Rethinking BPF conntrack

Aditi Ghag
Isovalent at Cisco
LSF/MM/BPF’24

Outline

- Cilium conntrack background
- Problems
- Areas of improvements
- Leveraging kernel constructs
- Kernel extensions

BPF conntrack in Cilium (1)

- Native connection tracking for load-balancing and policy enforcement
- 5-tuple flow tracking based on a BPF LRU map
- Enables data sharing between Cilium TC and XDP programs

BPF conntrack in Cilium (2)

Egress

- Does a packet belong to an existing flow?
- Kubernetes service load-balancing DNAT

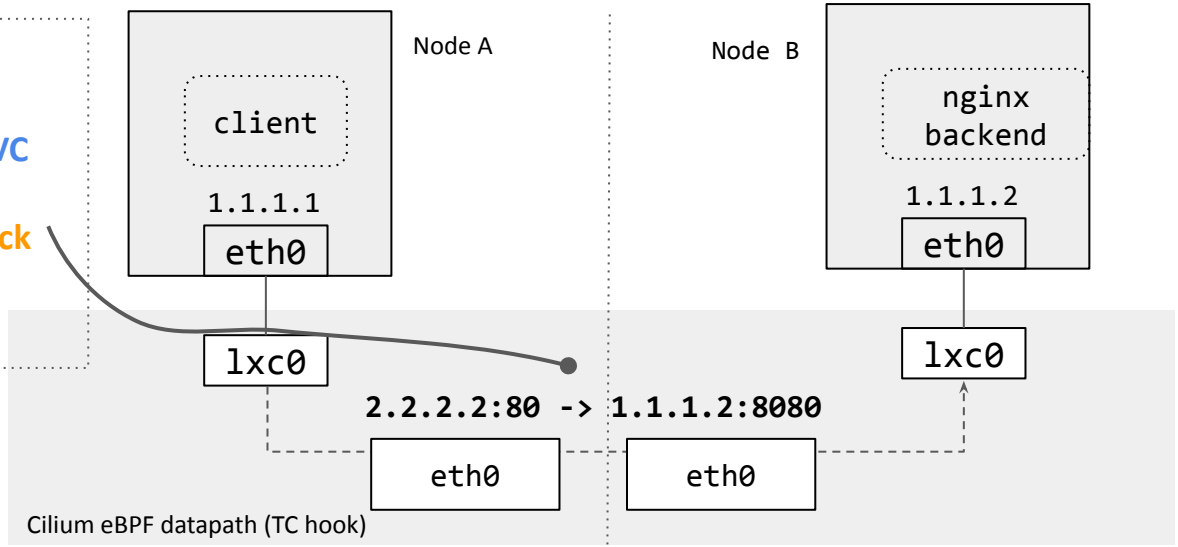
Ingress

- Is a packet reply for an existing flow?
- Kubernetes service load-balancing reverse DNAT
- Policy enforcement (if podA is allowed to connect to podB => allow podB replies to podA)

Life of a packet (pod <-> svc vip) and conntrack

REQUEST

1. Lookup svc VIP
2. Select svc backend (Create **SVC conntrack entry**)
3. DNAT (Create **Egress conntrack entry**)



eBPF conntrack LRU map (node A)

srcIP	sPort	dstIP	dPort	Type	=>	{backend,svc}IP:port
1.1.1.1	4000	2.2.2.2	80	SVC	=>	1.1.1.2:8080
1.1.1.1	4000	1.1.1.2	8080	Egress	=>	2.2.2.2:80

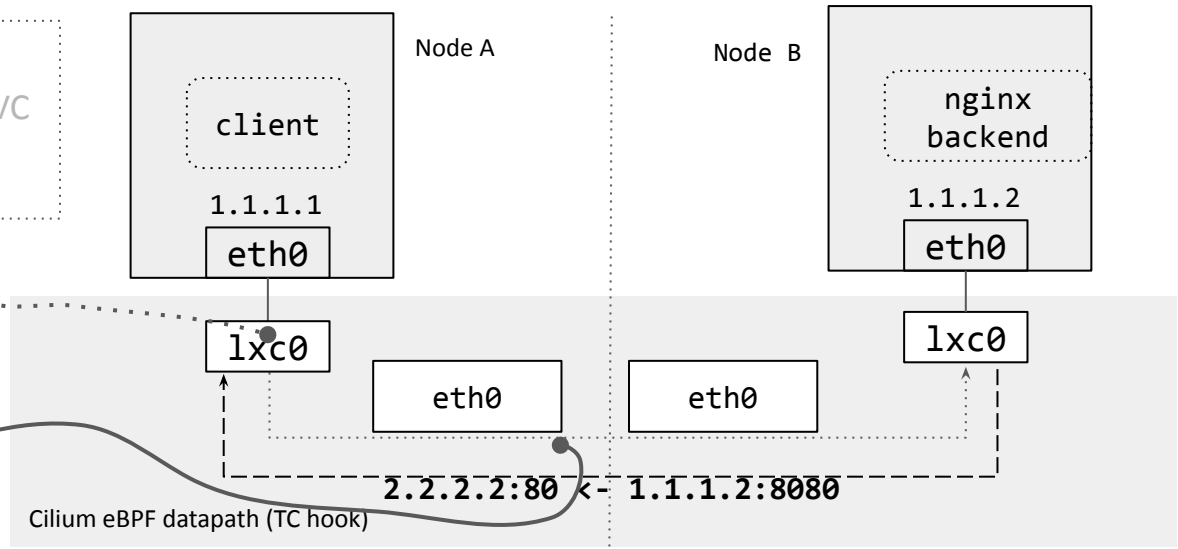
Life of a packet (pod <-> svc vip) and conntrack

