Estimating Network Bandwidth with TCP

Mark Allman

NASA GRC/BBN Technologies

`mallman@grc.nasa.gov`

http://roland.grc.nasa.gov/~mallman/

5600 Brown Bag Seminar

December, 1999

“I wish I was a neutron bomb...” – P.J.

Background

- Joint work with Vern Paxson (ACIRI/LBNL)
- Preliminary experiments over ACTS presented at the Satellite Networks and Architectures Workshop (Cleveland, June 1998).
- The work being discussed today was presented at ACM SIGCOMM (September, 1999).
  - In addition to investigations of appropriate retransmission timeouts.

Definitions

- A couple TCP-ish items for background...
  - The *congestion window* is the sender's estimate of the network capacity.
    - Number of segments that can be transmitted before receiving an acknowledgment (ACK).
  - Many TCP implementations use *delayed ACKs*:
    - An ACK is sent for every second arriving segment
    - If a second segment does not arrive within a certain amount of time we send an ACK
      - (the timeout cannot exceed 500 ms)

Introduction

- A fundamental internetworking problem: When I start up a connection between two hosts across some network path, how do I know what the appropriate sending rate is?
- The "appropriate" rate is based on many factors:
  - The bit-rate of the bottleneck link in the network path.
  - The RTT across the network path.
  - How many other flows are traversing the network path.
  - Possibly QoS provisioning (ala diffserv).

Estimating Bandwidth: Current Method

- Currently we use the “slow start” algorithm to get an idea about the available bandwidth on the network path.
  - We start transmission by injecting a single segment (or, maybe a few) into the network.
  - When we receive an ACK for that segment we know the network path had enough capacity for the packet.
  - So, we (roughly) double the number of segments we inject into the network and transmit again.
  - We continue until we reach the network path’s limit, which is detected when we observe packet loss.

Estimating Bandwidth: Current Method (cont.)

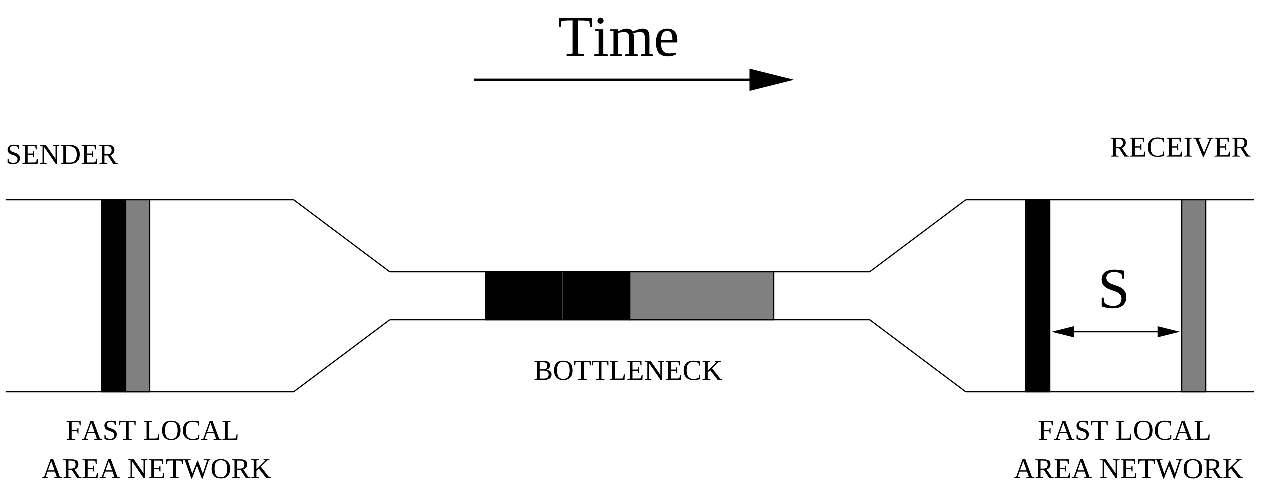
- Advantages of the current approach:
  - Conservative.
  - Slow start really isn't all that slow (in general).
  - Because of the increase rate we only overload the network by a factor or 2 in the worst case.
- Disadvantages of the current approach:
  - Because the algorithm requires feedback from the receiver it is slow when the round-trip time (RTT) is large.
  - We can load the network with more segments than it can handle leading to a potentially messy loss recovery procedure.

Estimating Bandwidth: A Better Way?

