Removing The Linux Routing Cache

David S. Miller

Red Hat Inc.

Columbia University, New York, 2012
1. Linux Maintainership

2. Networking: An Introduction

3. Linux Routing Cache

4. Removing the Routing Cache

5. The End
My Background

- Started working on the kernel 18+ years ago.
- First project: helping with ELF executable support
- Second project: Porting Linux to SPARC cpus
- Third project: Taking over Networking maintainence
- Can be overwhelming at times.
What exactly do you do?

- Like being an editor for a major publication
- Get to write my own articles from time to time
- Majority of time is spent reviewing work of others...
- ... and putting out the occasional fire.
- Merging, lots and lots of merging.
- Strong benefits from tools such as GIT and patchwork.
GIT

- Distributed version control system
- Originally written by Linus Torvalds
- Full disconnected operation (unlike CVS/SVN)
- Buildup of trust networks, who I pull from
- Side topic: bisection
- Contributors don’t have to use GIT ...
- ... but for heavy contributors it helps a lot.
- GIT is used for many projects other than the Linux kernel
- GIT can be used on top of other source code control systems
Patchwork

- Web based patch tracking system
- Patchwork is subscribed to various developer mailing lists
- It recognizes software patch submissions and GIT pull requests
- Maintainers can manage as objects in various states
- Allows maintainer as well as submitted to see a changes state
Computer Networking

- A collection of computers
- “interconnected” by communication channels
- Allows the sharing of resources or information
- A group of devices connected to each other
Protocols

- Prearranged method for successful communication
- Precisely defined message formats
- Signaling
- Authentication
- Error detection and correction
- Syntax, Semantics, Synchronization
Networking from an OS Perspective

- Drivers, Protocol Implementation, System Calls
- BSD “Sockets”
  - De-facto interface for network programming
  - Created in 4.2 BSD Unix
  - socket()
  - bind(), connect(), listen(), accept()
  - send(), recv() (read(), write())
  - close()
- Drivers send and receive packets “on the wire”
- Most low level protocols implemented in the kernel
Internet Protocol

- Successor to the ARPANET
- A DARPA project funded by the US Defense Department
- Uses an addressing system of 32-bit “IP Addresses”
- Transport Protocols include TCP, UDP, and SCTP
IPV4 Routing

- Routers implement “store and forward”
- Move packets to their final destination
- Decisions are made using a “routing table”
- Route selection uses “longest matching prefix”
- Static Tables vs. Routing Protocols
Routing Example

- Two routes:
  - 172.168.82.0/24 “A”
  - 172.168.0.0/16 “B”
- Lookup for 172.168.82.6 would match “A”
- Lookup for 172.168.2.2 would match “B”
- The “default” route 0.0.0.0/0
Hierarchy of Tables

- Cache is implemented as a hash table
- Cache is populated by lookup requests
- Information for cache entries comes from “Forwarding Information Base” or FIB
- FIB is populated by administrator or routing daemon
- Cache is therefore dynamic, whereas FIB is relatively static
- Cache lookup is simple and extremely low latency
- FIB lookups are more complicated and thus more expensive
- FIB tables implemented using trie data structure (LC-Trie)
Features Largely Unique to Linux

- Routing by source address ("routing realms" or "policy routing")
- Routing by Type-of-Service ("TOS based routing")
- Routing by firewall marking ("MARK based routing").
- Front end is called "routing rules"
- Initial lookup finds a matching "rule"
- A "rule" leads to a "routing table"
- "routing table" holds routes by destination address
Source Address Validation

- We receive packet from address “A” on interface “I”
- If I were to route a packet to “A” ...
- ... would that route use interface “I”?  
- Nearly useless on end-hosts.  
- Very useful at points of egress, routers, and firewalls.  
- Can add up to two route table lookups per route lookup.
Routing Cache Keying

- Hash table keyed on all possible route attributes:
  - Destination Address, Source Address
  - Type-of-Service
  - Firewalling MARK
  - Security Context
  - Outgoing or Incoming Interface

- Entries therefore have a high level of granularity
Population of Routing Cache

- Created on demand by lookups
- Every packet sent causes a lookup
- Every packet received causes a lookup
- Anyone can send us packets
- Anyone can create a lookup
- Therefore, anyone can influence contents of our cache
Hash Function is Important

