**PROJECT BASED LEARNING**

Computer Networks

Forwarding Protocol

|  |  |
| --- | --- |
|  | Submitted By |
| CHAVA SAI TEJA | (13BCE1032) |
| SAILESH PATRA | (13BCE1119) |
| MARLA AKHIL REDDY | (13BCE1078) |

**Faculty In charge: Prof.Dr.Rajesh Kumar**

A project report submitted to the

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

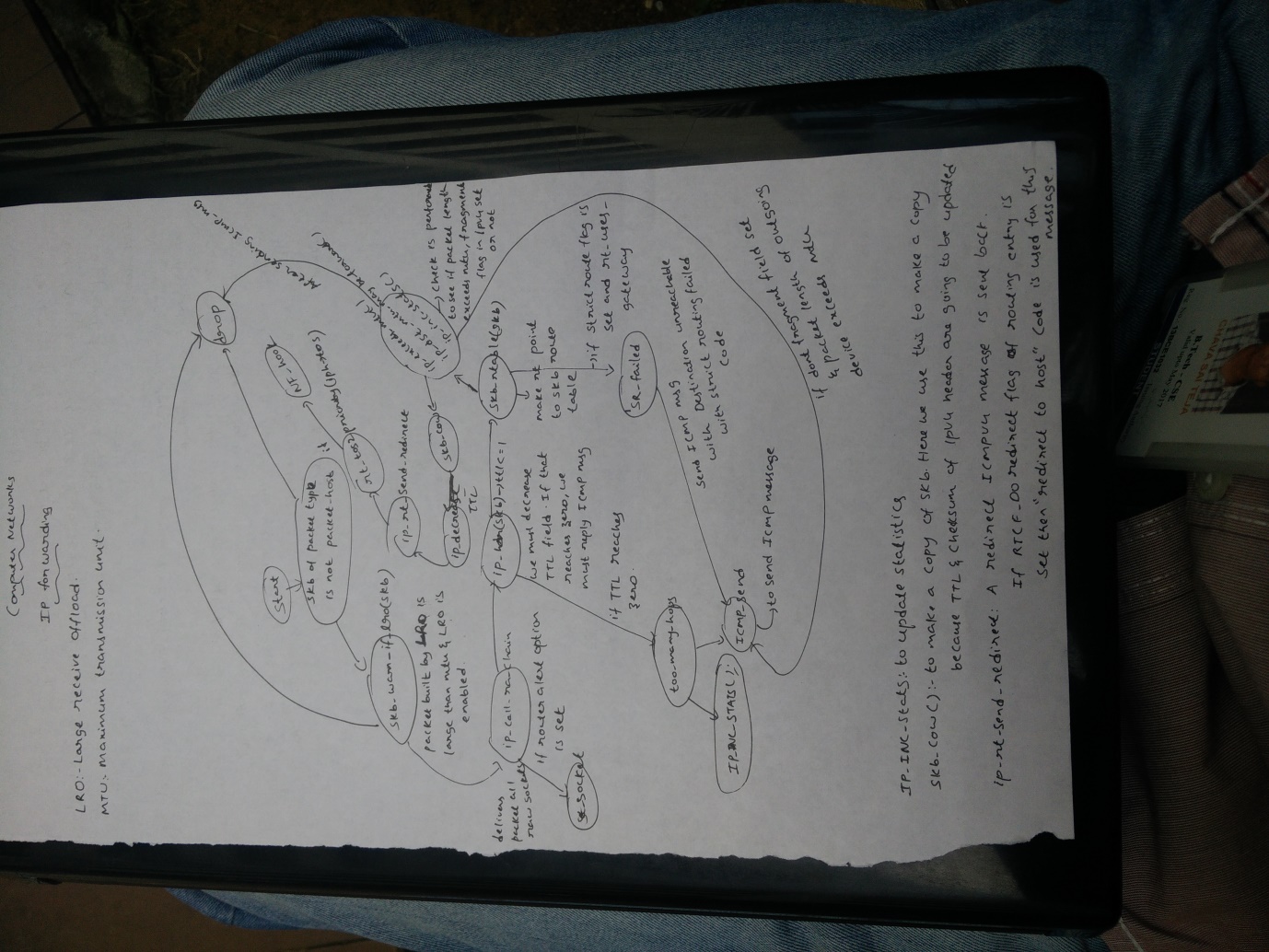


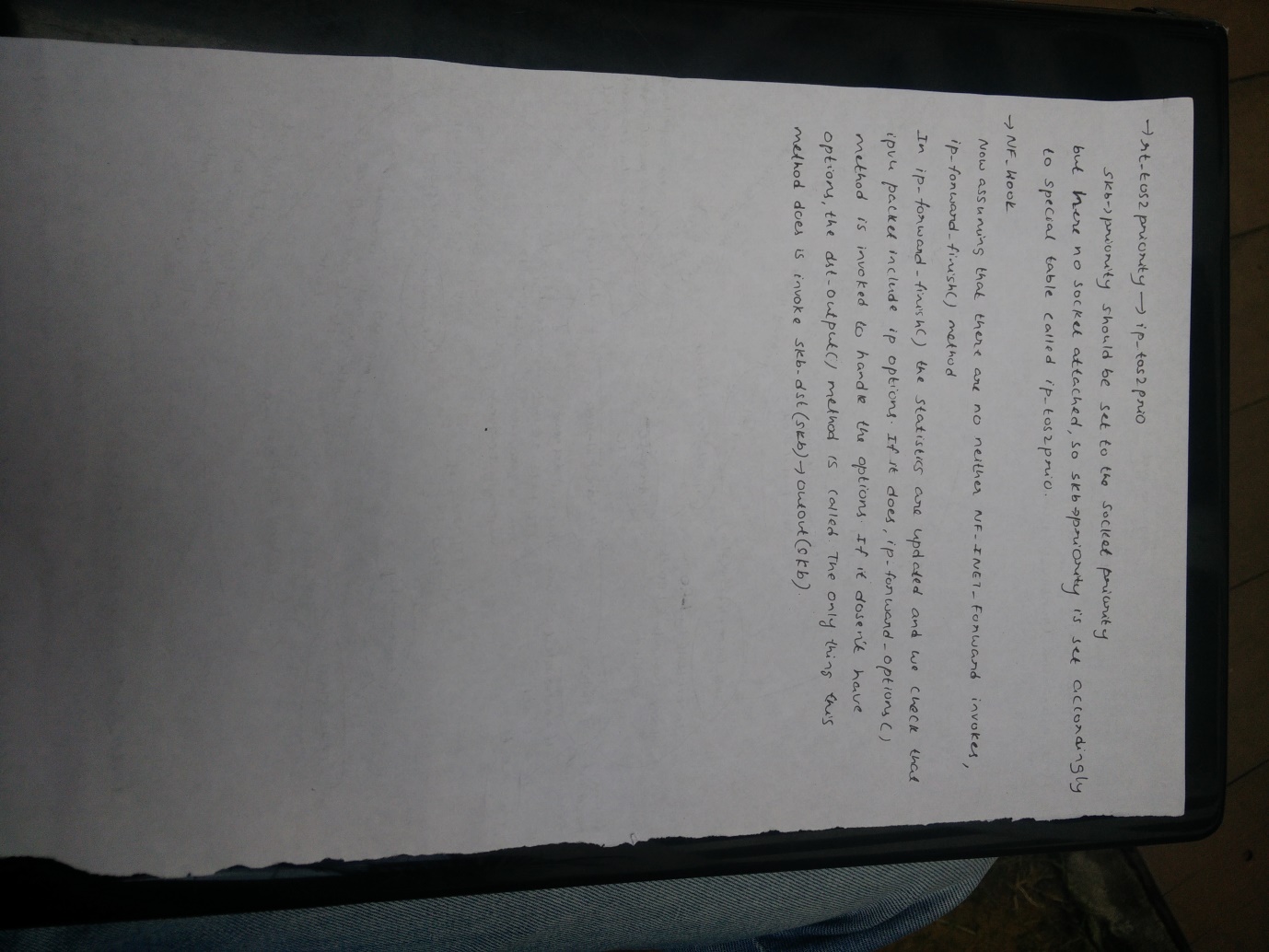
**VIT UNIVERSITY, CHENNAI**

**Vandalur – Kelambakkam Road**

**Chennai–600127.**

**DIAGRAM**





**IP\_FORWARD(Code Explained)**

[**u32**](http://lxr.free-electrons.com/ident?i=u32)[**mtu**](http://lxr.free-electrons.com/ident?i=mtu)**;**

