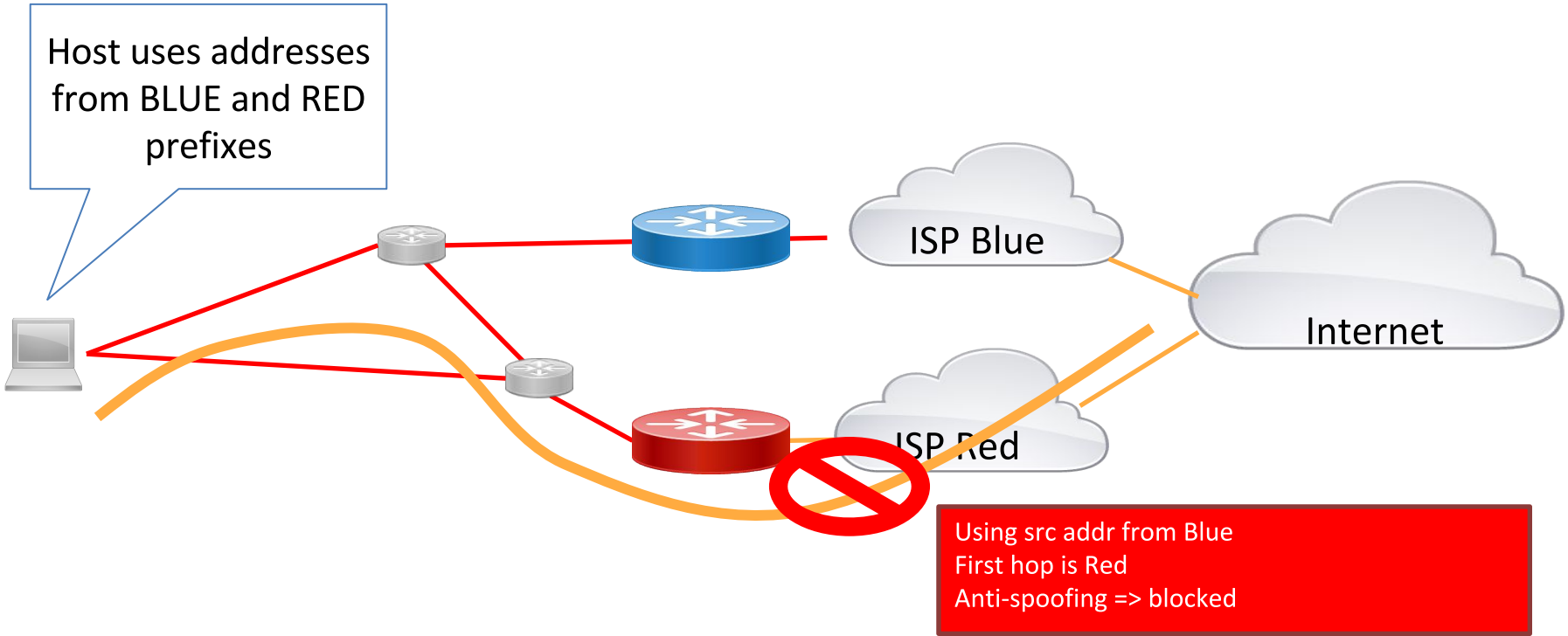
Experience implementing SADR in VPP

Ole Trøan - Pierre Pfister

VPP @ Hackathon IETF‘95

- VPP - Open source software data plane
  - Commodity hardware, ~10Mpps/core,
  - http://fd.io Source: git clone https://gerrit.fd.io/r/vpp
- IETF - Running code:
  Before: Proprietary hardware
  Now: Production high performance forwarding with commodity hardware and open source
- Project:
  - Source Address Dependent Routing:
    - draft-ietf-rtgwg-dst-src-routing
    - Implementation and performance
    - https://github.com/Oryon/openvpp/blob/ietf95/vnet/vnet/ip/ip6_forward.c#L84
- Champion: Ole Trøan <ot@cisco.com>

Need for Source Address Dependent Routing (SADR)



SADR in VPP - what we did

- **Victoria, Ezequiel**, Pierre (in Paris), Eric, Ole
- Setup development environment. Pulled code, built
- Two competing proposals for data structure:
    i. Extend hash key to 256 bit (D, S) lookup
    ii. D table entry point to S tables
- Running implementation of (i).
- Test setup, Auto generation of FIBs, PCAP files, performance measurements

Existing dest-only VPP IPv6 FIB data-structure

FIB is arranged as a hash-table + a bit vector of used prefix length in the table.

Key for hash-table lookup is `(dst_address | prefix_length | fib_index)`

```
kv.key[0] = dst->as_u64[0];
kv.key[1] = dst->as_u64[1];
fib = ((u64)((fib_index))<<32);
for (i = 0; i < len; i++)  {
      int dst_address_length = im->prefix_lengths_in_search_order[i];
      ip6_address_t * mask = &im->fib_masks[dst_address_length];
      kv.key[0] &= mask->as_u64[0];
      kv.key[1] &= mask->as_u64[1];
      kv.key[2] = fib | dst_address_length;
      rv = BV(clib_bihash_search_inline_2)(&im->ip6_lookup_table, &kv, &value);
      if (rv == 0) return value.value;
}
```

