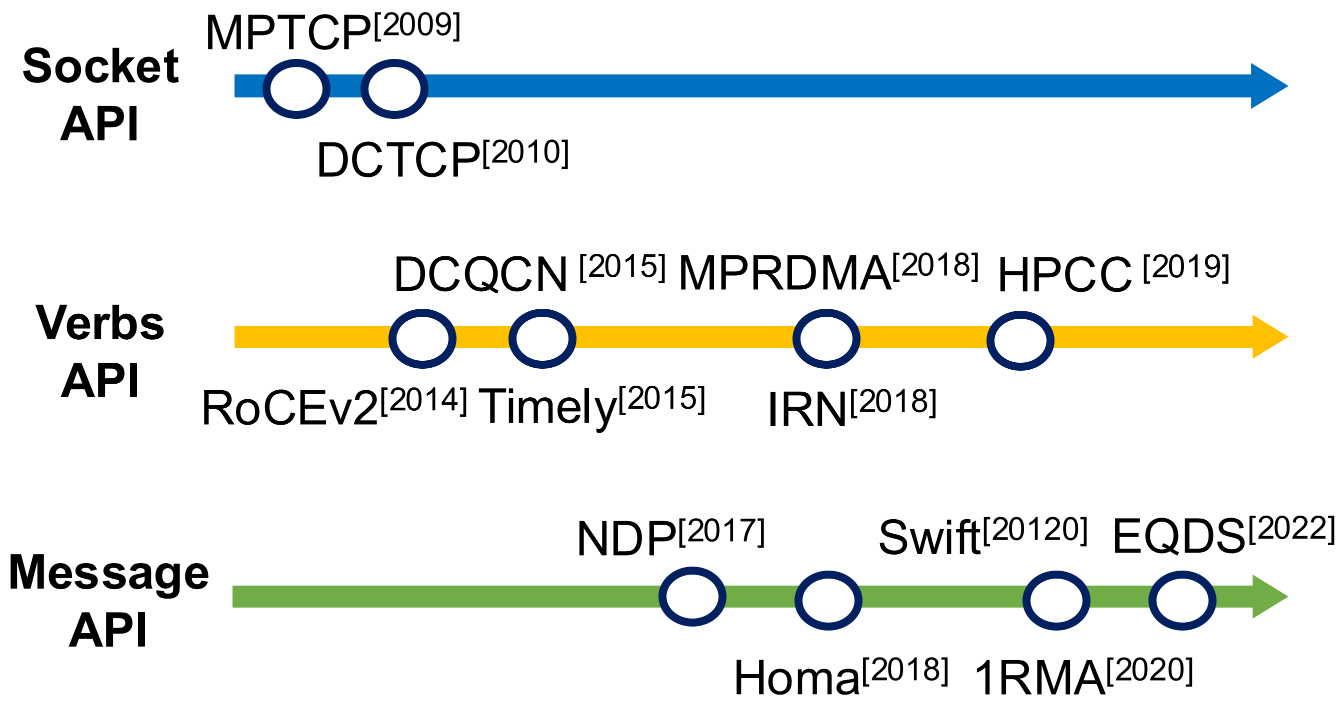
eTran: Extensible Kernel Transport with eBPF

Zhongjie Chen[1] Qingkai Meng[2] ChonLam Lao[3] Yifan Liu[1]
Fengyuan Ren[1] Minlan Yu[3] Yang Zhou[4]

[1]*Tsinghua University* [2]*Nanjing University*
[3]*Harvard University* [4]*UC Berkeley & UC Davis*

Evolving DC transports over the years

- Datacenter applications are evolving and increasingly diverse.
  - microservice, storage, etc.
- There is no *one-size-fits-all* transport for all workloads.



Semantics

➢ stream, message, connection, connectionless

Congestion control

➢ sender-driven, receiver-driven, switch-driven

Loss recovery

➢ Go-Back-N, SACK, packet trimming

However……

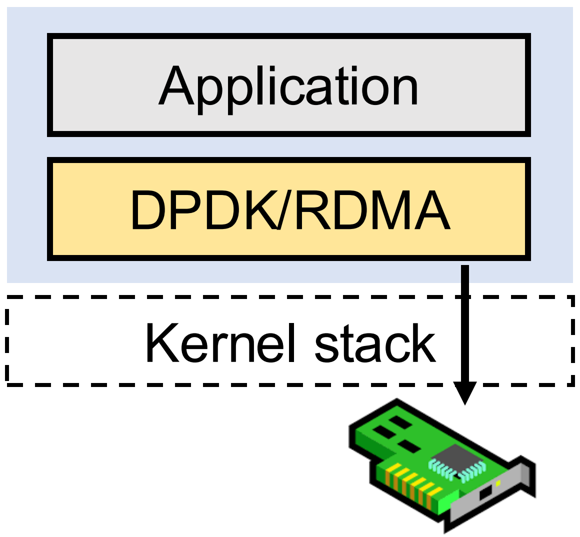
Transport	Proposal		Landing kernel
Softiwarp	2009	**~9 years**	2019
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
SoftRoCE	2014	**~2 years**	2016
Homa	2021	—	—

And more transports stay at the kernel-bypass prototype stage.

