

Autotuning in Web100

John W. Heffner

August 1, 2002

Boulder, CO

Web100

A Senior Thesis

- Wrote senior honors thesis on autotuning.
- Since integrated the code into the Web100 kernel. Currently in CVS, not released.
- This talk is mainly based on that work.
- My thesis can be found (in PS or PDF) at <http://www.psc.edu/~jheffner/papers/>

The Problem (in a nutshell)

- Network bandwidth is increasing exponentially, TCP is not keeping up.
- One reason: TCP queues require space proportional to bandwidth, buffers are statically sized.
 - Can use large static sizes, but not a general solution.

Goal

- Understand TCP queuing.
- Implement network-based queue sizes.
No arbitrary limits.

TCP Window

- A window is the amount of data TCP keeps “in flight” on a connection.
- TCP sends one window of data per round-trip time.
- Assume RTT constant.
- Throughput proportional to window size.

The BDP

- If full path bandwidth is used, $\text{window} = \text{bandwidth} * \text{RTT}$. This is known as the Bandwidth-Delay Product (BDP).
- Path bandwidths follow a variant of Moore's Law, increasing exponentially with time.
- Round-trip times are bounded by the speed of light.
- Therefore, BDPs and window sizes increase exponentially with time.

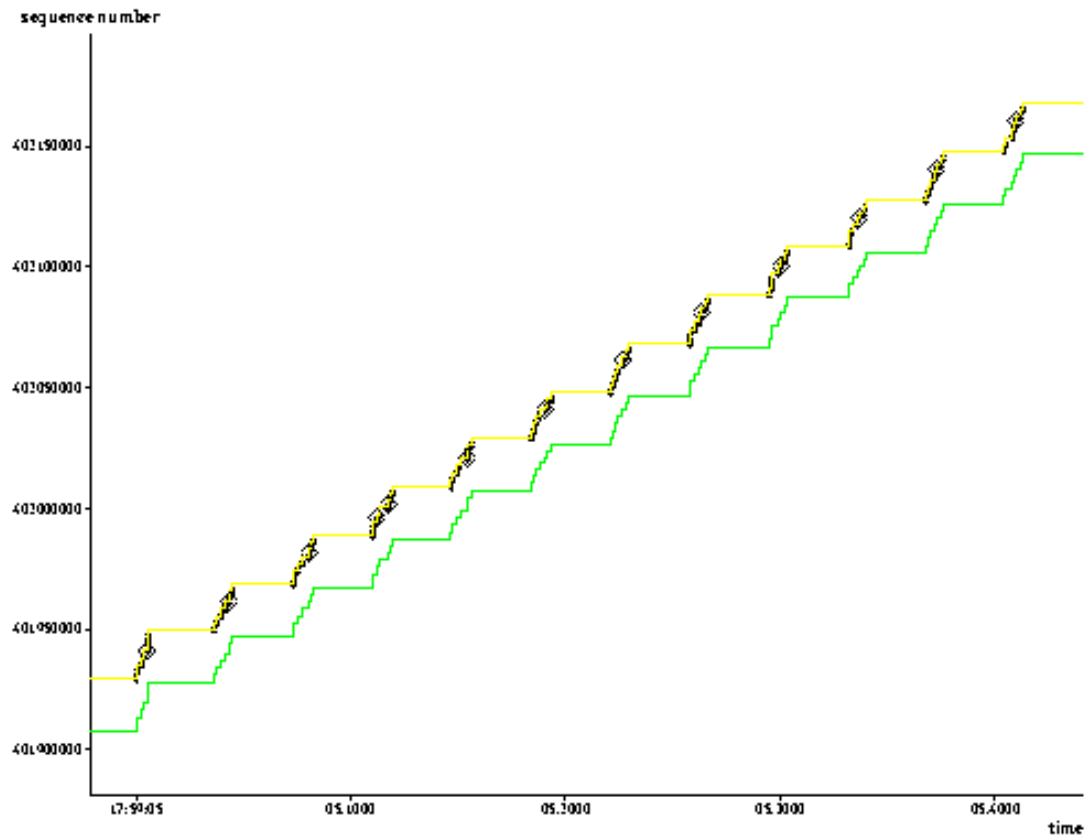
TCP Window Limits

- Window size bounded by:
 - Sending application
 - Congestion window
 - Announced receive window (receiver problem)
 - Retransmit queue buffer size (sender problem)
- Improper resource allocation may cause last two to improperly limit window.

