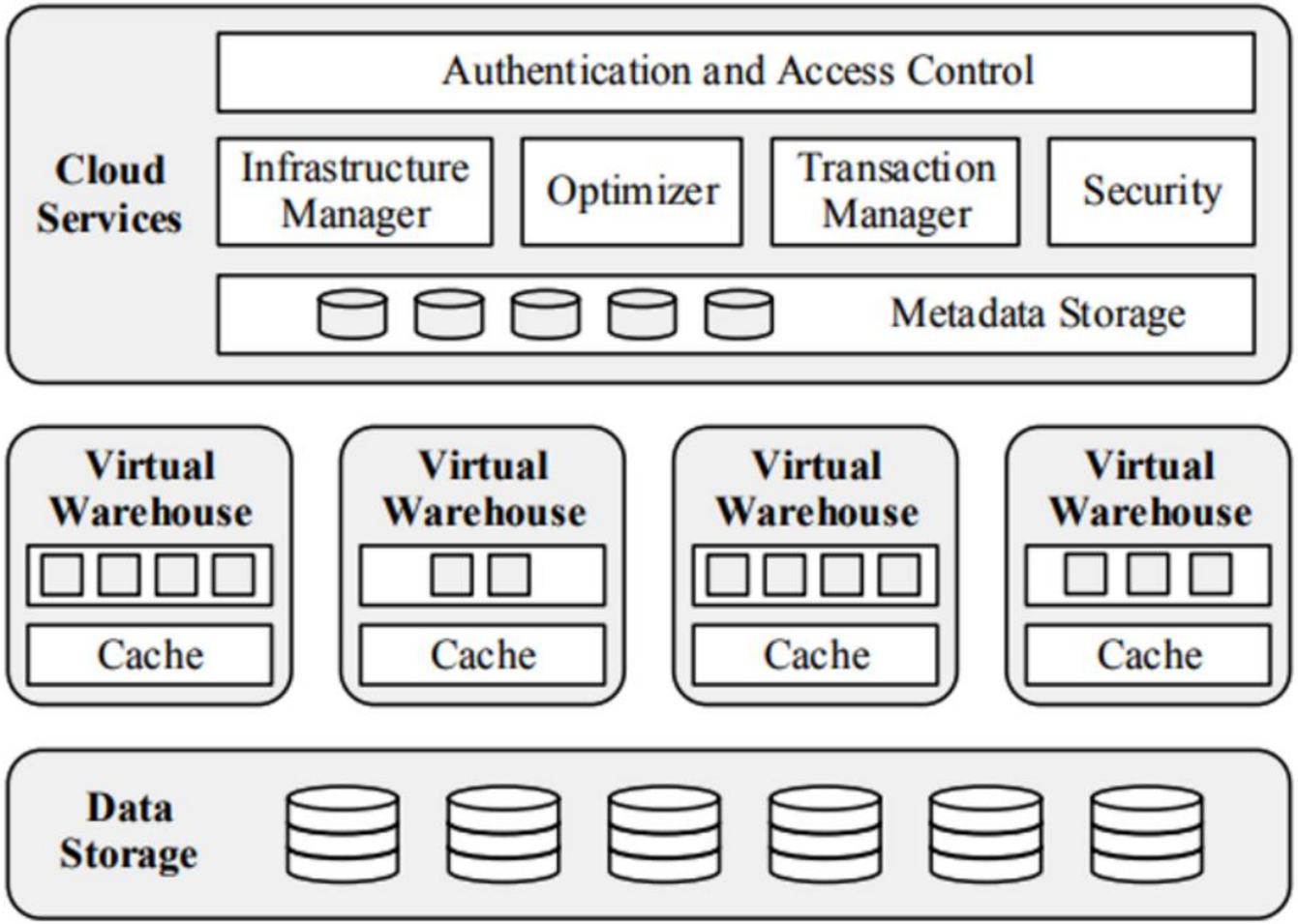
Secure Egress In Serverless Compute

Haowei Yu

Agenda

- Snowflake Introduction
- Serverless in Snowflake
- eBPF Introduction
- How to leverage eBPF to support secure egress
- Closing

