File created: 17-Aug-90 12:01:50 **{DSK}<LISPFILES>ETHERNET>TCP>NEW>TCP.;5**

changes to: (FILES TCPLLIP)
(FNS \TCP.DELETE.TCB)

previous date: 13-Feb-89 21:04:17 {DSK}<LISPFILES>ETHERNET>TCP>NEW>TCP.;3

Read Table: INTERLISP

Package: INTERLISP

Format: XCCS

;;


```
(RPAQQ TCPCOMS
       [(COMS ;; Transmission Control Protocol.  RFC 793, September 1981
              )
        (COMS (DECLARE%: EVAL@LOAD (FILES (SYSLOAD)
                                          TCPLLIP))
              (GLOBALVARS \TCP.LOCK \TCP.CONTROL.BLOCKS \TCP.CHECKSUMS.ON \TCP.PSEUDOHEADER \TCP.MSL
                     \TCP.DEFAULT.USER.TIMEOUT \TCP.DEFAULT.RECEIVE.WINDOW \TCP.DEVICE \TCP.MASTER.SOCKET))
        (COMS ;; DoD Internet addresses
              (FNS SET.IP.ADDRESS STRING.TO.IP.ADDRESS IP.ADDRESS.TO.STRING \LOCAL.IP.ADDRESS))
        [COMS ;; TCP segments
              (DECLARE%: EVAL@COMPILE DONTCOPY
                     ;; control bits for TCP.CTRL field of TCP header
                     (EXPORT (CONSTANTS \TCP.CTRL.ACK \TCP.CTRL.FIN \TCP.CTRL.PSH \TCP.CTRL.RST \TCP.CTRL.SYN
                                    \TCP.CTRL.URG)
                            ;; option definitions
                            (CONSTANTS \TCPOPT.END \TCPOPT.NOP \TCPOPT.MAXSEG)
                            ;; TCP protocol number for IP level dispatch
                            (CONSTANTS \TCP.PROTOCOL)
                            ;; TCP header length in bytes (= 4 * min data offset)
                            (CONSTANTS \TCP.HEADER.LENGTH)
                            ;; minimum offset of data from segment in 32-bit words (= header length / 4)
                            (CONSTANTS \TCP.MIN.DATA.OFFSET)
                            ;; default maximum segment size
                            (CONSTANTS \TCP.DEFAULT.MAXSEG)
                            ;; TCP segment
                            (RECORDS TCPSEGMENT]
        (COMS ;; TCP sequence numbers
              (DECLARE%: EVAL@COMPILE DONTCOPY
                     ;; macros for comparing TCP sequence numbers
                     (MACROS \32BIT.EQ \32BIT.LT \32BIT.LEQ \32BIT.GT \32BIT.GEQ)
                     ;; fast multiply by 3 -- evaluates its argument twice
                     (MACROS \3TIMES))
              (FNS \TCP.SELECT.ISS))
        (COMS ;; TCP control blocks
              (DECLARE%: EVAL@COMPILE DONTCOPY
                     ;; TCP control block
                     (EXPORT (RECORDS TCP.CONTROL.BLOCK TCPSTREAM))
                     ;; TCP stream
                     )
              (INITRECORDS TCP.CONTROL.BLOCK TCPSTREAM)
              ;; global lock for TCP-related mutual exclusion
              (INITVARS (\TCP.LOCK (CREATE.MONITORLOCK)))
              ;; list of TCP control blocks for connection lookup
              (INITVARS (\TCP.CONTROL.BLOCKS NIL))
              (FNS \TCP.CREATE.TCB \TCP.SELECT.PORT \TCP.LOOKUP.TCB \TCP.DELETE.TCB \TCP.NOSOCKETFN
                   \TCP.PORTCOMPARE))
        (COMS ;; TCP checksums
              (DECLARE%: EVAL@COMPILE DONTCOPY
                     ;; pseudo-header for checksum calculation
                     (RECORDS TCP.PSEUDOHEADER)
                     (CONSTANTS \TCP.PSEUDOHEADER.LENGTH)
```

```
                (MACROS \16BIT.COMPLEMENT \16BIT.1C.PLUS))
        (INITRECORDS TCP.PSEUDOHEADER)
        (INITVARS (\TCP.PSEUDOHEADER NIL))
        ;; this variable controls whether checksums are performed on incoming segments
        (INITVARS (\TCP.CHECKSUMS.ON NIL))
        ;; checksum routines
        (FNS \COMPUTE.CHECKSUM \TCP.CHECKSUM.INCOMING \TCP.CHECKSUM.OUTGOING))
(COMS   (DECLARE%: EVAL@COMPILE DONTCOPY
                ;; constants for retransmission timeout calculation
                ;; initial retransmission timeout
                (CONSTANTS \TCP.INITIAL.RTO)
                ;; upper and lower bounds on retransmission timeout
                (CONSTANTS (\TCP.UBOUND 5000)
                        (\TCP.LBOUND 1000)))
