TCP Data structures

struct tcp_opt {
    int  tcp_header_len; /* Bytes of tcp header to send */
    __u32 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) */
    __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 */
} ack;
/* Data for direct copy to user */

struct {
    struct sk_buff_head prequeue;
    int memory;
    struct task_struct *task;
    struct iovec *iov;
    int len;
} 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 pmtu_cookie; /* Last pmtu seen by socket */
__u32 snd_wnd; /* The window we expect to receive */
__u32 max_window; /* Maximal window ever seen from peer */
__u32 pmtu_cookie; /* Last pmtu seen by socket */
__u16 mss_cache; /* Cached effective mss, not including SACKS */
__u16 mss_clamp; /* Maximal mss, negotiated at connection setup */
__u16 ext_header_len; /* Network protocol overhead (IP/IPv6 options) */
__u8 ca_state; /* State of fast-retransmit machine */
__u8 retransmits; /* Number of unrecovered RTO timeouts. */
__u8 reordering; /* Packet reordering metric. */
__u8 queue_shrunk; /* Write queue has been shrunk recently. */
__u8 defer_accept; /* User waits for some data after accept() */
__u8 backoff; /* backoff */
__u32 srtt; /* smothed 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 rto; /* retransmit timeout */
__u32 packets_out; /* Packets which are "in flight" */
__u32 left_out; /* Packets which leaved network */
__u32 retrans_out; /* Retransmitted packets out */

/* RTT measurement */
Slow start and congestion control (see also Nagle, and Karn & Partridge)

- `__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;`

- `unsigned long timeout;` /* Two commonly used timers in both sender and receiver paths. */
- `struct timer_list retransmit_timer;` /* Resend (no ack) */
- `struct timer_list delack_timer;` /* Ack delay */

- `struct sk_buff_head out_of_order_queue;` /* Out of order segments go here */

- `struct tcp_func *af_specific;` /* Operations which are AF_INET{4,6} specific */
- `struct sk_buff *send_head;` /* Front of stuff to transmit */
- `struct page *sndmsg_page;` /* Cached page for sendmsg */
- `u32 sndmsg_off;` /* Cached offset for sendmsg */

- `__u32 rcv_wnd;` /* Current receiver window */
- `__u32 rcv_wup;` /* rcv_nxt on last window update sent */
- `__u32 write_seq;` /* Tail(+) 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 */
/*
 * Options received (usually on last packet, some only on SYN packets).
 */
char tstamp_ok,      /* TIMESTAMP seen on SYN packet */
wscale_ok,          /* Wscale seen on SYN packet */
sack_ok;            /* SACK seen on SYN packet */
char saw_tstamp;    /* Saw TIMESTAMP on last packet */
__u8 snd_wscale;    /* Window scaling received from sender */
__u8 rcv_wscale;    /* Window scaling to send to receiver */
__u8 nonagle;       /* Disable Nagle algorithm? */
__u8 keepalive_probes; /* num of allowed keep alive probes */

PaWS/RTTM data */
__u32 rcv_tsval;    /* Time stamp value */
__u32 rcv_tsecr;    /* Time stamp echo reply */
__u32 ts_recent;    /* Time stamp to echo next */
long ts_recent_stamp; /* Time we stored ts_recent (for aging) */

SACKs data */
__u16 user_mss;     /* mss requested by user in ioctl */
__u8 dsack;         /* D-SACK is scheduled */
__u8 eff_sacks;     /* Size of SACK array to send with next packet */
struct tcp_sack_block duplicate_sack[1]; /* D-SACK block */
struct tcp_sack_block selective_acks[4]; /* The SACKS themselves*/
__u32 window_clamp; /* Maximal window to advertise */
__u32 rcv_ssthresh; /* Current window clamp */
__u8 probes_out;    /* unanswered 0 window probes */
__u8 num_sacks;     /* Number of SACK blocks */
__u16 advmss;       /* Advertised MSS */
__u8 syn_retries;   /* num of allowed syn retries */
__u8 ecn_flags;     /* ECN status bits. */
__u16 prior_ssthresh; /* ssthresh saved at recovery start */
__u32 lost_out;     /* Lost packets */
__u32 sacked_out;   /* SACK'd packets */
__u32 packets_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 pending; /* Scheduled timer event */
__u8 urg_mode; /* In urgent mode */
__u32 snd_up; /* Urgent pointer */
/* The syn_wait_lock is necessary only to avoid tcp_get_info having
 * to grab the main lock sock while browsing the listening hash
 * (otherwise it's deadlock prone).
 * This lock is acquired in read mode only from tcp_get_info() and
 * it's acquired in write mode _only_ from code that is actively
 * changing the syn_wait_queue. All readers that are holding
 * the master sock lock don't need to grab this lock in read mode
 * too as the syn_wait_queue writes are always protected from
 * the main sock lock.
 */

rwlock_t syn_wait_lock;
struct tcp_listen_opt *listen_opt;

/* FIFO of established children */
struct open_request *accept_queue;
struct open_request *accept_queue_tail;

int write_pending; /* A write to socket waits to start. */

unsigned int keepalive_time; /* time before keep alive takes place */
unsigned int keepalive_intvl; /* time interval between keep alive probes */
int linger2;

unsigned long last_synq_overflow;
};
The tcp header

```c
23 struct tcphdr {
24    __u16    source;
25    __u16    dest;
26    __u32    seq;
27    __u32    ack_seq;
28 #if defined(__LITTLE_ENDIAN_BITFIELD)
29    __u16    res1:4,
30        doff:4,
31        fin:1,
32        syn:1,
33        rst:1,
34        psh:1,
35        ack:1,
36        urg:1,
37        ece:1,
38        cwr:1;
39 #elif defined(__BIG_ENDIAN_BITFIELD)
40    __u16    doff:4,
41        res1:4,
42        cwr:1,
43        ece:1,
44        urg:1,
45        ack:1,
46        psh:1,
47        rst:1,
48        syn:1,
49        fin:1;
50 #else
51 #error "Adjust your <asm/byteorder.h> defines"
52 #endif
53    __u16    window;
54    __u16    check;
55    __u16    urg_ptr;
56    }
```