**struct** [**iphdr**](http://lxr.free-electrons.com/ident?i=iphdr) **\*iph; */\* Our header \*/***

**struct** [**rtable**](http://lxr.free-electrons.com/ident?i=rtable) **\***[**rt**](http://lxr.free-electrons.com/ident?i=rt)**; */\* Route we use \*/***

**struct** [**ip\_options**](http://lxr.free-electrons.com/ident?i=ip_options) **\*opt = &(**[**IPCB**](http://lxr.free-electrons.com/ident?i=IPCB)**(**[**skb**](http://lxr.free-electrons.com/ident?i=skb)**)->opt);**

In this firstly we are initalising 32 mtu (**maximum transmission unit** (**MTU**),which is the largest size packet or frame, specified in octets (eight-bit bytes), that can be sent in a packet- or frame-based **network** such as the Internet)

Structures iphdr and rtable deal with initalising structures of header and route we use in forwarding.

***/\* that should never happen \*/***

**if (**[**skb**](http://lxr.free-electrons.com/ident?i=skb)**->pkt\_type !=** [**PACKET\_HOST**](http://lxr.free-electrons.com/ident?i=PACKET_HOST)**)**

**goto** [**drop**](http://lxr.free-electrons.com/ident?i=drop)**;**

If packet type in skb is not equal to PACKET\_HOST (when an incoming ethernet frame is to a destination MAC address matching the MAC address of the ethernet device it arrived on, this field will be set to 'PACKET\_HOST') i.e like 'PACKET\_BROADCAST', 'PACKET\_MULTICAST' (we get diffferent packet types from 'linux/if\_packet.h' header file).

**if (**[**unlikely**](http://lxr.free-electrons.com/ident?i=unlikely)**(**[**skb**](http://lxr.free-electrons.com/ident?i=skb)**->sk))**

**goto** [**drop**](http://lxr.free-electrons.com/ident?i=drop)**;**

In this unlikely is just a prefrocessor micro which takes condition and activate condition i.e (**#define** [**unlikely**](http://lxr.free-electrons.com/ident?i=unlikely)**(**[**cond**](http://lxr.free-electrons.com/ident?i=cond)**) (**[**cond**](http://lxr.free-electrons.com/ident?i=cond)**)).** So when there is error associated with socket with respect to skb (generally in memory allocation)then we need to drop it.

**if (**[**skb\_warn\_if\_lro**](http://lxr.free-electrons.com/ident?i=skb_warn_if_lro)**(**[**skb**](http://lxr.free-electrons.com/ident?i=skb)**))**

**goto** [**drop**](http://lxr.free-electrons.com/ident?i=drop)**;**

In this function we send warning by passing true value in return if large receive offload(It works by aggregating multiple incoming [packets](https://en.wikipedia.org/wiki/Packet_(information_technology)) from a single [stream](https://en.wikipedia.org/wiki/Stream_(computing)) into a larger buffer before they are passed higher up the networking stack, thus reducing the number of packets that have to be processed.) sets a incompatible size(i.e gso\_size) and may also set wrong gso\_type(This used for drivers receiveing super-sized skb's. These are indicated to the driver by skb\_shinfo(skb)->gso\_size being non-zero. The gso\_size is the size the hardware should fragment the TCP data. TSO may change how and when TCP decides to send data.)

Drop:

void kfree\_skb(struct sk\_buff \*skb);

void dev\_kfree\_skb(struct sk\_buff \*skb);

void dev\_kfree\_skb\_irq(struct sk\_buff \*skb);

void dev\_kfree\_skb\_any(struct sk\_buff \*skb);

Free a buffer. The *kfree\_skb* call is used internally by the kernel. A driver should use one of the forms of *dev\_kfree\_skb* instead: *dev\_kfree\_skb* for noninterrupt context,*dev\_kfree\_skb\_irq* for interrupt context, or *dev\_kfree\_skb\_any* for code that can run in either context.

**if (!**[**xfrm4\_policy\_check**](http://lxr.free-electrons.com/ident?i=xfrm4_policy_check)**(**[**NULL**](http://lxr.free-electrons.com/ident?i=NULL)**, XFRM\_POLICY\_FWD,** [**skb**](http://lxr.free-electrons.com/ident?i=skb)**))**

**goto** [**drop**](http://lxr.free-electrons.com/ident?i=drop)**;**

This function returns a flag if network device condition is not properly set along with device destination .It will check inside calling another functions from net.h header.So here if not flag, We will drop it.

[**skb\_forward\_csum**](http://lxr.free-electrons.com/ident?i=skb_forward_csum)**(**[**skb**](http://lxr.free-electrons.com/ident?i=skb)**);**

Here we check skb->ip\_summed is CHECKSUM\_COMPLETE or not. Unfortunately we don’t support this one. So we change that value to CHECKSUM\_NONE

**if (**[**ip\_hdr**](http://lxr.free-electrons.com/ident?i=ip_hdr)**(**[**skb**](http://lxr.free-electrons.com/ident?i=skb)**)->**[**ttl**](http://lxr.free-electrons.com/ident?i=ttl) **<= 1)**

**goto too\_many\_hops;**

ip\_hdr(skb):

Takes skb as parameter and returns skb->head + skb->network\_header

If its ttl is less than or equal to 1,signifies maximum count of hops is attained and we are gonna drop the packet and send the sender an ICMP message.

[**rt**](http://lxr.free-electrons.com/ident?i=rt) **=** [**skb\_rtable**](http://lxr.free-electrons.com/ident?i=skb_rtable)**(**[**skb**](http://lxr.free-electrons.com/ident?i=skb)**);**

skb\_rtable which returns the routing table entry

**if (opt->is\_strictroute &&** [**rt**](http://lxr.free-electrons.com/ident?i=rt)**->rt\_uses\_gateway) goto sr\_failed;**

Strict routing implies that the entire set of SIP nodes which may be

visited is listed, in order of visitation, in the Route header. No other

nodes may be visited by this message, and all the listed nodes MUST be

visited in the given order or the message has "failed".

Loose routing implies that the indicated nodes MUST be visited before the

message can be delivered to the target indicated in the original request

URI. The message may visit other nodes before, between or after any node

specified on the loose route.

opt is pointer object to ip\_options

If opt is strict route and rt uses gateway which is not possible since strict route dosent allow gatewaying. The packet is dropped after sending ICMP message.

