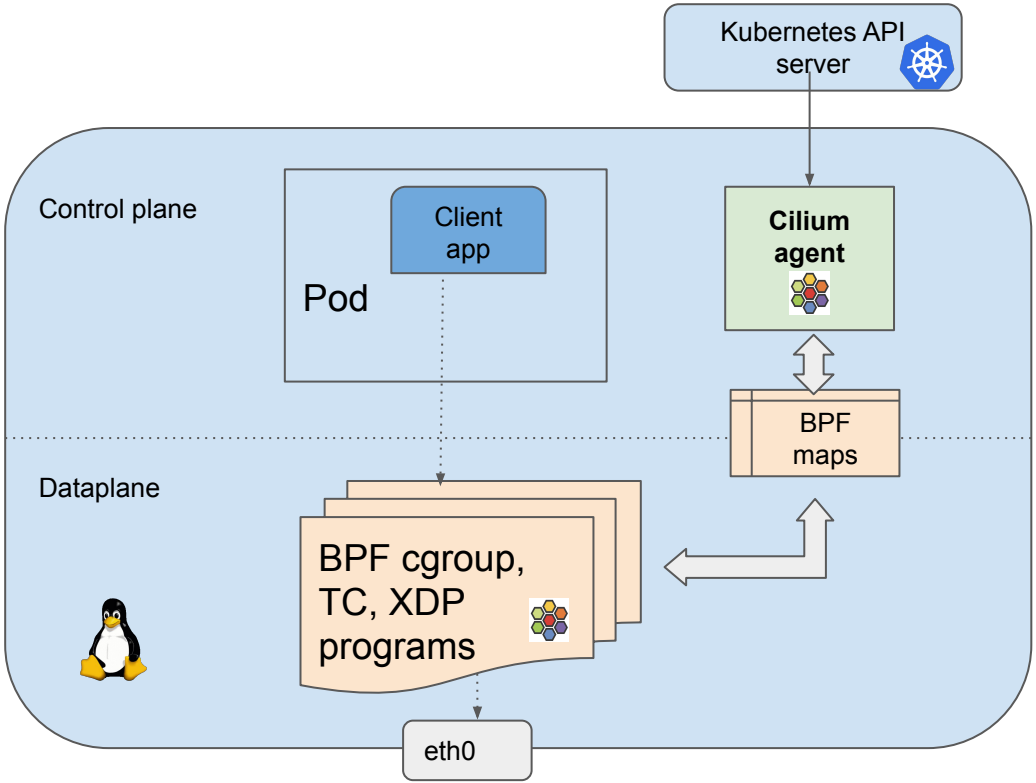
Socket termination with intent

Aditi Ghag
Software Engineer, Isovalent

Agenda

- Introduction to Cilium
- Motivation and use cases
- Evaluated approaches
- Notes from mailing list
- Next steps