Snowflake Introduction - Architecture



Serverless in Snowflake

- Snowflake Abstract Query Execution Resource as Virtual Warehouse
  - “Semi Serverless”
  - Users still need to configure warehouse size (number of servers in cluster)
- Data Loading Service v1
  - Auto-scalable
  - Users only need to define transformation logic (by sql functions)
  - Might co-allocation tasks from different customers on the same physical ec2 instances
    - COGS

Data Loading Service - v3

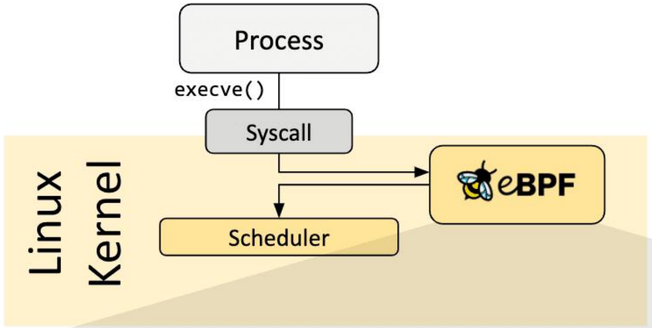
- Users want to define data transformation logic by writing Java/Python Code
- Snowflake expose UDF operator
- Implementation: need an untrusted user code execution environment
- Solution
  - Sandbox user code execution in the local worker node
  - Lock down network access from worker node by default
    - All egress traffic needs to go through a fleet of egress proxies
      - Proxies are also internet gateway
    - Audit
    - Prevent data exfiltration

The problem not solved

- User code needs to connect to endpoint on public internet
  - Geocode
  - Weather Data
  - SaaS Endpoint Connection
- Admin should be able to config which endpoints is (dis)allowed
  - Packet that goes to un-desired endpoints should be dropped
- Proxy is transparent to user code
  - eBPF comes to rescue

eBPF Introduction

- Run sandboxed program in Linux kernel
- Without changing source code
- Event-driven on different hook points
  - System calls
  - Network Event
  - Kprobe




```c
int syscall__ret_execve(struct pt_regs *ctx)
{
        struct comm_event event = {
                .pid = bpf_get_current_pid_tgid() >> 32,
                .type = TYPE_RETURN,
        };

        bpf_get_current_comm(&event.comm, sizeof(event.comm));
        comm_events.perf_submit(ctx, &event, sizeof(event));

        return 0;
}
```

How to use eBPF?

- Write C code and compile with clang
  - Target is eBPF byte code
- Load ebpf byte code with bpf syscall
  - Libbpf wraps different bpf syscall
  - Higher libraries like tc linked with libbpf
- Interact with your eBPF code from userspace with maps
  - Configure the program
  - Collect stats/results
- Verifier
  - Null pointer dereference is not allowed
  - Loop is not allowed in eBPF