**iph =** [**ip\_hdr**](http://lxr.free-electrons.com/ident?i=ip_hdr)**(**[**skb**](http://lxr.free-electrons.com/ident?i=skb)**);**

**What ip\_hdr(skb) returns is**

**return (struct** [**iphdr**](http://lxr.free-electrons.com/ident?i=iphdr) **\*)**[**skb\_network\_header**](http://lxr.free-electrons.com/ident?i=skb_network_header)**(**[**skb**](http://lxr.free-electrons.com/ident?i=skb)**);**

**You are assigning this value to iph.**

**if (**[**skb\_cow**](http://lxr.free-electrons.com/ident?i=skb_cow)**(**[**skb**](http://lxr.free-electrons.com/ident?i=skb)**,** [**LL\_RESERVED\_SPACE**](http://lxr.free-electrons.com/ident?i=LL_RESERVED_SPACE)**(**[**rt**](http://lxr.free-electrons.com/ident?i=rt)**->**[**dst**](http://lxr.free-electrons.com/ident?i=dst)**.**[**dev**](http://lxr.free-electrons.com/ident?i=dev)**)+**[**rt**](http://lxr.free-electrons.com/ident?i=rt)**- >**[**dst**](http://lxr.free-electrons.com/ident?i=dst)**.**[**header\_len**](http://lxr.free-electrons.com/ident?i=header_len)**))**

**goto** [**drop**](http://lxr.free-electrons.com/ident?i=drop)**;**

**skb->cow:-**

copy header of skb when it is required.

**Arguments**

*skb*

buffer to cow

*headroom*

needed headroom

If the skb passed lacks sufficient headroom or its data part is shared, data is reallocated. If reallocation fails, an error is returned and original skb is not changed.The result is skb with writable area skb->head...skb->tail and at least headroom of space at head.

[**ip\_decrease\_ttl**](http://lxr.free-electrons.com/ident?i=ip_decrease_ttl)**(iph);**

Decrements the ttl after the skb\_cow is done.

**if (**[**IPCB**](http://lxr.free-electrons.com/ident?i=IPCB)**(**[**skb**](http://lxr.free-electrons.com/ident?i=skb)**)->**[**flags**](http://lxr.free-electrons.com/ident?i=flags) **&** [**IPSKB\_DOREDIRECT**](http://lxr.free-electrons.com/ident?i=IPSKB_DOREDIRECT) **&& !opt->**[**srr**](http://lxr.free-electrons.com/ident?i=srr) **&&**

**!**[**skb\_sec\_path**](http://lxr.free-electrons.com/ident?i=skb_sec_path)**(**[**skb**](http://lxr.free-electrons.com/ident?i=skb)**))**

IPSKB\_DOREDIRECT:-

**#define** [**IPSKB\_DOREDIRECT**](http://lxr.free-electrons.com/ident?i=IPSKB_DOREDIRECT)[**BIT**](http://lxr.free-electrons.com/ident?i=BIT)**(5)**

**sr\_failed:**

[**icmp\_send**](http://lxr.free-electrons.com/ident?i=icmp_send)**(**[**skb**](http://lxr.free-electrons.com/ident?i=skb)**,** [**ICMP\_DEST\_UNREACH**](http://lxr.free-electrons.com/ident?i=ICMP_DEST_UNREACH)**,** [**ICMP\_SR\_FAILED**](http://lxr.free-electrons.com/ident?i=ICMP_SR_FAILED)**, 0);**

**goto** [**drop**](http://lxr.free-electrons.com/ident?i=drop)**;**

Strict routing permits no gateway

So send an icmp message with destination unreachable and source failed set.

[**IPCB**](http://lxr.free-electrons.com/ident?i=IPCB)**(**[**skb**](http://lxr.free-electrons.com/ident?i=skb)**)->**[**flags**](http://lxr.free-electrons.com/ident?i=flags) **|=** [**IPSKB\_FORWARDED**](http://lxr.free-electrons.com/ident?i=IPSKB_FORWARDED)**;**

IPSKB\_FORWARDED:

Preprocessor macro

**#define** [**IPSKB\_FORWARDED**](http://lxr.free-electrons.com/ident?i=IPSKB_FORWARDED)[**BIT**](http://lxr.free-electrons.com/ident?i=BIT)**(0)**

**return** [**NF\_HOOK**](http://lxr.free-electrons.com/ident?i=NF_HOOK)**(NFPROTO\_IPV4, NF\_INET\_FORWARD,** [**NULL**](http://lxr.free-electrons.com/ident?i=NULL)**,** [**skb**](http://lxr.free-electrons.com/ident?i=skb)**,** [**skb**](http://lxr.free-electrons.com/ident?i=skb)**->**[**dev**](http://lxr.free-electrons.com/ident?i=dev)**,** [**rt**](http://lxr.free-electrons.com/ident?i=rt)**->**[**dst**](http://lxr.free-electrons.com/ident?i=dst)**.**[**dev**](http://lxr.free-electrons.com/ident?i=dev)**,** [**ip\_forward\_finish**](http://lxr.free-electrons.com/ident?i=ip_forward_finish)**);**

NF\_HOOK:

* NF\_HOOK macro:
  + NF\_HOOK(pf, hook, skb, indev, outdev, okfn)
* For each registered module under the hook nf\_hooks[pf][hook]
  + Pass the packet to the module and wait for the "verdict"
  + If verdict is drop, drop the packet
  + If verdict is okay, continue next module
* If final verdict is okay, call okfn(skb)

1. **IP\_INC\_STATS([dev\_net](http://lxr.free-electrons.com/ident?i=dev_net)(**[**rt**](http://lxr.free-electrons.com/ident?i=rt)**->**[**dst**](http://lxr.free-electrons.com/ident?i=dst)**.**[**dev**](http://lxr.free-electrons.com/ident?i=dev)**), IPSTATS\_MIB\_FRAGFAILS);**

**IP\_INC\_STATS()** is a macro defined in net/ip.h header file and is used to determine the amount of padding added to the ethernet frame.

1. if ([IPCB](http://lxr.free-electrons.com/ident?i=IPCB)([skb](http://lxr.free-electrons.com/ident?i=skb))->[flags](http://lxr.free-electrons.com/ident?i=flags) & [IPSKB\_DOREDIRECT](http://lxr.free-electrons.com/ident?i=IPSKB_DOREDIRECT) && !opt->[srr](http://lxr.free-electrons.com/ident?i=srr) &&![skb\_sec\_path](http://lxr.free-electrons.com/ident?i=skb_sec_path)([skb](http://lxr.free-electrons.com/ident?i=skb)))

[ip\_rt\_send\_redirect](http://lxr.free-electrons.com/ident?i=ip_rt_send_redirect)([skb](http://lxr.free-electrons.com/ident?i=skb));

Here we generate an ICMP HOST REDIRECT giving the route which is calculated in the previous instructions.

1. [rt](http://lxr.free-electrons.com/ident?i=rt) = [skb\_rtable](http://lxr.free-electrons.com/ident?i=skb_rtable)([skb](http://lxr.free-electrons.com/ident?i=skb));

here, the object of the structure rtable i.e., rt, is initialized to the return value of the function skb\_rtable(). Means the router table accepts the incoming packets and redirects them according to the routing table.

**Some Key points**

In computer networking, large receive offload (LRO) is a technique for increasing inbound throughput of

high-bandwidth network connections by reducing CPU overhead.

LRO packets are dropped in forwarding.LRO is 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 bcz it will be large than the outgoing MTU.

Therfore, when LRO is enabled the SKB is freed and mthod returns

NET\_RX\_DROP.Generic Receive Offload(GRO) design included forwarding

ability,but LRO didnot:

if (skb\_warn\_if\_lro(skb))

goto drop;

drop:

kfree\_skb(skb);

return NET\_RX\_DROP;

---------------------------------------------------------------------

If the router\_alert option is set,the ip\_call\_ra\_chain() method should

be invoked to handle the packet.When calling setsocket() with

IP\_ROUTER\_ALERT on raw socket,socket is added to a global list

named ip\_ra\_chain().The ip\_call\_ra\_chain() method delivers the packet

to all raw sockets.In raw packets there is no proper ports which

socets listen(X to TCP/UDP) so w are sending to all raw's.

If the pkt\_type which is determined by eth\_type\_trans() methosd,which

should be called from the network driver and which is not PACKET\_HOST,

packt 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;

-----------------------------------------------------------------

TTL field of IPV4 header is a conuter which is decrsed by 1 in each

forwarding device.If the ttl reaches 0,that is an indication that the

packet should be dropped and that a correspoding time exceeded ICMPv4

message with "TTL Count Exceeded" code should be snt.