The Receiver Problem

- The receive window is primarily a *flow control* device.
- Standard sockets-based TCPs have a per-connection receive buffer of a fixed size. Uses receive window to ensure this buffer does not overflow.
- When receive buffer is too small, throughput is limited.

Example: rwin too small



Large Receive Buffer

- Why not just set the receive buffer very large?
- Flow control will break!

Re-think Receive Buffer

- Do we need strict per-connection memory limits? Not really.
 - Note: implications for possible DOS attacks not fully explored.
- Announce window based on observed network properties, not memory limits.
- Don't have to worry about protocol and queuing overhead space.

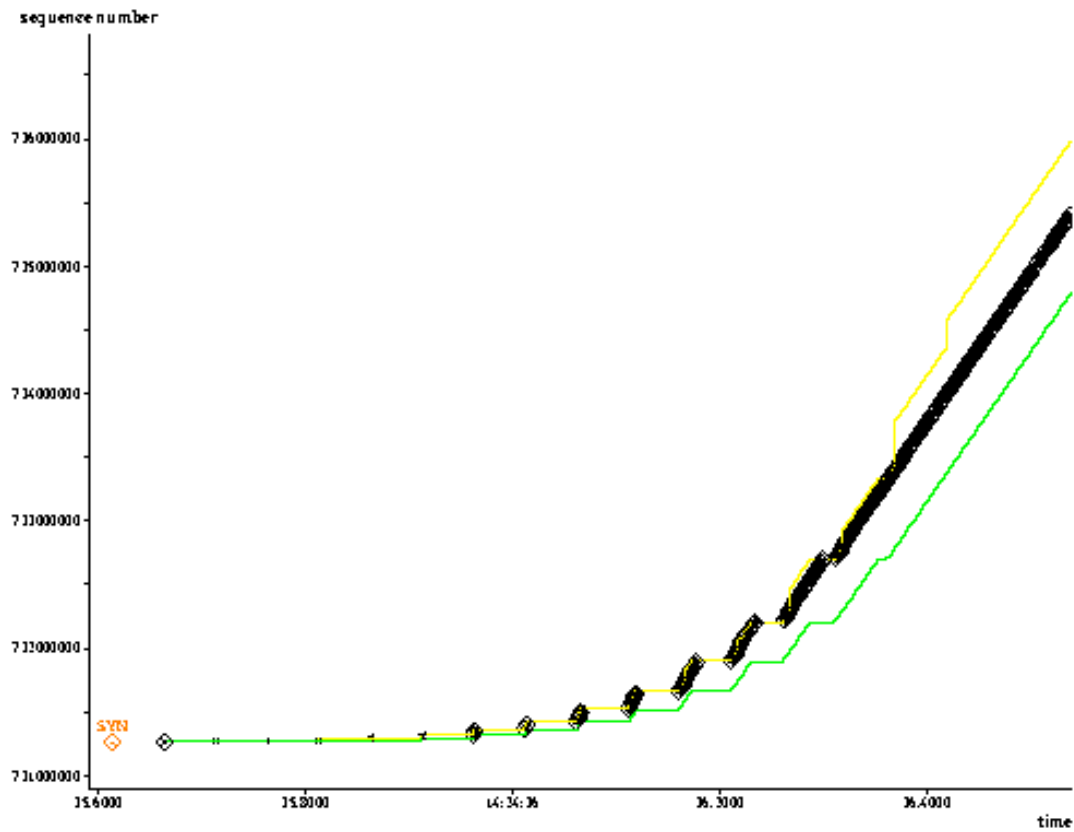
Measuring Network from Receiver

- Use the DRS algorithm [Wu Feng, Mike Fisk]
 - Measure window as bytes received in RTT, use twice this value in announcement calculation.
 - Bound RTT by measuring time taken to receive one window.
- Additional RTT measurement using timestamps.
 - May be able to track RTT variations better.

