SOSCON

Faster Packet Processing in Linux: XDP

Kosslab | **Software Engineer** | 이호연
2019.10.17

이호연 @Daniel T. Lee

- Kosslab Software Engineer

- Opensource Developer
(Linux Kernel – BPF, XDP, uftrace, etc.)

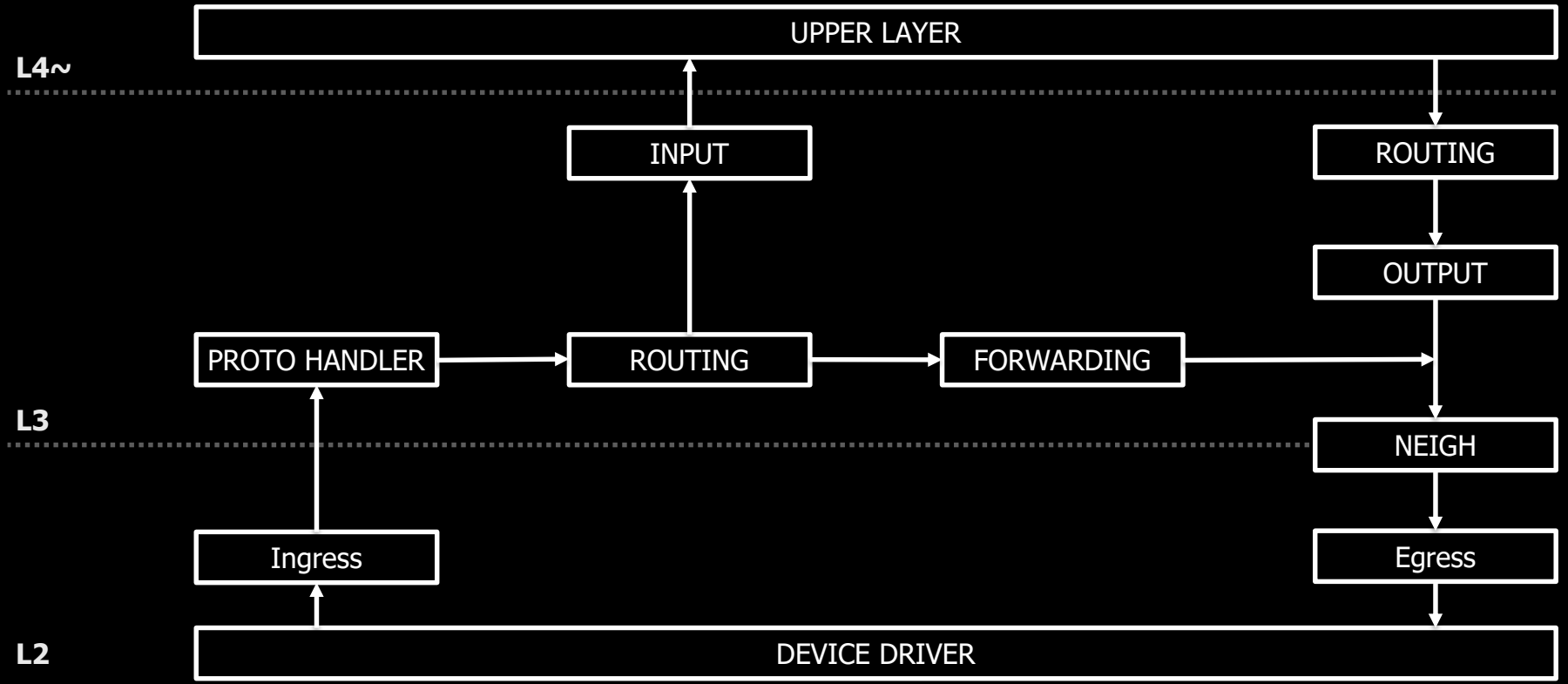
Today’s agenda

- Packet Processing in Linux
- How fast are we talking?
- What is XDP?
- How to use XDP?
- More about XDP

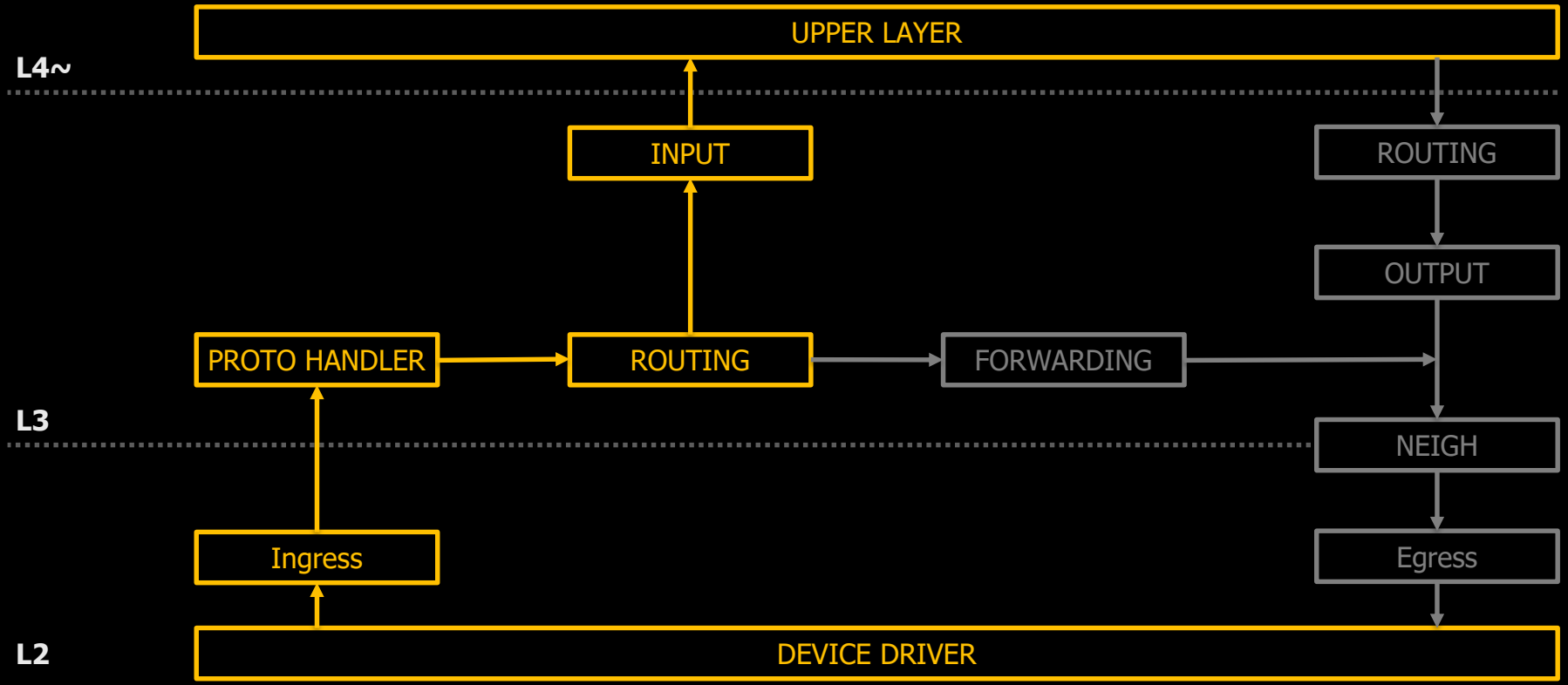
Packet Processing in Linux

From basic path to processing hooks

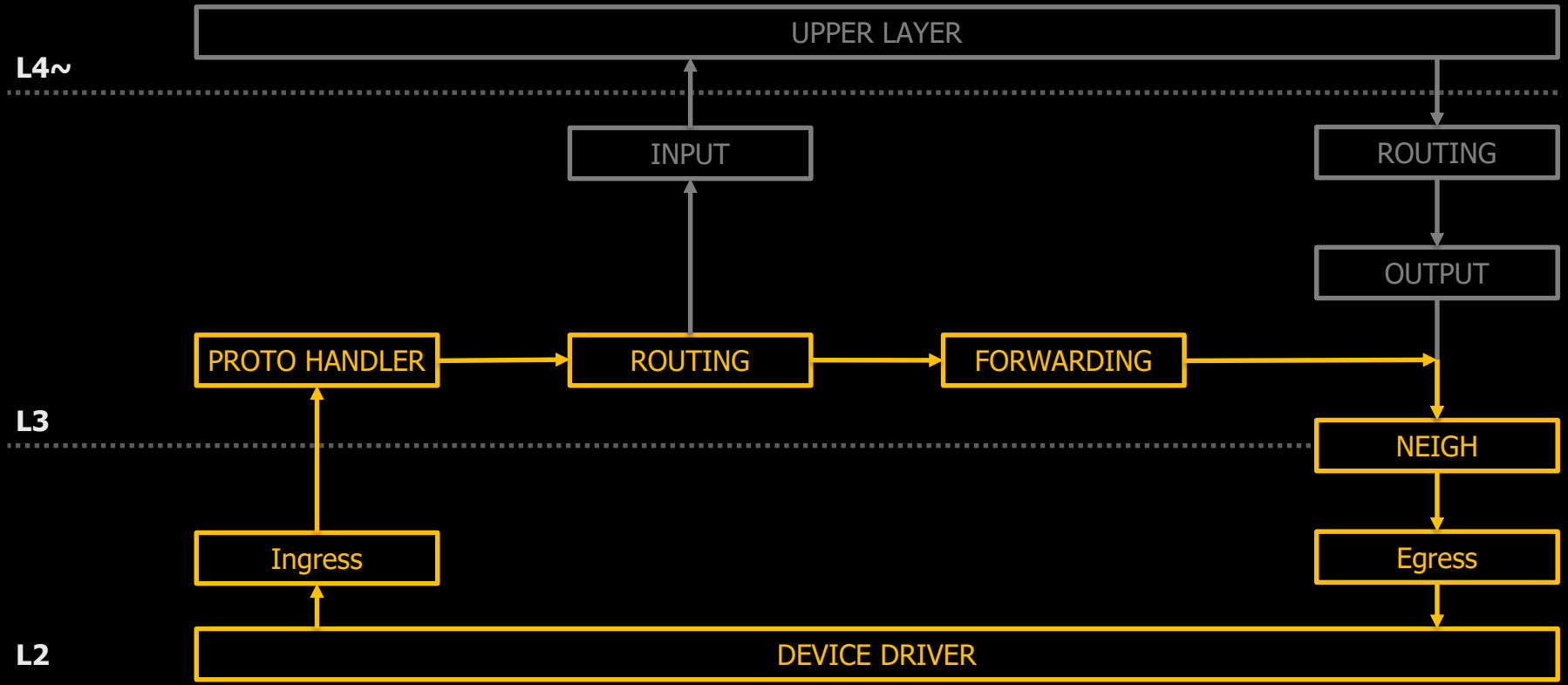
Packet Path in Kernel



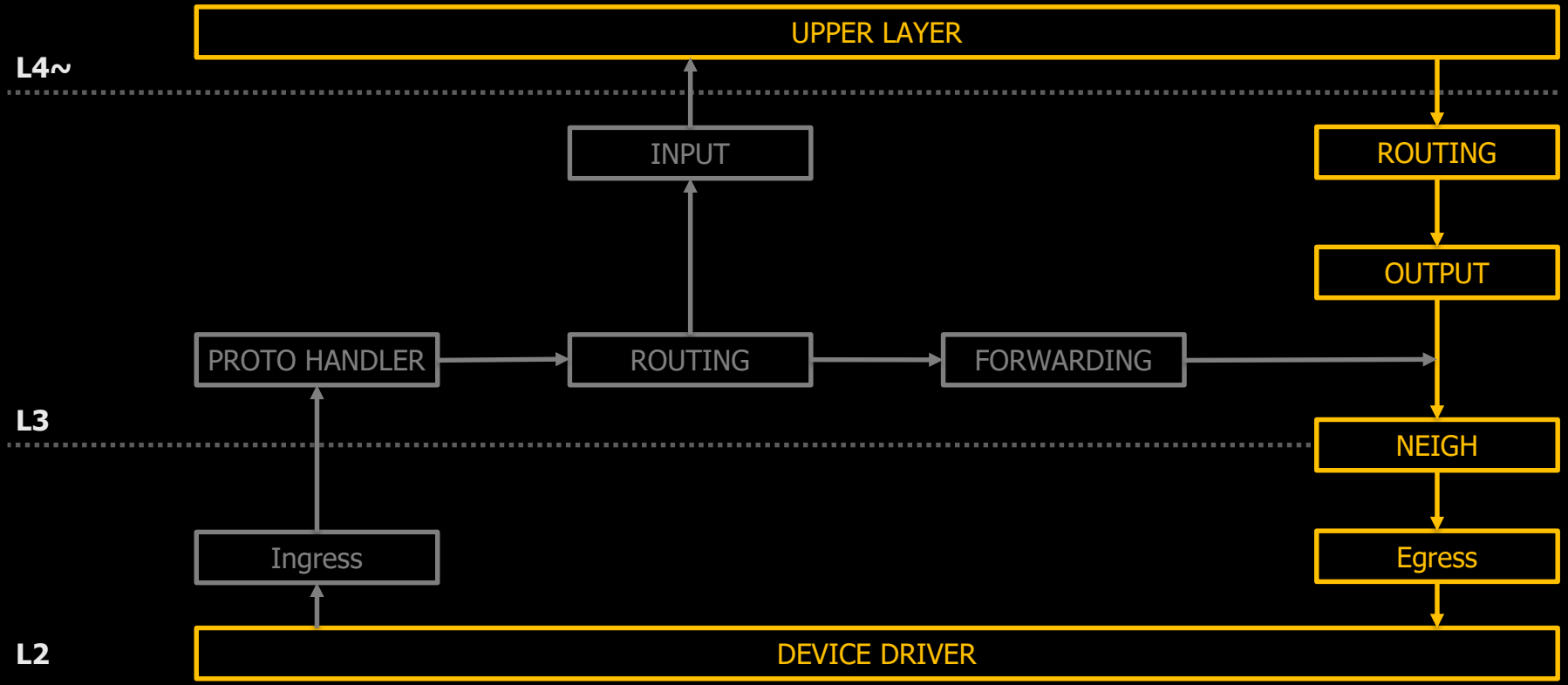
Packet Receiving Path



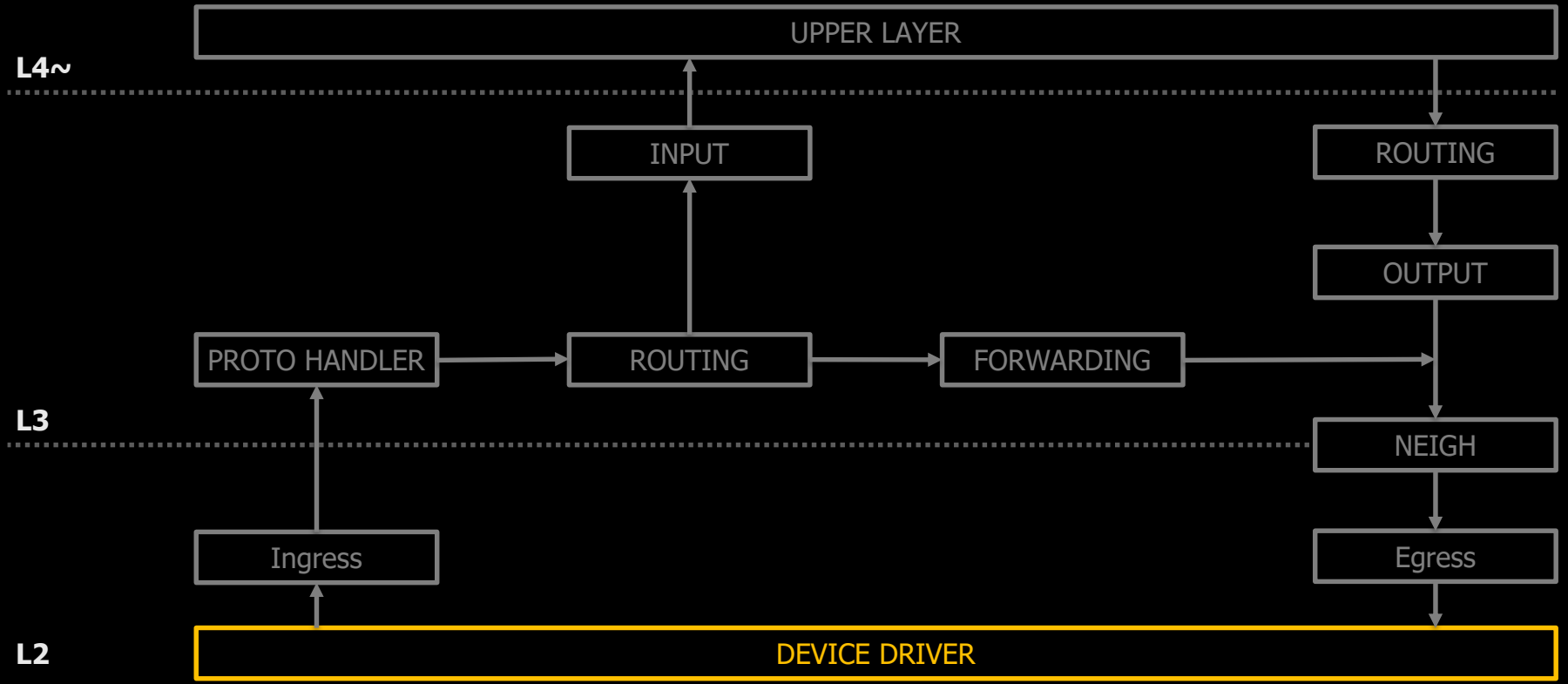
Packet Forwarding Path



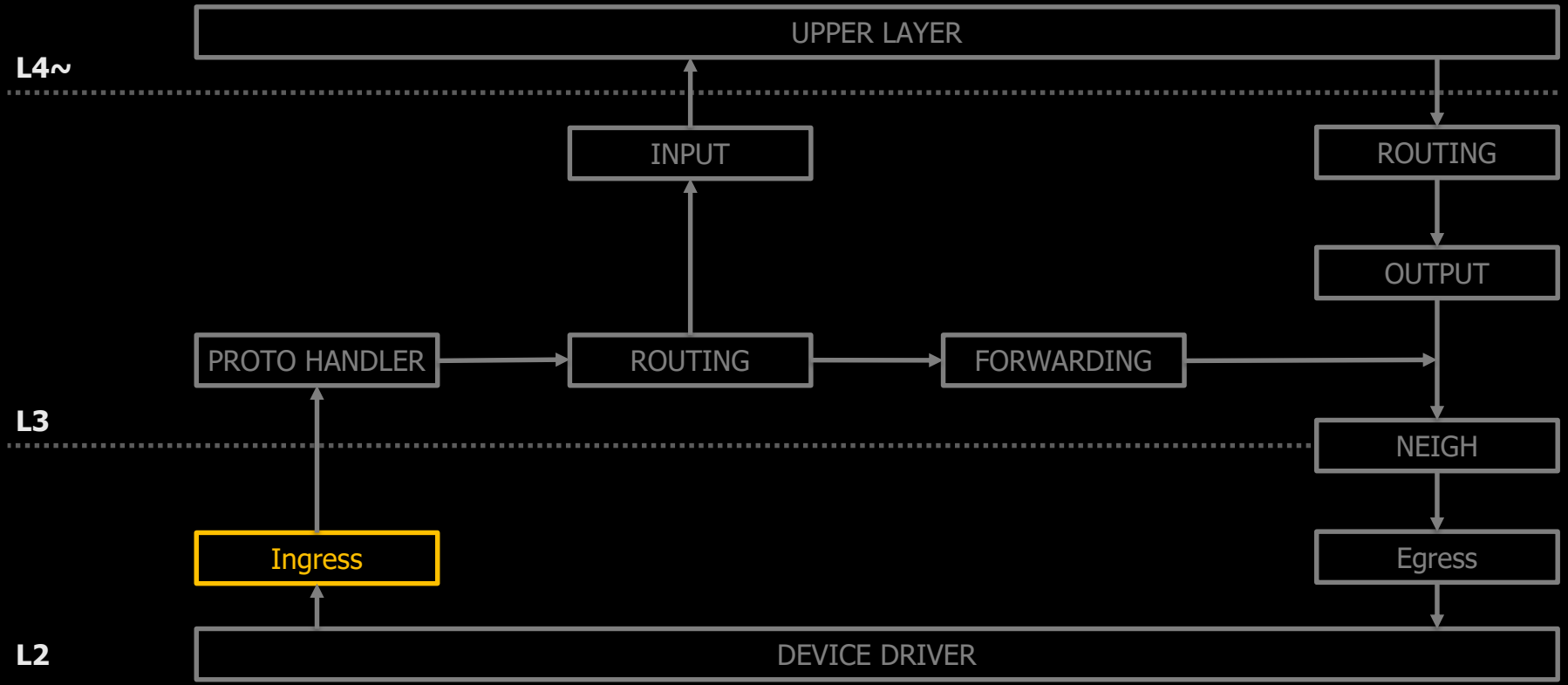
Packet Sending Path



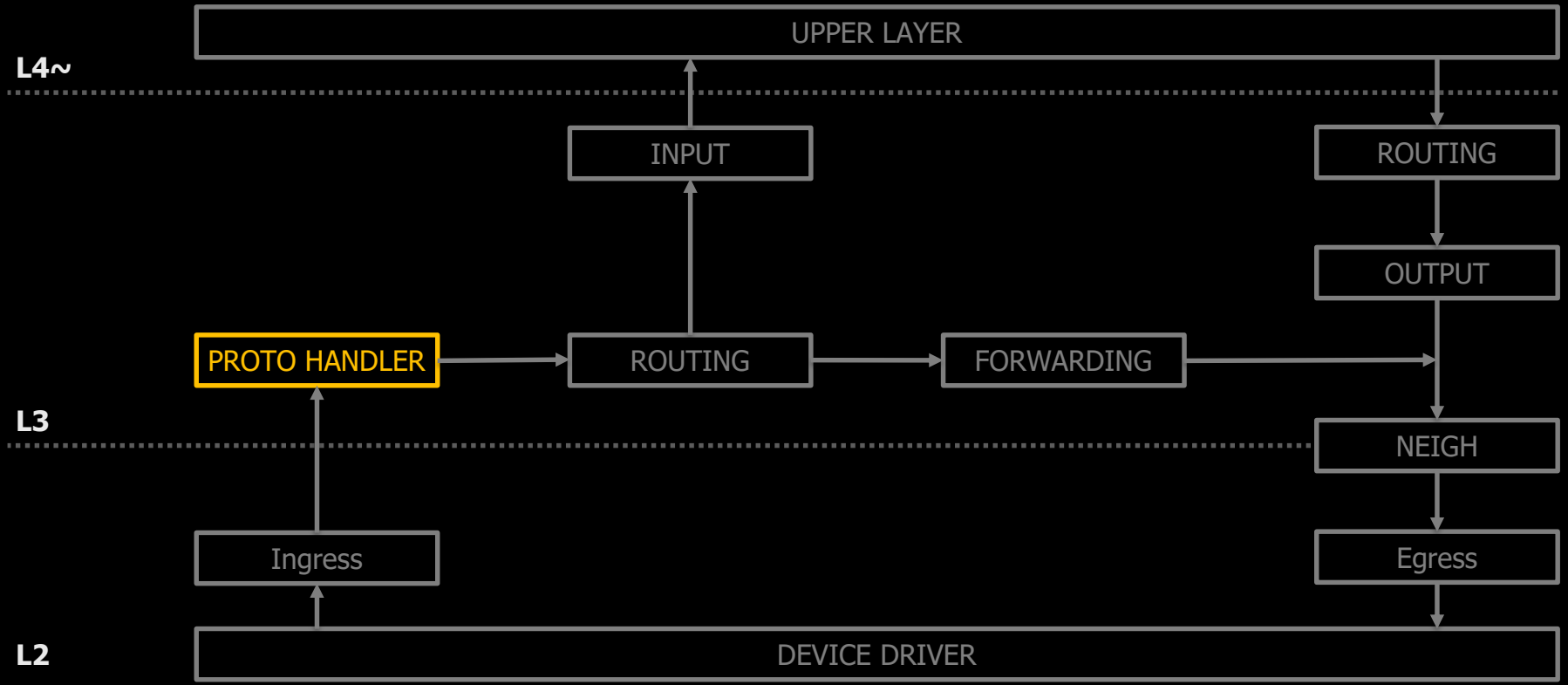
Packet Receiving Path



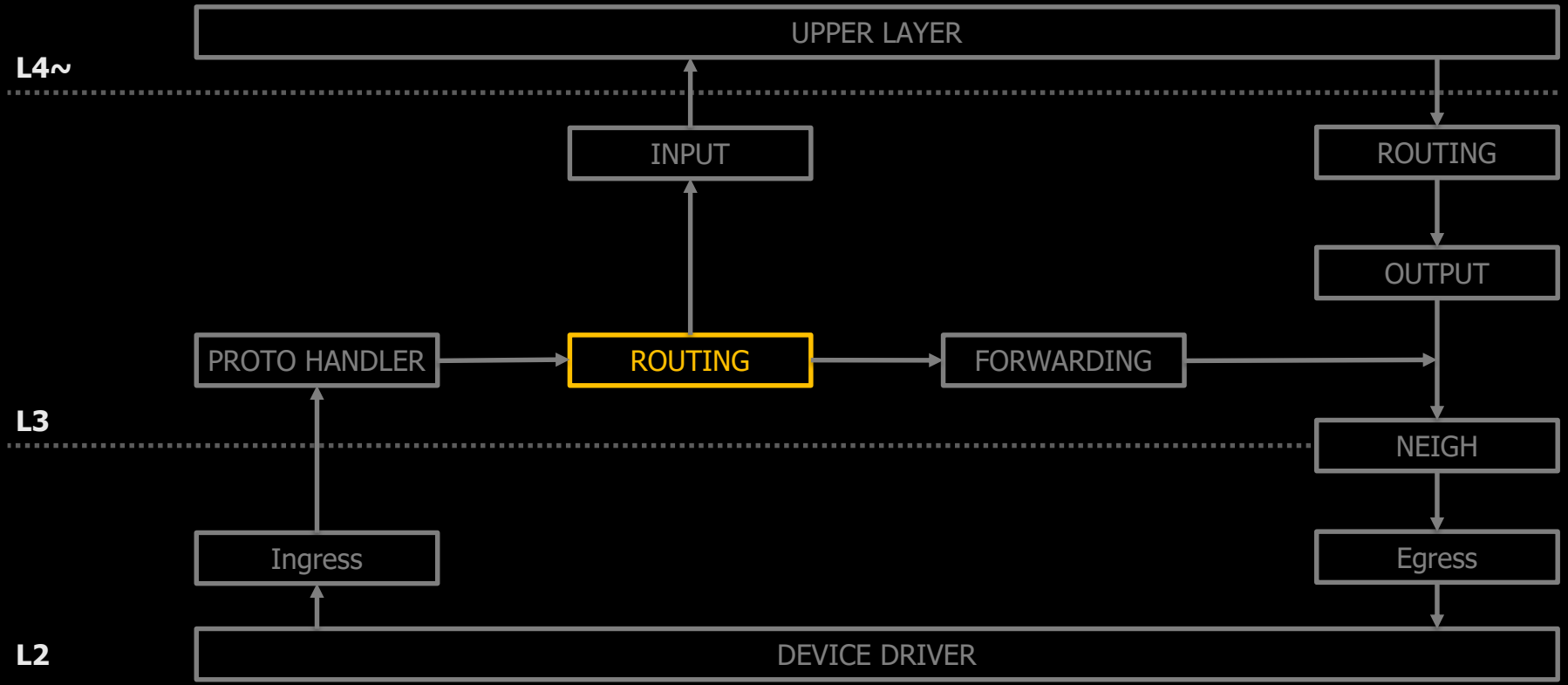
Packet Receiving Path



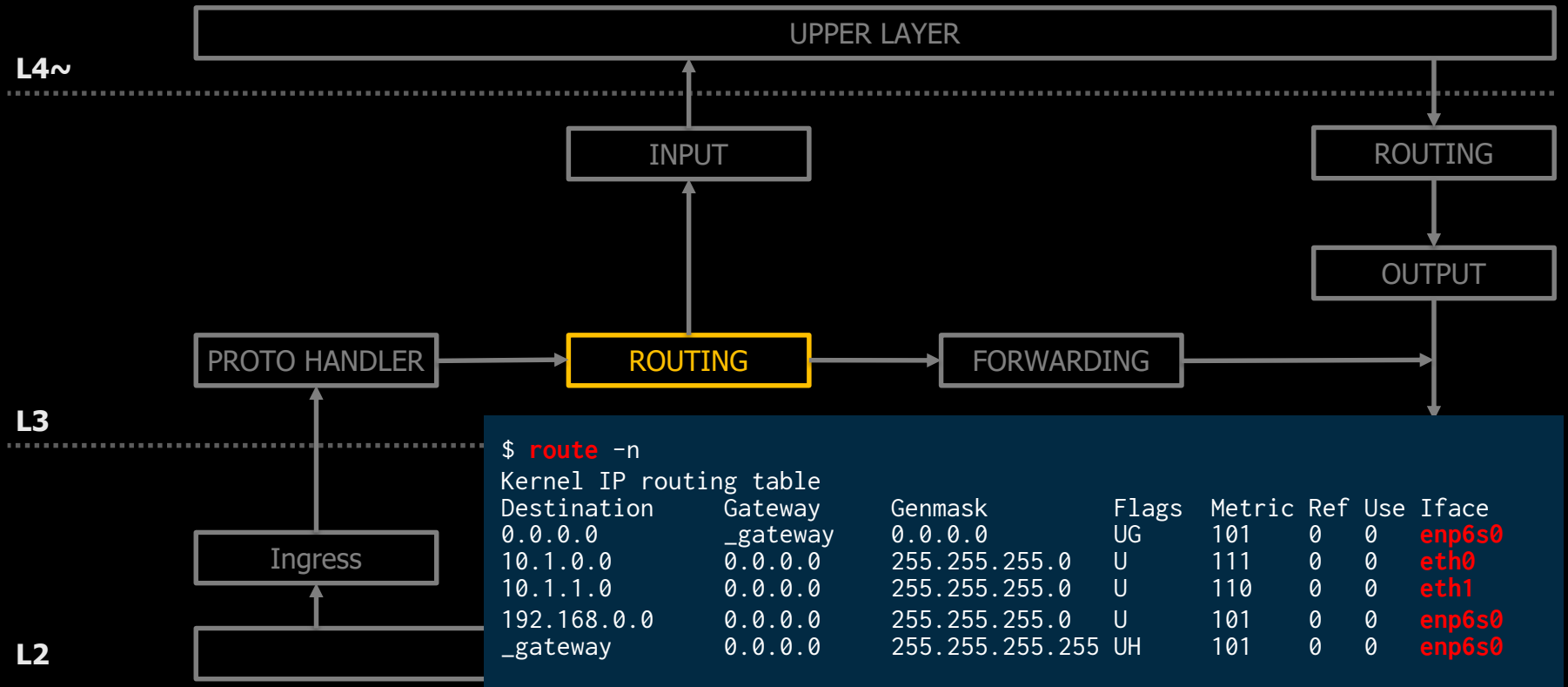
Packet Receiving Path



Packet Receiving Path

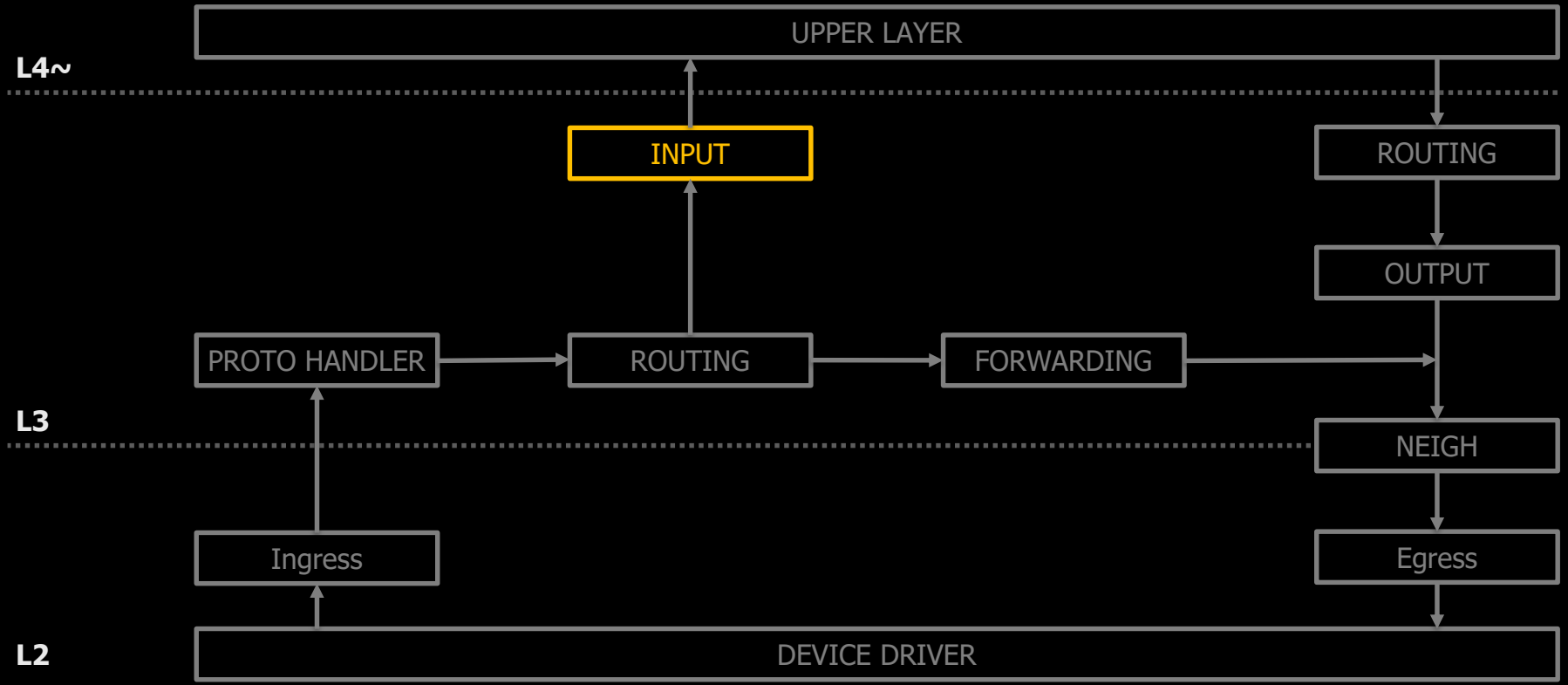


Packet Receiving Path

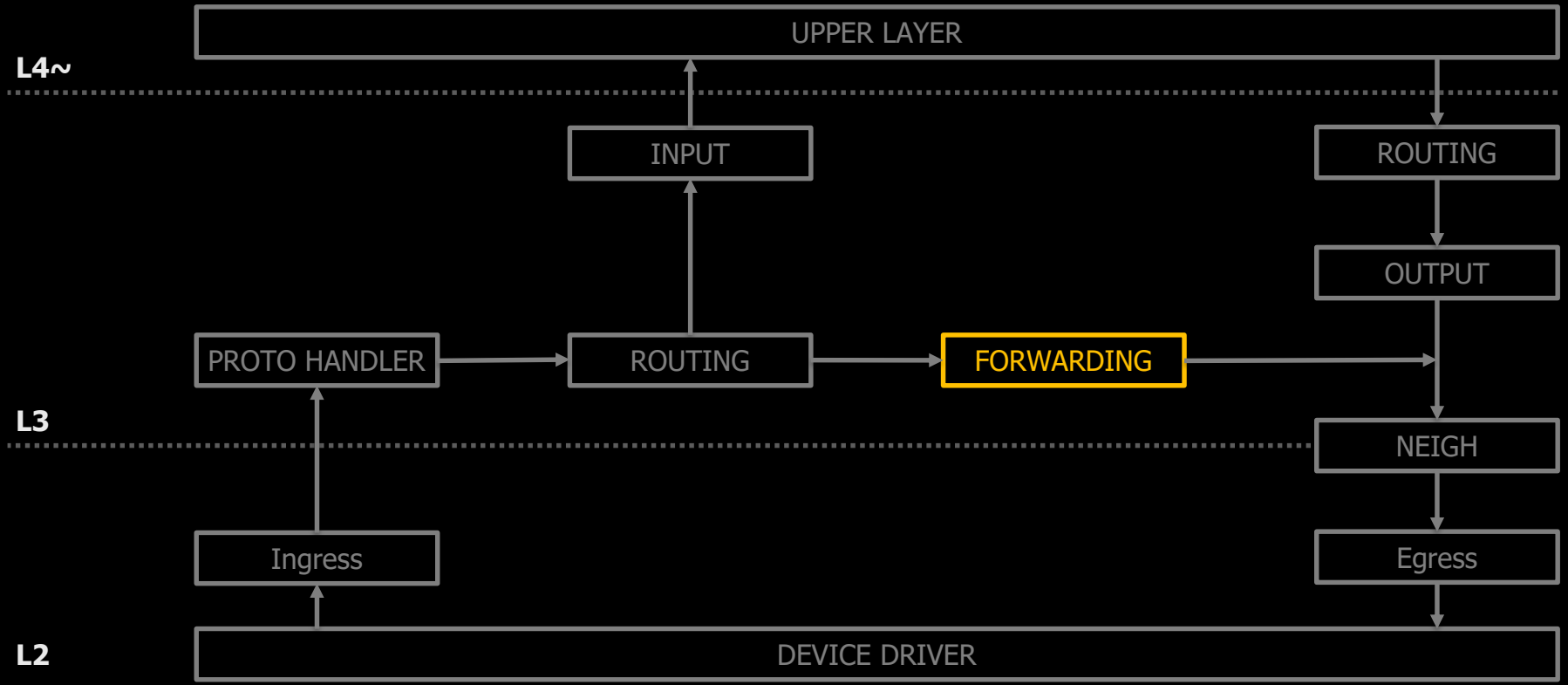



```
$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags  Metric Ref Use Iface
0.0.0.0         _gateway        0.0.0.0         UG     101    0   0   enp6s0
10.1.0.0        0.0.0.0         255.255.255.0   U      111    0   0   eth0
10.1.1.0        0.0.0.0         255.255.255.0   U      110    0   0   eth1
192.168.0.0     0.0.0.0         255.255.255.0   U      101    0   0   enp6s0
_gateway        0.0.0.0         255.255.255.255 UH     101    0   0   enp6s0
```

