

RDMATracer: Lessons from scaling BPF to detect RDMA Device Drivers Bugs in real time

LPC 2025

Prankur Gupta, Maxim Samoylov, Theophilus A. Benson,

Training Infrastructure at Meta

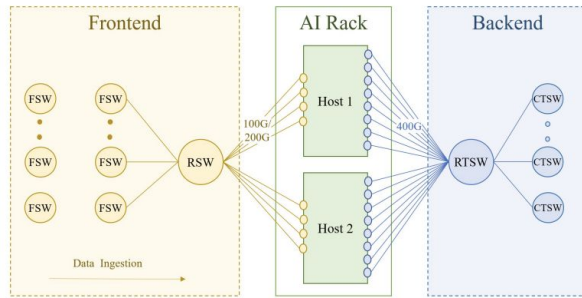


Figure 5: Frontend and Backend networks

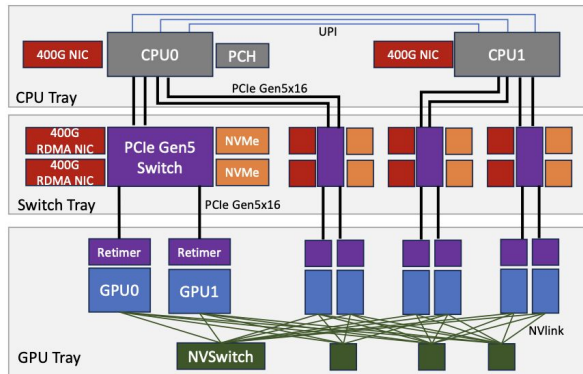
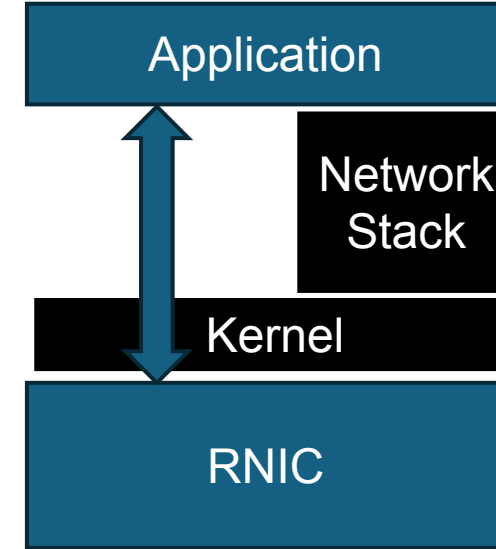


Figure 4: Grand Teton platform



RDMA is used extensively for GPU-based training for proven performance benefits

- Avoids network stack
- Allows direct transfers to memory

High level steps for Training

Step 2: Server Configuration
(kernel programs NIC and allocate memory)