        ;; maximum segment lifetime
        (INITVARS (\TCP.MSL 5000))
        (INITVARS (\TCP.DEFAULT.USER.TIMEOUT 60000)
                (\TCP.DEFAULT.RECEIVE.WINDOW 4096)
                (\TCP.DEVICE NIL))
        ;; TCP protocol routines
        (FNS \TCP.ACK# \TCP.PACKET.FILTER \TCP.SETUP.SEGMENT \TCP.RELEASE.SEGMENT \TCP.CONNECTION
             \TCP.FIX.INCOMING.SEGMENT \TCP.DATA.LENGTH \TCP.SYN.OR.FIN \TCP.INPUT \TCP.INPUT.INITIAL
             \TCP.INPUT.UNSYNC \TCP.INPUT.LISTEN \TCP.INPUT.SYN.SENT \TCP.CHECK.WINDOW \TCP.CHECK.RESET
             \TCP.CHECK.SECURITY \TCP.CHECK.NO.SYN \TCP.CHECK.ACK \TCP.HANDLE.ACK \TCP.HANDLE.URG
             \TCP.QUEUE.INPUT \TCP.HANDLE.FIN \TCP.OUR.FIN.IS.ACKED \TCP.SIGNAL.URGENT.DATA \TCP.PROCESS
             \TCP.TEMPLATE \TCP.SETUP.SEGMENT.OPTIONS \TCP.SEND.CONTROL \TCP.SEND.ACK \TCP.SEND.RESET
             \TCP.FIX.OUTGOING.SEGMENT \TCP.SEND.DATA \TCP.SEND.SEGMENT \TCP.NEW.TEMPLATE
             \TCP.START.PROBE.TIMER \TCP.RETRANSMIT \TCP.START.TIME.WAIT \TCP.CONNECTION.DROPPED
             \TCP.CHECK.OPTIONS \TCP.PROCESS.OPTIONS))
(COMS   ;; support for ICMP messages that affect TCP connections
        (DECLARE%: EVAL@COMPILE DONTCOPY
                ;; ICMP protocol number for IP level dispatch
                (CONSTANTS \ICMP.PROTOCOL)
                ;; number of 32 bit words in ICMP message before start of original datagram
                (CONSTANTS \ICMP.32BIT.WORDS)
                ;; relevant ICMP message types
                (CONSTANTS \ICMP.DESTINATION.UNREACHABLE \ICMP.SOURCE.QUENCH))
        (FNS \TCP.HANDLE.ICMP))
(COMS   ;; TCP stream routines
        (FNS TCP.OPEN TCP.OTHER.STREAM \TCP.BOUTS \TCP.OTHER.BIN \TCP.OTHER.BOUT \TCP.BIN \TCP.BACKFILEPTR
             \TCP.GETNEXTBUFFER \TCP.GET.SEGMENT \TCP.PEEKBIN \TCP.GETFILEPTR \TCP.READP \TCP.EOFP
             TCP.URGENTP TCP.URGENT.EVENT \TCP.BOUT \TCP.FLUSH \TCP.FORCEOUTPUT TCP.URGENT.MARK
             \TCP.FILL.IN.SEGMENT \TCP.CLOSE \TCP.RESETCLOSE TCP.CLOSE.SENDER TCP.DESTADDRESS TCP.STOP))
(COMS   ;; well-known ports for network standard functions
        (CONSTANTS * \TCP.ASSIGNED.PORTS))
(COMS   ;; Stub for debugging
        (INITVARS (\TCP.DEBUGGABLE)
                (TCPTRACEFLG))
        (GLOBALVARS \TCP.DEBUGGABLE TCPTRACEFLG)
        (FNS PPTCB \TCP.TRACE.SEGMENT \TCP.TRACE.TRANSITION))
(COMS   ;; TCP initialization
        (FNS \TCP.INIT)
        (P (\TCP.INIT])

;; Transmission Control Protocol.  RFC 793, September 1981

(DECLARE%: EVAL@LOAD

(FILESLOAD (SYSLOAD)
       TCPLLIP)
)

(DECLARE%: DOEVAL@COMPILE DONTCOPY

(GLOBALVARS \TCP.LOCK \TCP.CONTROL.BLOCKS \TCP.CHECKSUMS.ON \TCP.PSEUDOHEADER \TCP.MSL \TCP.DEFAULT.USER.TIMEOUT
       \TCP.DEFAULT.RECEIVE.WINDOW \TCP.DEVICE \TCP.MASTER.SOCKET)
)

;; DoD Internet addresses

(DEFINEQ
```

```
(SET.IP.ADDRESS
  [LAMBDA NIL                                                    (* ejs%: "28-Dec-84 18:45")
                                                                 (* set local IP address manually)
    (PROG [(ADDR (\IP.READ.STRING.ADDRESS (PROMPTFORWORD "Enter IP address:" (\IP.ADDRESS.TO.STRING
                                                                                (OR (CAR \IP.LOCAL.ADDRESSES)
                                                                                    0]
          (SETQ \IP.LOCAL.ADDRESSES (LIST ADDR])

(STRING.TO.IP.ADDRESS
  [LAMBDA (STR)                                                  (* ecc "14-May-84 15:01")
    (APPLY (FUNCTION IP\Make\Address)
           (to 4 bind (I _ 0)
                    OFFSET
              collect (SETQ OFFSET (ADD1 I))
                      (MKATOM (SUBSTRING STR OFFSET (AND (SETQ I (STRPOS "." STR OFFSET))
                                                         (SUB1 I])

(IP.ADDRESS.TO.STRING
  [LAMBDA (IPADDR)                                               (* ecc "14-May-84 14:32")
    (PROG ((A (LOADBYTE IPADDR 24 8))