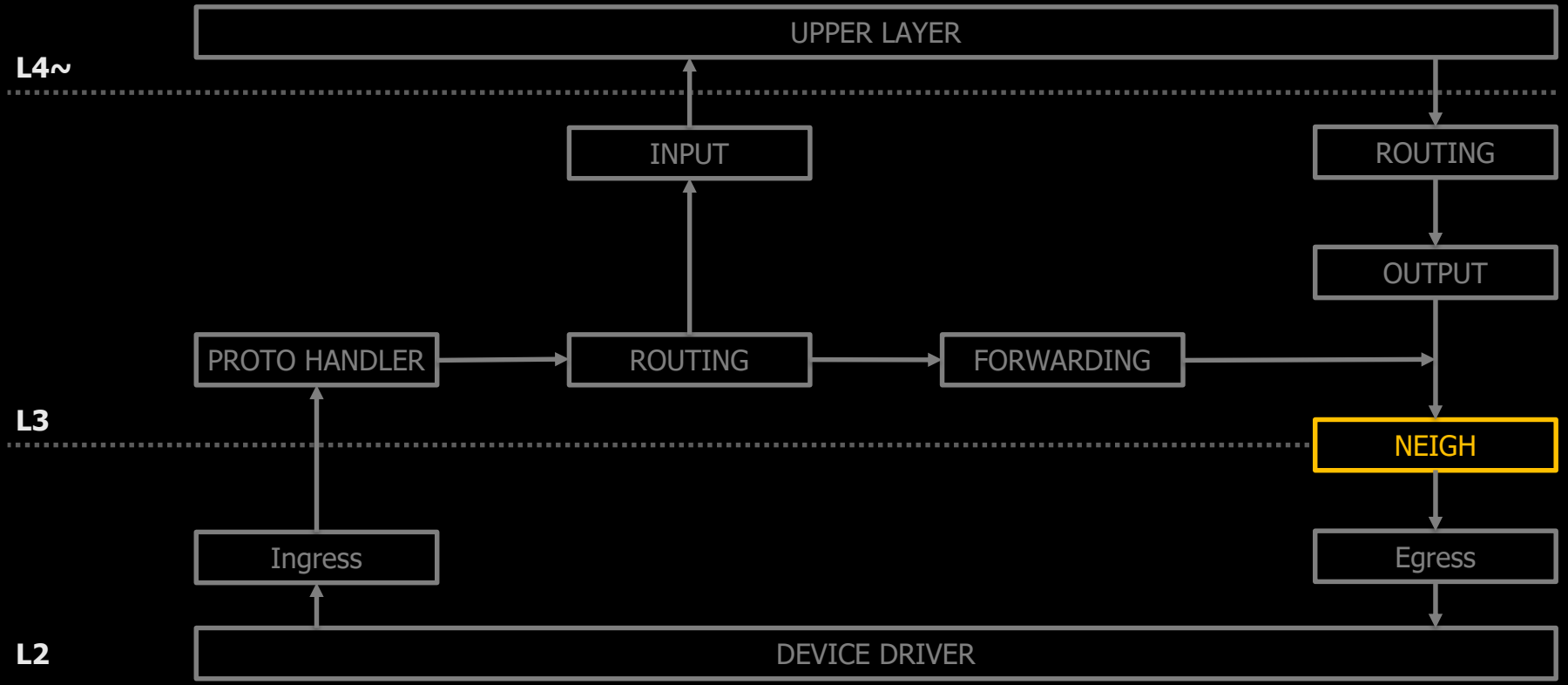
Packet Receiving Path



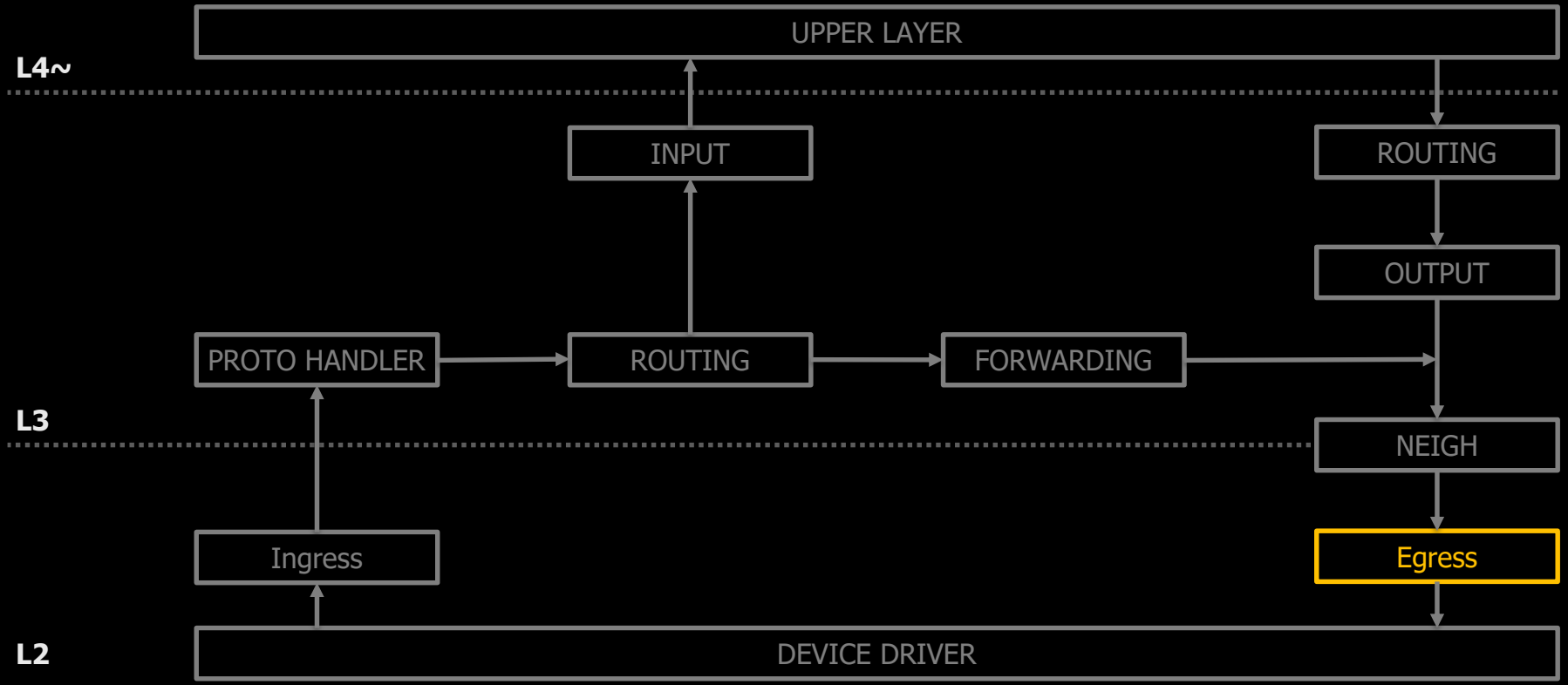
Packet Forwarding Path



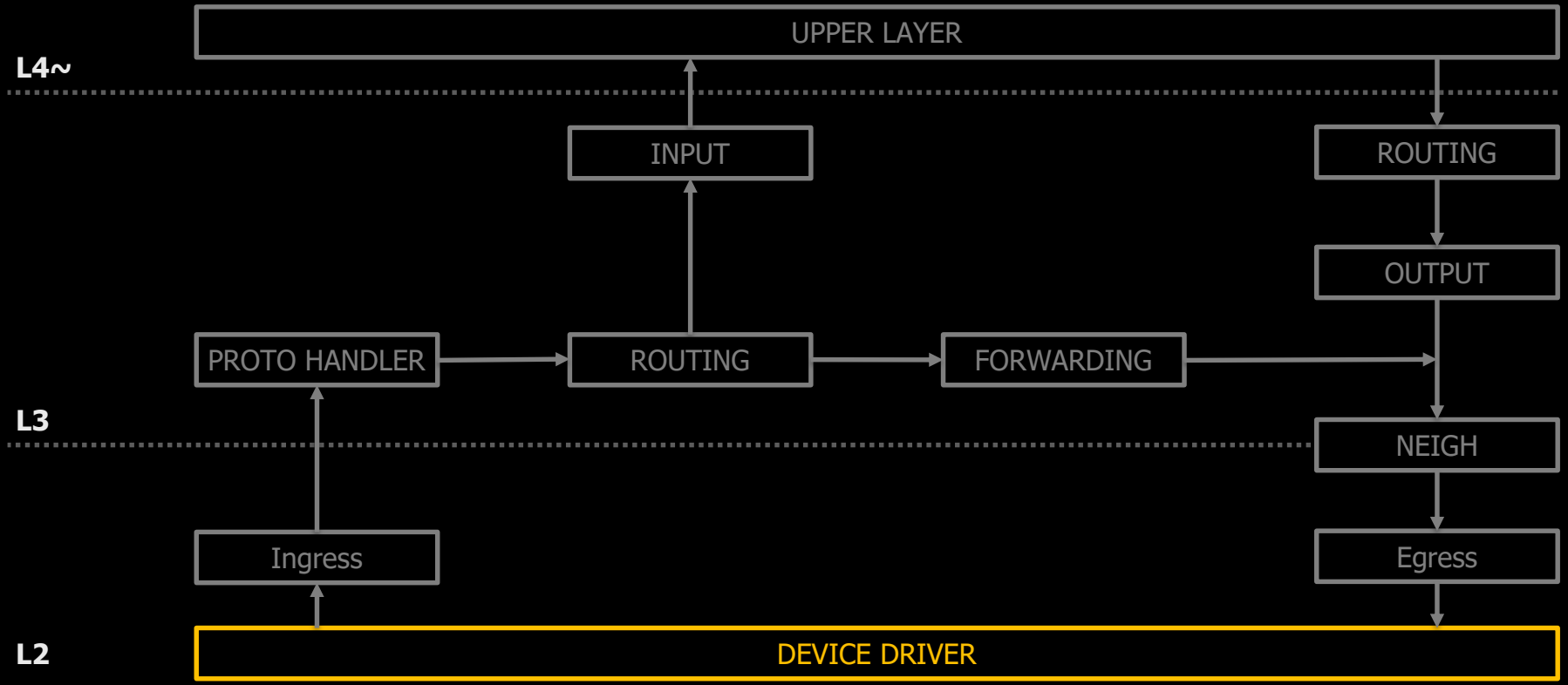
Packet Forwarding Path



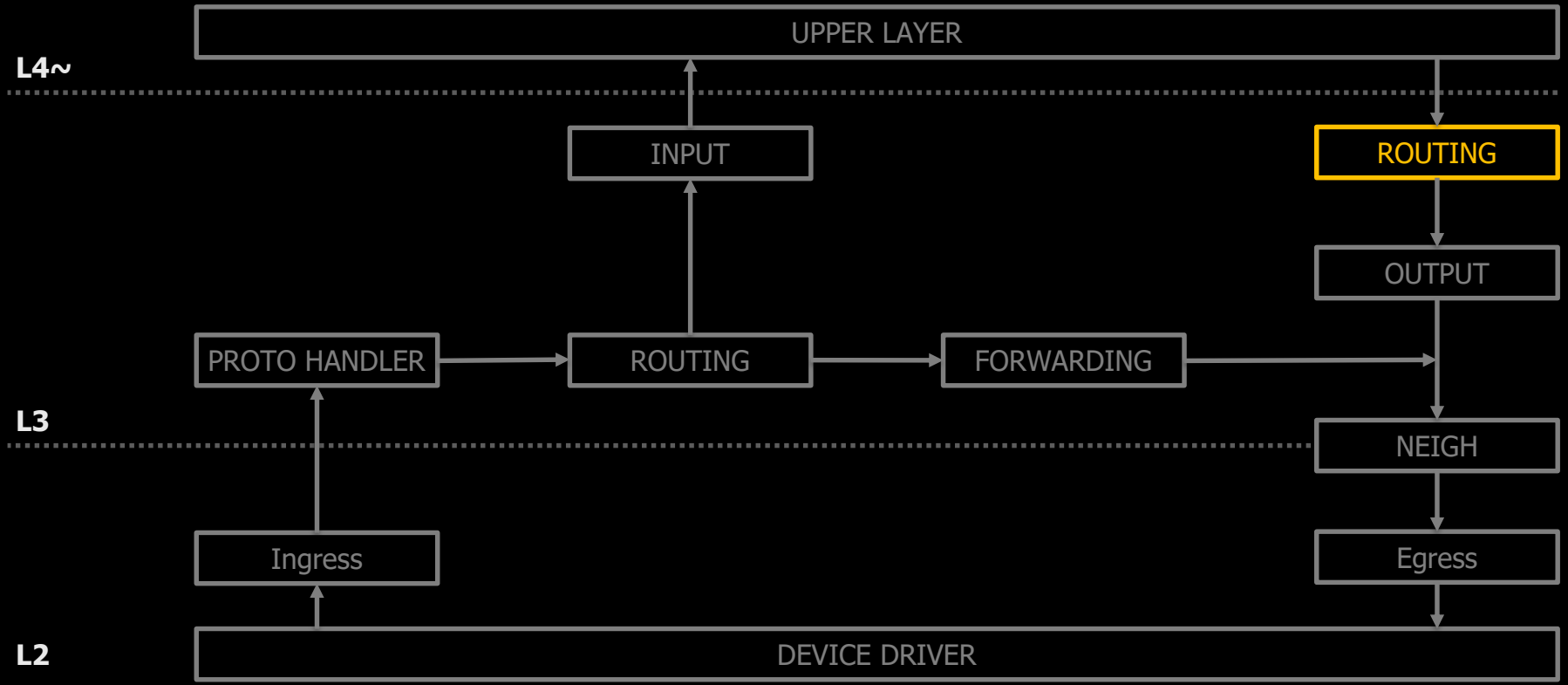
Packet Forwarding Path



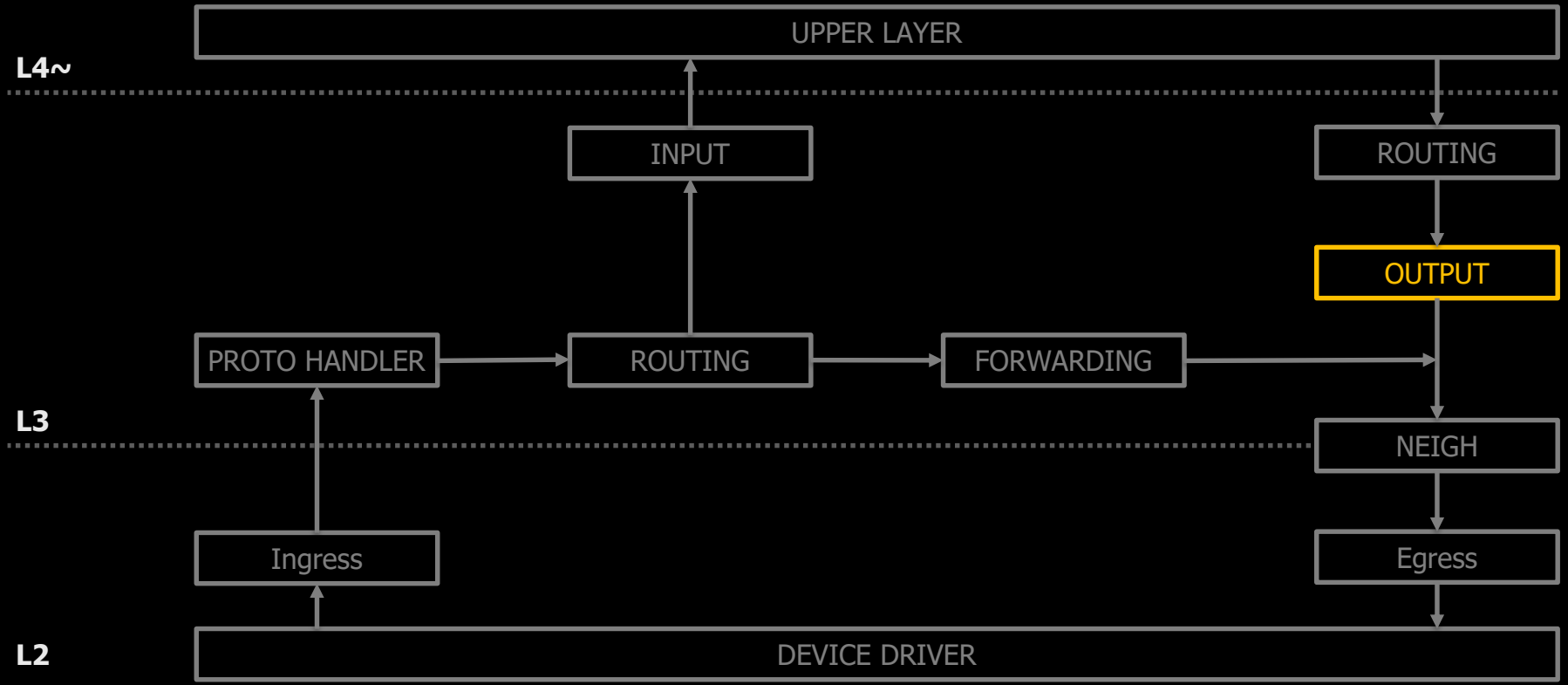
Packet Forwarding Path



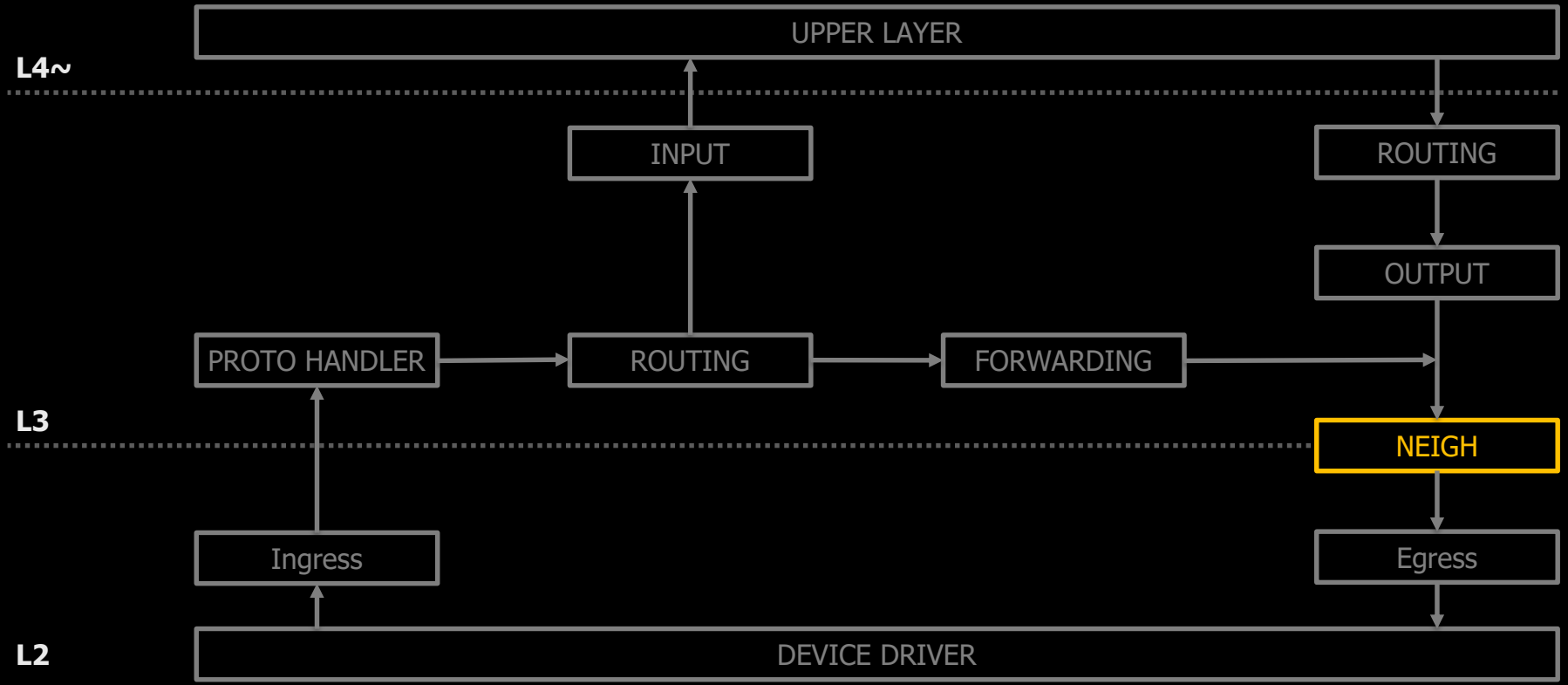
Packet Sending Path



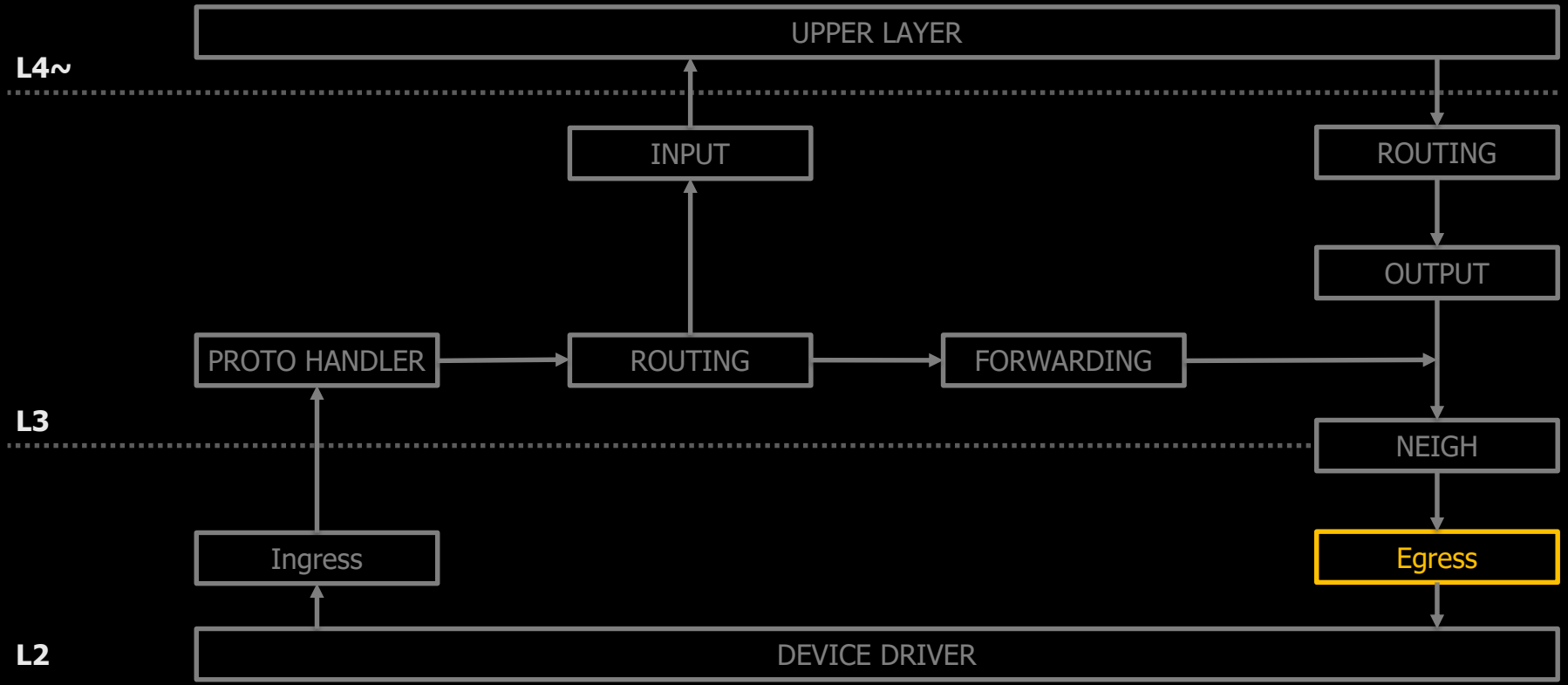
Packet Sending Path



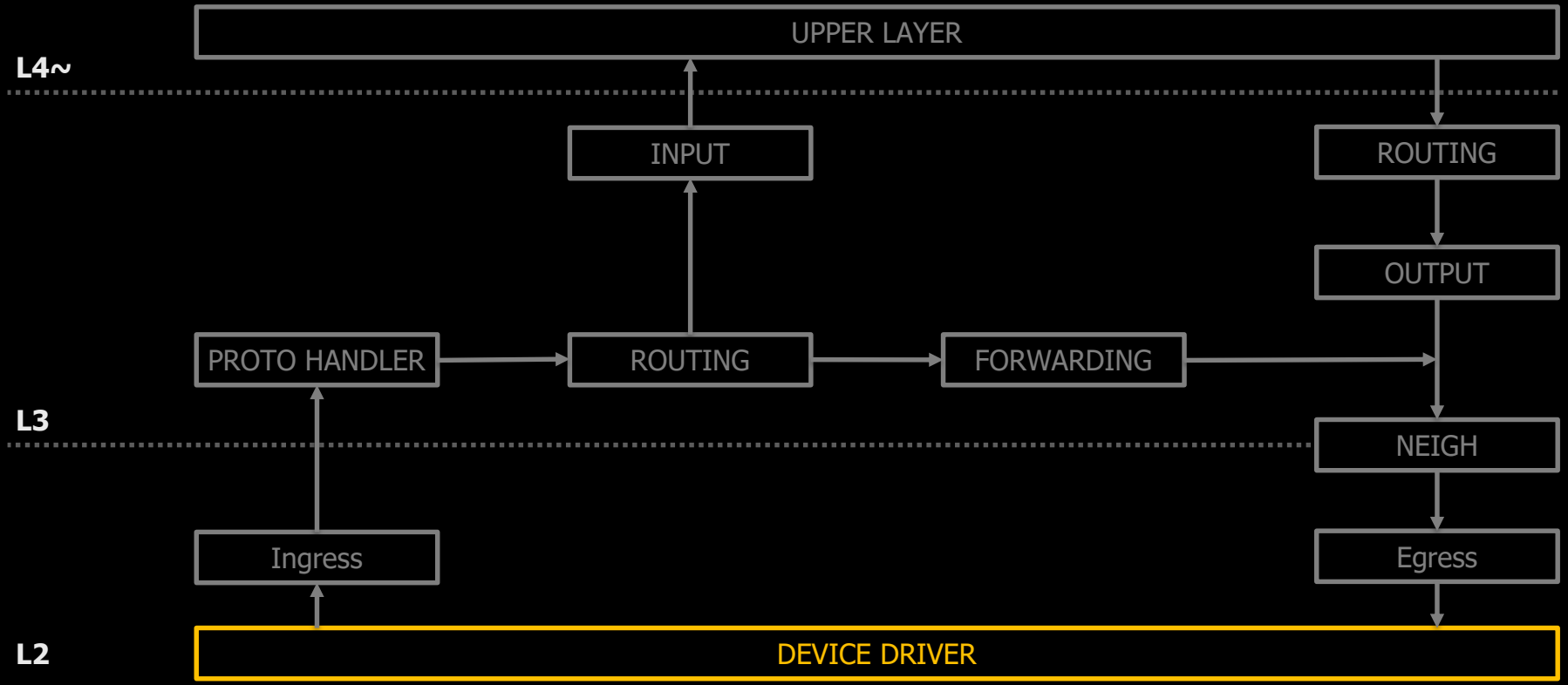
Packet Sending Path



Packet Sending Path



Packet Sending Path



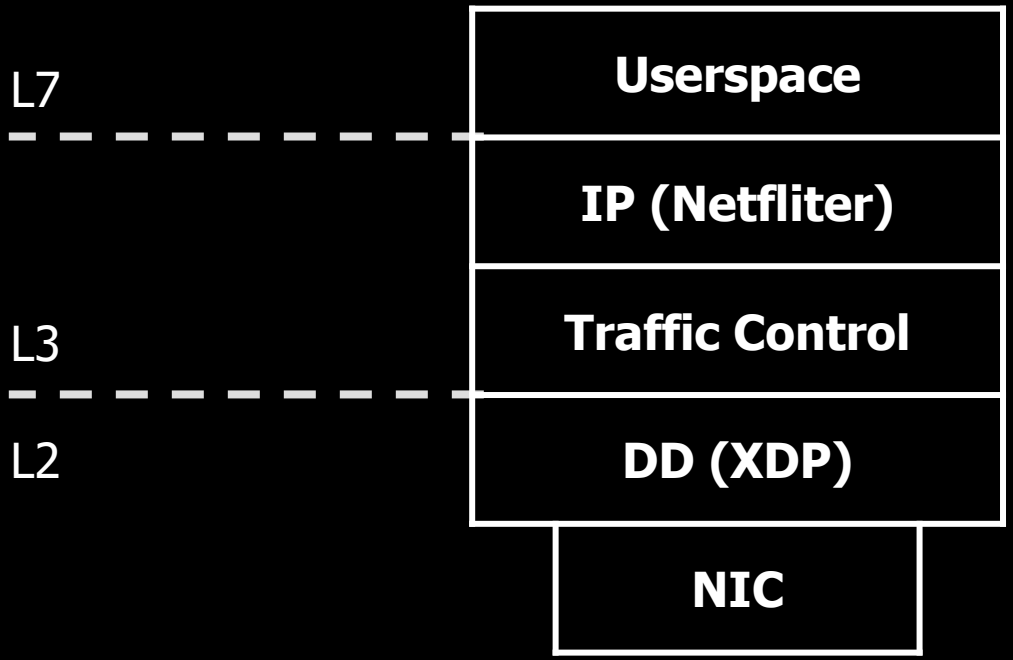
Packet processing?

Packet Processing

- Filtering (Drop, Pass)
- Forwarding (Routing)
- NAT (Masquerade, Source, Destination)
- Packet Tunneling (Encapsulation)
- Packet Mangling (ToS, TTL, mark)

Hooks to process packet?

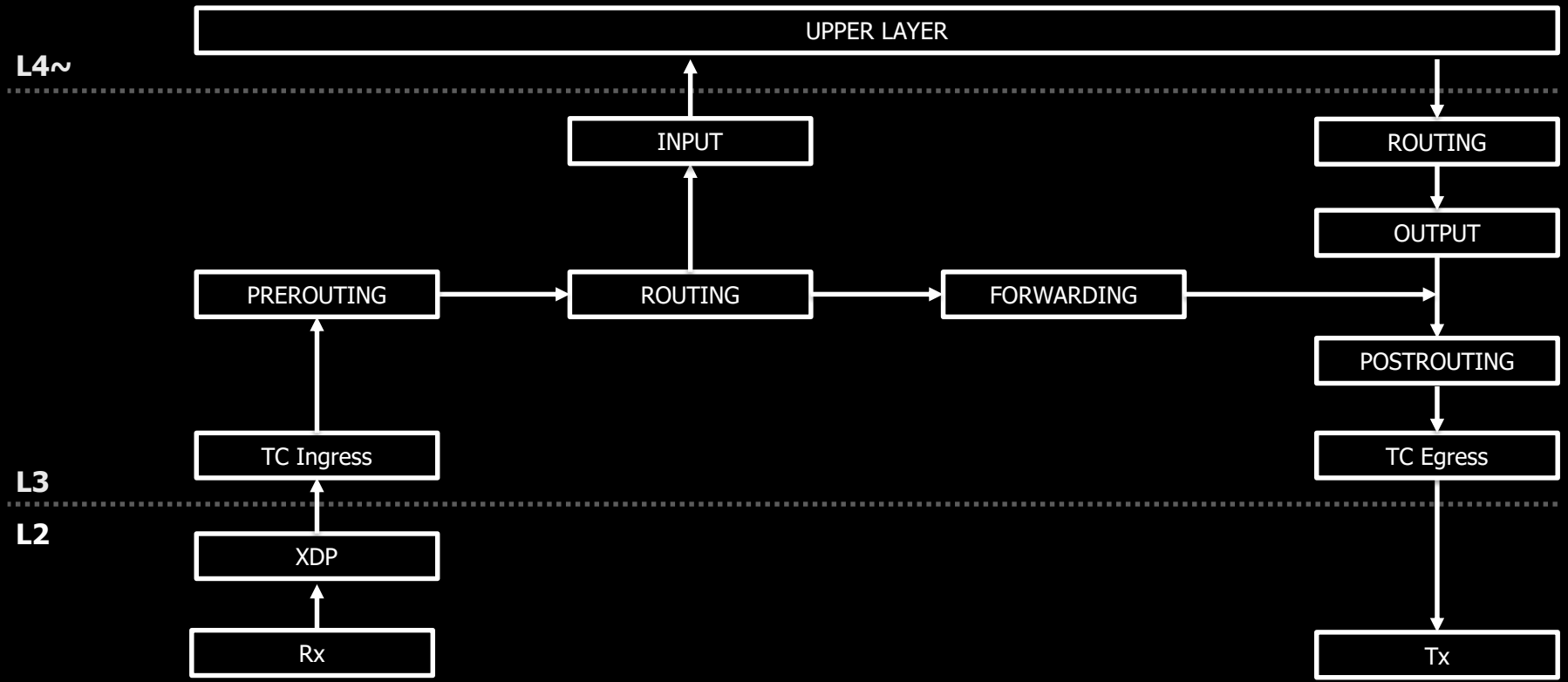
Hooks to process packet?



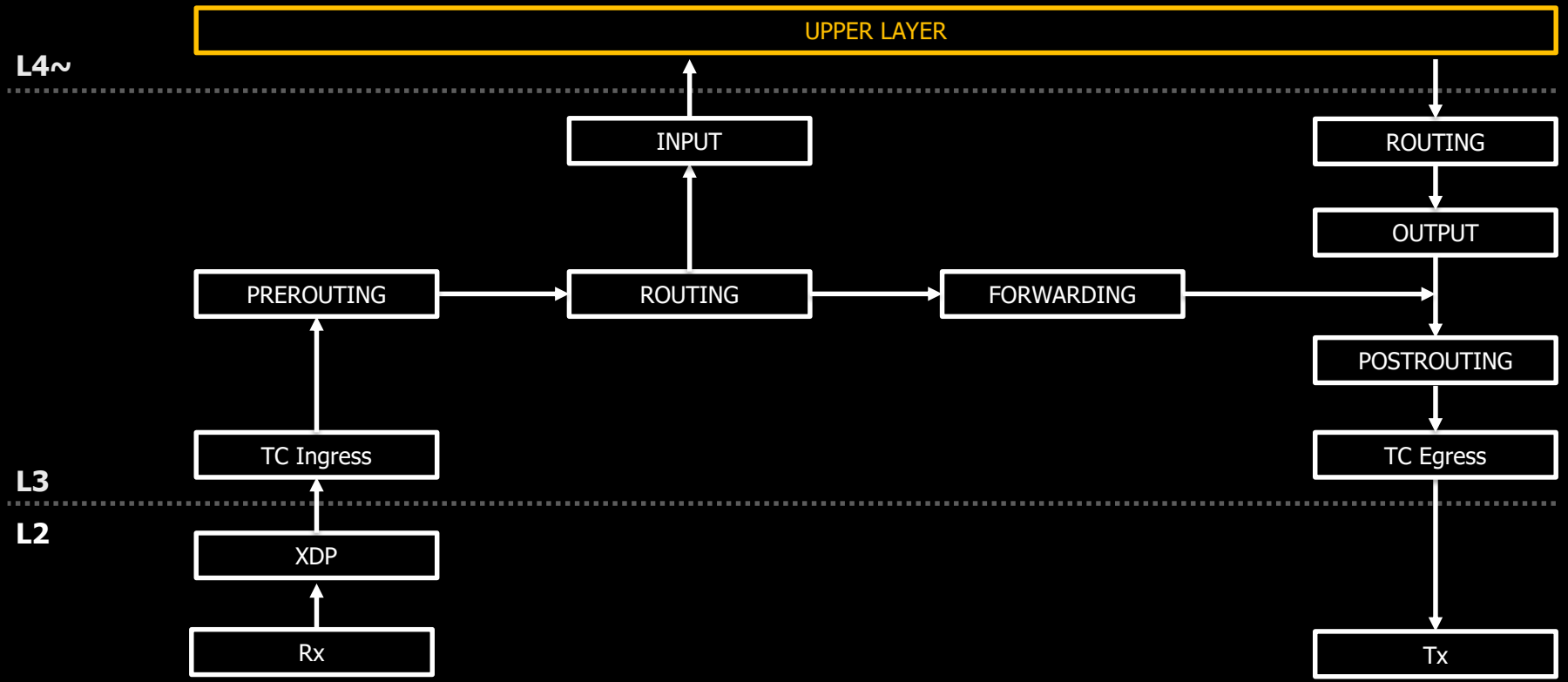
Linux Socket

Packet processing by receive message

Packet-Processing Path



Userspace Packet Path



Userspace

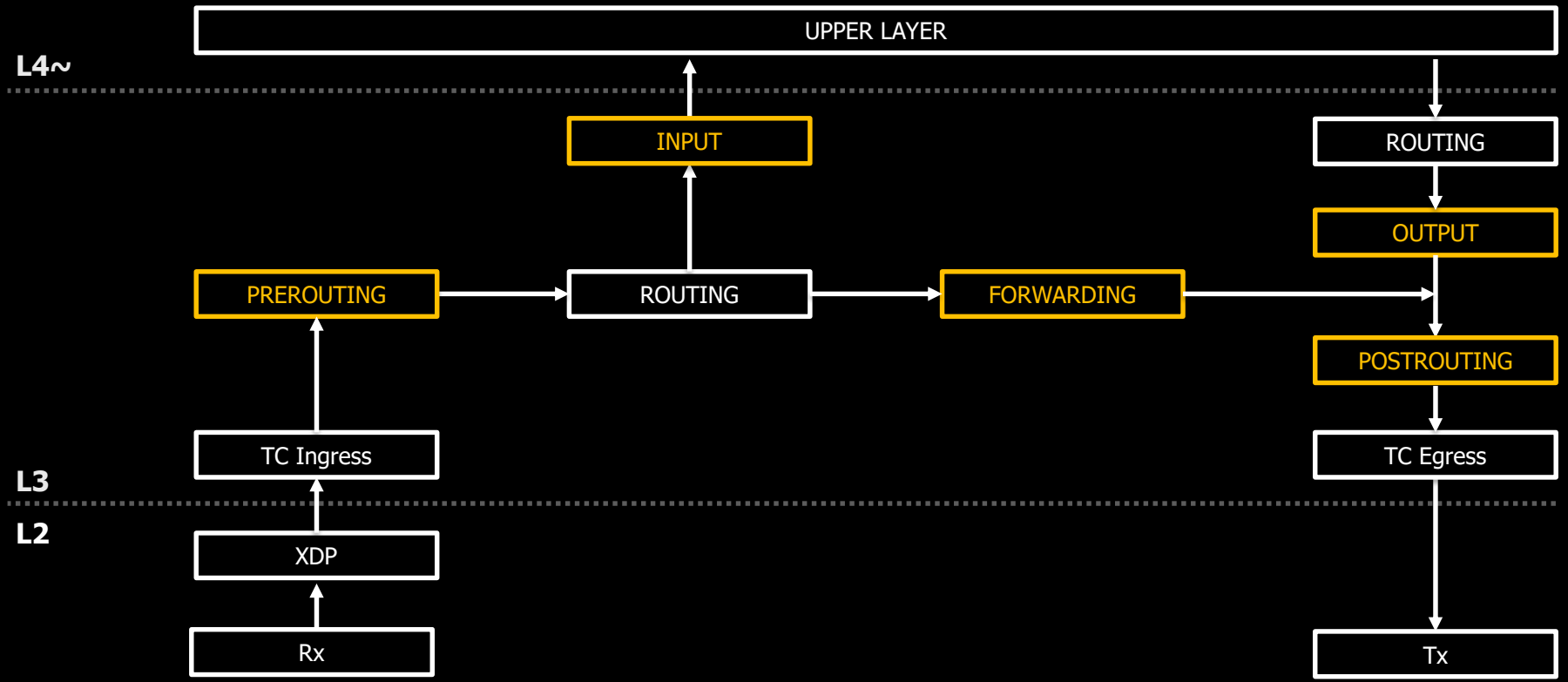
```
int fd = socket(AF_INET, SOCK_DGRAM, 0);
struct sockaddr_in in;
int res, pkts, bytes;
char buf[MTU_SIZE];

memset(&in, 0, sizeof(in));
in.sin_family = AF_INET;
in.sin_addr.s_addr = INADDR_ANY;
in.sin_port = htons(1234);

if (bind(fd, (struct sockaddr*)&in, sizeof(in)) < 0)
    exit(EXIT_FAILURE);

while (1) {
    int res = read(fd, buf, MTU_SIZE);

    if (res <= 0)
        return 0;

    pkts += 1;
    bytes += r;
}
```

Any kind of packet processing is possible!

