The Path of a Packet Through the Linux Kernel

Introduction

- Networking is fundamental to modern computing, and the Linux network stack is a crucial component, especially in servers. Understanding the intricacies of packet processing in Linux is essential for performance optimization, security analysis, debugging, and network observability. This document outlines the ingress (receiving) and egress (sending) packet paths in the Linux kernel, focusing on TCP/IPv4 and UDP/IPv4.

Background

- The Linux networking stack comprises multiple layers, from the socket (user space) down to the network card (driver). Key layers include TCP/UDP, IPv4, and Ethernet. Packets are stored in kernel structures called `sk_buff`, which efficiently manage packet data and metadata, such as protocol headers, timestamps, routing information, and buffer pointers.

Egress Path (Sending Packets)

The egress path involves constructing protocol headers, pushing them to `sk_buff` structures, and sending them out via the NIC.

Key stages include:

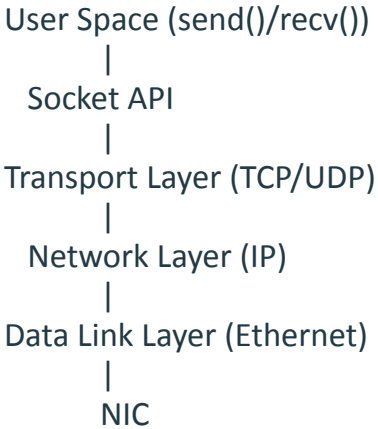
1. **Socket Layer:**
   - User space applications use system calls like `write()` or `sendto()` to send data to a socket.
   - This triggers a transition from user space to kernel space.
2. **Transport Layer (TCP & UDP):**
   - **TCP (`tcp_sendmsg()`):** Manages connections, segmentation, flow control, and retransmission.
   - **UDP (`udp_sendmsg()`):** Provides connectionless datagram service.
3. **IP Layer (`ip_queue_xmit()`):**
   - Determines routing, builds the IP header, and handles fragmentation.
4. **Ethernet Layer (`dev_queue_xmit()`):**
   - Queues the packet, adds the Ethernet header, and hands it off to the NIC.
5. **NIC (Network Interface Card):**
   - Transmits the packet onto the network.
6. **Netfilter Hooks:**
   - Packet may be processed by firewall/NAT rules before transmission.

Ingress Path (Receiving Packets)

The ingress path involves receiving packets from the NIC, processing headers, and passing the data to the user space application.

Key stages include:

1. **NIC (Network Interface Card):**
   - Receives the packet and transfers it to system memory.
   - Uses **NAPI** (New API) to reduce interrupt load via polling.
2. **Ethernet Layer (`netif_receive_skb()`):**
   - Performs sanity checks and removes the Ethernet header.
3. **IP Layer (`ip_rcv()`):**
   - Validates IP header, performs routing decisions, reassembles fragments.
4. **Transport Layer (TCP & UDP):**
   - **TCP (`tcp_v4_rcv()`):** Handles sequence tracking, acknowledgments, and buffering.
   - **UDP (`udp_rcv()`):** Delivers datagram to socket buffers.
5. **Socket Layer:**
   - User space applications read the data using `read()`, `recv()`, etc.
6. **Netfilter Hooks:**
   - Inbound packets may be filtered, modified, or dropped based on firewall rules.


User Space (send()/recv())
Socket API
Transport Layer (TCP/UDP)
Network Layer (IP)
Data Link Layer (Ethernet)
NIC