Cilium high-level overview



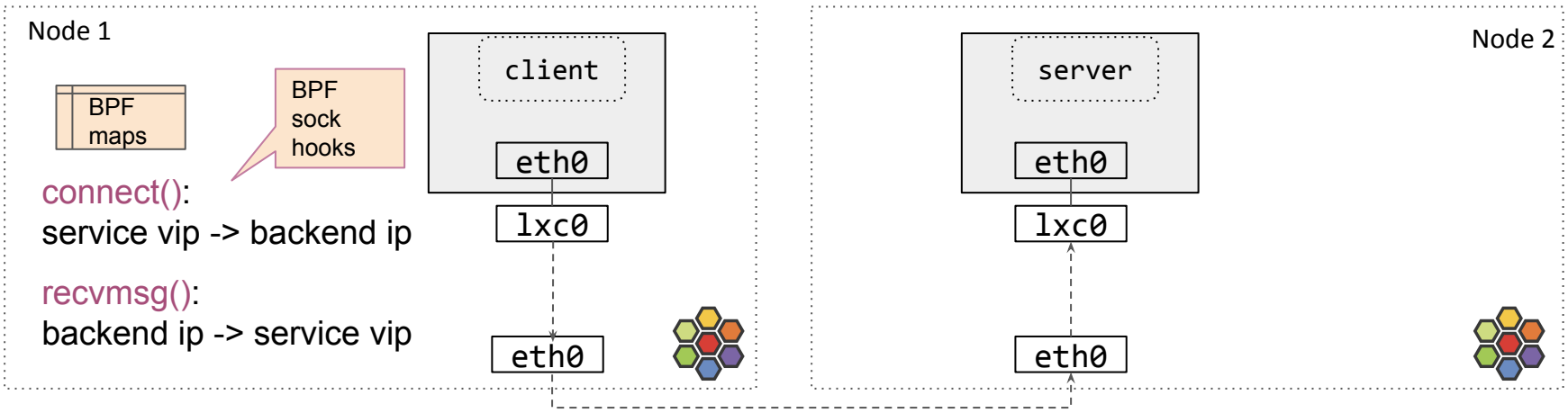
Control plane:

- Receives events for cluster entities
- Shares state with datapath via BPF maps

Dataplane:

- BPF programs executed on cgroup, tc, xdp hooks
- Load-balancing, policy enforcement, encapsulation, …

Use case #1: Socket load balancing in Cilium



- **BPF_PROG_TYPE_CGROUP_SOCK_ADDR**
- TCP, connected UDP: Service translation in connect() and recvmsg() events
- UDP: Service translation in sendmsg() and recvmsg()

What happens when remote backends go away?

- Service backend selection happens once at connect()
- TCP clients may get FIN/RST
- Connected UDP clients are unaware of disappeared remote backends
- Extended connectivity disruption for long-lived idle connections

Intent #1: Terminate client sockets connected to stale backends so that they can reconnect to active ones

Use case #2: Policy enforcement

- On-the-fly policies

- Example: Cilium supports BPF based node-local redirection for use cases like node-local DNS [1]. Consistent enforcement even for proxies with existing long-lived connections.

Intent #2: Terminate client socket connections prevented by applied policies

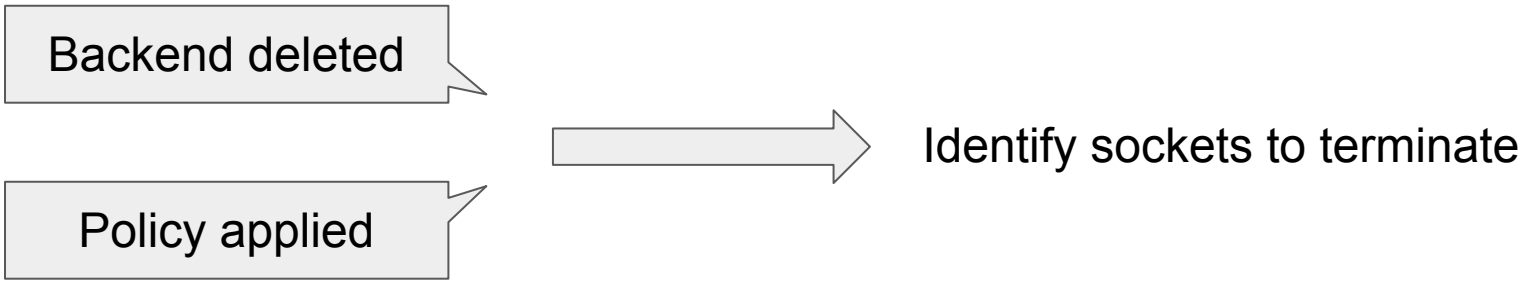
[1] https://kubernetes.io/docs/tasks/administer-cluster/nodelocaldns/

Socket termination with intent:

- How to filter sockets to terminate?

- How to forcefully terminate filtered sockets?