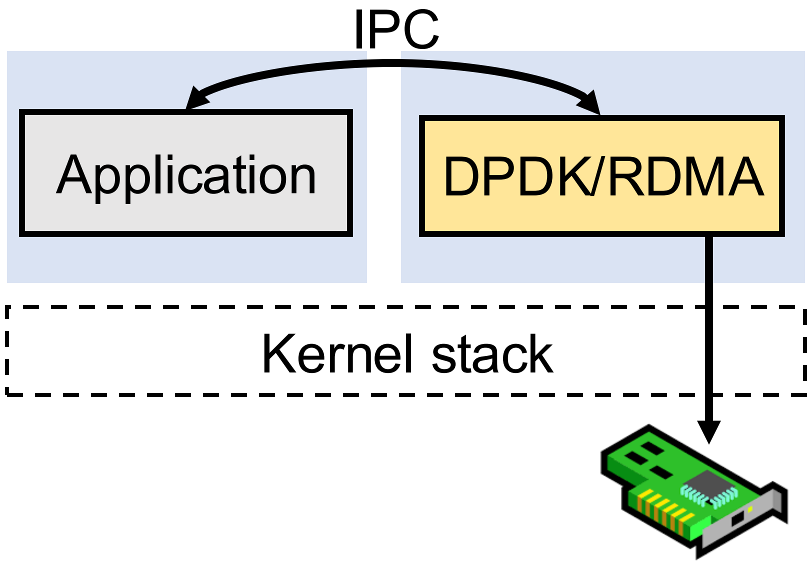
It takes a long time to **extend**, **customize** and **evolve** kernel transport.

Why don't we focus on kernel-bypass?

(a) LibraryOS



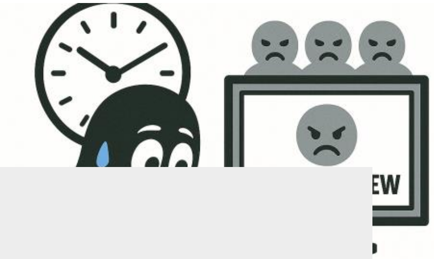
(b) Microkernel



- Direct HW access, security concern
- Dedicated resources

- Moderate performance
- Implementation from scratch

Why is it hard to extend kernel transport?



How to achieve agile customization, kernel safety, strong protection, and high performance for kernel transport ?

Heavy development and maintenance effort

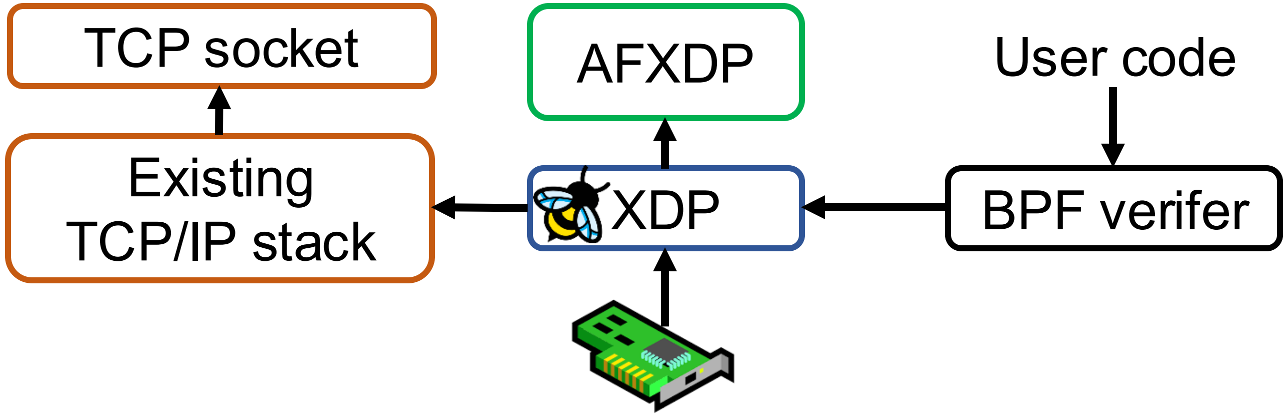
Unsatisfactory performance

Long time to land on kernel mainline

But it does provide **safety**, **multi-tenancy, out-of-the-box deployment**……

eBPF to the rescue

- eBPF (extended Berkeley Packet Filter)
  - ✓ Safely run programs in kernel at runtime guaranteed by BPF verifier
  - ✓ With an active community, it has impacted many OS subsystems
- XDP (eXpress Data Path) and AF_XDP
  - ✓ Fast kernel packet processing
  - ✓ Fast userspace packet processing based on XDP

