Picking Low Hanging Fruit from the FIB Tree

Networking Services Team, Red Hat
Alexander Duyck

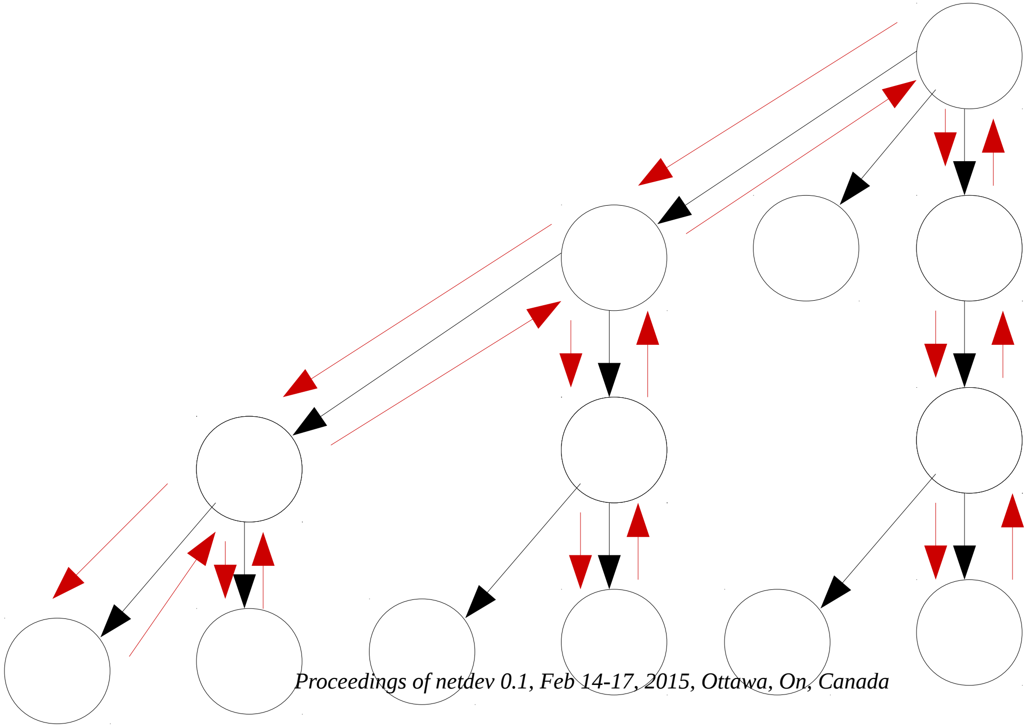
February 17th, 2015

Agenda

- What was wrong with the FIB trie?
- Identifying the low hanging fruit
- Results of current work
- What more can be done?
- Conclusions

What Was Wrong with the FIB Trie?

- The trie was designed for look-up, not prefix match
- For large trie, prefix match could be very slow
- Finding a node was O(N) time where N is trie depth
- Finding a longest prefix match could be $O(N^2)$ time



Proceedings of netdev 0.1, Feb 14-17, 2015, Ottawa, On, Canada

Identifying the Low Hanging Fruit

- Reduce code complexity
- Avoid spending cycles on unnecessary steps
- Make use of unused memory where possible

Making a Leaf Look like a Tnode

- Before

```
struct rt_trie_node {
        unsigned long parent;
        t_key key;
};

struct leaf {
        unsigned long parent;
        t_key key;
        struct hlist_head list;
        struct rcu_head rcu;
};

struct tnode {
        unsigned long parent;
        t_key key;
        unsigned char pos;
        unsigned char bits;
        unsigned int full_children;
        unsigned int empty_children;
        union {
                struct rcu_head rcu;
                struct tnode *tnode_free;
        };
        struct rt_trie_node __rcu *child[0];
};
```

- After

```
struct tnode {
        t_key key;
        unsigned char bits;
        unsigned char pos;
        struct tnode __rcu *parent;
        struct rcu_head rcu;
        union {
                struct {
                        t_key empty_children;
                        t_key full_children;
                        struct tnode __rcu *child[0];
                };
                struct hlist_head list;
        };
};
```