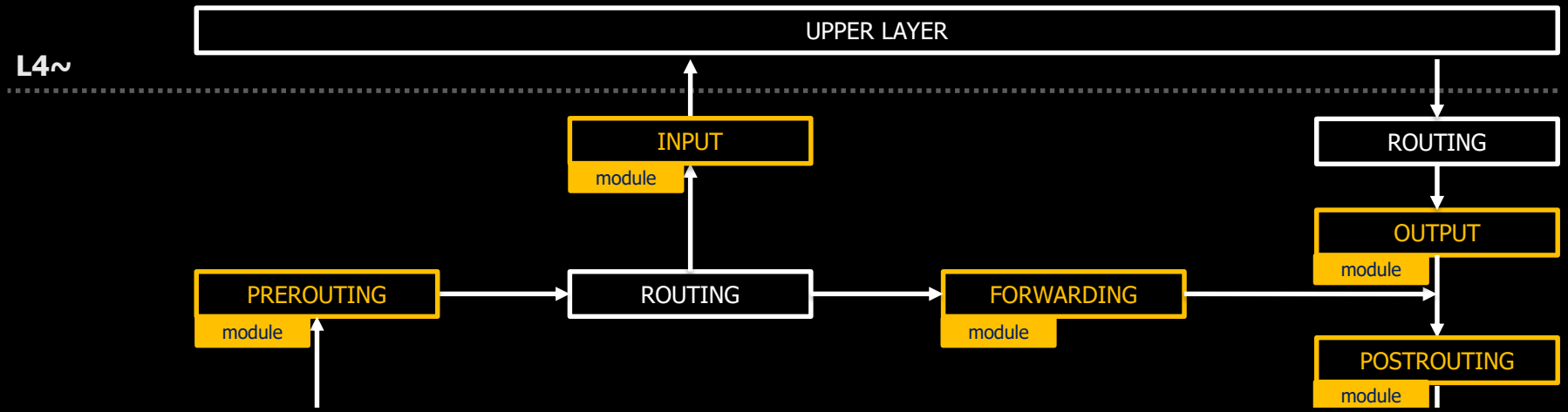
Linux Firewall Framework

set of hooks inside the Network stack

Netfilter Packet Path



Netfilter Packet Path



By registering kernel modules to hooks,
Intercepts network traffic

ex) iptables...

Netfilter

By using iptables, nftables, etc..

Packet Processing is available

- Packet filtering
- NAT (network address translation)
- Packet Mangling.
- Stateless/Stateful Firewalling
- ETC..

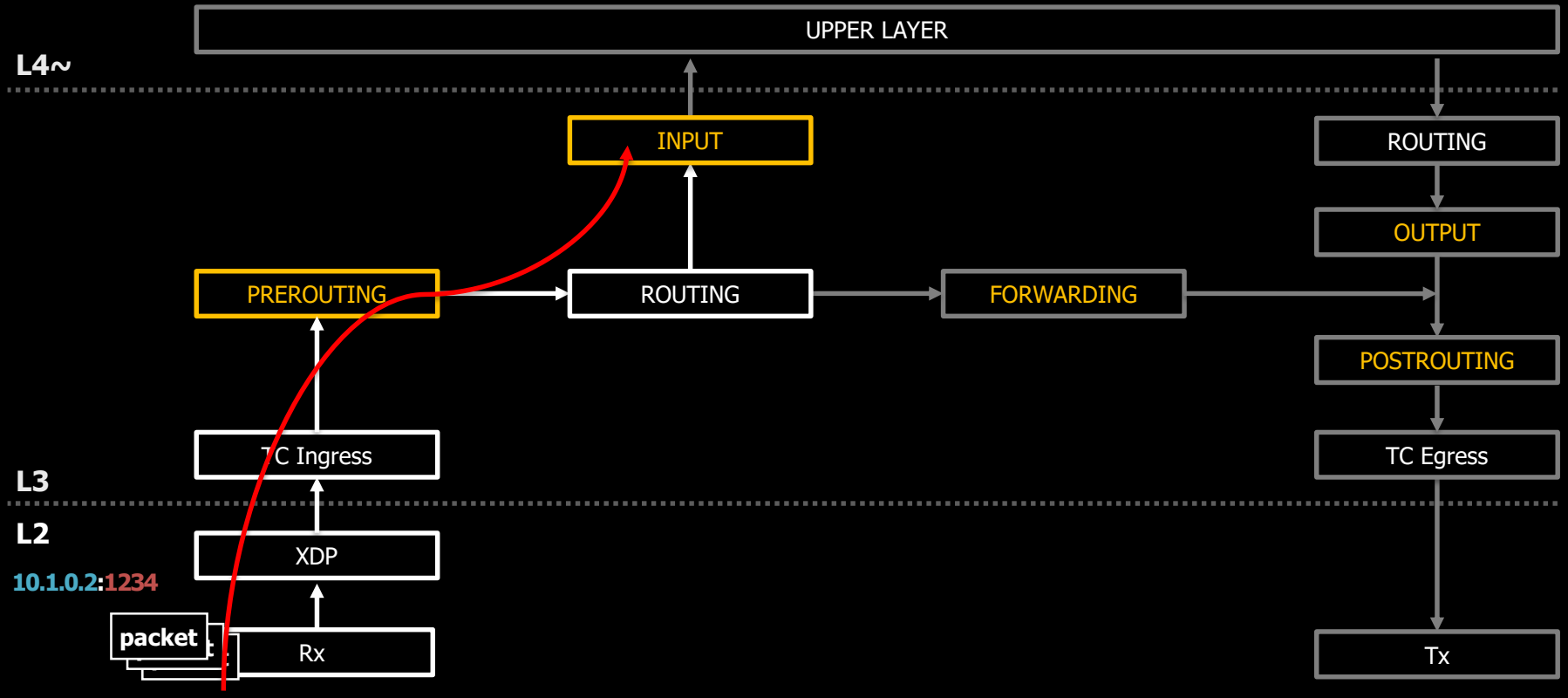
Netfilter Example 1 – Packet Filtering

Netfilter (iptables) packet drop

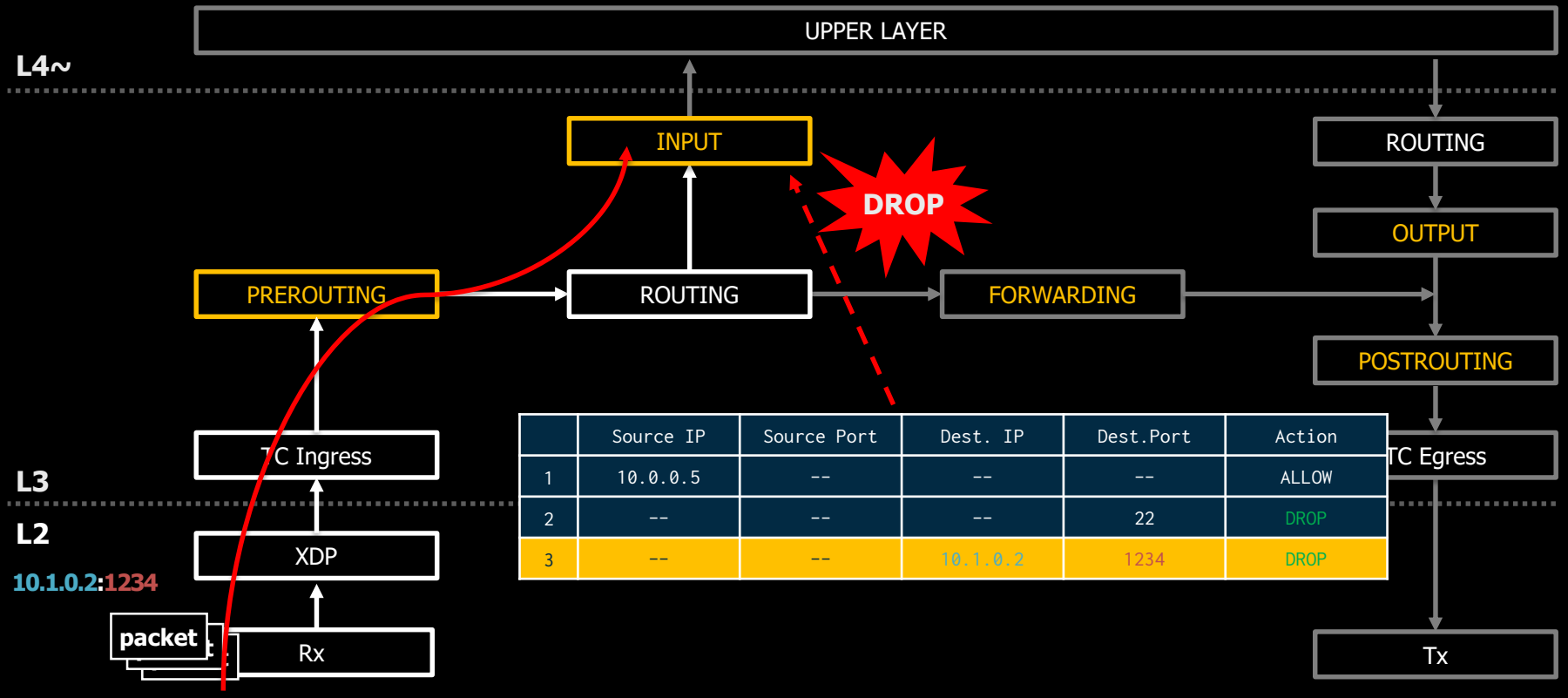
DROP UDP 10.1.0.2:1234

```
$ iptables -A INPUT -d 10.1.0.2 -p udp --dport 1234 -j DROP
```

Netfilter Example 1 – Packet Filtering



Netfilter Example 1 – Packet Filtering



	Source IP	Source Port	Dest. IP	Dest.Port	Action
1	10.0.0.5	--	--	--	ALLOW
2	--	--	--	22	DROP
3	--	--	10.1.0.2	1234	DROP

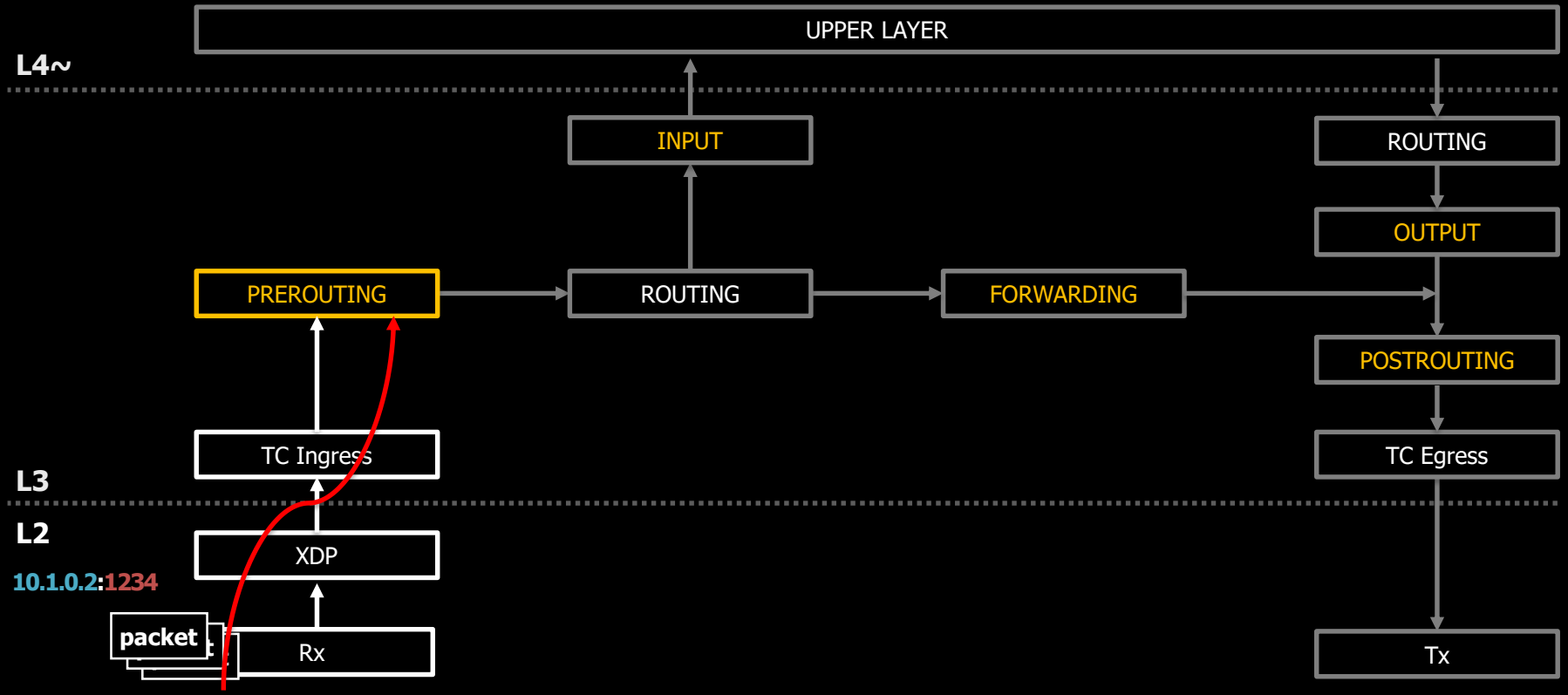
Netfilter Example 2 – NAT

Netfilter (iptables) Destination NAT

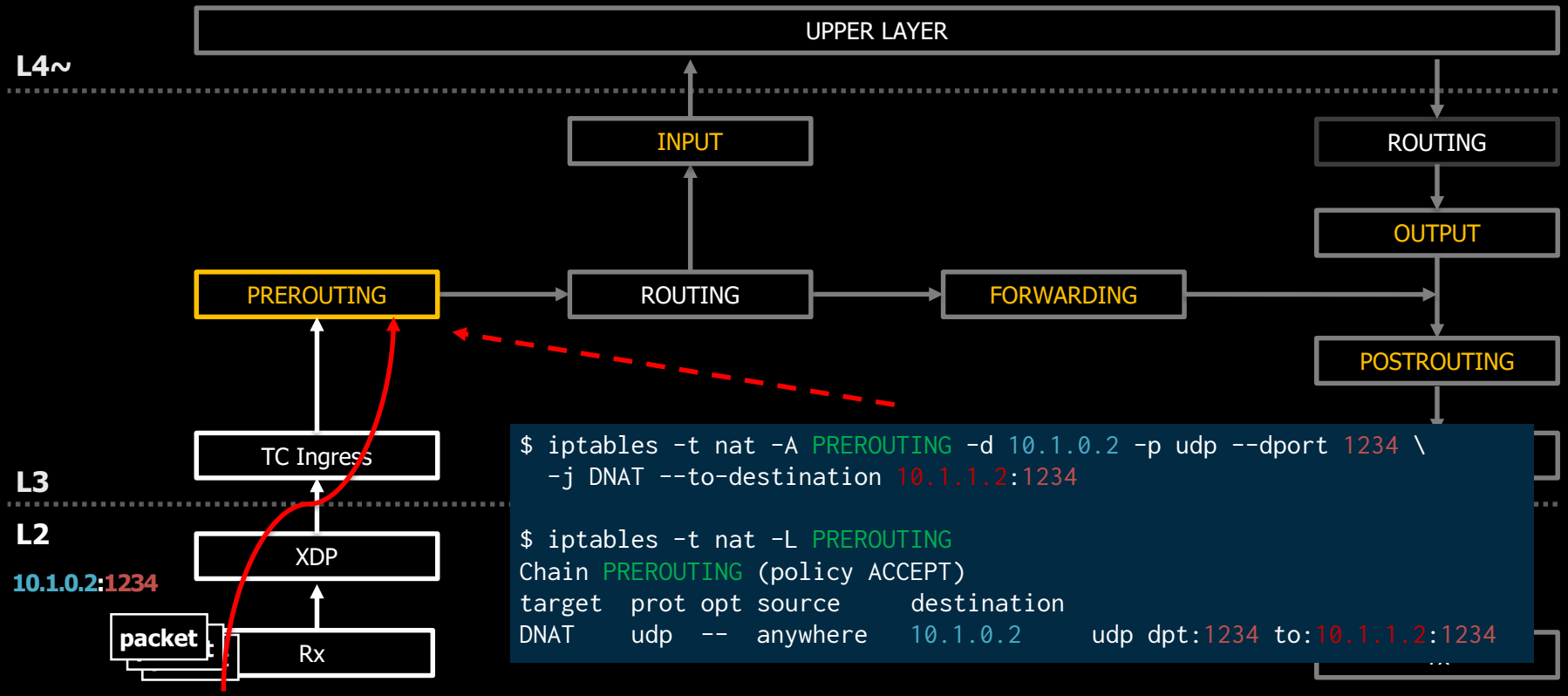
NAT UDP 10.1.0.2:1234 -> 10.1.1.2:1234

```
$ iptables -t nat -A PREROUTING \
  -d 10.1.0.2 -p udp --dport 1234 \
  -j DNAT --to-destination 10.1.1.2:1234
```

Netfilter Example 2 – NAT

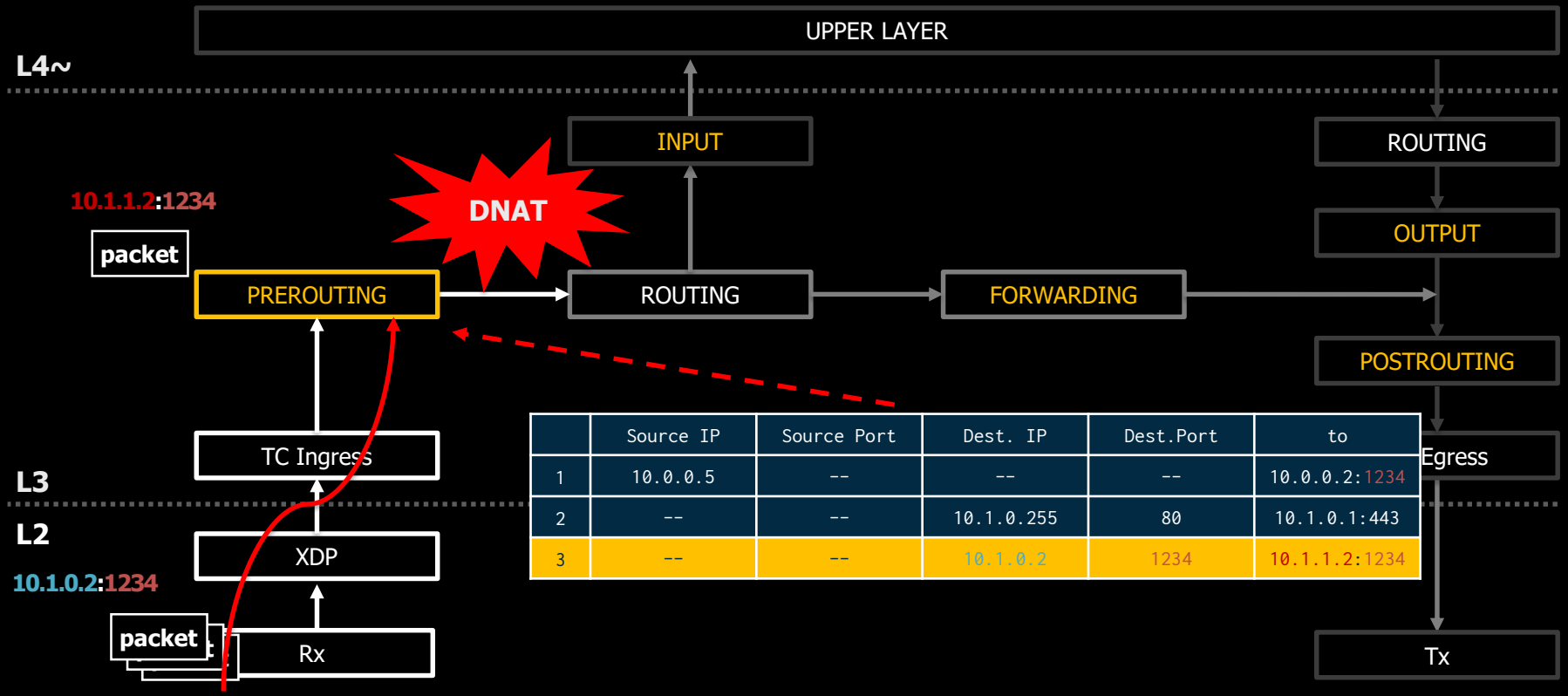


Netfilter Example 2 – NAT



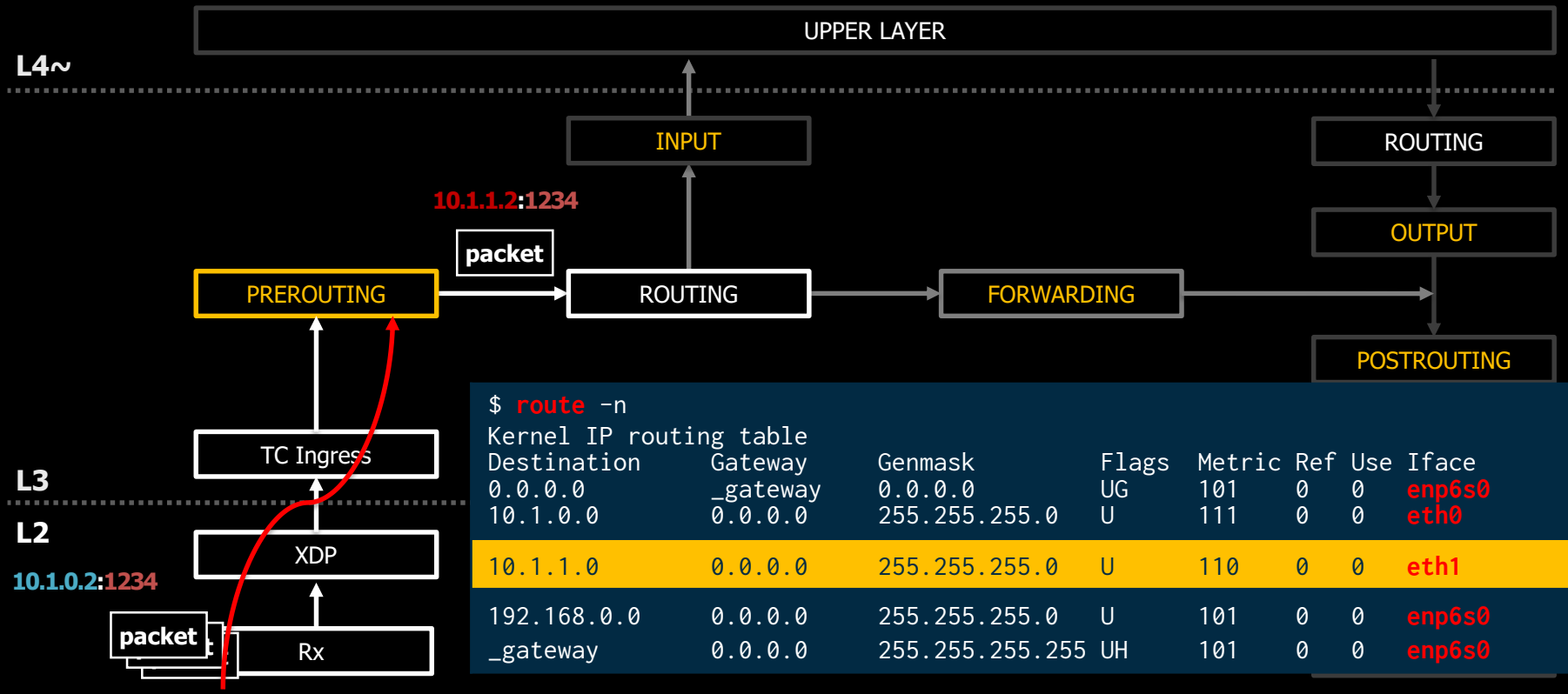

```
$ iptables -t nat -A PREROUTING -d 10.1.0.2 -p udp --dport 1234 \
  -j DNAT --to-destination 10.1.1.2:1234

$ iptables -t nat -L PREROUTING
Chain PREROUTING (policy ACCEPT)
target  prot opt source     destination
DNAT    udp  --  anywhere   10.1.0.2     udp dpt:1234 to:10.1.1.2:1234
```

Netfilter Example 2 – NAT



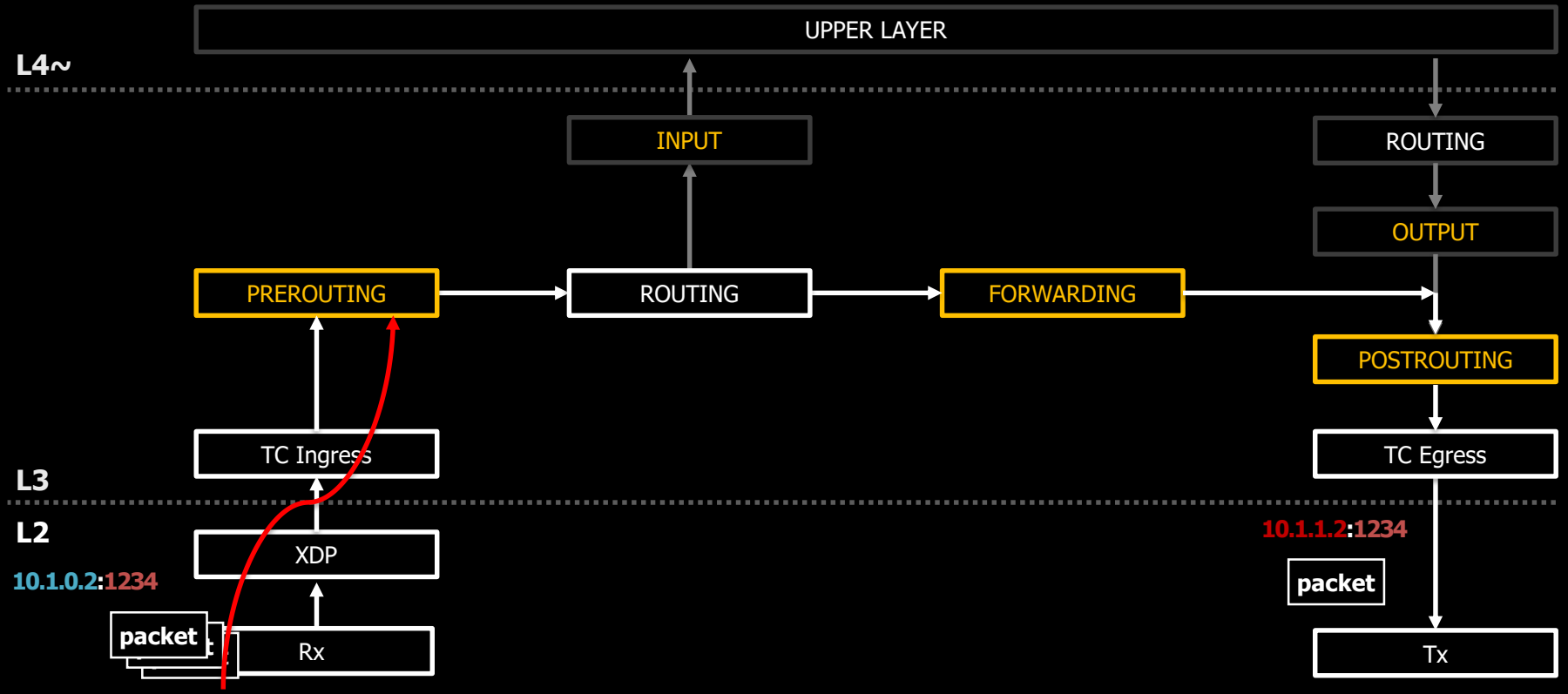
	Source IP	Source Port	Dest. IP	Dest.Port	to
1	10.0.0.5	--	--	--	10.0.0.2:1234
2	--	--	10.1.0.255	80	10.1.0.1:443
3	--	--	10.1.0.2	1234	10.1.1.2:1234

Netfilter Example 2 – NAT




```
$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         _gateway        0.0.0.0         UG    101    0        0 enp6s0
10.1.0.0        0.0.0.0         255.255.255.0   U     111    0        0 eth0
10.1.1.0        0.0.0.0         255.255.255.0   U     110    0        0 eth1
192.168.0.0     0.0.0.0         255.255.255.0   U     101    0        0 enp6s0
_gateway        0.0.0.0         255.255.255.255 UH    101    0        0 enp6s0
```

Netfilter Example 2 – NAT

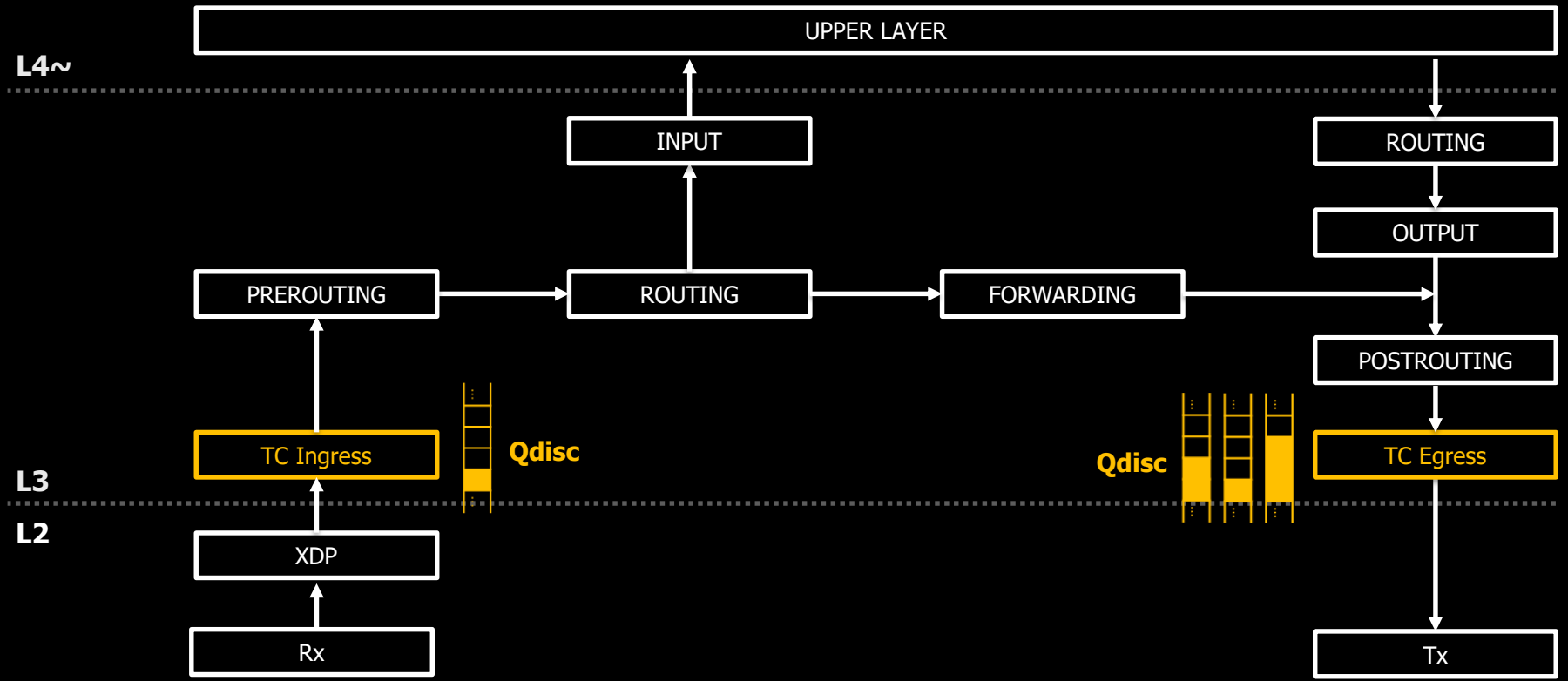


TC

Traffic Control – Packet Scheduler

Control network traffic with Queuing policy

TC Packet Path



TC Packet Path

With Packet Scheduler (Qdisc),
can determine how to receive & transmit packets

ex) priority, delay, dropping

TC Packet Path

By attaching Filter to Qdisc,
User can take action with matched packet

With filters attached to Qdisc

Packet Processing is available

- Packet filtering
- NAT (network address translation)
- Packet Mangling (skb_edit)
- Packet Forwarding (mirroring)
- QoS
- ETC..