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.

if (ip\_hdr(skb)->ttl <= 1)

goto too\_many\_hops;

too\_many\_hops:

It's like tlling the sender its packet died.

IP\_INC\_STATS(dev\_net(rt->dst.dev), IPSTATS\_MIB\_FRAGFAILS);

icmp\_send(skb, ICMP\_DEST\_UNREACH, ICMP\_FRAG\_NEEDED,0);

----------------------------------------------------------------------

Now a check is performed if both the strict route flag(is\_strictroute)

is set and 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:Strict routing permits no gatewaying

icmp\_send(skb, ICMP\_DEST\_UNREACH, ICMP\_SR\_FAILED,0);

goto drop;

-------------------------------------------------------------------------

Now a ceck 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 prmitted

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 thse conditions ar met,it means that when the

packet raches the ip\_output() mthod,it will not be fragmented with the

ip\_fragment() method.This means the packet cannot be sent as in,and it also

cannot cannot 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 "Fragmention Needed" code is sent back,the packet is dropped, and the statistics(IPSTATS\_MIB\_FRAGMENTS) are updated:

IPCB(skb)->flags |= IPSKB\_FORWARDED;

mtu = ip\_dst\_mtu\_maybe\_forward(&rt->dst, true);

if (ip\_exceeds\_mtu(skb, mtu)) {

IP\_INC\_STATS(dev\_net(rt->dst.dev), IPSTATS\_MIB\_FRAGFAILS);

icmp\_send(skb, ICMP\_DEST\_UNREACH, ICMP\_FRAG\_NEEDED,

htonl(mtu));

goto drop;

}

---------------------------------------------------------------------------

Bcz 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);

-------------------------------------------------------------------------

Result of ttl change and checksumare also updated accordingly in i\_decrese\_ttl()

/\* Decrease ttl after skb cow done \*/

ip\_decrease\_ttl(iph);

----------------------------------------------------------------------------

Now a redirect ICMPv4 message is sent back.If the RTCF\_DOREDIRECT flag of

the routing entry is set then a "Redirect To Host" code is used for this message

/\*

\* We now generate an ICMP HOST REDIRECT giving the route

\* we calculated.

\*/

if (IPCB(skb)->flags & IPSKB\_DOREDIRECT && !opt->srr && !skb\_sec\_path(skb))

ip\_rt\_send\_redirect(skb);

----------------------------------------------------------------------------

The skb->priority in the Tx Path is set to be the socket priority

( skb->priority). socket priority,inturn ,can be set by calling setsocket()

system call with SOL\_SOCKET and SO\_PRIORITY.How ever,when forwarding the

packet,there is no socket attached to the skb.So in the ip\_forward() method

and skb-> priority is set accordingly to a special table called ip\_tos2prio.

This table has 16 entries(I got it from route.h file...Just for refence I addded).

skb->priority = rt\_tos2priority(iph->tos);

----------------------------------------------------------------------------

Now assuming that there are no neither NF\_INET\_FORWARD invokes,ip\_forward\_finish()

method

return NF\_HOOK(NFPROTO\_IPV4, NF\_INET\_FORWARD, NULL, skb,

skb->dev, rt->dst.dev, ip\_forward\_finish);

In ip\_forward\_finish(),the statics are updated and we check that IPv4 packet include

IP options.If it does,ip\_forward\_options()method is invoked to handle the options.

If it doen't have options,the dst\_output() method is called.The only thing

this method deos is invoke skb\_dst(skb)->outout(skb):

static int ip\_forward\_finish(struct sock \*sk, struct sk\_buff \*skb)

{

struct ip\_options \*opt = &(IPCB(skb)->opt);

IP\_INC\_STATS\_BH(dev\_net(skb\_dst(skb)->dev), IPSTATS\_MIB\_OUTFORWDATAGRAMS);

IP\_ADD\_STATS\_BH(dev\_net(skb\_dst(skb)->dev), IPSTATS\_MIB\_OUTOCTETS, skb->len);

if (unlikely(opt->optlen))

ip\_forward\_options(skb);

skb\_sender\_cpu\_clear(skb);

return dst\_output\_sk(sk, skb);

}

So by this code we earned about methods for forwarding Packets(ip\_forward()

and ip\_forward\_finish()),

about cases when a packet is discarded in forwarding,about cases when an

ICMP redirect is sent and others.

**SK\_BUFF Header file**

**The socket buffer, or "SKB", is the most fundamental data structure in networking code. Every packet sent or received is handled using this data structure.**

The most fundamental parts of the SKB structure are as follows:

struct sk\_buff {

/\* These two members must be first. \*/

struct sk\_buff \*next;

struct sk\_buff \*prev;

struct sk\_buff\_head \*list;

...

The first two members implement list handling. Packets can exist on several kinds of lists and queues. For example, a TCP socket send queue generally. The third member says which list the packet is on. **{many lists of skb’s and in it which list’s skb is on is got by using sk\_buff\_head( ). }**

struct sock \*sk;

 A **socket** is one endpoint of a two-way communication link between two programs running on the **network**. A **socket** is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to. An endpoint is a combination of an IP address and a port number.

This is where we record the socket assosciated with this SKB. When a packet is sent or received for a socket, the memory assosciated with the packet must be charged to the socket for proper memory accounting. So we define it for solving this issue.

struct timeval stamp;

This structure is used to get timestamp for the packet, either when it arrived or when it was sent. **Calculating this is somewhat expensive, so this value is only recorded if necessary.** When something happens that requires that we start recording timestamps, **net\_enable\_timestamp() function**  is called. If that need goes away, **net\_disable\_timestamp() function** is called.

Timestamps are mostly used to **packet sniffers(**a sniffer program which targets packets of data transmitted over the Internet.). But they are also used to implement certain socket options, and also some netfilter modules make use of this value as well.

struct net\_device \*dev;

struct net\_device \*real\_dev;

These two members help keep track of the devices assosciated with a packet. The reason we have three different device pointers is that the main 'skb->dev' member can change as we encapsulate and decapsulate via a virtual device.

Packet enters networking via function **'netif\_receive\_skb()' .**Then we save 'skb->dev' away in 'skb->real\_dev' and update 'skb->dev' to point to the bonding device.

union {

struct tcphdr \*th;

struct udphdr \*uh;

struct icmphdr \*icmph;

struct igmphdr \*igmph;

struct iphdr \*ipiph;

struct ipv6hdr \*ipv6h;

unsigned char \*raw;

} h;

union {

struct iphdr \*iph;

struct ipv6hdr \*ipv6h;

struct arphdr \*arph;

unsigned char \*raw;

} nh;

union {

unsigned char \*raw;

} mac;

Here we store the location of the various protocol layer headers as we build outgoing packets, and resolve incoming packets. For example, 'skb->mac.raw' is set by **'eth\_type\_trans()**', when an eternet packet is received. Later, we can use this to find the location of the MAC header.

struct dst\_entry \*dst;

This structure is the general route for the packet. It tells us how to get the packet to it's destination and routes are used for both input and output.

struct sec\_path \*sp;

In this structure, we store the security path traversed by the packet. When we are trying to validate the security policy against a packet, we make sure that the transformations applied match the ones allowed by the policy.

char cb[40];

This is the SKB control block. It is an storage area usable by protocols, and to store private per-packet information.

unsigned int len,

data\_len,

mac\_len,

csum;

The three length members are pretty straight-forward. The total number of bytes in the packet is 'len'. If there are page buffers, the total number of bytes in the page buffer area is 'data\_len'. Therefore the number of bytes in the linear buffer is 'skb->len - skb->data\_len'.

The 'mac\_len' holds the length of the MAC header.

