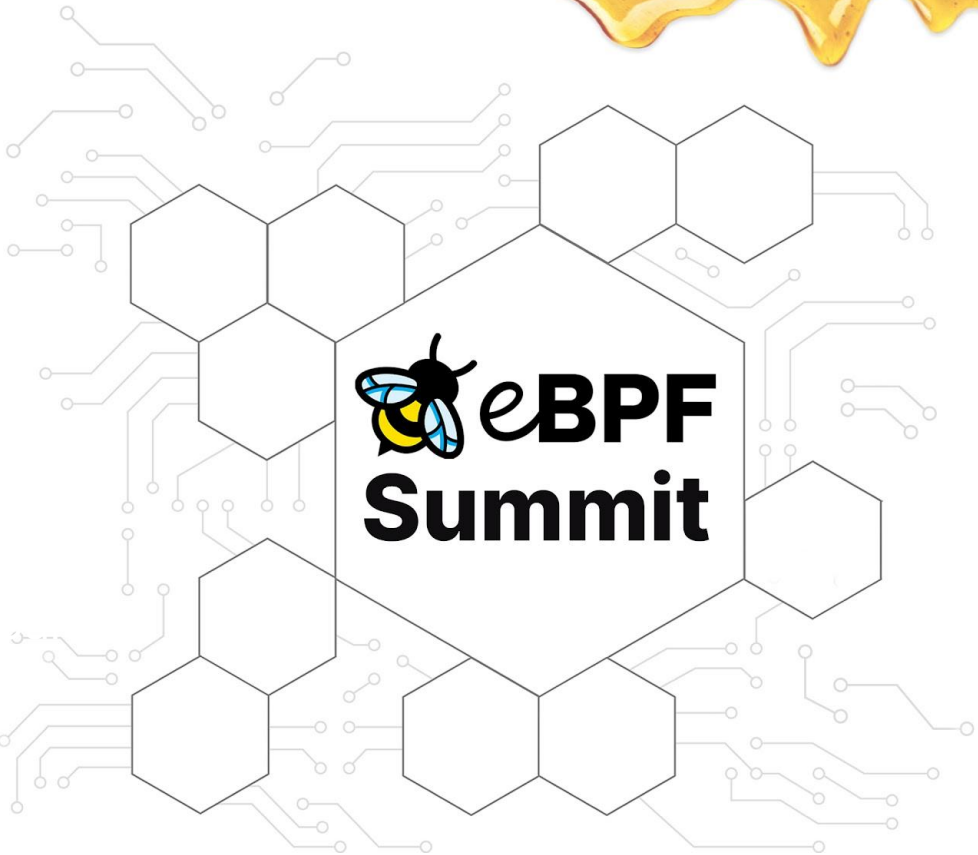
Fighting and overcoming the complexity limit



or: Using a verifier friendly algorithm

Daniel Xu

@__dxu

Agenda

- More or less a chronology of events
- While pointing out some (subjectively) interesting tidbits

Context

- “Micro-segmentation” product
- An applied form of packet classification
- Problem statement
    1. Categorize packets into user-specified flows
    2. Perform flow-specific action (drop, route, etc.)
- 5.4 kernel

First pass

```
/* [...] */

for (int i = 0; (i < MAX_ENTRIES) && (i < nr_policies); ++i) {
    policy = bpf_map_lookup_elem(&policies, &idx);
    if (!policy)
        goto out;

    /* Flow domains are bidirectional */
    src = match(&info, policy, SRC, false);
    dst = match(&info, policy, DST, false);
    rsrc = match(&info, policy, SRC, true);
    rdst = match(&info, policy, DST, true);

    if (!(src && dst) && !(rsrc && rdst))
        continue;

    return policy->action;
}

/* [...] */
```

aviatrix

Second pass (more features)

```
for (i = 0; (i < MAX_PER_TC) && (polidx(cur, i) < nr_policies); ++i) {
    policy = bpf_map_lookup_elem(&policies, &idx);
    if (!policy)
            goto out;

    /* Flow domains are bidirectional */
    src = match(&info, policy, SRC, false);
    dst = match(&info, policy, DST, false);
    rsrc = match(&info, policy, SRC, true);
    rdst = match(&info, policy, DST, true);

    if (!(src && dst) && !(rsrc && rdst))
        continue;

    if (policy->watch) {
        /* [...] */
        continue;
    }

    if (chosen->log)
        log_packet(xdp, &info, chosen);

    /* Policy matched, perform requested action */
    chosen = policy;
    goto out;
}
```