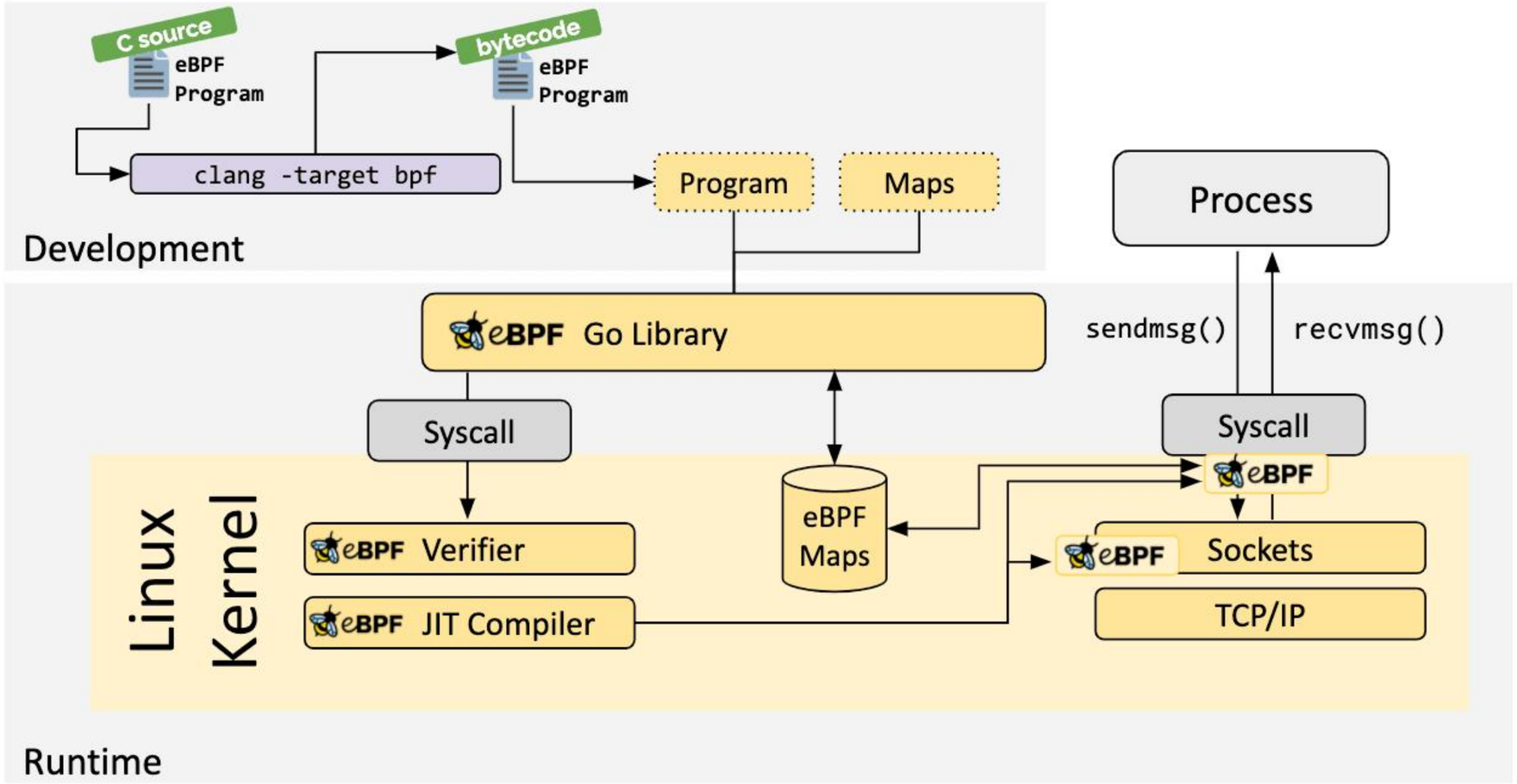
How to use eBPF?

- Write C code and compile with clang
  - Target is eBPF byte code
- Load ebpf byte code with bpf syscall
  - Libbpf wraps different bpf syscall
  - Higher libraries like tc linked with libbpf
- Interact with your eBPF code from userspace with maps
  - Configure the program
  - Collect stats/results
- Verifier
  - Null pointer dereference is not allowed
  - Loop is not allowed in eBPF

eBPF Maps

- Different Data Structures
  - (LRU) Hashmap
  - (LRU) Array
  - LPM
  - Ring Buffer
- Usage
  - A communication channel between kernel and userspace
    - Collect time spent on each syscall
  - Share state between different ebpf function invocation
    - Share state between different packet processing