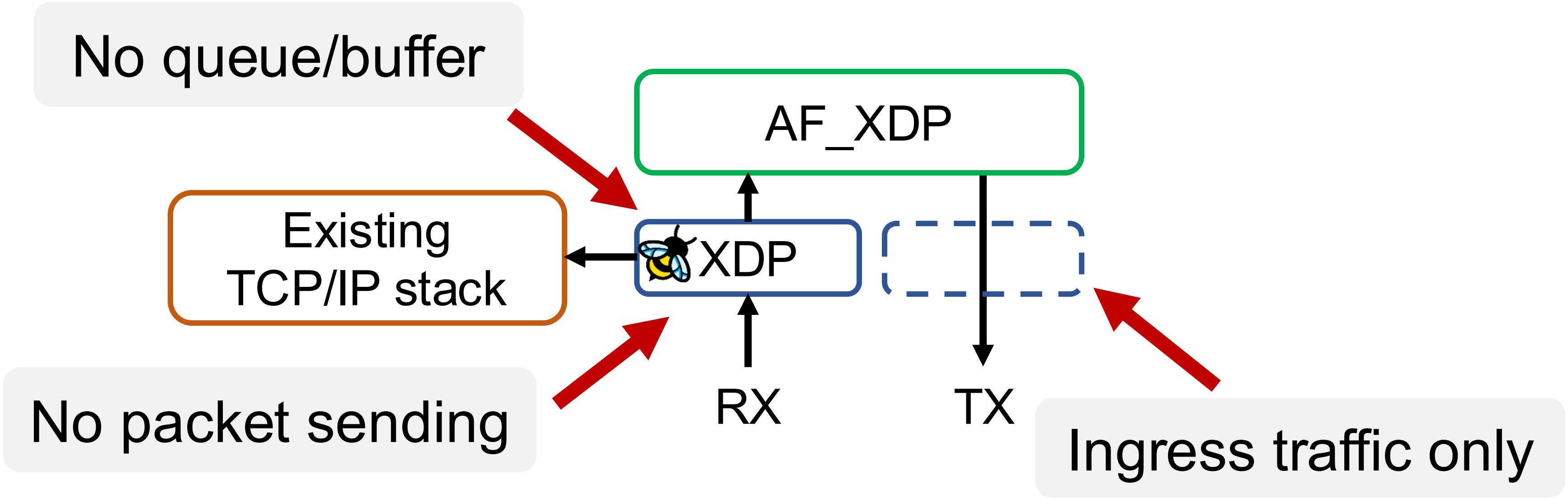


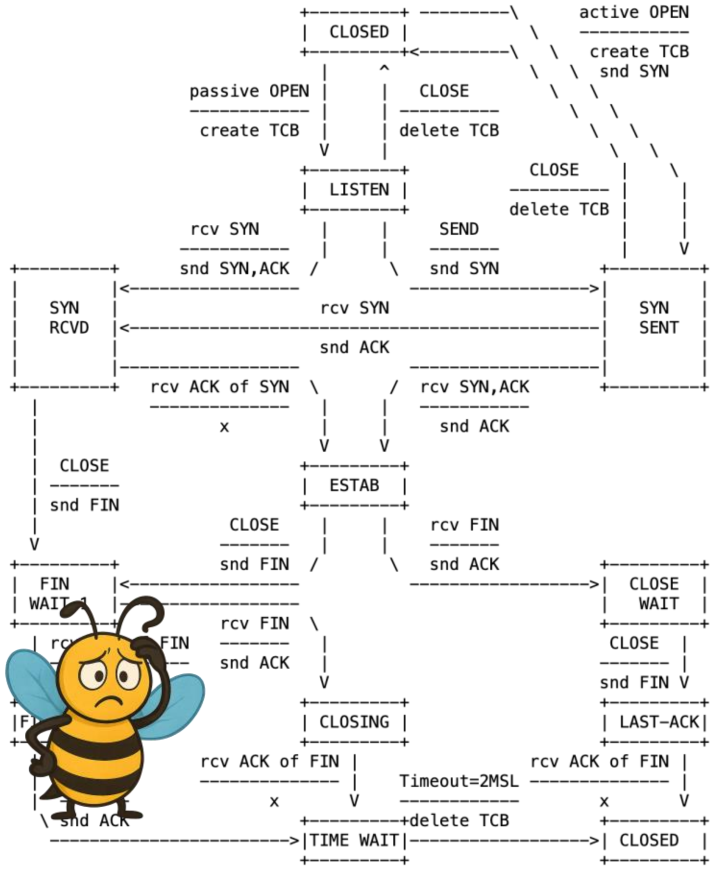
Challenges in implementing kernel transport with eBPF/XDP

C1: Full transport logic (e.g., TCP) is too complex for the eBPF programming model, even though eBPF is increasingly more powerful

- e.g., memory allocator, kfuncs, dynptr, rb-tree, etc

C2: Limitted operating points with eBPF/XDP





RFC793: TCP state machine

eTran design I: split transport paths

Challenge1

Network Transport

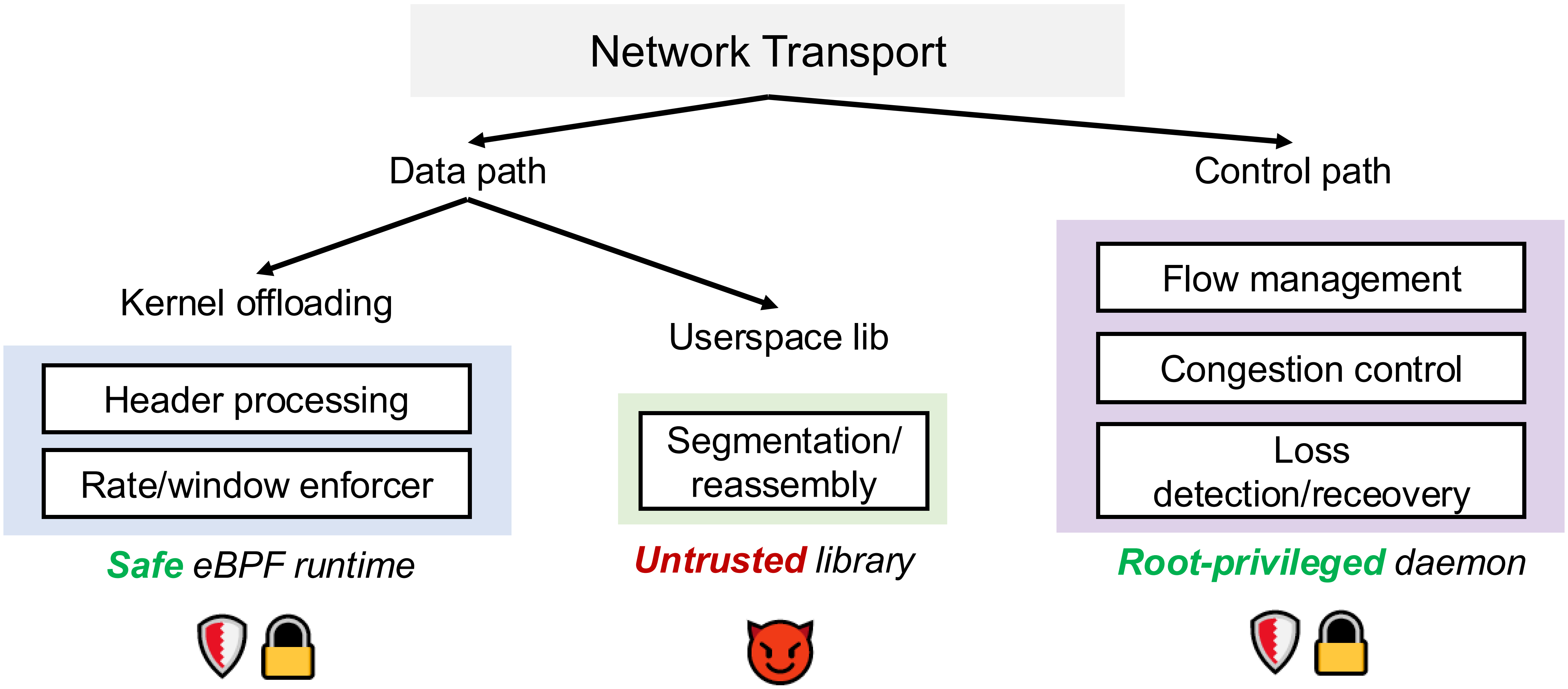
- Header processing
- Rate/window enforcer
- Segmentation/ reassembly