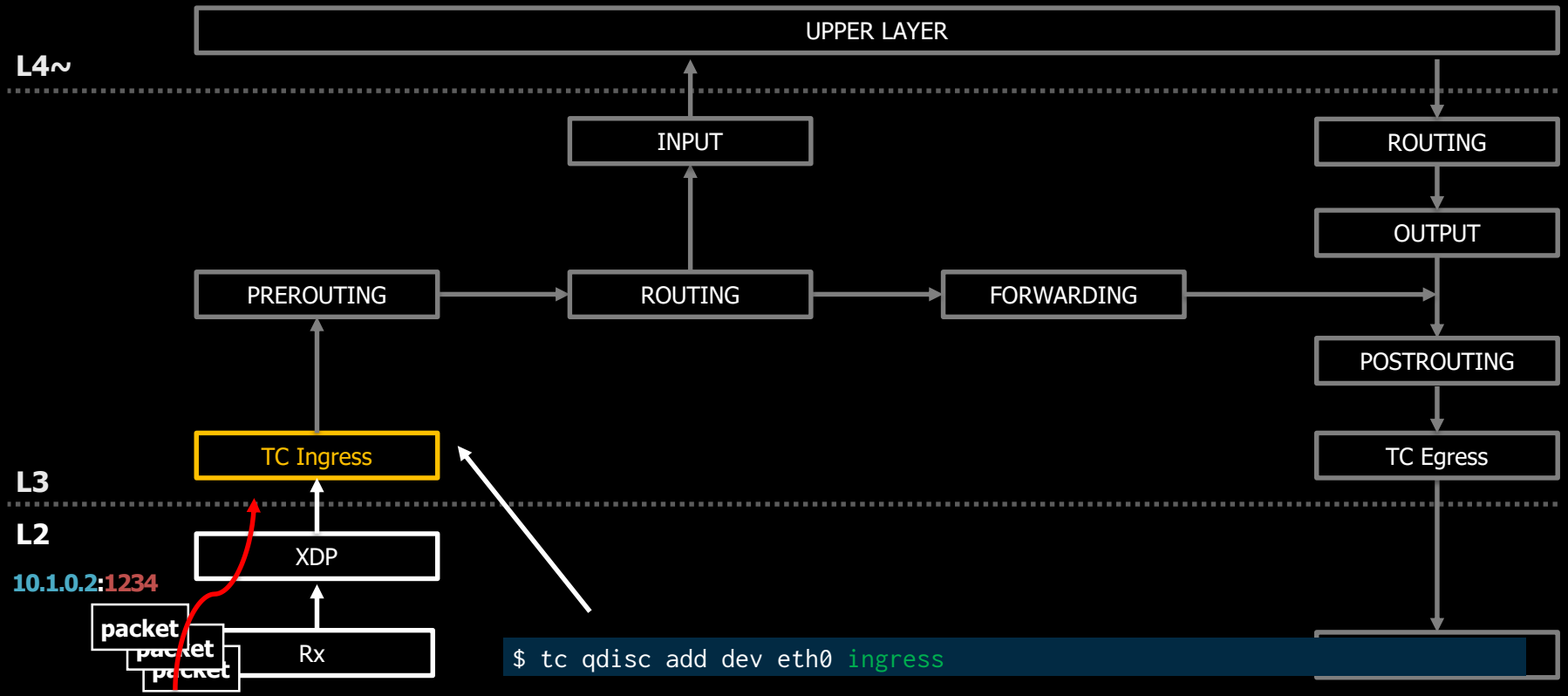
TC Example 1 – Packet Filtering

TC Ingress packet drop

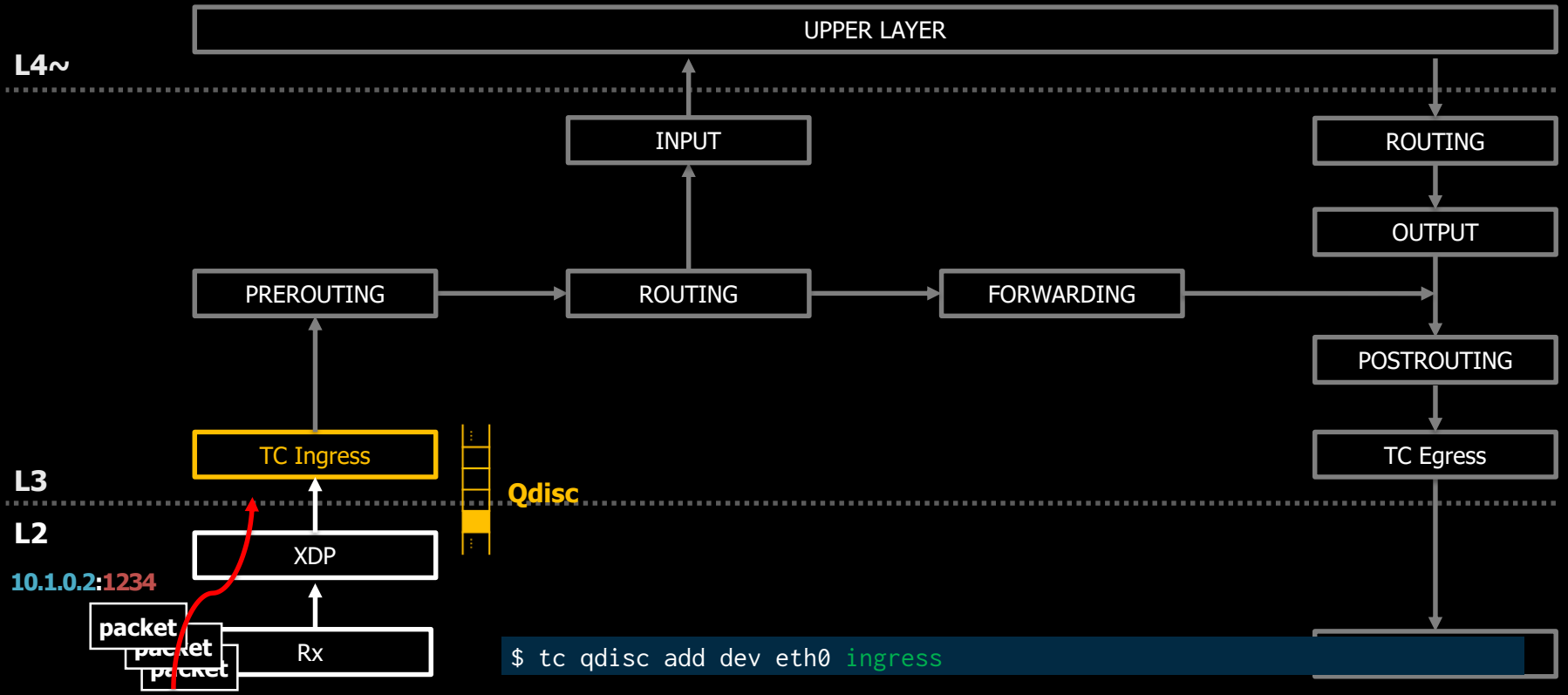
DROP UDP 10.1.0.2:1234

```
$ tc qdisc add dev eth0 ingress
$ tc filter add dev eth0 parent ffff: protocol ip u32 \
    match ip dst 10.1.0.2 match ip dport 1234 0xffff \
    action drop
```

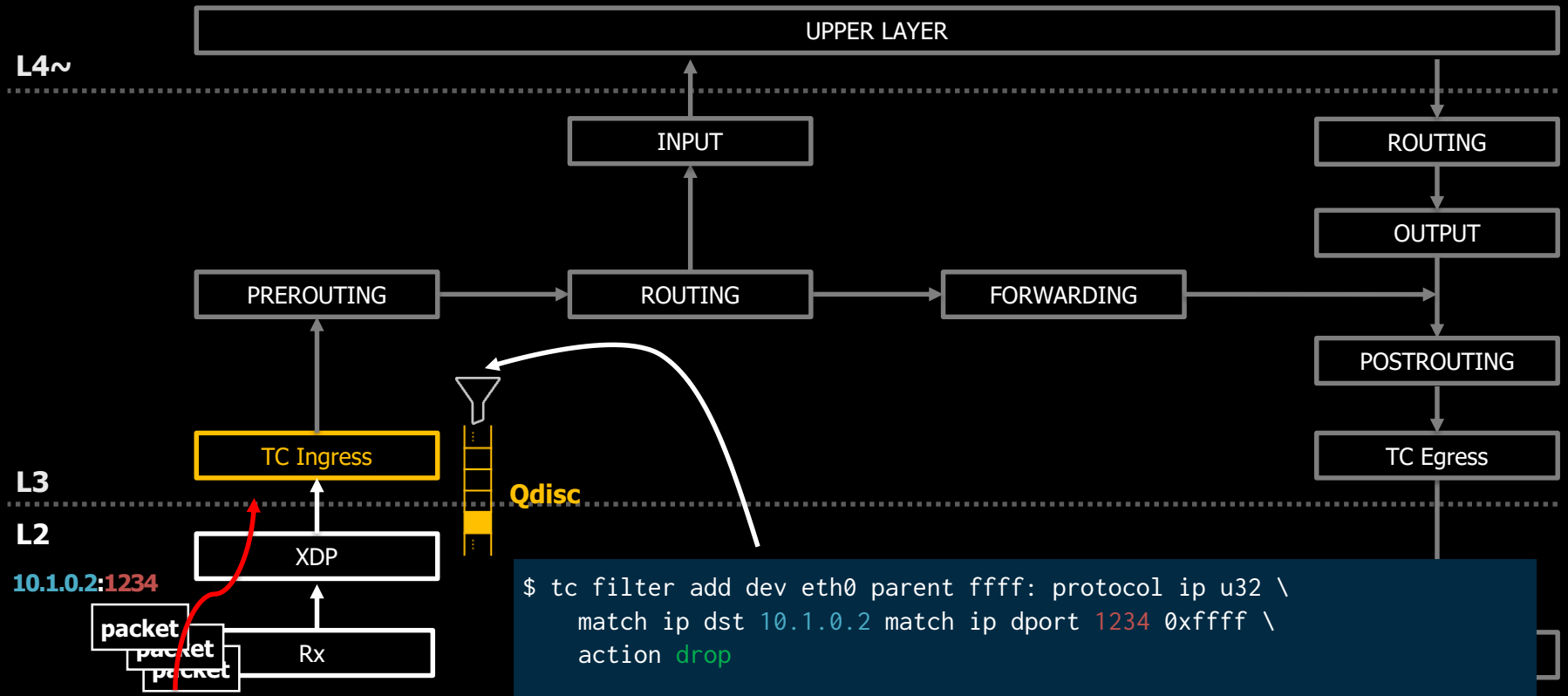
TC Example 1 – Packet Filtering



TC Example 1 – Packet Filtering

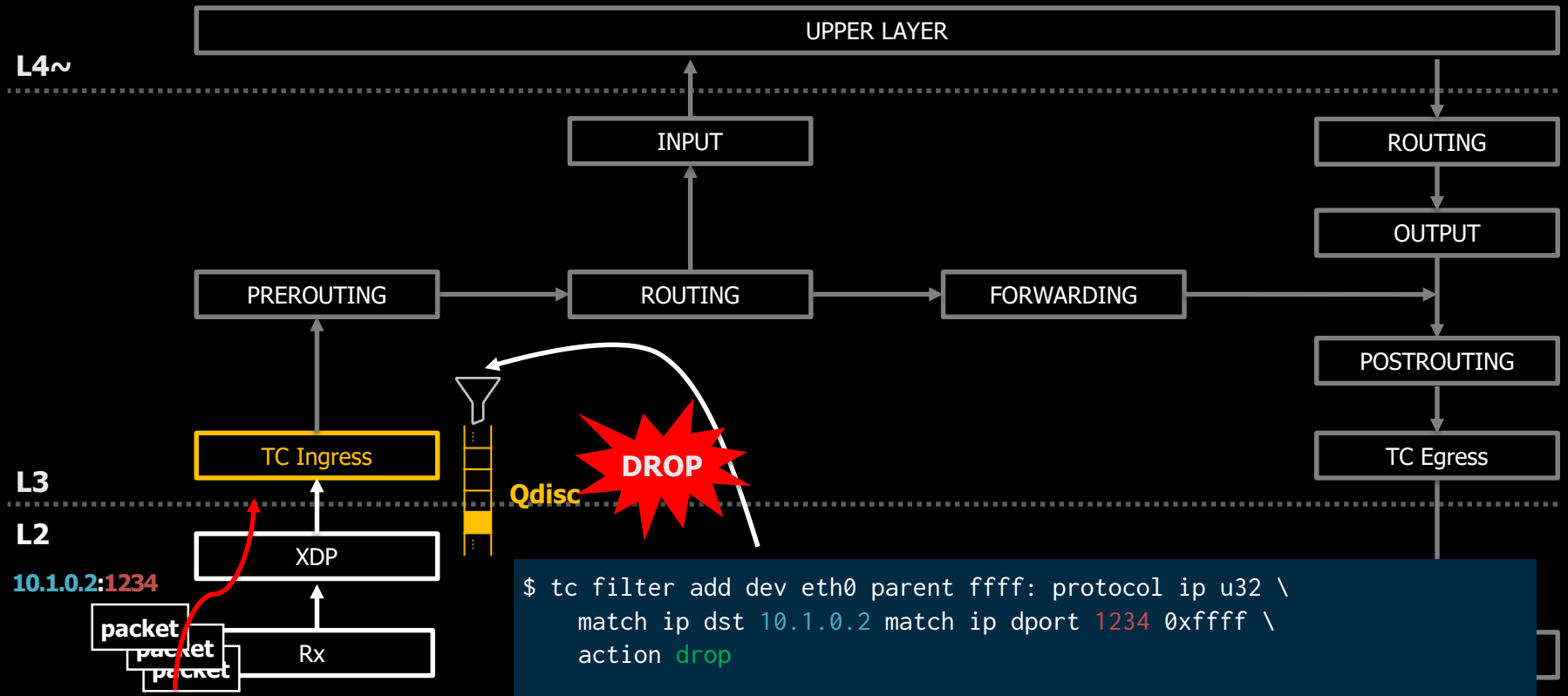


TC Example 1 – Packet Filtering




```
$ tc filter add dev eth0 parent ffff: protocol ip u32 \
    match ip dst 10.1.0.2 match ip dport 1234 0xffff \
    action drop
```

TC Example 1 – Packet Filtering




```
$ tc filter add dev eth0 parent ffff: protocol ip u32 \
    match ip dst 10.1.0.2 match ip dport 1234 0xffff \
    action drop
```

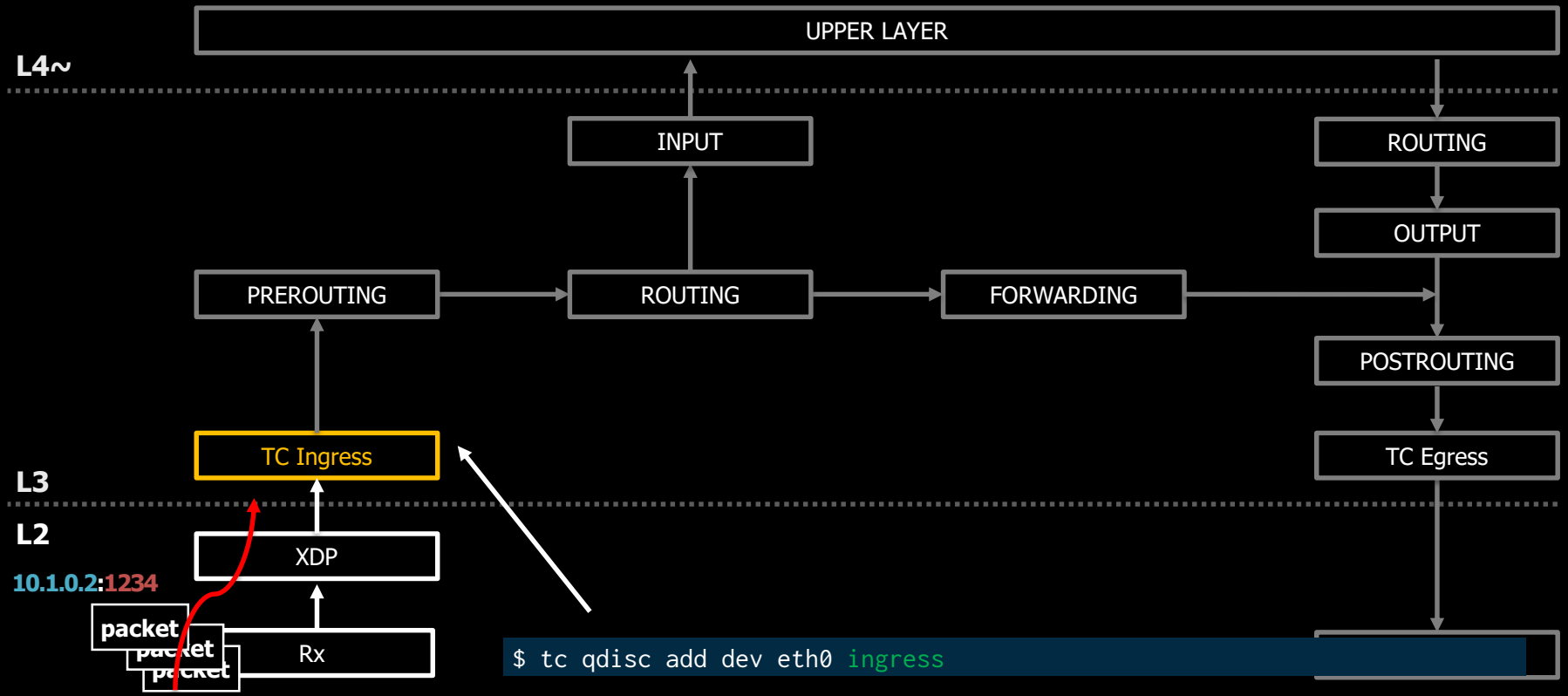
TC Example 2 – NAT

TC Ingress Destination NAT

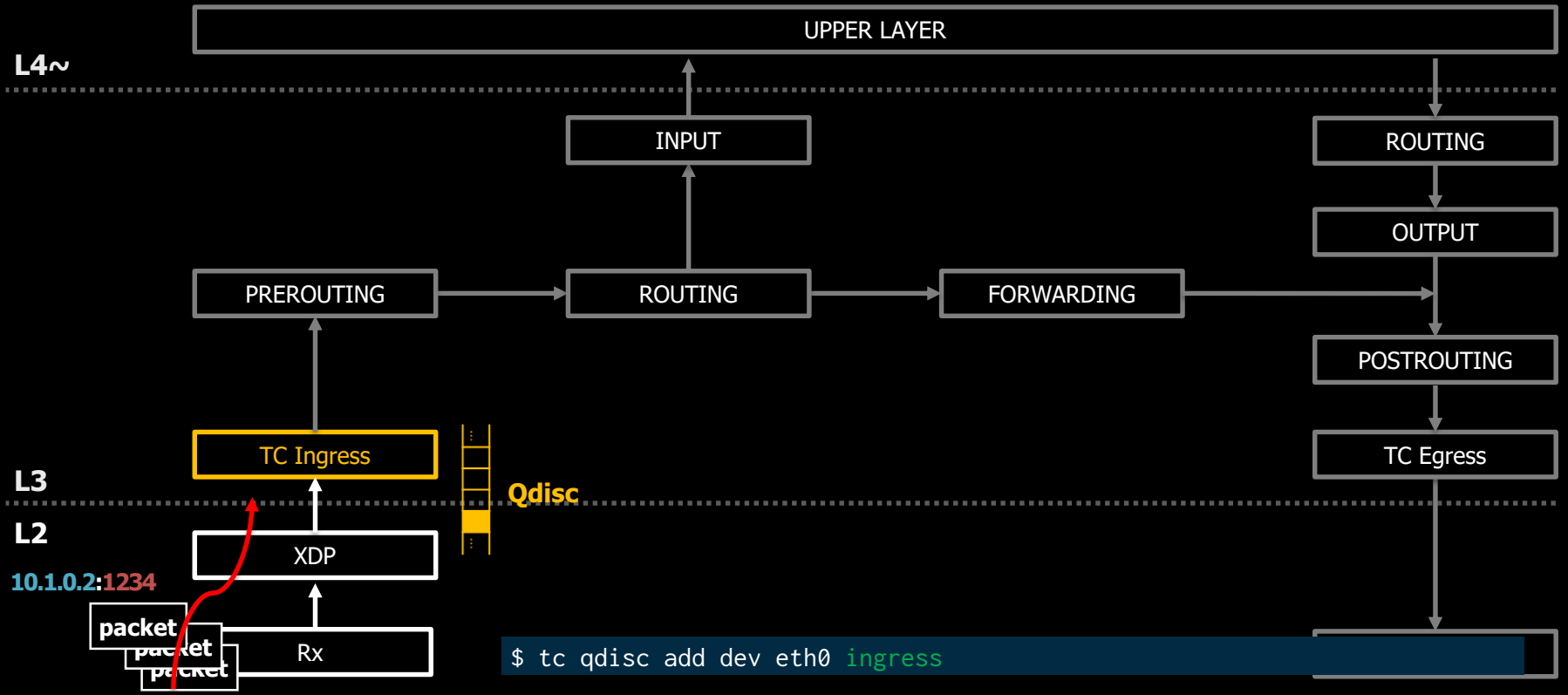
NAT UDP 10.1.0.2:1234 -> 10.1.1.2:1234

```
$ tc qdisc add dev eth0 ingress
$ tc filter add dev eth0 parent ffff: protocol ip u32 \
    match ip dst 10.1.0.2 match ip dport 1234 0xffff \
    action nat ingress 10.1.0.2/32 10.1.1.2/32 pipe \
    action mirred egress redirect dev eth1
```

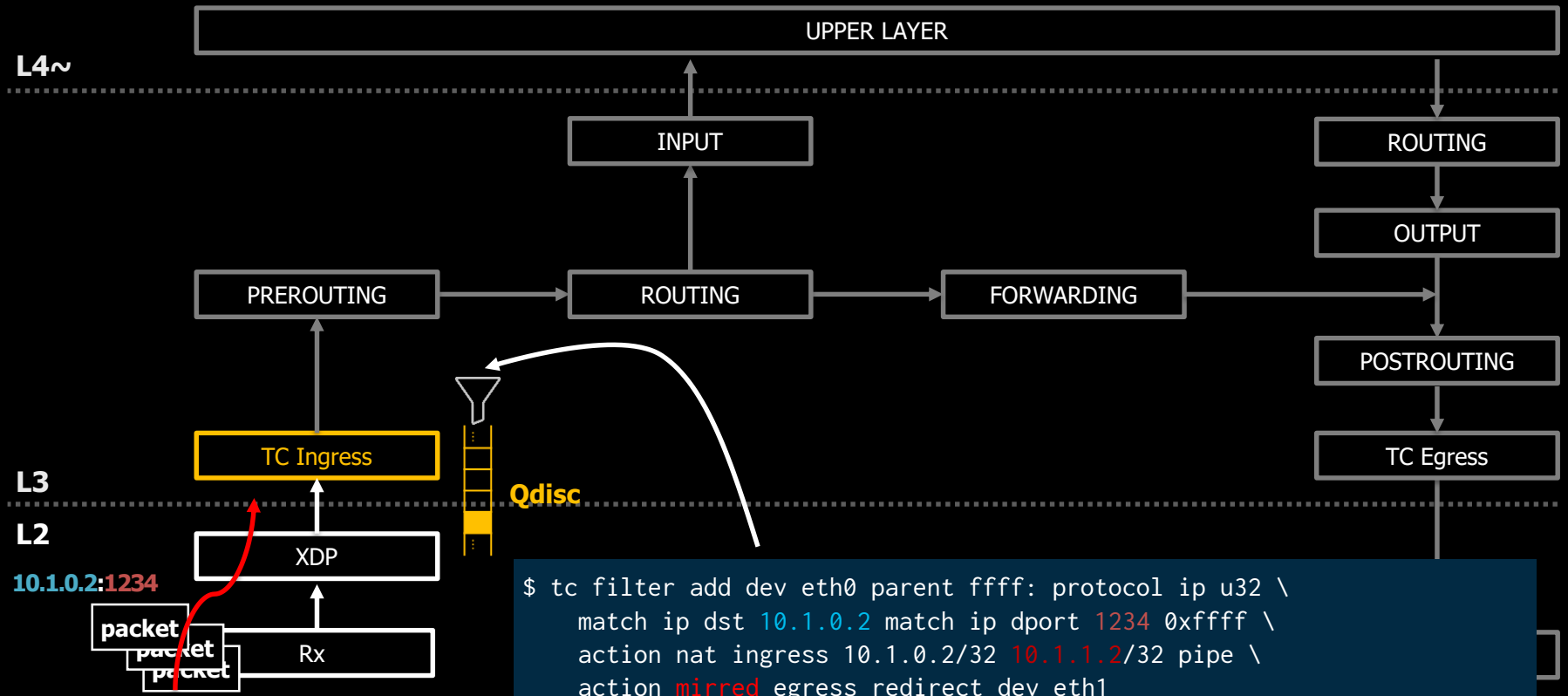
TC Example 2 – NAT



TC Example 2 – NAT

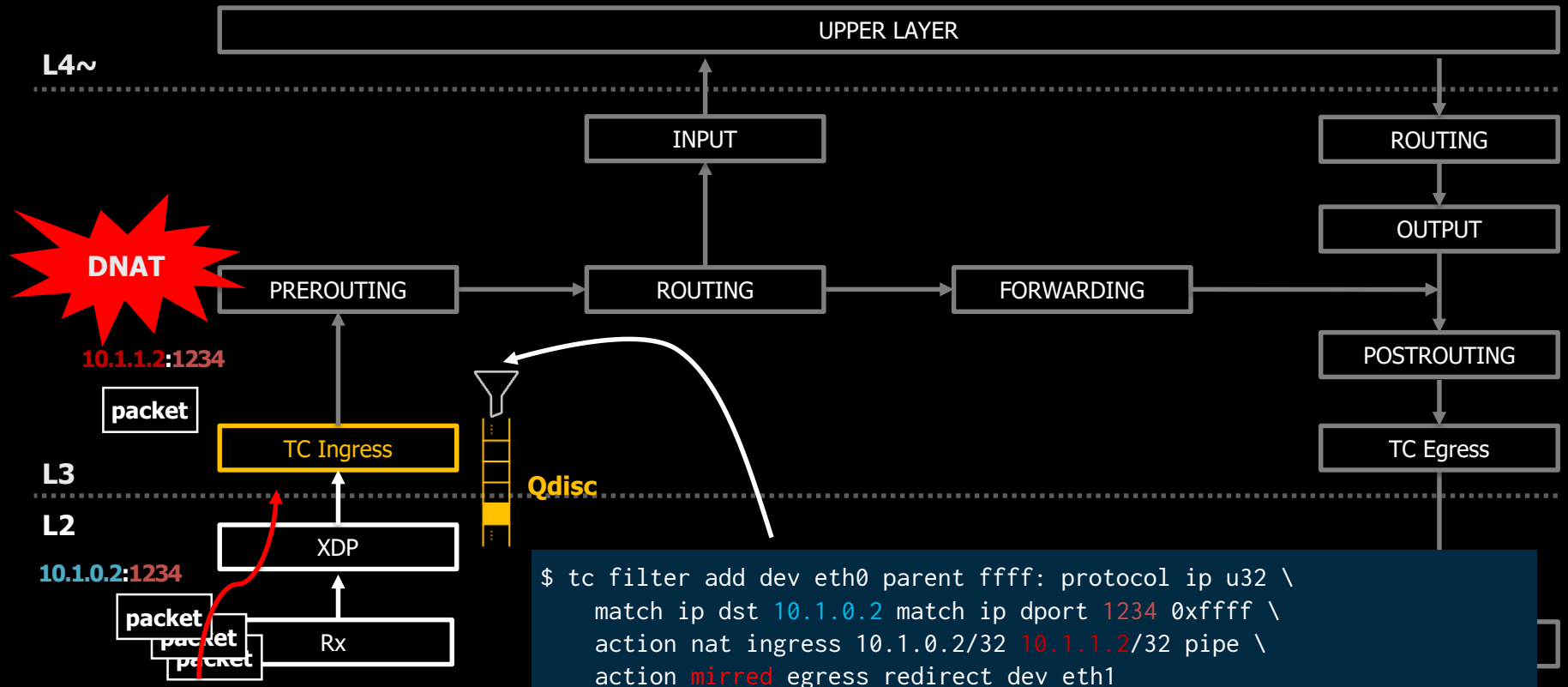


TC Example 2 – NAT



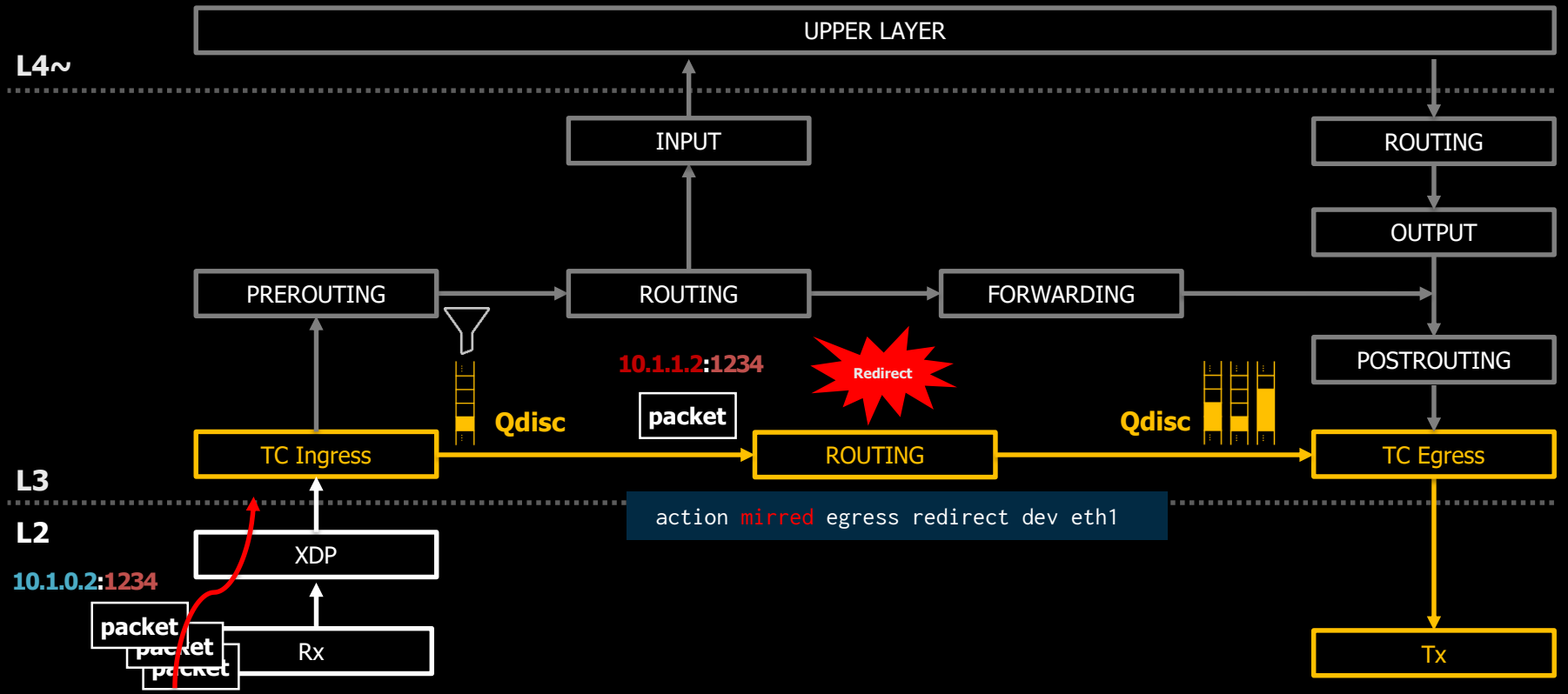

```
$ tc filter add dev eth0 parent ffff: protocol ip u32 \
    match ip dst 10.1.0.2 match ip dport 1234 0xffff \
    action nat ingress 10.1.0.2/32 10.1.1.2/32 pipe \
    action mirred egress redirect dev eth1
```

TC Example 2 – NAT



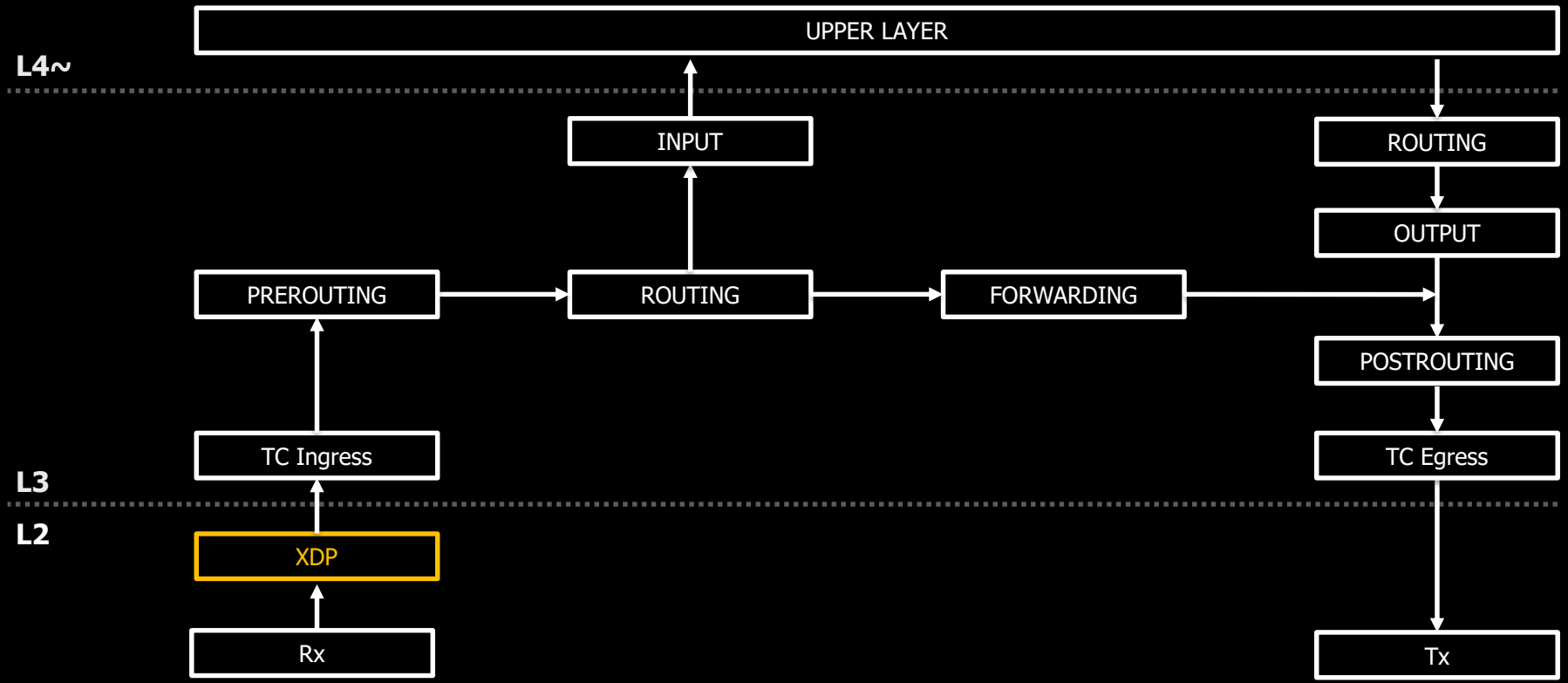

```
$ tc filter add dev eth0 parent ffff: protocol ip u32 \
    match ip dst 10.1.0.2 match ip dport 1234 0xffff \
    action nat ingress 10.1.0.2/32 10.1.1.2/32 pipe \
    action mirred egress redirect dev eth1
```

TC Example 2 – NAT



eBPF based fast data-path

XDP Packet Path



XDP

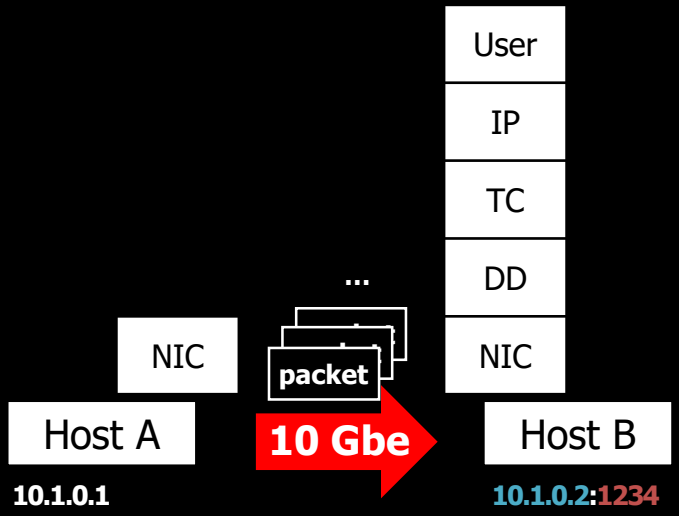
eBPF based fast data-path

How FAST are we talking?

Packet Drop

From zero to 14 Mpps

Test environment

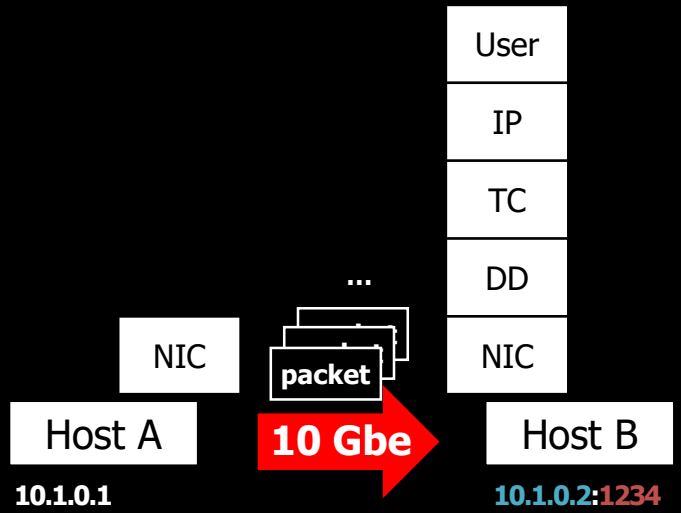


Connected with 10Gbe Ethernet

Test environment

Pktgen : Send UDP packet
(linux/samples/pktgen)

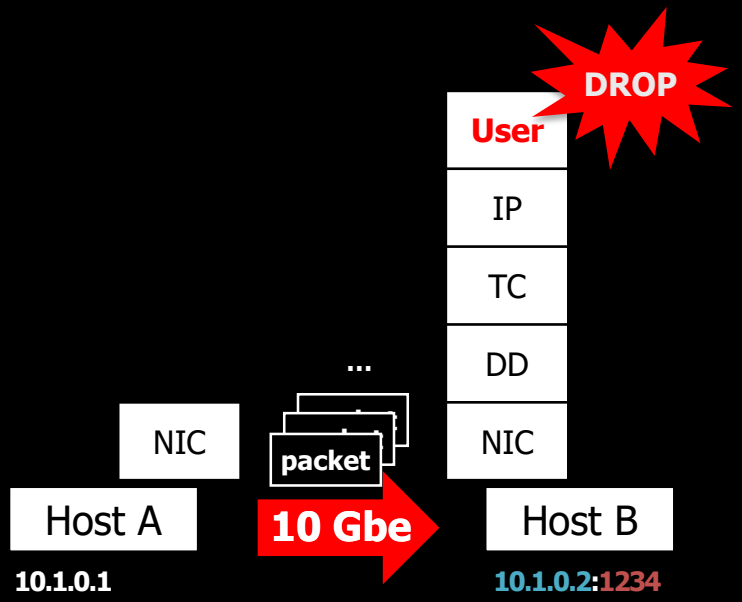
```
$ ./pktgen.sh -i eth0 -m $DST_MAC -d 10.1.0.2 -p 1234
Result device: eth0
Params: count 100000  min_pkt_size: 60  max_pkt_size: 60
...
     dst_min: 10.1.0.2
     dst_mac: $DST_MAC
     udp_dst: 1234
Current:     pkts-sofar: 100000  errors: 0
     started: 7802657us  stopped: 7862969us idle: 55us
     cur_saddr: 10.1.0.1  cur_daddr: 10.1.0.2
     cur_udp_dst: 1234  cur_udp_src: 109
     cur_queue_map: 0     flows: 0
Result: OK: 60312(c60257+d55) usec, 100000 (60byte,0frags)
1658039pps 795Mb/sec (795858720bps) errors: 0
```



Test environment

DROP UDP 10.1.0.2:1234

Packet Drop / Single Core?

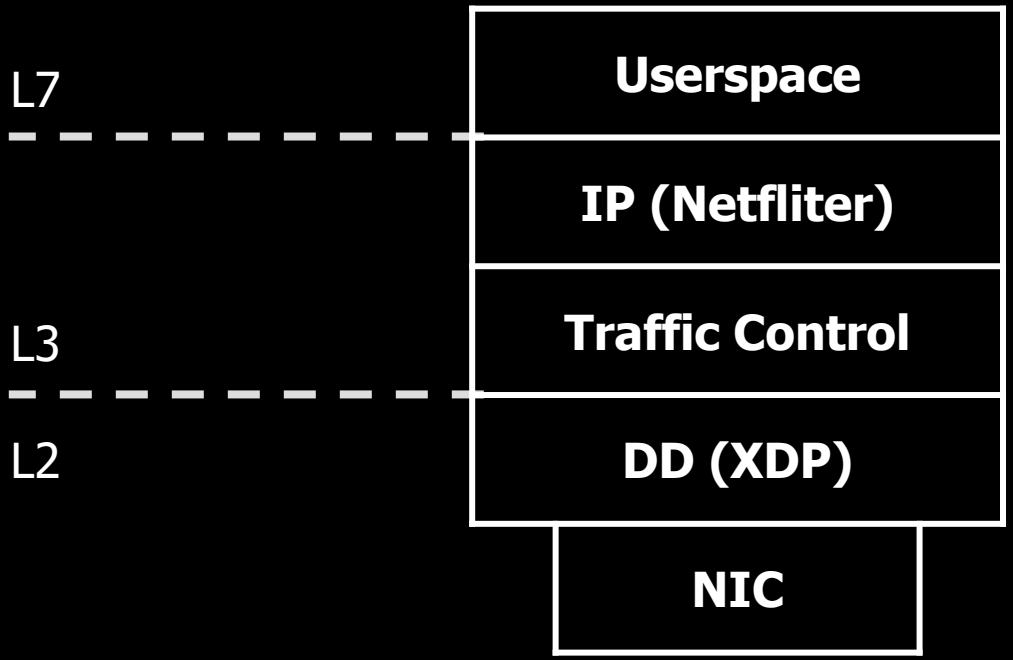


Theoretical speed of 10Gbe?

