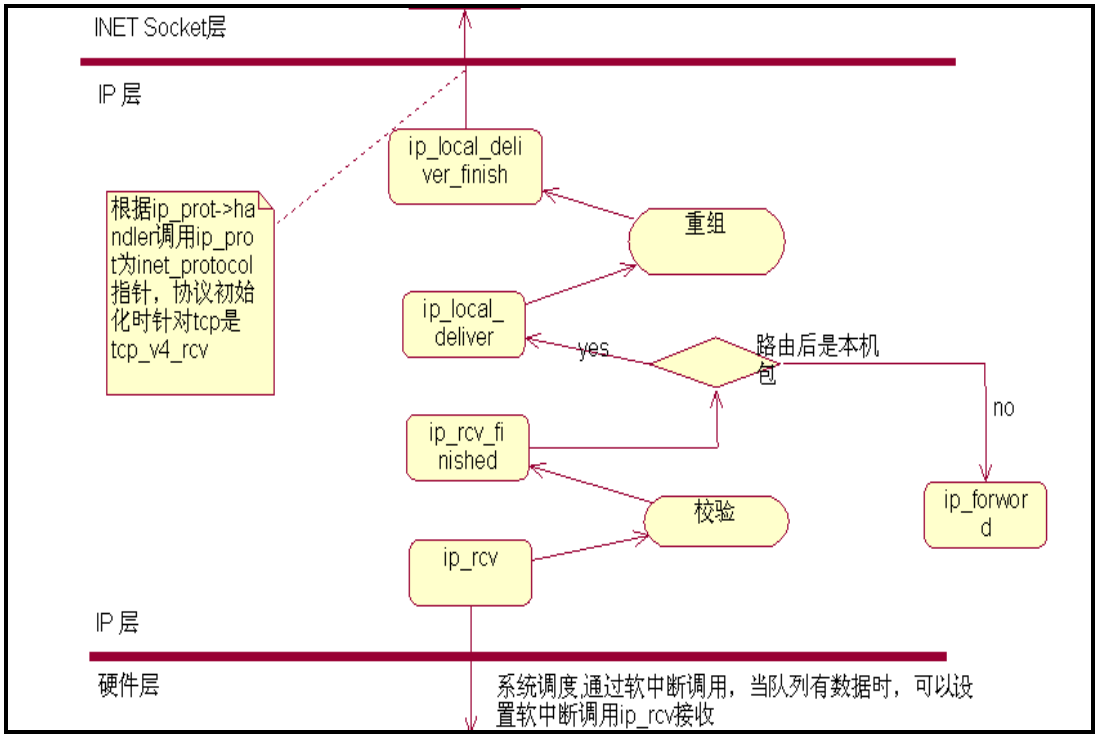
# 开源的 TCP/IP 协议栈中 IPv4 的简单分析

101220151 元玉慧

参考文档链接： http://blog.csdn.net/cz_hyf/article/details/602802

## IPv4 收发实现源代码

### IP 数据报的输入处理



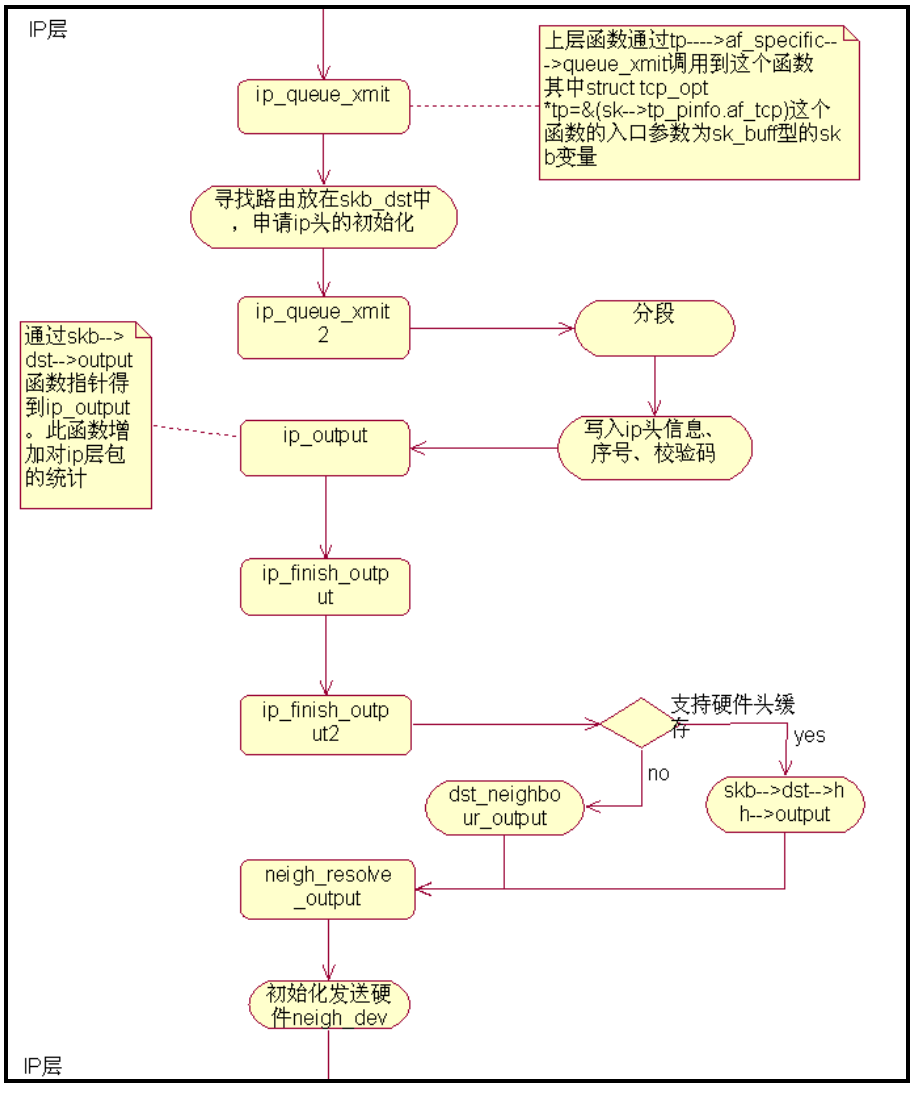

```
/*
 *	Main IP Receive routine.
 */
/* 参数说明如下：
skb，接收到的IP数据报
dev，接收到的IP数据报当前的输入网络设备
pt，输入此数据报的网络层输入接口，未使用
orig_dev，接收到的IP数据报原始的输入网络设备
*/
int ip_rcv(struct sk_buff *skb, struct net_device *dev, struct packet_type *pt, struct
net_device *orig_dev)
{
	const struct iphdr *iph;
```

```
    u32 len;

    /* When the interface is in promisc. mode, drop all the crap
     * that it receives, do not try to analyse it.
     */
//丢弃不去往本地的数据报，此处只接收发往本机的数据报
    if (skb->pkt_type == PACKET_OTHERHOST)
        goto drop;


    IP_UPD_PO_STATS_BH(dev_net(dev), IPSTATS_MIB_IN, skb->len);

//检测接收到的数据报是否为一个共享数据报。如果是，则必须复制一个副本，再做进一步处理，因为在处理过程中可能会修改数据报中的信息。
    if ((skb = skb_share_check(skb, GFP_ATOMIC)) == NULL) {
        IP_INC_STATS_BH(dev_net(dev), IPSTATS_MIB_INDISCARDS);
        goto out;
    }

    //通过判断数据报长度来检测数据报是否有效，不能小于IP首部长度。