           (B (LOADBYTE IPADDR 16 8))
           (C (LOADBYTE IPADDR 8 8))
           (D (LOADBYTE IPADDR 0 8)))
          (RETURN (CONCAT A "." B "." C "." D])

(\LOCAL.IP.ADDRESS
  [LAMBDA NIL                                                    (* ejs%: "28-Dec-84 18:45")
                                                                 (* return our IP address (or the first if we're multi-homed))
    (if (NULL \IP.LOCAL.ADDRESSES)
        then (ERROR "You must set \IP.LOCAL.ADDRESSES to a list of our local IP addresses"))
    (CAR \IP.LOCAL.ADDRESSES])

)

;; TCP segments

(DECLARE%: EVAL@COMPILE DONTCOPY

;; FOLLOWING DEFINITIONS EXPORTED

(DECLARE%: EVAL@COMPILE

(RPAQQ \TCP.CTRL.ACK 16)

(RPAQQ \TCP.CTRL.FIN 1)

(RPAQQ \TCP.CTRL.PSH 8)

(RPAQQ \TCP.CTRL.RST 4)

(RPAQQ \TCP.CTRL.SYN 2)

(RPAQQ \TCP.CTRL.URG 32)

(CONSTANTS \TCP.CTRL.ACK \TCP.CTRL.FIN \TCP.CTRL.PSH \TCP.CTRL.RST \TCP.CTRL.SYN \TCP.CTRL.URG)
)

(DECLARE%: EVAL@COMPILE

(RPAQQ \TCPOPT.END 0)

(RPAQQ \TCPOPT.NOP 1)

(RPAQQ \TCPOPT.MAXSEG 2)

(CONSTANTS \TCPOPT.END \TCPOPT.NOP \TCPOPT.MAXSEG)
)

(DECLARE%: EVAL@COMPILE

(RPAQQ \TCP.PROTOCOL 6)

(CONSTANTS \TCP.PROTOCOL)
)

(DECLARE%: EVAL@COMPILE

(RPAQQ \TCP.HEADER.LENGTH 20)

(CONSTANTS \TCP.HEADER.LENGTH)
)

(DECLARE%: EVAL@COMPILE
```

```
(RPAQQ \TCP.MIN.DATA.OFFSET 5)

(CONSTANTS \TCP.MIN.DATA.OFFSET)
)

(DECLARE%: EVAL@COMPILE

(RPAQQ \TCP.DEFAULT.MAXSEG 536)

(CONSTANTS \TCP.DEFAULT.MAXSEG)
)

(DECLARE%: EVAL@COMPILE

[ACCESSFNS TCPSEGMENT ((TCPHEADER (\IPDATABASE DATUM)))
       (BLOCKRECORD TCPHEADER ((TCP.SRC.PORT WORD)
                               (TCP.DST.PORT WORD)
                               (TCP.SEQ FIXP)
                               (TCP.ACK FIXP)
                               (TCP.DATA.OFFSET BITS 4)
                               (TCP.MBZ BITS 6)
                               (TCP.CTRL BITS 6)
                               (TCP.WINDOW WORD)
                               (TCP.CHECKSUM WORD)
                               (TCP.URG.PTR WORD)))
       (ACCESSFNS TCPSEGMENT ((TCP.DATA.LENGTH (fetch (IP IPHEADERCHECKSUM) of DATUM)
                                      (replace (IP IPHEADERCHECKSUM) of DATUM with NEWVALUE))
                              (TCP.SRC.ADDR (fetch (IP IPSOURCEADDRESS) of DATUM)
                                      (replace (IP IPSOURCEADDRESS) of DATUM with NEWVALUE))
                              (TCP.DST.ADDR (fetch (IP IPDESTINATIONADDRESS) of DATUM)
                                      (replace (IP IPDESTINATIONADDRESS) of DATUM with NEWVALUE))
                              (TCP.HEADER.LENGTH (LLSH (fetch TCP.DATA.OFFSET of DATUM)
                                                       2))
                              (TCP.CONTENTS (\ADDBASE (fetch TCPHEADER of DATUM)
                                                      (UNFOLD (fetch TCP.DATA.OFFSET of DATUM)
                                                              WORDSPERCELL)))
                              (TCP.OPTIONS (\ADDBASE (fetch TCPHEADER of DATUM)
                                                     (UNFOLD \TCP.MIN.DATA.OFFSET WORDSPERCELL]
)
)
```

;; END EXPORTED DEFINITIONS

;; TCP sequence numbers

```
(DECLARE%: EVAL@COMPILE DONTCOPY

(DECLARE%: EVAL@COMPILE

(PUTPROPS \32BIT.EQ MACRO ((A B)
                           (IEQP A B)))

(PUTPROPS \32BIT.LT MACRO ((A B)
                           (ILESSP (IDIFFERENCE A B)
                                   0)))

(PUTPROPS \32BIT.LEQ MACRO ((A B)
                            (ILEQ (IDIFFERENCE A B)
                                  0)))

(PUTPROPS \32BIT.GT MACRO ((A B)
                           (IGREATERP (IDIFFERENCE A B)
                                      0)))

(PUTPROPS \32BIT.GEQ MACRO ((A B)
                            (IGEQ (IDIFFERENCE A B)
                                  0)))
)

(DECLARE%: EVAL@COMPILE

(PUTPROPS \3TIMES MACRO ((N)
                         (IPLUS (LLSH N 1)
                                N)))
