DPDK SUMMIT
Powering the Future of Networking Software

12–13 May 2026

Packet capture tool based on eBPF for DPDK (netcap)

ByteDance STE - Kernel Network

@Tengteng Yang

• Introduction

Introduction

- Standard tcpdump has zero visibility into DPDK traffic, as DPDK packets bypass the kernel network stack entirely
- Existing DPDK packet capture solutions require intrusive modifications to application code and full recompilation

Comparison	tcpdump	netcap
target traffic	kernel network stack	DPDK userspace + kernel skb
capture point	tc ingress or egress	any function in pipeline
filter syntax	tcpdump	tcpdump

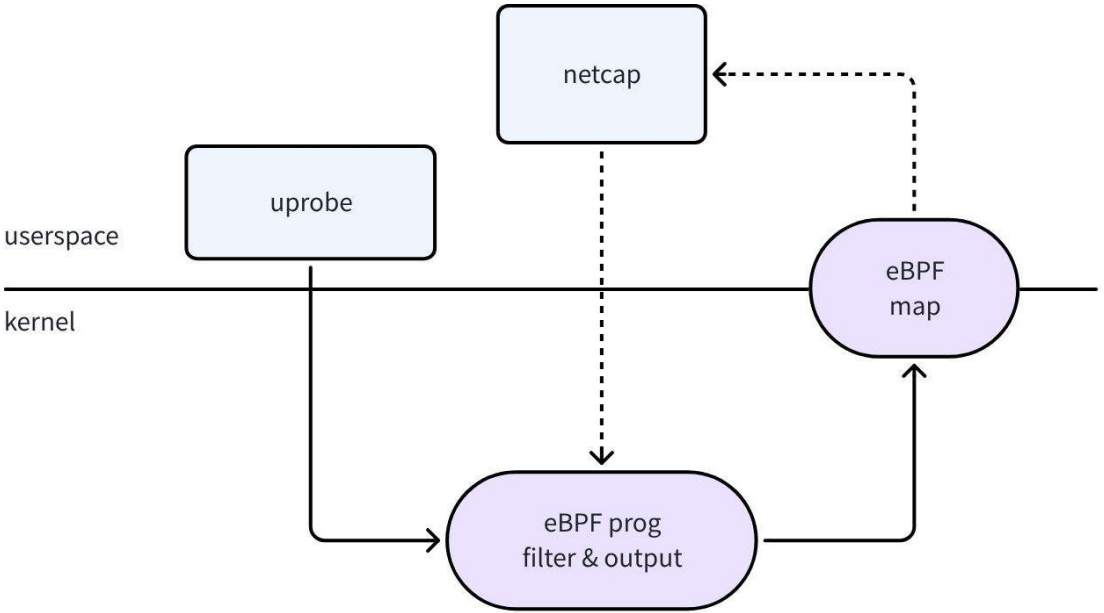
Introduction

- Existing approaches for DPDK packet capture

Comparison	DPDK pdump + librte_bpf	netcap
target traffic	DPDK userspace	DPDK userspace + kernel skb
attach model	app-integrated (link-time)	external uprobe (runtime)
capture point	pdump-instrumented hooks only	any function in pipeline
filter	librte_bpf (in-process)	tcpdump