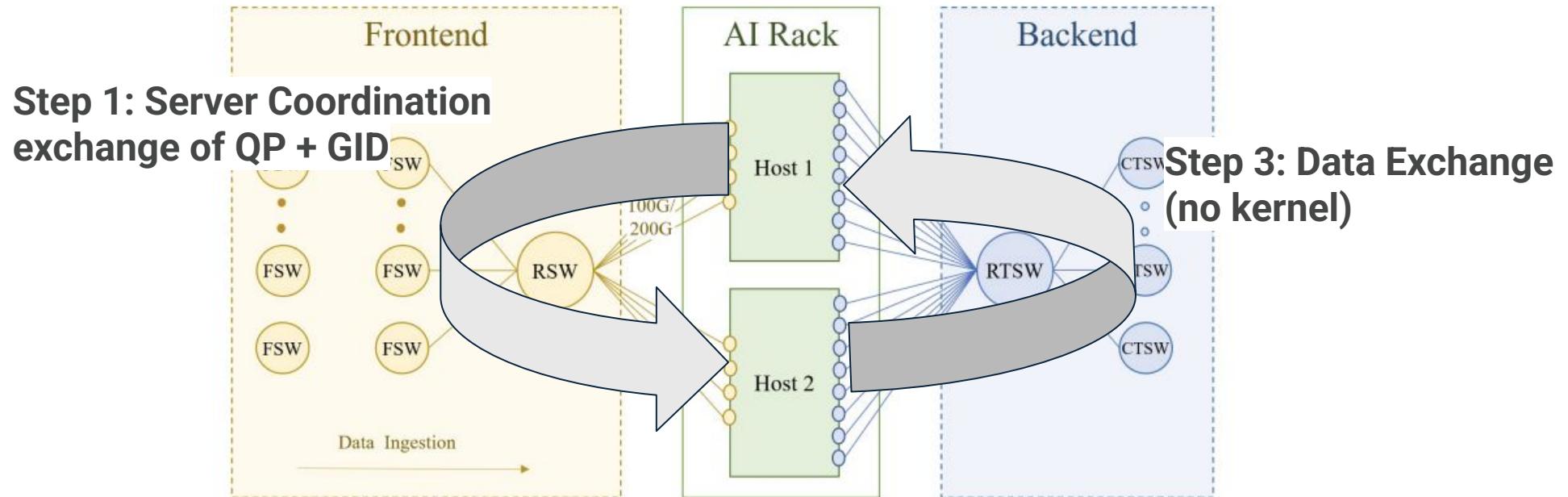
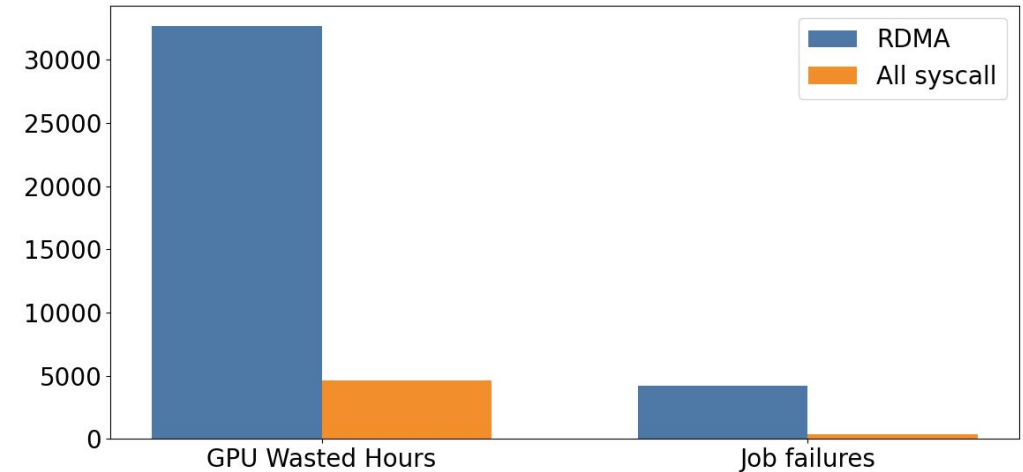
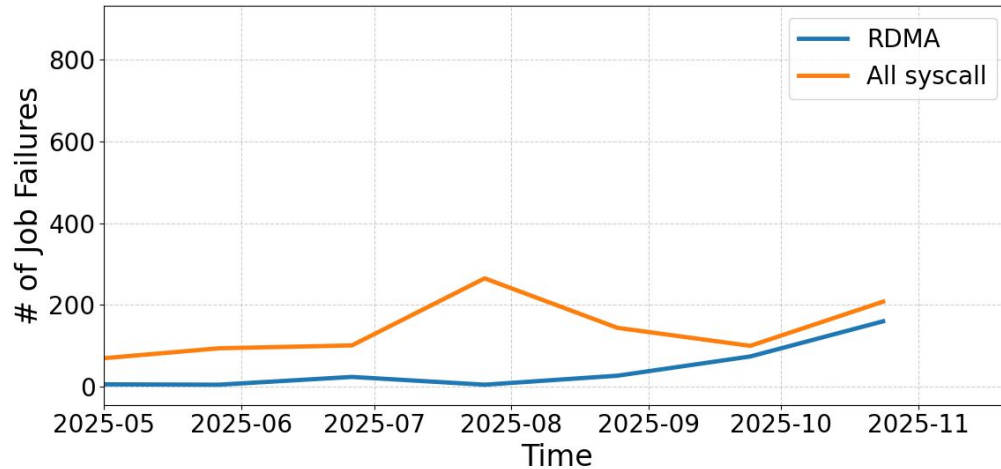