Packet-level events, common case, data path

- Flow management
- Congestion control
- Loss detection/receovery

Flow/RTT/RTO-level events, rare case, control path

eTran design I: split transport paths



eTran design II: extend eBPF subsystem

Challenge2

New XDP hooks

XDP_EGRESS

- Intercept egress traffic

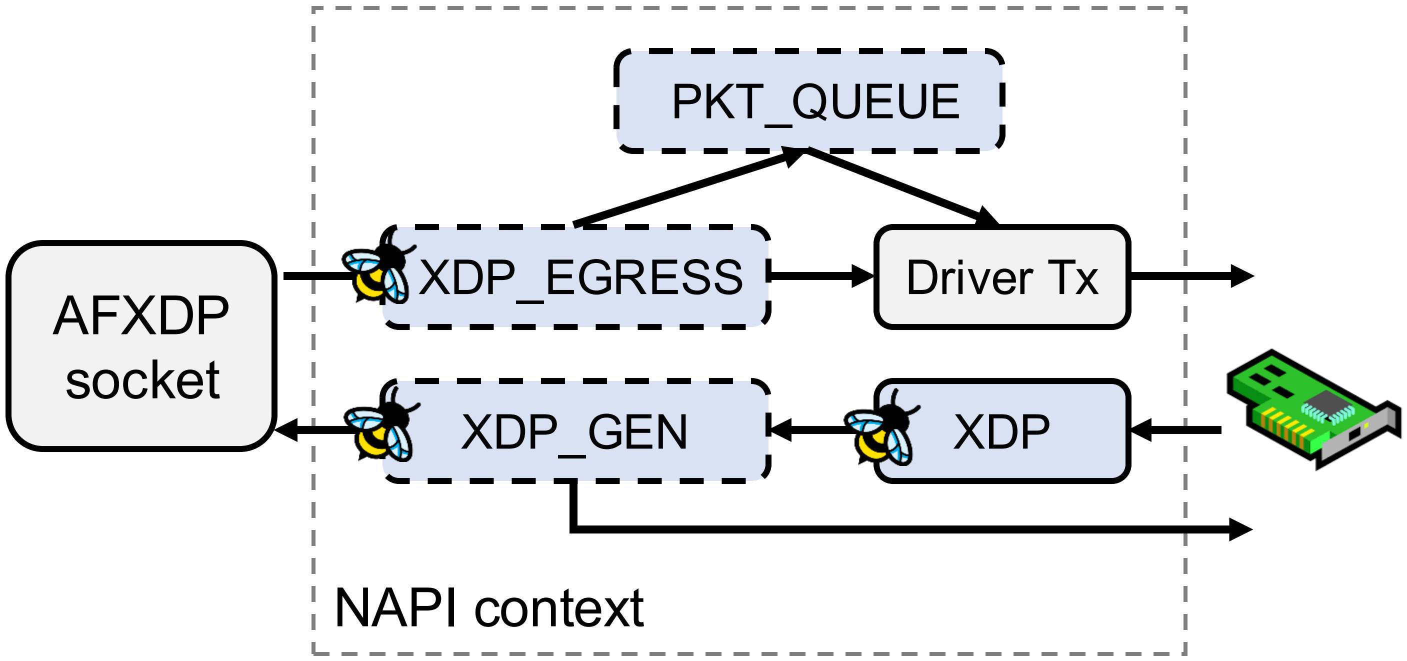
XDP_GEN

- Send packets proactively

New eBPF map

BPF_MAP_TYPE_PKT_QUEUE

- buffer/queue XDP frames

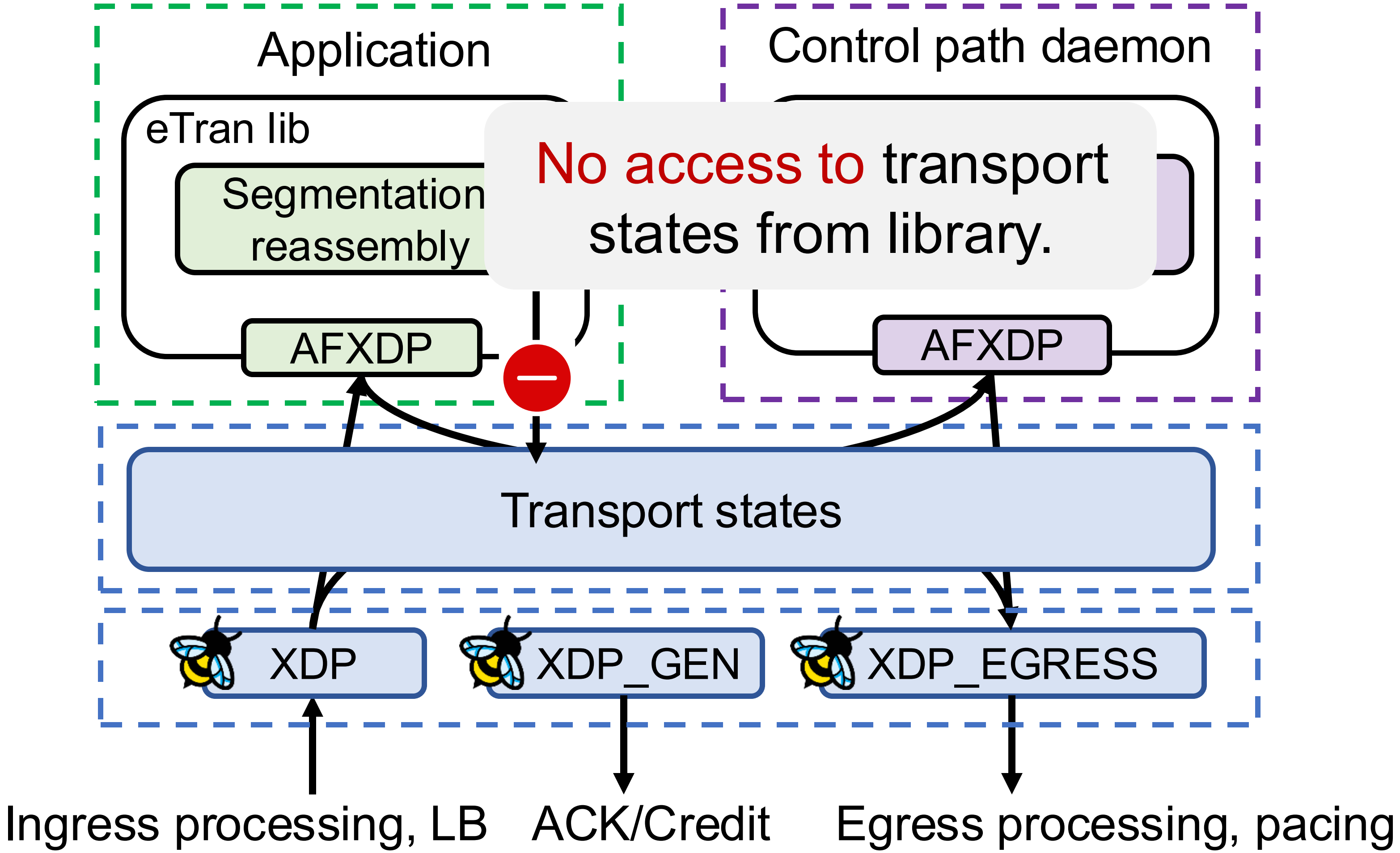


A bunch of optimizations for high performance.

- pre-allocated memory pool, batching, etc.

See the paper

eTran overall architecture



Extend eBPF subsystem: new hooks

```
int tcp(struct xdp_md *ctx) { // TCP logic...}
int homa(struct xdp_md *ctx){ // Homa logic...}
SEC(''XDP_EGRESS'')
