rethinking

Routing in FreeBSD 13

ALEXANDER CHERNIKOV <MELIFARO@FREEBSD.ORG>
Agenda

- Motivation
- Nexthops & Nexthop groups
- fib(9) KPI
- Lookup algorithms framework
- Algorithms overview
- Performance
- Next steps
Motivation

- Working multipath
- Improving control plane performance under traffic load
- Desire for the new high-performance lookup algs
- Build the base for other new features
Problems with routing subsystem

- Lack of isolation
- Tightly coupled with almost all other network parts
- struct rtentry widely used by external callers (~100 places)
- Extremely hard to change
New concepts

- Clear KPI with isolation goal in mind
  - Explicit "private" and "public" parts
- Most lookup consumers don’t require prefix data
  - Nexthop-related information is sufficient
- Construct KPI around the nexthops
Basic building blocks: Nexthops

- Structure with data enough to push packet
- Mostly immutable
- epoch(9)-based reclamation
- refcount(9) for tracking
  - control plane & route caching
- Stored in auto-resizable hash table
Building blocks: Nexthops #2

- Public part: returned by KPI
- Private part: used internally
  - refcounting
  - housekeeping

```
struct nhop_object {
    uint16_t nh_flags;
    uint16_t nh_mtu;
    struct sockaddr gw_sa;
    struct ifnet *nh_ifp;
    struct ifaddr *nh_ifa;
    struct ifnet *nh_aifp;
    counter_u64_t nh_pkSENT;
    uint8_t nh_prepend_len;
    uint8_t spare[3];
    uint32_t spare1;
    char nh_prepend[48];
    struct nhop_priv *nh_priv;
};

struct nhop_priv {
    uint8_t nh_family;
    uint8_t spare;
    uint16_t nh_type;
    uint32_t rt_flags;
    uint32_t nh_idx;
    void *cb_func;
    u_int nh_refcnt;
    u_int nh_linked;
    struct nhop_object *nh;
    struct nh_control *nh_control;
    struct nhop_priv *nh_next;
    struct vnet *nh_vnet;
    struct epoch_context nh_epoch_CTX;
};
```
Building blocks: Nexthops #3

- Better for data plane: smaller cache-friendly footprint
- Better for control plane: pre-calculated entities to deal with
- Peering routers have hundreds nhops, not millions
- Easier to store additional info / iterate

<table>
<thead>
<tr>
<th>Idx</th>
<th>Type</th>
<th>IFA</th>
<th>Gateway</th>
<th>Flags</th>
<th>Use</th>
<th>Mtu</th>
<th>Netif</th>
<th>Addrif</th>
<th>Refcnt</th>
<th>Prepend</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>v4/resolve</td>
<td>127.0.0.1</td>
<td>lo0/resolve</td>
<td>H</td>
<td>292</td>
<td>16384</td>
<td>lo0</td>
<td>0</td>
<td>1500</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>v4/resolve</td>
<td>10.0.0.8</td>
<td>vtinet0/resolve</td>
<td>0</td>
<td>1500</td>
<td>vtinet0</td>
<td>0</td>
<td>16384</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>v4/resolve</td>
<td>127.0.0.1</td>
<td>lo0/resolve</td>
<td>HS</td>
<td>0</td>
<td>16384</td>
<td>lo0</td>
<td>vtinet0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>v4/gw</td>
<td>10.0.0.8</td>
<td>10.0.0.1</td>
<td>GS</td>
<td>0</td>
<td>1500</td>
<td>vtinet0</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

netstat -4onW (inet.0)
Building blocks: nexthop groups

- Used to store multipath route data
- Groups of nexhop-weight pairs.
- Internal to the routing subsystem
- Immutable
- refcount(9)
- epoch(9) backed reclamation
- Stored in an auto-resizable hash table
- Can be referenced by pointer or by index

<table>
<thead>
<tr>
<th>nhgrp_priv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flags</td>
</tr>
<tr>
<td>Index</td>
</tr>
<tr>
<td>num_nhops</td>
</tr>
<tr>
<td>Nhop#</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

struct nhgrp_priv
Building blocks: nexthop groups (cont)

- Need effective dataplane implementation
- Currently uses array of nexthops
- Can be compiled differently
- Max datapath group width: 64
- Default weight: 1 (RT_DEFAULT_WEIGHT)
- `route add .. -weight 100`

### Control plane

<table>
<thead>
<tr>
<th>Nhop#</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>200</td>
</tr>
<tr>
<td>4</td>
<td>300</td>
</tr>
</tbody>
</table>