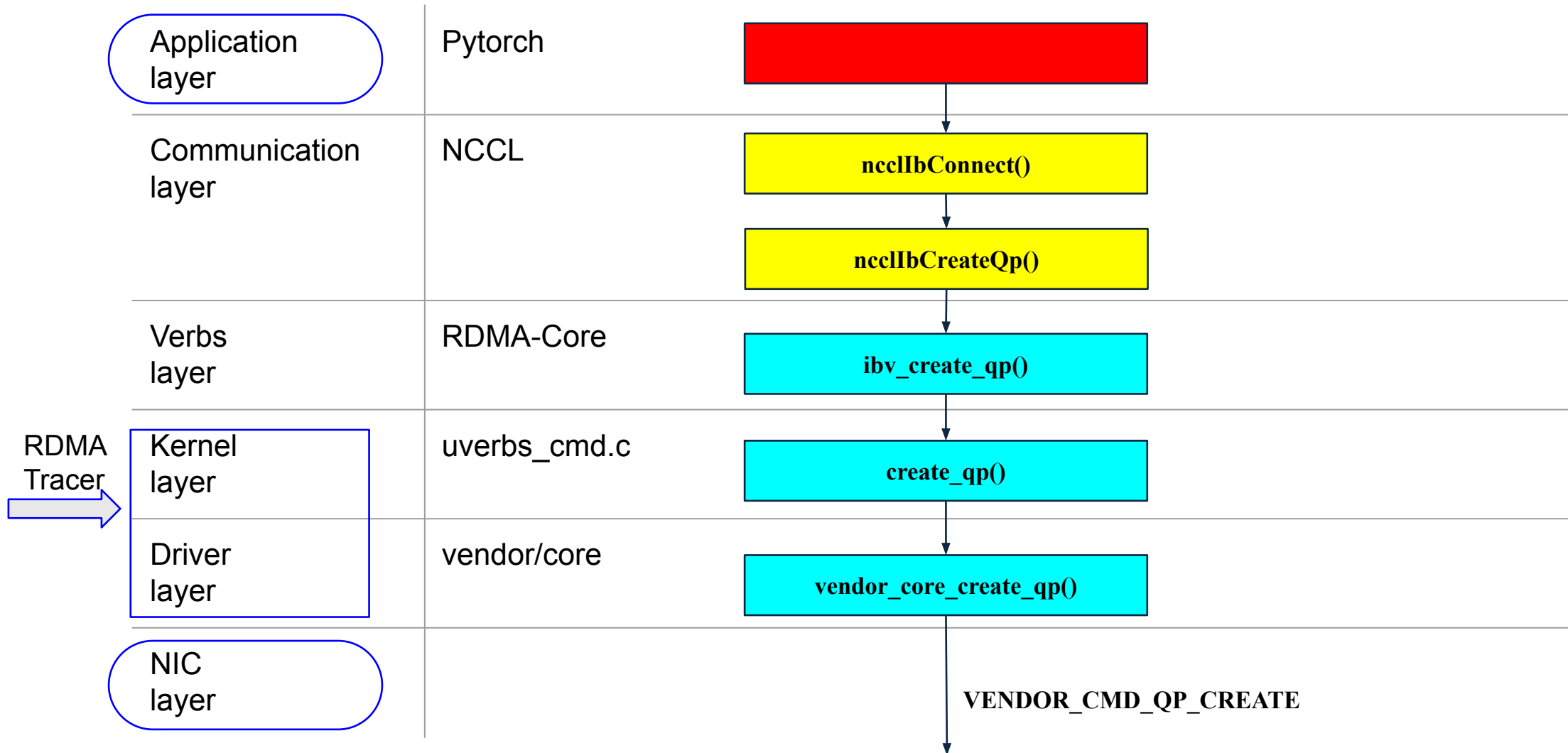
Figure 5: Frontend and Backend networks

Case Study: Impact of Syscall Errors at Meta Scale

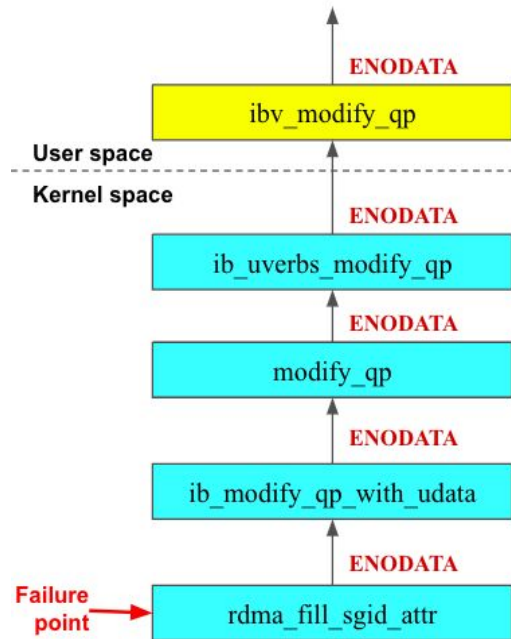


- On average, 10-20% of AI job crashes per day, are associated with RDMA (not network-related)
- For the last 6 month, 32k+ GPU Hours wasted, 5k+ for RDMA subsys
- Multiple blocking issues for flagship AI model trainings in the past, distracting 10+ people for weeks to root-cause and mitigate.

RDMA stack - Control Path



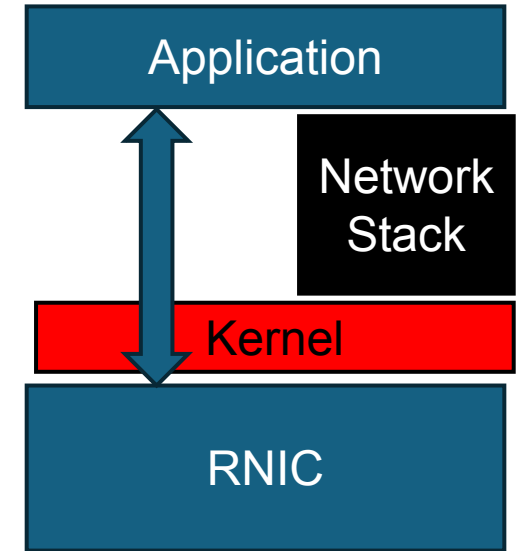
Syscall Errors in RDMA Network



- Syscall/Async errors impact 29% of job
- Manual diagnosis with ftrace
 - Ftrace does not expose call return values
 - Required trace comparison: bad trace is shorter!
- Issue motivated the need for a system
 - Process all NCCL failures in production
 - Dynamically compare traces
 - Localize cause of differences to state differences

Why build our RDMATracer with eBPF?

