The Cost of Causality: High-Frequency, Causal Observability with eBPF Sequencers

Linnix Project
`https://github.com/linnix-os/linnix`

December 2025

Abstract

Production observability faces a fundamental tradeoff: standard eBPF primitives force a choice between throughput and ordering. `BPF_PERF_EVENT_ARRAY` delivers high throughput through per-CPU buffers but sacrifices global event ordering—making it impossible to diagnose race conditions and process lifecycle bugs. `BPF_RINGBUF` provides ordering through a shared ring but collapses under spinlock contention at high core counts.

This paper introduces **Linnix-Sequencer**, a wait-free eBPF primitive that achieves strict global ordering at line rate. Based on the LMAX Disruptor pattern, our approach replaces kernel spinlocks with an atomic serialization point: a cache-line-aligned ticket counter that moves contention from software (OS scheduler) to hardware (CPU mesh interconnect). Combined with BTF-powered raw tracepoints and zero-copy mmap, the sequencer **captures 100% of kernel events** while standard perf buffers drop up to 77% at high core counts—maintaining **zero ordering violations** across 8 to 192 cores on AMD EPYC systems.

1 Introduction: The Observability Uncertainty Principle

Modern distributed systems require precise event ordering to debug race conditions, reconstruct attack timelines, and ensure correctness. Yet the very act of observing high-frequency kernel events introduces a fundamental tradeoff—what we call the **Observability Uncertainty Principle**:

> *High-throughput event capture destroys the ordering required to diagnose the bugs it detects.*

1.1 The Problem

Consider a common production scenario: a process forks, the child executes, and an exit event occurs. A security or debugging tool must capture these events in the exact order they occurred. Any misordering could lead to false positives, missed attacks, or debugging failures.

Standard Linux eBPF primitives force a choice:

Primitive	**Throughput**	**Global Ordering**	**Wait-Free**	**Scalability**
BPF_PERF_EVENT_ARRAY	High	No (per-CPU)	Yes	Linear
BPF_RINGBUF	Medium	Yes	No (spinlock)	Collapses at 64+ co
Linnix-Sequencer	**High**	**Yes**	**Yes**	**Linear**

Table 1: Comparison of eBPF event delivery primitives

1.2 Our Contribution

We present the first wait-free, strictly-ordered eBPF primitive that doesn't sacrifice throughput. Our key insight: **move contention from software (kernel spinlocks) to hardware (atomic CPU operations)**.

2 Background: Why Standard Primitives Fail

2.1 BPF_PERF_EVENT_ARRAY: High Throughput, No Ordering

The perf event array provides one ring buffer per CPU. Each CPU writes events to its own buffer, eliminating cross-core synchronization.

Problem: Userspace receives N independent streams. Merging by timestamp is unreliable—clock skew between cores, NMI delays, and scheduler jitter introduce reordering.

2.2 BPF_RINGBUF: Ordering via Spinlock

The ring buffer uses a kernel spinlock to serialize writes from all CPUs to a single buffer.

Problem: At high core counts (64+), spinlock contention becomes catastrophic. CPUs spend more time waiting for the lock than processing events.

3 Design: LMAX Disruptor in Kernel Space

The Linnix-Sequencer is built on three pillars:

3.1 The Atomic Ticket Counter

Instead of a spinlock, we use an atomic ticket counter placed in a dedicated cache line:

```
#[repr(C, align(64))]  // Cache-line aligned
struct GlobalSequencer {
    value: u64,          // Ticket counter
    _padding: [u8; 56], // Fill to 64 bytes
}
```

Each producer atomically increments this counter to claim a unique sequence number. Hardware atomic operations complete in ~20-50 cycles; spinlock contention can cost thousands of cycles plus context switches.

3.2 The Zero-Copy Ring Buffer

Events are written directly to an mmap-backed BPF array:

- 128MB ring with 1 million 128-byte slots
- BPF_F_MMAPABLE flag enables direct userspace access
- No syscalls on the hot path—consumer reads directly from RAM

3.3 The Consumer Protocol

The consumer advances through slots using sequence number validation. Events are processed in exact sequence order, regardless of which CPU produced them.