    if (!pskb_may_pull(skb, sizeof(struct iphdr)))
        goto inhdr_error;

    iph = ip_hdr(skb);

    /*
     *    RFC1122: 3.2.1.2 MUST silently discard any IP frame that fails the checksum.
     *
     *    Is the datagram acceptable?
     *
     *    1.    Length at least the size of an ip header
     *    2.    Version of 4
     *    3.    Checksums correctly. [Speed optimisation for later, skip loopback checksums]
     *    4.    Doesn't have a bogus length
     */

//由于IP数据报首部长度至少为20B（即5个32位），因此IP数据报首部中首部长度域若小于5，则说明长度异常。或者IP版本域不为4，则版本异常。遇到以上两种情况，作相应的异常处理
    if (iph->ihl < 5 || iph->version != 4)
```

```
            goto inhdr_error;

//根据IP数据报首部中的首部长度值重新检测IP数据报是否有效
        if (!pskb_may_pull(skb, iph->ihl*4))
            goto inhdr_error;

//根据IP数据报首部中的校验和检测IP首部是否有效
        iph = ip_hdr(skb);

        if (unlikely(ip_fast_csum((u8 *)iph, iph->ihl)))
            goto inhdr_error;

//根据IP数据报首部中的数据包总长度值检测数据报是否有效
        len = ntohs(iph->tot_len);
        if (skb->len < len) {
            IP_INC_STATS_BH(dev_net(dev), IPSTATS_MIB_INTRUNCATEDPKTS);