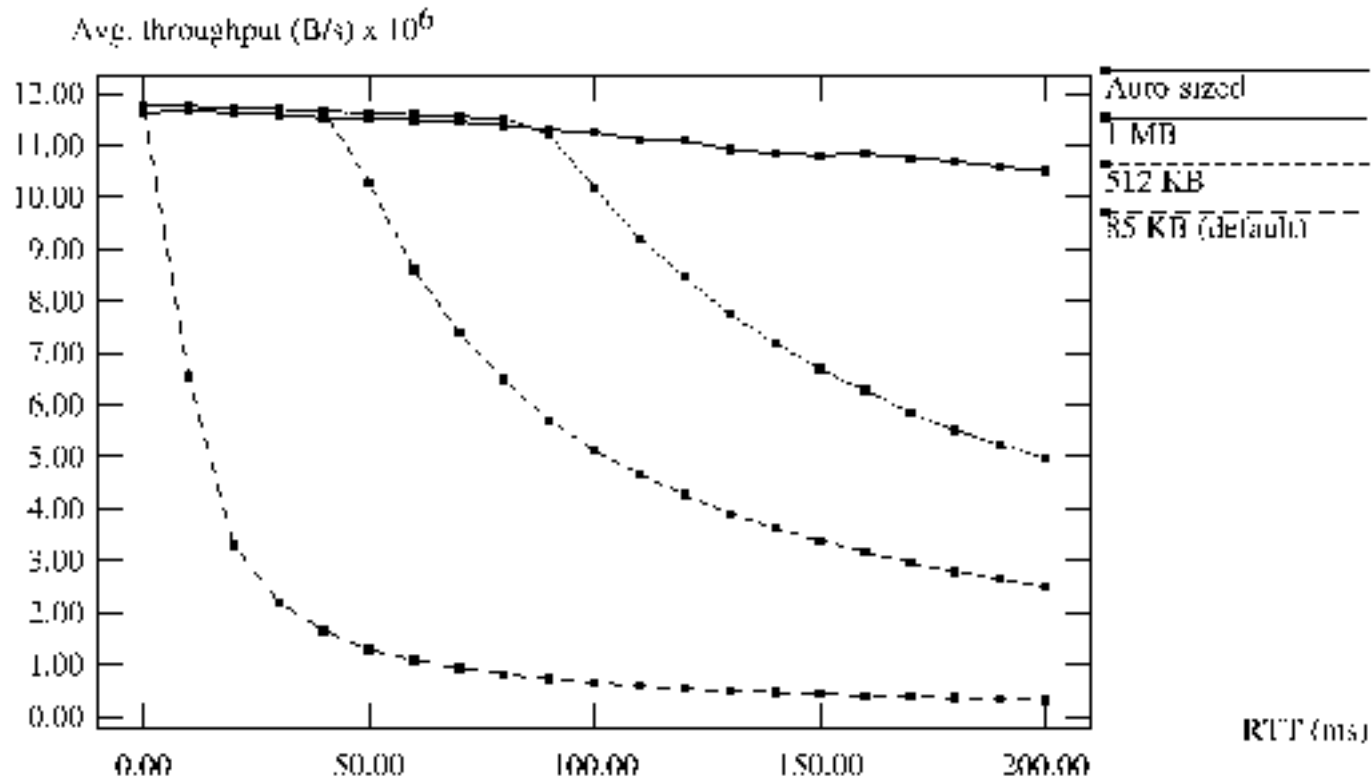
Announcing Receive Window

- New variables:
 - rcv_space: twice the most recently measured window (DRS)
 - rcv_alloc: the amount of unread in-order data queued
 - ofo_alloc: the amount of out-of-order data queued
- $rwin = rcv_space - rcv_alloc + ofo_alloc$

Behavior Trace



Receiver Test Results



The Sender Problem

- TCP must keep at least one window of data in the retransmit queue.
- Why not just use a large send buffer?
- Bulk transfer applications will fill any send buffer, and may waste a large amount of memory *per connection*.