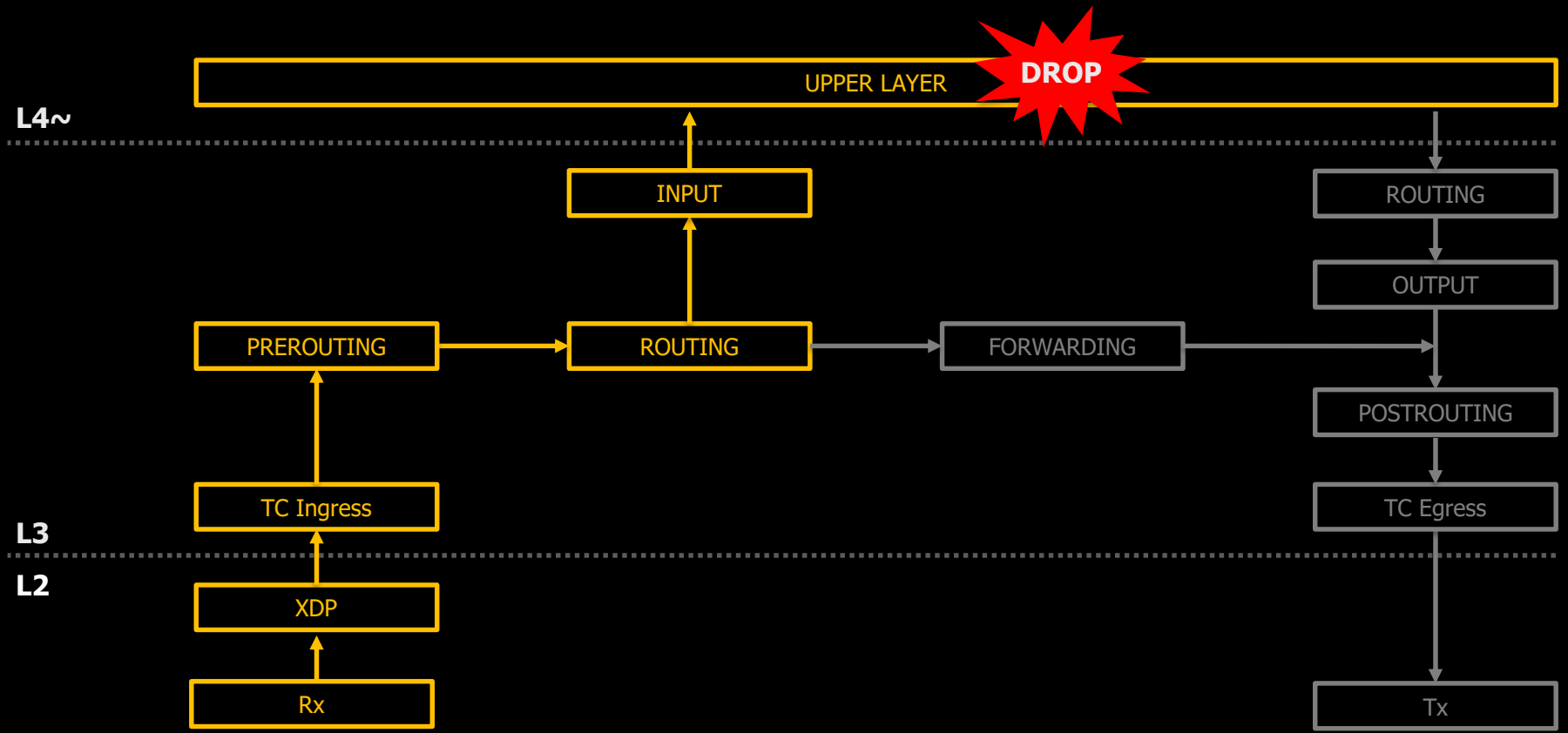
Preamble	Ethernet Frame					Inter Frame Gap
	MAC. Destination	MAC. Source	Type	PAYLOAD (IP/IPv6/ARP…)	CRC	
8 Bytes	6 Bytes	6 Bytes	2 Bytes	46-1500 Bytes	4 Bytes	12 Bytes

10Gbit = (10 * 10^9 bit) / (84 * 8bit)
= 14,880,952 pps (14Mpps)

Hooks to process packet?



Userspace Packet Drop

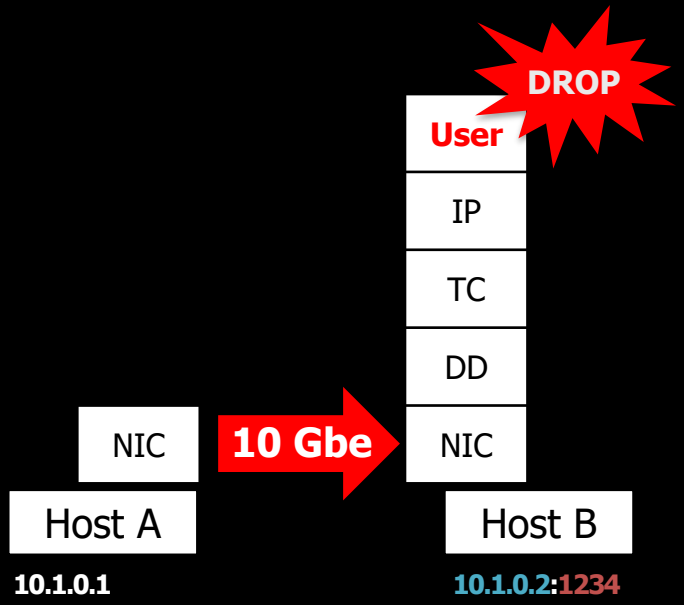


Userspace Packet Drop

UDP socket server
packet drop by reading socket

```
char buf[MTU_SIZE];

while (1) {
    int res = read(fd, buf, MTU_SIZE);

    if (res <= 0)
        return 0;

    pkts += 1;
    bytes += res;
}
```

user-drop.c



Userspace Packet Drop

```
$ gcc -o user-drop user-drop.c

$ sudo ./user-drop
packets=778148 bytes=14006664
packets=782171 bytes=14079078
packets=784792 bytes=14126256
packets=786466 bytes=14156388
packets=784163 bytes=14114934
packets=782500 bytes=14085000
packets=783085 bytes=14095530
packets=783172 bytes=14097096
```

Average Packet Drop

783,063pps/core

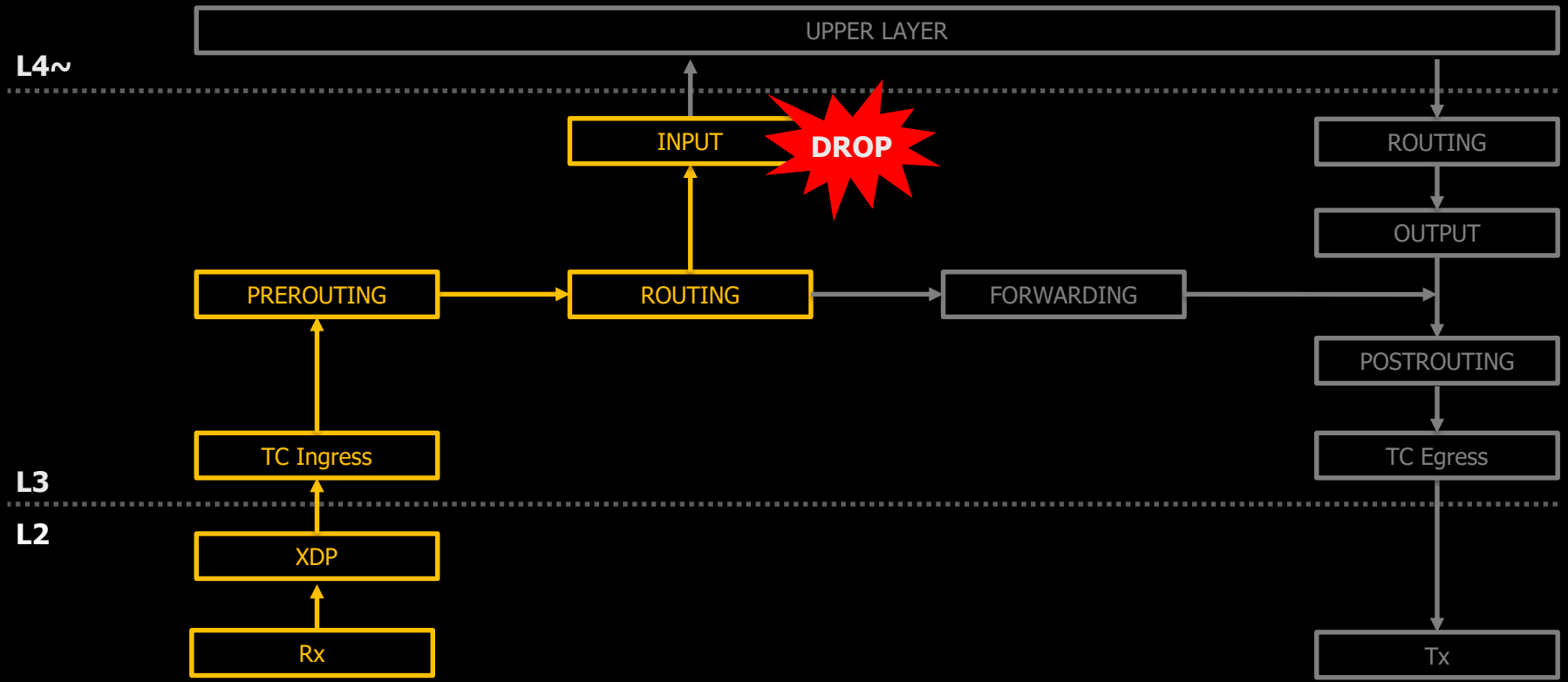
≈ 530Mbit/s

Userspace Packet Drop

Can it be faster?

YES!
By using Netfilter

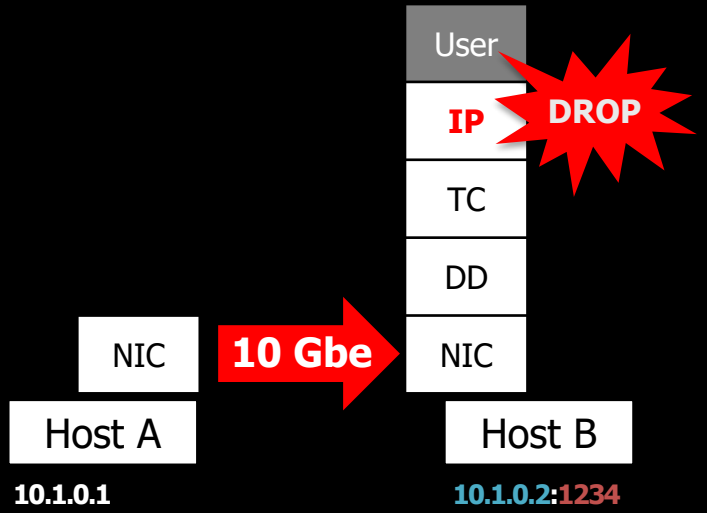
Netfilter Packet Drop



Netfilter Packet Drop

Netfilter (iptables)
packet drop at INPUT chain

```
$ iptables -A INPUT -d 10.1.0.2 -p udp --dport 1234 -j DROP

$ iptables -L INPUT
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       udp  --  anywhere             10.1.0.2             udp dpt:1234
```



Netfilter Packet Drop

```
$ iptables -vxnL INPUT
Chain INPUT (policy ACCEPT 79 packets, 4318 bytes)
    pkts      bytes target     prot opt in     out     source               destination
 2927763  134677098 DROP       udp  --  *      *       0.0.0.0/0            10.1.0.2     udp dpt:1234
 4235054  194812484 DROP       udp  --  *      *       0.0.0.0/0            10.1.0.2     udp dpt:1234
 5541468  254907528 DROP       udp  --  *      *       0.0.0.0/0            10.1.0.2     udp dpt:1234
 6397986  294307356 DROP       udp  --  *      *       0.0.0.0/0            10.1.0.2     udp dpt:1234
 7706050  354478300 DROP       udp  --  *      *       0.0.0.0/0            10.1.0.2     udp dpt:1234
 9013710  414630660 DROP       udp  --  *      *       0.0.0.0/0            10.1.0.2     udp dpt:1234
10320569  474746174 DROP       udp  --  *      *       0.0.0.0/0            10.1.0.2     udp dpt:1234
11629331  534949226 DROP       udp  --  *      *       0.0.0.0/0            10.1.0.2     udp dpt:1234
12937107  595106922 DROP       udp  --  *      *       0.0.0.0/0            10.1.0.2     udp dpt:1234
14243987  655223402 DROP       udp  --  *      *       0.0.0.0/0            10.1.0.2     udp dpt:1234
15553372  715455112 DROP       udp  --  *      *       0.0.0.0/0            10.1.0.2     udp dpt:1234
16861793  775642478 DROP       udp  --  *      *       0.0.0.0/0            10.1.0.2     udp dpt:1234
```

Average Packet Drop

1,266,730pps/core

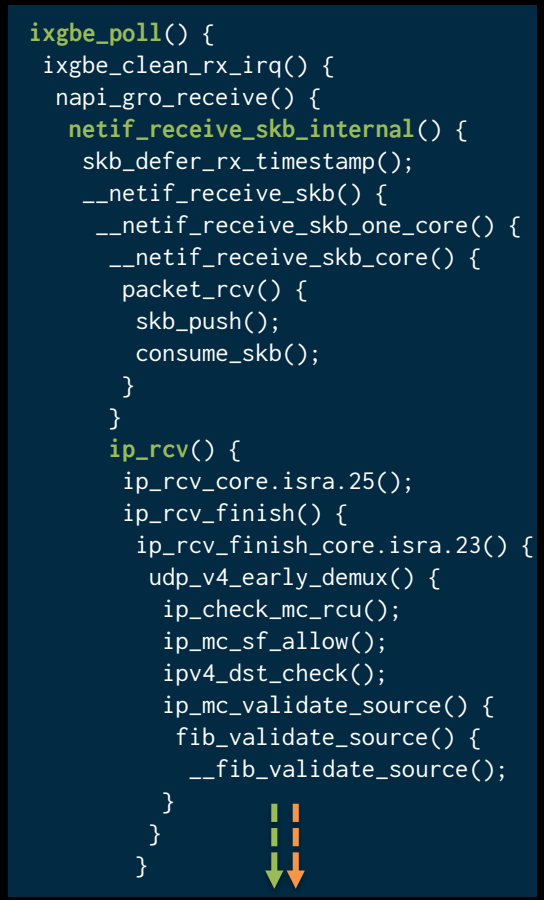
≈ 860Mbit/s

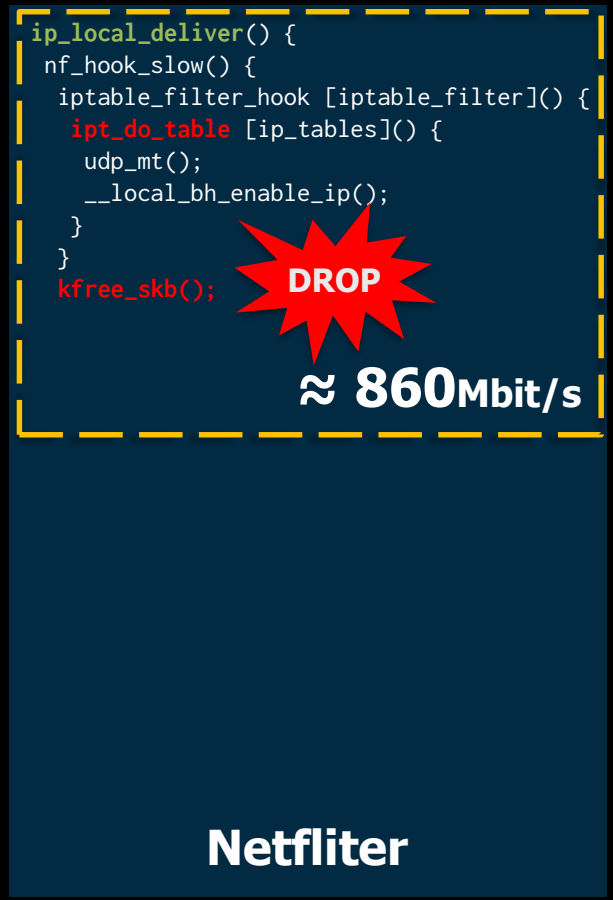
Netfilter Packet Drop

```
ixgbe_poll() {
 ixgbe_clean_rx_irq() {
  napi_gro_receive() {
   netif_receive_skb_internal() {
    skb_defer_rx_timestamp();
    __netif_receive_skb() {
     __netif_receive_skb_one_core() {
      __netif_receive_skb_core() {
       packet_rcv() {
        skb_push();
        consume_skb();
       }
      }
      ip_rcv() {
       ip_rcv_core.isra.25();
       ip_rcv_finish() {
        ip_rcv_finish_core.isra.23() {
         udp_v4_early_demux() {
          ip_check_mc_rcu();
          ip_mc_sf_allow();
          ipv4_dst_check();
          ip_mc_validate_source() {
           fib_validate_source() {
            __fib_validate_source();
           }
          }
         }
```

```
ip_local_deliver() {
 nf_hook_slow() {
  iptable_filter_hook [iptable_filter]() {
   ipt_do_table [ip_tables]() {
    udp_mt();
    __local_bh_enable_ip();
   }
  }
  kfree_skb();
```

Netfilter Packet Drop

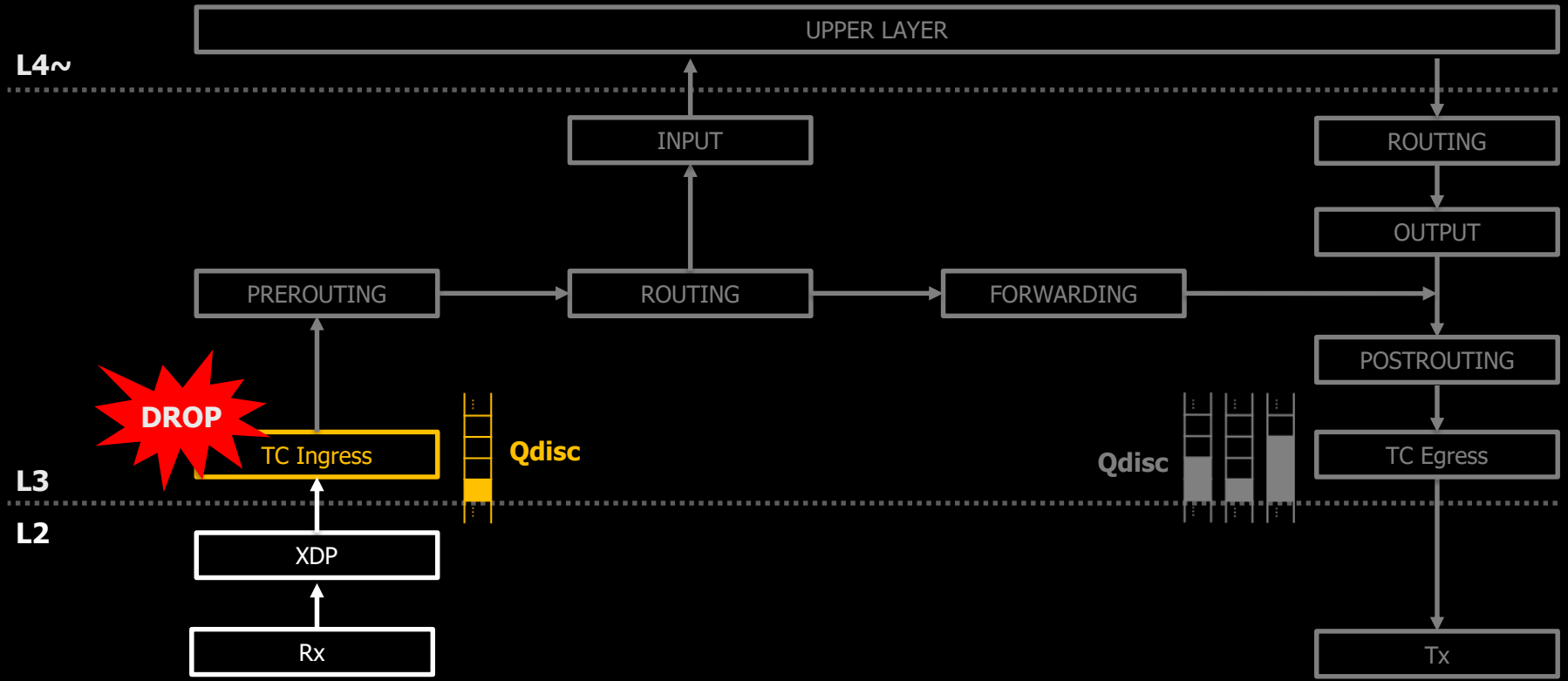




Is it fast enough?

NO!
With TC, it can be faster

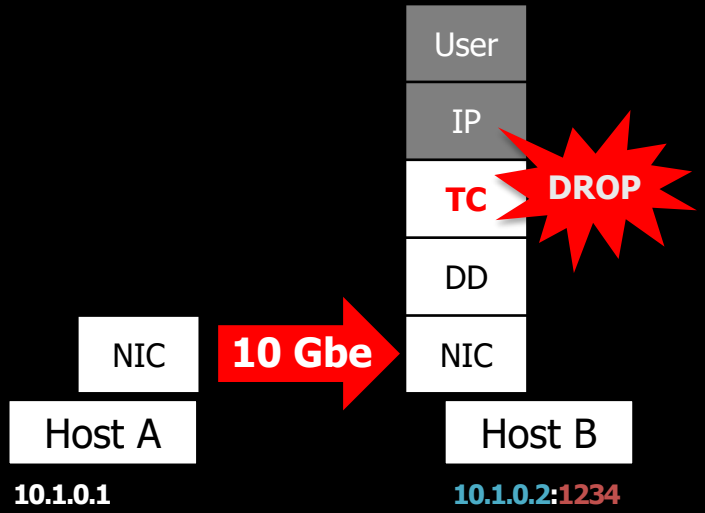
TC Ingress Packet Drop



TC Ingress Packet Drop

TC Ingress
packet drop with filter action

```
$ tc qdisc add dev eth0 ingress
$ tc filter add dev eth0 parent ffff: protocol ip u32 \
    match ip dst 10.1.0.2 match ip dport 1234 0xffff \
    action drop

$ tc filter show ingress dev eth0
...
  match 0a010002/ffffffff at 16
  match 000004d2/0000ffff at 20
        action order 1: gact action drop
         random type none pass val 0
         index 1 ref 1 bind 1
```



TC Ingress Packet Drop

```
$ tc -s filter show ingress dev eth0
…
  match 0a010002/ffffffff at 16
  match 000004d2/0000ffff at 20
        action order 1: gact action drop
         random type none pass val 0
         index 1 ref 1 bind 1 installed 62 sec used 62 sec
        Action statistics:
        Sent 154709776 bytes 3363256 pkt (dropped 3363260, ...)
        Sent 345955052 bytes 7520762 pkt (dropped 7520767, ...)
        Sent 537288326 bytes 11680181 pkt (dropped 11680186, ...)
        Sent 728875750 bytes 15845125 pkt (dropped 15845130, ...)
        Sent 920311328 bytes 20006768 pkt (dropped 20006772, ...)
        Sent 1112155754 bytes 24177299 pkt (dropped 24177304, ...)
        Sent 1276514214 bytes 27750309 pkt (dropped 27750309, ...)
        Sent 1453930004 bytes 31607174 pkt (dropped 31607179, ...)
        Sent 1645855758 bytes 35779473 pkt (dropped 35779477, ...)
        Sent 1838232266 bytes 39961571 pkt (dropped 39961577, ...)
        Sent 2029302696 bytes 44115276 pkt (dropped 44115282, ...)
        Sent 2221122512 bytes 48285272 pkt (dropped 48285278, ...)
        Sent 2412335864 bytes 52442084 pkt (dropped 52442089, ...)
        Sent 2603632476 bytes 56600706 pkt (dropped 56600711, ...)
```

Average Packet Drop

4,083,820pps/core

≈ 2.75Gbit/s

TC Ingress Packet Drop

```
ixgbe_poll() {
 ixgbe_clean_rx_irq() {
  napi_gro_receive() {
   netif_receive_skb_internal() {
    skb_defer_rx_timestamp();
    __netif_receive_skb() {
     __netif_receive_skb_one_core() {
      __netif_receive_skb_core() {
       tcf_classify() {
        u32_classify [cls_u32]() {
         tcf_action_exec() {
          tcf_gact_act [act_gact]();
         }
        }
       }
       kfree_skb();
```

TC Ingress Packet Drop

```
ixgbe_poll() {
 ixgbe_clean_rx_irq() {
  napi_gro_receive() {
   netif_receive_skb_internal() {
    skb_defer_rx_timestamp();
    __netif_receive_skb() {
     __netif_receive_skb_one_core() {
      __netif_receive_skb_core() {
       tcf_classify() {
        u32_classify [cls_u32]() {
         tcf_action_exec() {
          tcf_gact_act [act_gact]();
         }
        }
       }
       kfree_skb();
```

DROP

≈ 2.75Gbit/s

TC

```
ixgbe_poll() {
 ixgbe_clean_rx_irq() {
  napi_gro_receive() {
   netif_receive_skb_internal() {
    skb_defer_rx_timestamp();
    __netif_receive_skb() {
     __netif_receive_skb_one_core() {
      __netif_receive_skb_core() {
       packet_rcv() {
        skb_push();
        consume_skb();
       }
      }
      ip_rcv() {
       ip_rcv_core.isra.25();
       ip_rcv_finish() {
        ip_rcv_finish_core.isra.23() {
         udp_v4_early_demux() {
          ip_check_mc_rcu();
          ip_mc_sf_allow();
          ipv4_dst_check();
          ip_mc_validate_source() {
           fib_validate_source() {
            __fib_validate_source();
           }
          }
         }
```

```
ip_local_deliver() {
 nf_hook_slow() {
  iptable_filter_hook [iptable_filter]() {
   ipt_do_table [ip_tables]() {
    udp_mt();
    __local_bh_enable_ip();
   }
  }
  kfree_skb();
```

DROP

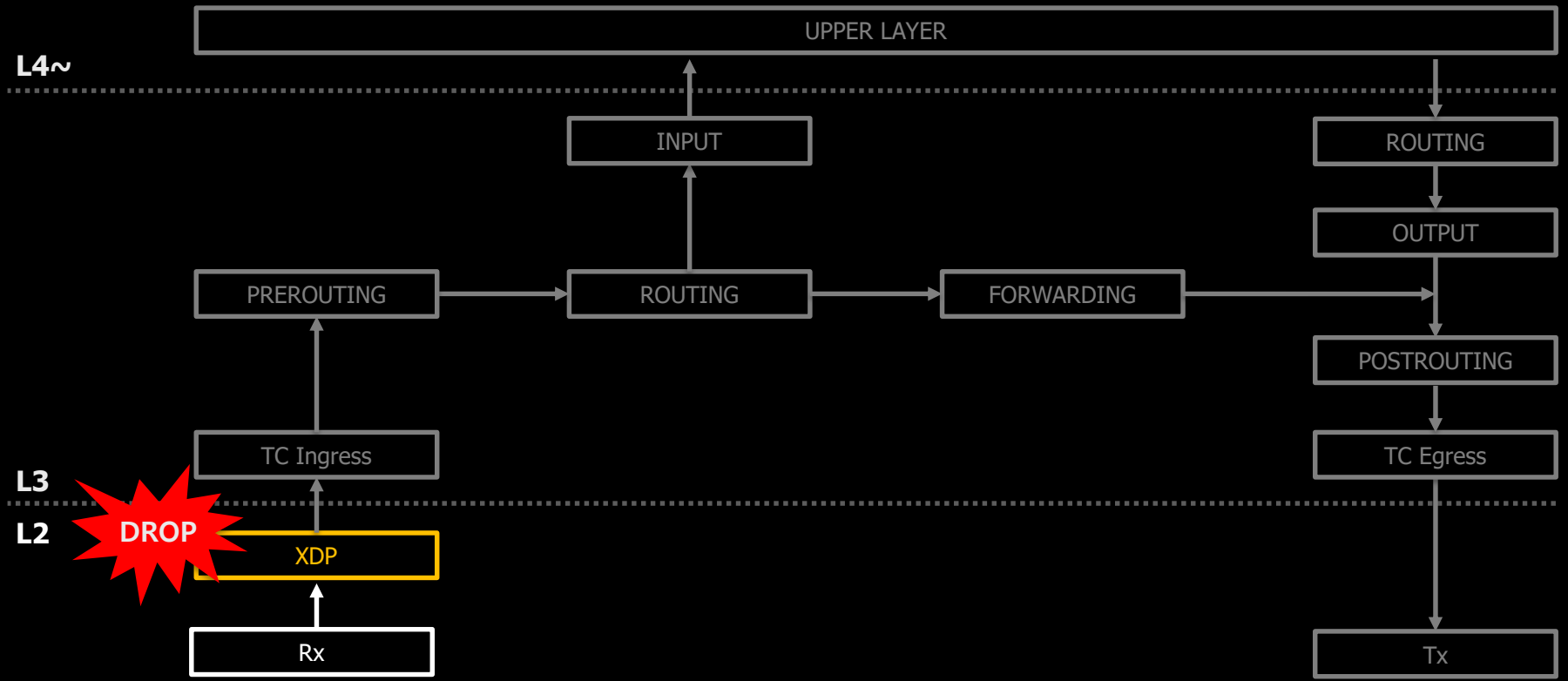
≈ 860Mbit/s

Netfliter

More faster?

Sure thing!
It's faster with XDP

XDP Packet Drop



XDP Packet Drop

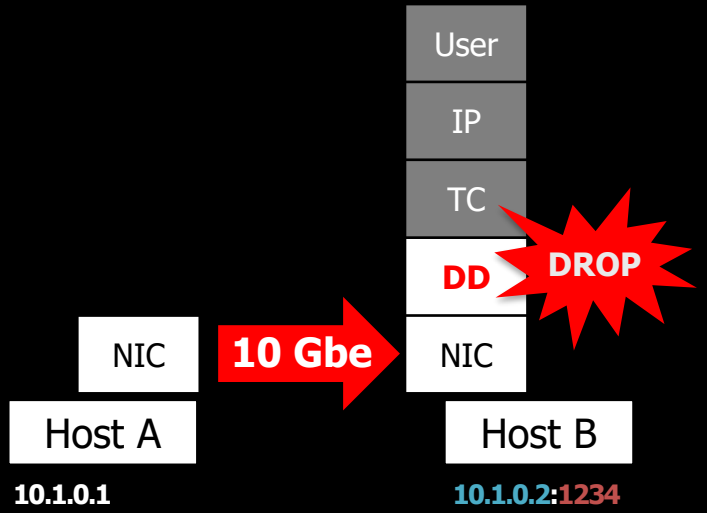
```
SEC("xdp1")
int xdp_prog1(struct xdp_md *xdp) {
  void *data_end = (void *)(long)xdp->data_end;
  void *data = (void *)(long)xdp->data;
  struct ethhdr *eth = data;
  struct iphdr *iph;
  struct udphdr *uh;
  ...
  if (eth->h_proto == htons(ETH_P_IP)) {
    iph = data + sizeof(*eth);
    ...
    if (iph->daddr == _htonl(0xa010002)) {
      if (iph->protocol == IPPROTO_UDP) {
        uh = data + sizeof(*eth) + sizeof(*iph);
        ...

        if (uh->dest == htons(1234))
          return XDP_DROP;
      }
    }
  }
   return XDP_PASS;
}
```

DROP UDP 10.1.0.2:1234

XDP Packet Drop

XDP
packet drop with XDP_DROP

```
$ bpftool prog load ./xdp-drop.o /sys/fs/bpf/drop
$ bpftool prog show
...
24: xdp  name xdp_prog1  tag 6f8c2e06dfa2abcb  gpl
        loaded_at 2019-10-11T17:17:33+0900  uid 0
        xlated 544B  jited 344B  memlock 4096B  map_ids 17

$ bpftool net attach xdp id 24 dev eth0
$ bpftool net
xdp:
eth0(8) driver id 24
```



XDP Packet Drop

```
$ ethtool -S eth0 | grep rx_xdp_drop
     rx_xdp_drop: 6161010
     rx_xdp_drop: 16114329
     rx_xdp_drop: 26079532
     rx_xdp_drop: 36025920
     rx_xdp_drop: 45958488
     rx_xdp_drop: 55869376
     rx_xdp_drop: 65835136
     rx_xdp_drop: 75748898
     rx_xdp_drop: 85670845
     rx_xdp_drop: 95635591
     rx_xdp_drop: 105569230
     rx_xdp_drop: 115515714
     rx_xdp_drop: 125480832
     rx_xdp_drop: 135432211
     rx_xdp_drop: 1455190421
```

Average Packet Drop

9,941,337 pps/core

≈ 6.69 Gbit/s

XDP Packet Drop

```
ixgbe_poll() {
 ixgbe_clean_rx_irq() {
  ixgbe_get_rx_buffer();
  ixgbe_run_xdp() {
   bpf_prog_run_xdp();
  }
```

XDP Packet Drop

```
ixgbe_poll() {
 ixgbe_clean_rx_irq() {
  ixgbe_get_rx_buffer();
  ixgbe_run_xdp() {
   bpf_prog_run_xdp();
  }
```

DROP

≈ 6.69Gbit/s

XDP

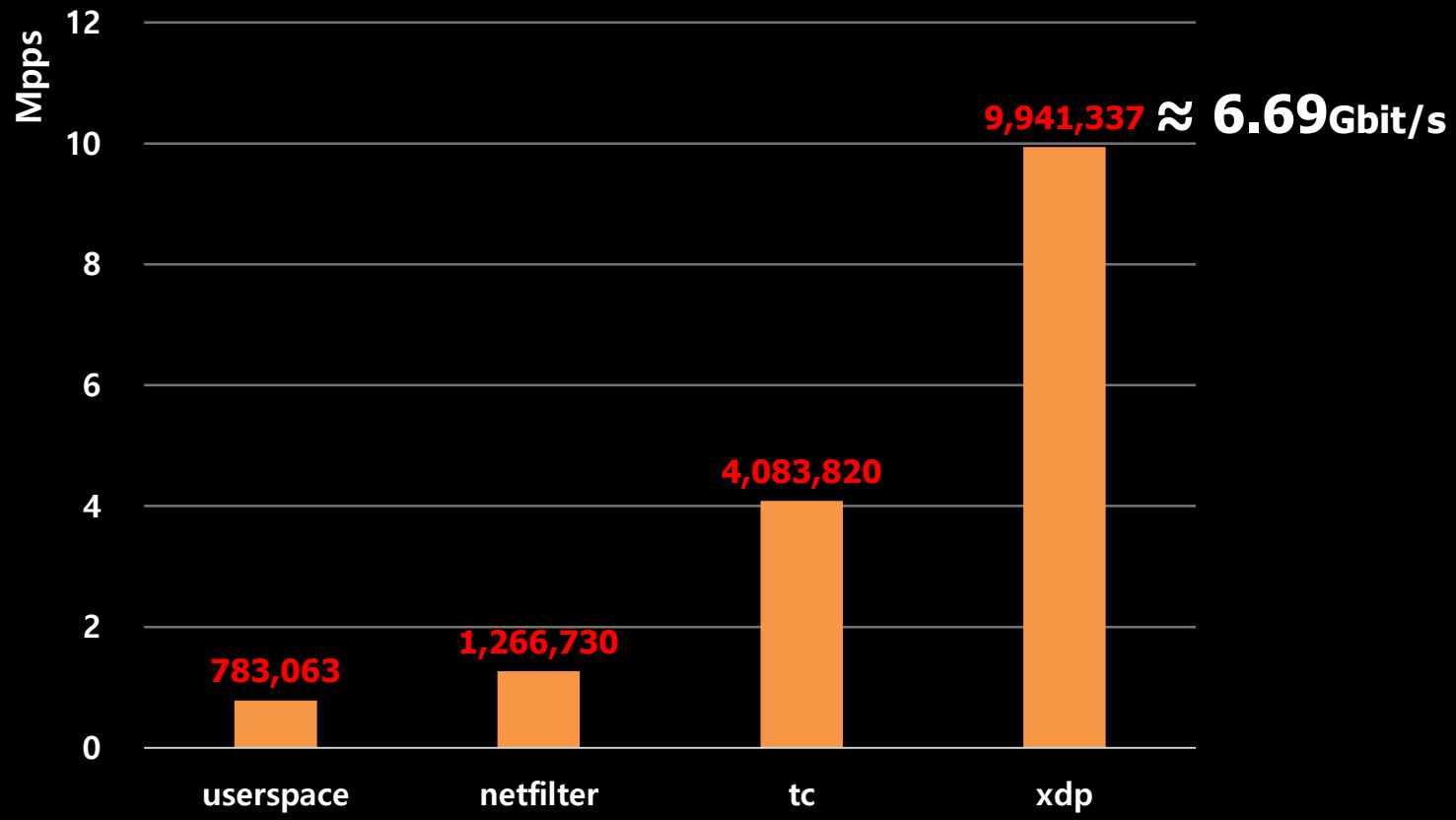
```
ixgbe_poll() {
 ixgbe_clean_rx_irq() {
  napi_gro_receive() {
   netif_receive_skb_internal() {
    skb_defer_rx_timestamp();
    __netif_receive_skb() {
     __netif_receive_skb_one_core() {
      __netif_receive_skb_core() {
       tcf_classify() {
        u32_classify [cls_u32]() {
         tcf_action_exec() {
          tcf_gact_act [act_gact]();
         }
        }
       }
       kfree_skb();
```

DROP

≈ 2.75Gbit/s

TC

Packet Drop Results



XDP? Super FAST!

What is XDP?

The definition of XDP

What is XDP?

eBPF based fast data-path

What is XDP?

eBPF based fast data-path

What is BPF?

in the kernel is similar with
V8 in Chrome browser

What is BPF?

In kernel Virtual Machine

What is BPF?

An virtual machine?
What does BPF look like?

What is BPF?

You may already used it!

What is BPF?

You may already used it!

An tcpdump!

What is BPF?

```
$ tcpdump -i eth0 'udp and dst 10.1.0.2'
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
00:19:42.244270 IP 192.168.10.1.221 > 10.1.0.2.1234: UDP, length 18
00:19:42.244270 IP 192.168.20.1.813 > 10.1.0.2.1234: UDP, length 18
00:19:42.244270 IP 192.168.30.1.856 > 10.1.0.2.1234: UDP, length 18
00:19:42.244270 IP 192.168.40.1.959 > 10.1.0.2.1234: UDP, length 18
00:19:42.244271 IP 192.168.10.1.160 > 10.1.0.2.1234: UDP, length 18
00:19:42.244271 IP 192.168.20.1.102 > 10.1.0.2.1234: UDP, length 18
00:19:42.244276 IP 192.168.30.1.114 > 10.1.0.2.1234: UDP, length 18
7 packets captured
10 packets received by filter
3 packets dropped by kernel
```

What is BPF?

Packet filter by BPF!