'csum' holds the checksum of the packet. When building send packets, we copy the data in from userspace and calculate the 16-bit two's complement sum in parallel for performance. This sum is accumulated in 'skb->csum'. This helps us compute the final checksum stored in the protocol packet header checksum field. **This field can end up being ignored if, for example, the device will checksum the packet for us.**

On input, the 'csum' field can be used to store a checksum calculated by the device. If the device indicates 'CHECKSUM\_HW' in the SKB **'ip\_summed'** field, this means that 'csum' is the two's complement checksum of the entire packet data area starting at 'skb->data'.

unsigned char local\_df,

cloned:1,

nohdr:1,

pkt\_type,

ip\_summed;

The 'local\_df' field is used by the IPV4 protocol, and when set allows us to locally fragment frames which have already been fragmented. This situation can arise with IPSEC.

In order to make quick references to SKB data, We use SKB clones. **When a clone of an SKB is made, all of the 'struct sk\_buff' structure members of the clone are private to the clone.** When an SKB is cloned, the 'cloned' field will be set in both the primary and clone SKB. Otherwise is will be zero.

The 'nohdr' field is used in the support of TCP Segmentation Offload ('TSO')

The type of the packet is stored in the 'pkt\_type' field.

When an incoming ethernet frame is to a destination MAC address matching the MAC address of the ethernet device it arrived on, this field will be set to 'PACKET\_HOST'. Broadcast frame is received 🡪 'PACKET\_BROADCAST'

Multicast packet is received 🡪'PACKET\_MULTICAST'

The 'ip\_summed' field describes what kind of checksumming assistence the card has provided for a receive packet.

It takes on one of three values:

* 'CHECKSUM\_NONE' (the card provided no checksum assistence)
* 'CHECKSUM\_HW' (two's complement checksum over the entire packet has been provided)
* 'CHECKSUM\_UNNECESSARY' (the device only provides a 'checksum OK' indication for receive packet checksum offload)

\_\_u32 priority;

The 'priority' field is used in the implement of Quality of Service.

unsigned short protocol,

security;

Protocol type values to indicate what protocol should receive the packet.

The 'security' field was meant to be used in the implementation of IP Security.

void (\*destructor)(struct sk\_buff \*skb);

...

unsigned int truesize;

The SKB 'destructor' and 'truesize' fields are used for socket buffer accounting.

atomic\_t users;

We reference count SKB objects using the 'users' field.

unsigned char \*head,

\*data,

\*tail,

\*end;

These four pointers provide the core management of the linear packet data area of an SKB.

**SK\_BUFF Head**

Most SKBs are stored on a list, whose head is implemented by 'struct sk\_buff\_head':

I am just providing just general struct code of it:-

struct sk\_buff\_head {

/\* These two members must be first. \*/

struct sk\_buff \*next;

struct sk\_buff \*prev;

\_\_u32 qlen;

spinlock\_t lock;

};

The first two members implement the doubly linked list handling. The third member, 'qlen', keeps track of how many packets are on this list. The fourth member, 'lock', is used for SMP protection.Here SMP i.e. **Shared Mesh Protection is used** as a **means** of increasing the **network** resiliency without incurring failures.

There are many operations one can perform on SKBs wrt. List tracking.

Here are the primary interfaces used in the code:

int skb\_queue\_empty(const struct sk\_buff\_head \*list);

This function will check that whether the given list is empty?

struct sk\_buff \*skb\_peek(struct sk\_buff\_head \*list\_);

Return, but do not remove, the first SKB on the list, else NULL. We should only use this function on an SKB list we have exclusive access to( I mean our own private acess), else a thread in another context could remove the SKB this function returns.

struct sk\_buff \*skb\_peek\_tail(struct sk\_buff\_head \*list\_);

It is almost Exactly like skb\_peek(), except it returns the last SKB on the list.

\_\_u32 skb\_queue\_len(const struct sk\_buff\_head \*list\_);

It Returns the number of packets on the given list.

void skb\_queue\_head\_init(struct sk\_buff\_head \*list);

Initialize an sk\_buff\_head object. If we dynamically allocate a data structure with an sk\_buff\_head member in it, we should pass it to this function before trying to use it.

**SOCK FINAL**

When sockets send and receive packets, they must be properly charged for the amount of system memory consumed by that packet.

So different attributes of it are:

'skb->truesize' : We keep track of how many bytes of system memory are consumed by a packet using this. This is the total of how large a data buffer we allocated for the packet, plus the size of 'struct sk\_buff' itself.

'sk->sk\_sndbuf' : The total number of bytes of send packet memory, a socket may use is limited by this.

'sk->sk\_rcvbuf' : The total number of bytes of for receive packets memory, a socket may use is limited by this.

**{ Actually above one are used in used for datagram protocols. Stream protocols, such as TCP, use a more elaborate scheme. }**

**Functions:**

1.

static inline void skb\_set\_owner\_r(struct sk\_buff \*skb, struct sock \*sk)

{

skb->sk = sk;

skb->destructor = sock\_rfree;

atomic\_add(skb->truesize, &sk->sk\_rmem\_alloc);

}

This function is used when a received packet is to be charged to a socket.

First,It sets 'skb->sk' to the socket.Secondly, it calls up a destructor function, and accounts the data bytes in 'sk->sk\_rmem\_alloc' for memory allocation.

2.

void sock\_rfree(struct sk\_buff \*skb)

{

struct sock \*sk = skb->sk;

atomic\_sub(skb->truesize, &sk->sk\_rmem\_alloc);

}

It is destructor function.

Later when the packet is freed up the destructor is invoked. So destructor 'sock\_rfree()' is invoked to do this operation. It releases the buffer allocate space from 'sk->sk\_rmem\_alloc'.

There are many other function which are not so important for forwarding.

**IP\_EXCEEDS\_MTU**

**static** [**bool**](http://lxr.free-electrons.com/ident?i=bool)[**ip\_exceeds\_mtu**](http://lxr.free-electrons.com/ident?i=ip_exceeds_mtu)**(const struct** [**sk\_buff**](http://lxr.free-electrons.com/ident?i=sk_buff) **\***[**skb**](http://lxr.free-electrons.com/ident?i=skb)**, unsigned int** [**mtu**](http://lxr.free-electrons.com/ident?i=mtu)**)**

**{**

**if (**[**skb**](http://lxr.free-electrons.com/ident?i=skb)**->**[**len**](http://lxr.free-electrons.com/ident?i=len) **<=** [**mtu**](http://lxr.free-electrons.com/ident?i=mtu)**)**

**return** [**false**](http://lxr.free-electrons.com/ident?i=false)**;**

**if (**[**unlikely**](http://lxr.free-electrons.com/ident?i=unlikely)**((**[**ip\_hdr**](http://lxr.free-electrons.com/ident?i=ip_hdr)**(**[**skb**](http://lxr.free-electrons.com/ident?i=skb)**)->frag\_off &** [**htons**](http://lxr.free-electrons.com/ident?i=htons)**(**[**IP\_DF**](http://lxr.free-electrons.com/ident?i=IP_DF)**)) == 0))**

**return** [**false**](http://lxr.free-electrons.com/ident?i=false)**;**

***/\* original fragment exceeds mtu and DF is set \*/***

**if (**[**unlikely**](http://lxr.free-electrons.com/ident?i=unlikely)**(**[**IPCB**](http://lxr.free-electrons.com/ident?i=IPCB)**(**[**skb**](http://lxr.free-electrons.com/ident?i=skb)**)->frag\_max\_size >** [**mtu**](http://lxr.free-electrons.com/ident?i=mtu)**))**