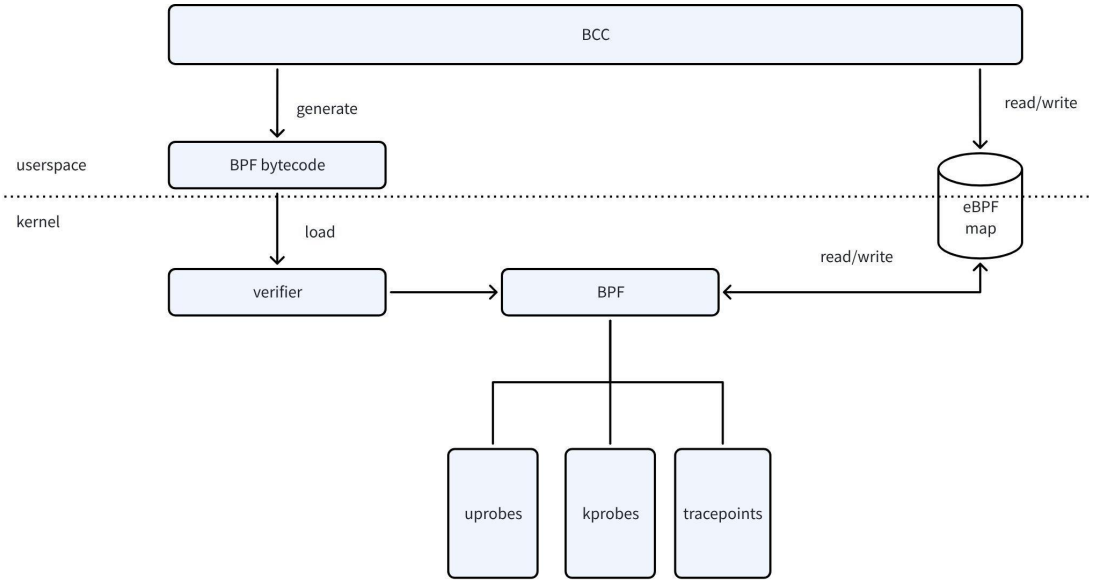
Introduction

- Supports flexible packet capture points in the DPDK processing pipeline
- Fully compatible with BPF/tcpdump filters



Introduction

- eBPF-based hooking for safety and flexibility
- Uses BCC for bytecode generation, loading, and attaching



Introduction

- Netcap command line interface

```
netcap help mbuf
Dump mbuf with tcpdump expression

Usage:
  netcap mbuf [flags]

Examples:
## mbuf mode is for the DPDK app

# capture dpdk function with tcpdump filter expression:
netcap mbuf -f you_func@1 -e "tcp and port 80" -t "-nnve" --pid 1111

# capture dpdk vector mbufs function, such as: vec_func(struct rte_mbuf **mbufs,
uint16_t mbuf_size):