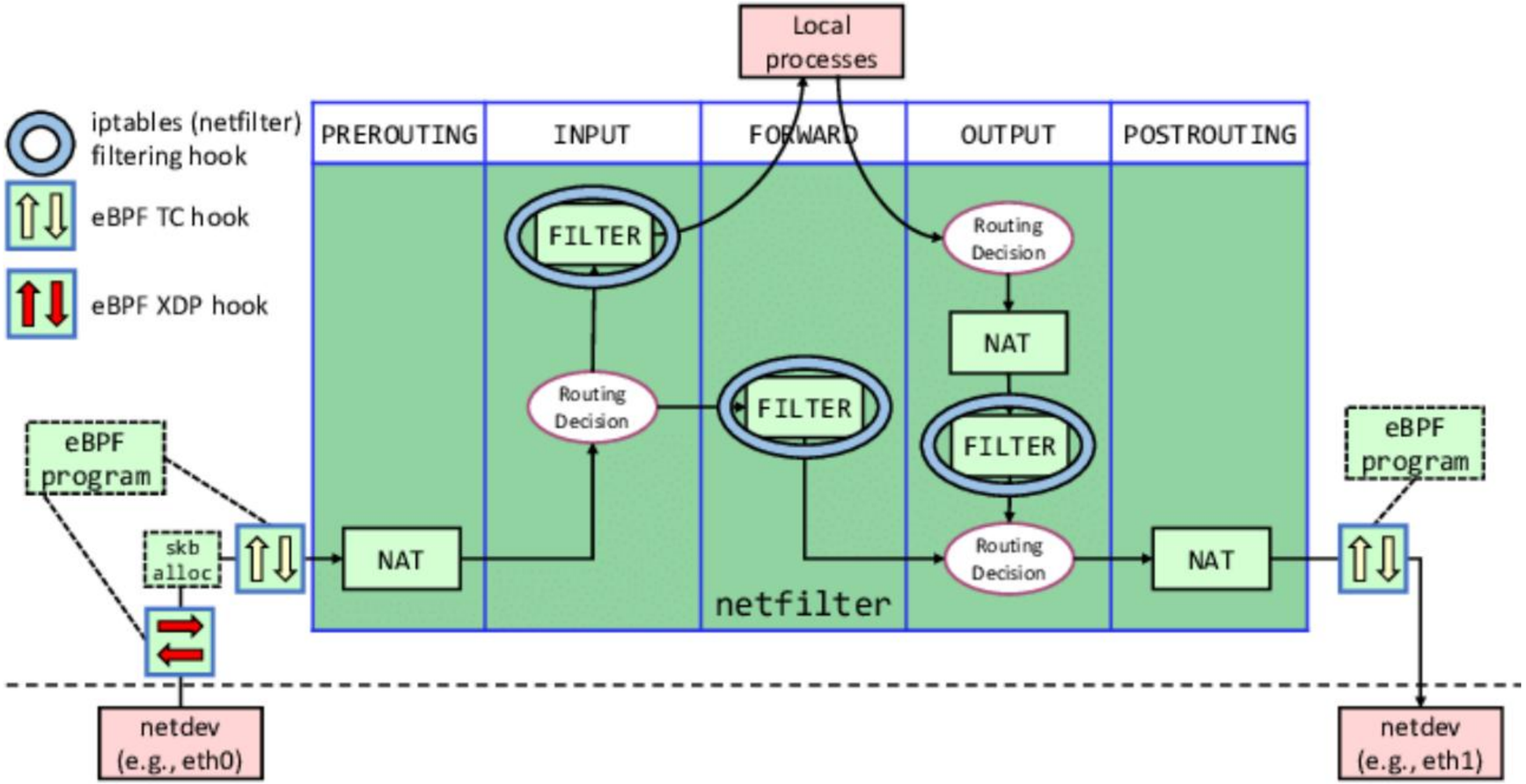
Put it all together



eBPF In Networking

- Monitoring
  - Within network stack on a single machine
  - Between machines inside a cluster
- Packet Forwarding/Redirect
  - Easier to manager than IP table
- Rate limiting
- Policy Enforcement