int xdp_egress_prog(struct xdp_md *ctx)
{
  int proto;
  struct hdr_cursor nh
  void *data=(void *)(long)ctx->data;
  void *data_end=(void *)(long)ctx->data_end;
  nh.pos=data;
  proto=parse_ethhdr(&nh, data_end, &eth);
  if (proto!=bpf_htons(ETH_P_IP))
    return XDP_DROP;
  proto=parse_iphdr(&nh, data_end, &iph);
  if (proto==IPPROTO_TCP||proto==IPPROTO_HOMA)
    bpf_tail_call(                   o);
  return XDP_DROP;
}
SEC(''XDP_GEN'')
int xdp_gen_prog(struct xdp_md *ctx)
{
  prepare_ack_pkt(ctx);
  return XDP_TX;
}
```

XDP_EGRESS

XDP_GEN

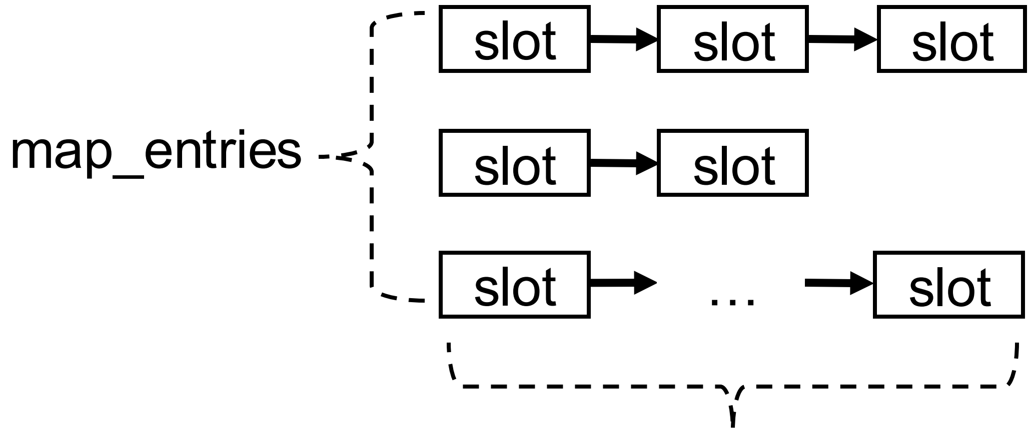
- Comply with all eBPF/XDP program constraints
  - # of instructions, bounded loop, etc
- Reuse most existing XDP infrastructures
  - Similar data structures and return codes
  - Most BPF helper functions for XDP