- Linux is still crucial in RDMA systems
 - Performs control and resource accounting
 - Processes all ibverbs ioctls - rdma core (no retries)
- Building eBPF provides
 - Flexibility: monitor and extract additional data from internal variables
 - Complexity: implement complex aggregation logic
 - Interoperability: no need to modify kernel or runtime

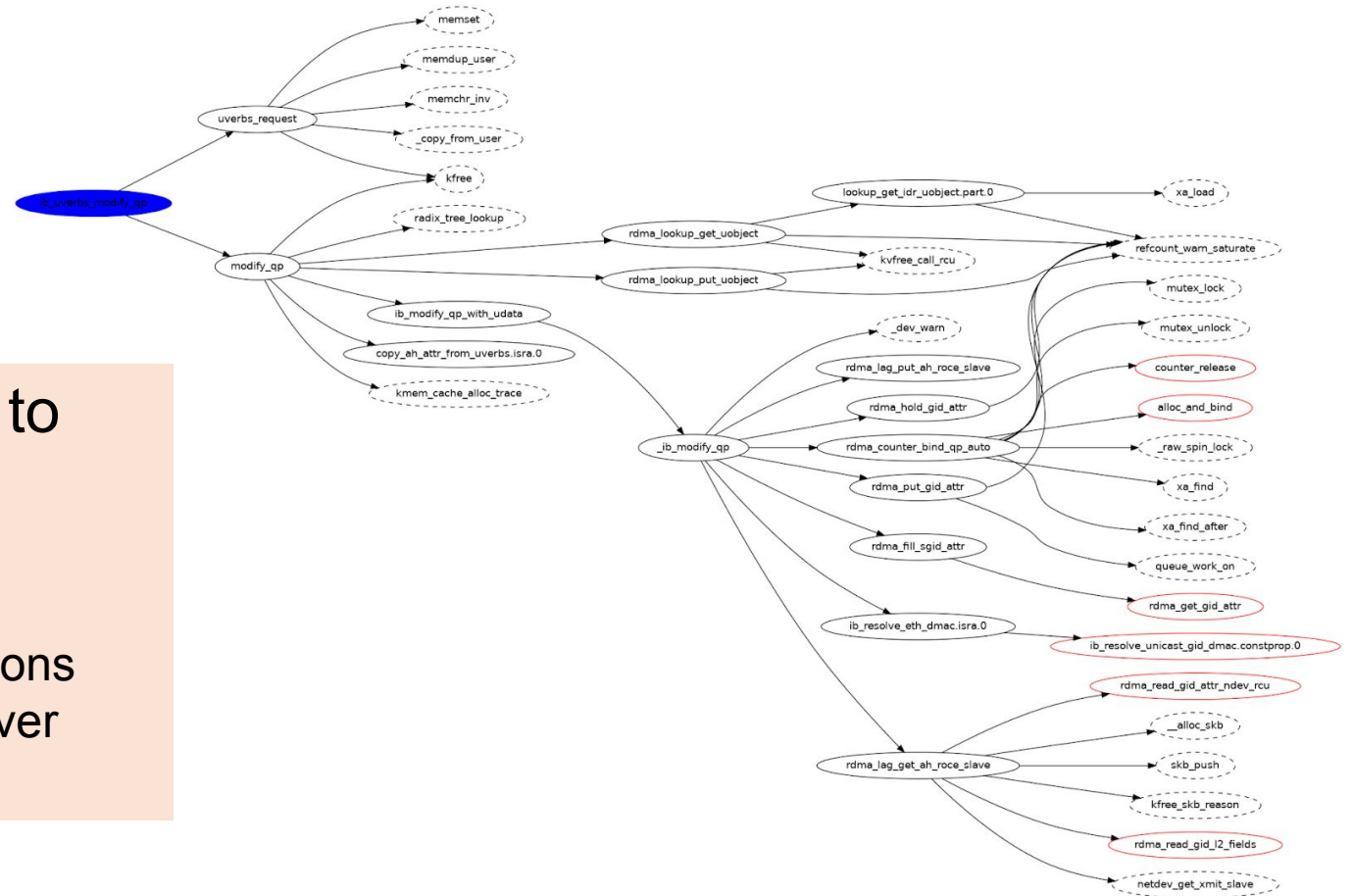


-

Scaling Syscall Tracing for RDMA Backend

Used static analysis to create rules to prune call path.

