

# High Bandwidth TCP Queuing

---

John W. Heffner

Advisor, Peter Steenkiste

May 8, 2002

# The Problem (in a nutshell)

---

- TCP is the nearly universal transport protocol for reliable data streams
  - Used by web traffic (HTTP), FTP, telnet, ssh, and many others
- Network bandwidth follows Moore's law -- increasing exponentially with time
- TCP isn't keeping up with networks. The *protocol*, not the network, often unnecessarily limits performance. Why?

# The Problem (in a nutshell)

---

- One reason: TCP queues require space based on network characteristics, but are statically sized. When its queues cannot grow, TCP must slow down.
- We can increase the static sizes, but this may waste memory, and requires user expertise.

# Goal

---

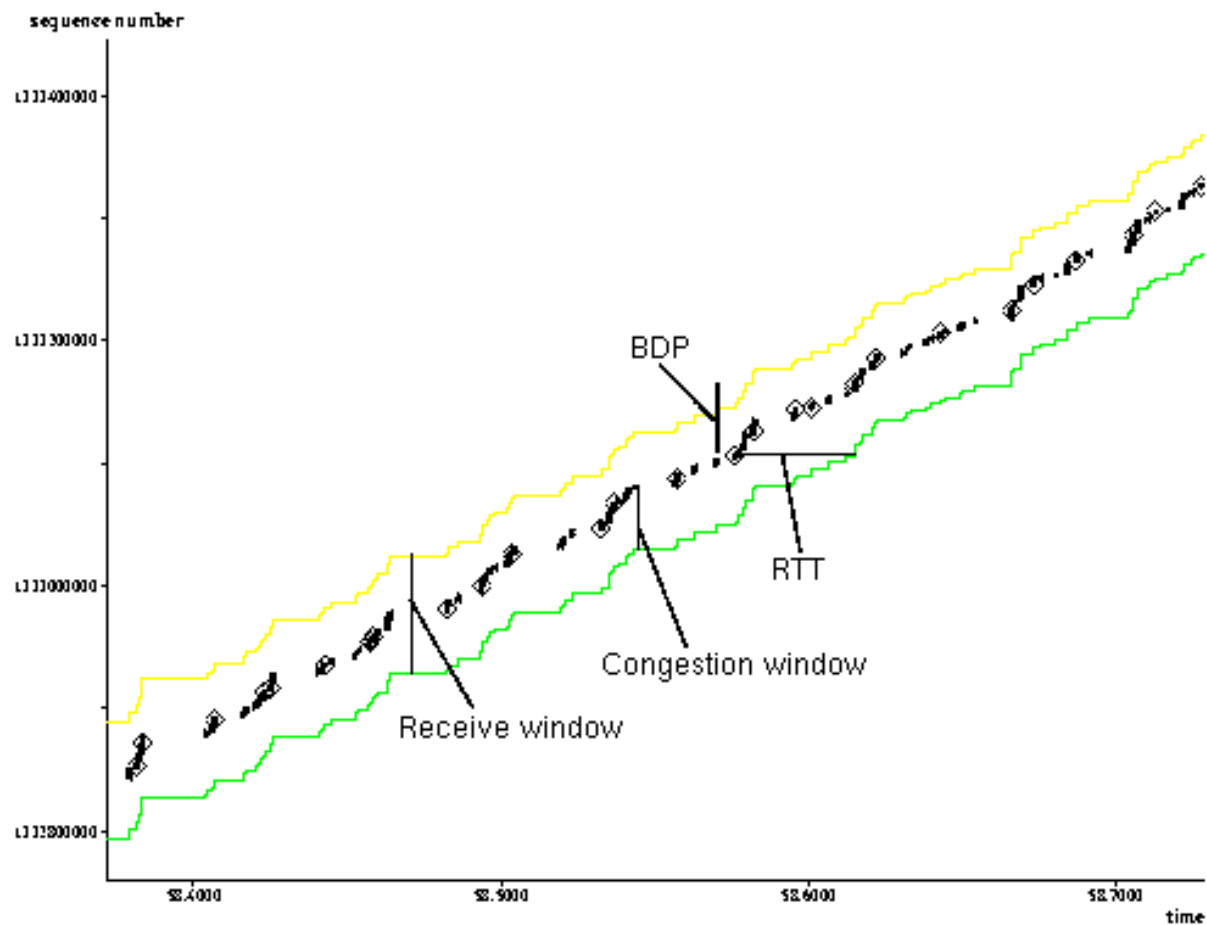
- Modify Linux 2.4 TCP so that its queues may grow as the network demands (but not waste memory).