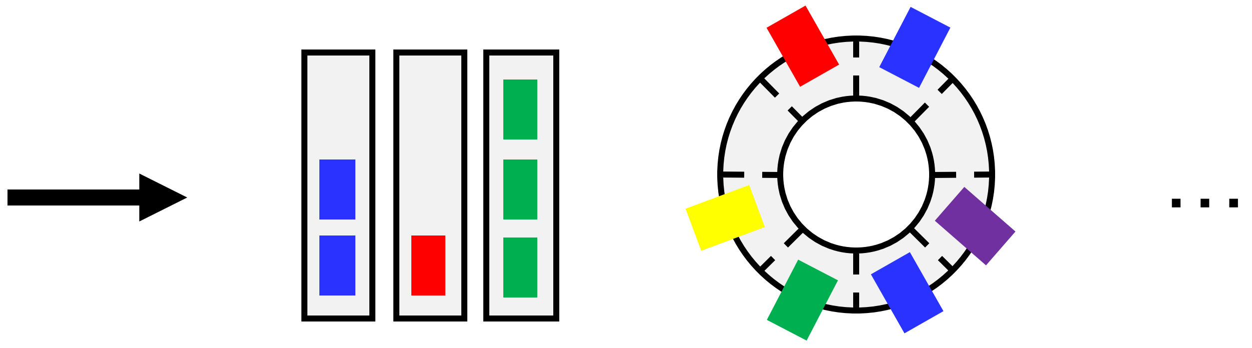
Extend eBPF subsystem: new BPF map

```
struct {
  __uint(type, BPF_MAP_TYPE_PKT_QUEUE);
  __type(key, __u32);
  __uint(max_entries, MAX_BKT);
  __uint(map_extra, MAX_PKT_PER_BKT);
} xdp_pkt_queue SEC(''.maps'')
```

- Only store frame pointers
- Memory size is static
- kfuncs for operations
- Cooperate with BPF timer

See the paper



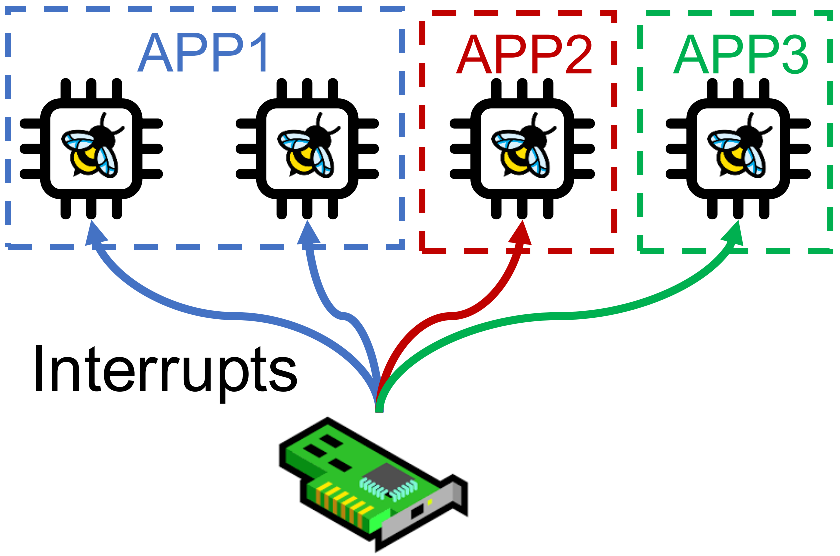


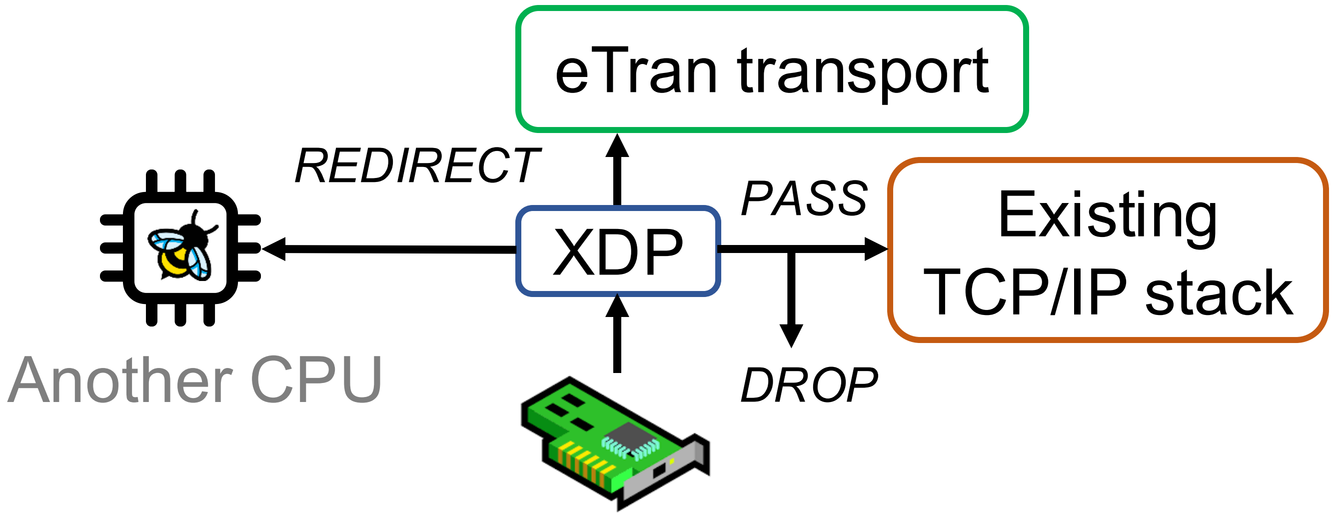
Per-flow queue

TimingWheel [SIGCOMM'17]