REQUEST

1. Lookup svc VIP
2. Select svc backend (Create SVC CT)
3. DNAT (Create Egress CT)

REPLY

1. Lookup **Egress** conntrack entry
2. If found: Rev-DNAT



eBPF conntrack LRU map (node A)

srcIP	sPort	dstIP	dPort	Type	=> {backend,svc}IP:port
1.1.1.1	4000	2.2.2.2	80	SVC	=> 1.1.1.2:8080
1.1.1.1	4000	1.1.1.2	8080	Egress	=> 2.2.2.2:80

Cilium conntrack entry lifecycle management

- Entries are created with default timeouts and refreshed on flow packets
- Long timeouts to accommodate long-lived connections
- Garbage collection
  - Userspace agent asynchronously deletes expired entries
  - Adaptive interval

Problems

- Complex out-of-band garbage collection
    - Balancing act: keep CPU usage under control
    - Entries are not expired until GC happens
    - Conntrack entry reuse on tuple collisions
- Sticky entries: unbalanced service LB backend selection
- BPF LRU map insertion failures at high packet rates

Areas of improvements

- Making load-balancing decisions based on per socket state
    - Socket storage to save flow state
- Garbage collection in BPF
    - BPF map iterator to expire entries: Still requires interval calculation
    - Tying in conntrack entries to their corresponding socket lifecycles

Exploring socket storage

- Save flow tracking state like LB decisions
- Delegated storage: no map (re)sizing
- Automatic GC

Exploring socket storage

- Reply path needs socket lookup APIs: extra bookkeeping for netns
- Socket lookup on ingress doesn’t work due to LB DNAT
- Cilium systems piggyback on conntrack GC to check if connections are active: no map like iteration capabilities

Can socket storage replace Cilium LRU map?

- Well, not quite…
- But there are nice properties that can make conntrack GC more efficient

Leverage socket storage for efficient GC

Caching for easy lookups

- Create socket storage with Cilium conntrack map keys

Fate sharing

- Hook into socket storage delete events to expire stored keys

PoC based on socket storage driven GC

- Load fentry BPF program on **bpf_sk_storage_free**
- Store Cilium conntrack map keys during creation: SVC and EGRESS entries
- Lookup conntrack keys to be deleted
- Delete the keys from Cilium conntrack map

```
SEC("fentry/bpf_sk_storage_free")
int BPF_PROG(sk_storage_free, struct sock *sk)
{
        struct ct_key keys;

        keys = bpf_sk_storage_get(&conntrack_map_keys, sk,
0, 0);
        if (!keys)
                return 0;
          /* Delete the entries from the conntrack map. */
        bpf_map_delete_elem(&conntrack_map,
&keys.svc_entry);
        bpf_map_delete_elem(&conntrack_map,
&keys.egress_entry);

        return 0;
}
```

PoC based on socket storage driven GC

```
0: (79) r6 = *(u64 *)(r1 +0)

func 'bpf_sk_storage_free' arg0 has btf_id 3696 type
STRUCT 'sock'

1: R1=ctx() R6_w=ptr_sock()

; p = bpf_sk_storage_get(&socket_cookies, sock, 0,
0); @ socket_storage.c:129

1: (18) r1 = 0xffff98e0c0a90000       ;
R1_w=map_ptr(map=socket_cookies,ks=4,vs=16)

3: (bf) r2 = r6                       ; R2_w=ptr_sock()
R6_w=ptr_sock()

4: (b7) r3 = 0                        ; R3_w=0

5: (b7) r4 = 0                        ; R4_w=0

6: (85) call bpf_sk_storage_get#107

helper call is not allowed in probe

processed 6 insns (limit 1000000)
max_states_per_insn 0 total_states 0 peak_states 0
mark_read 0

libbpf: prog 'sk_storage_free': failed to load: -22
```

```
SEC("fentry/bpf_sk_storage_free")
int BPF_PROG(sk_storage_free, struct sock *sk)
{
        struct *ct_key keys;

        keys =
bpf_sk_storage_get(&conntrack_map_keys, sk, 0,
0);
        if (!p)
                return 0;

          /* Delete the entries from the conntrack map. */
        bpf_map_delete_elem(&conntrack_map,
&keys.svc_entry);
        bpf_map_delete_elem(&conntrack_map,
&keys.egress_entry);
        return 0;
}
```

Discussion

- Can we have callbacks that can be executed in socket storage free helper?
- Context for not allowing `bpf_sk_storage_get` in fentry/bpf_sk_storage_free?
- Can the limitation be lifted?
- Alternative: iterate over the entire map to delete entries with socket ip address
    - Not efficient :(

Thank You

Questions?

Life of a packet (external client to svc vip N-S)