What is BPF?

```
$ tcpdump -i eth0 -d 'udp and dst 10.1.0.2'
```

tcpdump with –d option

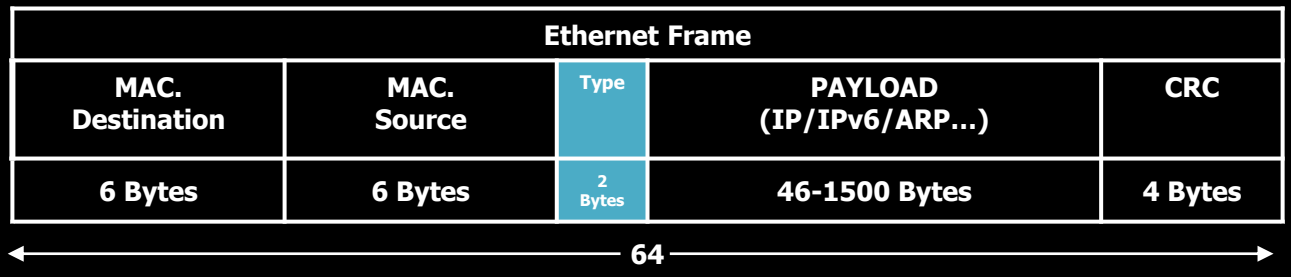
What is BPF?

```
$ tcpdump -i eth0 -d 'udp and dst 10.1.0.2'

(000) ldh      [12]
(001) jeq      #0x800           jt 2 jf 7

(002) ldb      [23]
(003) jeq      #0x11            jt 4 jf 7

(004) ld       [30]
(005) jeq      #0xa010002       jt 6 jf 7
(006) ret      #262144
(007) ret      #0
```

tcpdump with –d option

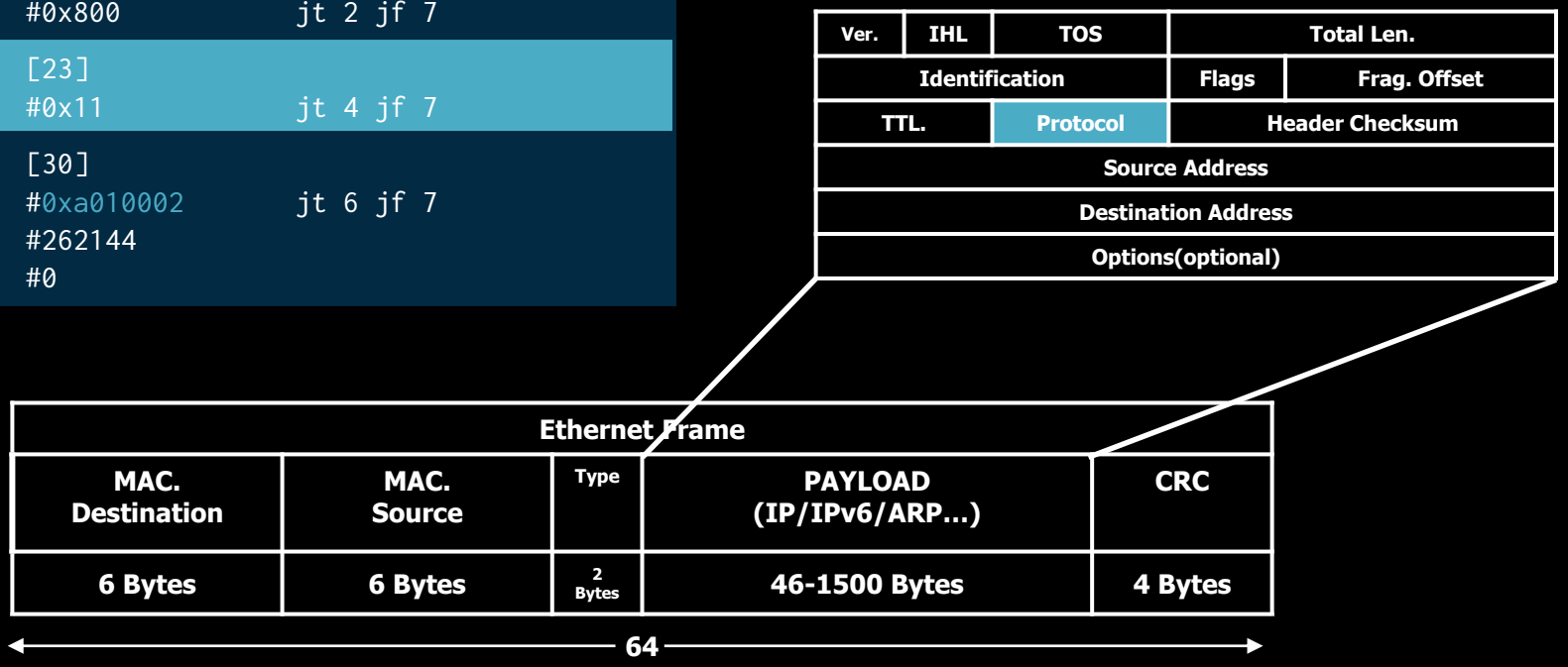
What is BPF?

```
$ tcpdump -i eth0 -d 'udp and dst 10.1.0.2'
(000) ldh      [12]
(001) jeq      #0x800           jt 2 jf 7
(002) ldb      [23]
(003) jeq      #0x11            jt 4 jf 7
(004) ld       [30]
(005) jeq      #0xa010002       jt 6 jf 7
(006) ret      #262144
(007) ret      #0
```



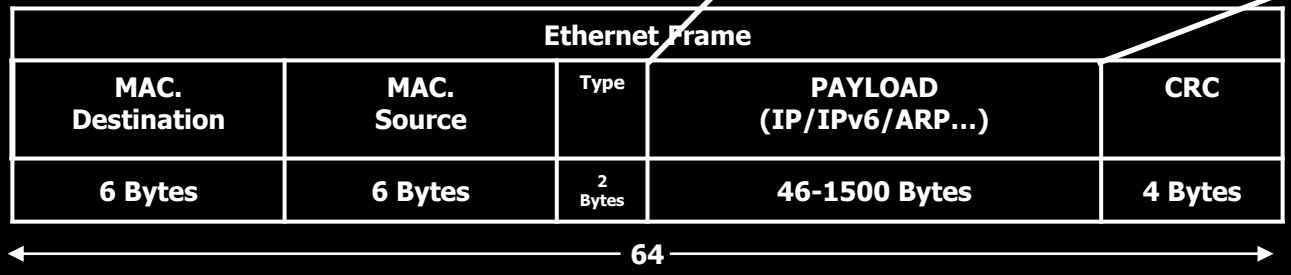
What is BPF?

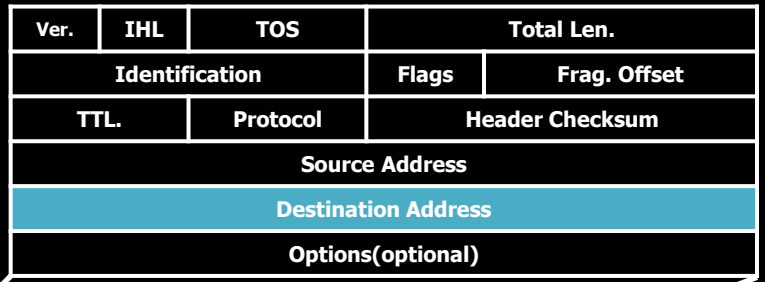
```
$ tcpdump -i eth0 -d 'udp and dst 10.1.0.2'
(000) ldh      [12]
(001) jeq      #0x800           jt 2 jf 7
(002) ldb      [23]
(003) jeq      #0x11            jt 4 jf 7
(004) ld       [30]
(005) jeq      #0xa010002       jt 6 jf 7
(006) ret      #262144
(007) ret      #0
```



What is BPF?

```
$ tcpdump -i eth0 -d 'udp and dst 10.1.0.2'

(000) ldh      [12]
(001) jeq      #0x800           jt 2 jf 7
(002) ldb      [23]
(003) jeq      #0x11            jt 4 jf 7

(004) ld       [30]
(005) jeq      #0xa010002       jt 6 jf 7

(006) ret      #262144
(007) ret      #0
```





What is BPF?

In kernel Virtual Machine

"Linux kernel code execution engine"

What is BPF?

"Run code in the kernel"

What is BPF?

"Run code in the kernel"

```
$ readelf -h bpf-prog.o  | grep Machine

  Machine:    Advanced Micro Devices X86-64
```

What is BPF?

"Run code in the kernel"

```
$ readelf -h bpf-prog.o  | grep Machine
  Machine:    Linux BPF
  Machine:    Advanced Micro Devices X86-64
```

writing C program

↓

clang / llc

↓

BPF instruction

What is BPF?

"Run code in the kernel"

```
$ readelf -h bpf-prog.o  | grep Machine
  Machine:    Linux BPF
  Machine:    Advanced Micro Devices X86-64
```

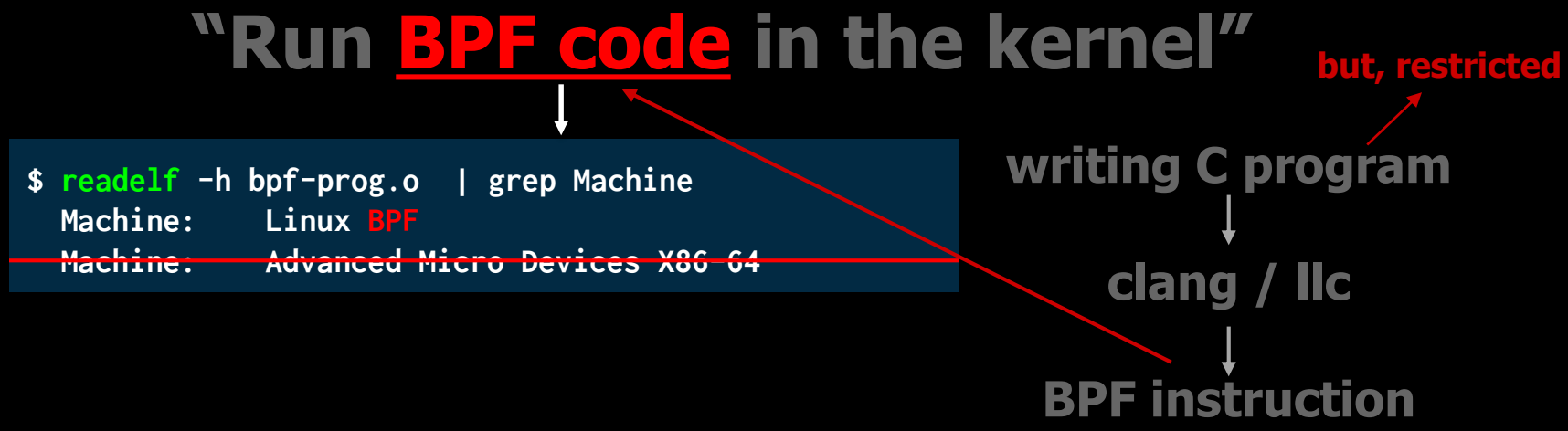
but, restricted

writing C program

clang / llc

BPF instruction

What is BPF?


"Run BPF code in the kernel"
but, restricted
$ readelf -h bpf-prog.o | grep Machine
Machine: Linux BPF
Machine: Advanced Micro Devices X86-64
writing C program
clang / llc
BPF instruction

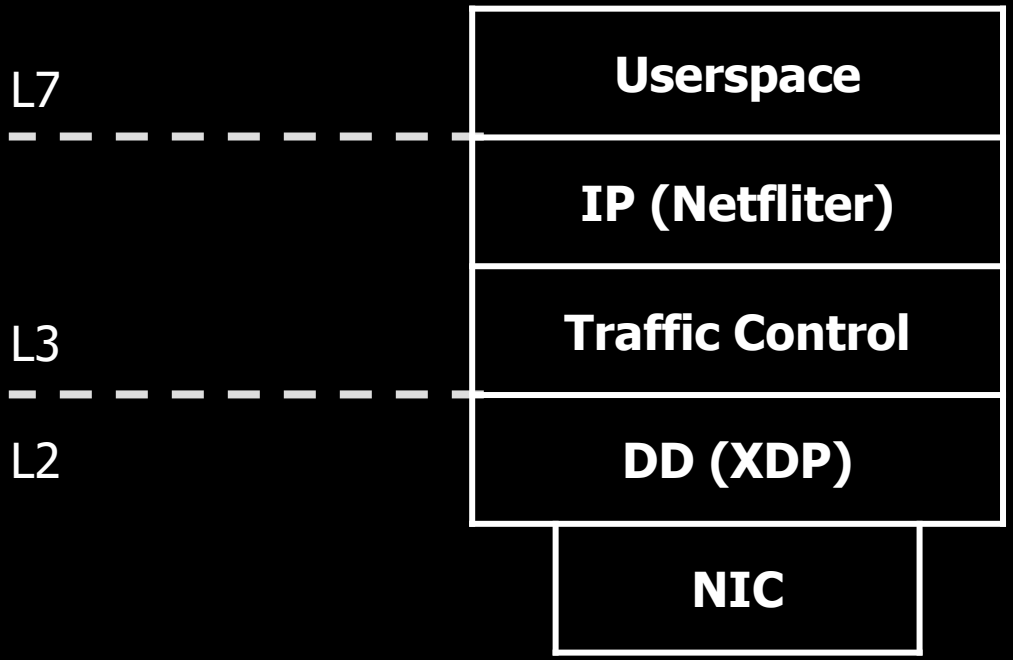
What is XDP?

eBPF based fast data-path

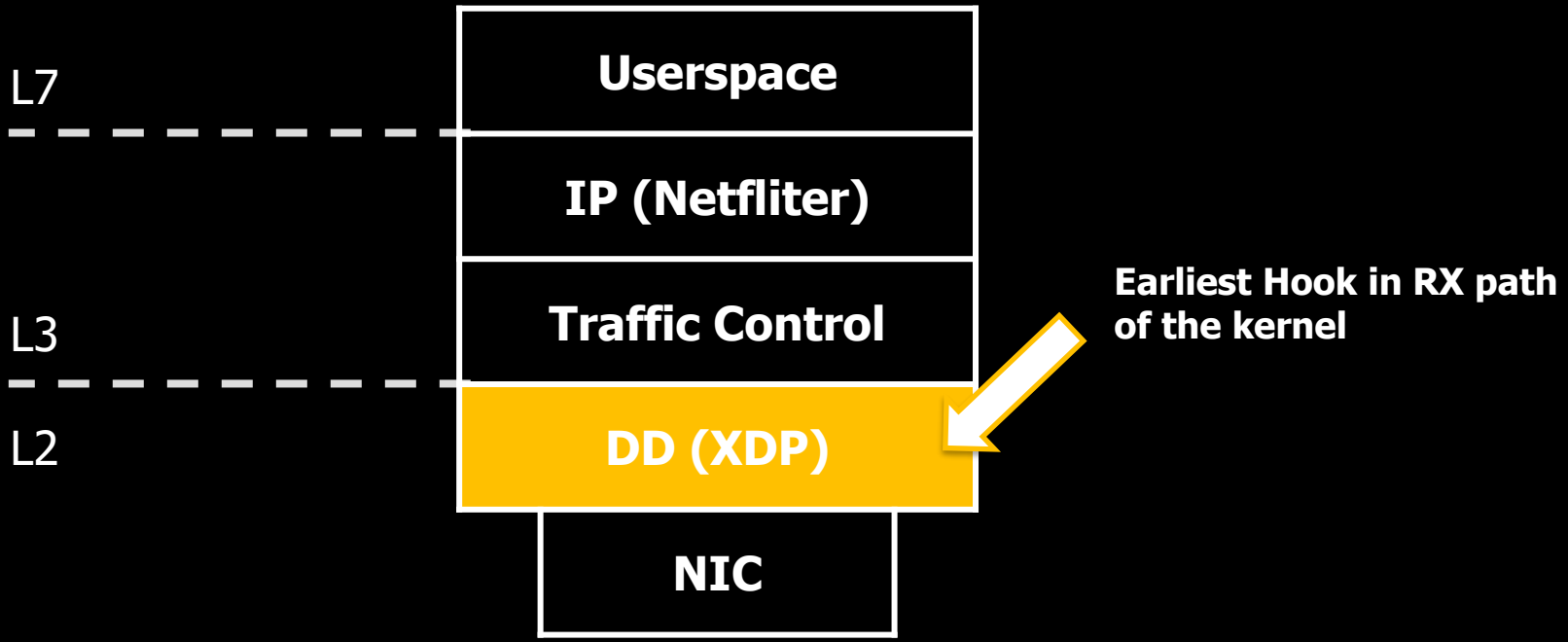
What is XDP?

eBPF based fast data-path

What is XDP?



What is XDP?


L7
L3
L2
Userspace
IP (Netfliter)
Traffic Control
DD (XDP)
NIC
Earliest Hook in RX path of the kernel

XDP Call Path

```
__do_softirq() {
 net_rx_action() {
  ixgbe_poll() {
   ixgbe_clean_rx_irq() {
    ixgbe_get_rx_buffer();

    ixgbe_run_xdp() {
      bpf_prog_run_xdp();
    }

    ixgbe_build_skb();
    ixgbe_rx_skb() {
     napi_gro_receive() {
      netif_receive_skb_internal();
```

Intel Device Driver
RX function Call stack

XDP Call Path

```
__do_softirq() {
 net_rx_action() {
  ixgbe_poll() {
   ixgbe_clean_rx_irq() {
    ixgbe_get_rx_buffer();
    ixgbe_run_xdp() {
      bpf_prog_run_xdp();
    }
    ixgbe_build_skb();
    ixgbe_rx_skb() {
     napi_gro_receive() {
      netif_receive_skb_internal();
```

Right After Interrupt Processing

Before any memory allocation (Expensive operation)

XDP Call Path

```
__do_softirq() {
 net_rx_action() {
  ixgbe_poll() {
   ixgbe_clean_rx_irq() {
    ixgbe_get_rx_buffer();
    ixgbe_run_xdp() {
      bpf_prog_run_xdp();
    }
    ixgbe_build_skb();
    ixgbe_rx_skb() {
     napi_gro_receive() {
      netif_receive_skb_internal();
```

Right After
Interrupt Processing

Decide the fate of the packet

Before any memory allocation
(Expensive operation)

*** User written XDP program**

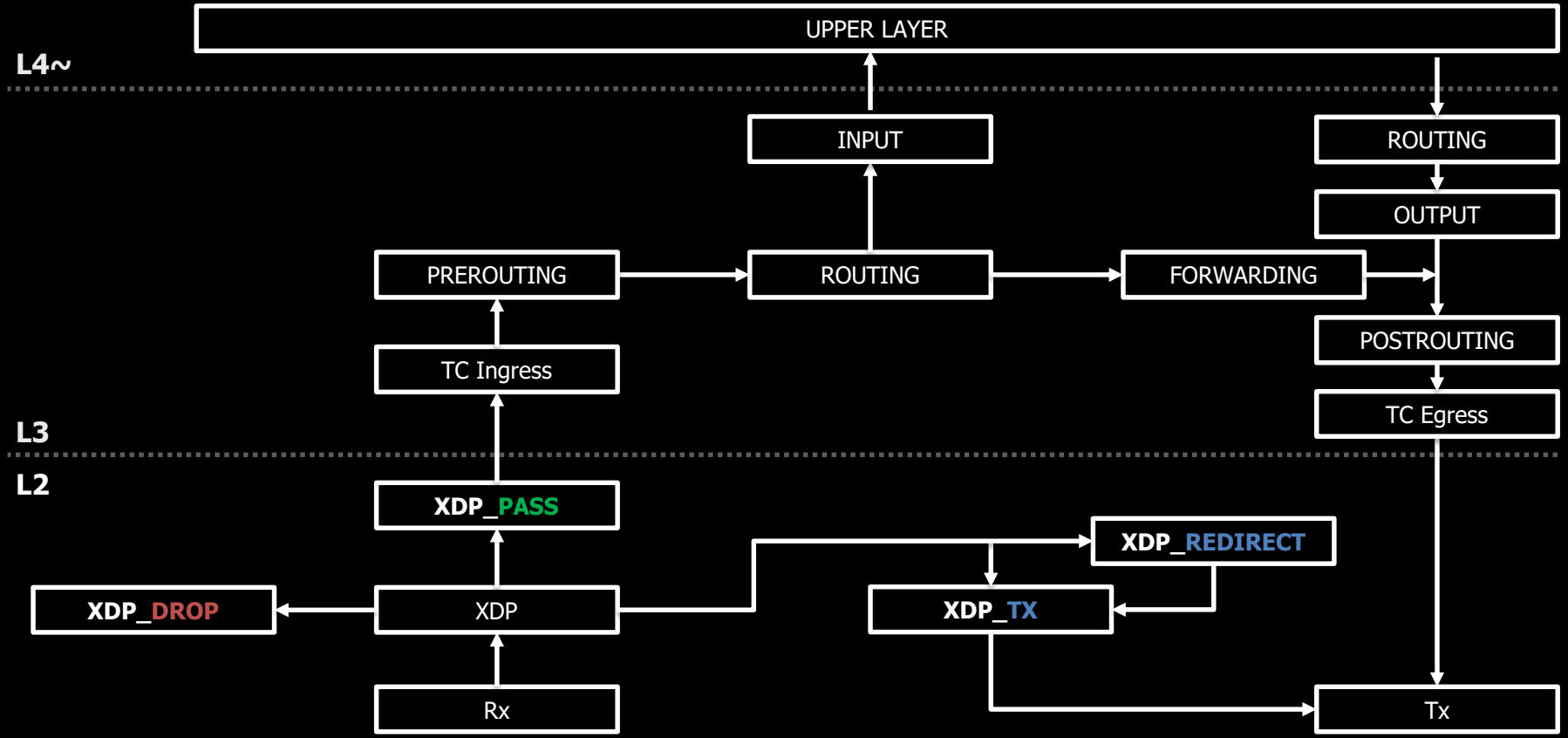
XDP Actions

XDP_DROP
XDP_ABORT
XDP_PASS
XDP_TX
XDP_REDIRECT

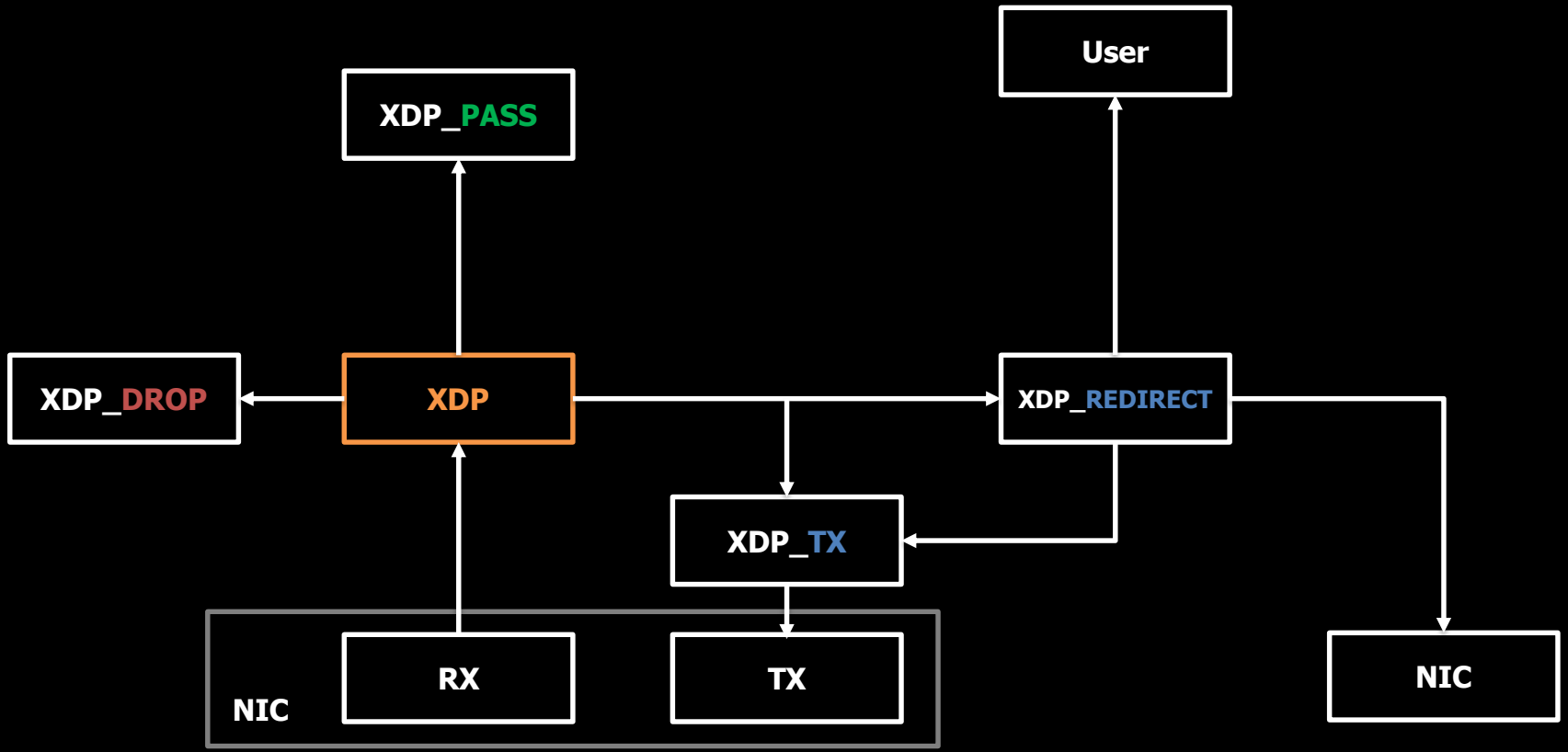
XDP Actions

XDP_DROP	- Very fast drop by recycling
XDP_ABORT	- Also drop, but with tracepoint
XDP_PASS	- Toss packet to network stack
XDP_TX	- Send packet back to same interface
XDP_REDIRECT	- Transmit out other NICs

XDP Actions


UPPER LAYER
L4~
INPUT
ROUTING
OUTPUT
PREROUTING
ROUTING
FORWARDING
POSTROUTING
TC Ingress
TC Egress
L3
L2
XDP_PASS
XDP_REDIRECT
XDP_DROP
XDP
XDP_TX
Rx
Tx

XDP Actions



xdp_buff

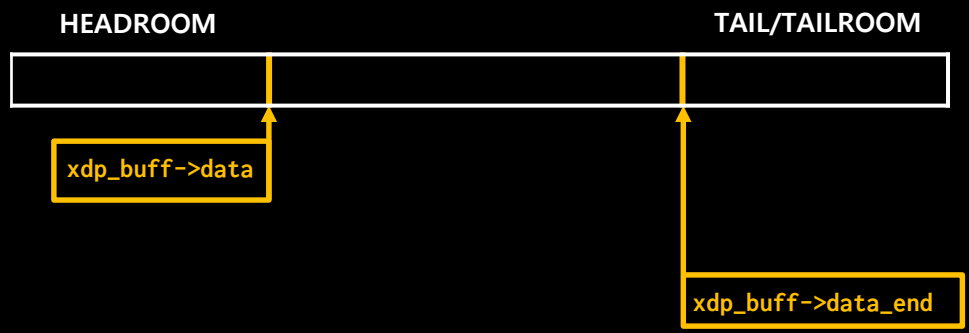
NO Memory Allocation?
NO sk_buff?

xdp_buff

```
struct xdp_buff {
        void *data;
        void *data_end;
        void *data_meta;
        void *data_hard_start;
        unsigned long handle;
        struct xdp_rxq_info *rxq;
};
```

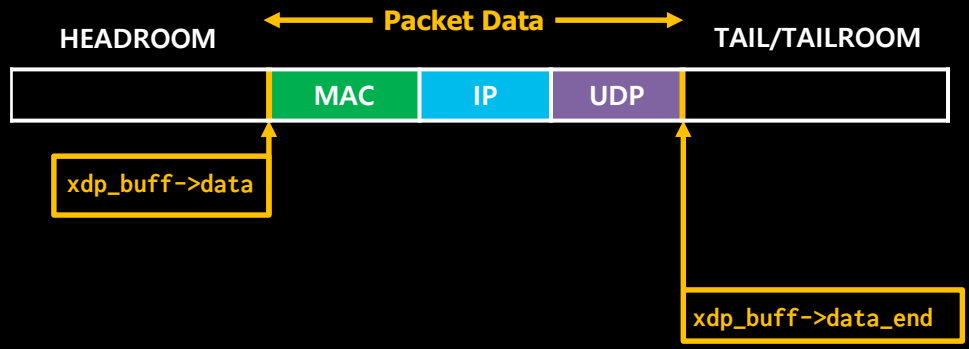
xdp_buff

```
struct xdp_buff {
        void *data;
        void *data_end;
        void *data_meta;
        void *data_hard_start;
        unsigned long handle;
        struct xdp_rxq_info *rxq;
};
```



xdp_buff

```
struct xdp_buff {
        void *data;
        void *data_end;
        void *data_meta;
        void *data_hard_start;
        unsigned long handle;
        struct xdp_rxq_info *rxq;
};
```



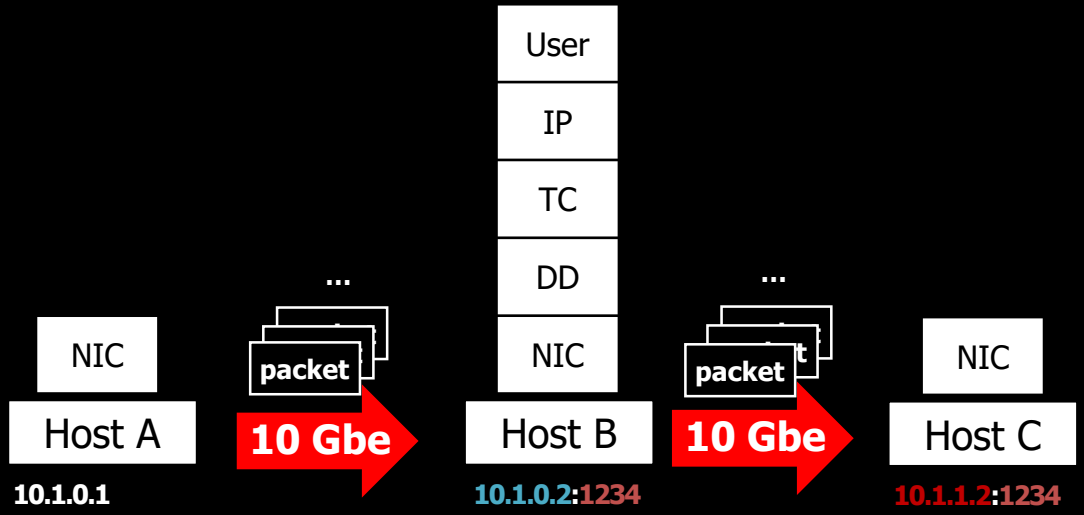
How to use XDP?

Packet Forward

With understanding XDP code

DEMO

Test environment



10.1.0.2 -> 10.1.1.2

1. **Destination NAT**
2. **Packet Forward**

```
1 "HOST_A"
HOST_A:~/soscon19_xdp
$
```

```
2 "HOST-B"
HOST_B:~/soscon19_xdp
$
```

```
3 "HOST_C"
HOST_C:~/soscon19_xdp
$
```

```
4 "HOST-B"
HOST_B:~/soscon19_xdp
$
```

XDP Packet Forward

```
SEC("xdp_fwd")
int xdp_fwd_prog(struct xdp_md *xdp) {

    void *data = (void *)(long)xdp->data;
    struct bpf_fib_lookup fib;
    struct ethhdr *eth = data;
    struct iphdr *iph;
    ...
    if (eth->h_proto == htons(ETH_P_IP)) {
        iph = data + sizeof(*eth);

        if (iph->daddr == _htonl(0xa010002))
            iph->daddr = _htonl(0xa010102);
    ...
```

10.1.0.2 -> 10.1.1.2

→ 1. Destination NAT

```
    rc = bpf_fib_lookup(xdp, &fib, sizeof(fib), 0);

    if (rc == BPF_FIB_LKUP_RET_SUCCESS) {
        memcpy(eth->h_dest, fib.dmac, ETH_ALEN);
        memcpy(eth->h_source, fib.smac, ETH_ALEN);
        return bpf_redirect(fib.ifindex, 0);
    }

    return XDP_PASS;
}
```

→ 2. Packet Forward

XDP Packet Forward

```
SEC("xdp_fwd")
int xdp_fwd_prog(struct xdp_md *xdp) {
    void *data_end = (void *)(long)xdp->data_end;
    void *data = (void *)(long)xdp->data;
    struct ethhdr *eth = data;
    struct iphdr *iph;
    struct udphdr *uh;
…
    if (eth + 1 > data_end)
        return XDP_DROP;

    if (eth->h_proto == htons(ETH_P_IP)) {
        iph = data + sizeof(*eth);
        if (iph + 1 > data_end)
            return XDP_DROP;

        if (iph->daddr == _htonl(0xa010002))
            if (iph->protocol == IPPROTO_UDP) {
                uh = data + sizeof(*eth) + sizeof(*iph);
                if (uh + 1 > data_end)
                    return XDP_DROP;

                if (uh->dest == htons(1234))
                    iph->daddr = _htonl(0xa010102);
            }
…
```

XDP Packet Forward

```
SEC("xdp_fwd")
 -> ELF Section where XDP program will be located
    void *data = (void *)(long)xdp->data;
    struct ethhdr *eth = data;
    struct iphdr *iph;
    struct udphdr *uh;
…
    if (eth + 1 > data_end)
        return XDP_DROP;

    if (eth->h_proto == htons(ETH_P_IP)) {
        iph = data + sizeof(*eth);
        if (iph + 1 > data_end)
            return XDP_DROP;

        if (iph->daddr == _htonl(0xa010002))
            if (iph->protocol == IPPROTO_UDP) {
                uh = data + sizeof(*eth) + sizeof(*iph);
                if (uh + 1 > data_end)
                    return XDP_DROP;

                if (uh->dest == htons(1234))
                    iph->daddr = _htonl(0xa010102);
            }
…
```

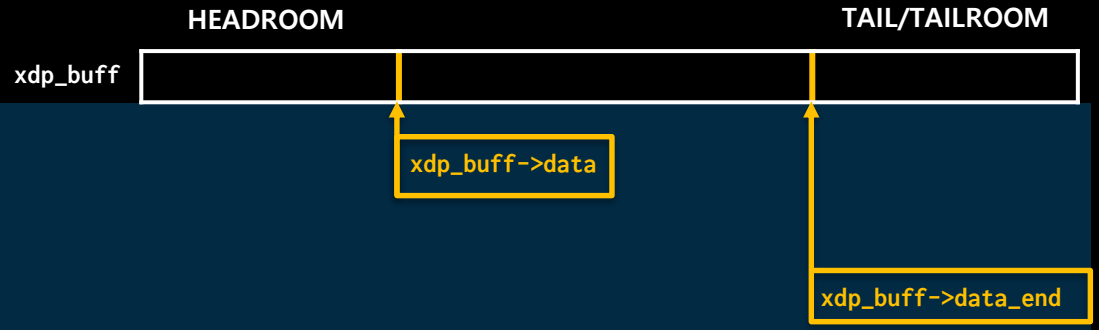
XDP Packet Forward

```
SEC("xdp_fwd")
int xdp_fwd_prog(struct xdp_md *xdp) {
```

-> Name of the XDP program

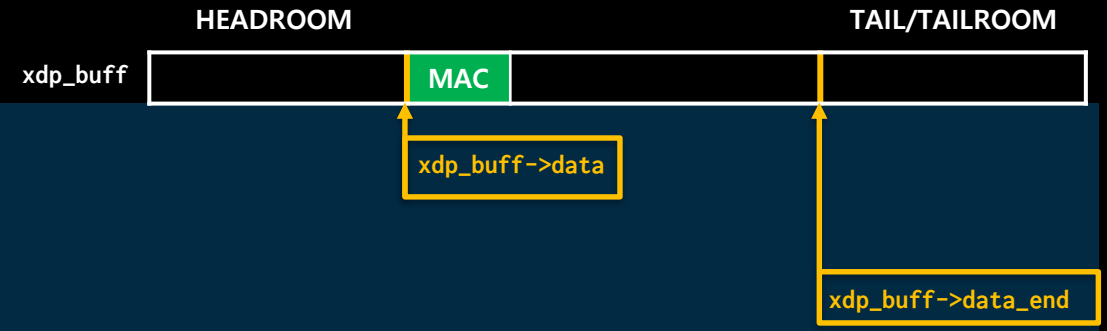
```
    struct ethhdr *eth = data;
    struct iphdr *iph;
    struct udphdr *uh;
…
    if (eth + 1 > data_end)
        return XDP_DROP;

    if (eth->h_proto == htons(ETH_P_IP)) {
        iph = data + sizeof(*eth);
        if (iph + 1 > data_end)
            return XDP_DROP;

        if (iph->daddr == _htonl(0xa010002))
            if (iph->protocol == IPPROTO_UDP) {
                uh = data + sizeof(*eth) + sizeof(*iph);
                if (uh + 1 > data_end)
                    return XDP_DROP;

                if (uh->dest == htons(1234))
                    iph->daddr = _htonl(0xa010102);
            }
…
```

XDP Packet Forward



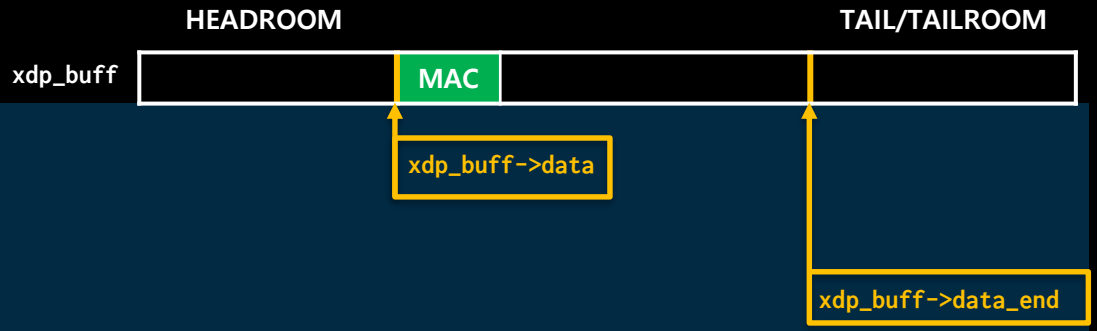

```
SEC("xdp_fwd")
int xdp_fwd_prog(struct xdp_md *xdp) {
    void *data_end = (void *)(long)xdp->data_end;
    void *data = (void *)(long)xdp->data;
    struct ethhdr *eth = data;
    struct iphdr *iph;
    struct udphdr *uh;
…
    if (eth + 1 > data_end)
        return XDP_DROP;

    if (eth->h_proto == htons(ETH_P_IP)) {
        iph = data + sizeof(*eth);
        if (iph + 1 > data_end)
            return XDP_DROP;

        if (iph->daddr == _htonl(0xa010002))
            if (iph->protocol == IPPROTO_UDP) {
                uh = data + sizeof(*eth) + sizeof(*iph);
                if (uh + 1 > data_end)
                    return XDP_DROP;

                if (uh->dest == htons(1234))
                    iph->daddr = _htonl(0xa010102);
            }
…
```

