Linux Kernel Network Stack

Dr. Mythili Vutukuru

Device Drivers

Network Interface Card (NIC) -> Network Device Driver
Communicates via Memory Mapped I/O (MMIO)
NIC performs Direct Memory Access (DMA) of network packets
NIC interrupts the MMIO

eBPF

!XDP is one of the BPF hooks!
eBPF programs can be safely embedded into special hook points to add custom functionality to kernel network stack.

Executed when the device driver runs

DMA directly into user space memory (bypassing the kernel space

Can work in <u>Zero-Copy Mode</u> too (hardware support is needed) [**zero-copy** refers to techniques that enable data transfer between memory spaces without requiring the CPU to copy the data]

DPDK

Completely in User Space Application [kernel has no role apart from bypassing]
Heavily optimised with cache, page size optimizations, pre-loaded buffers (unlike fresh loading of sk_buff)

The problem is there can't be specific packet picking - kernel is completely bypassed - none of tcpdump nothing works anymore

Current work on making XDP application as fast as compared to DPDK performance
> With all the kernel bypass mechanisms, we no longer have the only payload - we just receive the RAW PACKETS (raw bytes with all the headers) - kernel no longer removes all the headers, our application program itself has to deal with that!

Memory Access Bottleneck

DRAM speeds have not increased as compared to CPU on network hardware.
We work in nanosecond budget per packet BUT accessing the memory takes few 100s of nanoseconds
Very high Cache miss rate

Direct Cache Access/DDIO

Packet directly into the last Cache L3 [Does not DMA into main memory].
User/kernel space program can access packet quickly from cache.
Intel's DDIO -> mostly enabled on modern server!

Problems -> Not fully usable, effective usage, not easy to write into caches! Many ways in LLC reserved for DDIO
If not processed fast enough, the LLC will be full and will be evicted -> gone to DRAM (write-back traffic from DDIO)