```
struct inet_sock {
	/* sk and pinet6 has to be the first two members of inet_sock */
	struct sock		sk;
#if defined(CONFIG_IPV6) || defined(CONFIG_IPV6_MODULE)
	struct ipv6_pinfo	*pinet6;
#endif
	/* Socket demultiplex comparisons on incoming packets. */
	__be32			daddr;
	__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	*opt;
		struct rtable		*rt;
		int			length; /* Total length of all frames */
		__be32			addr;
		struct flowi		fl;
	} cork;
};
```

```
struct inet_connection_sock {
	/* inet_sock has to be the first member! */
	struct inet_sock	  icsk_inet;
	struct request_sock_queue icsk_accept_queue;
	struct inet_bind_bucket	  *icsk_bind_hash;
	unsigned long		  icsk_timeout;
	struct timer_list	  icsk_retransmit_timer;
	struct timer_list	  icsk_delack_timer;
	__u32			  icsk_rto;
	__u32			  icsk_pmtu_cookie;
	const struct tcp_congestion_ops *icsk_ca_ops;
	const struct inet_connection_sock_af_ops *icsk_af_ops;
	unsigned int		  (*icsk_sync_mss)(struct sock *sk, u32 pmtu);
	__u8			  icsk_ca_state;
	__u8			  icsk_retransmits;
	__u8			  icsk_pending;
	__u8			  icsk_backoff;
	__u8			  icsk_syn_retries;
	__u8			  icsk_probes_out;
	__u16			  icsk_ext_hdr_len;
	struct {
		__u8		  pending;	 /* ACK is pending			*/
		__u8		  quick;	 /* Scheduled number of quick acks	*/
		__u8		  pingpong;	 /* The session is interactive		*/
		__u8		  blocked;	 /* Delayed ACK was blocked by socket
lock */
		__u32		  ato;		 /* Predicted tick of soft clock	*/
		unsigned long	  timeout;	 /* Currently scheduled timeout
*/
		__u32		  lrcvtime;	 /* timestamp of last received data packet */
		__u16		  last_seg_size; /* Size of last incoming segment	*/
		__u16		  rcv_mss;	 /* MSS used for delayed ACK decisions
		*/
	} icsk_ack;
	struct {
		int		  enabled;

		/* Range of MTUs to search */
		int		  search_high;
		int		  search_low;

		/* Information on the current probe. */
		int		  probe_size;
	} icsk_mtup;
	u32			  icsk_ca_priv[16];
#define ICSK_CA_PRIV_SIZE	(16 * sizeof(u32))
};
```

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 << 16 + 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	*task;
		struct iovec		*iov;
		int			memory;
		int			len;
#ifdef CONFIG_NET_DMA
		/* members for async copy */
		struct dma_chan		*dma_chan;
		int			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 */
	u32	srtt;		/* smoothed round trip time << 3	*/
	u32	mdev;		/* medium deviation			*/
	u32	mdev_max;	/* maximal mdev for the last rtt period	*/
	u32	rttvar;		/* smoothed mdev_max			*/
	u32	rtt_seq;	/* sequence number to update rttvar	*/
	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;
	int	fastpath_cnt_hint;
	int	lost_cnt_hint;
	int	retransmit_cnt_hint;
	int	forward_cnt_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 */
```

由此可见，sock，inet_sock, inet_connection_sock, tcp_sock 是逐次扩展的关系，前者作为后者的最开始一部分。