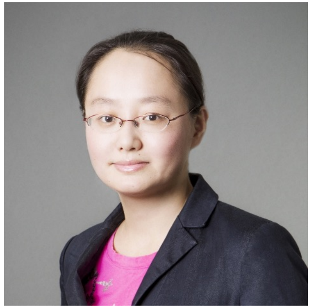
Electrode: Accelerating Distributed Protocols with eBPF



*** Yang Zhou**

Harvard University

* Zezhou Wang

Peking University

Sowmya Dharanipragada

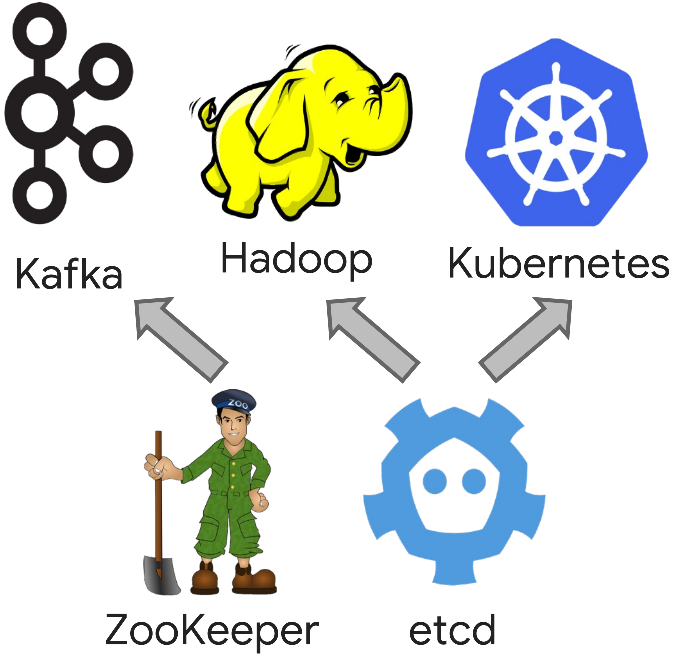
Cornell University

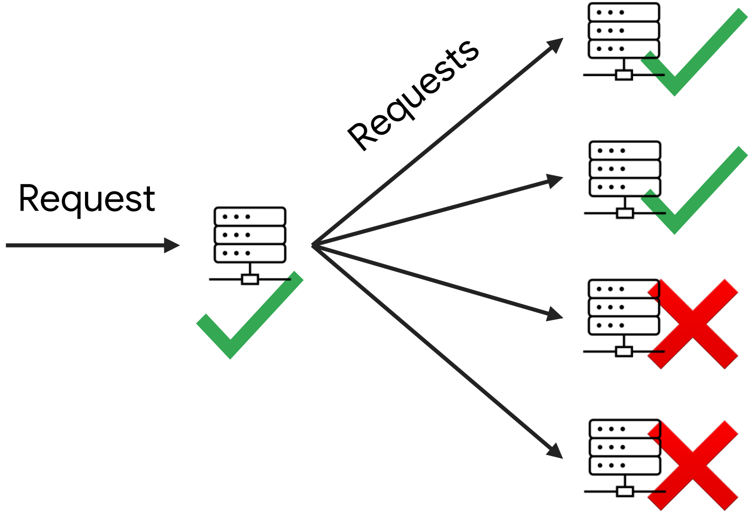
Minlan Yu