)
)

(DEFINEQ

(\TCP.SELECT.ISS
  [LAMBDA NIL                                                    (* ecc "16-May-84 11:40")

            (* select an initial send sequence number -- use the time of day to make sure we won't repeat after a crash)

    (LOGAND (DAYTIME)
            65535])
```

```
)

;; TCP control blocks

(DECLARE%: EVAL@COMPILE DONTCOPY

;; FOLLOWING DEFINITIONS EXPORTED

(DECLARE%: EVAL@COMPILE

(DATATYPE TCP.CONTROL.BLOCK ((TCB.LOCK POINTER)                    (* monitor lock for synchronizing access)
                             (TCB.STATE POINTER)                   (* one of CLOSED LISTEN SYN.SENT SYN.RECEIVED
                                                                   ESTABLISHED FIN.WAIT.1 FIN.WAIT.2 CLOSE.WAIT
                                                                   CLOSING LAST.ACK TIME.WAIT)
                             (TCB.SND.STREAM POINTER)              (* user's send stream)
                             (TCB.SND.SEGMENT POINTER)             (* current output packet being filled)
                             (TCB.RCV.STREAM POINTER)              (* user's receive stream)
                             (TCB.RCV.SEGMENT POINTER)             (* current input packet being read)
                             (TCB.2MSL.TIMER POINTER)              (* 2*MSL quiet time)
                             (TCB.MAXSEG POINTER)                  (* maximum segment size)
                             (TCB.CLOSEDFLG POINTER)               (* T if user has initiated close (no more data to send))
                             (TCB.FINSEQ POINTER)                  (* one past the sequence number of the FIN we sent)
                             (TCB.ACKFLG POINTER)                  (* when to ACK peer%: NOW or LATER)
                             (TCB.TEMPLATE POINTER)                (* TCP header template)
                             (TCB.PH POINTER)                      (* TCP pseudo-header for checksumming)
                             (TCB.SRC.PORT WORD)                   (* local port)
                             (TCB.DST.PORT WORD)                   (* remote port)
                             (TCB.DST.HOST FIXP)                   (* remote host address)
                             (TCB.INPUT.QUEUE POINTER)             (* queue of received segments to be read)
                             (TCB.REXMT.QUEUE POINTER)             (* queue of unacked segments to be retransmitted)
                             (TCB.SND.UNA FIXP)                    (* first unacknowledged sequence number)
                             (TCB.SND.NXT FIXP)                    (* next sequence number to be sent)
                             (TCB.SND.UP FIXP)                     (* send urgent pointer)
                             (TCB.SND.WL1 FIXP)                    (* segment sequence number used for last window update)
                             (TCB.SND.WL2 FIXP)                    (* segment acknowledgment number used for last window
                                                                   update)
                             (TCB.ISS FIXP)                        (* initial send sequence number)
                             (TCB.SND.WND WORD)                    (* send window)
                             (TCB.RCV.WND WORD)                    (* receive window)
                             (TCB.RCV.NXT FIXP)                    (* next sequence number expected)
                             (TCB.RCV.UP FIXP)                     (* receive urgent pointer)
                             (TCB.IRS FIXP)                        (* initial receive sequence number)
                             (TCB.USER.TIMEOUT POINTER)            (* in milliseconds)
                             (TCB.ESTABLISHED POINTER)             (* processes waiting for this event are notified when the
                                                                   connection becomes established)
