NETDEV 0x1A

One Layer Deeper

From writing Intel Ethernet drivers to the Performance Team at Cloudflare.

Moving beyond the microbenchmark and nano-second measuring era of my career to solving within the 10-layer stack in production.

Jesse Brandeburg
Cloudflare

INTRODUCING

Jesse Brandeburg

Principal Engineer — Performance Team at Cloudflare

About Me

With over 30 years of expertise in Linux drivers, kernel development, and platform performance tuning, I thrive on solving complex system challenges across the entire network stack.

Professional Focus

Help hardware and software go faster, diagnose and summarize in a consumable way the performance of our systems, hardware, NICs, CPUs and memory. Share my knowledge and experience via mentoring and talks. Connect technical leaders.

Career Timeline

2024 — Present: Cloudflare Performance Team
Making the edge network and hardware go faster

1994 — 2024: Intel Corporation
Ethernet product group, Linux driver maintainer, architect, lead.

Contact

jbrandeburg@cloudflare.com

jbrandeb@kernel.org

LinkedIn

jessebrandeburg

CLOUDFLARE

The 10-layer cake

1. **NIC hardware**
   Queues, descriptors, DMA, and offloads
2. **Driver behavior**
   Rings, NAPI, page recycling, interrupt moderation
3. **CPU placement**
   IRQ affinity, RSS, XPS/RFS, NUMA locality
4. **Early packet hooks**
   XDP, AF_XDP, hardware/software steering
5. **Traffic control**
   tc ingress/egress, qdisc, pacing
6. **Firewall & security**
   nftables, netfilter, conntrack, policy BPF
7. **Kernel socket layer**
   Socket buffers, drops, SO_REUSEPORT, *mmsg
8. **Application runtimes**
   Go, Rust, async schedulers, queues, backpressure
9. **Microarchitecture**
   Cachelines, code size, iTLB/dTLB, AVX-512 tradeoffs
10. **Observability**
    Prometheus, counters, missing correlation

The frosting on the outside hides the complexity inside. This talk walks through them.

CLOUDFLARE

Turn everything off and go fast

THE DPDK / XDP-MICROBENCHMARK ERA

"How many Millions of packets/s can one core forward?"

Kill the firewall, and anything else that slows me down. Chase nanoseconds. Understand data structures (pahole!), descriptors and cachelines. Pin things. Disable coalescing or offloads. Measure packet rates and nanoseconds.

- ✓ Rebuild your kernel with tuning
- ✓ Pin IRQs to cores
- ✓ Optimize interrupt coalescing
- ✓ Measure one flow and **many** flows
- ✓ Measure packets per second

HISTORY - INTEL ETHERNET DRIVERS

Years spent writing and tuning Intel Ethernet drivers. The performance playbook was usually "turn everything off and go fast."

BUT...

That knowledge is still useful. However, it took me moving to Cloudflare to realize how much production changed the problem.

REALITY CHECK

CLOUDFLARE

Production keeps the layers on

In production, everything is on because it has to be.

- **Firewalls** — Mandatory for security policy
- **Observability** — Can't debug what you can't see
- **Multi-tenant** — Noisy neighbors cause real pain
- **Conntrack** — Certain types of rules require it
- **Offloads** — Some help, but some **cannot** be used
- **Logging** — Sampling and logging all the time

THE HARD PROBLEM CHANGED

It's no longer just making one machine scream.

It is understanding which layer is quietly spending the budget, hiding the drop, or adding the delay — and these are often stacked problems.

31.4

Tbps

Cloudflare has publicly [1] described mitigating attacks this large. The DoS mitigation layer becomes **very** important.

NEXT UP

Let's walk through some key learnings from production.

[1] https://blog.cloudflare.com/ddos-threat-report-2025-q4/

CLOUDFLARE

Syscalls and socket buffers

```
// One packet per syscall
recvfrom()
recvfrom()
recvfrom()
```

Treating the socket receive queue as a packet warehouse.

```
// Batch the syscalls
recvmmsg() →
```

sendmmsg() and recvmmsg() can be cheap optimizations for a busy app on a busy system.

The socket queue is not infinite

The kernel socket queue is not an infinite holding area. One packet per syscall manufactures latency and loss under load.

High-level languages / Abstractions

High-level languages don't always solve syscall and batching costs — they mostly hide them until production.

