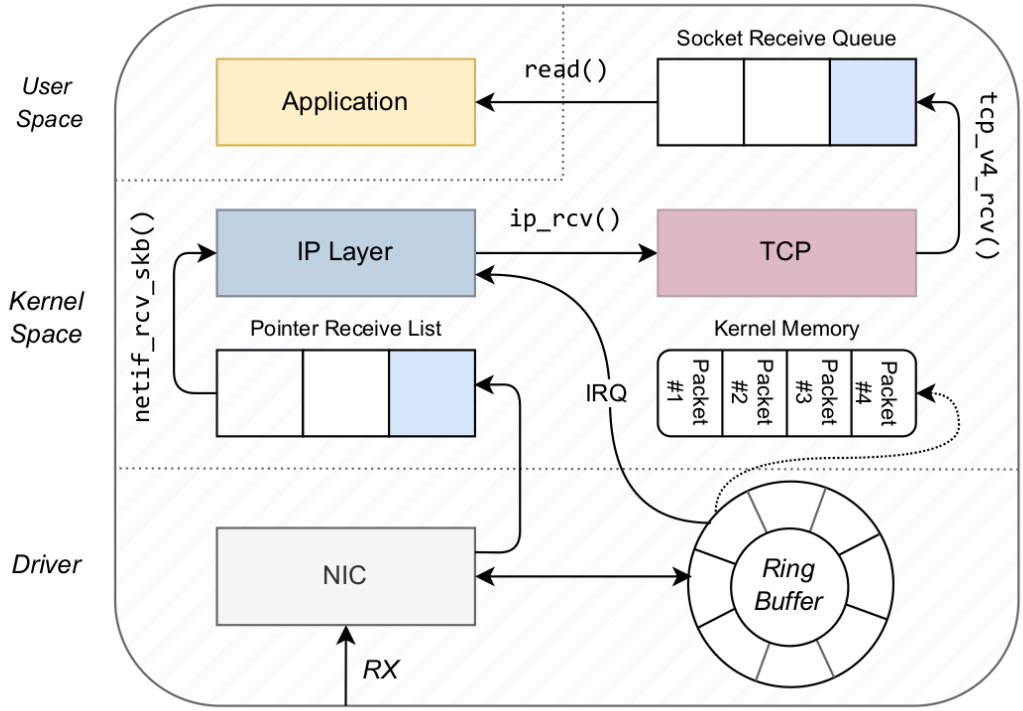
The Journey of a Network Packet Through the Linux Kernel

Modern computer networks rely on complex software systems to manage the flow of data packets between devices. At the heart of this process in Linux-based systems lies the kernel's networking stack, which handles both incoming (ingress) and outgoing (egress) packets. This report breaks down the entire pipeline, explaining each stage and function in simple terms.

Understanding the Ingress Path (Receiving Packets)



When a packet arrives at a computer's **Network Interface Card (NIC)**, it begins a multi-step journey through the Linux kernel before reaching its destination application. The ingress path involves four main layers:

1. **Ethernet Layer**: The NIC detects incoming data, copies it to memory, and notifies the kernel.

2. **IP Layer**: The kernel verifies the packet's integrity, checks if it's destined for the local machine, and strips away the IP header.

3. **Transport Layer (TCP/UDP)**: The kernel determines whether the packet uses TCP (reliable, connection-based) or UDP (fast, connectionless) and processes it accordingly.

4. **Socket Layer**: The packet is delivered to the target application via a socket, a software endpoint for communication.

This path ensures packets are validated, routed correctly, and handed to the right program1.

Understanding the Egress Path (Sending Packets)

When an application sends data (e.g., a web request), the packet follows the reverse path:

1. **Socket Layer**: The application writes data to a socket (like a file).

2. **Transport Layer**: TCP or UDP adds headers (e.g., port numbers) to the data.

3. **IP Layer**: The kernel adds source/destination IP addresses and routes the packet.

4. **Ethernet Layer**: The NIC transmits the packet over the physical network.

Each layer adds critical information, ensuring the packet reaches its destination1.

Detailed Pipeline: Stages and Functions

Ingress Path

Stage 1: Ethernet Layer (Hardware to Kernel)

1. **NIC Reception**: The NIC receives electrical signals, converts them into binary data, and uses **Direct Memory Access (DMA)** to copy the packet into the kernel's memory.

2. **Interrupt Handling**: The NIC triggers a hardware interrupt, alerting the kernel to process the new packet.

3. **sk_buff Allocation**: The kernel creates an sk_buff structure—a "container" storing the packet data and metadata (e.g., protocol type, interface).

4. **Ethernet Header Processing**:

- **netif_receive_skb()**: Checks if the packet is for the local machine or needs forwarding.
- **MAC Address Filtering**: Drops packets not meant for the device.
- **VLAN Handling**: Processes virtual LAN tags if present1.

Stage 2: IP Layer

1. **ip_rcv()**: Validates the IP header (checksum, version, length).
2. **Netfilter Hook (NF_INET_PRE_ROUTING)**: Applies firewall rules (e.g., iptables) to filter or modify packets.
3. **Routing Decision**:
   - **ip_route_input_noref()**: Determines if the packet is for the local machine, another device (forwarding), or multicast.
   - **ip_local_deliver()**: If the packet is for the local machine, it proceeds up the stack.
4. **IP Defragmentation**: If the packet was split into fragments, ip_defrag() reassembles them1.

Stage 3: Transport Layer (TCP/UDP)

1. **Protocol Identification**: The kernel checks the IP header to see if the packet uses TCP or UDP.
   - **TCP**:
     - **tcp_v4_rcv()**: Validates the TCP header (checksum, sequence numbers).
     - **__inet_lookup_skb()**: Finds the associated socket based on port and IP.
     - **tcp_rcv_established()**: Updates the TCP state machine (e.g., acknowledges receipt).
   - **UDP**:
     - **udp_rcv()**: Validates the UDP header and checksum.
     - **__skb_recv_udp()**: Matches the packet to the correct socket.