# The BDP

---

- TCP is a *sliding window protocol*
- Windows are advanced when data sent is acknowledged. Takes 1 round-trip time (RTT)
- Amount of data “in flight” (unacknowledged) will be ideally bandwidth times delay (RTT)
- Sender finds BDP by doing *congestion control*. Maintains a *congestion window* (cwnd).

# Time Sequence Graph



# TCP Window

---

- TCP sends one *window* of data per RTT
- Window bounds
  - Sending application
  - Congestion window
  - Announced receive window
  - Retransmit queue size

# Retransmit Queue

---

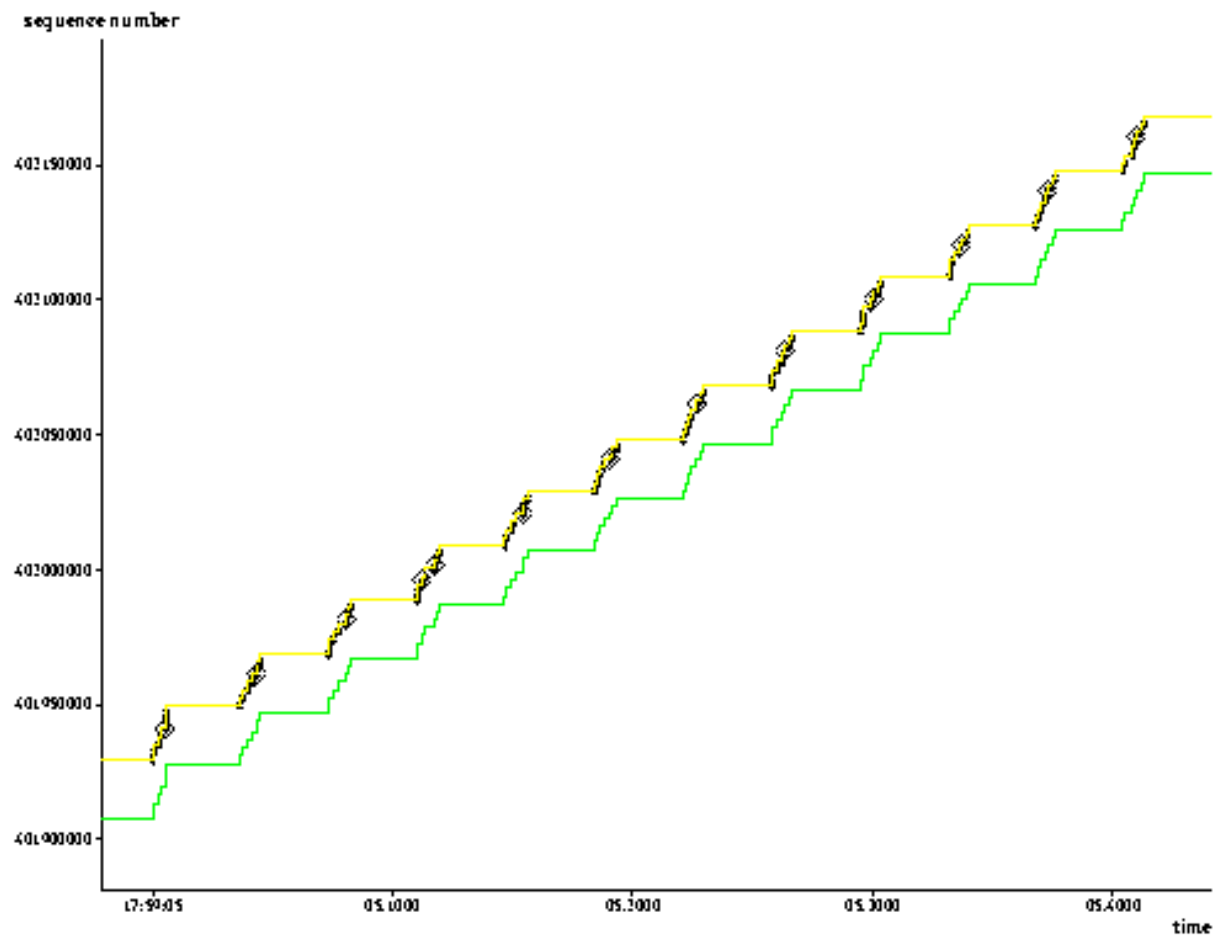
- TCP must be able to retransmit any unacknowledged data
- Queue should be one BDP in size if network limits flow
- Most TCPs keep in the socket buffer, whose size must be bounded as an application may always fill it