Step 1: Filtering sockets to terminate



SOCK_DIAG infrastructure

- Netlink based system to query sockets data
- Query supports filtering based on socket states
- Code highlights

```
struct nlmsghdr

nlh->nlmsg_type = SOCK_DIAG_BY_FAMILY

struct inet_diag_req_v2

diag_req->idiag_ext = INET_DIAG_INFO
```

SOCK_DIAG infrastructure

- Netlink based system to query sockets data
- Query supports filtering based on socket states
- Code highlights

```
struct nlmsghdr

nlh->nlmsg_type = SOCK_DIAG_BY_FAMILY

struct inet_diag_req_v2

diag_req->idiag_ext = INET_DIAG_INFO
```

- Limited filtering capability
- Requires entering all network namespaces

BPF (sockets) iterator

- Iterator makes kernel data available to BPF programs
- Facilitates flexible filtering of sockets
- Most up-to-date view of sockets for further processing

BPF (sockets) iterator

- Iterator makes kernel data available to BPF programs
- Facilitates flexible filtering of sockets
- Most up-to-date view of sockets for further processing

- Network namespace aware

Step 2: Terminating filtered sockets

Unreachable routes

- Add unreachable routes for deleted backends

```
ip route add unreachable 192.168.60.11/32
```

Unreachable routes

- Add unreachable routes for deleted backends

  ip route add unreachable 192.168.60.11/32

- Effective only for new connections: not too useful as Cilium supports graceful termination

- ICMP errors are ignored in established state (TCP and connected UDP)

sock_destroy

- Invokes handlers to abort sockets (protocols: TCP, UDP, raw sockets)
- Sets socket error to **ECONNABORTED**
- Disconnect/Send RST
- Currently exposed via Netlink:

```
struct nlmsghdr

nlmsg_type = SOCK_DESTROY

populate socket fields retrieved from SOCK_DIAG
```

sock_destroy

- Invokes handlers to abort sockets (protocols: TCP, UDP, raw sockets)
- Sets socket error to **ECONNABORTED**
- Disconnect/Send RST
- Currently exposed via Netlink:

  `struct nlmsghdr`

  `nlmsg_type = SOCK_DESTROY`

  `populate socket fields retrieved from SOCK_DIAG`

- Disabled by default behind CONFIG_INET_DIAG_DESTROY
- Network namespace checks
- No BPF helper

Notes from mailing list

- Sent RFC with use cases and evaluated approaches on the mailing list
- People see value in having a global BPF (socket) iterator
- Suggestions for an “all-netns” socket iterator target for TCP and UDP
- sock_destroy, or connecting to other backends in some cases

Thanks Martin and others for the discussions!

Next steps

- RFC patches for adding a BPF helper to abort sockets tested using selftests

  New self test mirrors intended usage of API

```
BPF_CALL_2(bpf_sock_destroy, struct sock *, sk,  int, err)
{
        bool lock = false;
        sk = sk_to_full_sk(sk);

        if (!sk || !sk_fullsock(sk))
                return 0s;
        if (!sk->sk_prot->diag_destroy)
                return -EOPNOTSUPP;

        return sk->sk_prot->diag_destroy(sk, err, lock);
}
```

Putting it all together

- Iterate over sockets upon LB events using BPF iterator

- Filter sockets based on the destination address or socket metadata

- Invoke BPF helper for sock_destroy

Putting it all together

- Iterate over sockets upon LB events using BPF iterator
- Filter sockets based on the destination address or socket metadata
- Invoke BPF helper for sock_destroy

Optimization: Can we get away from iterating over all sockets?

- Cilium BPF programs record client sockets to backend mappings
- BPF_MAP_TYPE_SOCKHASH to store client sockets

Summary

- Load balancing and policy enforcement may need forceful socket termination
- BPF iterator to filter sockets
- BPF helper for sock_destroy internal API to terminate sockets

Open questions/discussions

- Connecting to other UDP backends for some use cases instead of abort?
- Per netns socket hash tables

Thank you!
Questions?