Exact Match

- Move pos from matched bits to yet to be matched bits
- pos + bits <= 32 && bits > 0 is tnode
- pos == 0 && bits == 0 is leaf
- Mask lower bits of tnode to be consistent with leaf
- Much easier to both test array bounds and key
    - (key ^ n->key) >> n->pos provides index into node
    - index < (1ul << n->bits) both verifies bounds and key
    - n->bits already in register when we check for leaf

Prefix Mismatch

- Before

```
pref_mismatch = mask_pfx(cn->key ^ key, cn->pos);
if (pref_mismatch) {
        /* fls(x) = __fls(x) + 1 */
        int mp = KEYLENGTH - __fls(pref_mismatch) - 1;

        if (tkey_extract_bits(cn->key, mp, cn->pos - mp) != 0)
                goto backtrace;

        if (current_prefix_length >= cn->pos)
                current_prefix_length = mp;
}
```

- After

```
if ((key ^ n->key) & (n->key | -n->key))
        goto backtrace;
```

- New code takes least significant of n->key and generates mask
  - (X | -X) == (~0 << ffs(n->key))
  - X = 192.168.1.0; (X | -X) == 255.255.255.0

Stripping Bits Simplified

- Before

```
while ((chopped_off <= pn->bits)
       && !(cindex & (1<<(chopped_off-1))))
        chopped_off++;

cindex &= ~(1 << (chopped_off-1));
```