- So, can we devise a new and better way to estimate bandwidth that will diminish the disadvantages of the current algorithm while keeping the advantages?
  - Maybe, but it's a hard problem!
- Our study focused on reducing the number of segments lost at the end of slow start, rather than on increasing the speed of slow start.
  - Increasing the speed of slow start is future work.

Bandwidth Estimation Theory

- Keshav (SIGCOMM 1991) observed that using two packets sent back-to-back could be used to measure the bandwidth (the *packet pair* algorithm).
  - Part of a large control-theoretic approach to congestion control that is not currently used.



Bandwidth Estimation Theory (cont.)

- Hoe (SIGCOMM 1996) took Keshav's work and applied it to TCP.
  - The sender can observe the spacing between incoming ACKs and form a bandwidth estimate.
  - From the bandwidth estimate we can set the *slow start threshold*.
  - When the congestion window surpasses the slow start threshold, we terminate slow start (and move into congestion avoidance).
  - Hoe presented very promising simulation results.

Bandwidth Estimation Theory (cont.)

- But, a question remained: Does it work "in the wild"?
  - Under the influence of Microsoft-subsidized Red Hook, we decided that there was a decent chance.

Methodology

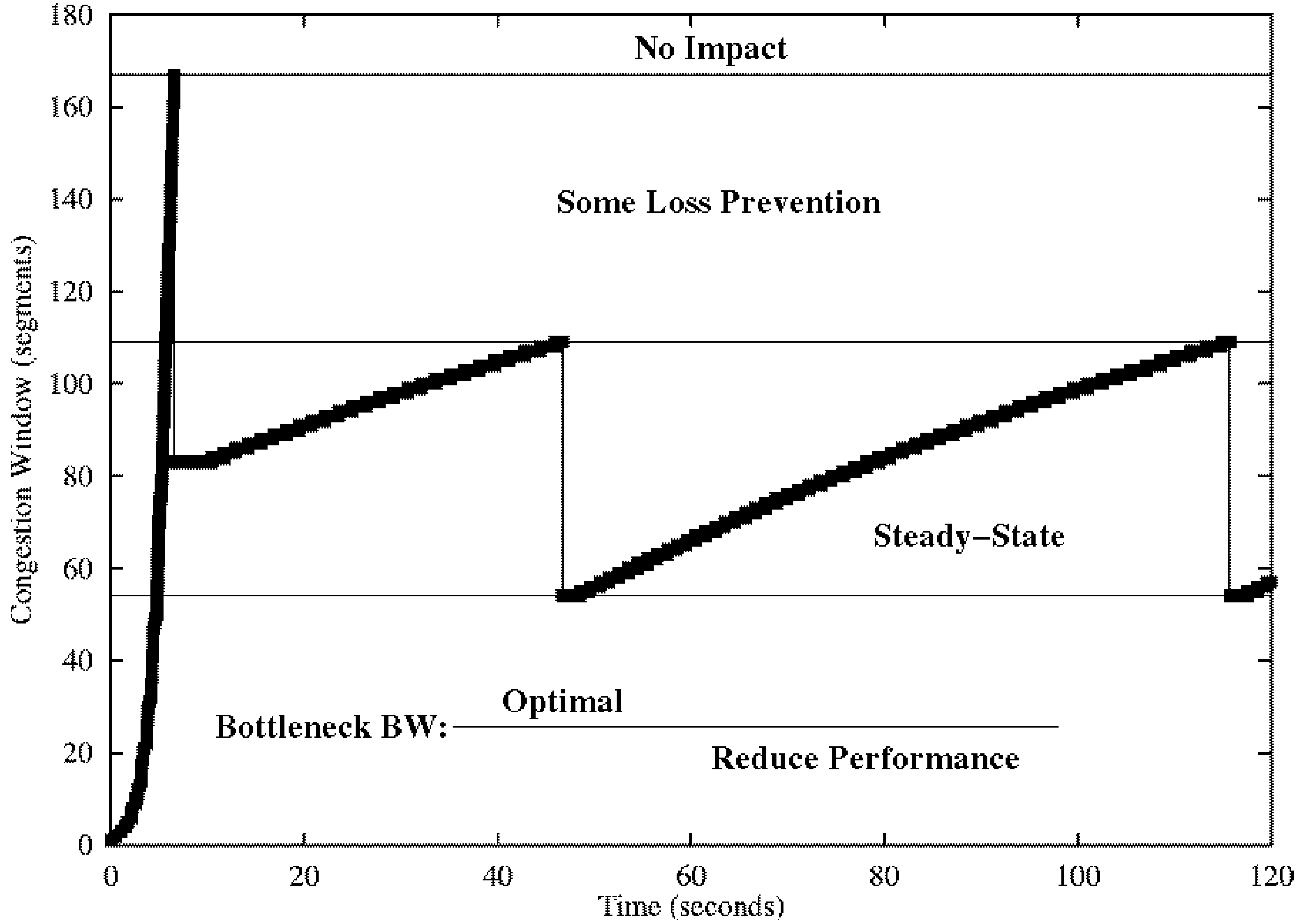
- Trace-driven simulation of $\mathcal{N}_2$ dataset [Pax97]
- 18,490 connections, 100 KB each, 31 hosts
- Compare our bandwidth estimates ...
  ... with what actually happened.
- We developed four classes of bandwidth estimation techniques that were investigated.
  - Some TCP-specific, others not