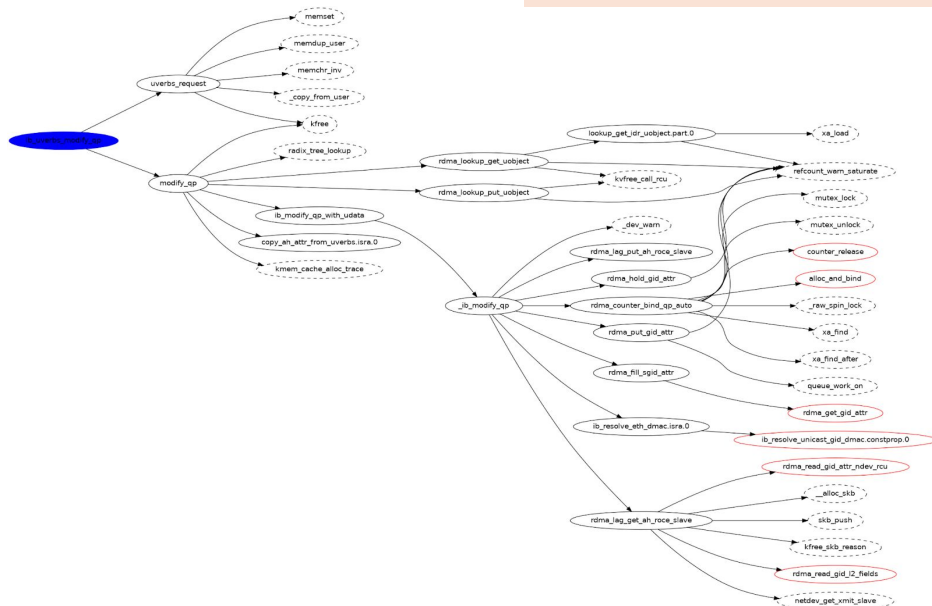
- Avoid reads calls
- Avoid internals of “pure kernel” functions
- Avoid “cleanup” or “reference counter” functions
- Depending on HW vendor, focus on their driver system calls



Scaling Syscall Tracing for RDMA Backend

Used static analysis to create rules to prune call path.

- Avoid reads calls
- Avoid internals of “pure kernel” functions
- Avoid “cleanup” or “reference counter” functions
- Depending on HW vendor, focus on their driver system calls

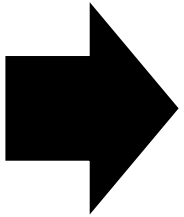


Traced functions for ib_uverbs_modify_qp

- modify_qp,
- rdma_lookup_get_uobject,
- _ib_modify_qp,
- ib_resolve_eth_dmac.isra.0,
- rdma_lag_get_ah_roce_slave,
- rdma_get_gid_attr,
- alloc_and_bind,
- rdma_counter_bind_qp_auto.

Challenges Tracing Syscalls

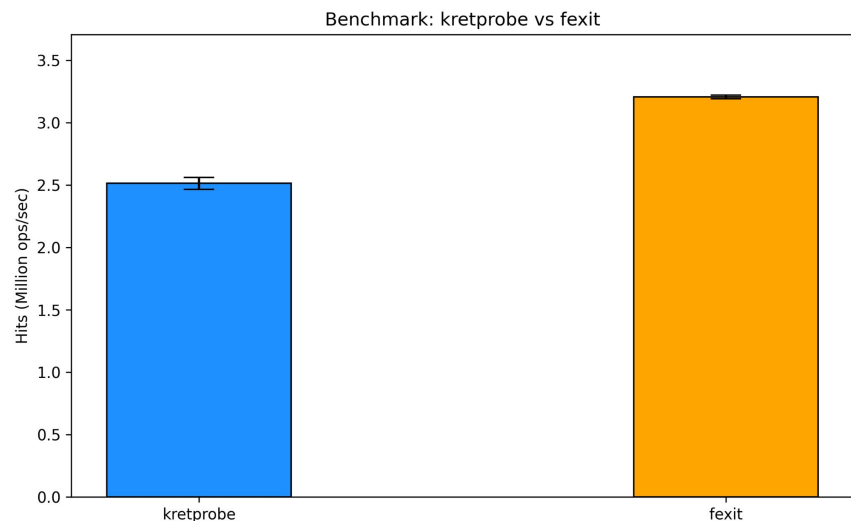
- All RDMA interactions maps to one Syscall
 - Many different paths → significant tracing overheads
 - Heuristic: focus on paths which translate state
 - RDMA hardware state → Software state
- BPF provides many options
 - Selecting optimal is non trivial
- Always on probe is expensive
 - Reactive approach to data collection



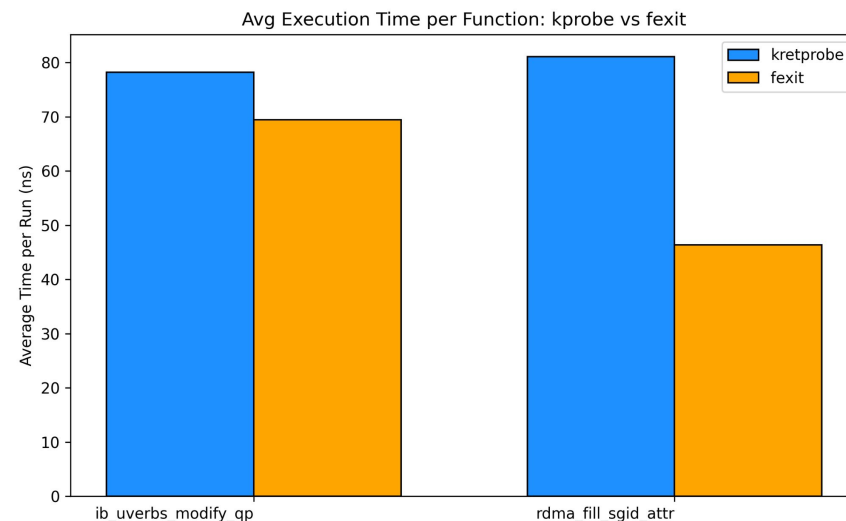
Monitoring Solution Design: Kprobe V Kfunc