netcap mbuf -f vec_func@1@2 -e "tcp and port 80" -t "-nnve" --pid 1111
```

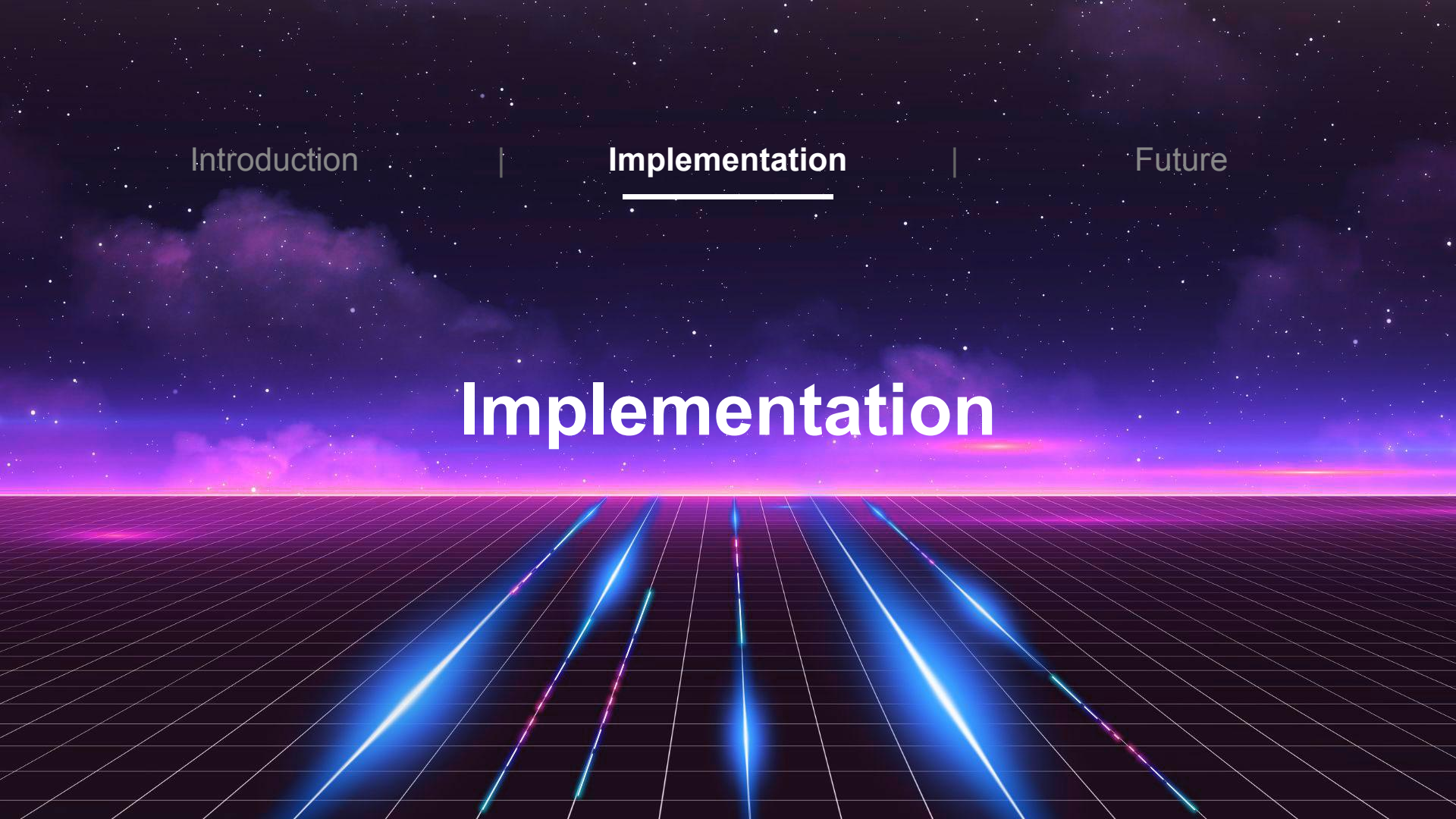
```
netcap help skb
Dump skb with tcpdump expression

Usage:
  netcap skb [flags]

Examples:
## skb mode is for the kernel

# simple used to caputre skb in icmp_rcv with tcpdump filter expression:
netcap skb -f icmp_rcv@1 -e "host 192.168.0.2"

# capture at tracepoint, and DON'T need @param at tracepoint
netcap skb -f tracepoint:net:netif_receive_skb -i eth0 -e "host 192.168.0.2"

# capture pcap with dump kstack:
netcap skb -f tracepoint:skb:kfree_skb -e "host 192.168.0.2" -S

# capture kernel function dev_queue_xmit at eth0 with tcpdump flags -nnve:
netcap skb -f dev_queue_xmit@1 -e "tcp and port 80" -i eth0 -t "-nnve"
```