Methodology (cont.)

- As a point of comparison we used the Packet Bunch Mode (PBM) [Pax97] algorithm as a "correct" estimate of bandwidth.
  - Used as an upper bound on how well we might be able to do.
  - PBM is not appropriate for use "on the fly".

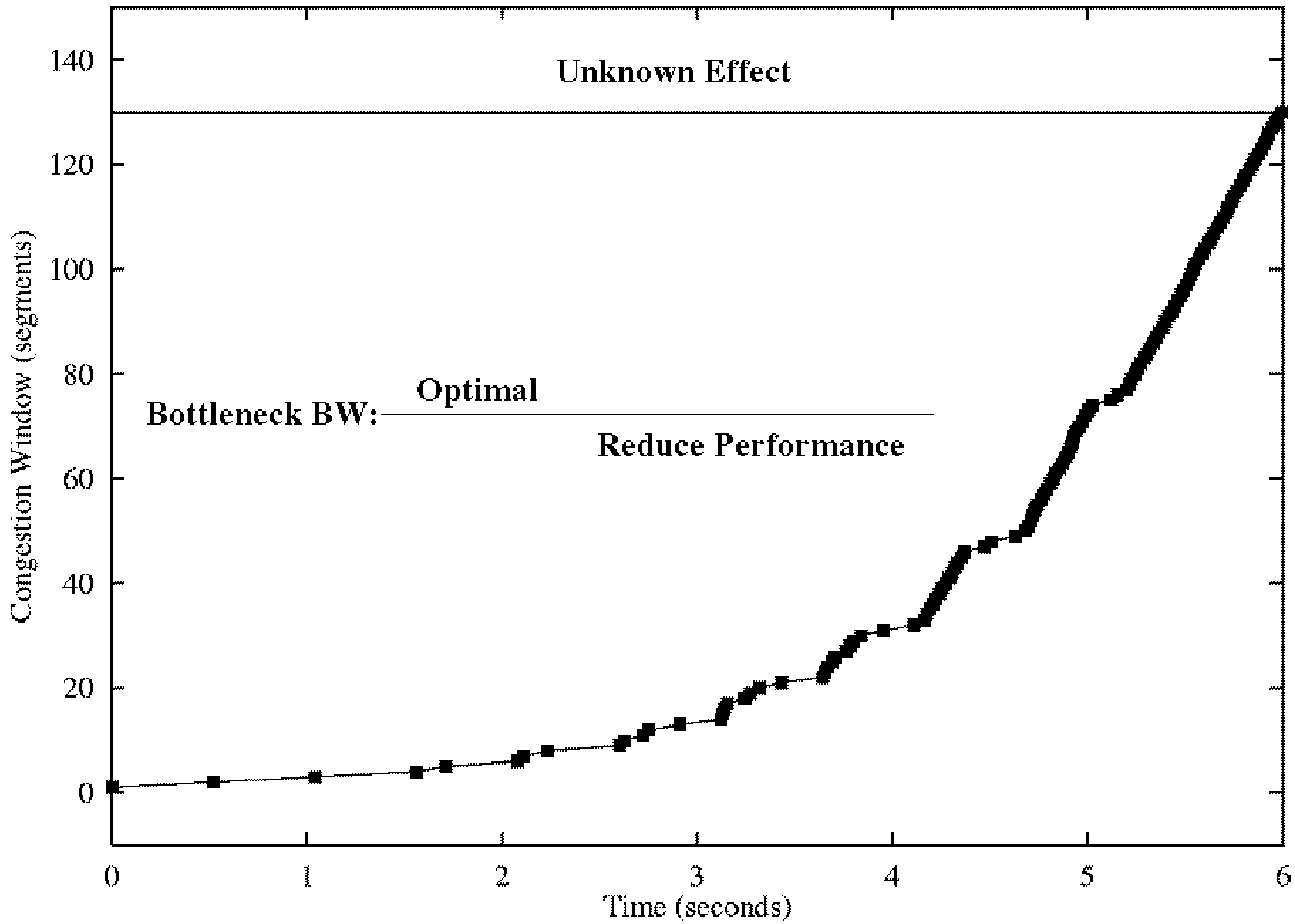
Methodology (cont.)

