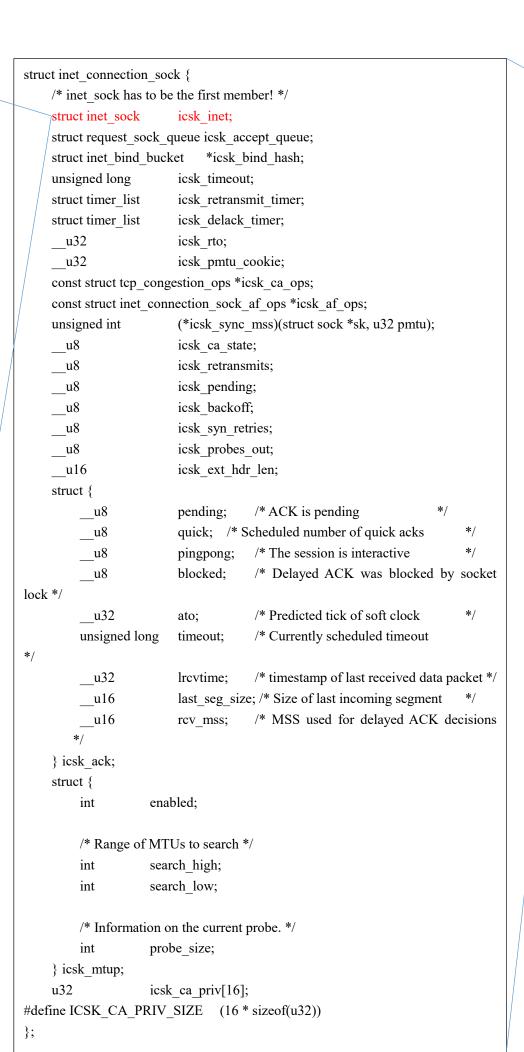
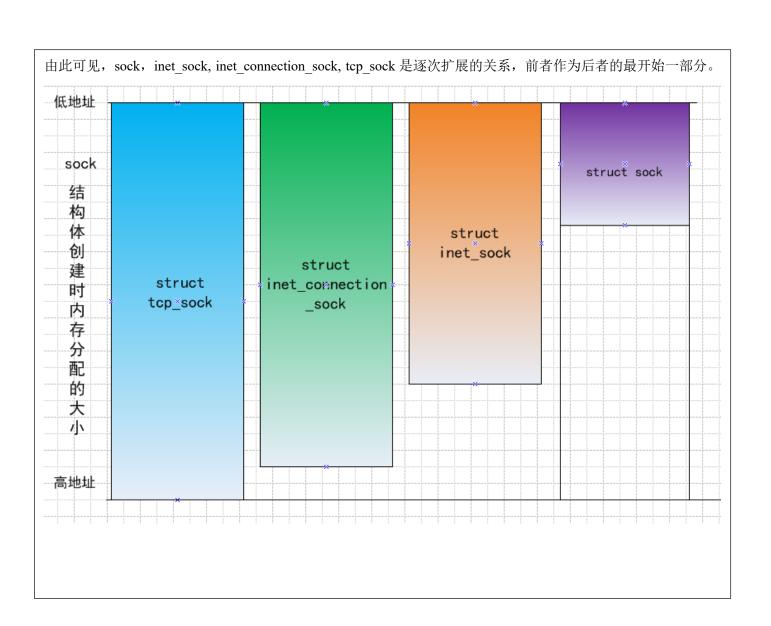
```
struct inet sock {
    /* sk and pinet6 has to be the first two members of inet sock */
    struct sock
#if defined(CONFIG IPV6) || defined(CONFIG IPV6 MODULE)
    struct ipv6 pinfo *pinet6;
#endif
    /* Socket demultiplex comparisons on incoming packets. */
    __be32
    __be32
                       rcv saddr;
    __be16
                       dport;
    __u16
                       num;
    __be32
                       saddr;
    __s16
                       uc ttl;
     __u16
                       cmsg flags;
    struct ip_options
                       *opt;
    __be16
                       sport;
    __u16
                       id;
     __u8
                       tos;
     __u8
                       mc ttl;
    __u8
                       pmtudisc;
    __u8
                       recverr:1.
