Linux Kernel Networking: Implementation and Theory

By Rami Rosen

About this book[1]

Clear search☒Result **2** of **6** in this book for **ip_call_ra_chain**- Order by: relevance | **pagesrelevance** | pages- ‹ PreviousNext ›View all

```
too_many_hops:
    /* Tell the sender its packet died... */
    IP_INC_STATS_BH(dev_net(skb_dst(skb)->dev), IPSTATS_MIB_INHDRERRORS);
    icmp_send(skb, ICMP_TIME_EXCEEDED, ICMP_EXC_TTL, 0);
    . . .
```

Now a check is performed if both the strict route flag (`is_strictroute`) is set and the `rt_uses_gateway` flag is set; in such a case, strict routing cannot be applied, and a "Destination Unreachable" ICMPv4 message with "Strict Routing Failed" code is sent back:

```
    rt = skb_rtable(skb);

    if (opt->is_strictroute && rt->rt_uses_gateway)
        goto sr_failed;
    . . .
sr_failed:
    icmp_send(skb, ICMP_DEST_UNREACH, ICMP_SR_FAILED, 0);
    goto drop;
    . . .
```

Now a check is performed to see whether the length of the packet is larger than the outgoing device MTU. If it is, that means the packet is not permitted to be sent as it is. Another check is performed to see whether the DF (Don't Fragment) field in the IPv4 header is set and whether the `local_df` flag in the SKB is not set. If these conditions are met, it means that when the packet reaches the `ip_output()` method, it will not be fragmented with the `ip_fragment()` method. This means the packet cannot be sent as is, and it also cannot be fragmented; so a destination unreachable ICMPv4 message with "Fragmentation Needed" code is sent back, the packet is dropped, and the statistics (IPSTATS_MIB_FRAGFAILS) are updated:

```
  if (unlikely(skb->len > dst_mtu(&rt->dst) &&
      !skb_is_gso(skb) && (ip_hdr(skb)->frag_off & htons(IP_DF)))
        && !skb->local_df) {
  IP_INC_STATS(dev_net(rt->dst.dev), IPSTATS_MIB_FRAGFAILS);
  icmp_send(skb, ICMP_DEST_UNREACH, ICMP_FRAG_NEEDED,
          htonl(dst_mtu(&rt->dst)));
  goto drop;    }
```

Because the `ttl` and checksum of the IPv4 header are going to be changed, a copy of the SKB should be kept:

```
   /* We are about to mangle packet. Copy it! */
    if (skb_cow(skb, LL_RESERVED_SPACE(rt->dst.dev)+rt->dst.header_len))
            goto drop;
    iph = ip_hdr(skb);
```

As mentioned earlier, each node that forwards the packet should decrease the `ttl`. As a result of the `ttl` change, the checksum is also updated accordingly in the `ip_decrease_ttl()` method:

```
    /* Decrease ttl after skb cow done */
    ip_decrease_ttl(iph);
```

```
        /* Allocate a new buffer for the datagram. */
        ihlen = ip_hdrlen(head);
        len = ihlen + qp->q.len;

        err = -E2BIG;
        if (len > 65535)
               goto out_oversize;
        ...
        skb_push(head, head->data - skb_network_header(head));
```

Forwarding

The main handler for forwarding a packet is the `ip_forward()` method:

```
int ip_forward(struct sk_buff *skb)
{
    struct iphdr          *iph;    /* Our header */
    struct rtable         *rt;     /* Route we use */
    struct ip_options     *opt    = &(IPCB(skb)->opt);
```

I should describe why Large Receive Offload (LRO) packets are dropped in forwarding. LRO is a performance-optimization technique that merges packets together, creating one large SKB, before they are passed to higher network layers. This reduces CPU overhead and thus improves the performance. Forwarding a large SKB, which was built by LRO, is not acceptable because it will be larger than the outgoing MTU. Therefore, when LRO is enabled the SKB is freed and the method returns NET_RX_DROP. Generic Receive Offload (GRO) design included forwarding ability, but LRO did not:

```
    if (skb_warn_if_lro(skb))
        goto drop;
```

If the `router_alert` option is set, the `ip_call_ra_chain()` method should be invoked to handle the packet. When calling `setsockopt()` with IP_ROUTER_ALERT on a raw socket, the socket is added to a global list named `ip_ra_chain` (see `include/net/ip.h`). The `ip_call_ra_chain()` method delivers the packet to all raw sockets. You might wonder why is the packet delivered to all raw sockets and not to a single raw socket? In raw sockets there are no ports on which the sockets listen, as opposed to TCP or UDP.

If the `pkt_type`—which was determined by the `eth_type_trans()` method, which should be called from the network driver, and which is discussed in Appendix A—is not PACKET_HOST, the packet is discarded:

```
    if (IPCB(skb)->opt.router_alert && ip_call_ra_chain(skb))
        return NET_RX_SUCCESS;

    if (skb->pkt_type != PACKET_HOST)
        goto drop;
```

The `ttl` (Time To Live) field of the IPv4 header is a counter which is decreased by 1 in each forwarding device. If the `ttl` reaches 0, that is an indication that the packet should be dropped and that a corresponding time exceeded ICMPv4 message with "TTL Count Exceeded" code should be sent:

```
    if (ip_hdr(skb)->ttl <= 1)
        goto too_many_hops;. . .
        . . .
```

Now the location for adding the fragment is found by looking for the first place which is after the fragment offset (the linked list of fragments is ordered by offset):

```
. . .
prev = NULL;
for (next = qp->q.fragments; next != NULL; next = next->next) {
       if (FRAG_CB(next)->offset >= offset)
              break;  /* bingo! */
       prev = next;
}
```

Now, prev points to where to add the new fragment if it is not NULL. Skipping handling overlapping and some other checks, let's continue to the insertion of the fragment into the list:

```
FRAG_CB(skb)->offset = offset;
/* Insert this fragment in the chain of fragments. */
skb->next = next;
if (!next)
    qp->q.fragments_tail = skb;
if (prev)
    prev->next = skb;
else
    qp->q.fragments = skb;
. . .
qp->q.meat += skb->len;
```

Note that the qp->q.meat is incremented by skb->len for each fragment. As mentioned earlier, qp->q.len is the total length of all fragments, and when it is equal to qp->q.meat, it means that all fragments were added and should be reassembled into one packet with the ip_frag_reasm() method.

Now you can see how and where reassembly takes place: (reassembly is done by calling the ip_frag_reasm() method):

```
if (qp->q.last_in == (INET_FRAG_FIRST_IN | INET_FRAG_LAST_IN) &&
    qp->q.meat == qp->q.len) {
    unsigned long orefdst = skb->_skb_refdst;

    skb->_skb_refdst = 0UL;
    err = ip_frag_reasm(qp, prev, dev);
    skb->_skb_refdst = orefdst;
    return err;
}
```

Let's take a look at the ip_frag_reasm() method:

```
static int ip_frag_reasm(struct ipq *qp, struct sk_buff *prev,
                         struct net_device *dev)
{
        struct net *net = container_of(qp->q.net, struct net, ipv4.frags);
        struct iphdr *iph;
        struct sk_buff *fp, *head = qp->q.fragments;
        int len;
      ...
```


Links

1. https://books.google.co.in/books?id=RpsQAwAAQBAJ&dq=ip_call_ra_chain&source=gbs_navlinks_s