            goto drop;
        } else if (len < (iph->ihl*4))
            goto inhdr_error;

        /* Our transport medium may have padded the buffer out. Now we know it
         * is IP we can trim to the true length of the frame.
         * Note this now means skb->len holds ntohs(iph->tot_len).
         */
//根据IP数据报首部中的数据包总长度重新设置skb的长度
        if (pskb_trim_rcsum(skb, len)) {
            IP_INC_STATS_BH(dev_net(dev), IPSTATS_MIB_INDISCARDS);
            goto drop;
        }

        /* Remove any debris in the socket control block */
//将skb中的IP控制块清零，以便后续对IP选项的处理
        memset(IPCB(skb), 0, sizeof(struct inet_skb_parm));

        /* Must drop socket now because of tproxy. */
        skb_orphan(skb);

//最后通过netfilter模块处理后，调用ip_rcv_finish()完成IP数据报的输入
        return NF_HOOK(NFPROTO_IPV4, NF_INET_PRE_ROUTING, skb, dev, NULL,
                       ip_rcv_finish);
```

```
//在此处理无效数据报
inhdr_error:
        IP_INC_STATS_BH(dev_net(dev), IPSTATS_MIB_INHDRERRORS);
drop:
        kfree_skb(skb);
out:
        return NET_RX_DROP;
}
```