                   is icsk:1,
                  freebind:1,
                  hdrincl:1,
                   mc loop:1;
    int
                  mc index;
     be32
                       mc addr;
    struct ip_mc_socklist
                           *mc list;
    struct {
         unsigned int
                            flags;
         unsigned int
                            fragsize;
         struct ip_options
         struct rtable
                            *rt;
                       length; /* Total length of all frames */
         int
         __be32
         struct flowi
    } cork;
};
```





Tcp sock 是 TCP 协议特有的 socket, 在 inet connection sock 的基础上进行扩展,填加 了 tcp 滑动窗口中与流量控制、拥塞控制有关的一些属性,比如接收窗口、发送窗口、 拥塞窗口大小、慢启动门限值, rtt 的估计等; struct tcp_sock { /* inet_connection_sock has to be the first member of tcp_sock */ struct inet connection sock inet conn; u16 tcp_header_len; /* Bytes of tcp header to send */ u16 xmit_size_goal; /* Goal for segmenting output packets */ * Header prediction flags * $0x5?10 \ll 16 + \text{snd}$ wnd in net byte order be32 pred flags; u32 rcv nxt; /* What we want to receive next */ u32 snd nxt; /* Next sequence we send u32 snd una; /* First byte we want an ack for */ u32 snd sml; /* Last byte of the most recently transmitted small packet */ u32 rcv_tstamp; /* timestamp of last received ACK (for keepalives) */ u32 lsndtime; /* timestamp of last sent data packet (for restart window) */ /* Data for direct copy to user */ struct { struct sk_buff_head prequeue; struct task struct struct iovec *iov; memory; len; #ifdef CONFIG NET DMA /* members for async copy */ struct dma chan *dma chan; wakeup; struct dma_pinned_list *pinned_list; dma_cookie_t dma_cookie; #endif } ucopy; u32 snd_wl1; /* Sequence for window update u32 snd wnd; /* The window we expect to receive */ u32 max_window; /* Maximal window ever seen from peer */ u32 mss cache;/* Cached effective mss, not including SACKS */ u32 window clamp; /* Maximal window to advertise u32 rcv ssthresh; /* Current window clamp u32 frto_highmark; /* snd_nxt when RTO occurred */ u8 reordering; /* Packet reordering metric. u8 frto counter; /* Number of new acks after RTO */ u8 nonagle; /* Disable Nagle algorithm? u8 keepalive_probes; /* num of allowed keep alive probes /* RTT measurement */ /* smoothed round trip time << 3 */ u32 srtt; u32 mdev; /* medium deviation u32 mdev_max; /* maximal mdev for the last rtt period /* smoothed mdev_max u32 rttvar; /* sequence number to update rttvar u32 rtt_seq; u32 packets out; /* Packets which are "in flight" u32 left out; /* Packets which leaved network u32 retrans_out; /* Retransmitted packets out struct tcp_options_received rx_opt; u32 snd_ssthresh; /* Slow start size threshold u32 snd cwnd; /* Sending congestion window u16 snd cwnd cnt; /* Linear increase counter u16 snd cwnd clamp; /* Do not allow snd cwnd to grow above this */ u32 snd cwnd used; u32 snd_cwnd_stamp; struct sk_buff_head out_of_order_queue; /* Out of order segments go here */ u32 rcv wnd; /* Current receiver window u32 rcv_wup; /* rcv_nxt on last window update sent */ u32 write seq; /* Tail(+1) of data held in tcp send buffer */ u32 pushed_seq; /* Last pushed seq, required to talk to windows */ u32 copied seq; /* Head of yet unread data /* SACKs data */ struct tcp_sack_block_duplicate_sack[1]; /* D-SACK_block */ struct tcp_sack_block selective_acks[4]; /* The SACKS themselves*/ struct tcp sack block recv sack cache[4]; /* from STCP, retrans queue hinting */ struct sk buff* lost skb hint; struct sk buff *scoreboard skb hint; struct sk_buff *retransmit_skb_hint; struct sk buff *forward skb hint; struct sk buff *fastpath skb hint; fastpath_cnt_hint; lost_cnt_hint; retransmit_cnt_hint; forward ent hint; u16 advmss; /* Advertised MSS u16 prior ssthresh; /* ssthresh saved at recovery start u32 lost_out; /* Lost packets u32 sacked_out; /* SACK'd packets u32 fackets_out; /* FACK'd packets u32 high_seq; /* snd_nxt at onset of congestion */ u32 retrans_stamp; /* Timestamp of the last retransmit, also used in SYN-SENT to remember stamp of the first SYN. */ u32 undo_marker; /* tracking retrans started here. */ int undo retrans; /* number of undoable retransmissions. */ u32 urg_seq; /* Seq of received urgent pointer */ u16 urg_data; /* Saved octet of OOB data and control flags */ u8 urg_mode; /* In urgent mode u8 ecn_flags; /* ECN status bits. u32 snd up; /* Urgent pointer */

u32 total_retrans; /* Total retransmits for entire connection */
u32 bytes_acked; /* Appropriate Byte Counting - RFC3465 */