4 Implementation Details

4.1 BTF-Powered Raw Tracepoints

Standard tracepoints incur overhead from kernel argument marshaling. BTF raw tracepoints provide direct struct access. Offsets are discovered at runtime via `/sys/kernel/btf/vmlinux`, enabling portable binaries.

4.2 Huge Page Optimization

A 128MB ring spans 32,768 pages at 4KB. Each TLB miss costs ∼100ns. We advise the kernel to use 2MB huge pages via `MADV_HUGEPAGE`, reducing TLB entries from 32K to ∼64.

4.3 Cache-Line Alignment

Each slot spans exactly two cache lines (128 bytes), ensuring producer writes don't cause false sharing on consumer reads.

5 Evaluation

5.1 Test Environment

Property	**Value**
Hardware	AWS c6a.48xlarge (192 vCPUs)
CPU	AMD EPYC 7R13 (Milan)
Kernel	Linux 6.8.0-1044-aws
Workload	stress-ng –fork N –fork-ops 50000
Core Counts	8, 32, 64, 128, 192

Table 2: Test environment configuration

5.2 Throughput Results

We ran both modes against an identical fixed workload (50,000 fork operations = ∼100K kernel events) across increasing core counts.

Cores	**Perf Buffer**	**Sequencer**	**Capture Rate**	**Violations**
8	90,191	100,150	90%	0
32	59,650	100,269	60%	0
64	38,312	100,103	38%	0
128	26,862	100,231	27%	0
192	23,327	100,232	**23%**	**0**

Table 3: Event capture comparison: Perf Buffer vs Sequencer

Key findings:

1. Perf buffer capture degrades from 90% at 8 cores to 23% at 192 cores
2. Sequencer captures 100% at all core counts with no degradation
3. Zero ordering violations across all tests
4. 4.3x capture gap at 192 cores

6 Related Work

Tool	**Buffer Type**	**Ordering**	**Wait-Free**	**Scales 100+ Cores**
Falco	Ringbuf*	Global	No	No
Tetragon	Ringbuf*	Global	No	No
Tracee	Ringbuf*	Global	No	No
BCC/bpftrace	Perf/Ringbuf	Varies	Yes	No
Linnix-Sequencer	**Custom**	**Global**	**Yes**	**Yes**

Table 4: Comparison with existing eBPF observability tools. *Modern tools prefer `BPF_RINGBUF` on Linux 5.8+, falling back to perf buffers on older kernels.

The key difference: `BPF_RINGBUF` provides global ordering but uses a kernel spinlock that causes contention at high core counts. The Linnix-Sequencer replaces the spinlock with an atomic ticket counter, achieving the same ordering guarantees without lock contention.

7 Conclusion

The Linnix-Sequencer demonstrates that wait-free observability with strict global ordering is possible. By applying the LMAX Disruptor pattern to kernel space—replacing spinlocks with atomic ticket counters—we achieve:

- **100% event capture** at all core counts (vs 23-90% for perf buffers)
- **Zero ordering violations** across all tests from 8 to 192 cores
- **No degradation at scale**: Sequencer maintains full capture while perf buffer drops from 90% to 23%

7.1 Future Work

- NUMA-aware allocation to reduce cross-socket traffic
- Batch ticket reservation to amortize atomic overhead
- Hardware timestamp integration using CPU TSC

References

1. LMAX Disruptor: High Performance Alternative to Bounded Queues. Trisha Gee and Martin Thompson, 2011.
2. BPF and XDP Reference Guide. Linux Kernel Documentation.
3. BTF Type Format. `https://www.kernel.org/doc/html/latest/bpf/btf.html`
4. Falco: Cloud-Native Runtime Security. `https://falco.org/`
5. Tetragon: eBPF-based Security Observability. `https://github.com/cilium/tetragon`

Linnix is open source software available at `https://github.com/linnix-os/linnix`