

Autotuning in Web100

John Heffner
February 18, 2003

The Adgenda

- Quick review of the problem
 - Should be old news for all of you!
- Previous work
- High level description of our solution
- Delve in to the technical details

- Feel free to interrupt with questions

The Problem

- TCP's queues must be proportional in size to bandwidth-delay product
 - If queue too small, throughput proportional to queue size
- Queues kept in "socket buffers"
 - Fixed size
 - Used in other socket types as just user/kernel buffer
- Setting size too large has negative effects
 - Wastes maximum BDP of memory *per flow*
 - Breaks application flow control
- Average BDP usually used for buffer size
 - Badly hurts high performance users

Previous Work

- Some application-specific user-mode solutions
 - Out of scope
 - We are interested in general solutions

- "Auto-tuning"
 - 1998, PSC
 - Sender side solution
 - Sets sndbuf to $2 * cwnd$
 - Linux 2.4 does something similar

- "Dynamic Right-Sizing"
 - 2001, LANL
 - Post-dates Web100 funding
 - Receiver solution
 - Estimates sender's window, sets $rwin = 2 * win$

Problem solved?

- We have sender and receiver solution. All done?
- No, there is still more to do.

- Memory management
 - Even with autotuning, Linux keeps sndbuf limits.
 - We want fast fair memory sharing, no artificial limits

- Window measurement
 - DRS cannot handle paths with changing RTT

- Some details are just ugly
 - Wasting memory
 - Memory vs. sequence space issues

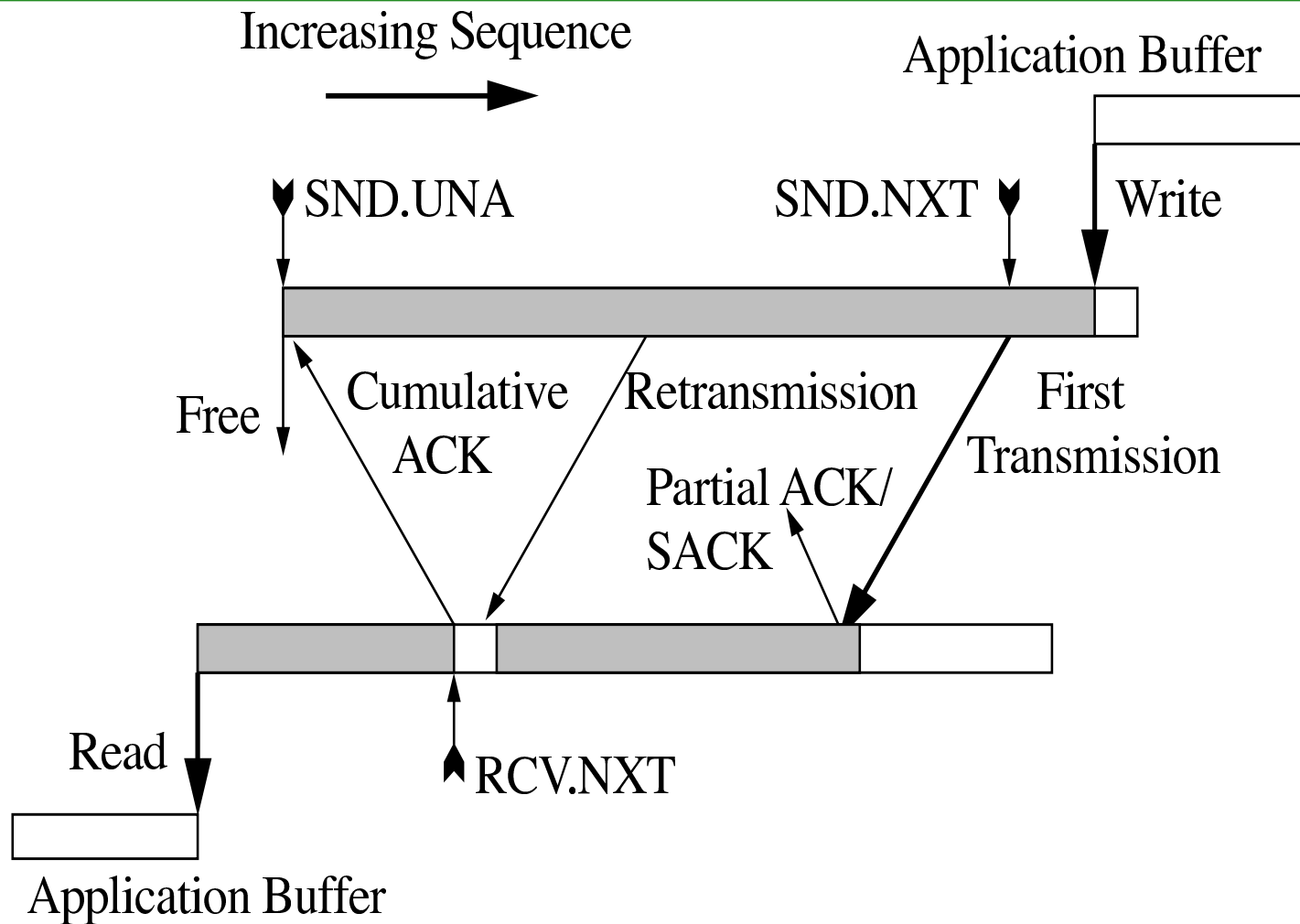
Our Approach

- Re-think socket buffers
- Create TCP memory management framework
- Keep robustness in mind
 - Want to be distributed, ON by default

