**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.