GRP-13:
ADITYA DIXIT :22030
KANAV MEENA :23266
PRANJAL BHARTI:21080
—-------------------------ASSIGNMENT–2---------------------------

CHANGES IN e1000_main.c

- **Added two helper functions (`udp_to_icmp_if_udpping` and `icmp_to_udp_if_udpping`) above `e1000_receive_skb`.**

```
// Helper: convert UDP packet to ICMP if it's a UDP-Ping
static void udp_to_icmp_if_udpping(struct sk_buff *skb)
{
    struct iphdr *iph = ip_hdr(skb);
    struct udphdr *udph;
    struct icmphdr *icmph;
    unsigned char *payload;
    u16 src_port;

    if (!iph || iph->protocol != IPPROTO_UDP)
        return;

    udph = udp_hdr(skb);
    src_port = ntohs(udph->source);

    // Replace UDP header with ICMP header
    icmph = (struct icmphdr *)udph;
    icmph->type = ICMP_ECHO;
    icmph->code = 0;
    icmph->un.echo.id = htons(src_port);
    icmph->un.echo.sequence = htons(1);
    icmph->checksum = 0;

    payload = (unsigned char *)(icmph + 1);

    icmph->checksum = ip_compute_csum((void *)icmph, skb->len - ip_hdrlen(skb));
    iph->protocol = IPPROTO_ICMP;
}

// Helper: convert ICMP reply back to UDP
static void icmp_to_udp_if_udpping(struct sk_buff *skb)
{
    struct iphdr *iph = ip_hdr(skb);
    struct icmphdr *icmph;
    struct udphdr *udph;

    if (!iph || iph->protocol != IPPROTO_ICMP)
        return;

    icmph = (struct icmphdr *)(skb_network_header(skb) + ip_hdrlen(skb));

    if (icmph->type != ICMP_ECHOREPLY)
        return;

    // Replace ICMP header with UDP header
    udph = (struct udphdr *)icmph;
    udph->source = icmph->un.echo.id;
    udph->dest   = icmph->un.echo.id;
    udph->len    = htons(8 + skb->len - ip_hdrlen(skb) - sizeof(struct icmphdr));
    udph->check  = 0;

    iph->protocol = IPPROTO_UDP;
}
```

- Modified **`e1000_receive_skb`** to call these helpers before `napi_gro_receive()`.

Design of e1000_main.c

Provides the **glue** between the **hardware** (via registers, descriptors) and the **Linux networking stack**.

Defines the **adapter structure**, initializes hardware, manages packet **transmit/receive**, and integrates with **ethtool**, **net_device**, and **NAPI**.

Major Components in `e1000_main.c`

1. Driver + PCI Registration

- Functions like `e1000_probe()` and `e1000_remove()` register/unregister the driver with the Linux PCI subsystem.
- They allocate the `e1000_adapter` (driver state), request IRQs, and hook into the Linux **net_device** structure.

Key APIs:

- `pci_register_driver()`
- `register_netdev()`

2. Adapter / net_device Setup

- The driver uses a structure `struct e1000_adapter` to store:
  - Pointers to rings (TX, RX descriptors)
  - Hardware-specific info
  - NAPI context
  - Statistics
- Functions like `e1000_open()` and `e1000_close()` start/stop the NIC:
  - Allocate rings
  - Enable interrupts
  - Start DMA engines

3. Transmit Path (TX)

- `e1000_xmit_frame()` is the **entry point** when Linux sends a packet.
- It:
  - Maps `skb` into DMA
  - Fills a TX descriptor
  - Notifies the NIC to send the packet

4. Receive Path (RX)

- The NIC writes incoming packets into RX rings (DMA).
- An interrupt/NAPI poll calls driver RX handlers.

- In `e1000_main.c`, the **core helper** is:
  static void e1000_receive_skb(struct e1000_adapter *adapter, u8 status,

  __le16 vlan, struct sk_buff *skb);

This function:

1. Classifies packet protocol (`eth_type_trans`)
2. Attaches VLAN tags if present.
3. Passes packet to the Linux stack via `napi_gro_receive()`

5. Interrupt Handling & NAPI

- Hardware signals interrupts when packets arrive or TX completes.
- Driver uses **NAPI** (`napi_poll`) to batch process packets efficiently.
- Functions like `e1000_clean_rx_irq()` and `e1000_clean_tx_irq()` do the heavy lifting.

6. Configuration & ethtool

- Driver supports settings via `ethtool` (speed, duplex, stats).
- Functions like `e1000_get_settings()` and `e1000_set_settings()` integrate with the Linux admin tools.

module_init(e1000_init_module);
module_exit(e1000_exit_module);

This ensures the driver registers itself with the kernel when `insmod`'d and cleans up on `rmmod`.

OUR CHANGES IN THE DESIGN:
modified the Receive Path (`e1000_receive_skb`).

This is the place to insert packet transformations, because every inbound packet flows through here before reaching the Linux stack.

If a packet is UDP "ping," it gets rewritten into ICMP.

If an ICMP reply arrives, it gets rewritten into UDP.