Introduction

- Example: packet capture in a DPDK environment

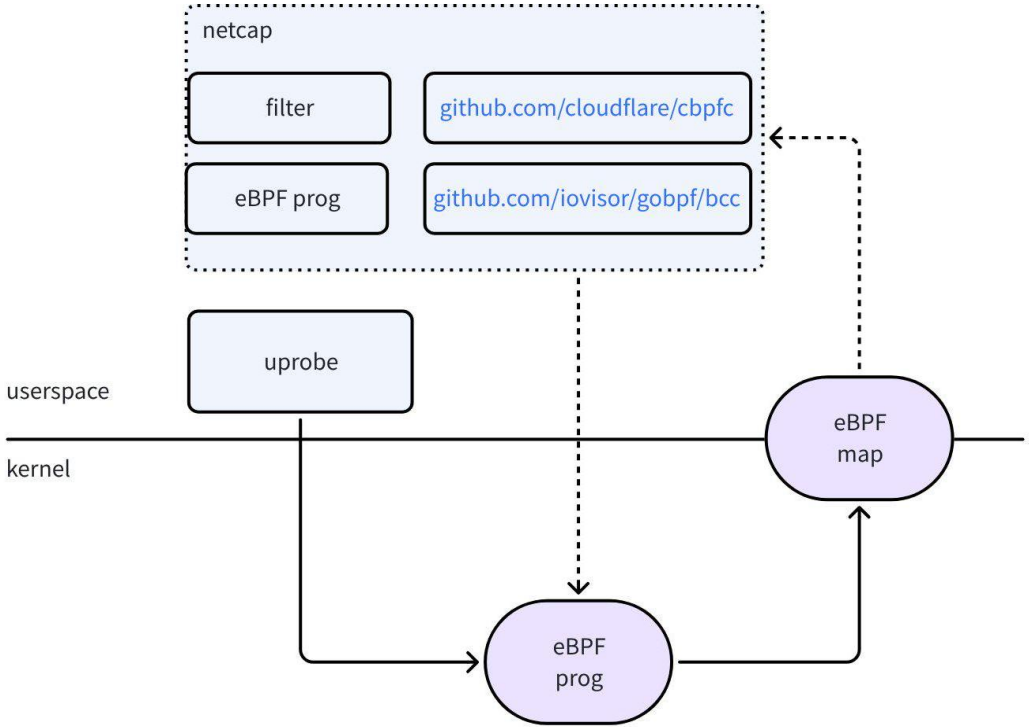
```
netcap mbuf -f l2fwd_simple_forward@1  -e "tcp and port 80" -t "-nnve" --pid $(pidof l2fwd)

17:37:29.075629 52:54:63:22:21:10 > 52:54:63:22:22:10, ethertype IPv4 (0x0800), length 74: (tos 0x0, ttl 64, id 49060, offset 0, flags [DF], proto TCP (6), length 60)
    192.168.1.221.20104 > 192.168.1.222.80: Flags [S], cksum 0x120d (correct), seq 476663885, win 42340, options [mss 1460,sackOK, TS val 1507607398 ecr 0,nop,wscale 10], length 0
17:37:30.130728 52:54:63:22:21:10 > 52:54:63:22:22:10, ethertype IPv4 (0x0800), length 74: (tos 0x0, ttl 64, id 49061, offset 0, flags [DF], proto TCP (6), length 60)
    192.168.1.221.20104 > 192.168.1.222.80: Flags [S], cksum 0x0dee (correct), seq 476663885, win 42340, options [mss 1460,sackOK, TS val 1507608453 ecr 0,nop,wscale 10], length 0
```

