24

TCP: Transmission Control Protocol

24.1 Introduction

The Transmission Control Protocol, or TCP, provides a connection-oriented, reliable, byte-stream service between the two end points of an application. This is completely different from UDP's connectionless, unreliable, datagram service.

The implementation of UDP presented in Chapter 23 comprised 9 functions and about 800 lines of C code. The TCP implementation we're about to describe comprises 28 functions and almost 4,500 lines of C code. Therefore we divide the presentation of TCP into multiple chapters.

These chapters are not an introduction to TCP. We assume the reader is familiar with the operation of TCP from Chapters 17–24 of Volume 1.

24.2 Code Introduction

The TCP functions appear in six C files and numerous TCP definitions are in seven headers, as shown in Figure 24.1.

Figure 24.2 shows the relationship of the various TCP functions to other kernel functions. The shaded ellipses are the nine main TCP functions that we cover. Eight of these functions appear in the TCP `protosw` structure (Figure 24.8) and the ninth is `tcp_output`.

File	Description
netinet/tcp.h	tcphdr structure definition
netinet/tcp_debug.h	tcp_debug structure definition
netinet/tcp_fsm.h	definitions for TCP's finite state machine
netinet/tcp_seq.h	macros for comparing TCP sequence numbers
netinet/tcp_timer.h	definitions for TCP timers
netinet/tcp_var.h	tcpcb (control block) and tcpstat (statistics) structure definitions
netinet/tcpip.h	TCP plus IP header definition
netinet/tcp_debug.c	support for SO_DEBUG socket debugging (Section 27.10)
netinet/tcp_input.c	tcp_input and ancillary functions (Chapters 28 and 29)
netinet/tcp_output.c	tcp_output and ancillary functions (Chapter 26)
netinet/tcp_subr.c	miscellaneous TCP subroutines (Chapter 27)
netinet/tcp_timer.c	TCP timer handling (Chapter 25)
netinet/tcp_usrreq.c	PRU_*xxx* request handling (Chapter 30)

Figure 24.1 Files discussed in the TCP chapters.

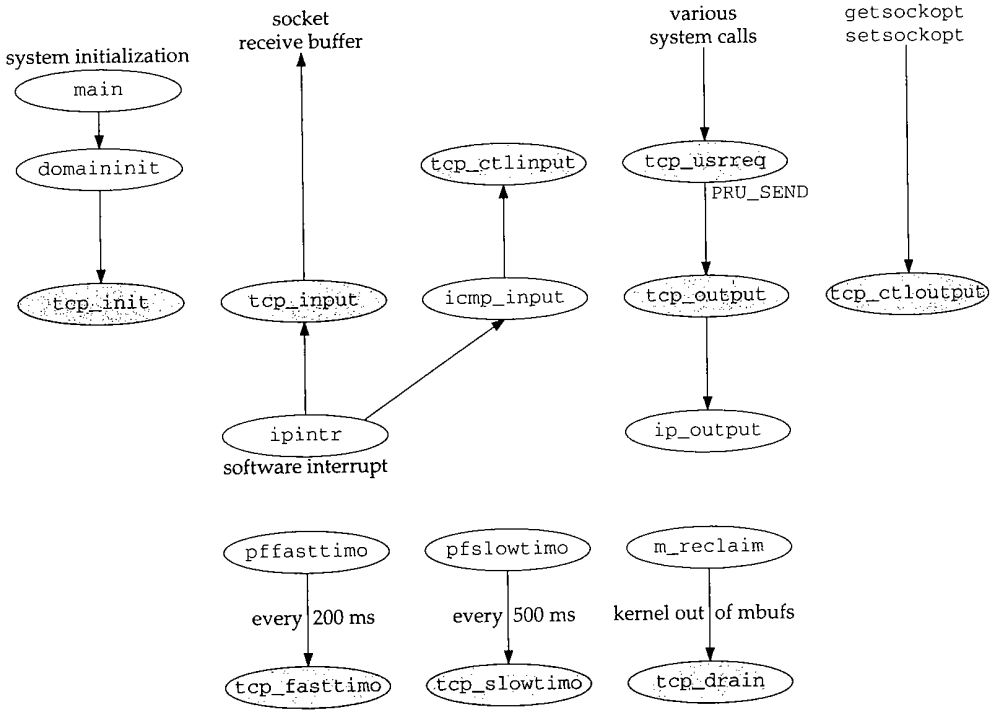

Figure 24.2 Relationship of TCP functions to rest of the kernel.