More features: multi-tenancy

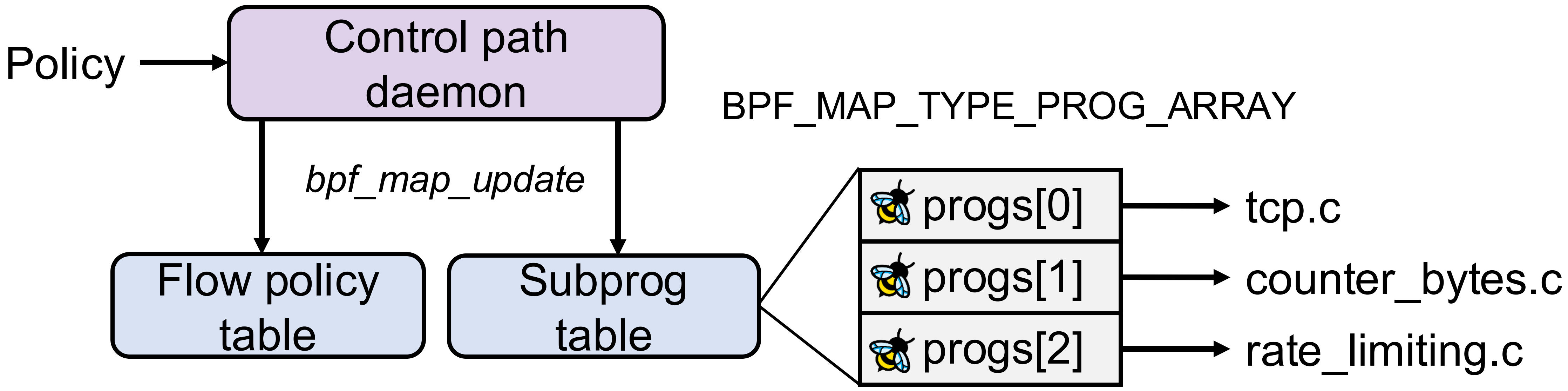
- Queue-level resource allocation by control path daemon
  - Isolate NAPI/IO buffers and avoid monopolizing the NIC (e.g., DPDK)
- Magic of XDP
  - Co-exist with other kernel transports flexibly





More features: traffic management

- Call subprograms with *bpf_tail_call* feature.
  - Traffic monitoring, ACL, rate limiting, etc.



More details of eTran

- Hook performance optimizations.
- Flow scheduling with BPF timer.
- AF_XDP virtual sockets for multi-queue.
- Loss recovery under eTran.
- Receiver-driven CC under eTran.

Not covered in this talk

Case studies & Implementation

DCTCP[SIGCOMM'10]

- ➢ Connection-based
- ➢ Stream semantic
- ➢ Sender-driven CC
- ➢ POSIX API

Homa[SIGCOMM'18]

- ➢ Connection-less
- ➢ Message semantic
- ➢ Receiver-driven CC
- ➢ RPC API

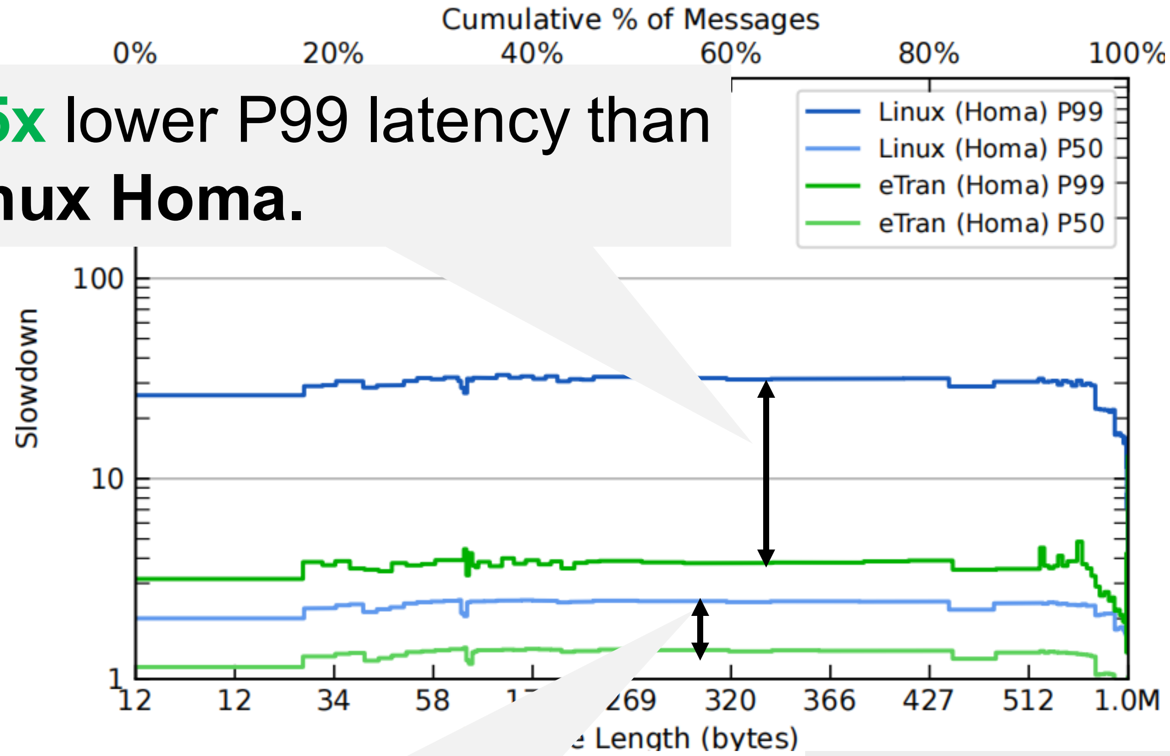
- Kernel modification
  - 2.6K LoC for extending eBPF subsyetem
- eBPF code
  - 7.5K LoC eBPF for TCP and Homa
- Userspace code
  - 8K LoC for control path dameon and 5.6K for transport libs