- Must distribute well.
- Must be non-deterministic for remote entities
- Implies some unknown component fed into hash.
- Routing cache uses Jenkins Hash ...
- ... with random number salting the input.
- By default random input recomputed every 10 minutes
This is true regardless of hash quality.
- Attacker can simply cycle through all values of all keys
- Each new packet sent modifies the lookup key in some way
- Each new packet creates a new routing cache entry
- Triggers garbage collection when size limit is reached
- Cache is no longer a cache, since every lookup misses
- This is more expensive than having no cache at all.
Even with “well behaved” traffic, cache is undesirable
Google sees hit rates on the order of only 10 percent
On simpler systems, cache is effective
But still exposed to key cycling denial of service
The cache has to be removed
Not Easy

- Multi-year project
- Issues:
  - Performance retention
  - No externally visible semantic changes
  - Dependencies on Cache
  - Finding the right mix of interfaces
- A lot of feeling around in the dark
Performance Testing

- Micro vs. Macro analysis
- Complete isolation of route lookup
- Top-level operation measurements
- Looking at route lookups “in context”
- Goal: less than 10 percent cost increase for real operations
Macro Analysis: udpflood

- Built upon a design by Eric Dumazet
- Basically: Loop sending N small UDP packets
- No external network connectivity necessary
- Uses special “dummy” network device
- As well as static ARP table entries
- All packets simply dropped when they hit device
- Allows pure software analysis of packet send cost
Micro Analysis: First Try

- "route_bench"
- Uses routing sockets, specifically GETROUTE requests.
- Basically: Loop looking up N routes
- Doesn’t work well at all.
- Measures the wrong thing.
- Routing socket message overhead dominates test.
- So test isn’t really measuring route lookups.
Micro Analysis: Second Try

- Specialized kernel module
- Calls route lookup function
- Takes CPU cycle counter before and after
- Module parameters configure the lookup parameters
- Loading the module runs the benchmark
- Even deeper analysis: “perf”
Micro Analysis: Perf

- Performance analysis tool (since Linux 2.6.31)
- Uses CPU performance monitor facilities
- Can monitor both user and kernel mode
- Can watch software events (page faults, context switches)
- Sampling records program counter at time of event
- Allows easy detection of “hot spots” during a test
Micro Analysis: Final Setup

- Run module load under “perf record”
- Analyze run using “perf report”
- Provides detailed cost analysis of all route lookup code paths
- Allowed quick fixing of lowest hanging fruit
- Profile counts with and without routing cache could be compared
- All base perf runs using CPU cycle counter as “event”
- Deeper analysis with “cache miss” and “data access” events
Route Identities

- Routing Cache entry was “fully specified”
- Contained saddr, daddr, TOS, etc.
- Lots of dependencies exist
- Remove such dependencies
- Replace with other methods of reconstituting values
Routing Metrics

- Path parameters
- TCP round trip times
- Path MTU
- Specific to exact path used
- Therefore must be maintained with strict keying
- Metrics cannot be shared amongst several destinations
TCP Metrics Cache

- Entries created by successful TCP connection establishment
- Not susceptible to exploitability like routing cache
- TCP connection creation requires work by sender
- Attacker has to use his own resources and reveal his identity
- Entries keyed only on destination address
Route Exceptions

- Created for Path-MTU and Redirects
- Result from responses to packets we send
- Cannot be triggered arbitrarily by attackers
- These two cases are rare
Neighbours

- Hold link-level nexthop information (for ARP, etc.)
- Routing cache pre-computed neighbours
- Remember: One “route” can refer to several nexthops
- Need to disconnect neighbours from route entries.
- Solution:
  - Make neighbour lookups cheaper (faster hash, etc.)
  - Compute neighbours at packet send time ...
  - .. instead of using precomputed reference via route
- Most of work involved removing dependencies on old setup
Mitigating Factors: Part I

- “Generic Receive Offload” or simply GRO
- Accumulates RX packets for the same TCP connection
- Rules:
  - Same connection ID
  - No special flags
  - Consecutive sequence numbers
- Passes large accumulated packet as one object into stack
- Decreases “transactional overhead”
- Process 1 packet instead of N packets
- Means “perform 1 route lookup instead of N route lookups”
Mitigating Factors: Part II

- “TCP Pre-Demux”
- Lookup TCP socket on receive, before route lookup
- Cache input route in TCP socket
- Use cached input route to avoid route lookup
- TCP hash table not susceptible to routing cache problems
  - “attacker” must allocate his own resources
  - Must complete 3-way handshake
  - Must maintain lots of state
End Result

- Routing cache removed in kernel 3.6
- 10 percent goal achieved for udpflood
- Input route lookup for local TCP sockets actually faster
- Thanks to TCP pre-demux
- DoS attacks are completely eliminated
- Major improvements are still possible
- For example: Combine local and main tables
- Would half a major component of lookups
Thanks

- Jason Nieh
- Eric Dumazet
- Julian Anastasov
- Steffen Klassert
- Herbert Xu
- Tom Herbert