- Probe location: KProbe V. kretprobes
 - Kprobes: inserted into any location inside a kernel function.
 - Kretprobes: triggered when a specified function returns.
- Probe type: K{ret}probe V. fentry/fexit
 - Mostly interested in “what happened” along with input and return val
 - Stable Kernel Interface ?
- Goal: Lowest possible overhead
 - Measuring overheads with Kernel benchmark tool
 - Validate with NCCL-test tool

Performance Comparison: Kretprobe v. fexit



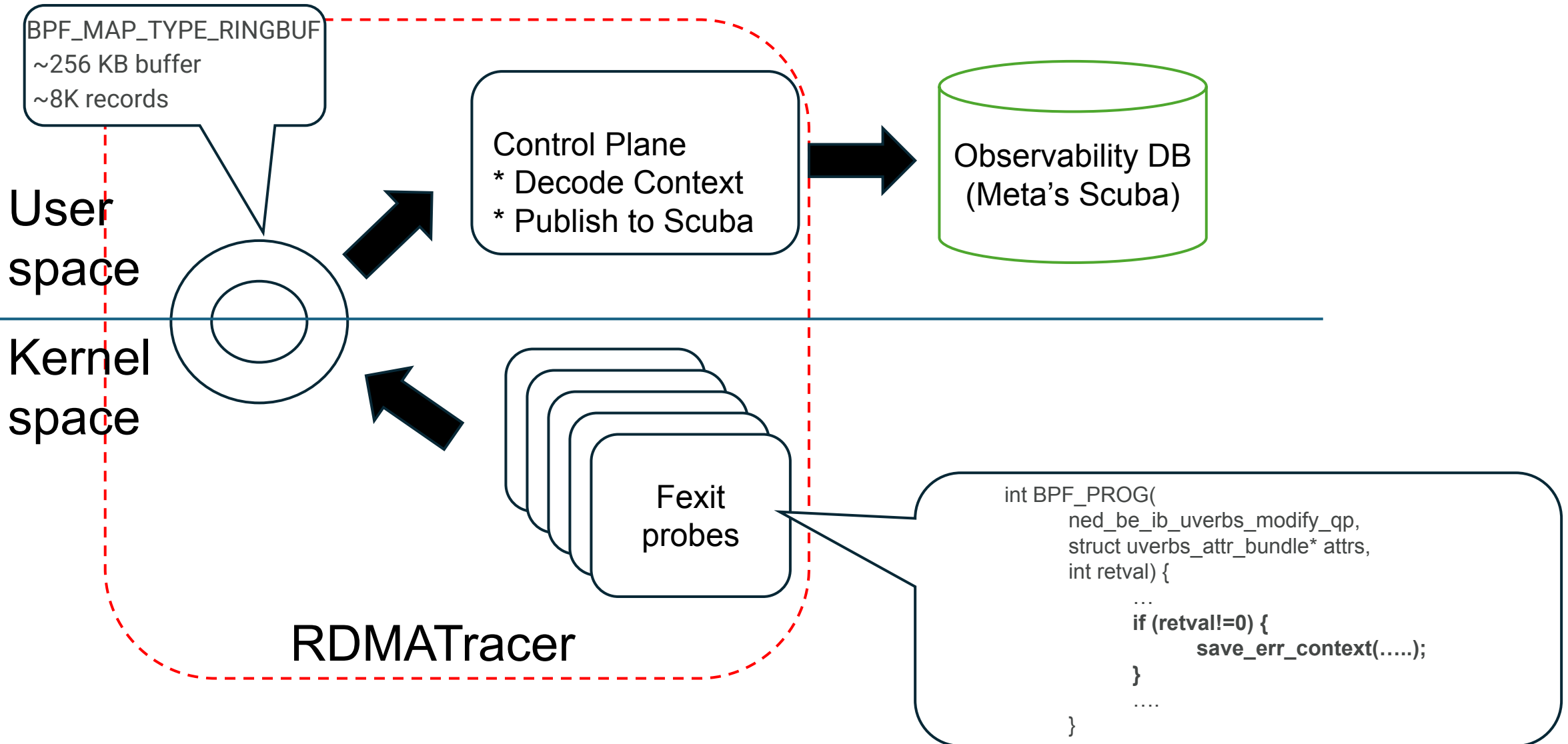
Kernel Benchmark tool



NCCL Benchmark tool

Across all benchmarks: fexit is roughly 72% of kprobe.

Solution Design



Solution Design

- Shared Maps for all fexit progs
 - Ring Buffer to store err_ctx
 - counter map (per-CPU) tracks how many times a syscall has been invoked.