Harvard University

* Co-primary author

Cloud applications need consensus protocols for high availability

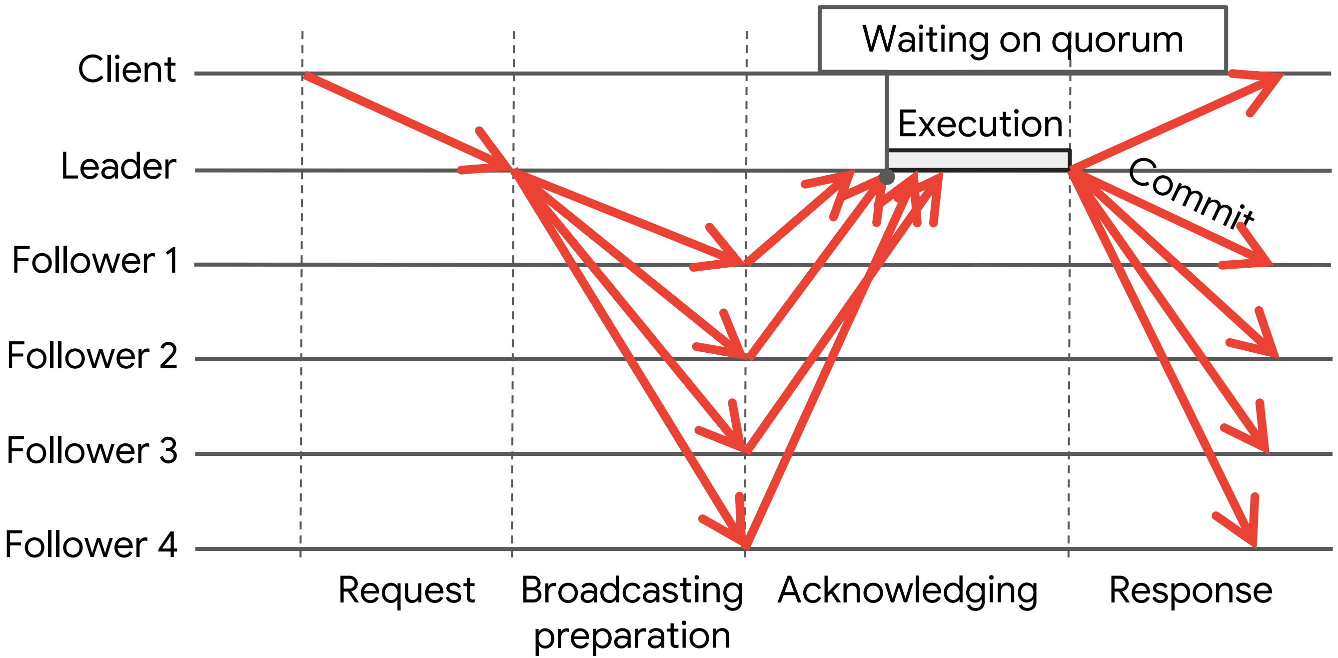




This talk: accelerating consensus protocol implementations for cloud apps

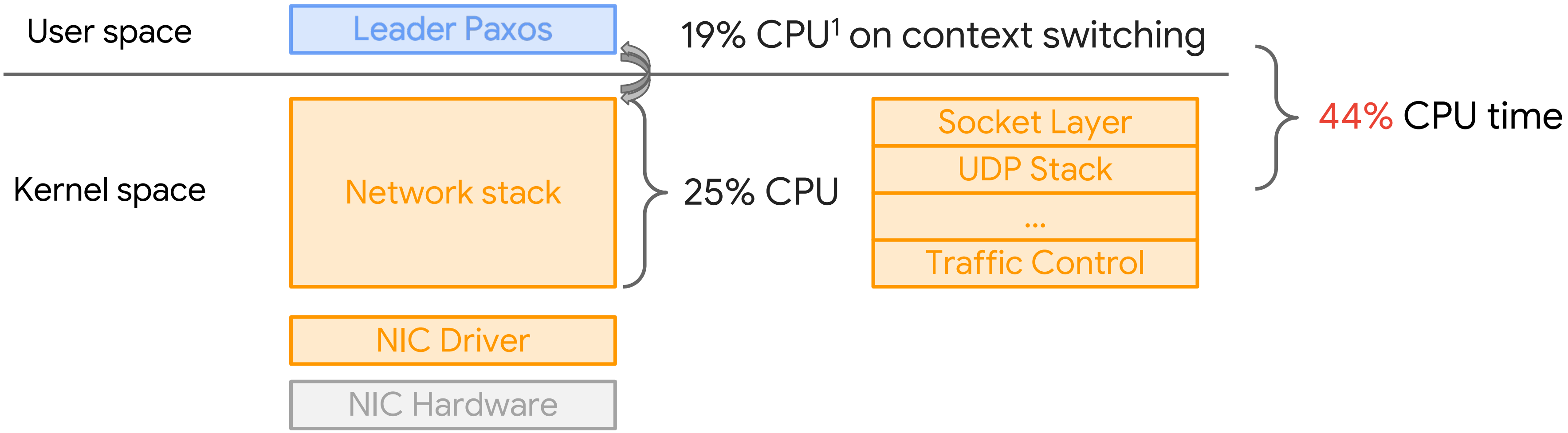
Example: a simplified Multi-Paxos consensus protocol