Global Variables

Figure 24.3 shows the global variables we encounter throughout the TCP functions.

Variable	Datatype	Description
`tcb`	`struct inpcb`	head of the TCP Internet PCB list
`tcp_last_inpcb`	`struct inpcb *`	pointer to PCB for last received segment: one-behind cache
`tcpstat`	`struct tcpstat`	TCP statistics (Figure 24.4)
`tcp_outflags`	`u_char`	array of output flags, indexed by connection state (Figure 24.16)
`tcp_recvspace`	`u_long`	default size of socket receive buffer (8192 bytes)
`tcp_sendspace`	`u_long`	default size of socket send buffer (8192 bytes)
`tcp_iss`	`tcp_seq`	initial send sequence number (ISS)
`tcprexmtthresh`	`int`	number of duplicate ACKs to trigger fast retransmit (3)
`tcp_mssdflt`	`int`	default MSS (512 bytes)
`tcp_rttdflt`	`int`	default RTT if no data (3 seconds)
`tcp_do_rfc1323`	`int`	if true (default), request window scale and timestamp options
`tcp_now`	`u_long`	500 ms counter for RFC 1323 timestamps
`tcp_keepidle`	`int`	keepalive: idle time before first probe (2 hours)
`tcp_keepintvl`	`int`	keepalive: interval between probes when no response (75 sec) (also used as timeout for `connect`)
`tcp_maxidle`	`int`	keepalive: time after probing before giving up (10 min)

Figure 24.3 Global variables introduced in the following chapters.

Statistics

Various TCP statistics are maintained in the global structure `tcpstat`, described in Figure 24.4. We'll see where these counters are incremented as we proceed through the code.

Figure 24.5 shows some sample output of these statistics, from the `netstat -s` command. These statistics were collected after the host had been up for 30 days. Since some counters come in pairs—one counts the number of packets and the other the number of bytes—we abbreviate these in the figure. For example, the two counters for the second line of the table are `tcps_sndpack` and `tcps_sndbyte`.

> The counter for `tcps_sndbyte` should be 3,722,884,824, not –22,194,928 bytes. This is an average of about 405 bytes per segment, which makes sense. Similarly, the counter for `tcps_rcvackbyte` should be 3,738,811,552, not –21,264,360 bytes (for an average of about 565 bytes per segment). These numbers are incorrectly printed as negative numbers because the `printf` calls in the `netstat` program use `%d` (signed decimal) instead of `%lu` (long integer, unsigned decimal). All the counters are unsigned long integers, and these two counters are near the maximum value of an unsigned 32-bit long integer ($2^{32} - 1 = 4,294,967,295$).

tcpstat member	Description	Used by SNMP
tcps_accepts	#SYNs received in LISTEN state	•
tcps_closed	#connections closed (includes drops)	
tcps_connattempt	#connections initiated (calls to connect)	•
tcps_conndrops	#embryonic connections dropped (before SYN received)	•
tcps_connects	#connections established actively or passively	
tcps_delack	#delayed ACKs sent	
tcps_drops	#connections dropped (after SYN received)	•
tcps_keepdrops	#connections dropped in keepalive (established or awaiting SYN)	
tcps_keepprobe	#keepalive probes sent	
tcps_keeptimeo	#times keepalive timer or connection-establishment timer expire	
tcps_pawsdrop	#segments dropped due to PAWS	
tcps_pcbcachemiss	#times PCB cache comparison fails	
tcps_persisttimeo	#times persist timer expires	
tcps_predack	#times header prediction correct for ACKs	
tcps_preddat	#times header prediction correct for data packets	
tcps_rcvackbyte	#bytes ACKed by received ACKs	
tcps_rcvackpack	#received ACK packets	
tcps_rcvacktoomuch	#received ACKs for unsent data	
tcps_rcvafterclose	#packets received after connection closed	
tcps_rcvbadoff	#packets received with invalid header length	•
tcps_rcvbadsum	#packets received with checksum errors	•
tcps_rcvbyte	#bytes received in sequence	
tcps_rcvbyteafterwin	#bytes received beyond advertised window	
tcps_rcvdupack	#duplicate ACKs received	
tcps_rcvdupbyte	#bytes received in completely duplicate packets	
tcps_rcvduppack	#packets received with completely duplicate bytes	
tcps_rcvoobyte	#out-of-order bytes received	
tcps_rcvoopack	#out-of-order packets received	
tcps_rcvpack	#packets received in sequence	
tcps_rcvpackafterwin	#packets with some data beyond advertised window	
tcps_rcvpartdupbyte	#duplicate bytes in part-duplicate packets	
tcps_rcvpartduppack	#packets with some duplicate data	
tcps_rcvshort	#packets received too short	•
tcps_rcvtotal	total #packets received	•
tcps_rcvwinprobe	#window probe packets received	
tcps_rcvwinupd	#received window update packets	
tcps_rexmttimeo	#retransmit timeouts	
tcps_rttupdated	#times RTT estimators updated	
tcps_segstimed	#segments for which TCP tried to measure RTT	
tcps_sndacks	#ACK-only packets sent (data length = 0)	
tcps_sndbyte	#data bytes sent	
tcps_sndctrl	#control (SYN, FIN, RST) packets sent (data length = 0)	
tcps_sndpack	#data packets sent (data length > 0)	
tcps_sndprobe	#window probes sent (1 byte of data forced by persist timer)	
tcps_sndrexmitbyte	#data bytes retransmitted	•
tcps_sndrexmitpack	#data packets retransmitted	•
tcps_sndtotal	total #packets sent	•
tcps_sndurg	#packets sent with URG-only (data length = 0)	
tcps_sndwinup	#window update-only packets sent (data length = 0)	
tcps_timeoutdrop	#connections dropped in retransmission timeout	