There are three types of counters:

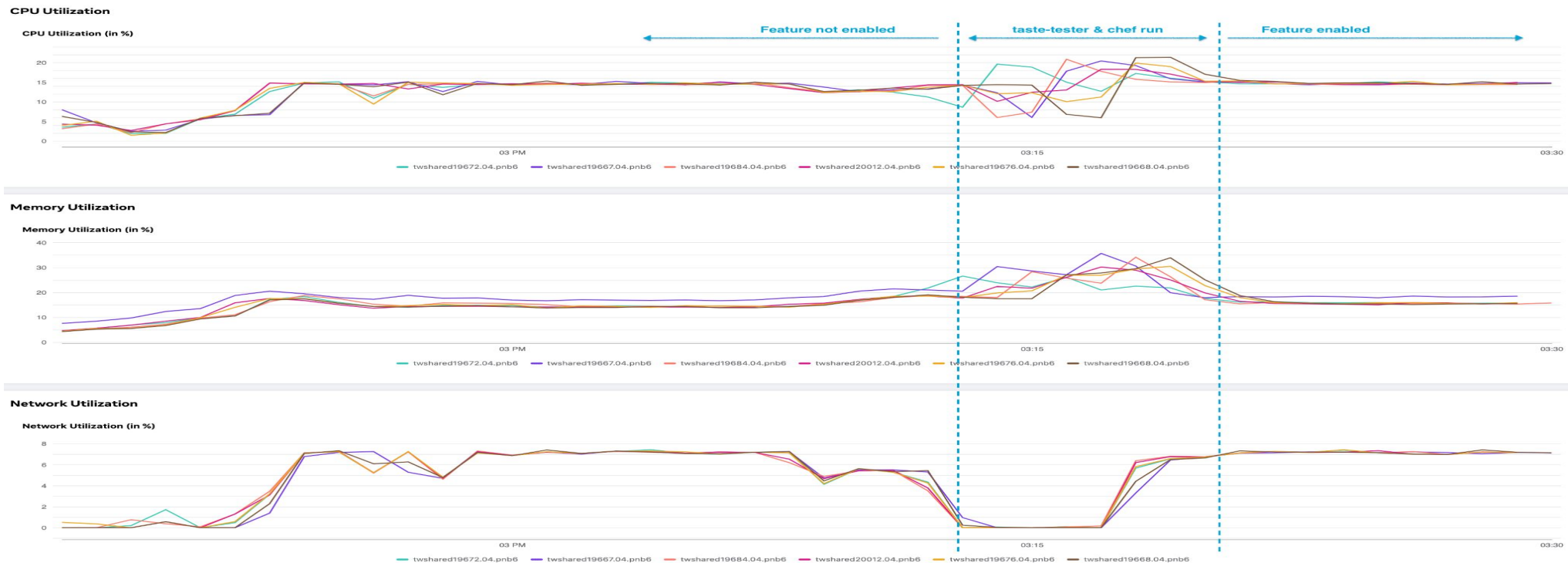
- Type 1: Read as <syscall>, track how many times a syscall has been invoked.
- Type 2: Read as <syscall>_err, track how many times a syscall has been failed.
- Type 3: Read as <syscall>_errno, track the errno returned by a syscall upon its failure.

Sun, Nov 30, 2025 12:19:48 AM (PST)	1	580.82.07	6.13.2-0_fbk7_0_gbc14455e13aa	0	[{"syscall":"mlx5_ib_post_send","errmsg":"Input/output error","errno":5,"process":"nvlsn","timestamp":"11/30/25 00:19:45 ...
Wed, Nov 26, 2025 06:21:04 AM (PST)	1	570.124.06	6.9.0-0_fbk10_0_gc5fa564d33e3	9	[{"syscall":"ib_peer_umem_get","errmsg":"Cannot allocate memory","errno":12,"process":"rdmaC_mlx5_2","timestamp":"11/26/ ...

Again Performance Comparison: With and without BPF

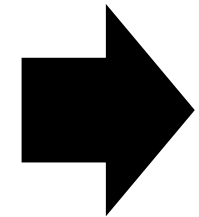
Observations:

- No major change on host metrics before and after feature enabled.
 - bpf_tax - p50 (88.7% less), p99 (98.9% less)
- Workflow QPS not impacted by enabling this feature.