                             (TCB.SND.EVENT POINTER)               (* processes waiting for this event are notified when the send
                                                                   window opens up)
                             (TCB.RCV.EVENT POINTER)               (* processes waiting for this event are notified when data is
                                                                   received)
                             (TCB.URGENT.EVENT POINTER)            (* processes waiting for this event are notified when urgent data
                                                                   is received)
                             (TCB.FINACKED.EVENT POINTER)          (* processes waiting for this event are notified when our FIN has
                                                                   been acked)
                             (TCB.MODE POINTER)                    (* ACTIVE or PASSIVE)
                             (TCB.RTFLG POINTER)                   (* T if round trip time being measured)
                             (TCB.RTSEQ POINTER)                   (* sequence number being timed)
                             (TCB.RTTIMER POINTER)                 (* round trip timer)
                             (TCB.SRTT POINTER)                    (* smoothed round trip time)
                             (TCB.RTO POINTER)                     (* retransmission timeout based on smoothed round trip time)
                             (TCB.PROBE.TIMER POINTER)             (* timer for delayed ACKs and window probes)
                             (TCB.IPSOCKET POINTER)                (* Pointer to open IP socket for this connection)
                             (TCB.PROCESS POINTER)                 (* TCP monitor process for this connection)
                             (TCB.SENT.ZERO FLAG)                  (* Sent a zero allocation last time)
                             (TCB.OUTPUT.HELD FLAG)                (* True if output window shut)
                             (TCB.NO.IDLE.PROBING FLAG)            (* True if we don't probe when nothing to output)
                             (NIL BITS 5)
                             (TCB.OUR.MAXSEG WORD)
                             (TCB.LAST.SENT.RCV.WND WORD)          (* The value of the last rcv window we sent)
                             )
      TCB.LOCK _ (CREATE.MONITORLOCK)
      TCB.STATE _ 'CLOSED TCB.RCV.WND _ \TCP.DEFAULT.RECEIVE.WINDOW TCB.USER.TIMEOUT _
      \TCP.DEFAULT.USER.TIMEOUT TCB.ESTABLISHED _ (CREATE.EVENT)
      TCB.SND.EVENT _ (CREATE.EVENT)
      TCB.RCV.EVENT _ (CREATE.EVENT)
      TCB.URGENT.EVENT _ (CREATE.EVENT)
      TCB.FINACKED.EVENT _ (CREATE.EVENT)
      TCB.MAXSEG _ \TCP.DEFAULT.MAXSEG TCB.OUR.MAXSEG _ \TCP.DEFAULT.MAXSEG TCB.SRTT _ \TCP.INITIAL.RTO TCB.RTO
      _ \TCP.INITIAL.RTO)

(ACCESSFNS TCPSTREAM ((TCB (fetch (STREAM F1) of DATUM)
                           (replace (STREAM F1) of DATUM with NEWVALUE))
                      (BYTECOUNT (fetch (STREAM F2) of DATUM)
                           (replace (STREAM F2) of DATUM with NEWVALUE))
                      (ACCESS (fetch (STREAM ACCESS) of DATUM)
                           (replace (STREAM ACCESS) of DATUM with NEWVALUE))
```

```
                    (ORIGINAL.COFFSET (fetch (STREAM FW6) of DATUM)
                                      (replace (STREAM FW6) of DATUM with NEWVALUE)))
       (CREATE (create STREAM
                       DEVICE _ \TCP.DEVICE)))
)

(/DECLAREDATATYPE 'TCP.CONTROL.BLOCK
       '(POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER
              WORD WORD FIXP POINTER POINTER FIXP FIXP FIXP FIXP FIXP FIXP WORD WORD FIXP FIXP FIXP POINTER
              POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER
              POINTER POINTER FLAG FLAG FLAG (BITS 5)
              WORD WORD)
       ;; ---field descriptor list elided by lister---
       '86)
)

;; END EXPORTED DEFINITIONS