... we target **in-memory** data replication (i.e., without persistence)



In this example, the leader node invokes networking APIs **14** times per request

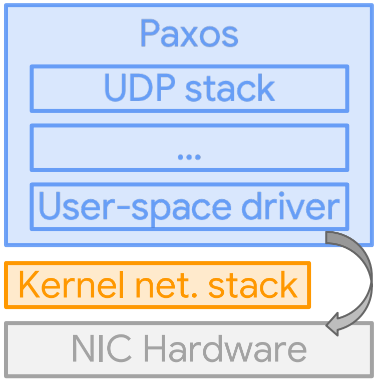
Kernel networking: Multi-Paxos incurs high kernel overhead



[1] Experiment settings: Multi-Paxos with 5 replicas using socket APIs from Linux kernel 5.8.0, measured on leader node.

Kernel bypassing: does it solve all problems?

DPDK: moving stacks to user space,
using busy polling instead of interrupt



+ Good performance
- Security and isolation vulnerability
- Not cloud-friendly: Busy polling discourages CPU sharing
- High maintenance overhead: compatibility with others

Kernel bypassing is not a panacea

Can we achieve both?

Approaches	Security, isolation, cloud-friendly, ease maintenance	Performance
Kernel	High	Low
Kernel bypassing	Low	High
Kernel customization for apps[1-3]	High	Medium - High

Electrode demonstrates it on modern Linux kernels without kernel modifications or rebooting.

... we target UDP-based applications inside data centers.

[1] Bershad, Brian N., et al. "Extensibility safety and performance in the SPIN operating system." *SOSP 1995*
[2] Engler, Dawson R., et al. "Exokernel: An operating system architecture for application-level resource management." *ACM SIGOPS Operating Systems Review 1995*
[3] Zhong, Yuhong, et al. "XRP: In-Kernel Storage Functions with eBPF." *OSDI 2022*

Talk Outline

High-level methodology and challenges

Electrode: three kernel customizations for Paxos

Evaluation

Talk Outline

High-level methodology and challenges

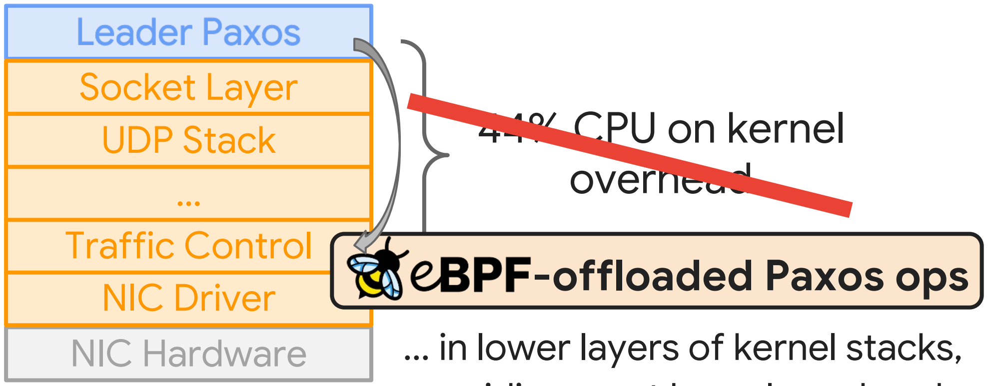
Electrode: three kernel customizations for Paxos

Evaluation

Leveraging eBPF to accelerate Paxos implementation

eBPF is a mechanism to offload functions to existing kernel at runtime and safely

- It achieves safety via static verification



... in lower layers of kernel stacks, avoiding most kernel overhead

+ Good performance
+ Secure, isolate well: kernel-native
+ Cloud-friendly: no busy polling
+ Reusing kernel networking stack