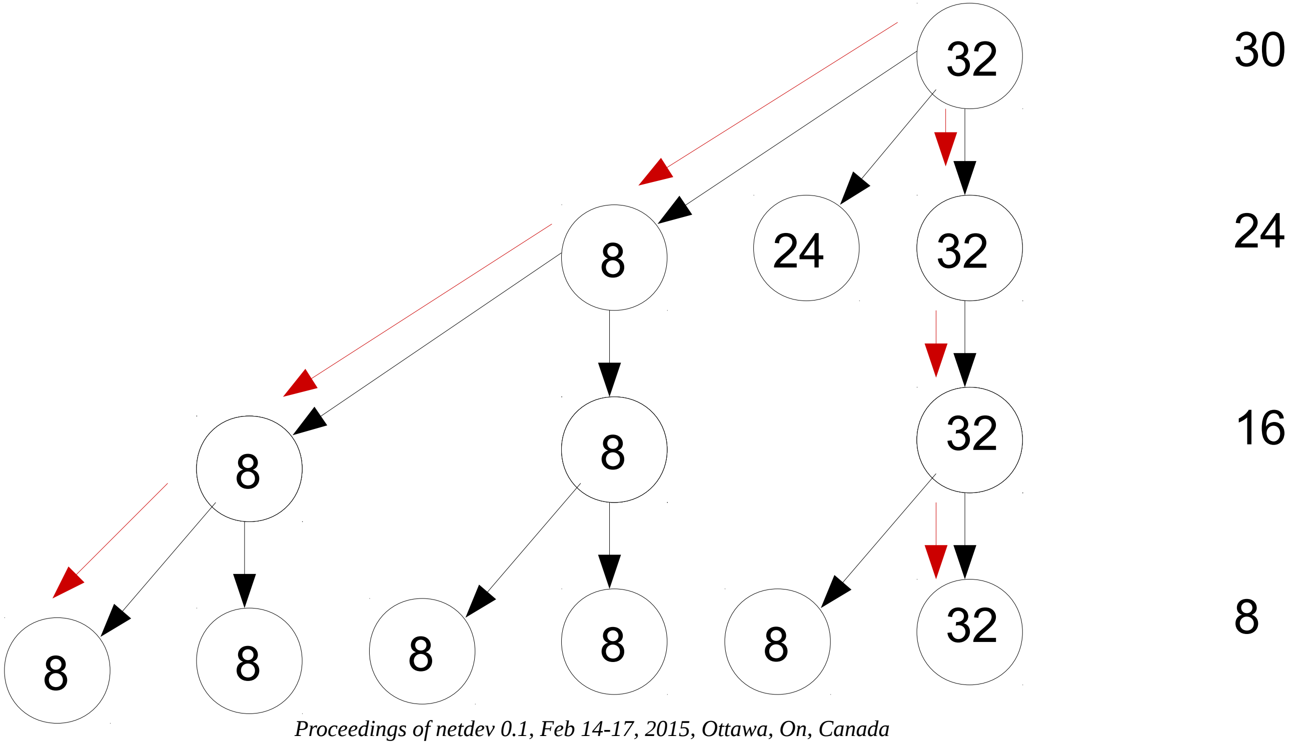
- After

```
cindex &= cindex – 1;
```

- New code can drop now unused variables from loop
  - chopped_off
  - current_prefix_length

Avoiding Shallow Suffixes

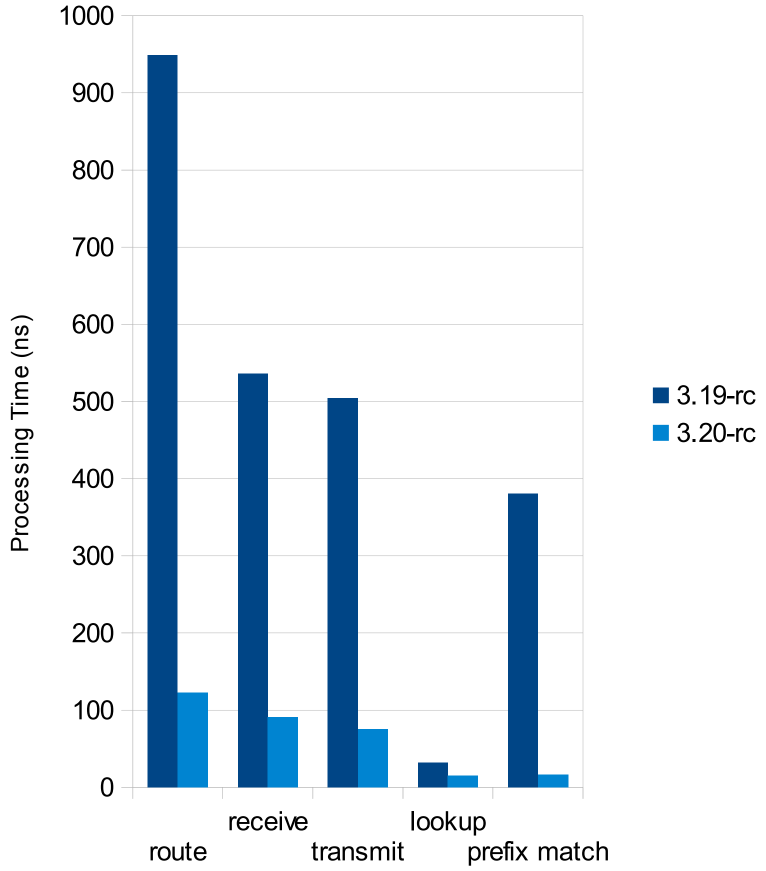
- Finding longest prefix match still O(N^2)
- Add tracking value slen for prefix/suffix lengths
- Tracking value reduces prefix match to O(N)



Proceedings of netdev 0.1, Feb 14-17, 2015, Ottawa, On, Canada

Results

- Look-up time reduced by half.
- Time spent on longest prefix match nearly eliminated.



What More Can Be Done?