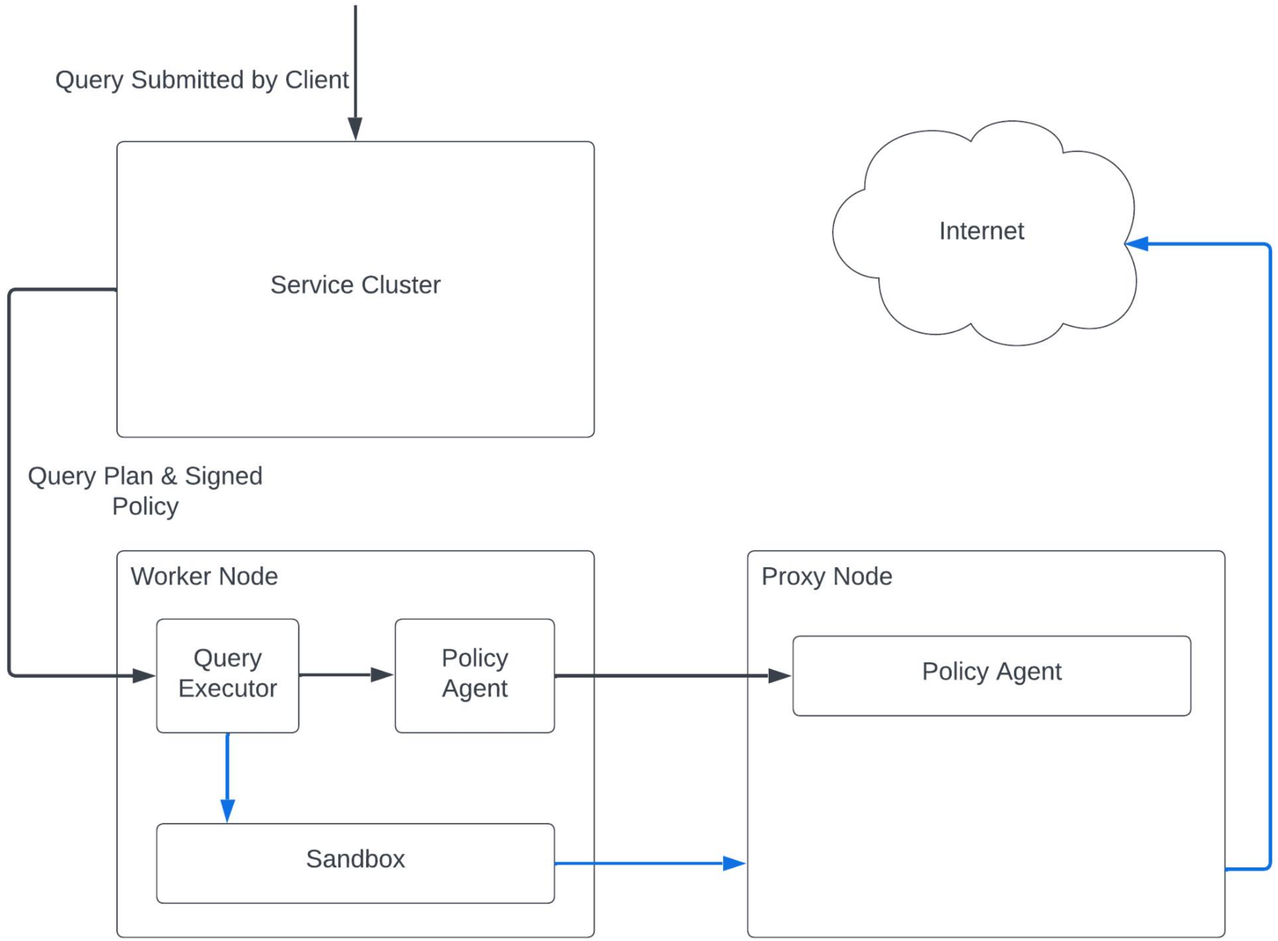
Attach eBPF in XDP and TC



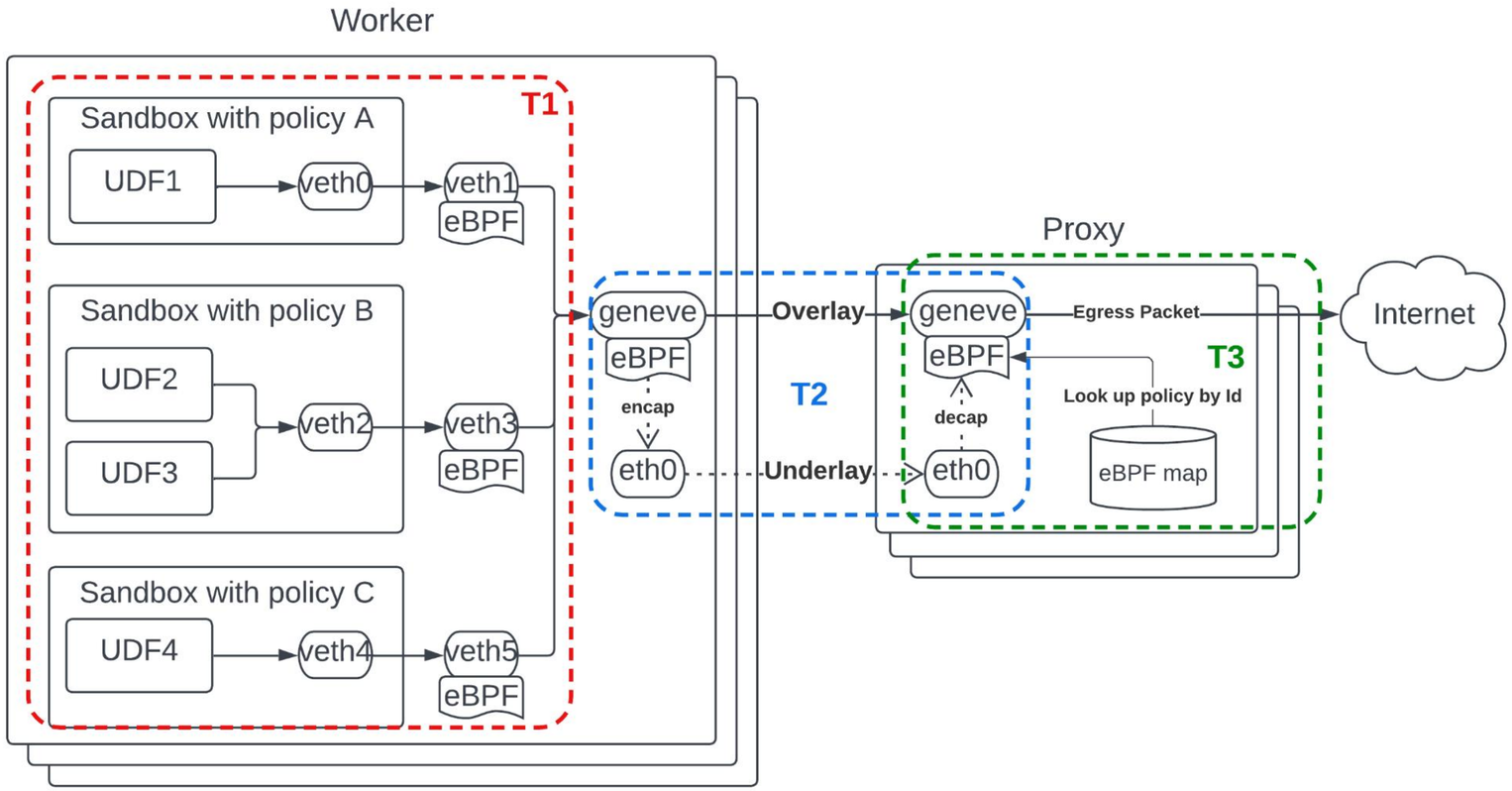
eBPF In Networking

- Monitoring
  - Within network stack on a single machine
  - Between machines inside a cluster
- Packet Forwarding/Redirect
  - Easier to manager than IP table
- Rate limiting
- Policy Enforcement

