



Network Performance improvements in Linux 2.6

Stephen Hemminger
Sr. Software Engineer
Linux World Expo
16 Feb 2005

Overview

Background on current status

Changes in last year

- TCP congestion control
- Auto tuning
- Driver performance tuning

Network performance testing

Areas of concern

TCP over high speed links

Network adapters

- 1G and 10G Ethernet
- TCP segmentation offloading

Advanced routing and traffic control

Testing

TCP congestion control

Control Theory optimization problem

- Bandwidth over multiple connections
- Range of delays and data rates

Inputs

- Round Trip Time (RTT)
- Data in flight (window)
- Timeouts

Outputs

- Send now or not?

Bandwidth Delay Product (BDP)

BDP = amount of data in transit

Examples

- DSL/Cable modem (international)

1,000,000 bit/sec

* 1/8 byte/bit

* .5 sec/bit = 62500 bytes

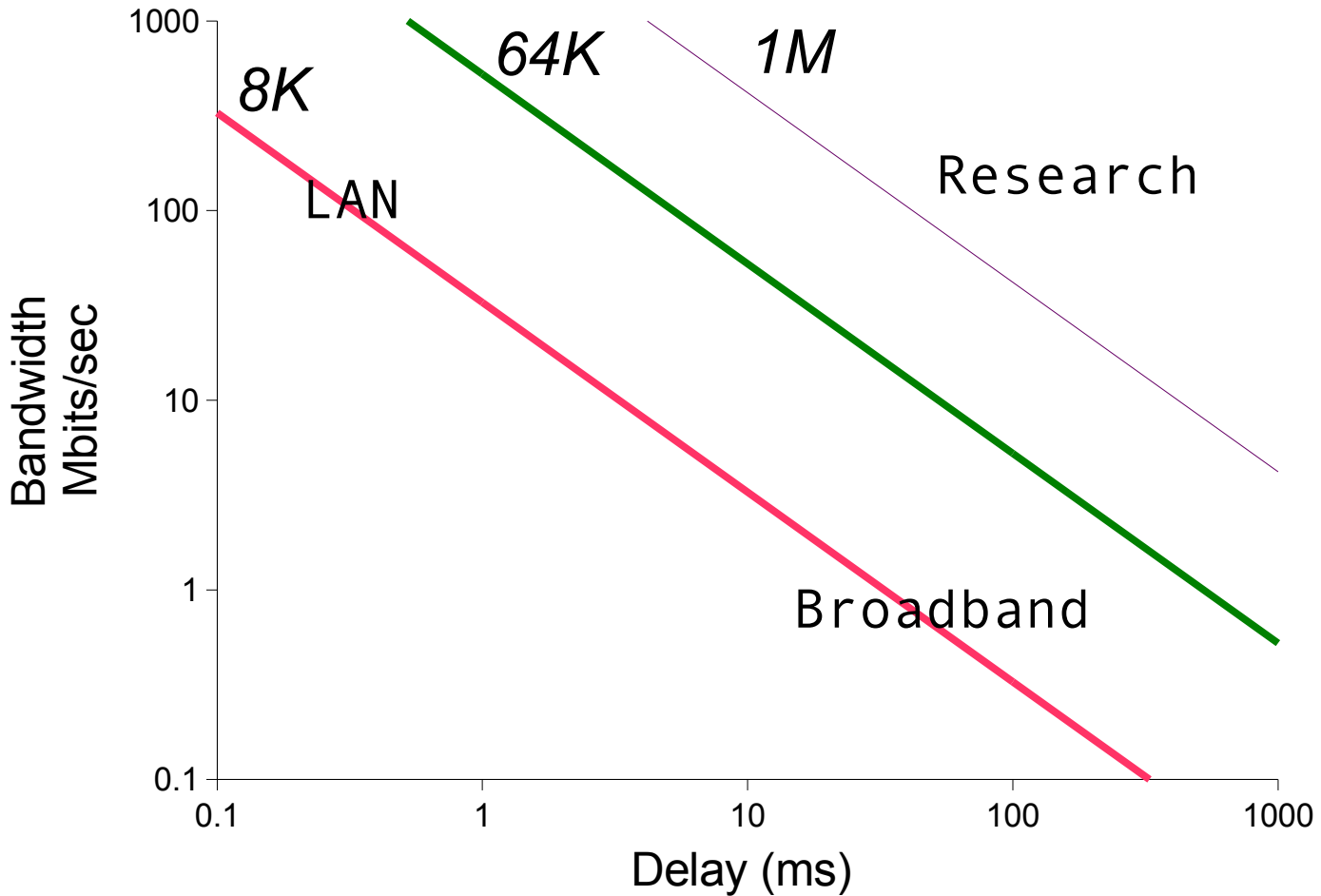
- Gigabit across US

1,000,000,000 bit/sec

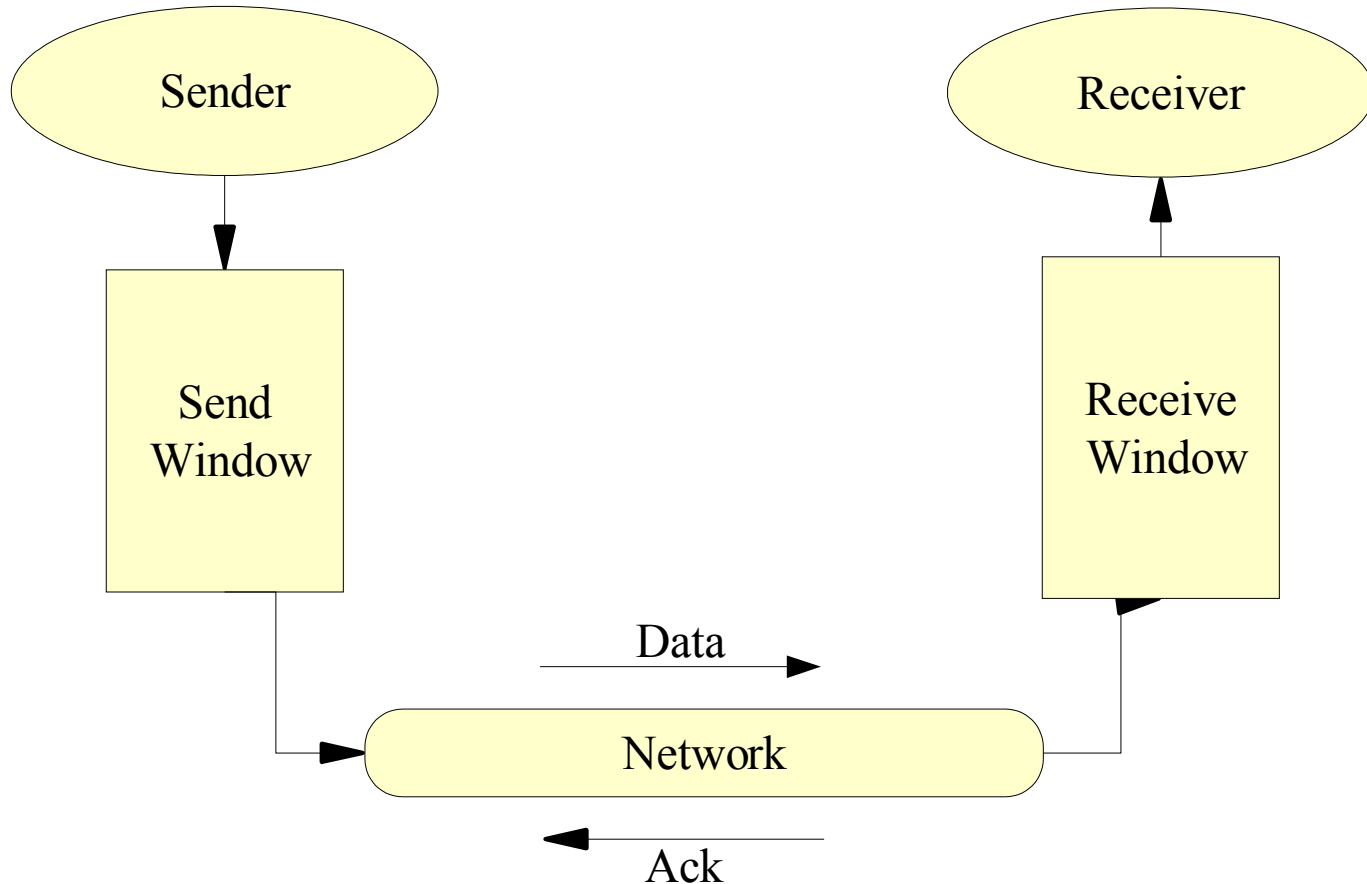
* 1/8 byte/bit

* .07 sec/bit = 8.75 Mbytes

Bandwidth Delay Product



TCP Bandwidth Delay Product



$$\text{BDP} = \text{Bandwidth (bytes/sec)} * \text{Delay (secs)}$$

Simple Test Environment

Private 1Gbit Ethernet

No firewall, filters or other software overhead

Emulate network delays

Non-invasive hooks to look at endpoint state

Make the lab look my DSL link

Existing Linux features