### Data plane

```
   len:5
   6
   6
   4
   4
   4
```

Compilation example
Using nexthop groups

- Transparent to consumers
- flowid-based nexthop selection
- flowid semantics agnostic to the function
- Dataplane part shares flag field with nexthop

```c
struct nhop_object *
fib4_lookup(uint32_t fibnum, struct in_addr dst,
             uint32_t scopeid, uint32_t flags, uint32_t flowid)
{
    struct nhop_object *nh;
    ...
    if (nh->nh_flags & NHF_MULTIPATH) {
        struct nhgrp_object *nhg;
        nhg = (struct nhgrp_object *)nh;
        nh = nhg->nhops[flowid % nhg->nhg_size];
    }
    return (nh);
}
```

nexthop selection in fib4_lookup()
Using nexthops: data plane KPI

- Per-family functions
- Do not require sockaddr
- Transparently handles multipath
- `fib4_lookup(fibnum, flags, addr, scopeid, flowid) -> nhop ptr`
- `fib6_lookup(fibnum, flags, *addr, scopeid, flowid) -> nhop ptr`
Using nexthops: data plane KPI #2

OLD

public

prefix
prefix
prefix
gw

Lookup result

prefix
gw

NEW

private

prefix
prefix
prefix

nhopgroup

public

nexthop

nexthop

nexthop

nexthop

Lookup result

nexthop

Part of new KPI: the only "public-visible" datastructure
Using nexthops: control plane KPI

- Single function to control routing table: `rib_action()`
- As before, `struct rt_addrinfo` contains all necessary data
- Nexthop and nexthop group creation happens within routing subsystem
- NET_EPOCH is required
Using nexthops: control plane KPI

- Some callers still need to know the matched prefix
- `route get`, netflow
- Provide fib[46]_lookup_rt() functions returning rtentry
  - With nhop/weight data at the time of lookup
- No direct rtentry access, special accessor functions
  - rt_get_inet[6]_prefix_plen()
  - rt_get_family()
Using nexthops: locking/refcounting

- Nhops/nhop groups are (mostly) immutable
- Change in the route attributes forces creation of a new nexthop
- No per-nexthop/nexhop groups locks
- Shared per-routing-table rwlock for hash operations
- refcount(9): control plane, route caching(nexthop caching)
- No refcounts/locking for rtentry
Userland

- Changes are mostly transparent
- Old binaries (route(8), routing daemons) should work on 13-S
- Multipath works with quagga out-of-the box
- More work required to support multipath in bird
Nexthops summary

- Decouples all routing specifics from the lookup algo
- Now it’s a pure function of IPv[46] -> #index
- Nexthop pointers/indexes are required for the high-perf algorithms
Lookup algorithms framework

- Optimal performance for the specific use-case
  - IPv6 is different from IPv4
  - Full-view is different from 10 routes
- Lockless lookups
  - Better control plane performance during convergence
  - Routing daemon does not compete with dataplane for the rtable locks
- Foundation for the other families (MPLS)
- Reducing the bar for the new algorithms implementation & testing
Lookup framework: features

- Algorithms can be loaded on the fly
- Automatic algorithm selection
- Based on the amount of routes/nhops
- No dataplane locks
- Control plane decoupled from the data plane
- Updates batching if requested
Lookup framework: internals

- Reliable subscriptions to the route changes
- Ability to keep multiple instances of same/different algo for a table in sync
- Handling every failure by spinning up new algo instance
- Delayed updates / batch updates
Lookup framework: algo selection

- Each algo has to implement a preference callback
- Based on the #of routes/nhops -> return value 0..255
- Framework compares values of all algos to select the best
- Periodic re-evaluation (100 routes or 30 seconds)
- New has to be at least 5% better to switch
Lookup framework: updates batching

- Conflicting requirement for batching
- Batch more to amortise cost
- Minimise update delay
- Need to find the sweet spot

- Force commit for non-gw or static routes
- Bucket updates in 50ms chunks
- Commit if # below the threshold (500 routes)
- Otherwise delay & check next bucket
- Max delay 1000ms
- Each value configurable in net.route.algo
Lookup framework: control plane perf

- Data path does not contest RIB lock
- Improved converge times
- 8x improvement for BGP full-view going down (~300 -> 40 seconds)
Lookup algorithms

- dpdk_lpm4
- dpdk_lpm6
- dxr
- bsearch4
- bsearch6
- radix4_lockless
- radix6_lockless
Lookup algorithms: dpdk_lpm

- IPv4: variation of DIR-24-8
- IPv6: 26 strides
- Immediate updates
- IPv6 link-local lookups handled by system radix
Lookup algorithms: DXR

• Tailored for the large-scale FIBs
• Works well for small-scale too
• Implementation by Marko Zec
• Uses update batching
Lookup algorithms: bsearch

- Array binary search
- Simple and cache-effective for small route scale
- Array is immutable and is rebuild on every route change
- 1-20 routes
Lookup algorithms: radix_lockless

- Immutable tree in contiguous memory chunk
- Backed by system-default trie
- Rebuild on every route change
- 20-1000 routes
Lookup algorithms: performance

- +30% for IPv6
- +21% for IPv4
Next steps

- Add direct nexthop/nhop group handling to rtsock
- Add netlink support