CS60203: Design Optimization of Computing Systems *October 1, 2024*

Topic Name

Instructor: Mainack Mondal *Scribe-By: Ronit Nanwani*

1 Overview

Kernel Bypass refers to techniques that allow network packets to bypass the operating system kernel's networking stack to improve performance. In traditional networking, packets go through the kernel, which handles tasks like buffering, routing, and security. However, this introduces overhead, making it inefficient for high-speed networks. Kernel bypass techniques allow applications to interact directly with the network interface card (NIC), reducing latency and increasing throughput.

2 Protocol Architecture and Layering Recap

1. **Application Layer:** This layer includes a large numbers of protocols implemented usually in software. Examples include ftp, telnet, http, etc.
2. **Transport Layer (L4):** Ensures end-to-end delivery between two applications (not just two machines). TCP and UDP are well known protocols in this layer.
3. **Network Layer (L3):** Primary function of this layer is routing - sending packets from source machine to destination machine when they are not on the same network. IP is the most well known protocol in this layer.
4. **Data Link Layer (L2):** This layer is responsible for communication between devices on the same network (subnet). Ethernet is the most common protocol in this layer.
5. **Physical Layer:** Deals with the transmission of raw bitstream. It is basically the interface between the transmission device and the transmission medium.

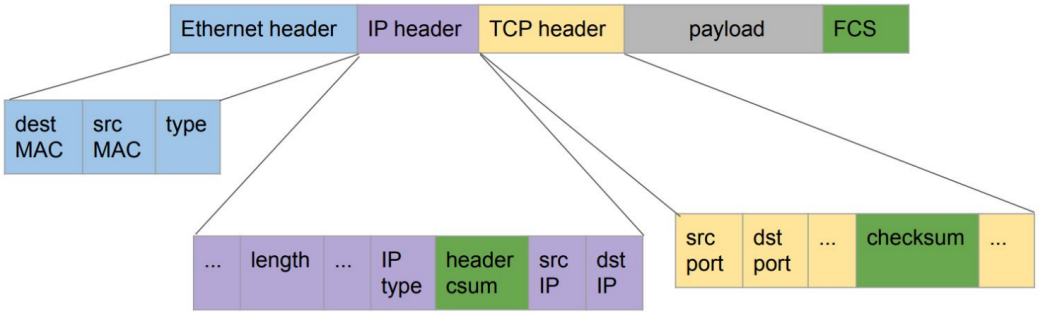

Figure 1: Packet Contents

3 The Linux Network Stack

This section covers the journey of a packet from being received at the NIC till it is delivered to the corresponding application. For our ease of understanding lets breakdown the entire process into various steps.

3.1 Packet Arrives at NIC

- Packet is received at the NIC and stored in the hardware RX queue (shown in Figure 2).
- Destination MAC and Ethernet checksum is verified.
- DMA (Direct Memory Access - minimizes CPU intervention for data transfer) the packet to the packet buffer and subsequently NIC generates an interrupt.

  The TX and RX ring buffers (shown in Figure 2) basically are circular queues in which each slot stores mainly the length of packet, its address in the packet buffer and some metadata.

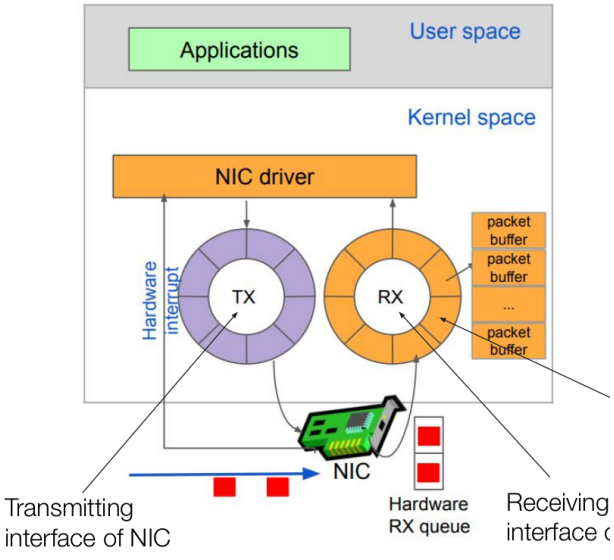

Figure 2: Hardware RX queue

3.2 Interrupt handling

- CPU interrupts the process in execution and a context switch takes place.
- Mode bit change: **user mode to kernel mode** (done by hardware) and the corresponding Interrupt service routine is called to acknowledge the interrupt and schedule a software interrupt (soft-irq), usually the NET_RX_SOFTIRQ.
- Mode bit change: **kernel mode to user mode**. CPU continues with the interrupted process or other user processes.