- Connections with loss:



Methodology (cont.)

- Connections without loss:



Tracking Slow Start Flights (TSSF)

- Sender-side, TCP specific algorithm.
- Watch for 3 ACKs within a flight and estimate from those.
- Results:

Case	No Est.	No Imp.	Prv. Loss	Stdy. State	Opt.	Tot.	Red. Perf.
With Loss	42%	1%	1%	3%	0%	4%	52%

Case	No Est.	Unk. Imp.	Opt.	Red. Perf.
Without Loss	13%	2%	2%	82%

- Major problem is underestimate of bandwidth.
  - Caused largely by delayed ACKs.

Closely-Spaced ACKs (CSA)

- Sender-side, TCP independent algorithm [Hoe96].
- Watch for n ACKs within $\nu \cdot RTT$ seconds.

Case	No Est.	No Imp.	Prv. Loss	Stdy. State	Opt.	Tot.	Red. Perf.
Loss, $CSA_{n=3}^{\nu=0.1}$	62%	20%	6%	9%	2%	17%	2%
Loss, $CSA_{n=2}^{\nu=0.05}$	53%	37%	5%	4%	0%	9%	1%
Loss, $CSA_{n=2}^{\nu=0.1}$	45%	32%	8%	10%	2%	19%	4%
Loss, $CSA_{n=2}^{\nu=0.2}$	38%	24%	9%	13%	3%	25%	13%

Case	No Est.	Unk. Imp.	Opt.	Red. Perf.
No Loss, $CSA_{n=3}^{\nu=0.1}$	24%	42%	13%	22%
No Loss, $CSA_{n=2}^{\nu=0.05}$	19%	59%	11%	10%
No Loss, $CSA_{n=2}^{\nu=0.1}$	14%	48%	11%	27%
No Loss, $CSA_{n=2}^{\nu=0.2}$	13%	34%	11%	43%

- Problems: No impact/estimate in many transfers with loss and reduces performance in many no loss transfers.

Tracking Closely-Spaced ACKs (TCSA)

- Sender-side, TCP independent algorithm [AD98]
- TCSA: Wait until CSAs converge to minimum CSA sample.
- TCSA': Observe CSAs until subsequent samples converge.

Case	No Est.	No Imp.	Prv. Loss	Stdy. State	Opt.	Tot.	Red. Perf.
Loss, TCSA	62%	14%	6%	11%	1%	19%	5%
Loss, TCSA′	70%	10%	6%	9%	2%	17%	2%

Case	No Est.	Unk. Imp.	Opt.	Red. Perf.
No Loss, TCSA	24%	25%	8%	44%
No Loss, TCSA′	27%	33%	11%	28%

- Problems:
  - TCSA underestimates often – TCSA' helps
  - Neither algorithm helps that often

Tracking Burst of Data Segments

- Receiver-side, TCP specific algorithm
- Tracks which segments will be liberated by a given ACK.

Case	No Est.	No Imp.	Prv. Loss	Stdy. State	Opt.	Tot.	Red. Perf.
Loss, $Recv_{min}$	11%	32%	6%	13%	4%	23%	34%
Loss, $Recv_{avg}$	11%	52%	10%	14%	9%	34%	3%
Loss, $Recv_{med}$	11%	48%	10%	14%	10%	34%	7%
Loss, $Recv_{max}$	11%	65%	7%	8%	8%	23%	0%

Case	No Est.	Unk. Imp.	Opt.	Red. Perf.
No Loss, $Recv_{min}$	1%	15%	2%	83%
No Loss, $Recv_{avg}$	1%	46%	23%	31%
No Loss, $Recv_{med}$	1%	45%	28%	26%
No Loss, $Recv_{max}$	1%	71%	27%	1%

- Problems: Overestimates or underestimates very often
- Winner: $Recv_{max}$ improves $\approx$ 25%, almost never impairs

Summary

- This is a hard problem!
- Receiver-side estimation a win because a lot less noisy, when compared to the sender-side algorithms.
  - And, we don't have to contend with delayed ACKs.
- But only a win for 25% of connections — compelling?

Future Work

- What about ramping up a connection's rate more quickly?
- Need to evaluate on more recent data and conduct live experiments.
- Effects of RED queueing?