# Receive Window

---

- Receiving application may not keep up with the network
- Need a flow control mechanism
- Receiver announces *receive window*
  - Limits flow rate to  $rwin / RTT$
- Most TCPs have a static buffer, window announced as space remaining
- If total space is  $< BDP$ , always limits flow

# Example: rwin too small



# The Solution (Sender)

---

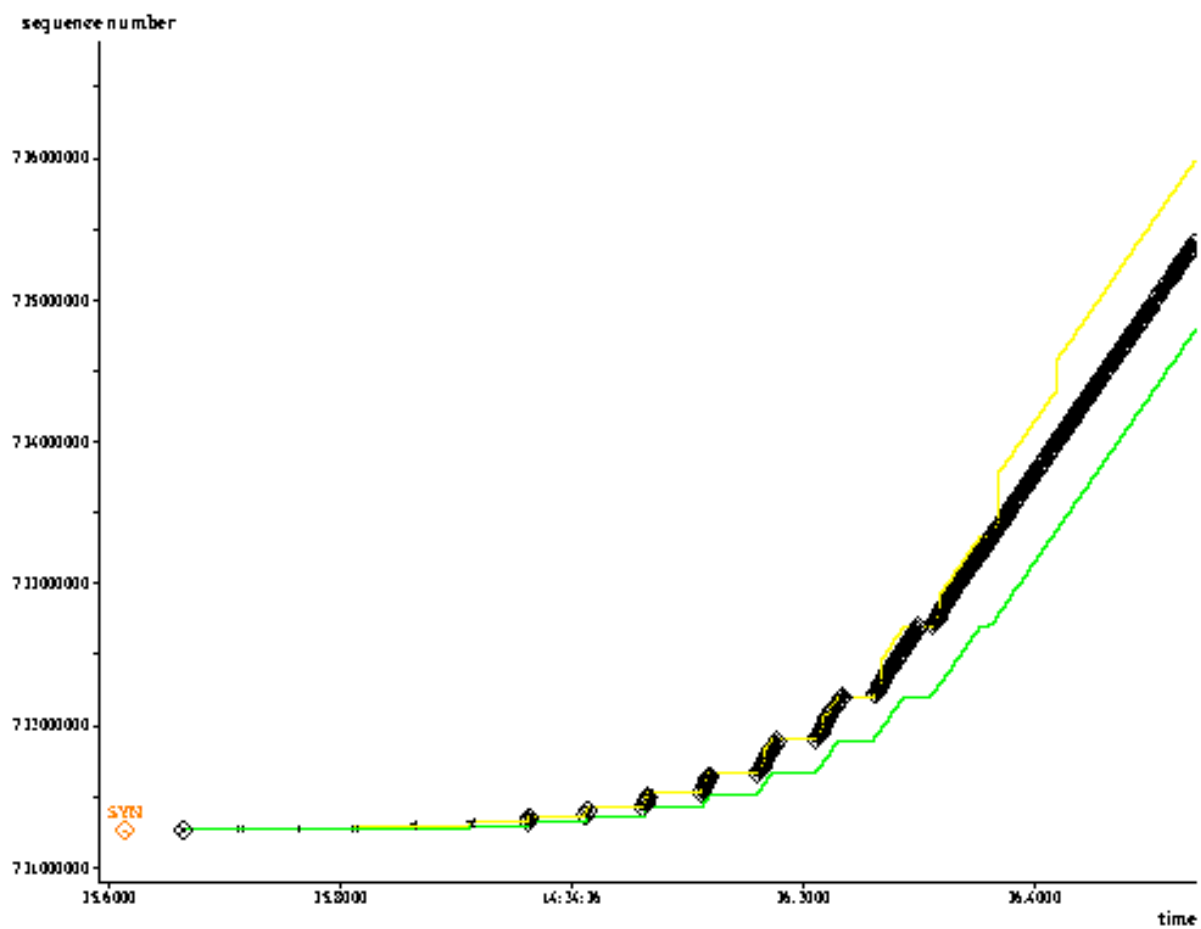
- Auto-tuning of socket sndbuf based on cwnd proposed in SIGCOM '98 by Semke, et al.
  - Wastes 1-3 BDP of memory in steady state
- There is an easier (and better) solution!
- Just keep the retransmit queue separate from the socket buffer.
  - Uses only as much memory as needed
  - Socket buffer size depends only on OS/hardware, retx queue only on network