**return** [**true**](http://lxr.free-electrons.com/ident?i=true)**;**

**if (**[**skb**](http://lxr.free-electrons.com/ident?i=skb)**->ignore\_df)**

**return** [**false**](http://lxr.free-electrons.com/ident?i=false)**;**

**if (**[**skb\_is\_gso**](http://lxr.free-electrons.com/ident?i=skb_is_gso)**(**[**skb**](http://lxr.free-electrons.com/ident?i=skb)**) &&** [**skb\_gso\_network\_seglen**](http://lxr.free-electrons.com/ident?i=skb_gso_network_seglen)**(**[**skb**](http://lxr.free-electrons.com/ident?i=skb)**) <=** [**mtu**](http://lxr.free-electrons.com/ident?i=mtu)**)**

**return** [**false**](http://lxr.free-electrons.com/ident?i=false)**;**

**return** [**true**](http://lxr.free-electrons.com/ident?i=true)**;**

**}**

**This function is used to check whether ip socket buffer (skb) exceeds mtu(maximum transmission unit)**

**static** [**bool**](http://lxr.free-electrons.com/ident?i=bool)[**ip\_exceeds\_mtu**](http://lxr.free-electrons.com/ident?i=ip_exceeds_mtu)**(const struct** [**sk\_buff**](http://lxr.free-electrons.com/ident?i=sk_buff) **\***[**skb**](http://lxr.free-electrons.com/ident?i=skb)**, unsigned int** [**mtu**](http://lxr.free-electrons.com/ident?i=mtu)**)**

In this function definition socket buffer (sk\_buff) and mtu (maximum transmission unit) are taken as input parameters for the function.

**if (**[**skb**](http://lxr.free-electrons.com/ident?i=skb)**->**[**len**](http://lxr.free-electrons.com/ident?i=len) **<=** [**mtu**](http://lxr.free-electrons.com/ident?i=mtu)**)**

**return** [**false**](http://lxr.free-electrons.com/ident?i=false)**;**

The total number of bytes in the packet is 'len'. So skb length is less than mtu then function returns false.

**if (**[**unlikely**](http://lxr.free-electrons.com/ident?i=unlikely)**((**[**ip\_hdr**](http://lxr.free-electrons.com/ident?i=ip_hdr)**(**[**skb**](http://lxr.free-electrons.com/ident?i=skb)**)->frag\_off &** [**htons**](http://lxr.free-electrons.com/ident?i=htons)**(**[**IP\_DF**](http://lxr.free-electrons.com/ident?i=IP_DF)**)) == 0))**

**return** [**false**](http://lxr.free-electrons.com/ident?i=false)**;**

**#define** [**IP\_DF**](http://lxr.free-electrons.com/ident?i=IP_DF) **0x4000 */\* Flag: "Don't Fragment" \*/***

Here It is preprocessor macro and unlikely condition is executed and comes out to be zero then function returns false i.e not exceeded. Here ip\_hdr is refering to fragment offset whether equal to zero as it will certainly results in skb less than size of mtu.

***/\* original fragment exceeds mtu and DF is set \*/***

**if (**[**unlikely**](http://lxr.free-electrons.com/ident?i=unlikely)**(**[**IPCB**](http://lxr.free-electrons.com/ident?i=IPCB)**(**[**skb**](http://lxr.free-electrons.com/ident?i=skb)**)->frag\_max\_size >** [**mtu**](http://lxr.free-electrons.com/ident?i=mtu)**))**

**return** [**true**](http://lxr.free-electrons.com/ident?i=true)**;**

**#define** [**IPCB**](http://lxr.free-electrons.com/ident?i=IPCB)**(**[**skb**](http://lxr.free-electrons.com/ident?i=skb)**) ((struct** [**inet\_skb\_parm**](http://lxr.free-electrons.com/ident?i=inet_skb_parm)**\*)((**[**skb**](http://lxr.free-electrons.com/ident?i=skb)**)->**[**cb**](http://lxr.free-electrons.com/ident?i=cb)**))**

So IPCB is defined as preprocessor macro as above and skb -> cb refers to SKB control block. It is an storage area usable by protocols, and to store private per-packet information.

Skb’s control block ‘s fragment max\_size is grater than mtu it is obvious that it exceeded mtu.So it returns true here.

**if (**[**skb**](http://lxr.free-electrons.com/ident?i=skb)**->ignore\_df)**

**return** [**false**](http://lxr.free-electrons.com/ident?i=false)**;**

If socket buffer skb’s ignoring don’t fragment bit is set i.e true then it refers to not exceeding mtu. So condition returns false.

**if (**[**skb\_is\_gso**](http://lxr.free-electrons.com/ident?i=skb_is_gso)**(**[**skb**](http://lxr.free-electrons.com/ident?i=skb)**) &&** [**skb\_gso\_network\_seglen**](http://lxr.free-electrons.com/ident?i=skb_gso_network_seglen)**(**[**skb**](http://lxr.free-electrons.com/ident?i=skb)**) <=** [**mtu**](http://lxr.free-electrons.com/ident?i=mtu)**)**

**return** [**false**](http://lxr.free-electrons.com/ident?i=false)**;**

skb\_gso\_network\_seglen - Return length of individual segments of a gso packet

Skb\_gso\_network\_seglen is used to determine the real size of the individual segments, including Layer3 (IP, IPv6) and L4 headers (TCP/UDP).

The gso\_size is the size the hardware should fragment the TCP data. And [**skb\_is\_gso**](http://lxr.free-electrons.com/ident?i=skb_is_gso)**(**[**skb**](http://lxr.free-electrons.com/ident?i=skb)**) returns**

**Gso\_type and skb\_gso\_TCPV6. If both of them are less than or equal to mtu ,**

**Function returns false.**

**return** [**true**](http://lxr.free-electrons.com/ident?i=true)**;**

If any of the conditions are not satisfied , whole of the function returns true i.e ie ip socket buffer skb has exceeds mtu.

**IP\_FORWARD\_FINISH**

FUNCTION :

static int ip\_forward\_finish(struct sock \*sk, struct sk\_buff \*skb)

{

struct ip\_options \*opt = &(IPCB(skb)->opt);

IP\_INC\_STATS\_BH(dev\_net(skb\_dst(skb)->dev), IPSTATS\_MIB\_OUTFORWDATAGRAMS);

IP\_ADD\_STATS\_BH(dev\_net(skb\_dst(skb)->dev), IPSTATS\_MIB\_OUTOCTETS, skb->len);

if (unlikely(opt->optlen))

ip\_forward\_options(skb);

skb\_sender\_cpu\_clear(skb);

return dst\_output\_sk(sk, skb);

}

1. static int ip\_forward\_finish(struct sock \*sk, struct sk\_buff \*skb);

This function is responsible to complete the routing table once it reaches every router. It accepts the arguments :

1. struct sock \*sk : a structure object(pointer) of structure “sock” which is responsible for the data variables managing the socket creation, use and close.
2. struct sk\_buff \*skb : a structure object(pointer) of structure “sk\_buff” which defines a packet structure in the network.

1. struct ip\_options \*opt = &(IPCB(skb)->opt);
2. Here the object of the structure “ip\_options” initializes itself to the address of the data variable “opt” corresponding to the structure “IPCB”.