3.3 Handling soft-irq

- CPU initiates further processing whenever it is free (depends when the soft-irq is scheduled by the scheduling algorithm).
- Mode bit change: **user mode to kernel mode**
- NIC driver dynamically allocates the sk buff structure (shown in Figure 3), which is basically an in memory data structure that contains pointer to packer header, payload and other packet related info.
- Ethernet header is removed and the sk buff struct is passed up the network stack for further processing

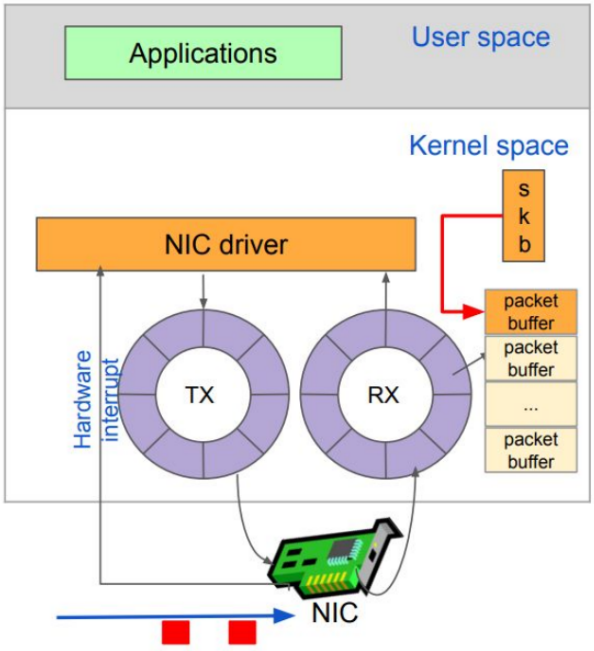

Figure 3: sk buff

3.4 L3/L4 processing

- Common processing of both layers include checksum verification and header removal
- L3 specific processing includes route lookup, combining fragmented packets and calling the corresponding L4 protocol
- L4 specific processing includes handling the TCP state machine, enqueuing the sk buff struct to socket read queue (yet another kernel data structure) and signalling the socket (in case the socket is blocked on a recv() call, it needs to wake up).
- Mode bit change: **kernel mode to user mode**

3.5 Application issues read() call

- Mode bit change: **user mode to kernel mode**
- Dequeue sk buff from the socket RQ.
- Copy packet from the packet buffer to application buffer (user space). Done through copy_to_user() syscall this time instead of DMA.
- Release skbuff as it was dynamically allocated.
- Mode bit change: **kernel mode to user mode** and return back to application.

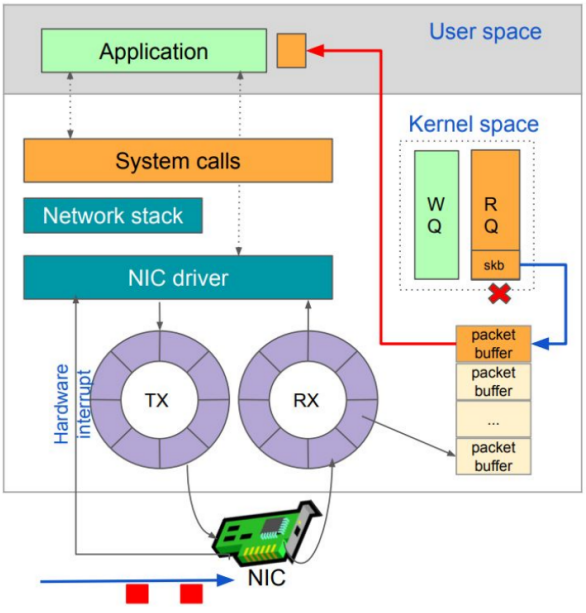

Figure 4: socket RQ and WQ for each TCP connection and the application buffer

4 Packet Processing Overheads

- Several context switches between kernel space and user space (highlighted in the above section as Mode bit change). This pollutes the CPU cache.
- Packet copy from packet buffer to application buffer
- Dynamic allocation of sk buff and per packet interrupt overhead

5 Kernel Bypass Techniques

To overcome the above challenges we can shift the entire packet processing in the user space. This reduces the overhead of packet copy and also context switches. Also pre-allocation of memory for sk buff helps.

5.1 User level packet I/O libraries - manage processing till L2 layer

5.1.1 Dataplane Development Kit - DPDK

- **User-space packet processing**: DPDK allows applications to bypass the kernel's networking stack by providing libraries for fast packet processing directly in user space.
- **Poll-mode drivers (PMD)**: It uses poll-mode drivers that interact directly with the network interface card (NIC) in a busy-wait loop, eliminating interrupts and reducing latency for high speed network traffic.
- **Hugepage memory**: DPDK uses pre-allocated large memory pages (hugepages) to minimize TLB misses, enhancing memory access speeds and optimizing buffer management for high throughput.

5.1.2 Netmap

- **Memory-mapped buffers**: It uses pre-allocated, memory-mapped buffers shared between user space and the NIC driver (kernel space), allowing fast packet transfers without system call overhead.
- **Efficient I/O rings**: Netmap implements circular buffer rings (netmap rings) for efficient packet input and output, reducing the overhead of context switching and packet copies.
- **Adaptive Polling and Interrupts**: If the system needs to conserve CPU resources, Netmap can reduce polling frequency and rely on interrupts to trigger packet processing.

5.2 mTCP - Userspace network stack till L4 processing

- **User-space TCP stack**: mTCP is a high-performance, user-space TCP/IP stack to provide faster packet processing. It extends the PacketShader I/O engine for user level packet I/O and implements its own L3/L4 processing stack.
- **Event-driven architecture**: mTCP leverages an event-driven architecture to efficiently manage network events like packet arrivals, eliminating the need for expensive kernel interactions.
- **Thread-per-application model**: Each application thread spawns a mTCP thread to ensure handling of network traffic independently, reducing lock contention, improving concurrency and also improving throughput by batched processing.

References

[1] Design Optimization of Computing Systems Course Slides

[2] Kernel-NIC Interaction, Understanding how OS uses the modern NIC

[3] DPDK Guide, DPDK

[4] Netmap Paper, Netmap

[5] mTCP Paper, mTCP