Supports RFC1323 TCP extensions

- Timestamps
- Selective Acknowledgment (SACK)
- Window scaling

Neighbor and route cache

Dynamic send window

Linux TCP window tuning

Send window - `net.ipv4.tcp_wmem`

- default is 4K 16K 128K

Receive window - `net.ipv4.tcp_rmem`

- default is 4K 85K 170K

Overall memory - `net.ipv4.tcp_mem`

- automatic value based on system memory

Application window - `net.ipv4.tcp_app_win`

- reserved space to handle slow applications

Receiver Tuning

Patches from John Heffner

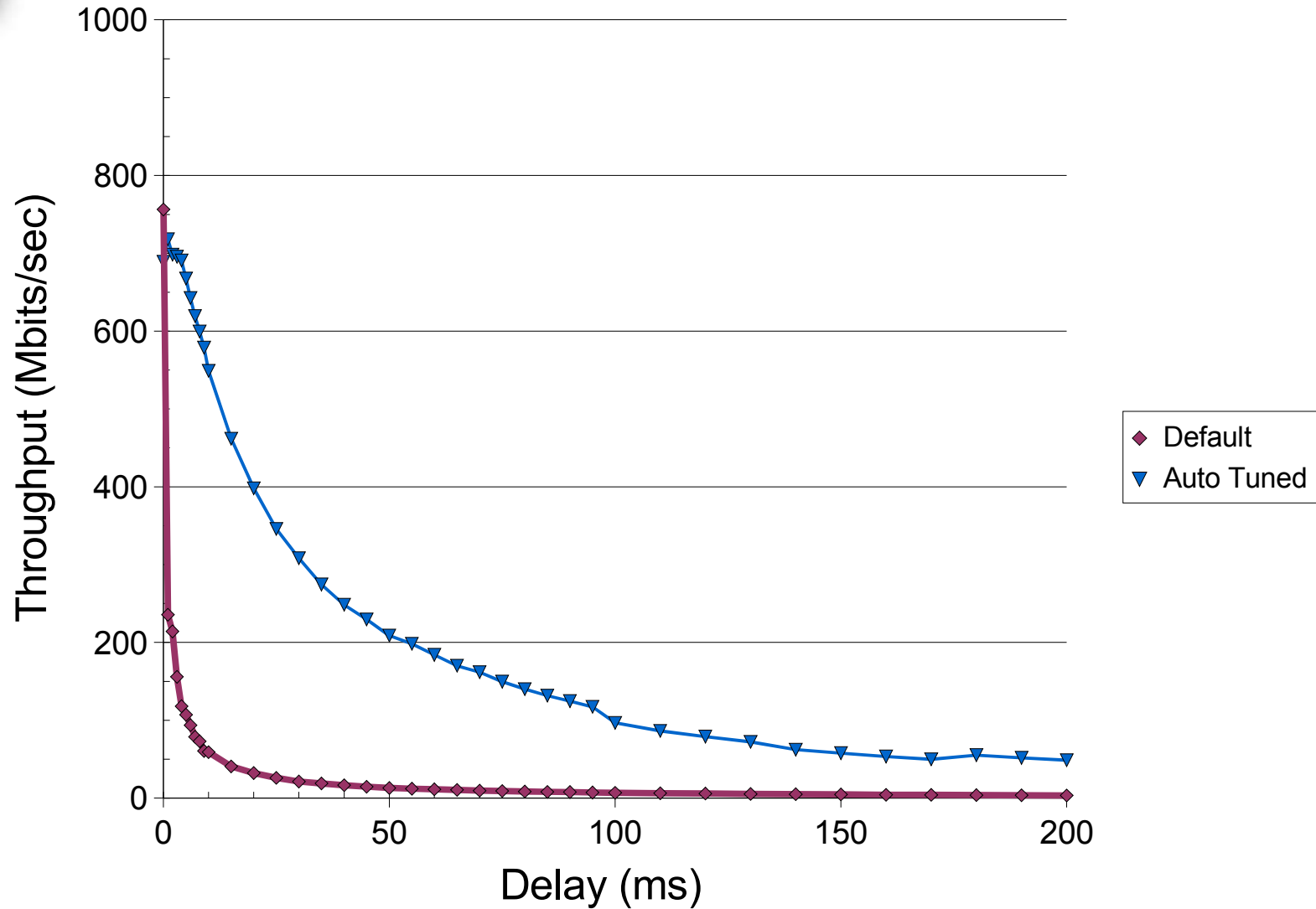
- `sysctl net.ipv4.tcp_moderate_rcvbuf`

Dynamic Right Sizing (DRS)

- adjust receive window based on RTT