2. **Socket Queueing**: The packet is added to the socket's receive queue, waiting for the application to read it1.

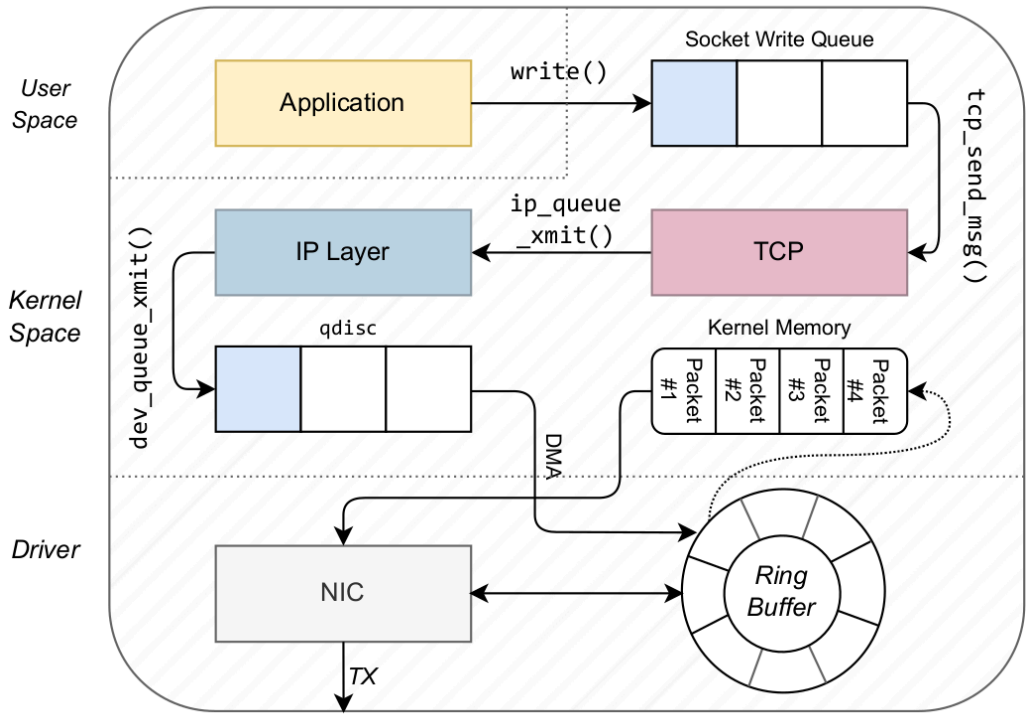
Stage 4: Socket Layer

1. **sys_recv()**: The application calls a system call (e.g., read()) to retrieve data from the socket.

2. **Security Checks**: Linux Security Modules (e.g., SELinux) verify the application has permission to access the data.

3. **Data Copy**: The kernel copies the packet's payload from the sk_buff to the application's memory1.

Egress Path (Step-by-Step)



Stage 1: Socket Layer

1. **Application Write**: The application sends data using a system call like write() or sendto().

2. **Socket Creation**: If no socket exists, one is created with parameters like IP version (IPv4) and protocol (TCP/UDP).

3. **sock_sendmsg()**: Manages socket metadata (e.g., process ID) and passes data to the transport layer1.

Stage 2: Transport Layer (TCP/UDP)

1. **TCP Processing**:
    - **tcp_sendmsg()**: Breaks data into segments, adds TCP headers (ports, sequence numbers), and enqueues them in the socket's write buffer.
    - **Congestion Control**: Adjusts transmission speed based on network conditions.
2. **UDP Processing**:
    - **udp_sendmsg()**: Adds UDP headers and passes packets directly to the IP layer (no retries or congestion control).
3. **Header Construction**: The transport_header field in sk_buff is set to point to the TCP/UDP header1.

Stage 3: IP Layer

1. **Routing**:
    - **__ip_queue_xmit()**: Uses the routing table to determine the next hop (router or destination).
    - **FIB Lookup**: Checks the Forwarding Information Base for the best route.
2. **IP Header Construction**: Adds source/destination IP addresses and sets the network_header field.
3. **Netfilter Hook (NF_INET_LOCAL_OUT)**: Applies firewall rules before transmission.

4. **Fragmentation**: If the packet exceeds the network's MTU (Maximum Transmission Unit), ip_fragment() splits it1.

Stage 4: Ethernet Layer

1. **MAC Address Resolution**:
    - **neigh_resolve_output()**: Uses ARP to find the MAC address of the next hop (router or destination).
2. **Ethernet Header Construction**: Adds source/destination MAC addresses.
3. **Queuing Discipline (QDisc)**:
    - **dev_queue_xmit()**: Enqueues packets in a traffic-shaping queue (e.g., prioritizing VoIP packets).
    - **__qdisc_run()**: Dequeues packets and sends them to the NIC's transmit buffer.
4. **DMA Transfer**: The NIC reads the packet from memory and transmits it over the network1.

Key Functions Simplified

Ingress Functions

- **ip_rcv()**: Validates IP headers and routes packets.
- **tcp_v4_rcv()**: Manages TCP connections and ensures data integrity.
- **udp_rcv()**: Quickly delivers UDP datagrams without error recovery.

Egress Functions

- **tcp_sendmsg()**: Splits data into TCP segments and manages retransmissions.
- **udp_sendmsg()**: Packages data into UDP datagrams for fast transmission.
- **__ip_queue_xmit()**: Routes packets using the Linux kernel's routing tables.