(/DECLAREDATATYPE 'TCP.CONTROL.BLOCK
       '(POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER
              WORD WORD FIXP POINTER POINTER FIXP FIXP FIXP FIXP FIXP FIXP WORD WORD FIXP FIXP FIXP POINTER
              POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER POINTER
              POINTER POINTER FLAG FLAG FLAG (BITS 5)
              WORD WORD)
       ;; ---field descriptor list elided by lister---
       '86)

;; global lock for TCP-related mutual exclusion

(RPAQ? \TCP.LOCK (CREATE.MONITORLOCK))

;; list of TCP control blocks for connection lookup

(RPAQ? \TCP.CONTROL.BLOCKS NIL)

(DEFINEQ

(\TCP.CREATE.TCB
  [LAMBDA (DST.HOST DST.PORT SRC.PORT MODE OUR.MAXSEG)          (* ejs%: "27-May-86 14:39")
                                                                (* create a new TCB and the input and output streams that go with it)
    (WITH.FAST.MONITOR \TCP.LOCK (PROG [(TCB (create TCP.CONTROL.BLOCK
                                                     TCB.DST.HOST _ DST.HOST
                                                     TCB.DST.PORT _ DST.PORT
                                                     TCB.SRC.PORT _ (if (ZEROP SRC.PORT)
                                                                        then (\TCP.SELECT.PORT)
                                                                      else SRC.PORT)
                                                     TCB.INPUT.QUEUE _ (create SYSQUEUE)
                                                     TCB.REXMT.QUEUE _ (create SYSQUEUE)
                                                     TCB.MODE _ MODE
                                                     TCB.OUR.MAXSEG _ (OR OUR.MAXSEG \TCP.DEFAULT.MAXSEG]
                                       (replace (STREAM STRMBOUTFN)
                                          of (replace TCB.RCV.STREAM of TCB
                                                with (create TCPSTREAM
                                                             ACCESS _ 'INPUT
                                                             TCB _ TCB
                                                             BYTECOUNT _ 0))
                                          with (FUNCTION \TCP.OTHER.BOUT))
                                       (replace (STREAM STRMBINFN)
                                          of (replace TCB.SND.STREAM of TCB
                                                with (create TCPSTREAM
                                                             ACCESS _ 'APPEND
                                                             TCB _ TCB
                                                             BYTECOUNT _ 0))
                                          with (FUNCTION \TCP.OTHER.BIN))
                                       (\TCP.START.PROBE.TIMER TCB)
                                       (push \TCP.CONTROL.BLOCKS TCB)

          (* put it on the global list of TCBs so it can be found by \TCP.LOOKUP.TCB)

                                       (replace TCB.IPSOCKET of TCB with (\IP.OPEN.SOCKET \TCP.PROTOCOL TCB))
                                                                (* Tell IP about it)
                                       (RETURN TCB])

(\TCP.SELECT.PORT
  [LAMBDA NIL                                                   (* ecc " 7-May-84 17:23")
                                                                (* find a port unique among all TCP connections on this host)
    (PROG ((PORT (LOGAND (DAYTIME)
                         65535)))
          (until (for TCB in \TCP.CONTROL.BLOCKS always (NEQ PORT (fetch TCB.SRC.PORT of TCB)))
             do (add PORT 1))
          (RETURN PORT])
```

(\TCP.LOOKUP.TCB

```
[LAMBDA (DST.HOST DST.PORT SRC.PORT NOWILDCARDFLG)          (* ejs%: "21-Mar-86 18:40")

        (* Find a TCB that matches the specified addresses. If NOWILDCARDFLG is non-NIL we match against a partially specified
        TCB if no fully specified one was found.)

