<td>685</td>
<td>1448</td>
<td>142</td>
<td>31</td>
<td>9</td>
<td>4</td>
<td>1</td>
<td>0.210</td>
</tr>
<tr>
<td>babel</td>
<td>tove</td>
<td>83944</td>
<td>1516</td>
<td>1</td>
<td>1364</td>
<td>118</td>
<td>17</td>
<td>7</td>
<td>5</td>
<td>3</td>
<td>0.194</td>
</tr>
<tr>
<td>pif</td>
<td>alps</td>
<td>83971</td>
<td>762</td>
<td>0</td>
<td>577</td>
<td>111</td>
<td>46</td>
<td>16</td>
<td>8</td>
<td>2</td>
<td>0.168</td>
</tr>
<tr>
<td>pif</td>
<td>imagine</td>
<td>44891</td>
<td>1346</td>
<td>15</td>
<td>1044</td>
<td>186</td>
<td>63</td>
<td>21</td>
<td>10</td>
<td>5</td>
<td>0.229</td>
</tr>
<tr>
<td>pif</td>
<td>manic</td>
<td>34251</td>
<td>1422</td>
<td>43</td>
<td>944</td>
<td>272</td>
<td>105</td>
<td>36</td>
<td>14</td>
<td>6</td>
<td>0.257</td>
</tr>
</tbody>
</table>

Table 2: Summary data from 1hr traces

TCP Congestion Control (Simon S. Lam) 48
Summary data from 100s traces

Each row represents results from 100 traces each 100 seconds long for same S-D pair

Totals are cumulative over 100 traces

\( RTT \) and \( T_0 \) are average values over 100 traces for same S-D pair

<table>
<thead>
<tr>
<th>Sender</th>
<th>Receiver</th>
<th>Packets Sent</th>
<th>Loss Indic.</th>
<th>TD</th>
<th>( T_0 )</th>
<th>( T_1 )</th>
<th>( T_2 )</th>
<th>( T_3 )</th>
<th>( T_4 ) or larger</th>
<th>RTT</th>
<th>Time Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>manic</td>
<td>ada</td>
<td>531533</td>
<td>6432</td>
<td>4320</td>
<td>2010</td>
<td>93</td>
<td>7</td>
<td>2</td>
<td>0</td>
<td>0.1419</td>
<td>2.2231</td>
</tr>
<tr>
<td>manic</td>
<td>after</td>
<td>2526/4</td>
<td>4577</td>
<td>2584</td>
<td>1898</td>
<td>83</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>0.1804</td>
<td>2.3009</td>
</tr>
<tr>
<td>manic</td>
<td>al</td>
<td>264002</td>
<td>4720</td>
<td>2841</td>
<td>1804</td>
<td>70</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0.1885</td>
<td>2.3542</td>
</tr>
<tr>
<td>manic</td>
<td>alps</td>
<td>667296</td>
<td>3797</td>
<td>841</td>
<td>2866</td>
<td>85</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0.1125</td>
<td>1.9151</td>
</tr>
<tr>
<td>manic</td>
<td>baskerville</td>
<td>89244</td>
<td>1638</td>
<td>627</td>
<td>955</td>
<td>42</td>
<td>11</td>
<td>2</td>
<td>1</td>
<td>0.4735</td>
<td>3.2269</td>
</tr>
<tr>
<td>manic</td>
<td>ganesf</td>
<td>160152</td>
<td>2470</td>
<td>1040</td>
<td>1308</td>
<td>89</td>
<td>18</td>
<td>6</td>
<td>1</td>
<td>0.2150</td>
<td>2.6078</td>
</tr>
<tr>
<td>manic</td>
<td>maria</td>
<td>171308</td>
<td>1332</td>
<td>9</td>
<td>1269</td>
<td>48</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0.2501</td>
<td>2.5127</td>
</tr>
<tr>
<td>manic</td>
<td>maria</td>
<td>316498</td>
<td>2476</td>
<td>5</td>
<td>2362</td>
<td>99</td>
<td>8</td>
<td>2</td>
<td>0</td>
<td>0.1166</td>
<td>1.8798</td>
</tr>
<tr>
<td>manic</td>
<td>modi4</td>
<td>282547</td>
<td>6072</td>
<td>3976</td>
<td>1988</td>
<td>99</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0.1749</td>
<td>2.2604</td>
</tr>
<tr>
<td>manic</td>
<td>pong</td>
<td>358535</td>
<td>4239</td>
<td>2328</td>
<td>1830</td>
<td>74</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0.1769</td>
<td>2.1371</td>
</tr>
<tr>
<td>manic</td>
<td>spiff</td>
<td>298465</td>
<td>2035</td>
<td>159</td>
<td>1781</td>
<td>75</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>0.2539</td>
<td>2.4545</td>
</tr>
<tr>
<td>manic</td>
<td>sutton</td>
<td>348926</td>
<td>6024</td>
<td>3694</td>
<td>2238</td>
<td>87</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0.1683</td>
<td>2.1852</td>
</tr>
<tr>
<td>manic</td>
<td>tove</td>
<td>262365</td>
<td>2603</td>
<td>6</td>
<td>2422</td>
<td>135</td>
<td>30</td>
<td>8</td>
<td>2</td>
<td>0.1153</td>
<td>1.9551</td>
</tr>
</tbody>
</table>