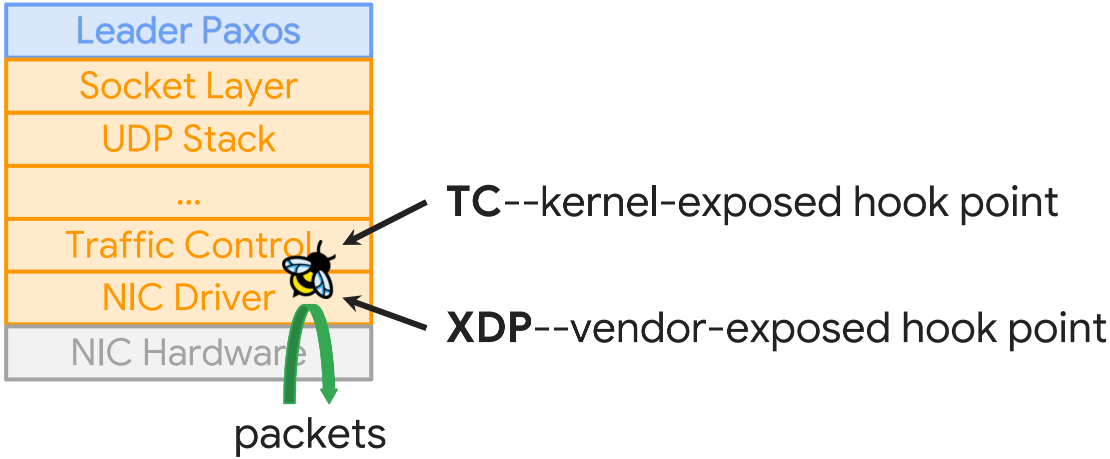
Paxos on eBPF

eBPF was commonly used for simple network functions:

- Packet filtering, monitoring, load balancing

Now we are using it for application functions:

- A Paxos message is usually small enough to fit into a single packet



Challenges of processing Paxos messages in eBPF

eBPF programming model is **constrained** because of static verification for safety

- Limited # of instructions, bounded loops, static memory allocation
- Challenging to support complex pointer arithmetics for memory accesses

What's the right division of labor between user and kernel

- that can greatly **reduce kernel overhead**
- while being **implementable** in eBPF for offloaded ops?

Division of labor between user and kernel

Perf-critical and simple to kernel

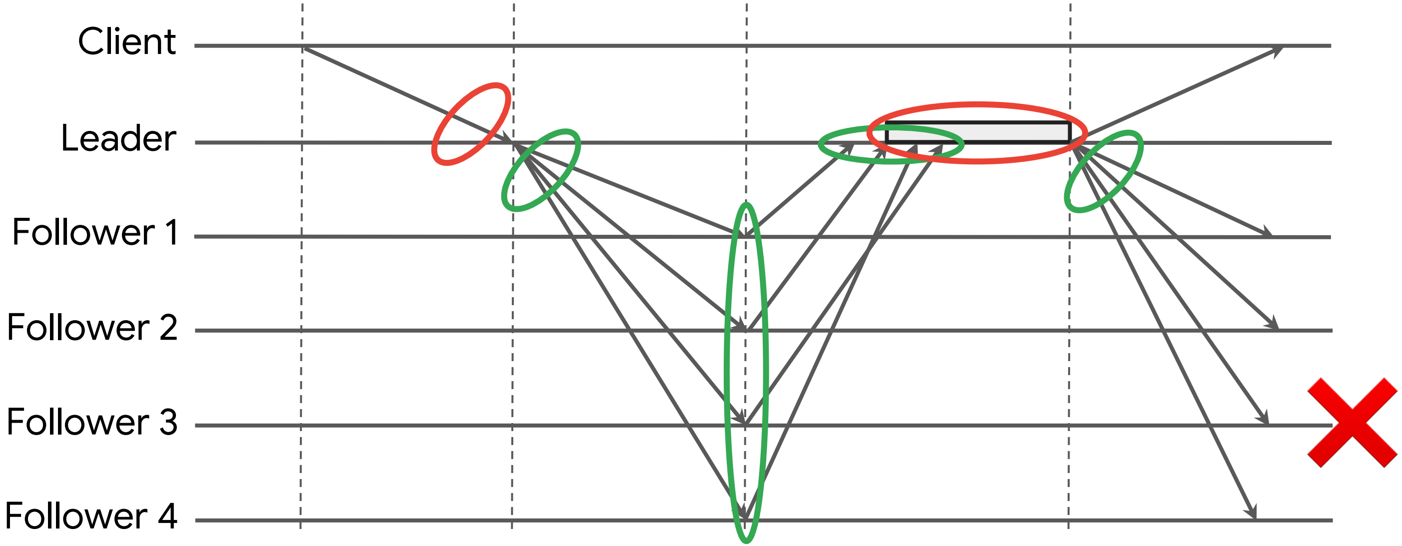
Broadcasting | **Acknowledging** | **Wait-on-quorum**