IPCB is defined as a macro in the header file “net/ip.h”

#define [IPCB](http://lxr.free-electrons.com/ident?i=IPCB)([skb](http://lxr.free-electrons.com/ident?i=skb)) ((struct [inet\_skb\_parm](http://lxr.free-electrons.com/ident?i=inet_skb_parm)\*)(([skb](http://lxr.free-electrons.com/ident?i=skb))->[cb](http://lxr.free-electrons.com/ident?i=cb)))

1. “inet\_skb\_parm” is the structure that holds the details of the flags, ip options and the packet size.

struct [inet\_skb\_parm](http://lxr.free-electrons.com/ident?i=inet_skb_parm)

{

struct ip\_options opt;

unsigned char flags;

#define IPSKB\_FORWARDED BIT(0)

#define IPSKB\_XFRM\_TUNNEL\_SIZE BIT(1)

#define IPSKB\_XFRM\_TRANSFORMED BIT(2)

#define IPSKB\_FRAG\_COMPLETE BIT(3)

#define IPSKB\_REROUTED BIT(4)

#define IPSKB\_DOREDIRECT BIT(5)

#define IPSKB\_FRAG\_PMTU BIT(6)

u16 frag\_max\_size;

};

1. “ip\_options” is the structure that holds different ip parameters and options that have to be initialized in order to define the proper ip format for the packets.

struct [ip\_options](http://lxr.free-electrons.com/ident?i=ip_options)

{

[\_\_be32](http://lxr.free-electrons.com/ident?i=__be32) faddr;

[\_\_be32](http://lxr.free-electrons.com/ident?i=__be32) nexthop;

unsigned char [optlen](http://lxr.free-electrons.com/ident?i=optlen);

unsigned char [srr](http://lxr.free-electrons.com/ident?i=srr);

unsigned char rr;

unsigned char [ts](http://lxr.free-electrons.com/ident?i=ts);

unsigned char is\_strictroute:1,

srr\_is\_hit:1,

is\_changed:1,

rr\_needaddr:1,

ts\_needtime:1,

ts\_needaddr:1;

unsigned char router\_alert;

unsigned char cipso;

unsigned char \_\_pad2;

unsigned char \_\_data[0];

};

1. IP\_INC\_STATS\_BH(dev\_net(skb\_dst(skb)->dev), IPSTATS\_MIB\_OUTFORWDATAGRAMS);

IP\_INC\_STATS\_BH() is a macro defined in net/ip.h header file and is used to determine amount of padding added to the ethernet frame.

1. IP\_ADD\_STATS\_BH(dev\_net(skb\_dst(skb)->dev), IPSTATS\_MIB\_OUTOCTETS, skb->len);

After determining the amount of padding to be added to the ethernet frame, now this macro adds the determined padding to the packet.

1. if (unlikely(opt->optlen))

ip\_forward\_options(skb);

Here the unlikely() function checks the error range of the given parameter and if that is true, calls the function ip\_forward\_options(skb).

1. ip\_forward\_options(skb)-

void ip\_forward\_options(struct sk\_buff \*skb)

{

struct ip\_options \*opt = &(IPCB(skb)->opt);

unsigned char \*optptr;

struct rtable \*rt = skb\_rtable(skb);

unsigned char \*raw = skb\_network\_header(skb);

if (opt->rr\_needaddr) {

optptr = (unsigned char \*)raw + opt->rr;

ip\_rt\_get\_source(&optptr[optptr[2]-5], skb, rt);

opt->is\_changed = 1;

}

if (opt->srr\_is\_hit) {

int srrptr, srrspace;

optptr = raw + opt->srr;

for ( srrptr = optptr[2], srrspace = optptr[1];

srrptr <= srrspace;

srrptr += 4

) {

if (srrptr + 3 > srrspace)

break;

if (memcmp(&opt->nexthop, &optptr[srrptr-1], 4) == 0)

break;

}

if (srrptr + 3 <= srrspace) {

opt->is\_changed = 1;

ip\_hdr(skb)->daddr = opt->nexthop;

ip\_rt\_get\_source(&optptr[srrptr-1], skb, rt);

optptr[2] = srrptr+4;

} else {

net\_crit\_ratelimited(*"%s(): Argh! Destination lost!\n"*,

\_\_func\_\_);

}

if (opt->ts\_needaddr) {

optptr = raw + opt->ts;

ip\_rt\_get\_source(&optptr[optptr[2]-9], skb, rt);

opt->is\_changed = 1;

}

}

if (opt->is\_changed) {

opt->is\_changed = 0;

ip\_send\_check(ip\_hdr(skb));

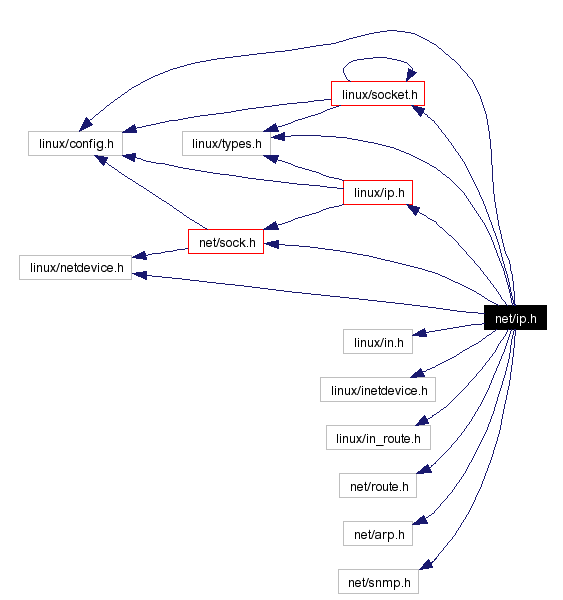
}

}

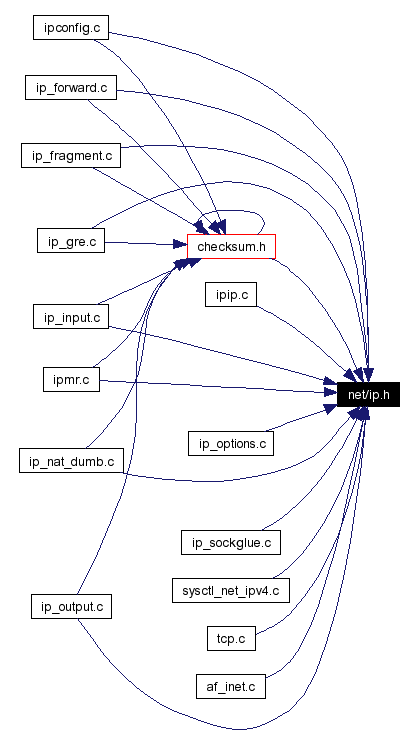
This function is responsible for packet forwarding with the help of routing table but after the hop count of 4, it stops the packet forwarding.

**IP**

Ip.h important header which is derived from net and deals with inturn connected headers like sock.h,socket.h,lenux/ip.h as shown in figure:



Added to this there are many files directly or indirectly include this file as shown below:



So our file ip\_forward.c is connected directly and internally (via checksum.c) to net/ip.h

Some of important structures which effect ip\_forward.c from ip.h are:

Ip\_options

Ip\_hdr

And functions are:

ip\_call\_ra\_chain

ip\_rt\_send\_redirect

ip\_hdr

ip\_decrease\_ttl

ip\_exceeds\_mtu

ip\_dst\_mtu\_maybe\_forward

**ip\_options:**

struct [ip\_options](http://www.cse.scu.edu/~dclark/am_256_graph_theory/linux_2_6_stack/structip__options.html) {

\_\_u32 faddr; /\* Saved first hop address \*/

unsigned char optlen;

unsigned char srr;

unsigned char rr;

unsigned char ts;

unsigned char is\_setbyuser:1, /\* Set by setsockopt? \*/

is\_data:1, /\* Options in \_\_data, rather than skb \*/

is\_strictroute:1, /\* Strict source route \*/

srr\_is\_hit:1, /\* Packet destination addr was our one \*/

is\_changed:1, /\* IP checksum more not valid \*/

rr\_needaddr:1, /\* Need to record addr of outgoing dev \*/

ts\_needtime:1, /\* Need to record timestamp \*/

ts\_needaddr:1; /\* Need to record addr of outgoing dev \*/

unsigned char router\_alert;

unsigned char \_\_pad1;

unsigned char \_\_pad2;

unsigned char \_\_data[0];

};

“ip\_options” is the structure that holds different ip parameters and options that have to be initialized in order to define the proper ip format for the packets.