# The Solution (Receiver)

---

- Harder problem – don't know about network already
- Very recent solution proposed by Mike Fisk and Wu-chen Feng at LANL called Dynamic Right-Sizing
  - Bounds RTT by observing time taken to send one receive window of data
  - Estimates BDP and announces rwin accordingly
- Implemented variation on this, measures RTT more accurately with timestamps
  - Timestamps not available on all TCPs, but should be available on all that have window scaling

# Solution Behavior

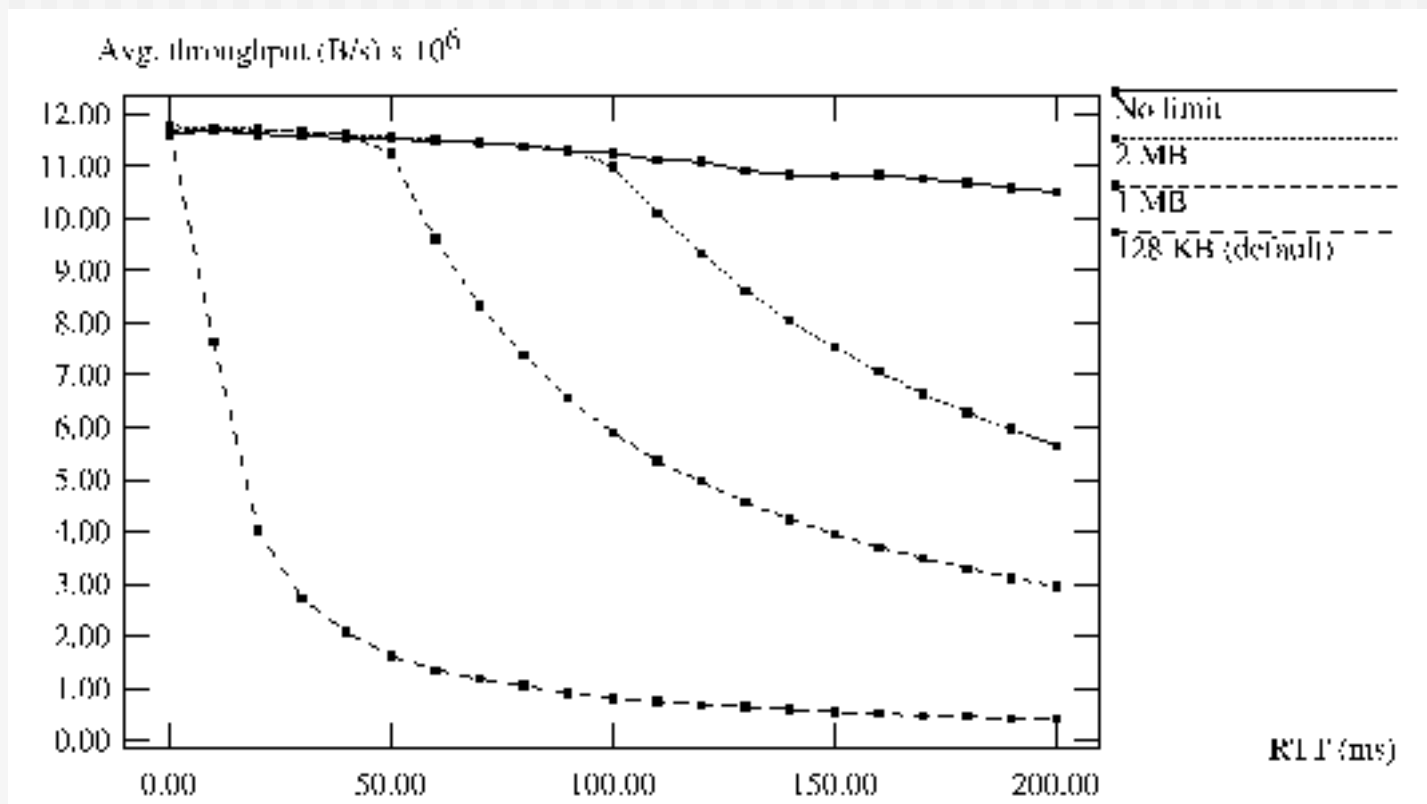


# Testing

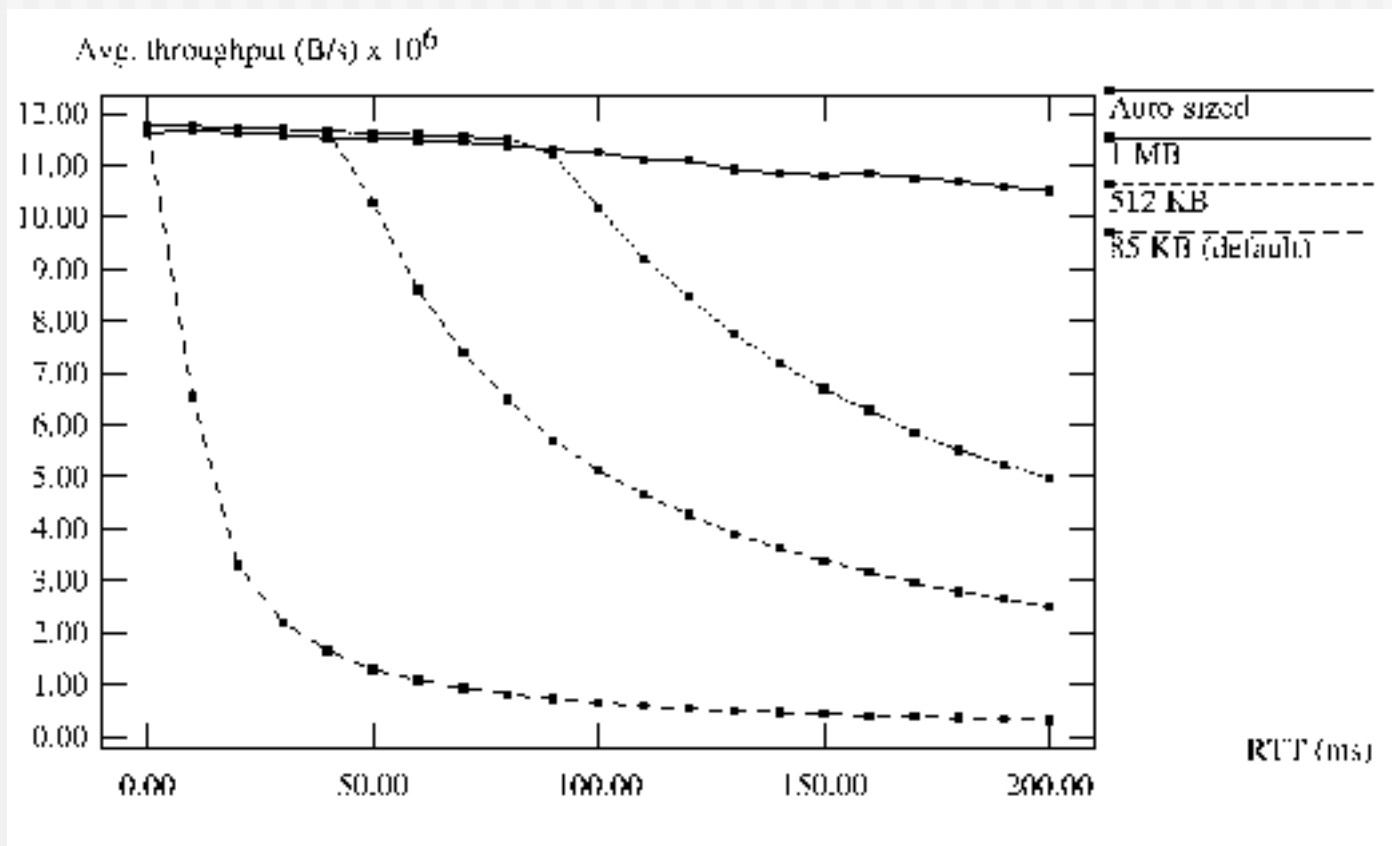
---

- Difficult. Need high-BDP path with custom kernels on both ends.
- Let's simulate high BDP instead
  - Add delay to IP routing code
  - Can't set timer for each packet
  - Drop packets into "buckets" which get emptied every clock tick
    - Causes some burstiness, not bad esp. on Alpha

# Results (Sender)



# Results (Receiver)



# “Real” Network Results

---

- Ran tests on GigE-attached machine over OC-48 link, between Pittsburgh and DC
- Able to sustain 350-400 Mb/s, 10ms RTT with both modified kernel and manually tuned default kernel
- Flow limited in both cases by congestion control mechanism

# Future work

---

- There may be DOS and memory starvation issues
- Window scale, negotiated before connection establishment, determines maximum window. May need protocol modification to be able to re-negotiate.

# Acknowledgements

---

- Matt Mathis of PSC
- Pittsburgh Supercomputing Center
- Peter Steenkiste