CLOUDFLARE

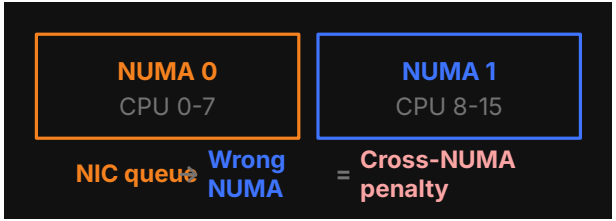
CPU placement: IRQs, RSS, and NUMA

Which CPU handles the interrupt?
IRQ affinity determines which core wakes up for each queue.

Which core runs NAPI?
NAPI polling may run on a different core than you expect. Could threaded_napi be better?

Right NUMA node?
Are descriptor rings and packet buffers on the right NUMA node?

RSS / XPS / RFS?
Receive Side Scaling, Transmit Packet Steering, Receive Flow Steering — all affect where packets land. It's hard to decide when to use these!



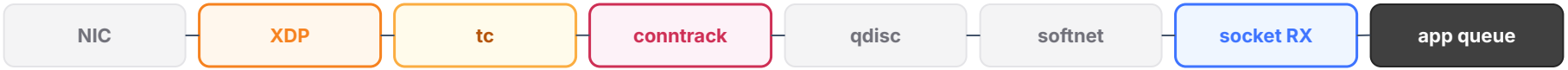
Sounds like sysadmin trivia

Getting them wrong means cross-NUMA memory access and cache thrashing on a box that looks idle.

The interface abstraction doesn't tell you where your packets land in the machine.

CLOUDFLARE

Hooks, firewalls, and places to drop



Packets can disappear at any of these points. Each layer costs something and a drop can hide that cost.

XDP

Can avoid skb allocation entirely.

Fastest drop point in the kernel. Extremely hard to observe.

tc

Sees a different layer than XDP.

Classification and redirect before the stack.

netfilter / conntrack

May be necessary for policy but is expensive.

Per-packet cost adds up over time - (yeah we're working on it).

Socket-layer drops

May not correlate directly with metrics.

Often disconnected from the service-level insights you wish existed.

Warning: Hooks, firewalls, and filtering mechanisms — XDP, tc, netfilter, conntrack, socket filters, eBPF — are extremely powerful but never free.

CLOUDFLARE

Leaky abstractions

TCP OFFLOADS

Mature and predictable

TSO/GSO on TX, GRO on RX. These have mature offload expectations and generally work as expected.

UDP OFFLOADS

Fragile and opt-in

UDP easily loses automatic segmentation/offload behavior. UDP GSO/GRO needs explicit support and preservation across the chosen path.

VLAN ACCELERATION

Helpful until it isn't

VLAN acceleration is helpful until userspace needs the tag or metadata. Hardware sometimes strips what software needs.

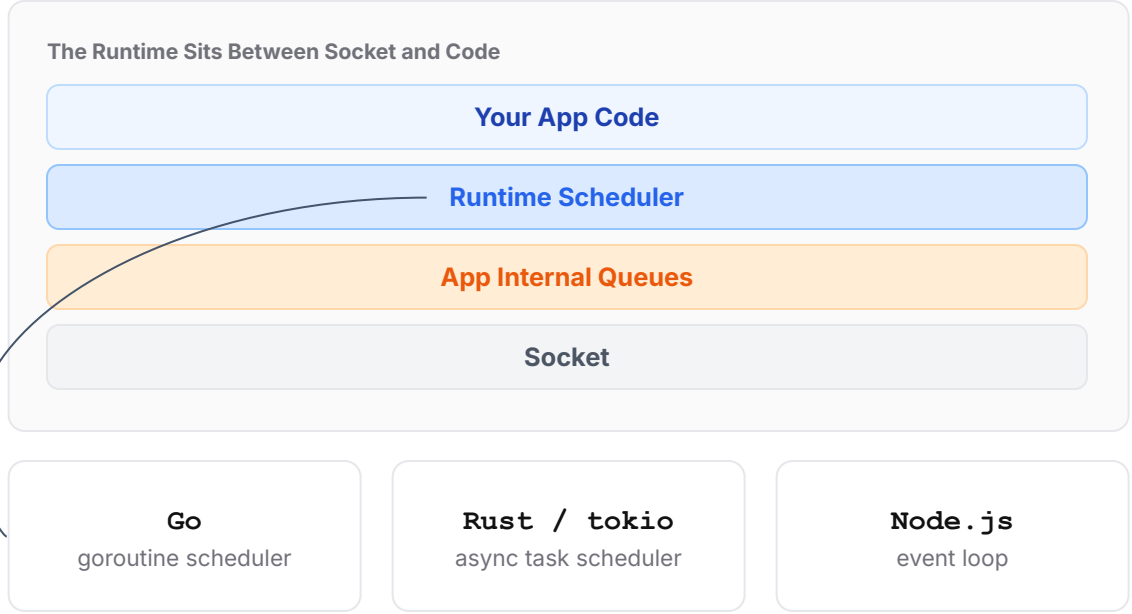
METADATA PATHS

Don't always match

Hardware and kernel metadata paths don't always match what a userspace program expects.