TCP Congestion Control (Simon S. Lam)
Experimental comparison (1)

- Each point represents number of packets in 100s interval of trace
- T0 ~ single TO, T1 ~ at least 1 double TO in trace, etc.
- “TD Only” is analytic model by Mathis et al.
- Note: $W_{\text{max}}$ is only 6 in Figure 7

Figure 7: manic to baskerville

Figure 8: pit to imagine

TCP Congestion Control (Simon S. Lam)
Experimental comparison (2)

Figure 9: pif to manic

\[ W_{\text{max}} = 33 \]

Figure 10: void to alps

\[ W_{\text{max}} = 44 \]

TCP Congestion Control (Simon S. Lam)
Experimental comparison (3)

Figure 11: void to tove

\[ W_{\text{max}} = 8 \]

Figure 12: babel to alps

\[ W_{\text{max}} = 48 \]
Accuracy of approximate model

Figure 18: manic to spiff, with predictions by both full and approximate models \((W_{\text{max}}=32)\)
Average errors

Figure 19: Comparison of the models for 1hr traces

\[
\text{ave. error} = \frac{\sum_{\text{observations}} \left| N_{\text{predicted}} - N_{\text{observed}} \right|}{\text{no. of observations}}
\]

Figure 20: Comparison of the models for 100s traces

TCP Congestion Control (Simon S. Lam)
Conclusions

- A more detailed analysis than the one by Mathis et al.
- Numerous assumptions and approximations used but (almost) all of them are explicitly stated
- Large amount of experimental measurements on the Internet to validate accuracy of the full model (less for the approximate model)
- Throughput formula accounts for loss indications due to TO as well as rwnd restriction
  - Using the formula requires accurate measurements of loss rate and RTT values (*which could be tricky*)
  - For TCP Reno and drop-tail router
- Accuracy (like beauty) is in the eye of the beholder. What do you think?

TCP Congestion Control (Simon S. Lam)
Challenge in the future

- TCP average throughput (approximate) in terms of loss (indication) rate, $p$

\[
\frac{1.22 \cdot MSS}{RTT \sqrt{p}}
\]

- Example: 1500-byte segments, 100ms RTT, to get 10 Gbps throughput, loss rate needs to be very low

\[p = 2 \times 10^{-10}\]

- New version of TCP needed for connections with large delay-bandwidth product
  - One proposal: Katabi et al. (*Sigcomm* 2002)

- Revised congestion control algorithms for data center networks
  - Very low delay, high bandwidth
The end