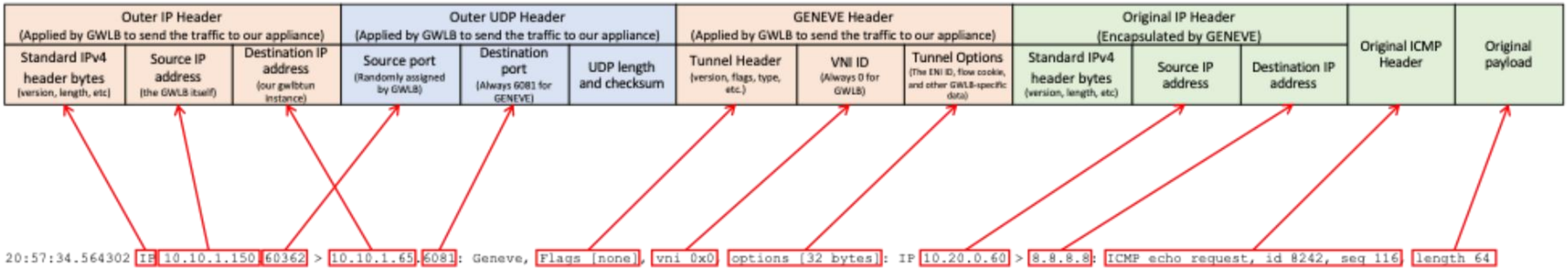
Solution - Control Flow



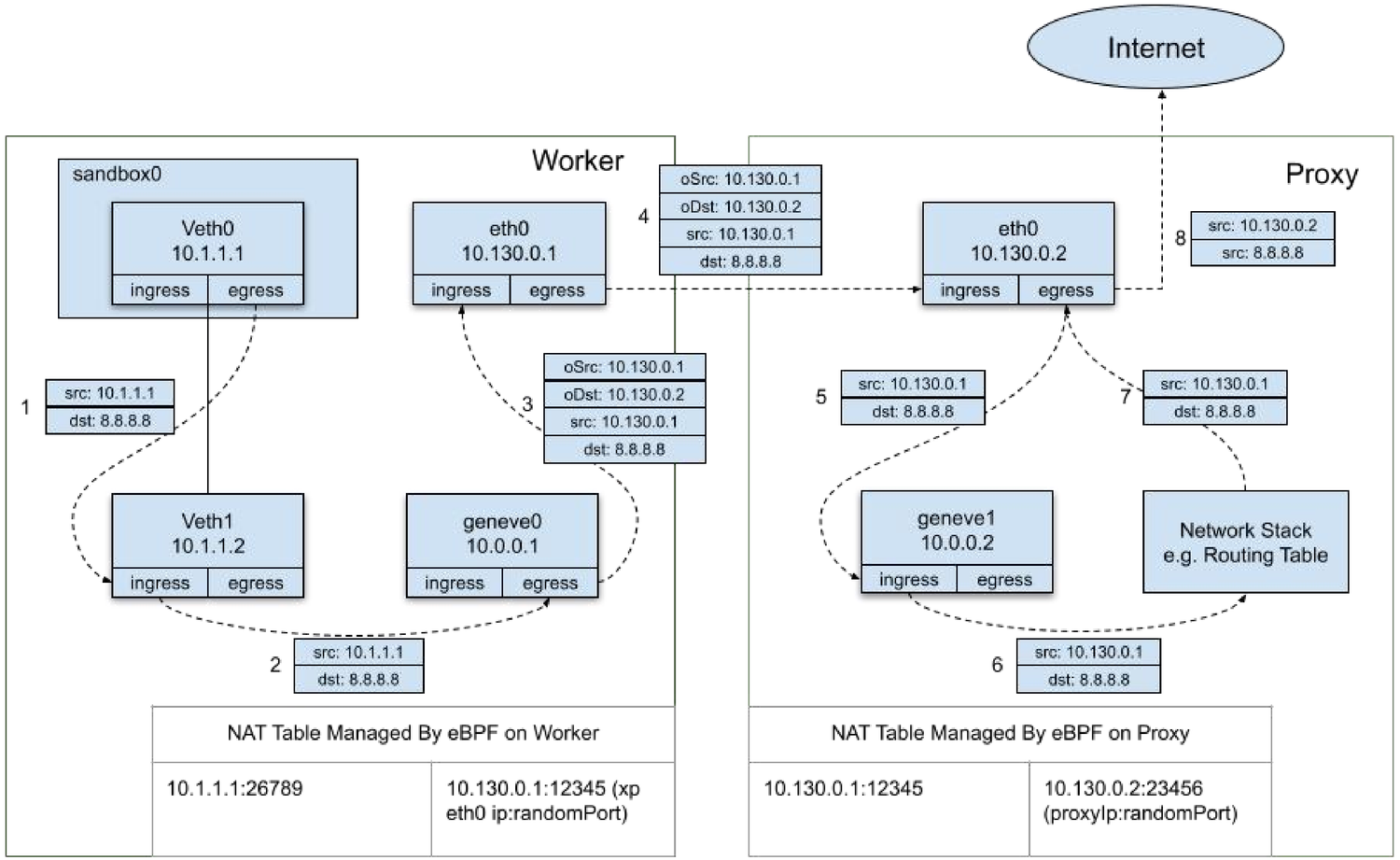
Solution - High Level



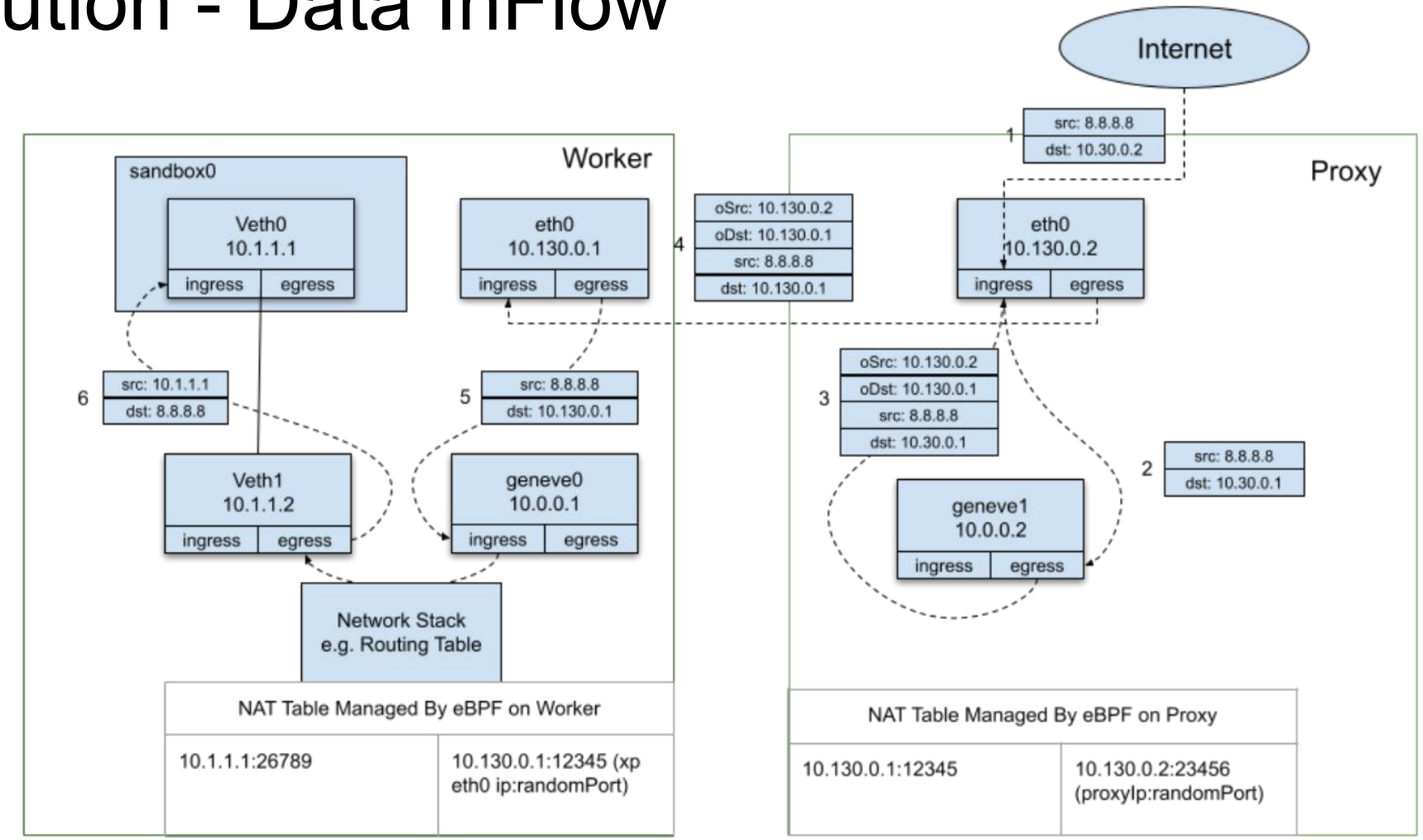
Solution - Packet Encapsulation



Solution - Data OutFlow



Solution - Data InFlow



Closing

- Connection to private endpoint
- Other stats/metric that can be collected from eBPF
- LSM BPF for advanced security features

THANKS

软件正在重新定义世界

Software Is Redefining The World