- Still a considerable amount of time spent in look-up
  - Leaf requires 3 cache lines minimum
    - leaf, leaf_info, fib_alias
  - Longest prefix match starts at first tnode/leaf
  - Up to 2 cache lines accessed per tnode for look-up

Removing Leaf Info

- The leaf_info structure contains redundant data

```
struct leaf_info {
        struct hlist_node hlist;
        int plen;
        u32 mask_plen; /* ntohl(inet_make_mask(plen)) */
        struct list_head falh;
        struct rcu_head rcu;
};
```

  - plen and prefix_mask can be derived from each other
  - plen could be reduced to a single byte
  - The fib_alias structure has room for an additional byte
- 1 cache line for 1 byte of data is a waste
- Solution is to move byte to fib_alias and drop leaf_info

Wrap Pointers in Key Vector

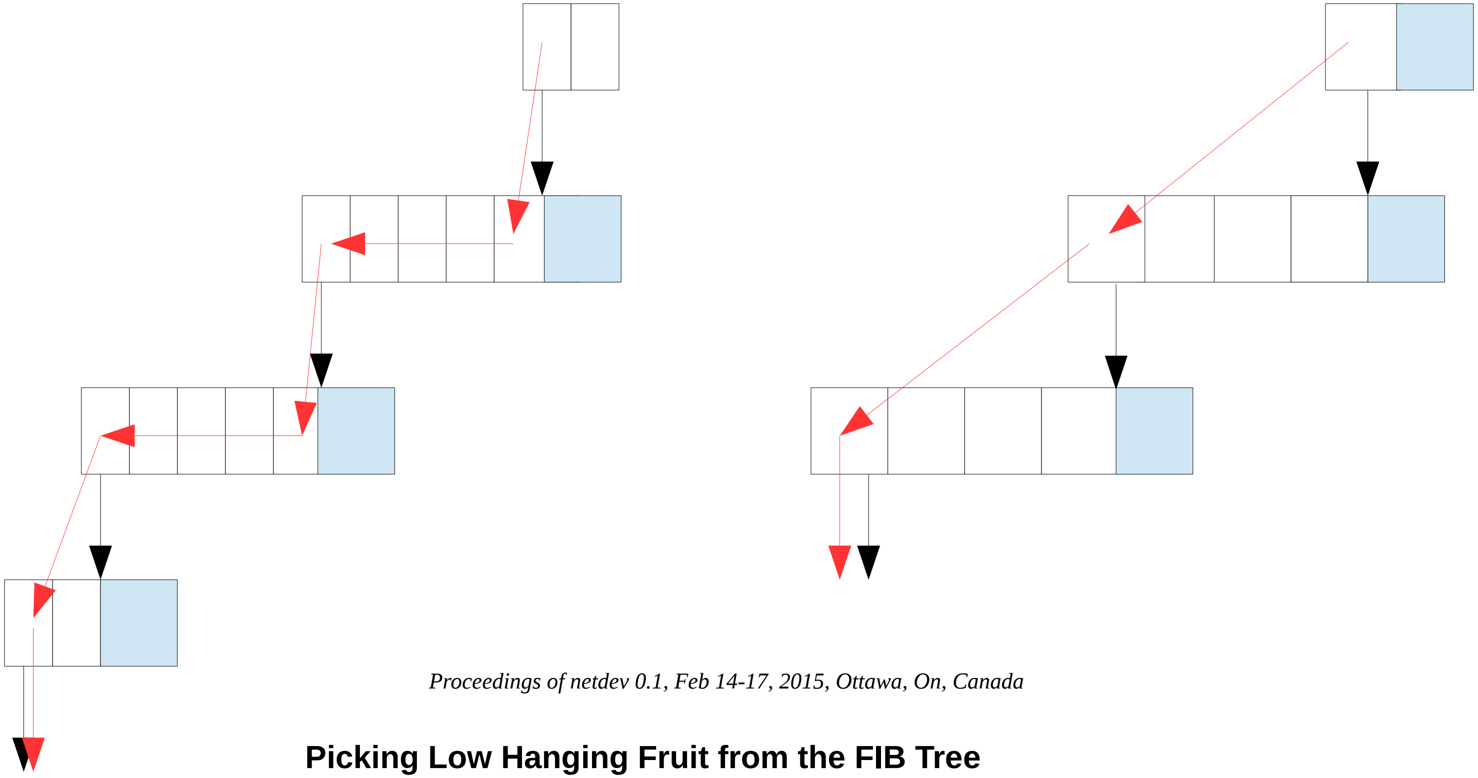
- Make it so that root pointer is also a pseudo-tnode

```
struct key_vector {
        t_key key;
        unsigned char pos;              /* 2log(KEYLENGTH) bits needed */
        unsigned char bits;             /* 2log(KEYLENGTH) bits needed */
        unsigned char slen;
        union {
                /* This list pointer if valid if bits == 0 (LEAF) */
                struct hlist_head leaf;
                /* The fields in this struct are valid if bits > 0 (TNODE) */
                struct key_vector __rcu *tnode[0];
        };
};
```