- If application doesn't set window then do it for them
- Window will grow from default to max

Uses TCP window scaling

Receive auto-tuning performance



But!

Some devices are buggy

- Corrupt window scale change N to 0
- Forget to track state (OpenBSD packet filter)
- Connections will hang because initial window looks like a silly window
- .01% of the net is buggy, but Linux finds it

Linux 2.6.9 chooses window scale based on maximum possible receive window

- Default tcp_rmem => window scale of 2
- Buggy devices will see $\frac{1}{4}$ of the real window

TCP Reno

Standard default in 2.4

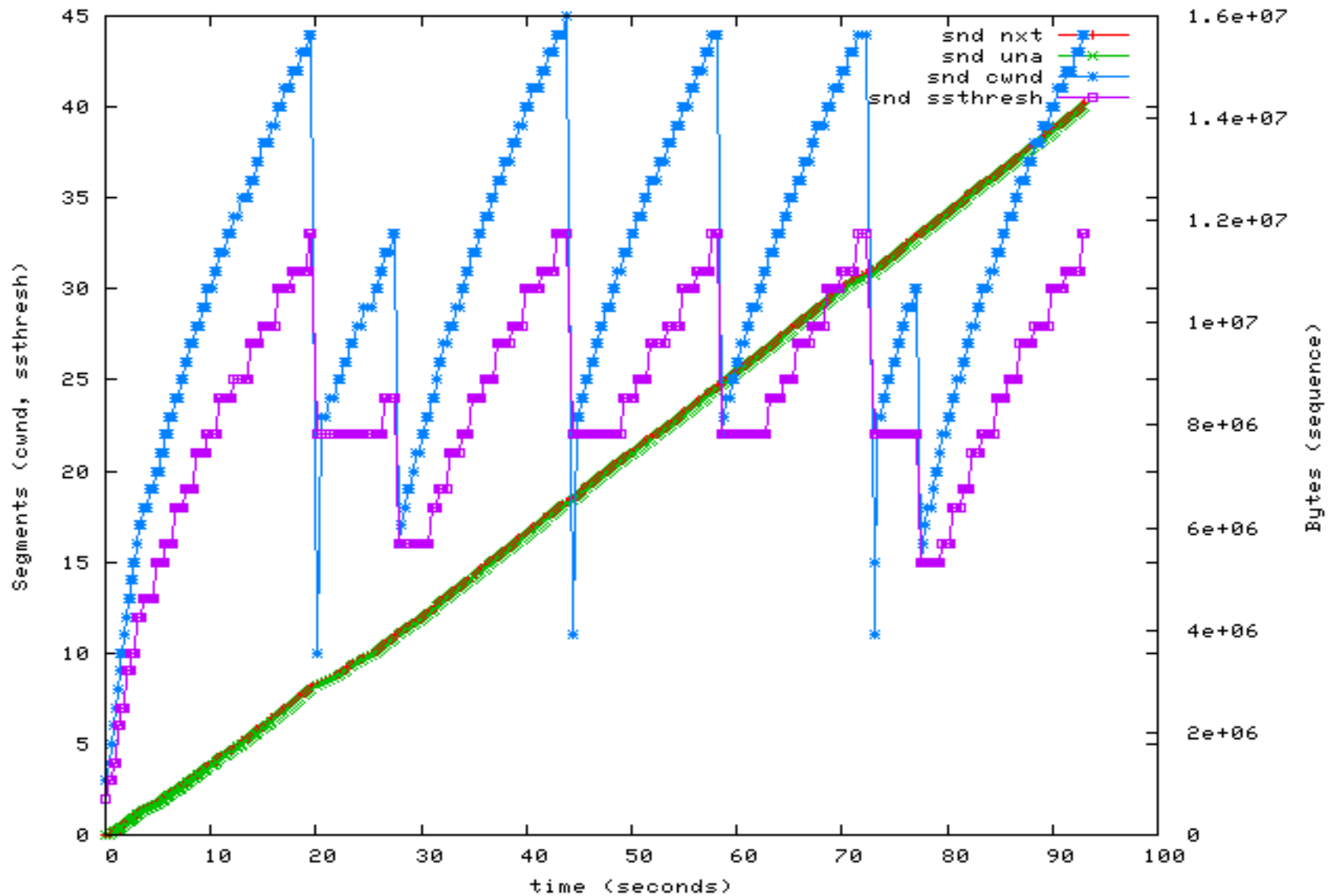
Adjusts congestion window based on packet loss