Socket buffers

- Application is scheduled, network is realtime
 - Send buffer stores data until it can be sent
 - Receive buffer stores data until it can be consumed
- Current definition
 - Send buffer holds all data including retransmit queue
 - Receive buffer holds all data including reassembly queue
 - Holds all TCP data
- New definition
 - Send buffer holds only untransmitted data
 - Receive buffer space used to avoid lowering window
 - Holds only data for buffering between app and net

A socket buffer diagram



Benifits

- Buffer sizes small relative to high-RTT BDPs
 - Current sysctl values make sense
- Retransmit, reassembly queues will be *exactly* the right size (given enough system memory)
- Sender requires no tuning
- Receiver can be done with autotuning

Sender definitions

- Send queue
 - Data held prior to transmission
- Retransmit queue
 - Data transmitted but not ACKed
- Send buffer
 - Space for the write queue.

Sender behavior

- Same as current.
- When send buffer fills, write() blocks.

Receiver definitions

■ Reassembly queue

- Data received not in order (not ready for delivery)
- Current span: `rcv_hi_seq - rcv_nxt`

■ Receive queue

- Data ready for delivery, but not consumed by application
- Current size: `rcv_alloc`
- (Auto-tuned) max size: `rcv_space`

■ Receive buffer

- Space reserved for variations in receive queue length
- Size: `rcvbuf`

■ Receive window

- Flow control device: how much will the app consume?

Receiver behavior

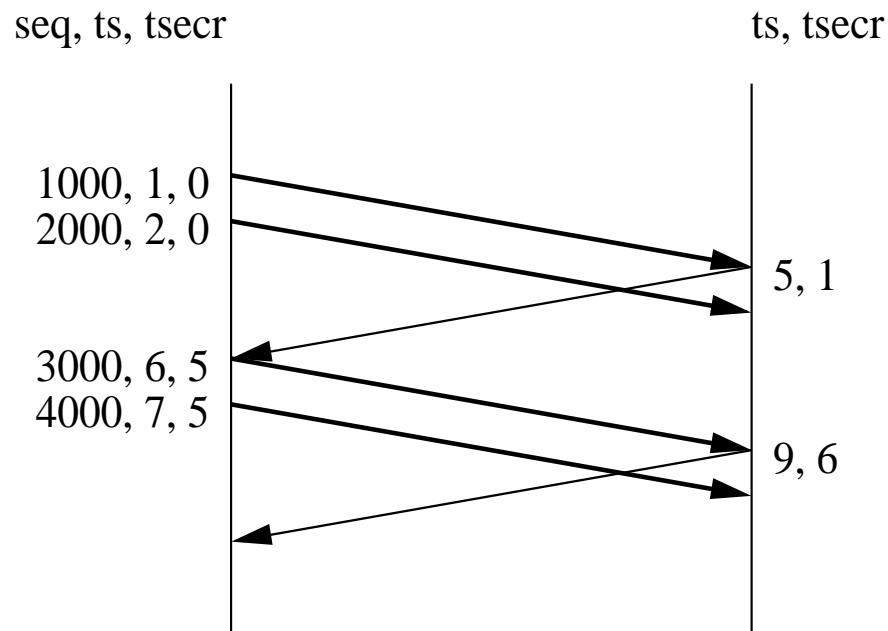
- $rwin = rcv_space - \max(rcv_alloc - rcvbuf, 0) + \min(rcv_hi_seq - rcv_nxt, rcv_space)$
- Simple case: $rwin = rcv_space - rcv_alloc$

Autotuning receive queue space

- Receiver requires some new information
- Same overall technique as DRS
 - $\text{rcv_space} = 2 * \text{used window}$
- With no timestamps, measure used window
 - $\text{MinRTT} = \min(\text{times to receive one rwin})$
 - $\text{rcv_space} = \text{bytes received in last MinRTT}$
 - Only accurately measures MinRTT when rwin limited
 - After slow start, becomes non rwin-limited
 - If RTT increased by cross traffic, will slow down

Our window measurement

- Use timestamps' alternate sequence space
 - Can match timestamp echos with a byte sequence number
 - Save ACK number (`rcv_tswin_seq`) and timestamp
 - Wait until `ts` echo comes \geq saved `ts`
 - That segment was freed by ACK if not sender limited
 - Sender window is `rcv_hi_seq - rcv_tswin_seq`



Memory management

- Not an issue in most cases
 - Should not require $\gg 1$ BDP of memory per interface
- May be an issue in some cases
 - Low memory machine
 - **Deliberate attacks**
- Sender
 - If connection using too much memory, write() blocks
- Receiver
 - Only advertise window we are willing to commit
 - Out of window packets will be thrown away
 - Keep track of total memory use, too
 - Can throw away reassembly queue if necessary

Determining the fair share

- Can be done many ways
- We chose a simple algorithm
 - Allow no one connection any more than free space
- This is NOT implemented yet