XDP Packet Forward




```
SEC("xdp_fwd")
int xdp_fwd_prog(struct xdp_md *xdp) {
    void *data_end = (void *)(long)xdp->data_end;
    void *data = (void *)(long)xdp->data;
    struct ethhdr *eth = data;
                -> Cast to Ethernet header
…
    if (eth + 1 > data_end)
        return XDP_DROP;

    if (eth->h_proto == htons(ETH_P_IP)) {
        iph = data + sizeof(*eth);
        if (iph + 1 > data_end)
            return XDP_DROP;

        if (iph->daddr == _htonl(0xa010002))
            if (iph->protocol == IPPROTO_UDP) {
                uh = data + sizeof(*eth) + sizeof(*iph);
                if (uh + 1 > data_end)
                    return XDP_DROP;

                if (uh->dest == htons(1234))
                    iph->daddr = _htonl(0xa010102);
            }
…
```

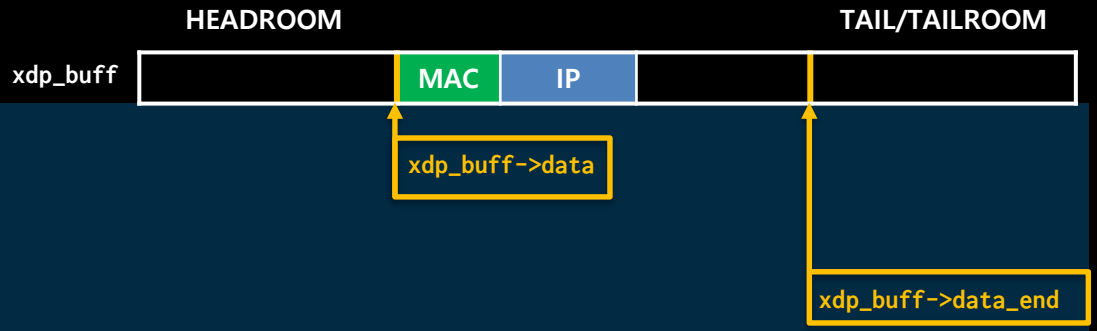
XDP Packet Forward




```
SEC("xdp_fwd")
int xdp_fwd_prog(struct xdp_md *xdp) {
    void *data_end = (void *)(long)xdp->data_end;
    void *data = (void *)(long)xdp->data;
    struct ethhdr *eth = data;
    struct iphdr *iph;
    struct udphdr *uh;
…
    if (eth + 1 > data_end)
        return XDP_DROP;

    if (eth->h_proto == htons(ETH_P_IP)) {
        iph = data + sizeof(*eth);
        if (iph + 1 > data_end)
            return XDP_DROP;

        if (iph->daddr == _htonl(0xa010002))
            if (iph->protocol == IPPROTO_UDP) {
                uh = data + sizeof(*eth) + sizeof(*iph);
                if (uh + 1 > data_end)
                    return XDP_DROP;

                if (uh->dest == htons(1234))
                    iph->daddr = _htonl(0xa010102);
            }
…
```

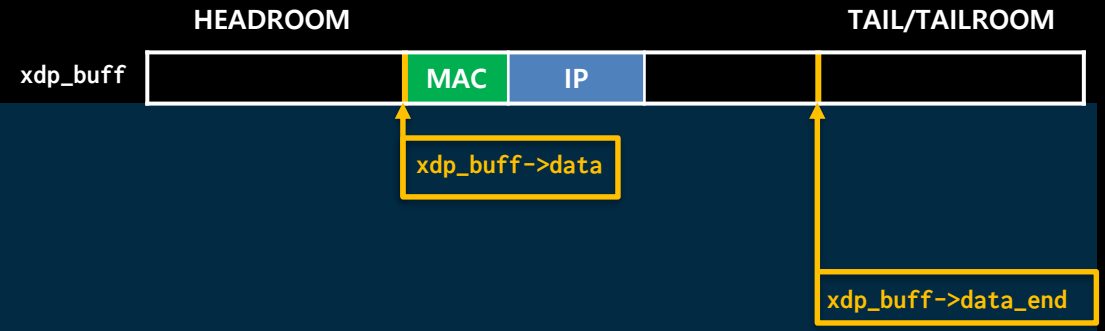
-> Validate the Ethernet header

XDP Packet Forward




```
SEC("xdp_fwd")
int xdp_fwd_prog(struct xdp_md *xdp) {
    void *data_end = (void *)(long)xdp->data_end;
    void *data = (void *)(long)xdp->data;
    struct ethhdr *eth = data;
    struct iphdr *iph;
    struct udphdr *uh;
…
    if (eth + 1 > data_end)
        return XDP_DROP;

    if (eth->h_proto == htons(ETH_P_IP)) { -> If eth->protocol is IP, Cast to IP header
        iph = data + sizeof(*eth);
        if (iph + 1 > data_end)
            return XDP_DROP;

        if (iph->daddr == _htonl(0xa010002))
            if (iph->protocol == IPPROTO_UDP) {
                uh = data + sizeof(*eth) + sizeof(*iph);
                if (uh + 1 > data_end)
                    return XDP_DROP;

                if (uh->dest == htons(1234))
                    iph->daddr = _htonl(0xa010102);
            }
…
```

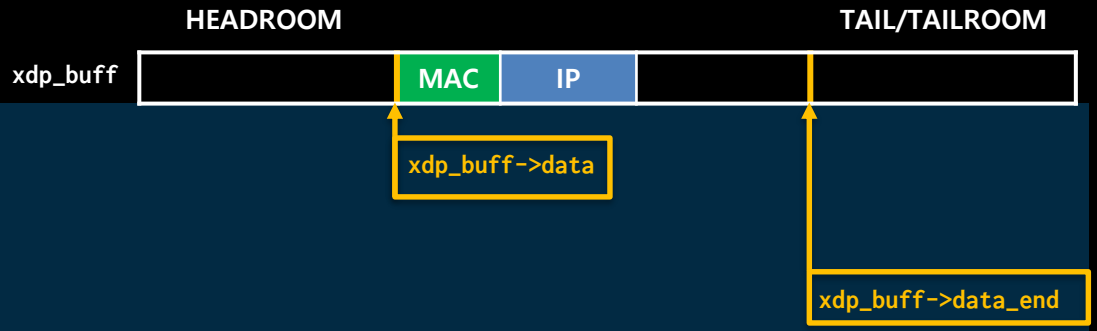
XDP Packet Forward




```
SEC("xdp_fwd")
int xdp_fwd_prog(struct xdp_md *xdp) {
    void *data_end = (void *)(long)xdp->data_end;
    void *data = (void *)(long)xdp->data;
    struct ethhdr *eth = data;
    struct iphdr *iph;
    struct udphdr *uh;
…
    if (eth + 1 > data_end)
        return XDP_DROP;

    if (eth->h_proto == htons(ETH_P_IP)) {
        iph = data + sizeof(*eth);
        if (iph + 1 > data_end)
            return XDP_DROP;

        if (iph->daddr == _htonl(0xa010002))
            if (iph->protocol == IPPROTO_UDP) {
                uh = data + sizeof(*eth) + sizeof(*iph);
                if (uh + 1 > data_end)
                    return XDP_DROP;

                if (uh->dest == htons(1234))
                    iph->daddr = _htonl(0xa010102);
            }
…
```

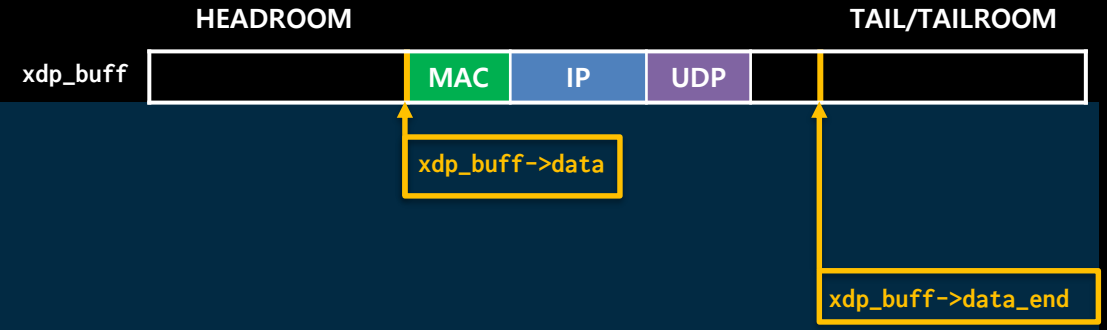
-> Validate the IP header

XDP Packet Forward




```
SEC("xdp_fwd")
int xdp_fwd_prog(struct xdp_md *xdp) {
    void *data_end = (void *)(long)xdp->data_end;
    void *data = (void *)(long)xdp->data;
    struct ethhdr *eth = data;
    struct iphdr *iph;
    struct udphdr *uh;
…
    if (eth + 1 > data_end)
        return XDP_DROP;

    if (eth->h_proto == htons(ETH_P_IP)) {
        iph = data + sizeof(*eth);
        if (iph + 1 > data_end)
            return XDP_DROP;

        if (iph->daddr == _htonl(0xa010002)) -> Check destination address 10.1.0.2
            if (iph->protocol == IPPROTO_UDP) {
                uh = data + sizeof(*eth) + sizeof(*iph);
                if (uh + 1 > data_end)
                    return XDP_DROP;

                if (uh->dest == htons(1234))
                    iph->daddr = _htonl(0xa010102);
            }
…
```

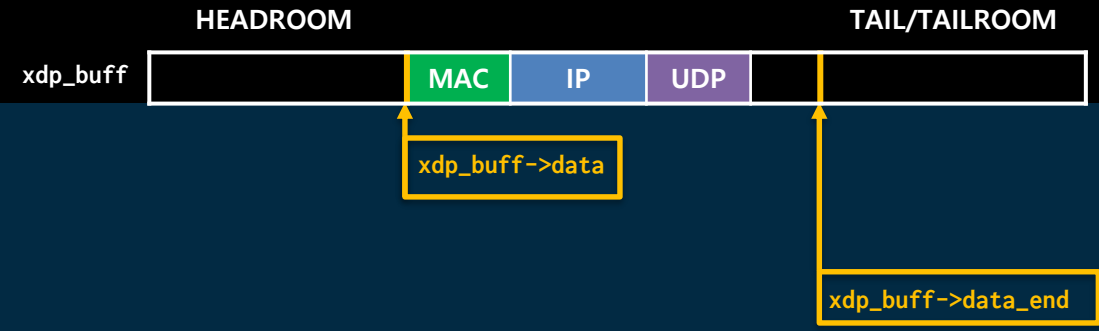
XDP Packet Forward




```
SEC("xdp_fwd")
int xdp_fwd_prog(struct xdp_md *xdp) {
    void *data_end = (void *)(long)xdp->data_end;
    void *data = (void *)(long)xdp->data;
    struct ethhdr *eth = data;
    struct iphdr *iph;
    struct udphdr *uh;
…
    if (eth + 1 > data_end)
        return XDP_DROP;

    if (eth->h_proto == htons(ETH_P_IP)) {
        iph = data + sizeof(*eth);
        if (iph + 1 > data_end)
            return XDP_DROP;

        if (iph->daddr == _htonl(0xa010002))
            if (iph->protocol == IPPROTO_UDP) {
                uh = data + sizeof(*eth) + sizeof(*iph);
                if (uh + 1 > data_end)
                    return XDP_DROP;

                if (uh->dest == htons(1234))
                    iph->daddr = _htonl(0xa010102);
            }
…
```

-> If iph->protocol is UDP, Cast to UDP header

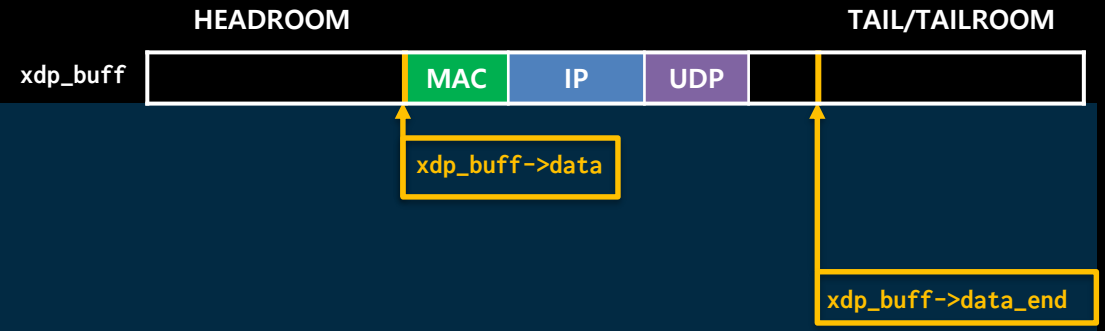
XDP Packet Forward




```
SEC("xdp_fwd")
int xdp_fwd_prog(struct xdp_md *xdp) {
    void *data_end = (void *)(long)xdp->data_end;
    void *data = (void *)(long)xdp->data;
    struct ethhdr *eth = data;
    struct iphdr *iph;
    struct udphdr *uh;
…
    if (eth + 1 > data_end)
        return XDP_DROP;

    if (eth->h_proto == htons(ETH_P_IP)) {
        iph = data + sizeof(*eth);
        if (iph + 1 > data_end)
            return XDP_DROP;

        if (iph->daddr == _htonl(0xa010002))
            if (iph->protocol == IPPROTO_UDP) {
                uh = data + sizeof(*eth) + sizeof(*iph);
                if (uh + 1 > data_end)
                    return XDP_DROP;

                if (uh->dest == htons(1234))
                    iph->daddr = _htonl(0xa010102);
            }
…
```

-> Validate the UDP header

XDP Packet Forward




```
SEC("xdp_fwd")
int xdp_fwd_prog(struct xdp_md *xdp) {
    void *data_end = (void *)(long)xdp->data_end;
    void *data = (void *)(long)xdp->data;
    struct ethhdr *eth = data;
    struct iphdr *iph;
    struct udphdr *uh;
…
    if (eth + 1 > data_end)
        return XDP_DROP;

    if (eth->h_proto == htons(ETH_P_IP)) {
        iph = data + sizeof(*eth);
        if (iph + 1 > data_end)
            return XDP_DROP;

        if (iph->daddr == _htonl(0xa010002))
            if (iph->protocol == IPPROTO_UDP) {
                uh = data + sizeof(*eth) + sizeof(*iph);
                if (uh + 1 > data_end)
                    return XDP_DROP;

                if (uh->dest == htons(1234))
                    iph->daddr = _htonl(0xa010102);
            }
…
```

-> if dst port is 1234,
Change dst address to 10.1.1.2

XDP Packet Forward

```
    struct bpf_fib_lookup fib;
    int rc;
...
    __builtin_memset(&fib, 0, sizeof(fib));  -> Prior to routing table lookup,
        fib.family    = AF_INET;                prepare bpf_fib_lookup struct
        fib.tos       = iph->tos;
        fib.l4_protocol = iph->protocol;
        fib.tot_len   = ntohs(iph->tot_len);
        fib.ipv4_src  = iph->saddr;
        fib.ipv4_dst  = iph->daddr;
        fib.ifindex   = xdp->ingress_ifindex;
    } else {
        return XDP_PASS;
    }

    rc = bpf_fib_lookup(xdp, &fib, sizeof(fib), 0);

    if (rc == BPF_FIB_LKUP_RET_SUCCESS) {
        ip_decrease_ttl(iph);                    /* from include/net/ip.h */
        memcpy(eth->h_dest, fib.dmac, ETH_ALEN);
        memcpy(eth->h_source, fib.smac, ETH_ALEN);
        return bpf_redirect(fib.ifindex, 0);
    }

    return XDP_PASS;
}
```

XDP Packet Forward

```
    struct bpf_fib_lookup fib;
    int rc;
...
        __builtin_memset(&fib, 0, sizeof(fib));
        fib.family    = AF_INET;
        fib.tos       = iph->tos;
        fib.l4_protocol = iph->protocol;
        fib.tot_len   = ntohs(iph->tot_len);
        fib.ipv4_src  = iph->saddr;
        fib.ipv4_dst  = iph->daddr;
        fib.ifindex   = xdp->ingress_ifindex;
    } else {
        return XDP_PASS;
    }

    rc = bpf_fib_lookup(xdp, &fib, sizeof(fib), 0);

    if (rc == BPF_FIB_LKUP_RET_SUCCESS) {
        ip_decrease_ttl(iph);                    /* from include/net/ip.h */
        memcpy(eth->h_dest, fib.dmac, ETH_ALEN);
        memcpy(eth->h_source, fib.smac, ETH_ALEN);
        return bpf_redirect(fib.ifindex, 0);
    }

    return XDP_PASS;
}
```

-> Fill the struct for query
ex) src / dst address

XDP Packet Forward

```
    struct bpf_fib_lookup fib;
    int rc;
...
        __builtin_memset(&fib, 0, sizeof(fib));
        fib.family    = AF_INET;
        fib.tos       = iph->tos;
        fib.l4_protocol = iph->protocol;
        fib.tot_len   = ntohs(iph->tot_len);
        fib.ipv4_src  = iph->saddr;
        fib.ipv4_dst  = iph->daddr;
        fib.ifindex   = xdp->ingress_ifindex;    -> Make sure where packet comes from
    } else {
        return XDP_PASS;
    }

    rc = bpf_fib_lookup(xdp, &fib, sizeof(fib), 0);

    if (rc == BPF_FIB_LKUP_RET_SUCCESS) {
        ip_decrease_ttl(iph);                          /* from include/net/ip.h */
        memcpy(eth->h_dest, fib.dmac, ETH_ALEN);
        memcpy(eth->h_source, fib.smac, ETH_ALEN);
        return bpf_redirect(fib.ifindex, 0);
    }

    return XDP_PASS;
}
```

XDP Packet Forward

```
    struct bpf_fib_lookup fib;
    int rc;
...
        __builtin_memset(&fib, 0, sizeof(fib));
        fib.family    = AF_INET;
        fib.tos       = iph->tos;
        fib.l4_protocol = iph->protocol;
        fib.tot_len   = ntohs(iph->tot_len);
        fib.ipv4_src  = iph->saddr;
        fib.ipv4_dst  = iph->daddr;
        fib.ifindex   = xdp->ingress_ifindex;
    } else {
        return XDP_PASS;
    }

    rc = bpf_fib_lookup(xdp, &fib, sizeof(fib), 0); -> Query routing table for redirect interface

    if (rc == BPF_FIB_LKUP_RET_SUCCESS) {
        ip_decrease_ttl(iph);                          /* from include/net/ip.h */
        memcpy(eth->h_dest, fib.dmac, ETH_ALEN);
        memcpy(eth->h_source, fib.smac, ETH_ALEN);
        return bpf_redirect(fib.ifindex, 0);
    }

    return XDP_PASS;
}
```

XDP Packet Forward

```
    struct bpf_fib_lookup fib;
    int rc;
...
        __builtin_memset(&fib, 0, sizeof(fib));
        fib.family    = AF_INET;
        fib.tos       = iph->tos;
        fib.l4_protocol = iph->protocol;
        fib.tot_len   = ntohs(iph->tot_len);
        fib.ipv4_src  = iph->saddr;
        fib.ipv4_dst  = iph->daddr;
        fib.ifindex   = xdp->ingress_ifindex;
    } else {
        return XDP_PASS;
    }

    rc = bpf_fib_lookup(xdp, &fib, sizeof(fib), 0); -> Query routing table for redirect interface

    if (rc == BPF_FIB_LKUP_RET_SUCCESS)
        ip_decrease_ttl(iph);
        memcpy(eth->h_dest, fib.dmac, E
        memcpy(eth->h_source, fib.smac,
        return bpf_redirect(fib.ifindex
    }

    return XDP_PASS;
}
```

```
$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags  Metric Ref  Use Iface
0.0.0.0         _gateway        0.0.0.0         UG     101    0      0   enp6s0
10.1.0.0        0.0.0.0         255.255.255.0   U      111    0      0   eth0
10.1.1.0        0.0.0.0         255.255.255.0   U      110    0      0   eth1
192.168.0.0     0.0.0.0         255.255.255.0   U      101    0      0   enp6s0
_gateway        0.0.0.0         255.255.255.255 UH     101    0      0   enp6s0
```

XDP Packet Forward

```
    struct bpf_fib_lookup fib;
    int rc;
...
        __builtin_memset(&fib, 0, sizeof(fib));
        fib.family    = AF_INET;
        fib.tos       = iph->tos;
        fib.l4_protocol = iph->protocol;
        fib.tot_len   = ntohs(iph->tot_len);
        fib.ipv4_src  = iph->saddr;
        fib.ipv4_dst  = iph->daddr;
        fib.ifindex   = xdp->ingress_ifindex;
    } else {
        return XDP_PASS;
    }

    rc = bpf_fib_lookup(xdp, &fib, sizeof(fib), 0);

    if (rc == BPF_FIB_LKUP_RET_SUCCESS) { -> If success, decrease ttl
        ip_decrease_ttl(iph);                          /* from include/net/ip.h */
        memcpy(eth->h_dest, fib.dmac, ETH_ALEN);
        memcpy(eth->h_source, fib.smac, ETH_ALEN);
        return bpf_redirect(fib.ifindex, 0);
    }

    return XDP_PASS;
}
```

XDP Packet Forward

```
    struct bpf_fib_lookup fib;
    int rc;
...
        __builtin_memset(&fib, 0, sizeof(fib));
        fib.family    = AF_INET;
        fib.tos       = iph->tos;
        fib.l4_protocol = iph->protocol;
        fib.tot_len   = ntohs(iph->tot_len);
        fib.ipv4_src  = iph->saddr;
        fib.ipv4_dst  = iph->daddr;
        fib.ifindex   = xdp->ingress_ifindex;
    } else {
        return XDP_PASS;
    }

    rc = bpf_fib_lookup(xdp, &fib, sizeof(fib), 0);

    if (rc == BPF_FIB_LKUP_RET_SUCCESS) {
        ip_decrease_ttl(iph);                          /* from include/net/ip.h */
        memcpy(eth->h_dest, fib.dmac, ETH_ALEN);
        memcpy(eth->h_source, fib.smac, ETH_ALEN);
        return bpf_redirect(fib.ifindex, 0);
    }

    return XDP_PASS;
}
```

-> Change Mac address for src, dst

XDP Packet Forward

```
    struct bpf_fib_lookup fib;
    int rc;
...
        __builtin_memset(&fib, 0, sizeof(fib));
        fib.family    = AF_INET;
        fib.tos       = iph->tos;
        fib.l4_protocol = iph->protocol;
        fib.tot_len   = ntohs(iph->tot_len);
        fib.ipv4_src  = iph->saddr;
        fib.ipv4_dst  = iph->daddr;
        fib.ifindex   = xdp->ingress_ifindex;
    } else {
        return XDP_PASS;
    }

    rc = bpf_fib_lookup(xdp, &fib, sizeof(fib), 0);

    if (rc == BPF_FIB_LKUP_RET_SUCCESS) {
        ip_decrease_ttl(iph);                    /* from include/net/ip.h */
        memcpy(eth->h_dest, fib.dmac, ETH_ALEN);
        memcpy(eth->h_source, fib.smac, ETH_ALEN);
        return bpf_redirect(fib.ifindex, 0);
    }

    return XDP_PASS;
}
```

-> Packet Forward with bpf_redirect!
(returns with XDP_REDIRECT)

XDP Packet Forward

```
$ clang -O2 -Wall -target bpf -c xdp-fwd.c -o xdp-fwd.o

$ bpftool prog load ./xdp-fwd.o /sys/fs/bpf/fwd
$ bpftool prog show
...
44: xdp  name xdp_fwd_prog  tag 1aa0135c2e55b38d  gpl
        loaded_at 2019-10-11T20:03:01+0900  uid 0
        xlated 760B  jited 442B  memlock 4096B

$ bpftool net attach xdp id 44 dev eth0
$ bpftool net
xdp:
eth0(8) driver id 44
```

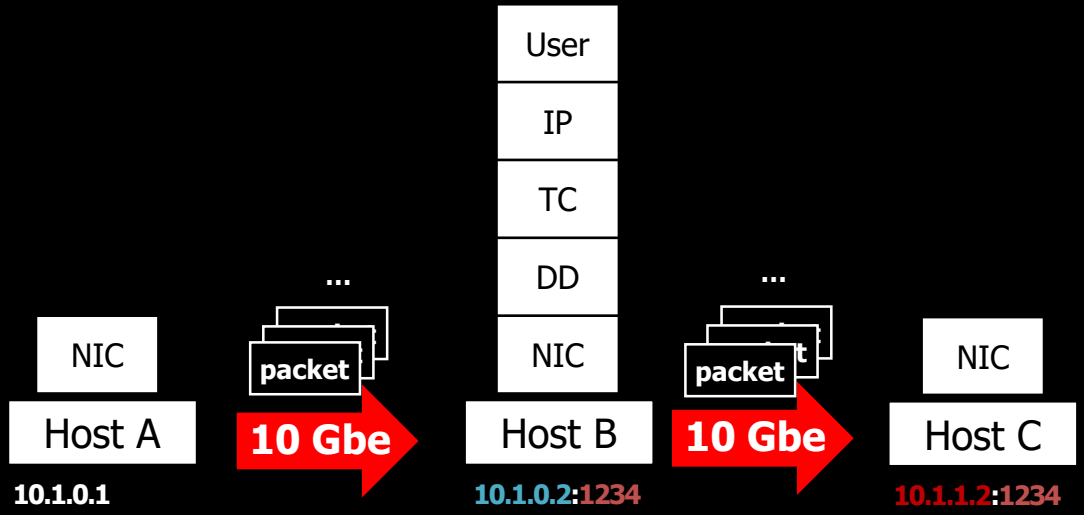
Compile with BPF target

Load program with bpftool

Attach to interface

Benchmark

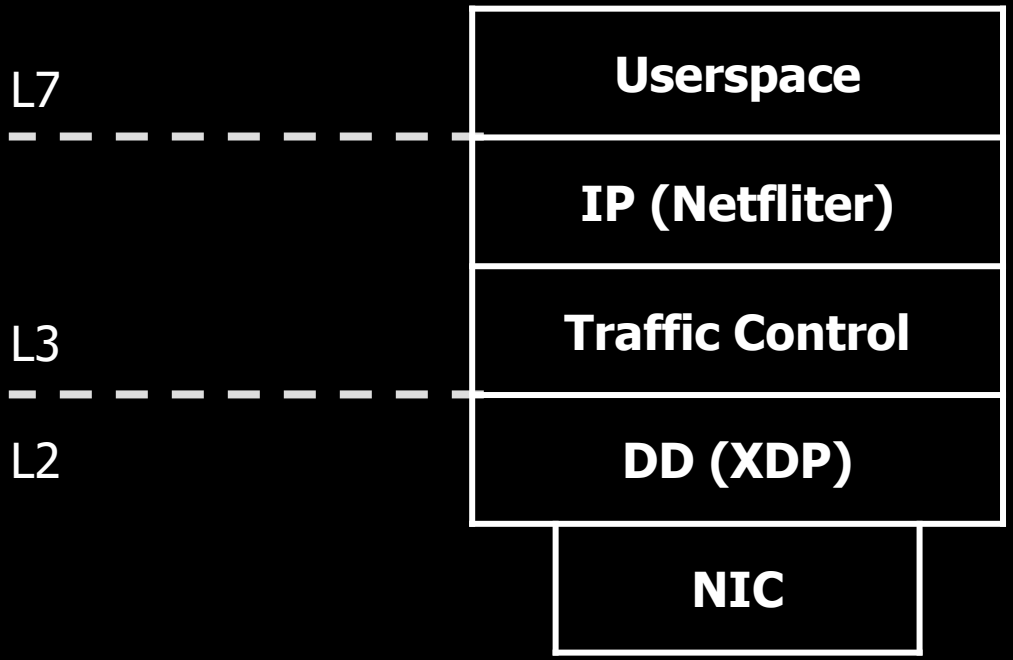
Test environment



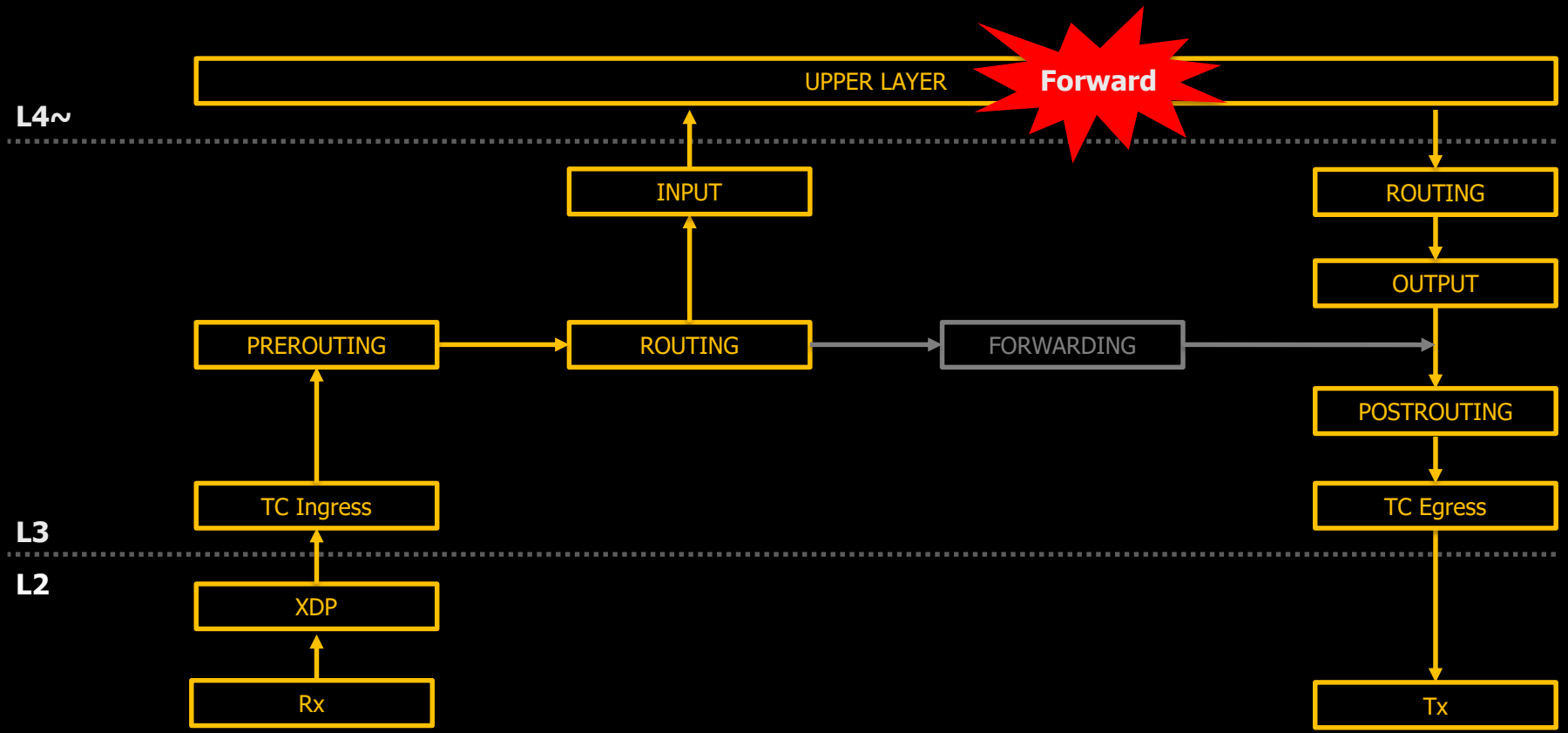
10.1.0.2 -> 10.1.1.2

1. **Destination NAT**
2. **Packet Forward**

Hooks to process packet?



Userspace Packet Forward

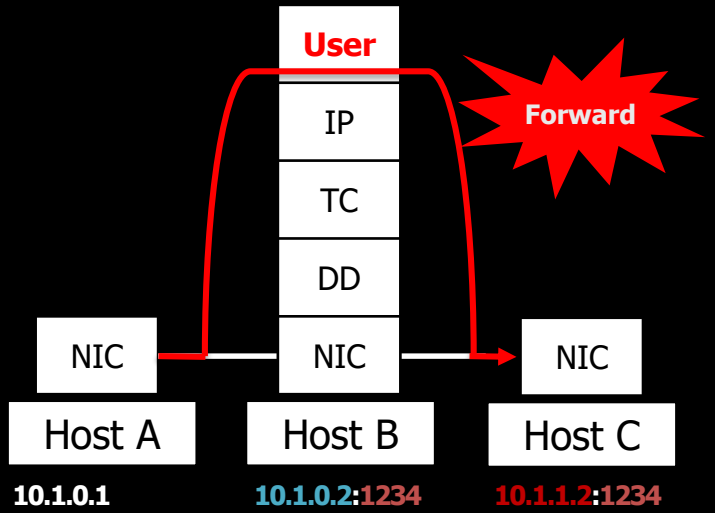


Userspace Packet Forward

UDP Load Balancer
packet forward with DNAT

```
stream {
  server {
    listen 1234 udp;
    proxy_pass udp_target;
  }

  upstream udp_target {
    server 10.1.1.2:1234;
  }

}
```

/etc/nginx/nginx.conf



Userspace Packet Forward

```
$ ethtool -S eth0 | grep rx_xdp_drop
     rx_xdp_drop: 370841
     rx_xdp_drop: 754718
     rx_xdp_drop: 1118515
     rx_xdp_drop: 1483198
     rx_xdp_drop: 1858322
     rx_xdp_drop: 2237080
     rx_xdp_drop: 2609916
     rx_xdp_drop: 2985062
     rx_xdp_drop: 3342212
     rx_xdp_drop: 3725703
     rx_xdp_drop: 4105888
```

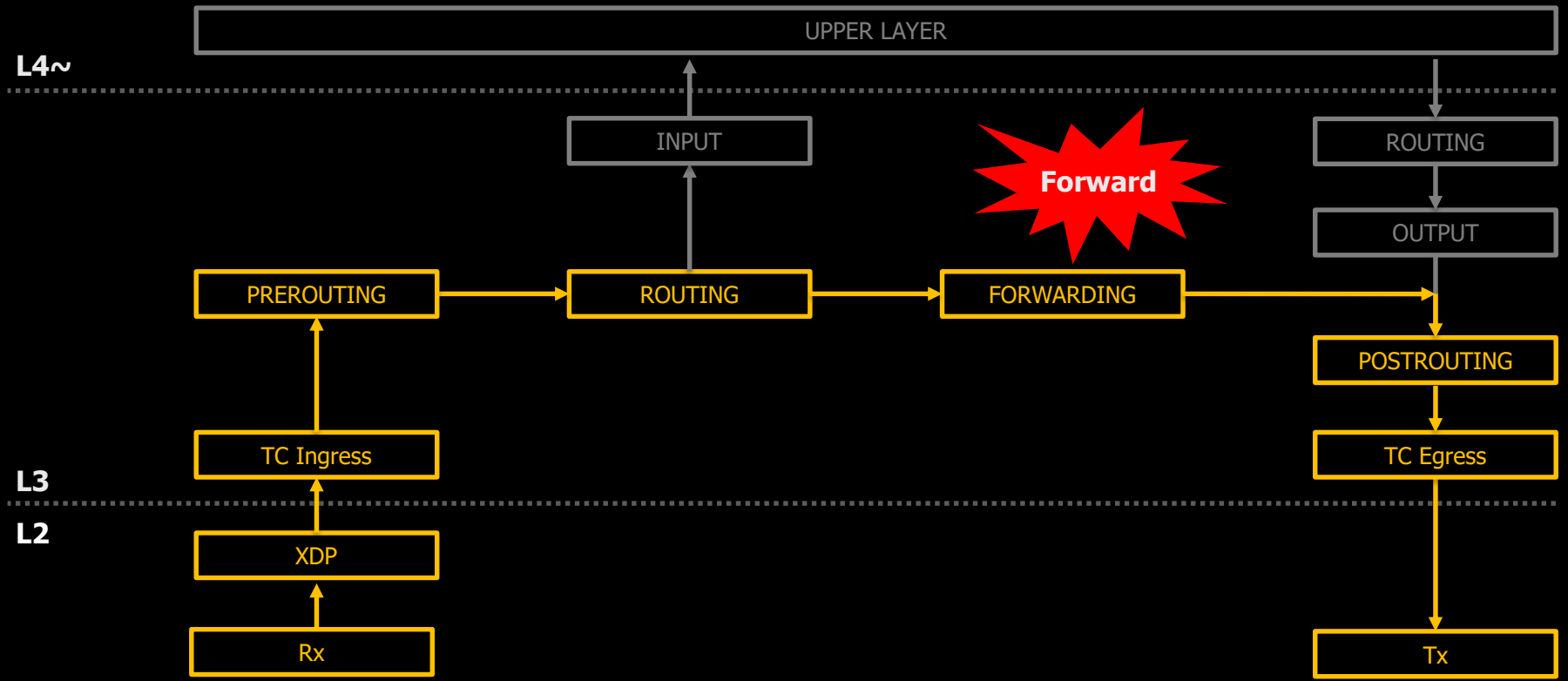
Average Packet Forward

373,263pps/core

≈ 260Mbit/s

How about Netfilter?

