Routing Cache: Just Say No

David S. Miller

Red Hat Inc.

Toronto, Ontario, CA, 2011
Semantically Could be Done Now

- Metrics separated into inetpeer cache
- On-stack flow key handling streamlined
- struct rtable minimized
- More could be done for ipv6
REMAINING OBSTACLES

- Performance
- Nothing else
Basic Issues

- Refcounting atomics
- Way too abstract
- Several levels of indirection
- "overflow" policies ridiculous
- Often serves more as a BUG_TRAP()
- Neighbour handling could be much simpler
RCU BASED REF-LESS NEIGH

- Routes do not refer to neighbour entries
- Neigh is found at ip_finish_output
- Within RCU section
- Demux needs to be trivial and fast
FRONT END BECOMES IPV4 SPECIFIC

- No more abstract OPS and method calls
- Only one "protocol" sits in table
- Neighbour hash and demux are inlined
- Simplify from jhash
- Remove neigh table limits
**Another Approach**

- Other idea is to put neigh info into inetpeer
- Brings us back to atomic refcount cost
- But this inetpeer cost already exists
The Problem

- Routes will cover many destination addresses
- So routes can’t refer to specific inetpeer
ROUTE_OBJECT

- Contains two pointers
- One to dst_entry
- One to inetpeer
- Problem of refcounting cost, again
Why Bother?

**WHY GO THROUGH ALL THIS TROUBLE?**

- Allows cached entries to live precomputed in route table
- Routes become identity’less
- All identity info obtained via flow key and other means
- Arguably this is just pushing costs to a new place
**Performance Goals**

- Output route lookup < twice as expensive as rtcache
- In return, lookup perform == deterministic
- Also, multihop routing behaves properly
- Input route lookup is another issue entirely
Why Bother?

**INPUT ROUTE LOOKUPS ARE EXPENSIVE**

- Up to 3 routing table lookups
- RP filtering is evil
- BSD does this in firewall layer
- Also provides saddr selection
- Ideas needed here