Challenges Tracing Syscalls

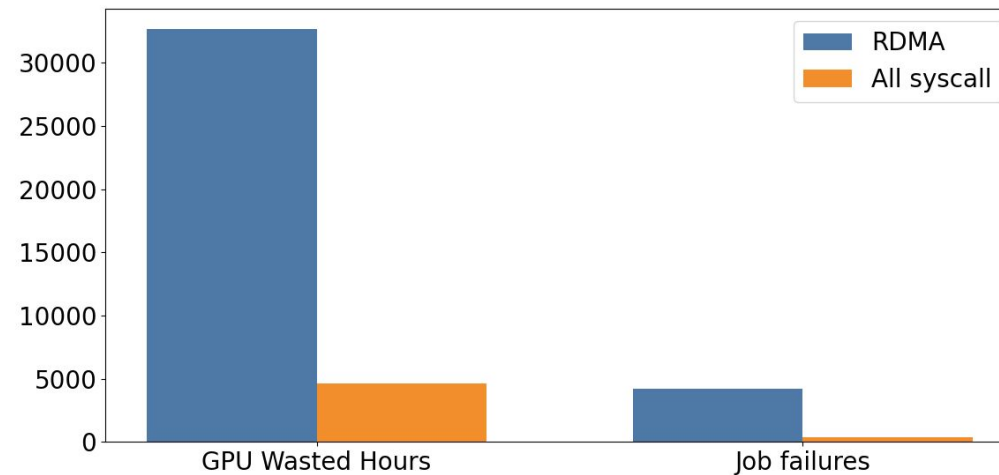
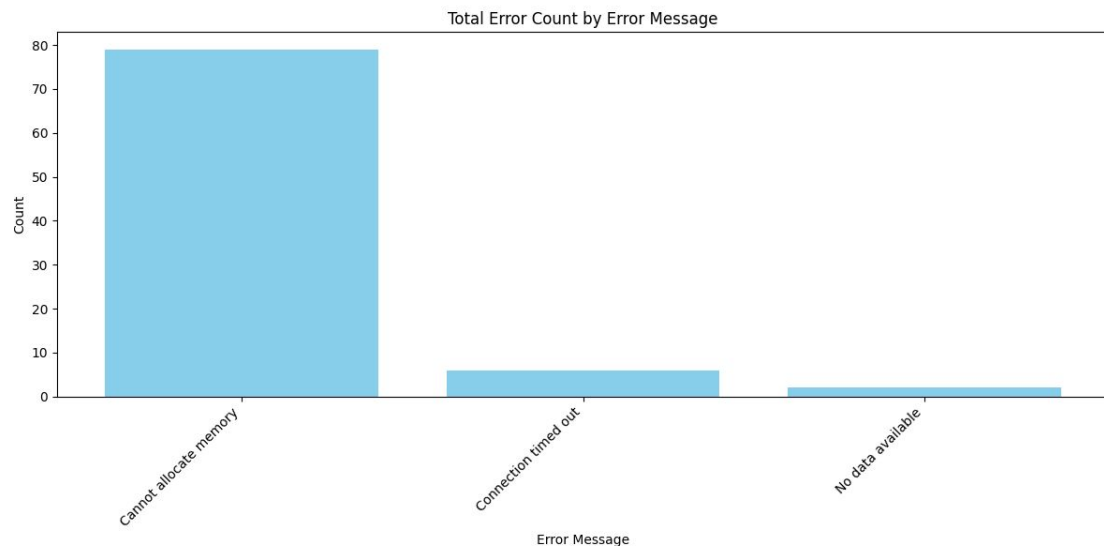
- All RDMA interactions maps to one Syscall
 - Many different paths → significant tracing overheads
 - Heuristic: focus on paths which translate state
 - RDMA hardware state → Software state
- BPF provides many options
 - Selecting optimal is non trivial
- Always on probe is expensive
 - Reactive approach to data collection



Auditing System Workflow

- Periodically capture and store “good” traces:
 - Zero / success as return code
 - “Quick” syscall completion (not greater than 50ms)
 - Challenge: there are also cases when syscall returns “non-critical” error code as expected - e.g. ENOENT for disabled functionality.
- Compare those traces with good ones and help narrow down the issue
 - Looking for bifurcation point to determine which exact spot is failing and producing syscall error.
- Dynamically trigger for a new syscall (or parameters)

RDMATracer in Production Today



- Significantly reduces diagnosis time from 10+ mins to seconds
 - Enables issue auto classification and accounting
 - Eliminates the need to retrieve and process gigabytes of logs
 - Provides a precise timelines for issues and provides dmesgs to aide correlations
 - Helps identifies driver bugs in vendor locked drivers (black box drivers)

Usecase: Triaging blackbox vendor drivers

AI jobs failed on a newly deployed vendor NIC driver (fb v5.19 kernel)

NCCL WARN Call to `ibv_reg_dmabuf_mr` failed with error `Operation not permitted`

RDMATracer exported syscall traces which helped identify:

- A chain of syscalls which returned the error
- A mismatch in the return values for these syscalls
- Triaged error to overflow problem

```
int ib_umem_dmabuf_map_pages:
```



```
return dma_resv_wait_timeout
```



```
long dma_resv_wait_timeout
```

Summary

- RDMA syscall errors have a significant impact at scale
 - 10-20% of Meta's AI job crashes, wasting GPU hours.
- RDMATracer: eBPF-based tool that proactively detects RDMA device driver bugs in real-time.
 - Scalability from Design Choices: minimize # of probes via Static analysis and strategic selection of trace points.
 - Scalability from eBPF Primitive Selected: fexit over Kretprobes.
- RDMATracer streamlines diagnosis and autotranslates issues
 - Diagnosis reduced from 10+ minutes to seconds