Implementation



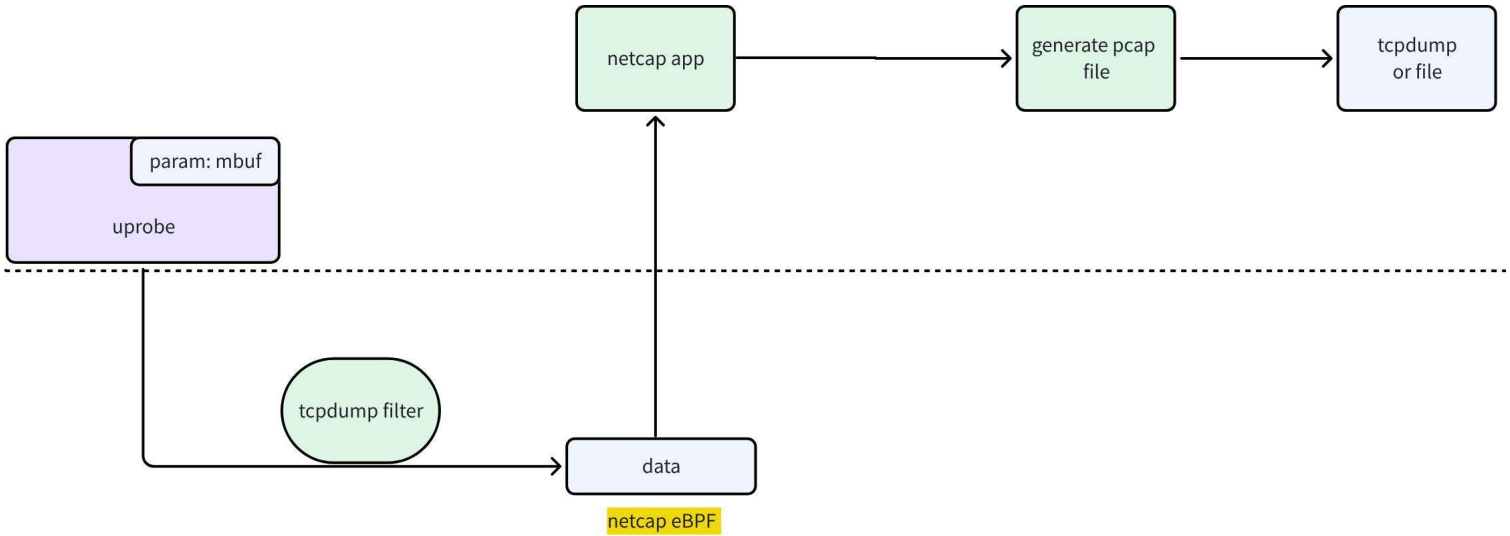
Implementation

- System architecture



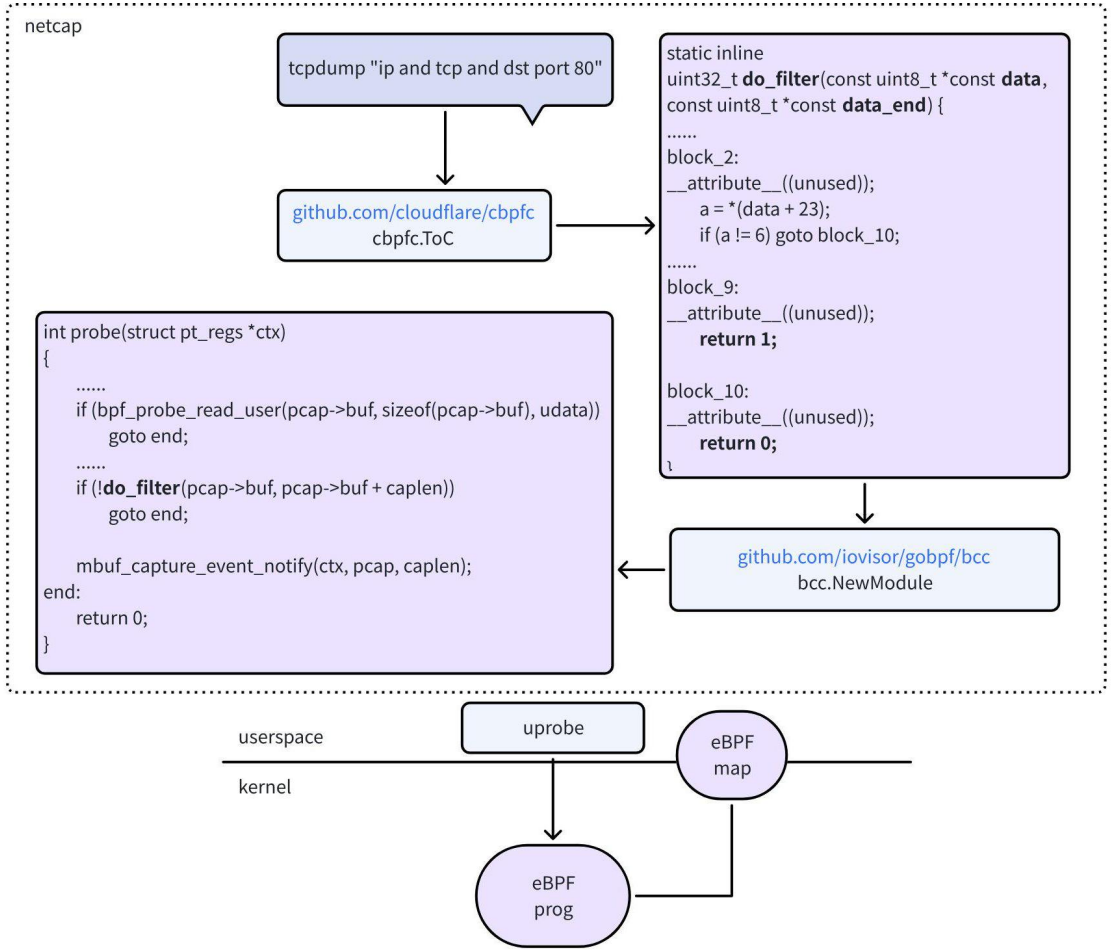
Implementation

- Overall processing pipeline



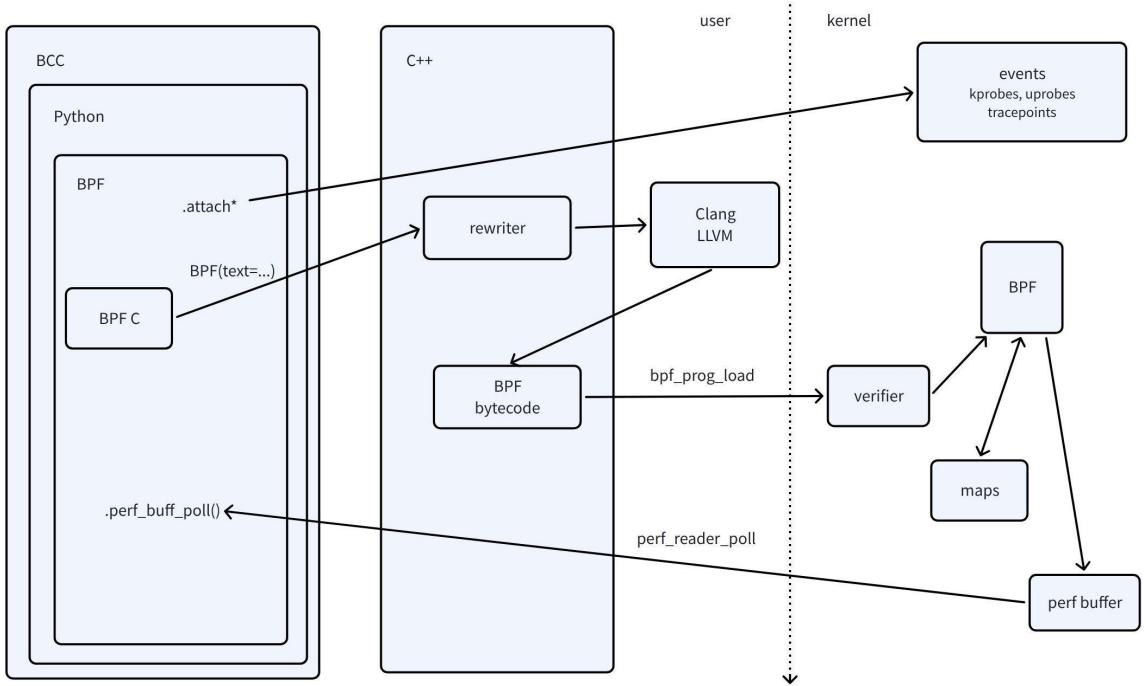