Autotuning '98

- Semke, et. al. at PSC did original “Autotuning” in NetBSD, published in SIGCOMM '98.
- Automatically tunes sndbuf to at least $2 * cwnd$.
- Implemented in Linux 2.4, though it still imposes a (small) per-connection limit.

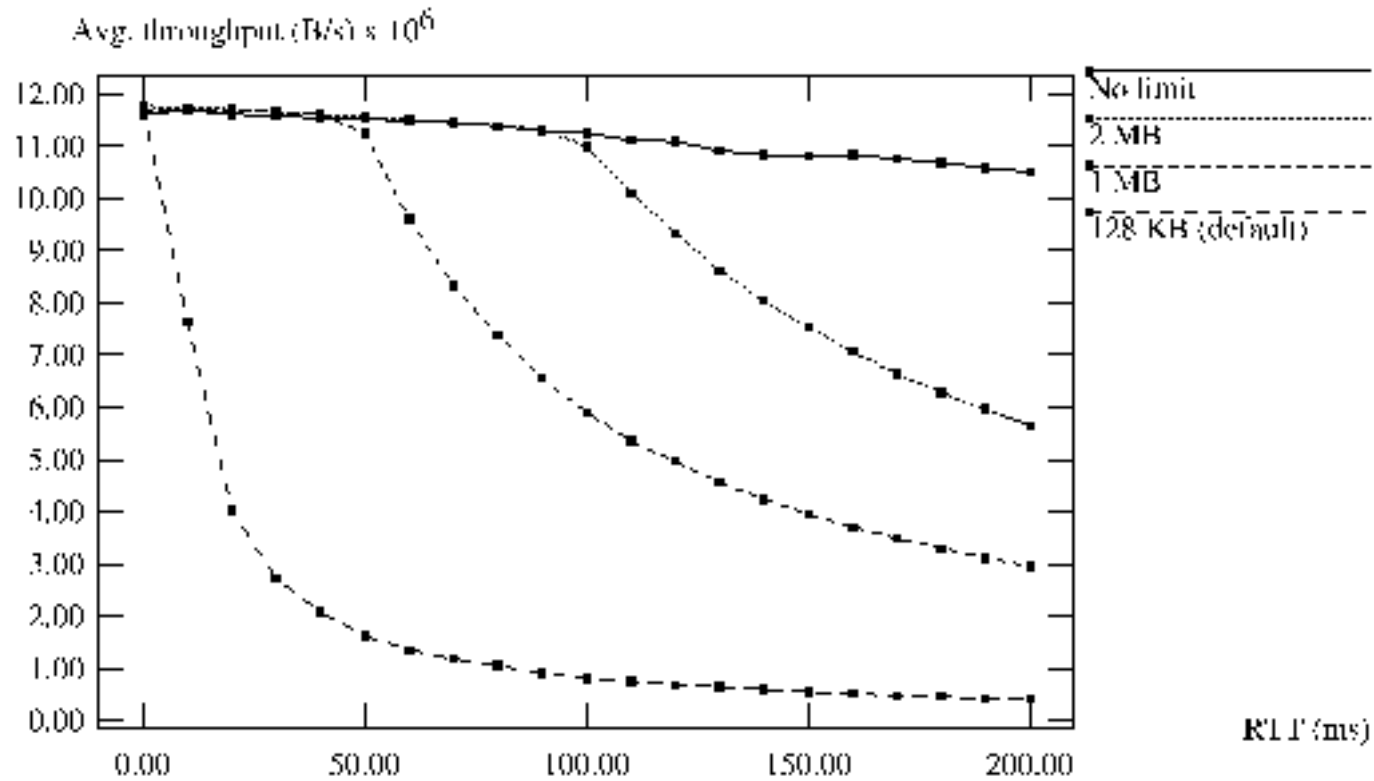
A Different Way

- Do we need a per-connection send buffer limit? Yes.
- Does the retransmit queue have to be in the send buffer? No.
- Just separate the retransmit queue, and do not limit it.

Benefits

- Don't have to worry about queuing overhead space
- Saves some memory
- Natural, simple implementation

Sender Test Results



Future Work

- Understand and defend against possible attacks.
- Demonstrate or improve robustness.
 - DRS fails when routing queue sizes vary due to other flows.