Slow start – window grows slowly

Additive Increase window on each Ack

Multiplicative Decrease on loss

Reno - sample trace



TCP Vegas

Original work by Larry Peterson

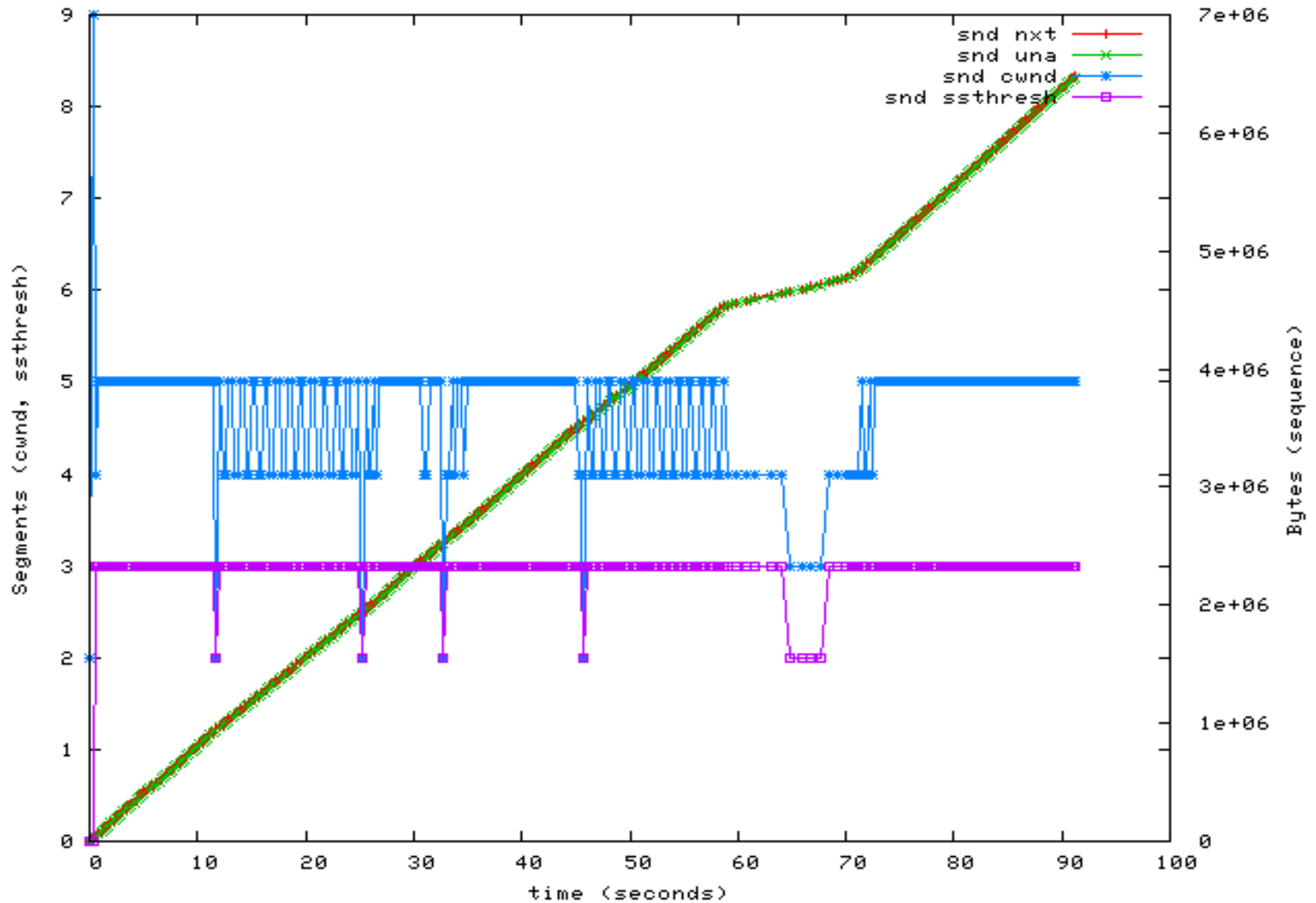
- Patches existed for 2.2, 2.4 and part of web100
- `sysctl net.ipv4.tcp_cong_avoid`