Evaluation

- Baseline
  - Linux Homa
  - Linux TCP, TAS[Eurosys'19](kernel-bypass TCP/IP stack)
- Experiment setup
  - 10 Cloudlab xl170 machines running **customized** Linux kernel 6.6.0.
  - RPC workload for Homa
  - Key-value store application for TCP.
- Open source
  - https://github.com/eTran-NSDI25/eTran

Linux Homa vs. eTran Homa

RPC workload



7.5x lower P99 latency than **Linux Homa.**

3.6x lower P50 latency than **Linux Homa.**

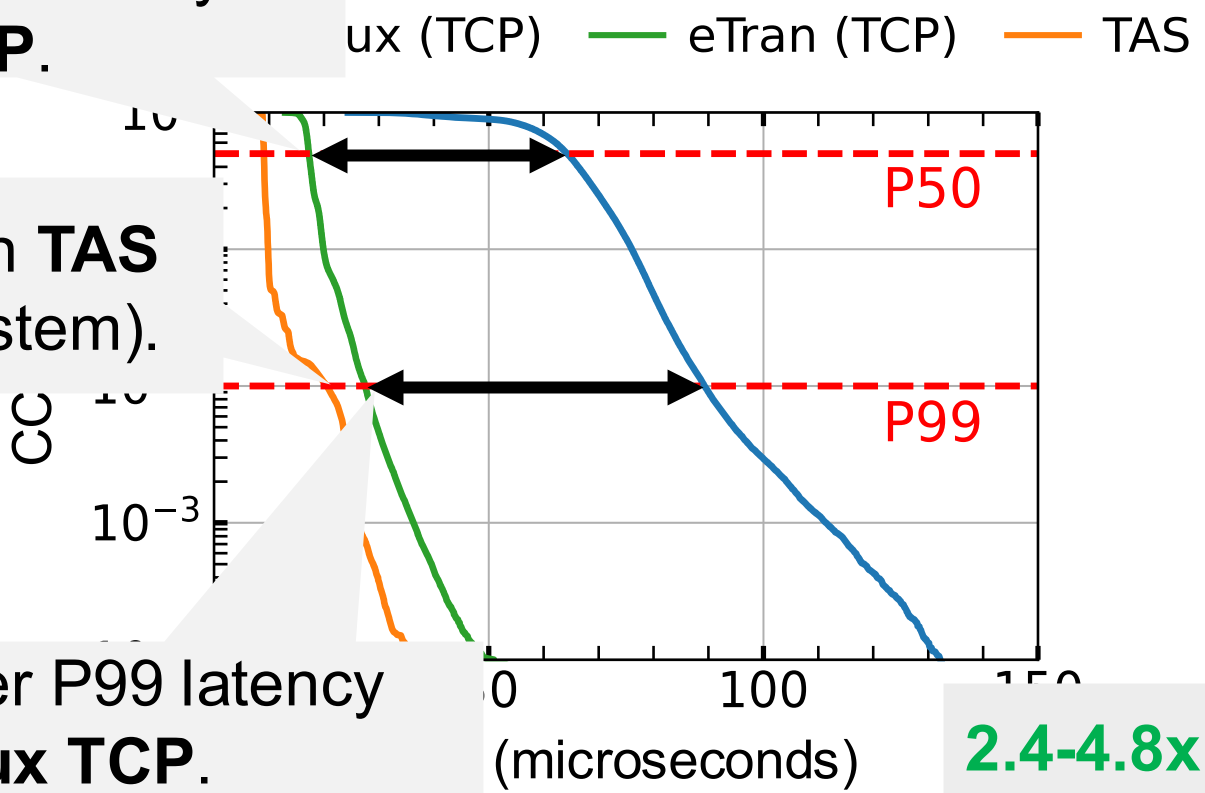
4.3/4.9x lower P50/P99 latency for **large message workload** .

Linux DCTCP, TAS vs. eTran DCTCP

Key-value store application

3.7x lower P50 latency than **Linux TCP**.

Slightly worse than **TAS** (kernel-bypass system).



3.2x lower P99 latency than **Linux TCP**.

2.4-4.8x higher throughput than **Linux TCP**.

eTran summary

- eTran is an extensible kernel transport system achieving
  - Agile customization
  - Kernel safety
  - Strong protection
  - High performance
- eTran achieves these goals by
  - ➢ Extending the kernel-safe eBPF
  - ➢ Hiding transport states inside the kernel
  - ➢ Absorbing techniques from user-space transports

Thank you!

Icons are generated by GenAI ☺