TECHNICAL DOSSIER // HOW-TO & GUIDES

Linux eBPF XDP vs. DPDK: High-Speed Packet Processing Architecture

Zero-Copy Ring Buffers, Kernel-Bypass Mechanics, and Line-Rate 100GbE Network Telemetry

Published: October 29, 2026 | Distribution: Open Engineering Community | Document ID: LT-2026-B15

EXECUTIVE SUMMARY & MICROARCHITECTURAL SCOPE

This comprehensive technical guide compares Linux eBPF eXpress Data Path (XDP) with the Data Plane Development Kit (DPDK) for multi-gigabit networking. We examine the low-level lifecycle of 64-byte packets on 100GbE interfaces, comparing DPDK's polling mode kernel-bypass with XDP's in-kernel driver-level execution and AF_XDP zero-copy UMEM memory architectures across throughput, latency, security, and CPU utilization metrics.

Networking Architecture	Standard Linux Kernel (sk_buff)	Linux eBPF XDP (Driver Mode)	DPDK (Kernel Bypass PMD)
Packet Ingress Path	Interrupt -> NAPI -> skb alloc -> Stack	Driver RX ring hook (pre-skb) -> Action	NIC DMA -> Userspace Hugepages (Bypass)
Throughput (64B, 100GbE, 1 Core)	1.8 to 2.5 Mpps	24.0 to 28.5 Mpps	29.0 to 34.0 Mpps
CPU Utilization Mode	Interrupt / NAPI adaptive polling	Driver hook / Interrupt / Polling	100% Core Pinning (Spinlock PMD)
Zero-Copy Socket Support	No (Kernel-to-user copy)	Yes (AF_XDP UMEM Ring Buffers)	Yes (Direct DPDK Mempool)
Security & Tooling Isolation	Native iptables/nftables/ethtool	Verified eBPF bytecode safety	No kernel security; complete driver isolation

1. The Inherent Cost of the Standard Linux Network Stack

In standard Linux networking, every incoming packet triggers driver allocations of an sk_buff (socket buffer) structure, which contains over 200 bytes of metadata. Across allocation, DMA mapping, L2/L3 parsing, firewall traversal (netfilter), and context switching into userspace, processing a single 64-byte frame consumes 400 to 800 CPU clock cycles. At 100GbE line rate (148.8 million packets per second), standard kernel overhead overwhelms modern CPUs.

2. The DPDK Kernel-Bypass Paradigm

The Data Plane Development Kit (DPDK) circumvents kernel overhead by completely unbinding the network interface controller (NIC) from the operating system kernel via UIO or VFIO drivers. A userspace Poll Mode Driver (PMD) runs in an infinite spinloop, continuously querying the NIC RX ring buffers without interrupts or context switches. DPDK achieves raw line-rate packet processing but permanently locks 100% of pinned CPU cores at maximum clock frequency.

3. eBPF XDP: In-Kernel Driver-Level Execution

eXpress Data Path (XDP) executes safety-verified eBPF bytecode directly inside the network driver subsystem before the sk_buff metadata structure is ever allocated. XDP programs can drop packets (XDP_DROP), reflect packets out the same interface (XDP_TX), or redirect frames across interfaces (XDP_REDIRECT) within ~25 cycles per frame. This preserves Linux driver telemetry, container integration, and security auditing without kernel-bypass risks.

4. AF_XDP Zero-Copy UMEM Architecture

To deliver line-rate performance directly into userspace applications, Linux provides AF_XDP (Address Family XDP). Utilizing lockless circular ring buffers (Fill Ring, RX Ring, TX Ring, and Completion Ring) mapped to a shared memory pool (UMEM) backed by 2MB Hugepages, AF_XDP enables NIC DMA engines to transfer packet data straight into userspace memory buffers without intermediate memory copies or kernel transitions.

STRATEGIC ENGINEERING TAKEAWAYS & DEPLOYMENT CHECKLIST

• **Throughput Parity:** Native XDP achieves over 85% of DPDK throughput while retaining Linux kernel networking tools, monitoring interfaces, and container namespaces.
• **Power & Resource Efficiency:** AF_XDP supports interrupt-driven busy-polling, releasing CPU cores during low-traffic periods, unlike DPDK's permanent 100% core spinlock.
• **Architectural Selection Guide:** Deploy DPDK for specialized dedicated telco appliances; deploy eBPF XDP for microservices, cloud-native firewalls, DDoS defense, and container routing.