Complex to user

Client-facing ser/deserialization
(complex pointer arithmetics)

Application ops
(dynamic memory allocation)

Failure, msg loss/reordering
(too complex for static verification)



Talk Outline

High-level methodology and challenges

- **Leveraging eBPF to offload perf-critical and simple ops to the kernel**

Electrode: three kernel customizations for Paxos

Evaluation

Talk Outline

High-level methodology and challenges

- Leveraging eBPF to offload perf-critical and simple ops to the kernel

Electrode: three kernel customizations for Paxos

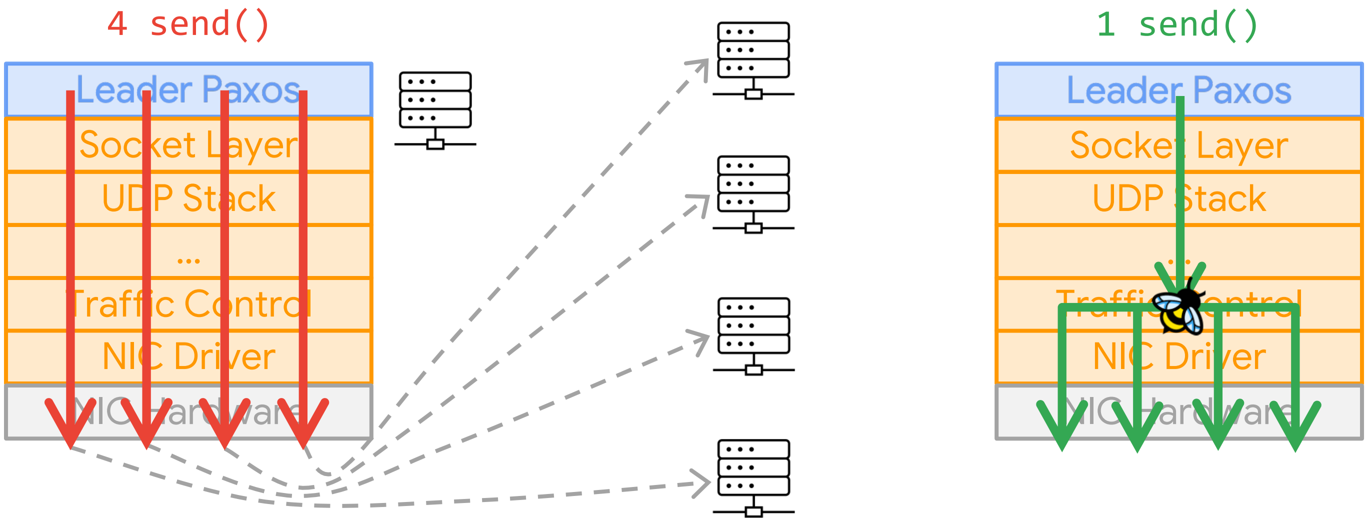
Evaluation

Electrode offload #1: message broadcasting

Perf-critical: # of context switching and stack traversing is linear to # of replicas

Simple for eBPF: TC to clone and modify packets (using `bpf_clone_redirect()`)

- Incur only once context switching and upper stack traversing
- Handle message loss in user space by resending messages (unlikely events)