Key Concept: "Packet bytes" and "packet metadata" are not the same thing.

Application runtimes: schedulers and backpressure



"I processed everything I received"

A service can honestly report this while the kernel is dropping packets before the application ever sees them.

Another layer of queues and drops

The application runtime is another layer with its own queues, its own drops, and its own visibility gaps. Sometimes even its own scheduler.

(tokio, I'm looking at you!)

CLOUDFLARE

Below the packet path: cache and TLB

Registers ~1 ns

L1 Cache ~1 ns

L2 Cache ~4 ns

L3 Cache ~12 ns

DRAM ~60 ns

TLB Miss + Page Walk ~100+ ns

The CPU doesn't know what a packet is

It sees memory accesses and branch patterns. Packet processing looks like compute but it's mostly memory access.

WHAT THE CPU IS ACTUALLY CHASING

- Descriptors scattered across cachelines
- sk_buffs and their metadata
- Queue structures and pointers
- Frontend stalls from iTLB/dTLB misses

Cache efficiency, cacheline utilization, frontend stalls, iTLB/dTLB misses — **these determine throughput.**

CLOUDFLARE

Code size vs instruction selection

- **"Less code is faster"**
  True when you're bound by instruction cache, frontend bandwidth, or translations.
- **"Bigger instructions are faster"**
  True when you're bound by instruction count — AVX-512 processes more data per cycle.
- **Both can be wrong**
  If the bottleneck is frontend bandwidth, instruction cache, or translations — neither strategy helps.

It depends on the bottleneck

Sometimes less code is faster. Sometimes bigger AVX-512 code is faster. You have to know whether you're bound by instructions, frontend bandwidth, or translations.

The answer is always "measure first" — but you need to know what to measure.

CLOUDFLARE

Monitoring the abstraction is not monitoring the system

`App`	"I processed everything I received."
`Kernel`	UDP receive buffer overflowed.
`NIC`	Rings are fine.
`qdisc`	Packets waited.
`perf`	CPU is busy missing TLBs.
`Prometheus`	May not expose the layer that failed.

Most engineers have Prometheus, not SSH

Linux has lots of counters, but not always the correlation you want at in an emergency. Prior expectations describe what to monitor — that's necessary, but not enough for debugging the unexpected.

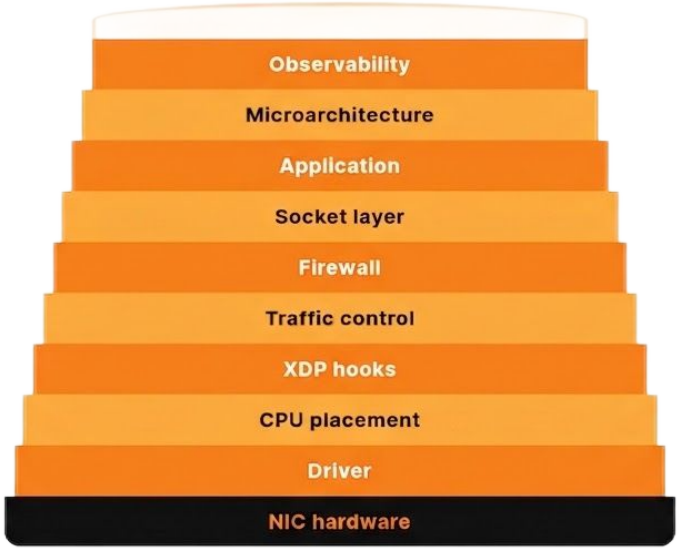
Do both

Performance debugging requires monitoring at multiple layers. Monitor the abstraction and the system underneath it.

CLOUDFLARE

Know one layer deeper

"Knowing one level deeper than the abstraction layer you're developing on makes you able to implement more efficient code."
— Jesper Dangaard Brouer



- Monitor at multiple layers.
- Understand the abstraction layer you're developing on.
- Know the layers and pay attention to the layers under your abstraction of choice.