Linux Networking: The RISE of the congestion window, the FALL of the routing cache, and the LOCALITY of packets.

David S. Miller

Red Hat Inc.

IBM Watson Research Center, 2010

Routing Cache: What is it?

- Hash table based cache of routing lookups.
- Keyed on many attributes
  - src and dest address
  - TOS
  - device index
  - etc.
- Assumes real route lookup is (relatively) slow
- Real route lookup is layered (f.e. policy routing)

Routing Cache: Problems

- Routing table to routing cache is one to many
- Entries are created in response to packets
- Prime target or focus for DOS attacks
- Mitigation strategies:
  - Secure hash keys
  - Garbage collection
- GC is very non-deterministic and hard to tune
- Routing table changes require careful cache flushing

Routing Cache: What backs it?

- Original algorithm, array of hash tables
  - One hash table per prefix length (0 –> 32)
  - Not the most optimal, but routing cache makes this OK
  - Relatively simple
- New algorithm, LC-trie
  - Multi-dimensional trie
  - Close to what's known to be optimal
  - Complicated
  - Performance tied to trie balancing heuristics

Routing Cache: Barriers to Removal

- Mainly performance, cache handled DOS attacks better
- No longer true after Eric Dumazet's work
- Handling of metrics
  - Move to existing inetpeer cache
  - Issues of metric granularity
- Storing of route lookup "result"
- IPSEC and route stacking
- And again, performance...

Routing Cache: Side Topics

- What does BSD do?
  - Uses a patricia tree.
  - Clones are created for specific routes.
- What does our IPV6 stack do?
  - See BSD above.
  - But with support for source address keying.
  - Thus two tiered tree layout.

TCP CWND: How to Kill the Internet

- CWND == congestion window
- Ironically by keeping things as they are now.
- Initial CWND has stayed constant for more than a decade.
- Meanwhile net capacity has increased dramatically.
- Current situation is a bit of a joke.

TCP CWND: Short Tutorial

- Connections start with an initial CWND
- Increased until loss is detected
- CWND is reduced at loss events
- Process repeats
- Critical aspect: aggressive probing of network capacity

TCP CWND: The Big Myth

- That we actually have an initial CWND
- Actually there is no real limit
- Applications can have as large of one as they want
- Openning up several connections at once
- N connections == “initial CWND X N”

TCP CWND: Google's Proposal

- "draft-hkchu-tcpm-initcwnd-00"
- Increase initial CWND to 10 packets
- Most web objects do not fit into existing initial CWND
- With 10 packets, most will fit
- Works well with technologies such as SPDY

TCP CWND: Knee Jerk Reactions

- This increase will cause congestion collapse
  - FALSE: Congestion avoidance still at work
  - TCP will still back off in the event of loss
- It will hurt clients with smaller pipes
  - FALSE: Smaller pipe end hosts get better performance
  - The key is ACK clocking and how fast recovery works
  - 3+ ACKs are necessary to trigger fast recovery
  - With old initial CWND that never happened at start

TCP CWND: ANYWAYS...

- Linux will adopt larger initial CWND real soon
- Nothing IETF can do about it (sorry Chicken Little the sky is not falling)
- You heard it here first

LOCALITY: SYSTEM HIERARCHY

- Welcome to the NUMA world
- Memory “distance” matters more than ever
- No longer a quaint optimization for “huge” servers
- NUMA is pervasive even on desktops
- Heck, even laptops...

LOCALITY: MULTIQUEUE NETWORKING

- Old systems, single RX queue, single TX queue
- Limited by event signaling in old PCI
- Welcome PCI-E and MSI-X interrupts
- Networking cards beging to have multi-Q functionality
- Now it's pervasive

Locality: Linux Support for Hardware Multi-Q

- Stephen Hemminger's NAPI split-up work
  - Pull NAPI state out of struct netdev
  - One NAPI instance per HW interrupt source
- Making TX path multi-Q capable
  - Pull queue flow control state ouf of struct netdev
  - Dealing with qdiscs.... ugh...
  - Only simplest qdiscs are fully multi-Q
  - Complex qdiscs force synchronization at the qdisc
  - In the future token based qdiscs (SFQ, etc.) can be multi-Q too
  - Hierarchical qdiscs fundamentally cannot (HFSC, HTB, etc.)
  - Create new multi-Q qdisc for high level flow management

LOCALITY: SOFTWARE MULTI-Q

- Use software facilities to implement multi-Q
- CPU cross calls and packet processing job placement
- Why even bother?
  - Lots of non-multi-Q capable hardware out there
  - Hardware multi-Q is stateless (as it should be)
  - Software schemes provide more flexibility
  - Posibility to optimize for application locality
- Initially I was against.
- Happily, Tom Herbert was able to convince me.

LOCALITY: STAGE ONE: RPS

- Receive Packet Steering by Tom Herbert
- Stateless flow seperation
- Perfectly mimicks hardware multi-Q on RX
- Each hardware RX queue has a configurable cpumask
- Packets received on RX queue hash to CPU in that mask

LOCALITY: STAGE TWO: RFS

- Receive Flow Steering, again from Tom Herbert
- Hash table of flow to CPU mappings
- Dynamically updated
- Kernel spies on application I/O calls
- CPU of I/O call becomes flow CPU mapping
- Table is sized and enabled via sysctl
- Issue: out-of-order packet delivery avoidance

LOCALITY: STAGE THREE: XPS

- Tom now gives us Transmit Packet Steering
- Transmit side locality
- Maps cpus to transmit queues, reverse of RPS
- Data structure locality
- Likelyhood packet free happens near sending thread
- Eric Dumazet's Transmit Completion Steering patch

LOCALITY: FUTURES

- Hardware assist for RPS/RFS (Ben Hutchings)
  - Test patches exist for SFC chips
  - Makes use of on-chip flow table facilities
- Having lots and lots of hardware queues
  - Negative matching for things like GRO
  - Steering “queues” themselves instead of flows
- Better defaults (all SW stuff off by defualt at the moment)

The End

- Thanks to:
  - Erich Nahum and IBM Watson Research Center
  - Oren Laadan
  - Stephen Hemminger
  - Eric Dumazet (AKA: The Networking Ninja)
  - Ben Hutchings
  - Tom Herbert and Google
  - Linus Torvalds