Set or get the IP options to be sent with every packet from the socket. The arguments are a pointer to a memory buffer containing the options and the option length. The [**setsockopt**](http://linux.die.net/man/2/setsockopt) call sets the IP options associated with a socket. The maximum option size for IPv4 is 40 bytes. When the initial connection request packet for a **SOCK\_STREAM** socket contains IP options, the IP options will be set automatically to the options from the initial packet with routing headers reversed. Incoming packets are not allowed to change options after the connection is established. The processing of all incoming source routing options is disabled by default and can be enabled by using the *accept\_source\_route /proc* interface. Other options like timestamps are still handled. For datagram sockets, IP options can be only set by the local user. Calling [**getsockopt**](http://linux.die.net/man/2/getsockopt) with **IP\_OPTIONS** puts the current IP options used for sending into the supplied buffer.

**struct** [**iphdr**](http://lxr.free-electrons.com/ident?i=iphdr) **{**

**#if defined(**[**\_\_LITTLE\_ENDIAN\_BITFIELD**](http://lxr.free-electrons.com/ident?i=__LITTLE_ENDIAN_BITFIELD)**)**

[**\_\_u8**](http://lxr.free-electrons.com/ident?i=__u8) **ihl:4,**

[**version**](http://lxr.free-electrons.com/ident?i=version)**:4;**

**#elif defined (**[**\_\_BIG\_ENDIAN\_BITFIELD**](http://lxr.free-electrons.com/ident?i=__BIG_ENDIAN_BITFIELD)**)**

[**\_\_u8**](http://lxr.free-electrons.com/ident?i=__u8)[**version**](http://lxr.free-electrons.com/ident?i=version)**:4,**

**ihl:4;**

**#else**

**#error *"Please fix <asm/byteorder.h>"***

**#endif**

[**\_\_u8**](http://lxr.free-electrons.com/ident?i=__u8) **tos;**

[**\_\_be16**](http://lxr.free-electrons.com/ident?i=__be16) **tot\_len;**

[**\_\_be16**](http://lxr.free-electrons.com/ident?i=__be16)[**id**](http://lxr.free-electrons.com/ident?i=id)**;**

[**\_\_be16**](http://lxr.free-electrons.com/ident?i=__be16) **frag\_off;**

[**\_\_u8**](http://lxr.free-electrons.com/ident?i=__u8)[**ttl**](http://lxr.free-electrons.com/ident?i=ttl)**;**

[**\_\_u8**](http://lxr.free-electrons.com/ident?i=__u8)[**protocol**](http://lxr.free-electrons.com/ident?i=protocol)**;**

[**\_\_sum16**](http://lxr.free-electrons.com/ident?i=__sum16)[**check**](http://lxr.free-electrons.com/ident?i=check)**;**

[**\_\_be32**](http://lxr.free-electrons.com/ident?i=__be32)[**saddr**](http://lxr.free-electrons.com/ident?i=saddr)**;**

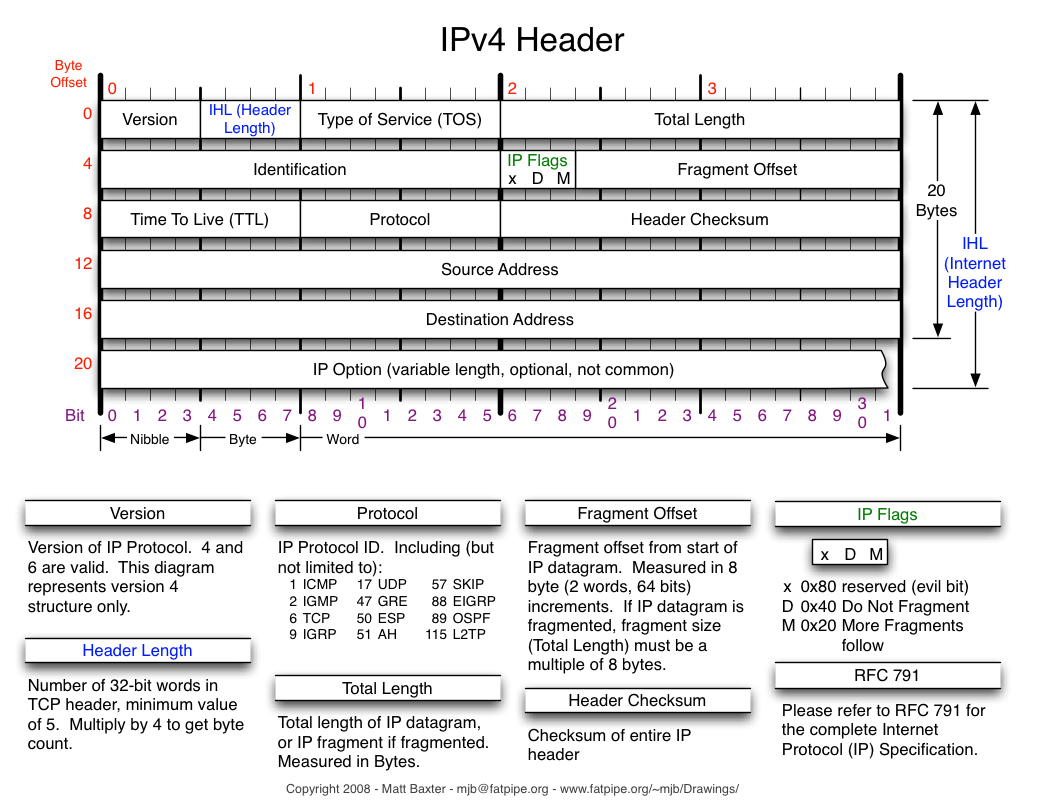
[**\_\_be32**](http://lxr.free-electrons.com/ident?i=__be32)[**daddr**](http://lxr.free-electrons.com/ident?i=daddr)**;**

***/\*The options start here. \*/***

**};**

So this structure defines all the components of structure one header of ip would have.

All the parameter are set to packet derived from it and used all along the trasmission to decode required information.



**Functions:**

ip\_call\_ra\_chain

If the router \_alert option is set , the ip\_call\_ra\_chain() method should be invoked to handle packet.When calling setsocket() with IP\_ROUTER\_Alert on raw socket, socket is added to global list named ip\_ra\_chain(net/ip.h) .This function delivers packets to all raw sockets.

Forwarding

**int ip\_forward(struct sk\_buff \*skb** )

* Decreases the ttl in the ip header; If the ttl is <=1, the methods send ICMP message (ICMP\_TIME\_EXCEEDED) with ICMP\_EXC\_TTL ("TTL count exceeded"), and drops the packet.
* Calls NF\_HOOK(PF\_INET,NF\_IP\_FORWARD, skb, skb->dev,rt- >u.dst.dev, ip\_forward\_finish);

ip\_forward\_finish()

sends the packet out by calling dst\_output(skb).

* dst\_output(skb) is just a wrapper, which calls skb->dst->output(skb).

**Sending the packet:**

We need to perform routing lookup also in the case of transmission.

* There are cases when we perform two lookups, like in ipip tunnels.
* Handling of sending a packet is done by ip\_route\_output\_key().
* In case of a cache miss, we calls ip\_route\_output\_slow(), which looks in the routing table (by calling fib\_lookup(), as also is done in ip\_route\_input\_slow().)
* If the packet is for a remote host, we set dst->output to ip\_output()
* ip\_output() will call ip\_finish\_output() – This is the NF\_IP\_POST\_ROUTING point.

**ip\_finish\_output()** will eventually send the packet from a neighbor by: – dst->neighbour->output(skb) – arp\_bind\_neighbour() sees to it that the L2 address of the next hop will be known.

**ip\_decrease\_ttl()**

Each node that forwards the packet should decrese the ttl andas a result of ttl change , checksum is also updated accordingly in the ip\_decrease\_ttl() method.