- pos == 32 && bits == 0 is trie root
- Allows for processing key before checking for root
- n->pos already in register as part of acquiring cindex

Up-level the Key Vector

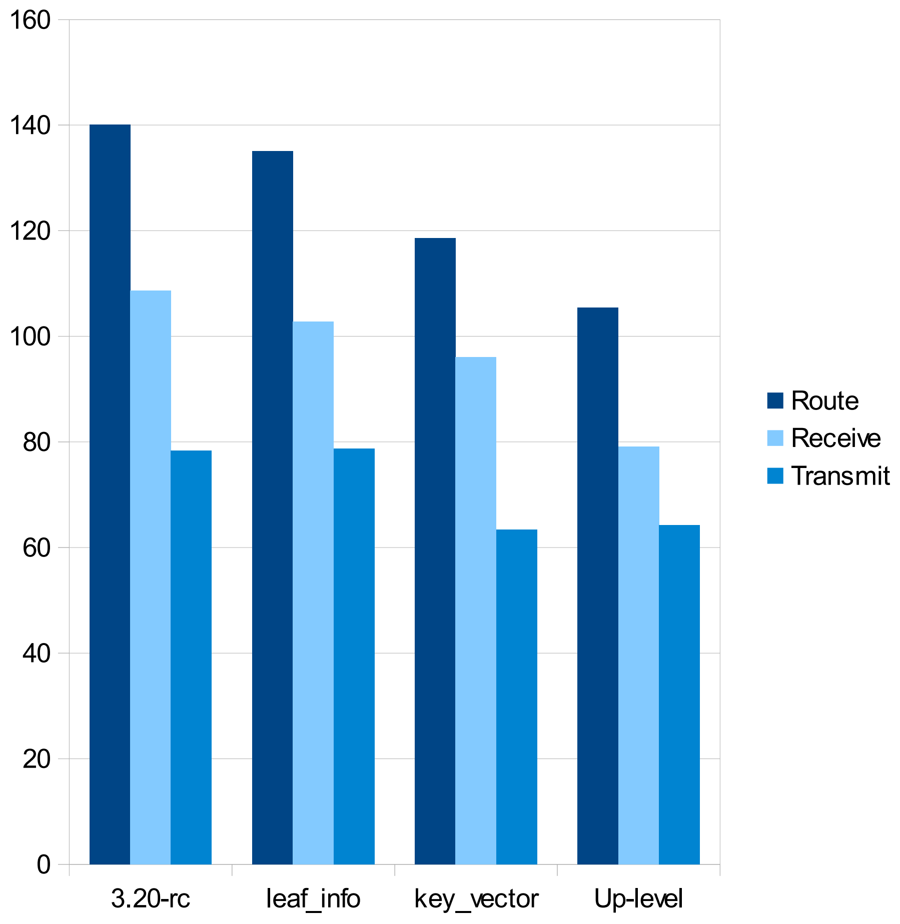
- Up to 2 cache lines to access next tnode
- Pushing key info up one level to same level as pointer could cut cache-line accesses in half
- Some RCU ugliness still needs to be resolved



Proceedings of netdev 0.1, Feb 14-17, 2015, Ottawa, On, Canada

Conclusions

- Reduced look-up by an additional 25%
- Routing look-up time now only 1.5x transmit instead of 2x
- Further gains may require drastic redesign



Proceedings of netdev 0.1, Feb 14-17, 2015, Ottawa, On, Canada

Questions?

- We’re Hiring
  - http://jobs.redhat.com