Measure bandwidth based on RTT

Adjust congestion window on bandwidth

Avoids packet loss

Vegas - trace



TCP Westwood

Work by Caludio Casetti

- Patches for 2.4 by Angelo Dell'Aera
- `sysctl net.ipv4.tcp_westwood`

Focused on wireless

- packet loss != congestion

Measure bandwidth based on RTT

Use normal Reno till congestion then adjust congestion window based on bandwidth

Binary Increase Congestion Control (BIC)

Work by Lisung Xu

- Patches for Web100 (2.4)
- `sysctl net.ipv4.tcp_bic`

Default choice in Linux 2.6

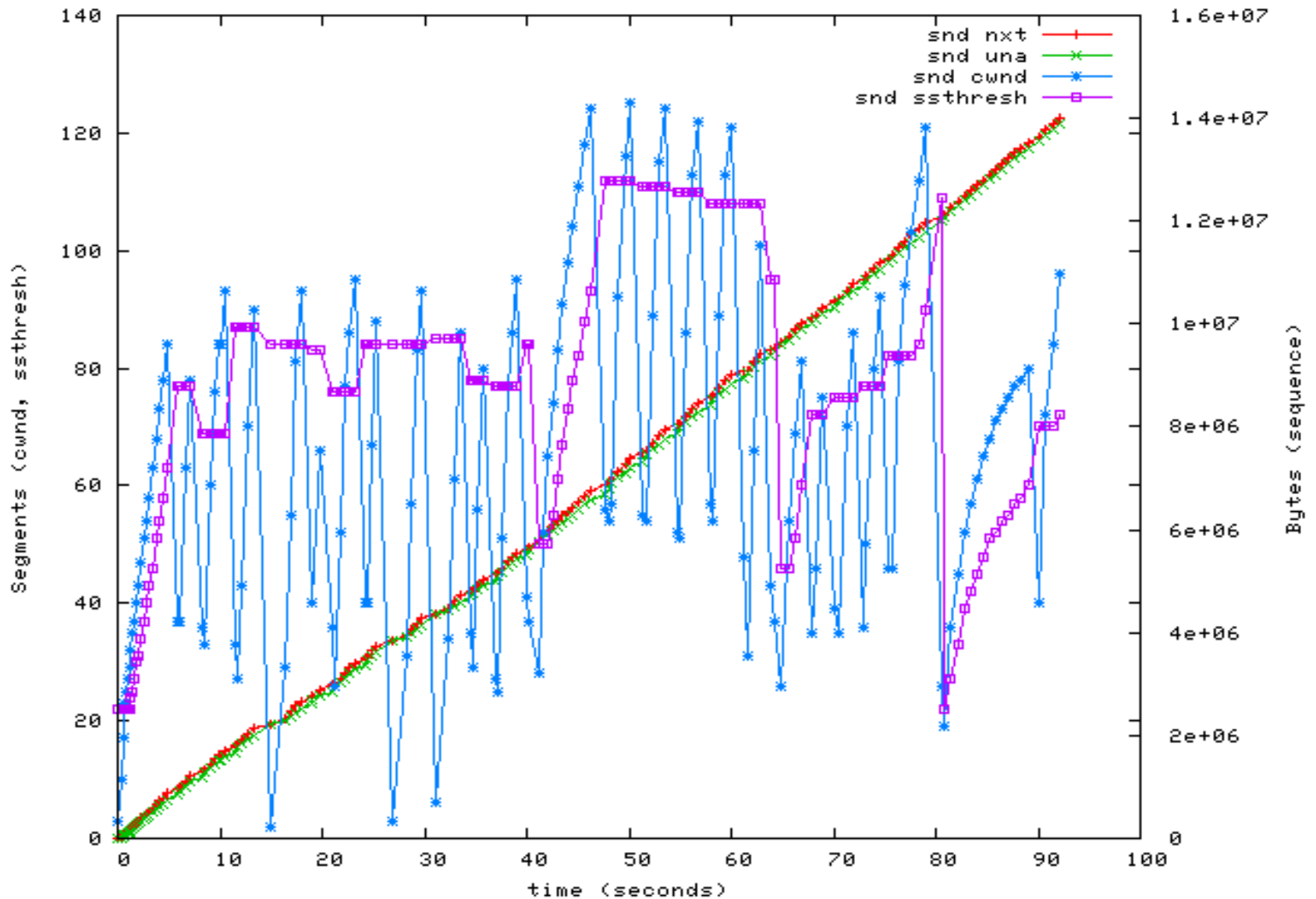
Designed for best high speed networks

Modification of Reno

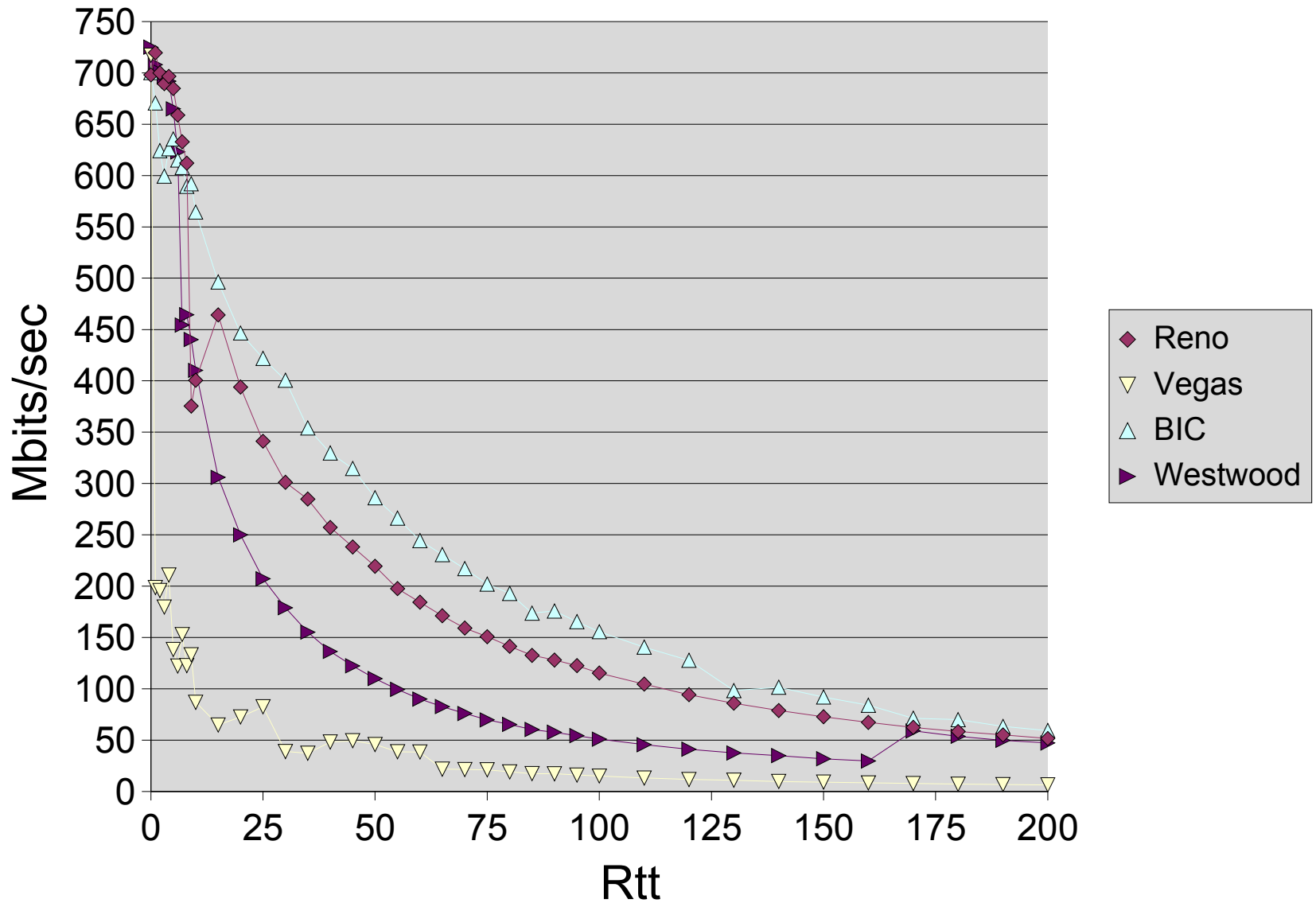
Use additive increase when congestion window is large