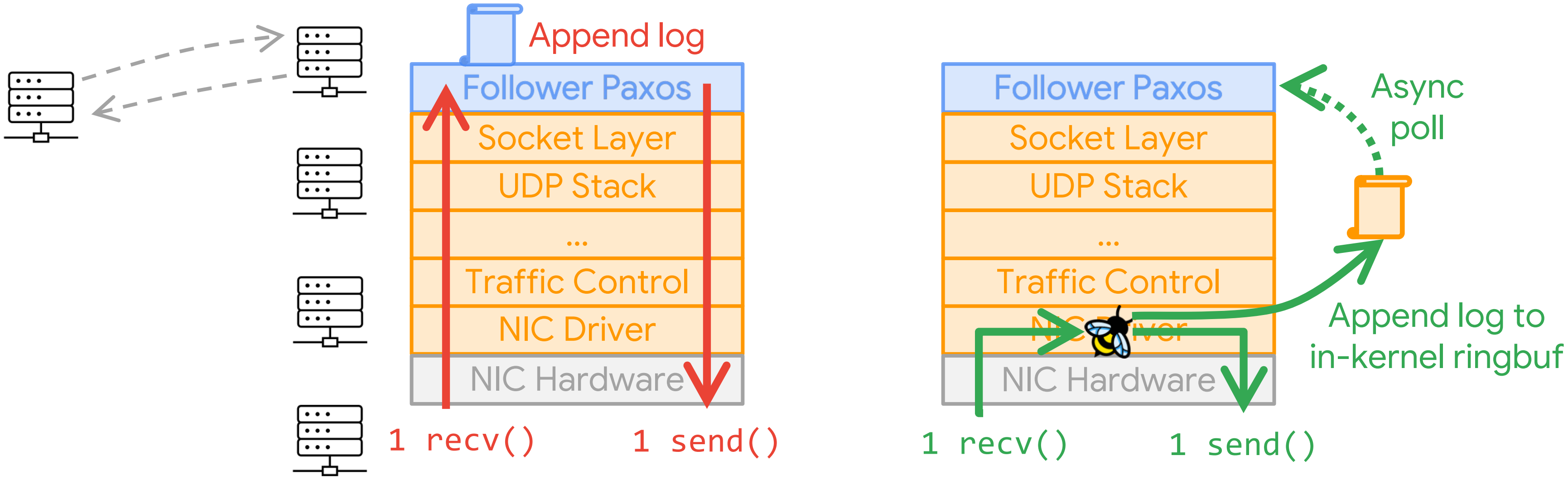


Electrode offload #2: fast acknowledging

Perf-critical: incurring twice the kernel latency on the critical path

Simple for eBPF: XDP to buffer log entries and quickly ack back

- Remove the kernel latency from the critical path
- Detect special cases (e.g., message loss, full buffer) and forward to user space

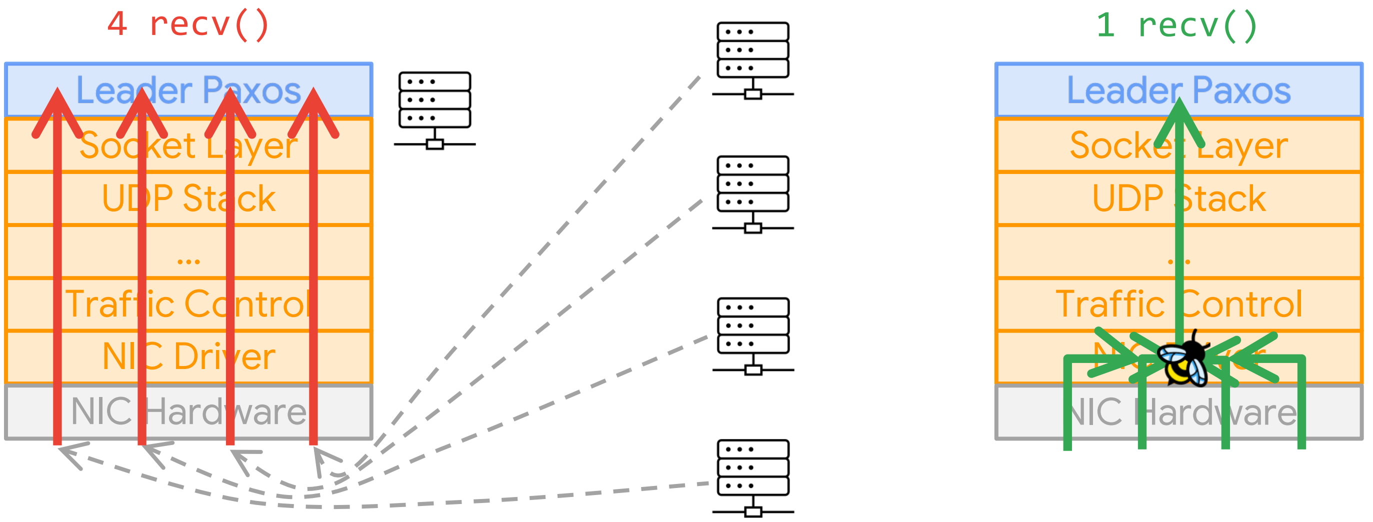


Electrode offload #3: waiting on quorum

Perf-critical: leader recv ACKs from all followers, each incurring kernel overhead