Implementation

- Converting tcpdump filter syntax to C functions



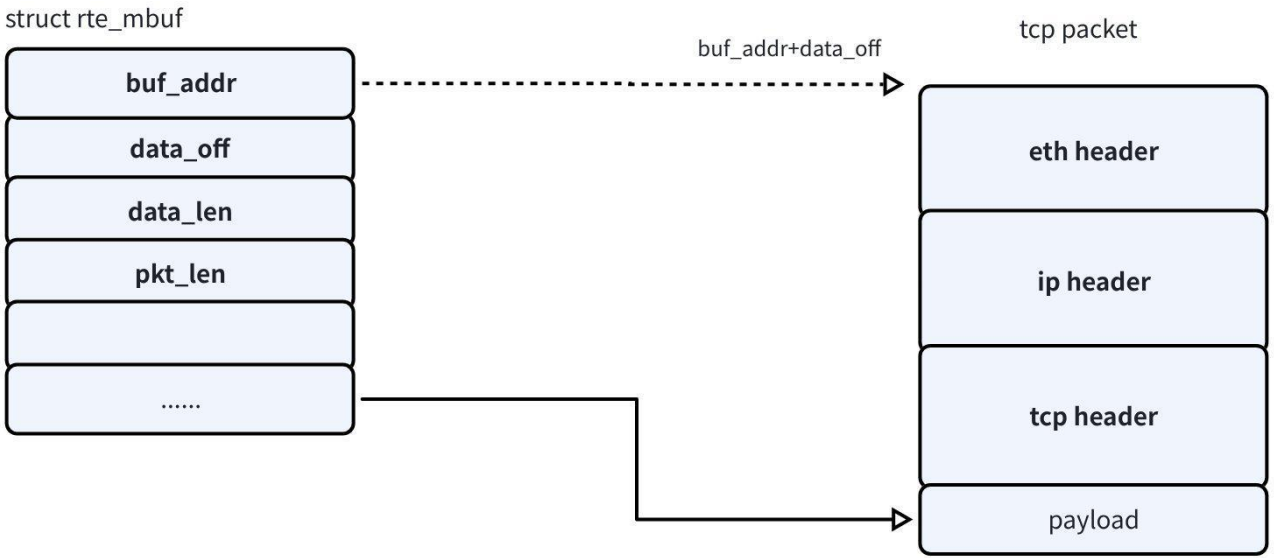
Implementation

- BCC integrates LLVM to JIT-compile and generate bytecode, then loads and attaches it