## IP 数据报的输出处理




输出接口最常用的是 ip_queue_xmit()，而 ip_build_and_send_pkt()和 ip_send_reply()只有在发送特定段时才会被调用，如 ip_build_and_send_pkt()打包输出 SYN+ACK 类型的 TCP 段，而 ip_send_reply()用于构造输出 RST 和 ACK 段。普通的数据输出最后都是由

ip_queue_xmit()进行打包处理的。

**以 ip_queue_xmit()为代表进行分析**

```
/*参数说明如下：
skb，待封装成 IP 数据报的 TCP 段
ipfragok，标识带输出的数据是否已经完成分片。由于在调用在调用该函数时 ipfragok 参数总为 0，因此输出的 IP 数据报是否已分片取决于是否启用 PMTU 发现
*/
int ip_queue_xmit(struct sk_buff *skb, int ipfragok)
{
	struct sock *sk = skb->sk;
	struct inet_sock *inet = inet_sk(sk);
	struct ip_options *opt = sk->protinfo.af_inet.opt;
	struct rtable *rt;
	struct iphdr *iph;

	/* Skip all of this if the packet is already routed,
	 * f.e. by something like SCTP.
	 */
// 如果待输出的数据报已经准备好路由缓存，则无需再查找路由，直接跳转到 packet_routed 处理
	rt = (struct rtable *) skb->dst;
	if (rt != NULL)
		goto packet_routed;

	/* Make sure we can route this packet. */
	rt = (struct rtable *)__sk_dst_check(sk, 0);
	if (rt == NULL) {
		u32 daddr;

		/* Use correct destination address if we have options. */
		daddr = sk->daddr;
		if(opt && opt->srr)
			daddr = opt->faddr;

		/* If this fails, retransmit mechanism of transport layer will
		 * keep trying until route appears or the connection times itself
```

```
             * out.
             */
            if (ip_route_output(&rt, daddr, sk->saddr,
                               RT_CONN_FLAGS(sk),
                               sk->bound_dev_if))
                goto no_route;
            __sk_dst_set(sk, &rt->u.dst);
            sk->route_caps = rt->u.dst.dev->features;
        }
        skb->dst = dst_clone(&rt->u.dst);
/*
    如果输出该数据报的传输控制块中缓存了输出路由缓存项 ,则需检测该路由缓存项是否过期。
    如果过期，重新通过输出网络设备、目的地址、源地址等信息查找输出路由缓存项。如果查找到对应的路由缓存项，则将其缓存到传输控制块中，否则丢弃该数据包。
    如果未过期，则直接使用缓存在传输控制块中的路由缓存项。
*/

packet_routed:
    //查找到输出路由后，先进行严格源路由选项的处理。如果存在严格源路由选项，并且数据报的下一跳地址和网关地址不一致，则丢弃该数据报。
    if (opt && opt->is_strictroute && rt->rt_dst != rt->rt_gateway)
        goto no_route;

    /* OK, we know where to send it, allocate and build IP header. */
    //设置 IP 首部中各字段的值
    iph = (struct iphdr *) skb_push(skb, sizeof(struct iphdr) + (opt ? opt->optlen : 0));

// 在 ip 首部填入版本号 4 ,ip 首部长度 5(20 字节 ,这个值在后面要根据选项的长度增加)，以及服务类型
    *((__u16 *)iph)    = htons((4 << 12) | (5 << 8) | (sk->protinfo.af_inet.tos & 0xff));
    iph->tot_len = htons(skb->len);

// 如果 socket 要求 ip 不分片(这是通过检测 sock->pmtudisc 做到的 ,如果使用路径 mtu 发现则说明要求不分片，否则允许分片)并且参数 ipfragok 等于 0，那么将 DF 标志置 1，否则清 0
    if (ip_dont_fragment(sk, &rt->u.dst) && !ipfragok)
```

```
            iph->frag_off = htons(IP_DF);
        else
            iph->frag_off = 0;
// 设置 ip 首部的 ttl(从 sock 的 uc_ttl 获得，如果小于 0 则从路由项的 metrics 获得)，protocol(从 sock->sk_protocol),源地址，目标地址(两者都从路由项获得)
        iph->ttl      = sk->protinfo.af_inet.ttl;
        iph->protocol = sk->protocol;
        iph->saddr    = rt->rt_src;
        iph->daddr    = rt->rt_dst;
        skb->nh.iph   = iph;
        /* Transport layer set skb->h.foo itself. */

// 若 opt 不为 NULL，则在 ip 首部长度中加上选项长度， 并且调用 ip_options_build 向 IP 首部中写入 ip 选项
        if(opt && opt->optlen) {
            iph->ihl += opt->optlen >> 2;
            ip_options_build(skb, opt, sk->daddr, rt, 0);
        }

        ip_select_ident_more(iph, &rt->dst, sk,
                             (skb_shinfo(skb)->gso_segs ?: 1) - 1);

        /* Add an IP checksum. */
        Ip_send_check(iph);

        //设置输出数据报的 QoS 类别
        skb->priority = sk->sk_priority;

        //最后通过 netfilter 处理后，由 dst_output()处理数据报的输出
        return NF_HOOK(PF_INET, NF_IP_LOCAL_OUT, skb, NULL, rt->u.dst.dev,
                       dst_output);

//如果查找不到对应的路由缓存项，在此处理，将该数据报丢弃
no_route:
        IP_INC_STATS(IpOutNoRoutes);
        kfree_skb(skb);
        return -EHOSTUNREACH;
}
```