- More control flow in loop body
- Scale with tail calls
- But now cannot mix with bpf2bpf calls

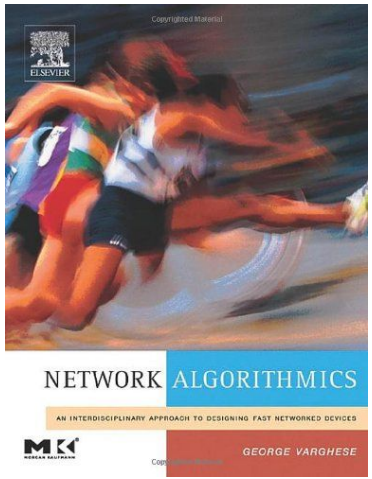
aviatrix

Tipping point

- Too much for verifier to handle
- Could not extend pass 256 rules
- Overhead (even at 64 rules) was too high (12GBit/s -> 8GBit/s)
    - 8 rules per tail call (horrible)
- Literature review time!

Breakthrough

- Sebastiano Miano, Matteo Bertrone, Fulvio Risso, Mauricio Vásquez Bernal, Yunsong Lu, and Jianwen Pi. 2019. Securing Linux with a faster and scalable iptables. SIGCOMM Comput. Commun. Rev. 49, 3 (July 2019), 2–17. https://doi.org/10.1145/3371927.3371929
-



aviatrix

Linear Bitvector Search (LBVS)

- Basic idea is that given P rules, decompose the packet into N dimensions, each dimension being a field we care about, eg. source IP
- Do N lookups and get back N P-bit bitvectors, where each bit represents a rule
- Do bitwise arithmetic over bitvectors to arrive at final bitvector, F
- **Do u64 sized strides over F to identify a non-zero u64**
- Do O(1) De Bruijn Sequence to find the first set bit in a u64

LBVS highlights

- Verifier friendly
  - Body of loop is tight; minimal control flow
- Cache friendly
  - Straight-line data access pattern
- CPU friendly
  - CPU can tear through word sized logic ops

Third pass (rewrite)

```
/* [...] */

__builtin_memcpy(lpm_key.addr, &info->saddr, sizeof(lpm_key.addr));
src_fwd = bpf_map_lookup_elem(&src_cidrs, &lpm_key);
if (!src_fwd)
    src_fwd = zeros;

proto = info->l4_proto;
protoz = bpf_map_lookup_elem(&protos, &proto);
if (unlikely(!protoz))
    return ret;

/* [...] */

/* Now process the final bvec using u64 sized strides */
found = false;
for (i = 0; i < MAX_ENTRIES / 64; ++i) {
    final[i] = ((src_fwd[i] & dst_fwd[i] & port_fwd[i]) |
                (src_rev[i] & dst_rev[i] & port_rev[i] & ntcp[i])) &
                protoz[i];

    if (final[i]) {
        found = true;
        break;
    }
}

/* [...] */
```

Bitvector lookups

All **u64***

Linear search

aviatrix

Outcome

- Reached baseline performance (ie. no performance impact)
- Scales to 2000+ rules in single tail call
- Complexity!

Thanks

- dxuuu.xyz
- @__dxu

aviatrix

BONUS SLIDES FOLLOW

Future directions?

- More building blocks for smarter algorithms
  - dynptrs; local kptrs; linked lists
  - Can progs walk a tree yet?
- Smarter data structures
  - BPF_MAP_TYPE_WILDCARD
- BPF libraries instead of writing more kernel code?
  - Upgrading kernel sucks
  - When do we reach critical mass of functionality?