Binary search increase when window is small

BIC trace



Performance Comparison over 1Gbit



Special case tuning

Device driver module overhead

- TLB call overhead

Larger MTU – Jumbo Packets (9K)

- Limited to private networks

Turn off timestamps

- only when no packet loss

Bind IRQ to processor

- avoid cache line bouncing

TCP Segmentation Offload

Reduce per-packet overhead by handing large (64K) chunks to device

Implemented for several Gigabit devices

But we were cheating (until 2.6.9)

- Pretending MTU was 64k
- Not obeying TCP slow start
- Made for great benchmarks..

Linux Advanced Routing and Traffic Control

Allow implementing traffic policy

Can classify traffic several ways

Restrict bandwidth and utilization

Useful for any server environment with multiple requirements

Netem – emulating real networks

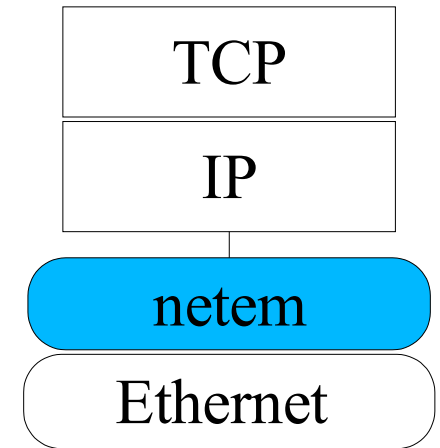
Successor to nistnet

Add delay

Simulate packet loss

Reorder packets

Use LARTC for rate control



<http://developer.osdl.org/shemminger/netem>

Network Filtering

Every host ends up doing firewalling

Rich array of policy options

Connection tracking adds overhead

Known scaling issues with lots of rules

- Average server uses 100's of rules

Network interface issues

Buses aren't keeping pace with networks

- TCP Offload processors won't help

Interrupt overhead

- NAPI helps
- Firmware interrupt moderation helps

Recent experiments with E1000 are encouraging

- 1,000,000 packets/sec (transmit only)

Still topic of active discussion

Route cache

Current architecture works well for multiple protocols

But scales poorly when going to 10,000 connections and routes

Real problem on hosts direct connected to Internet backbone

Investigating new route algorithms

Future work

TCP Selective Ack processing overhead

TCP algorithm overlap

- multiple RTT estimates
- future research

Network AIO

- reduce copies on send

Counter measures against misbehavior