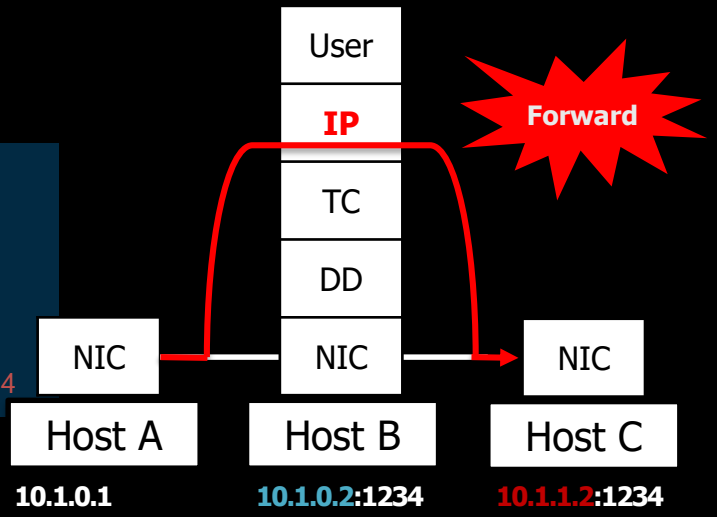
Netfilter Packet Forward



Netfilter Packet Forward

Netfilter (iptables)
packet forward with DNAT

```
$ iptables -t nat -A PREROUTING -d 10.1.0.2 -p udp --dport 1234 \
-j DNAT --to-destination 10.1.1.2:1234

$ iptables -t nat -L PREROUTING
Chain PREROUTING (policy ACCEPT)
target  prot opt source     destination
DNAT    udp  --  anywhere   10.1.0.2     udp dpt:1234 to:10.1.1.2:1234
```



Netfilter Packet Forward

```
$ ethtool -S eth0 | grep rx_xdp_drop
     rx_xdp_drop: 686991
     rx_xdp_drop: 1377146
     rx_xdp_drop: 2064571
     rx_xdp_drop: 2755620
     rx_xdp_drop: 3418211
     rx_xdp_drop: 4050284
     rx_xdp_drop: 4682626
     rx_xdp_drop: 5366274
     rx_xdp_drop: 6054439
     rx_xdp_drop: 6743962
     rx_xdp_drop: 7434054
     rx_xdp_drop: 8003538
     rx_xdp_drop: 8693454
```

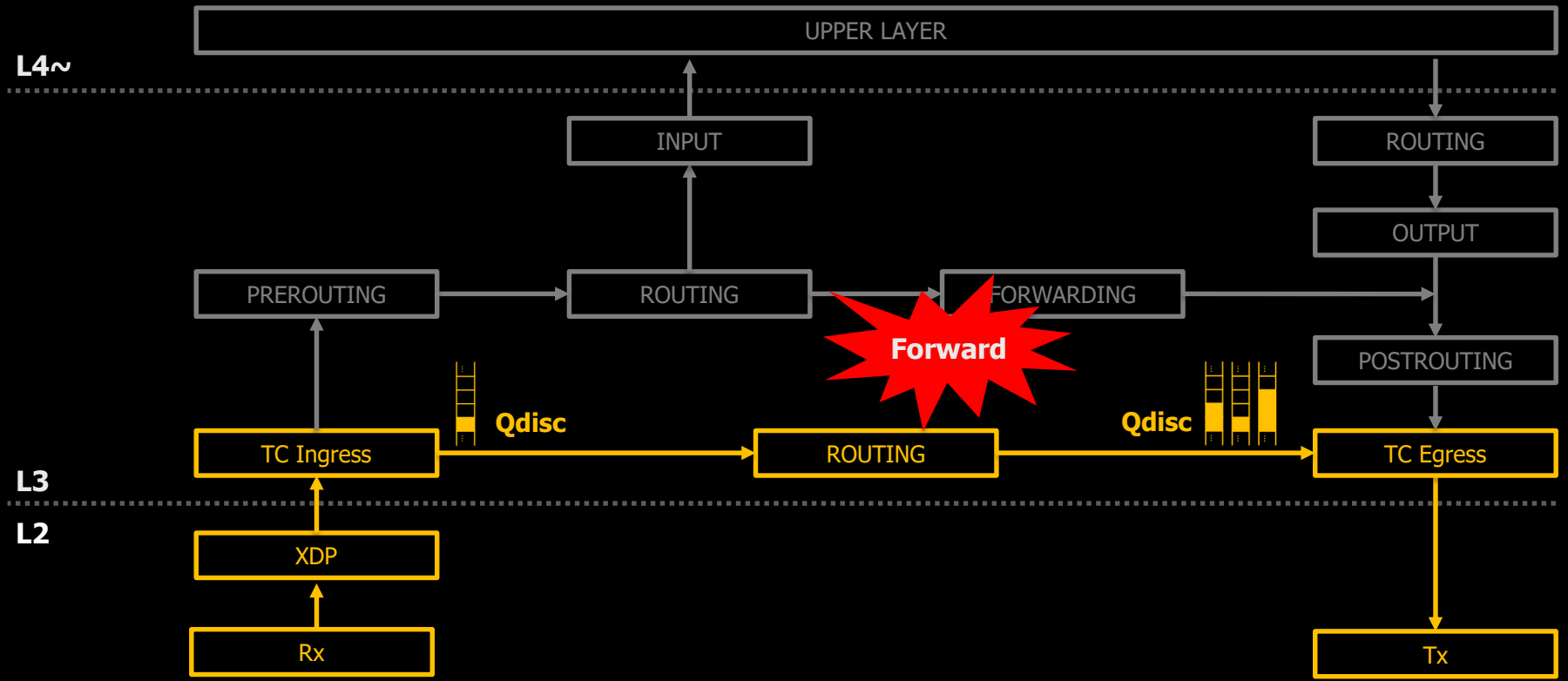
Average Packet Forward

668,728pps/core

≈ 960Mbit/s

What about TC?

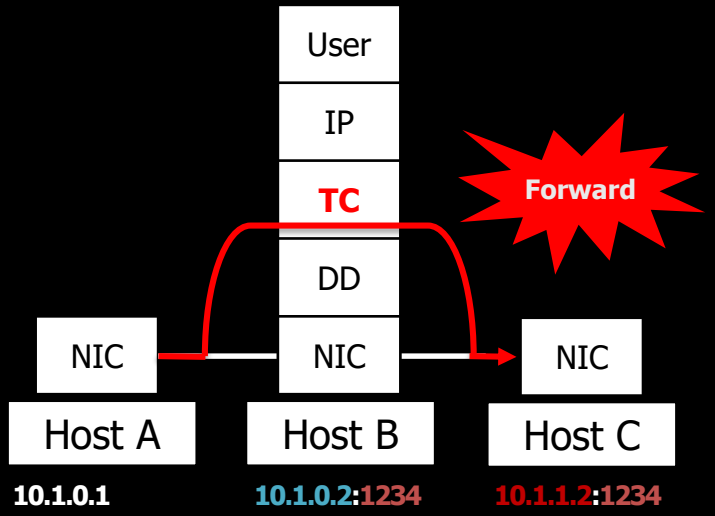
TC Ingress Packet Forward



TC Ingress Packet Forward

TC Ingress
packet forward with DNAT (redirect)

```
$ tc qdisc add dev eth0 ingress
$ tc filter add dev eth0 parent ffff: protocol ip u32 \
    match ip dst 10.1.0.2 match ip dport 1234 0xffff \
    action nat ingress 10.1.0.2/32 10.1.1.2/32 pipe \
    action mirred egress redirect dev eth1

$ tc filter show ingress dev eth0
...
  match 0a010002/ffffffff at 16
  match 000004d2/0000ffff at 20
        action order 1:  nat ingress 10.1.0.2/32 10.1.1.2 pipe
         index 1 ref 1 bind 1

        action order 2: mirred (Egress Redirect to eth1) stolen
        index 1 ref 1 bind 1
```



TC Ingress Packet Forward

```
$ ethtool -S eth0 | grep rx_xdp_drop
     rx_xdp_drop: 1407789
     rx_xdp_drop: 2848601
     rx_xdp_drop: 4274539
     rx_xdp_drop: 5695547
     rx_xdp_drop: 7132990
     rx_xdp_drop: 8533688
     rx_xdp_drop: 9943697
     rx_xdp_drop: 11367744
     rx_xdp_drop: 12808213
     rx_xdp_drop: 14234151
     rx_xdp_drop: 15685314
```

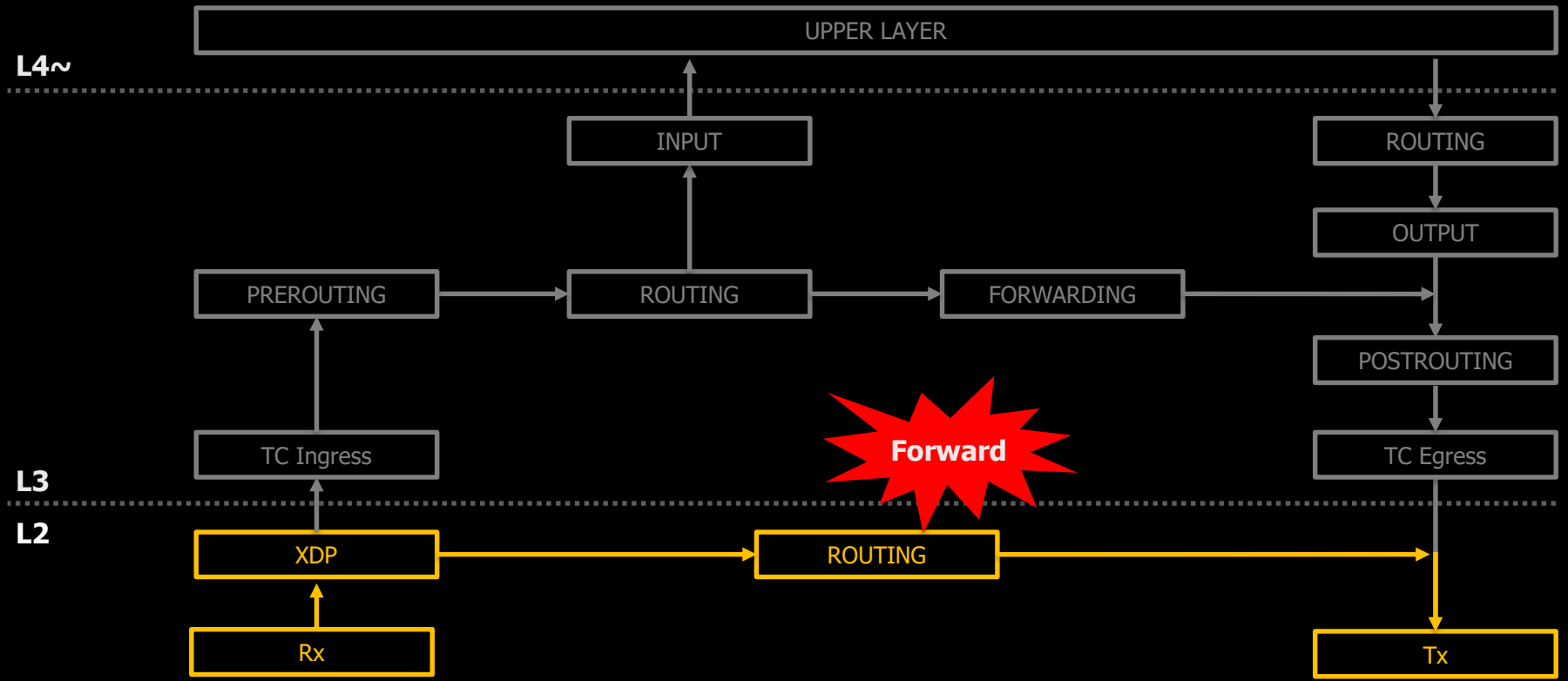
Average Packet Forward

1,425,938pps/core

≈ 1Gbit/s

And… XDP?

XDP Packet Forward



XDP Packet Forward

```
SEC("xdp_fwd")
int xdp_fwd_prog(struct xdp_md *xdp) {

    void *data = (void *)(long)xdp->data;
    struct bpf_fib_lookup fib;
    struct ethhdr *eth = data;
    struct iphdr *iph;
    ...
    if (eth->h_proto == htons(ETH_P_IP)) {
        iph = data + sizeof(*eth);

        if (iph->daddr == _htonl(0xa010002))
            iph->daddr = _htonl(0xa010102);
    ...
```

10.1.0.2 -> 10.1.1.2

1. Destination NAT

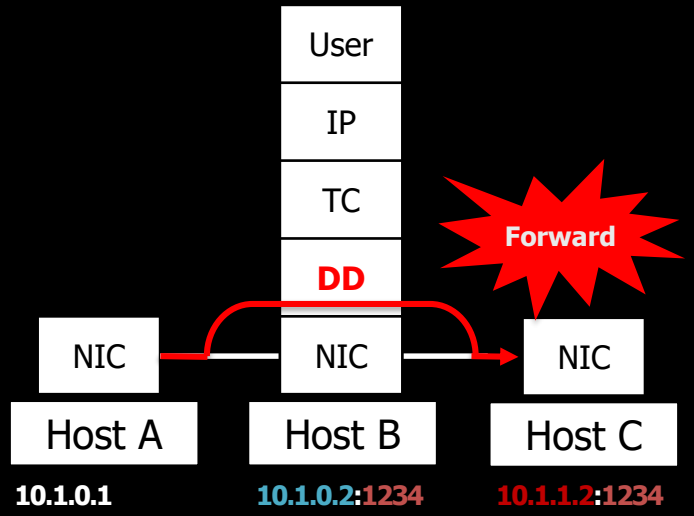
```
    rc = bpf_fib_lookup(xdp, &fib, sizeof(fib), 0);

    if (rc == BPF_FIB_LKUP_RET_SUCCESS) {
        memcpy(eth->h_dest, fib.dmac, ETH_ALEN);
        memcpy(eth->h_source, fib.smac, ETH_ALEN);
        return bpf_redirect(fib.ifindex, 0);
    }

    return XDP_PASS;
}
```

2. Packet Forward

XDP Packet Forward

XDP
packet forward with DNAT (XDP_FORWARD)

```
$ bpftool prog load ./xdp-fwd.o /sys/fs/bpf/fwd
$ bpftool prog show
...
44: xdp  name xdp_fwd_prog  tag 1aa0135c2e55b38d  gpl
        loaded_at 2019-10-11T20:03:01+0900  uid 0
        xlated 760B  jited 442B  memlock 4096B

$ bpftool net attach xdp id 44 dev eth0
$ bpftool net
xdp:
eth0(8) driver id 44
```



XDP Packet Forward

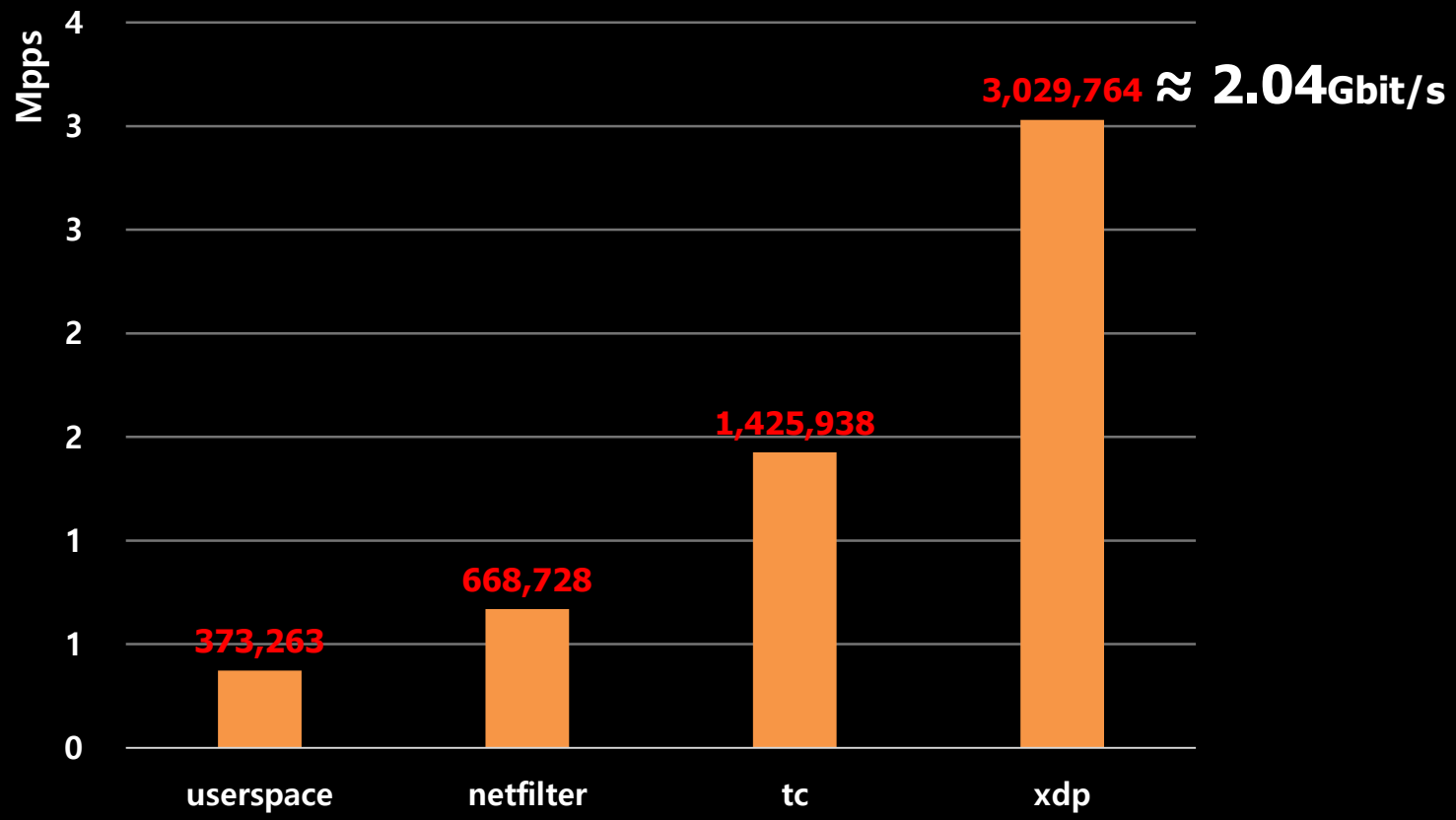
```
$ ethtool -S eth0 | grep rx_xdp_drop
     rx_xdp_drop: 3031344
     rx_xdp_drop: 6057680
     rx_xdp_drop: 9088916
     rx_xdp_drop: 12118560
     rx_xdp_drop: 15150319
     rx_xdp_drop: 18180559
     rx_xdp_drop: 21210383
     rx_xdp_drop: 24240271
     rx_xdp_drop: 27270543
     rx_xdp_drop: 30300559
     rx_xdp_drop: 33329483
     rx_xdp_drop: 36356788
```

Average Packet Forward

3,029,764pps/core

≈ 2.04Gbit/s

Packet Forward Results



More about XDP?

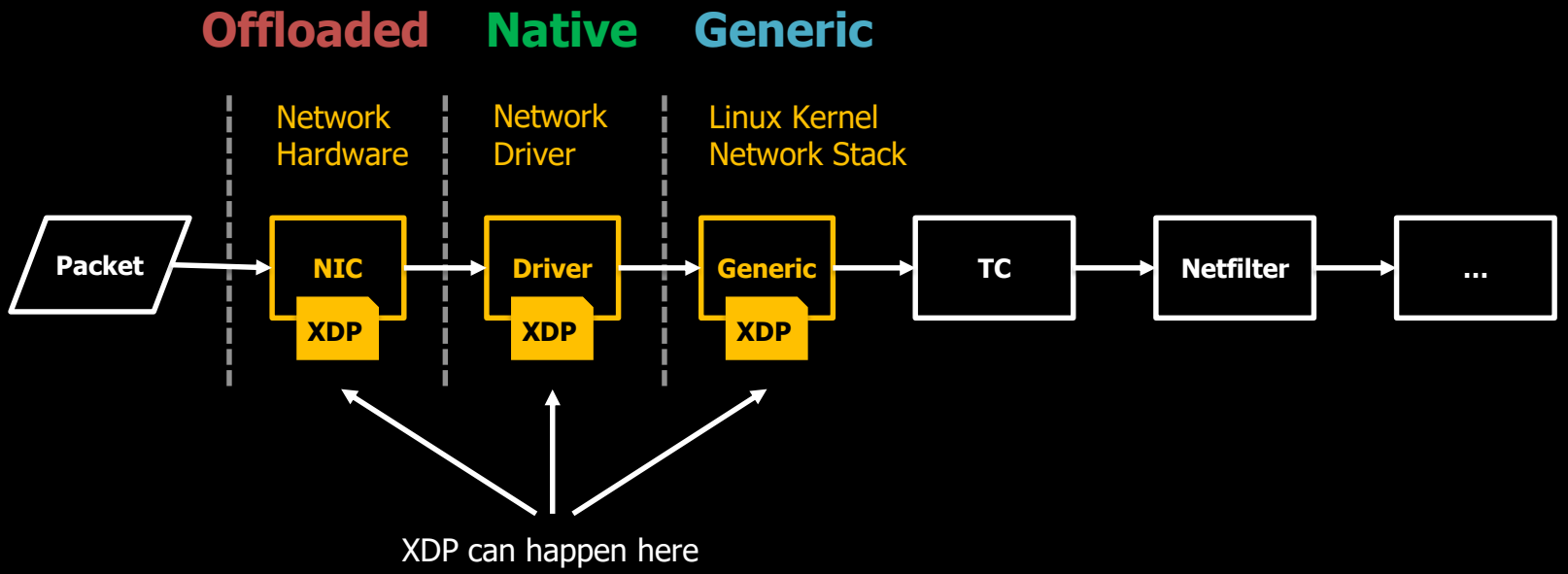
XDP Offload and Further usages

XDP Modes

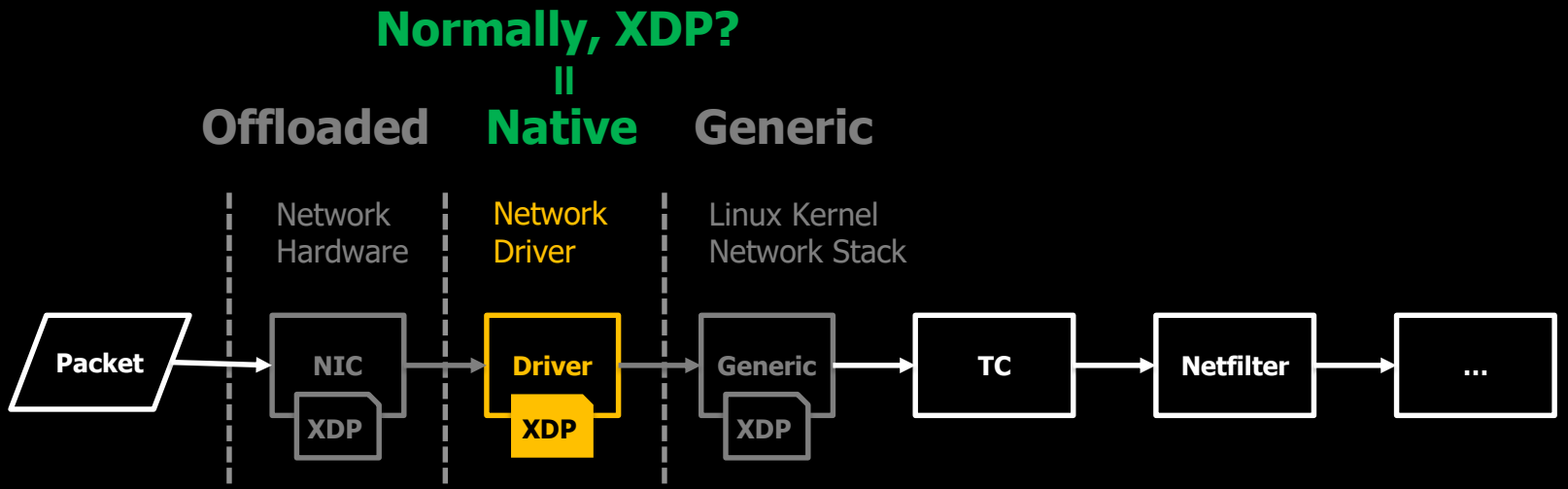
XDP Modes

Generic XDP
Native XDP
Offloaded XDP

XDP Modes


Offloaded
Native
Generic
Network Hardware
Network Driver
Linux Kernel Network Stack
Packet
NIC
XDP
Driver
XDP
Generic
XDP
TC
Netfilter
...
XDP can happen here

XDP Modes


Normally, XDP?
||
Offloaded
Native
Generic
Network Hardware
Network Driver
Linux Kernel Network Stack
Packet
NIC
XDP
Driver
XDP
Generic
XDP
TC
Netfilter
...

XDP Modes

Generic XDP

- **No driver support needed**

Native XDP

- **Intel** (ixgbe, ixgbevf, i40e)
- **Mellanox** (mlx4, mlx5)
- **Broadcom** (bnxt)
- **Qlogic** (qede)
- **Netronome** (nfp)
- **Others** (virtio, tun)

(Most of the 10Gbe Driver)

Offloaded XDP

- **Netronome** (nfp)

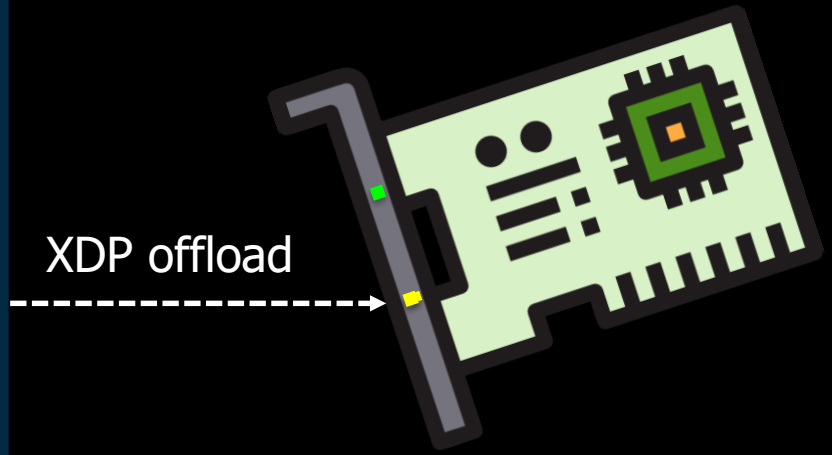
XDP OFFLOAD

What is XDP OFFLOAD?

BPF Program

```
 0: r0 = 1
 1: r2 = *(u32 *)(r1 + 4)
 2: r1 = *(u32 *)(r1 + 0)
 3: r3 = r1
 4: r3 += 14
 5: if r3 > r2 goto +18 <LBB0_8>
 6: r3 = *(u8 *)(r1 + 12)
 7: r4 = *(u8 *)(r1 + 13)
 8: r4 <<= 8
 9: r4 |= r3
10: if r4 != 8 goto +12 <LBB0_7>
11: r3 = r1
12: r3 += 34
13: if r3 > r2 goto +10 <LBB0_8>
14: r3 = *(u32 *)(r1 + 30)
15: if r3 != 33554698 goto +7 <LBB0_7>
16: r3 = *(u8 *)(r1 + 23)
17: if r3 != 17 goto +5 <LBB0_7>
18: r3 = r1
19: r3 += 42
...
```

XDP offload

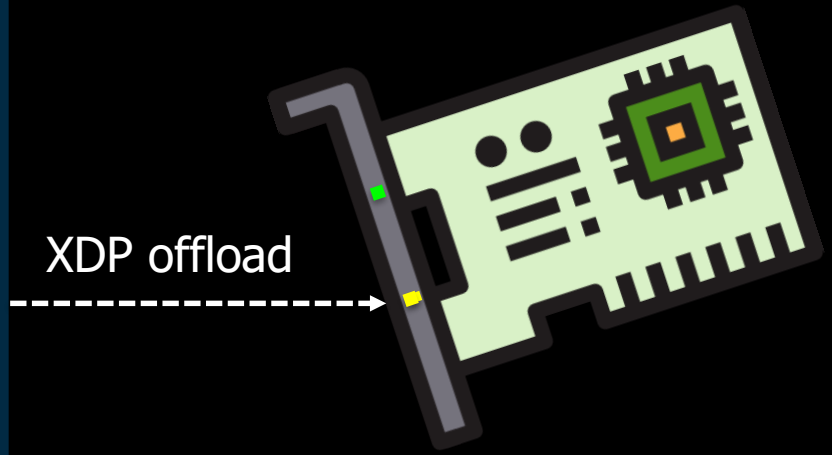


What is XDP OFFLOAD?

BPF Program

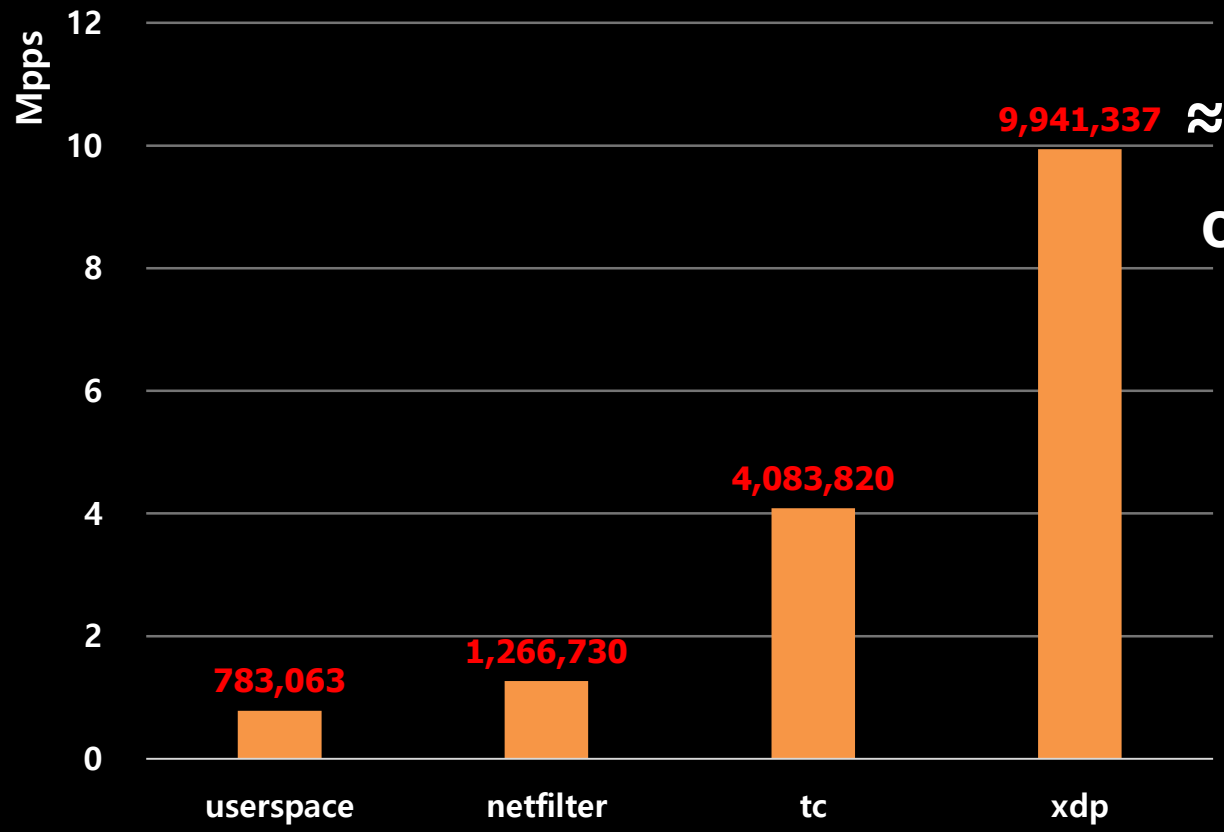
```
 0: r0 = 1
 1: r2 = *(u32 *)(r1 + 4)
 2: r1 = *(u32 *)(r1 + 0)
 3: r3 = r1
 4: r3 += 14
 5: if r3 > r2 goto +18 <LBB0_8>
 6: r3 = *(u8 *)(r1 + 12)
 7: r4 = *(u8 *)(r1 + 13)
 8: r4 <<= 8
 9: r4 |= r3
10: if r4 != 8 goto +12 <LBB0_7>
11: r3 = r1
12: r3 += 34
13: if r3 > r2 goto +10 <LBB0_8>
14: r3 = *(u32 *)(r1 + 30)
15: if r3 != 33554698 goto +7 <LBB0_7>
16: r3 = *(u8 *)(r1 + 23)
17: if r3 != 17 goto +5 <LBB0_7>
18: r3 = r1
19: r3 += 42
...
```

XDP offload



Runs more earlier
No CPU usage

Packet Drop Results



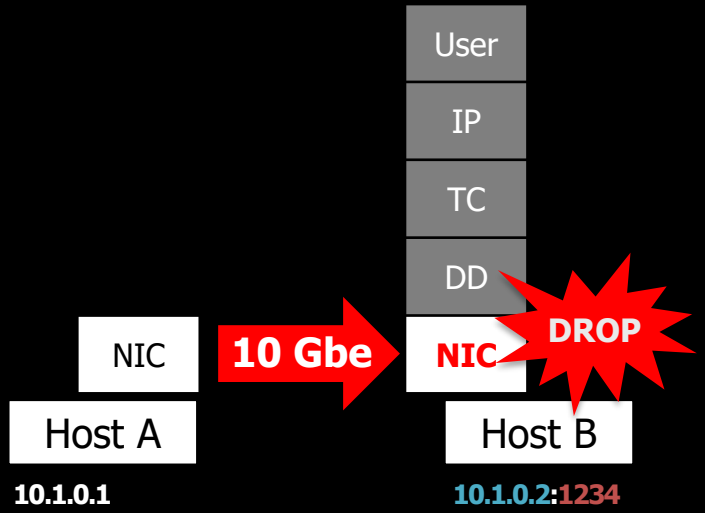
≈ 6.69Gbit/s

Out of 10Gbit/s

XDP Offload Packet Drop

XDP offload
packet drop with XDP_DROP

```
$ bpftool prog load ./xdp-drop.o /sys/fs/bpf/drop
$ bpftool prog show
...
24: xdp  name xdp_prog1  tag 6f8c2e06dfa2abcb  gpl
        loaded_at 2019-10-11T17:17:33+0900  uid 0
        xlated 544B  jited 344B  memlock 4096B  map_ids 17

$ bpftool net attach xdpoffload id 18 dev eth0
$ bpftool net
xdp:
eth0(8) offload id 18
```



XDP Offload Packet Drop

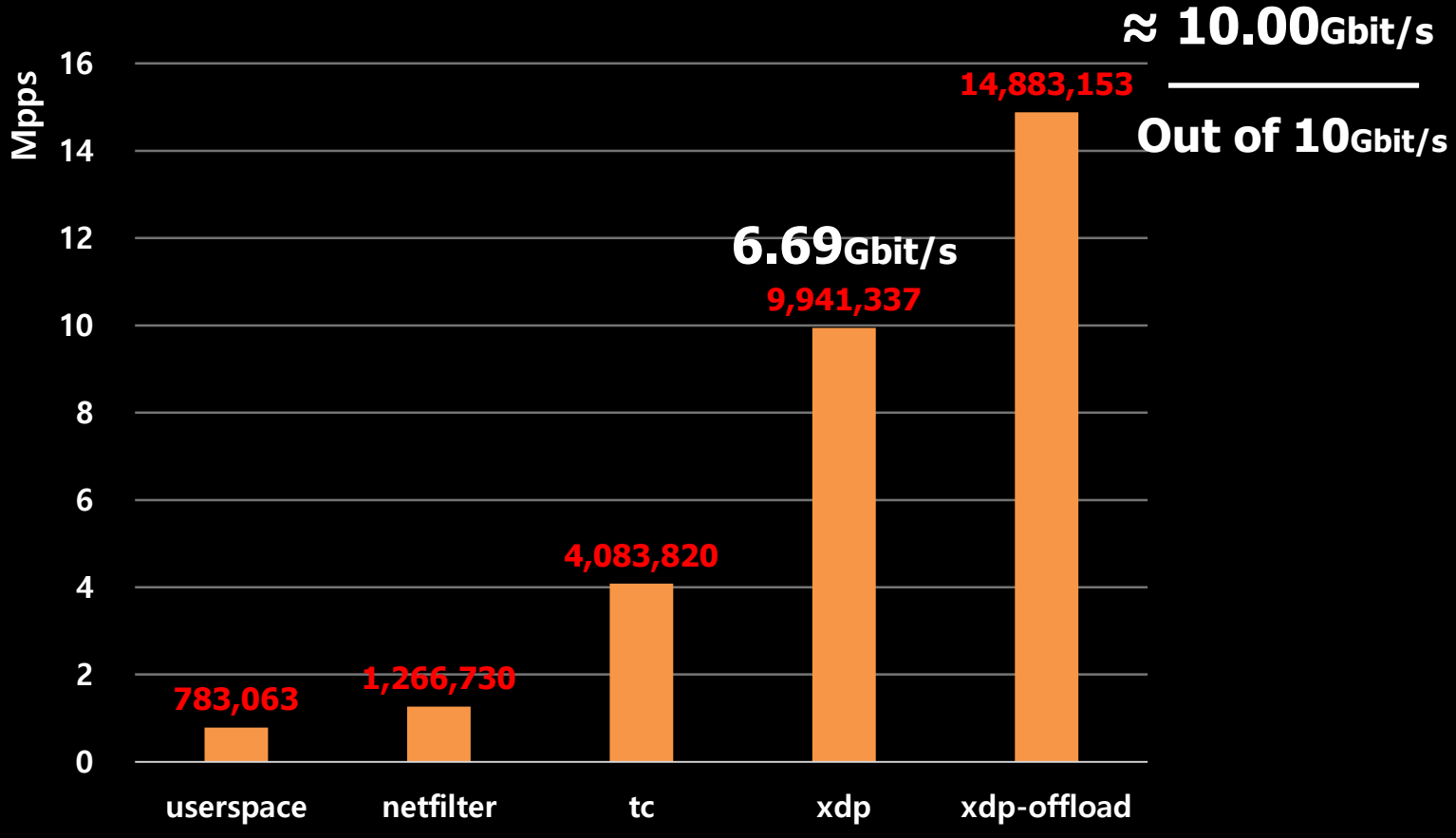
```
$ ethtool -S eth0 | grep bpf_app1_pkts

bpf_app1_pkts:  9340887
bpf_app1_pkts:  14274730
bpf_app1_pkts:  29233693
bpf_app1_pkts:  44165498
bpf_app1_pkts:  59097632
bpf_app1_pkts:  74057835
bpf_app1_pkts:  88990848
bpf_app1_pkts:  103947963
bpf_app1_pkts:  118881530
bpf_app1_pkts:  133838008
bpf_app1_pkts:  148770851
bpf_app1_pkts:  163705144
bpf_app1_pkts:  178664224
bpf_app1_pkts:  193596262
bpf_app1_pkts:  208554907
bpf_app1_pkts:  223488168
```

Average Packet Drop

14,883,153pps/core

≈ 10.00Gbit/s

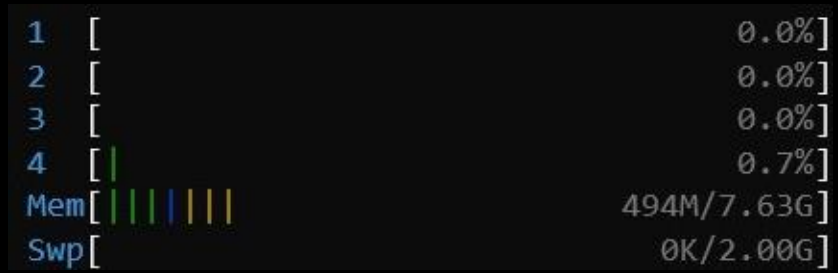
Packet Drop Results with XDP Offload



XDP Offload Packet Drop

XDP Offload Packet Drop

XDP



XDP Offload

XDP Use Cases

- Load Balancing
- Packet Tunneling (Encapsulation)
- DDoS attack mitigation
- Network monitoring
- ETC..

Sample code, test results can be found:

github.com/DanielTimLee/soscon19_XDP

THANK YOU