IPv6 FIB SADR Lookup

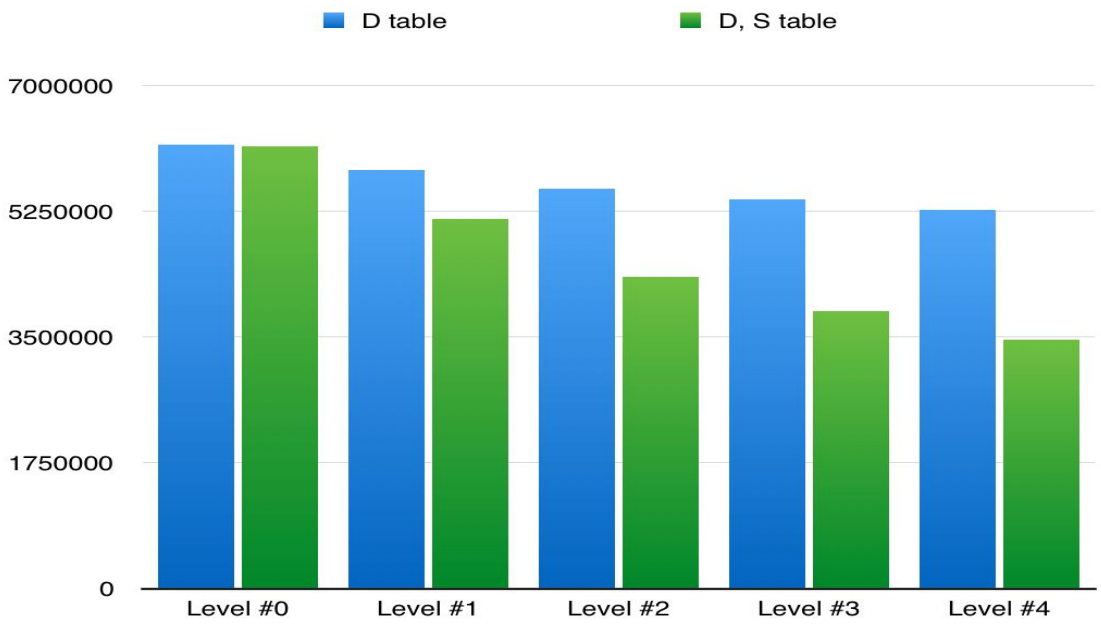
- Hash table indexed by: *<destination, **prefixLength**, FIBIndex>*
  - Iterate on ***prefixLength*** starting from the longest entry in the table until match found
- When match:
  - No source prefix -> Done (normal Dest lookup)
  - Lookup in a S,D hash table keyed on *<destination, destinationPrefixLength, source, sourcePrefixLength, FIBindex>*
  - Iterate on *sourcePrefixLength* starting from longest in the table until match is found
  - If no match on source prefix, then backtrack up the next longest matching D

VPP IPv6 SADR code

https://github.com/Oryon/openvpp/blob/ietf95/vnet/vnet/ip/ip6_forward.c#L84

```
BVT(clib_bihash_kv) kv, value;  clib_bihash_kv_40_8_t  kv2, value2;
len = vec_len (im->prefix_lengths_in_search_order); srclen =  vec_len (im->srcprefix_lengths_in_search_order);
for (i = 0; i < len; i++) {
     int dst_address_length = im->prefix_lengths_in_search_order[i];
     ip6_address_t  * mask = &im->fib_masks[dst_address_length];
     kv.key[ 0] = dst->as_u64[ 0] & mask->as_u64[ 0];
     kv.key[ 1] = dst->as_u64[ 1] & mask->as_u64[ 1];
     kv.key[ 2] = ((u64)((fib_index))<< 32) | dst_address_length;
     rv = clib_bihash_search_inline_2_24_8 (&im->ip6_lookup_table, &kv, &value);
     if (rv != 0) continue;
     if (PREDICT_TRUE (FIB_GET_SRCCOUNT (value.value) ==  0 && FIB_GET_ADJINDEX (value.value) != ~ 0))
        return FIB_GET_ADJINDEX (value.value);
     kv2.key[ 0] = dst->as_u64[ 0] & mask->as_u64[ 0];
     kv2.key[ 1] = dst->as_u64[ 1] & mask->as_u64[ 1];
     for (j = 0; j < srclen; j++) {
       int src_address_length = im->srcprefix_lengths_in_search_order[j];
       mask = &im->fib_masks[src_address_length];
       ASSERT(src_address_length >=  0 && src_address_length <=  128);
       kv2.key[ 2] = src->as_u64[ 0] & mask->as_u64[ 0];
       kv2.key[ 3] = src->as_u64[ 1] & mask->as_u64[ 1];
       kv2.key[ 4] = ((u64)((fib_index))<< 32) | dst_address_length | (src_address_length <<  8);
       rv = clib_bihash_search_inline_2_40_8 (&im->ip6_srclookup_table, &kv2, &value2);
       if (rv == 0) return value2.value;
     }
   }
```

Performance



Conclusion

- A D-FIB has a worst case of 128 lookups
- A D-FIB of S-FIBs has effective memory usage, but worst case is 128*128 lookups
- A D-FIB of S-FIBs is also the most scalable data structure, which would easily scale to millions of D, S routes.
- Backtracking is expensive. We saw a forwarding performance of about 10% per introduced level of backtracking.
- In VPP we will look at the tradeoffs between the two lookup algorithms: https://tools.ietf.org/html/draft-ietf-rtgwg-dst-src-routing-02#section-3.3
    - A S-FIB where we first do a 128 bit BMP lookup on the source.
    - The S-FIB lookup returns the index of a D-FIB
    - Normal 128 bit BMP lookup done in D-FIB table.
    - Separate D-FIB table per source prefix.
    - D,* routes are duplicated across all D-FIBs
    - Worst case is 128 + 128 lookups