Simple for eBPF: XDP to maintain # of ACKs in the driver layer

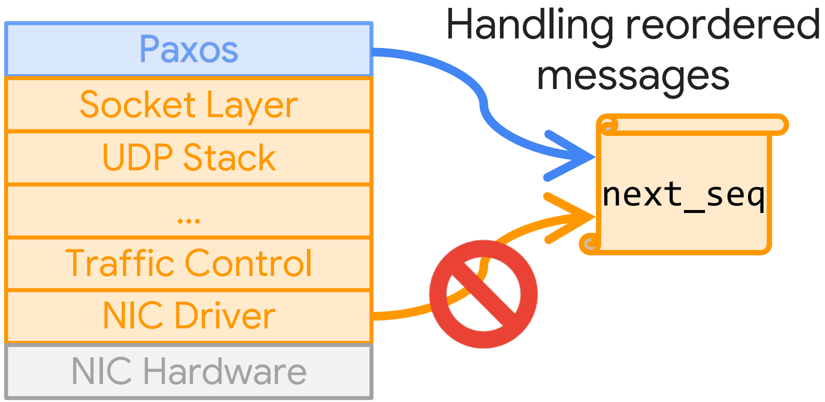
- Filter unnecessary ACKs: only the quorum-reaching ACK incurs kernel overhead
- Use bitset instead of counter to avoid double counting



State synchronization challenge

No shared memory between eBPF and user space for kernel safety

- Communicate by copying data back and forth



Our approach 1: detaching eBPF program

Our approach 2: using eBPF map as an on-off switch

Details in the paper

Talk Outline

High-level methodology and challenges

- Leveraging eBPF to offload perf-critical and simple ops to the kernel

Electrode: three kernel customizations for Paxos

- Broadcasting, fast ack'ing, waiting on quorum beneath network stacks

Evaluation

Talk Outline

High-level methodology and challenges

- Leveraging eBPF to offload perf-critical and simple ops to the kernel

Electrode: three kernel customizations for Paxos

- Broadcasting, fast ack’ing, waiting on quorum beneath network stacks

Evaluation

Evaluation overview

Workloads:

- Multi-Paxos on 3/5/7 replicas
- Transactional replicated key-value store on 3/5/7 replicas (skipped here)

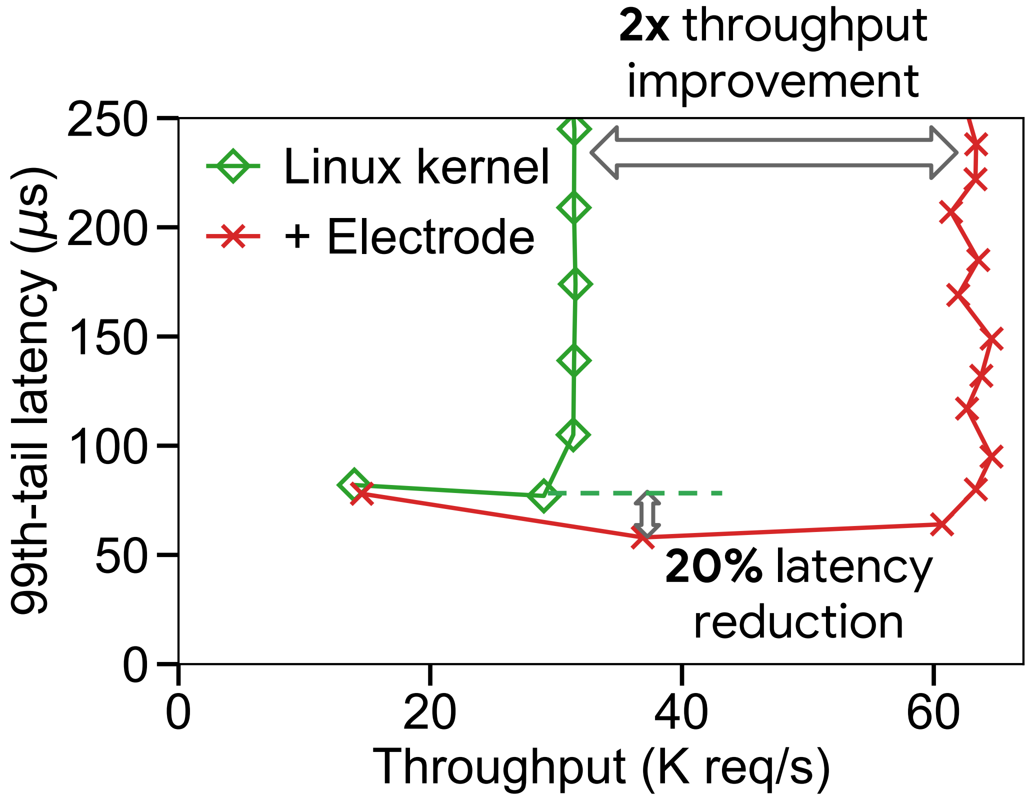
Metrics: we vary # of clients and measure:

- Throughput, median/99th-tail latency, and CPU utilization

Testbed:

- Bare metal machines from Cloudlab xl170
    - Stock Linux kernel 5.8.0 and ubuntu 20.04
    - Mellanox ConnectX-4 25Gbps NIC
- We **do not** use IP multicast (Cloudlab does not support either)

Load-latency curves (5 replicas)



14 times context switching and stack traversing

Electrode

5 times

Other results

7 replicas: 2.3x throughput improvement and 40% tail latency reduction

Comparison to kernel-bypassing:

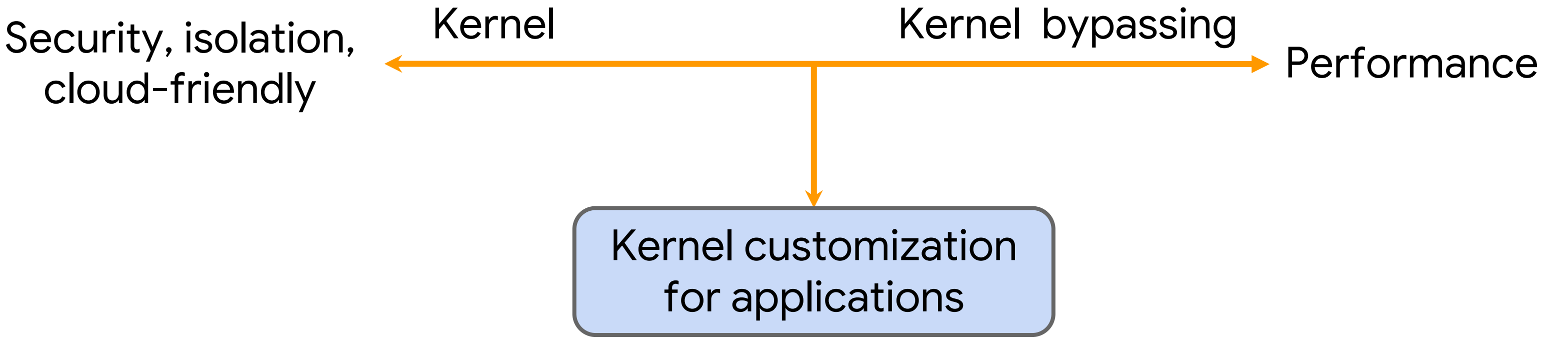
- Around half performance of DPDK-based one (throughput and latency)
  - Hard-to-offload operations in Paxos
  - eBPF with XDP/TC cannot beat DPDK, as it is interrupt-driven
- Electrode is a kernel-native approach (i.e., security, isolation, cloud-friendly, etc)

More in the paper!

- Improvement on the transactional replicated key-value store
- Performance contribution of each eBPF optimization
- Reduction of CPU usage

Electrode Summary

- Consensus protocols under kernel stacks suffer from high kernel overhead
- We design a set of eBPF-based kernel customizations to reduce such overhead
  - Without kernel modifications or rebooting
  - Up to 2.3x throughput speedup and 40% latency reduction for Multi-Paxos



Thank You!