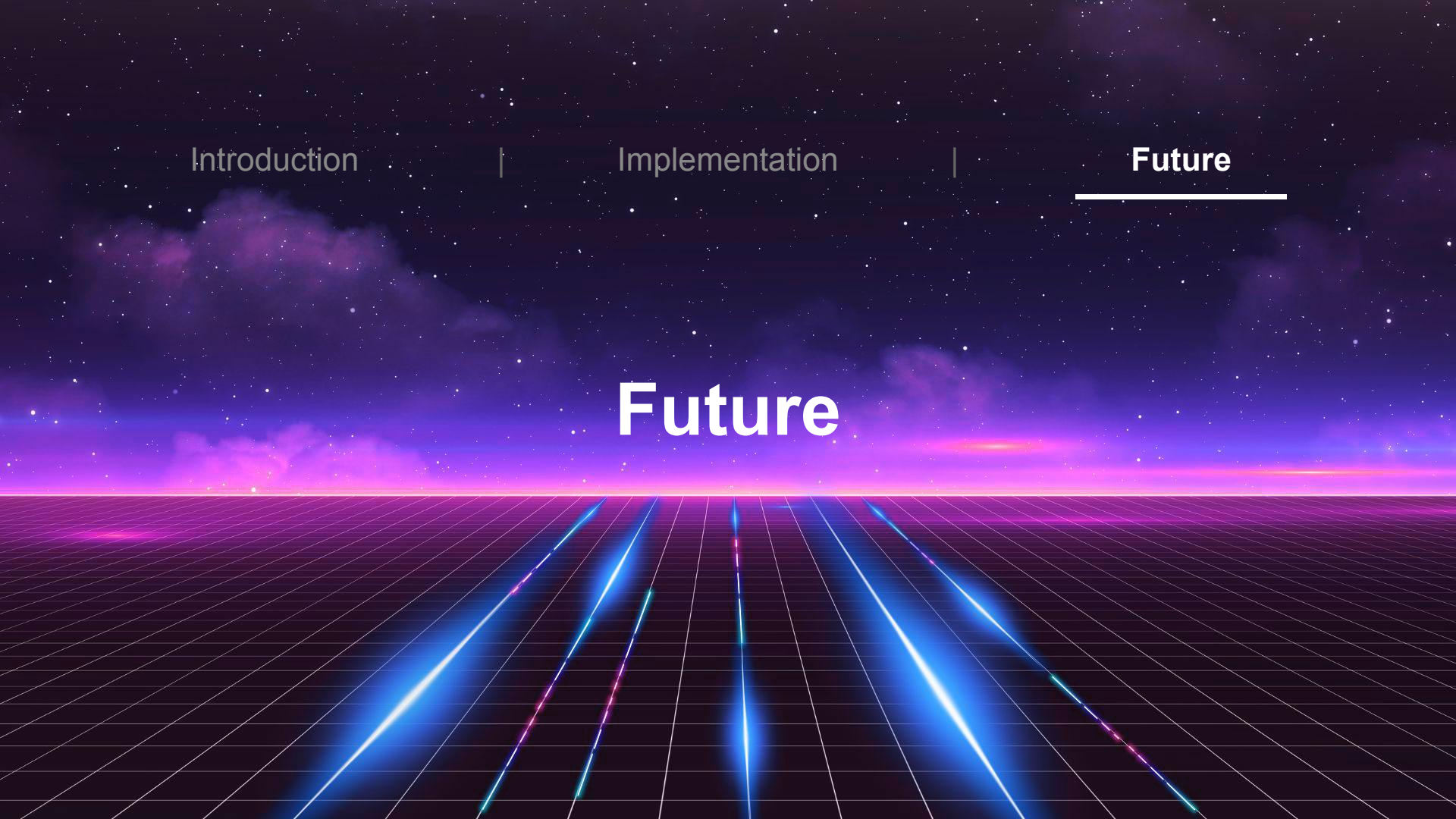


Implementation

- Mapping between mbuf structure and packet data



Future



Future

- Performance: reduce uprobe overhead with bpftime

Operation	kernel uprobe (ns)	bpftime userspace (ns)	speed up
uprobe trigger	2561.57	190.02	13.48×

Source: bpftime official benchmark (eunomia-bpf)

- Open source: https://github.com/bytedance/netcap

Thank you– Questions are welcome

ByteDance STE - Kernel Network