Figure 24.4 TCP statistics maintained in the tcpstat structure.

netstat -s output	tcpstat members
10,655,999 packets sent	tcps_sndtotal
9,177,823 data packets (-22,194,928 bytes)	tcps_snd{pack,byte}
257,295 data packets (81,075,086 bytes) retransmitted	tcps_sndrexmit{pack,byte}
862,900 ack-only packets (531,285 delayed)	tcps_sndacks,tcps_delack
229 URG-only packets	tcps_sndurg
3,453 window probe packets	tcps_sndprobe
74,925 window update packets	tcps_sndwinup
279,387 control packets	tcps_sndctrl
8,801,953 packets received	tcps_rcvtotal
6,617,079 acks (for -21,264,360 bytes)	tcps_rcvack{pack,byte}
235,311 duplicate acks	tcps_rcvdupack
0 acks for unsent data	tcps_rcvacktoomuch
4,670,615 packets (324,965,351 bytes) rcvd in-sequence	tcps_rcv{pack,byte}
46,953 completely duplicate packets (1,549,785 bytes)	tcps_rcvdup{pack,byte}
22 old duplicate packets	tcps_pawsdrop
3,442 packets with some dup. data (54,483 bytes duped)	tcps_rcvpartdup{pack,byte}
77,114 out-of-order packets (13,938,456 bytes)	tcps_rcvoo{pack,byte}
1,892 packets (1,755 bytes) of data after window	tcps_rcv{pack,byte}afterwin
1,755 window probes	tcps_rcvwinprobe
175,476 window update packets	tcps_rcvwindup
1,017 packets received after close	tcps_rcvafterclose
60,370 discarded for bad checksums	tcps_rcvbadsum
279 discarded for bad header offset fields	tcps_rcvbadoff
0 discarded because packet too short	tcps_rcvshort
144,020 connection requests	tcps_connattempt
92,595 connection accepts	tcps_accepts
126,820 connections established (including accepts)	tcps_connects
237,743 connections closed (including 1,061 drops)	tcps_closed,tcps_drops
110,016 embryonic connections dropped	tcps_conndrops
6,363,546 segments updated rtt (of 6,444,667 attempts)	tcps_{rttupdated,segstimed}
114,797 retransmit timeouts	tcps_rexmttimeo
86 connection dropped by rexmit timeout	tcps_timeoutdrop
1,173 persist timeouts	tcps_persisttimeo
16,419 keepalive timeouts	tcps_keeptimeo
6,899 keepalive probes sent	tcps_keepprobe
3,219 connections dropped by keepalive	tcps_keepdrops
733,130 correct ACK header predictions	tcps_predack
1,266,889 correct data packet header predictions	tcps_preddat
1,851,557 cache misses	tcps_pcbcachemiss

Figure 24.5 Sample TCP statistics.

SNMP Variables

Figure 24.6 shows the 14 simple SNMP variables in the TCP group and the counters from the `tcpstat` structure implementing that variable. The constant values shown for the first four entries are fixed by the Net/3 implementation. The counter `tcpCurrEstab` is computed as the number of Internet PCBs on the TCP PCB list.

Figure 24.7 shows `tcpTable`, the TCP listener table.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.