## IPv4 转发实现源代码

```
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);

    // 调用xfrm4_policy_check()查找IPsec策略数据库。如果查找失败，则丢弃该数据报。
    if (!xfrm4_policy_check(NULL, XFRM_POLICY_FWD, skb))
        goto drop;

    //如果数据报中存在路由警告选项 ,则调用ip_call_ra_chain()将数据报输入给对路由警告选项感兴趣的用户进程。如果成功，则不再转发数据报
    if (IPCB(skb)->opt.router_alert && ip_call_ra_chain(skb))
        return NET_RX_SUCCESS;

    //承载该IP数据报的以太网帧目的地址与收到它的网络设备的MAC地址相等才能转发，也就是说发给接收该数据报的主机的数据包才接收
    if (skb->pkt_type != PACKET_HOST)
        goto drop;

    //由于在转发过程中可能会修改IP首部，因此将ip_summed设置为CHECKSUM_NONE，在后续的输出时还得由软件来执行校验和。
    skb->ip_summed = CHECKSUM_NONE;

    /*
     *    According to the RFC, we must first decrease the TTL field. If
     *    that reaches zero, we must reply an ICMP control message telling
     *    that the packet's lifetime expired.
     */
    //待转发数据报的生存时间为0时，丢弃该数据报，并发送超时ICMP报文给发送方
    if (ip_hdr(skb)->ttl <= 1)
        goto too_many_hops;

    //进行IPsec路由选路和转发处理，如果失败，则丢弃该数据报
    if (!xfrm4_route_forward(skb))
        goto drop;

    rt = skb_rtable(skb);
```

```
//如果数据报启用严格源选路由选项，且数据报的下一跳不是网关，则发送超时ICMP报文给发送方，并丢弃该数据报
    if (opt->is_strictroute && opt->nexthop != rt->rt_gateway)
        goto sr_failed;

    /* We are about to mangle packet. Copy it! */
    //确保SKB有指定长度的headroom空间。当SKB的headroom空间小于指定长度或者克隆SKB时，会新建SKB缓冲并释放对原包的引用
    if (skb_cow(skb, LL_RESERVED_SPACE(rt->dst.dev)+rt->dst.header_len))
        goto drop;
    iph = ip_hdr(skb);

    /* Decrease ttl after skb cow done */
    //经过路由器，IP数据报生存时间递减1
    ip_decrease_ttl(iph);

    /*
     *    We now generate an ICMP HOST REDIRECT giving the route
     *    we calculated.
     */
    //如果该数据报的输出路由存在重定向标志，且该数据报中不存在源路由选项，则向发送方发送重定向ICMP报文。
    if (rt->rt_flags&RTCF_DOREDIRECT && !opt->srr && !skb_sec_path(skb))
        ip_rt_send_redirect(skb);

//数据包中TOS字段值将被转发数据包插入硬件缓冲区的不同级别队列中。硬件驱动程序在发送数据包时，将首先处理高优先级队列中的数据包
    skb->priority = rt_tos2priority(iph->tos);

//经netfilter处理后，调用ip_forward_finish()，完成IP层的转发操作
    return NF_HOOK(NFPROTO_IPV4, NF_INET_FORWARD, skb, skb->dev,
                   rt->dst.dev, ip_forward_finish);

//当处理过程中发生异常时跳转到此进行相关的异常处理
sr_failed:
    /*
     *    Strict routing permits no gatewaying
     */
     icmp_send(skb, ICMP_DEST_UNREACH, ICMP_SR_FAILED, 0);
     goto drop;
```

```
too_many_hops:
        /* Tell the sender its packet died... */
        //发送超时ICMP报文给发送方
        IP_INC_STATS_BH(dev_net(skb_dst(skb)->dev), IPSTATS_MIB_INHDRERRORS);
        icmp_send(skb, ICMP_TIME_EXCEEDED, ICMP_EXC_TTL, 0);
drop:
        //丢弃数据报
        kfree_skb(skb);
        return NET_RX_DROP;
}
```

（2）　ip_forward_finish()

```
// ip_forward_finish()完成输入IP数据报的转发
static int ip_forward_finish(struct sk_buff *skb)
{
        struct ip_options * opt      = &(IPCB(skb)->opt);

        IP_INC_STATS_BH(dev_net(skb_dst(skb)->dev),
IPSTATS_MIB_OUTFORWDATAGRAMS);

        //处理转发IP数据报中的IP选项，包括记录路由选项和时间戳选项
        if (unlikely(opt->optlen))
                ip_forward_options(skb);

        //通过路由缓存将数据报输出，最终会调用单播的ip_output()或组播的输出函数
ip_mc_output()
        return dst_output(skb);
}
```