   (WITH.FAST.MONITOR \TCP.LOCK
          (bind WILDCARD for TCB in \TCP.CONTROL.BLOCKS
             do (if (EQ SRC.PORT (fetch TCB.SRC.PORT of TCB))
                    then                                  (* only check further if the local ports match)
                         (if (AND (IEQP DST.HOST (fetch TCB.DST.HOST of TCB))
                                  (EQ DST.PORT (fetch TCB.DST.PORT of TCB)))
                             then                         (* a full match)
                                  (RETURN TCB)
                           elseif [AND (NOT NOWILDCARDFLG)
                                       (NULL WILDCARD)
                                       (OR (ZEROP (fetch TCB.DST.HOST of TCB))
                                           (IEQP DST.HOST (fetch TCB.DST.HOST of TCB)))
                                       (OR (ZEROP (fetch TCB.DST.PORT of TCB))
                                           (EQ DST.PORT (fetch TCB.DST.PORT of TCB]
                             then                         (* a wildcard match)
                                  (SETQ WILDCARD TCB)))
             finally (RETURN (if NOWILDCARDFLG
                                 then NIL
                                 else WILDCARD])
```

(\TCP.DELETE.TCB

```
[LAMBDA (TCB)                                             ; Edited 25-Aug-88 18:39 by bvm
   (WITH.FAST.MONITOR \TCP.LOCK (\TCP.TRACE.TRANSITION TCB 'CLOSED)
          (replace TCB.STATE of TCB with 'CLOSED)
          (\FLUSH.PACKET.QUEUE (fetch TCB.INPUT.QUEUE of TCB))
          (\FLUSH.PACKET.QUEUE (fetch TCB.REXMT.QUEUE of TCB))
          (SETQ \TCP.CONTROL.BLOCKS (DREMOVE TCB \TCP.CONTROL.BLOCKS))
          (\IP.CLOSE.SOCKET (fetch TCB.IPSOCKET of TCB)
                 \TCP.PROTOCOL T)
          (replace TCB.IPSOCKET of TCB with NIL)
          [LET [(WHENCLOSEDFN (PROCESSPROP (THIS.PROCESS)
                                           'WHENCLOSEDFN]
               (COND
                  (WHENCLOSEDFN (CL:FUNCALL WHENCLOSEDFN (fetch TCB.RCV.STREAM of TCB)
                                             (fetch TCB.SND.STREAM of TCB]
                                                          ; break circular links
          (replace TCB.SND.STREAM of TCB with NIL)
          (replace TCB.RCV.STREAM of TCB with NIL)        ; wake up anyone waiting for events to occur
          (NOTIFY.EVENT (fetch TCB.ESTABLISHED of TCB))
          (NOTIFY.EVENT (fetch TCB.SND.EVENT of TCB))
          (NOTIFY.EVENT (fetch TCB.RCV.EVENT of TCB))
          (NOTIFY.EVENT (fetch TCB.URGENT.EVENT of TCB))
          (NOTIFY.EVENT (fetch TCB.FINACKED.EVENT of TCB])
```

(\TCP.NOSOCKETFN

```
[LAMBDA (IP)                                              (* ejs%: " 1-Feb-86 18:12")

        (* * Called when no TCP port corresponding to IP packet is found.
        We try again, allowing for wildcards)

   (LET* ((PROTOCOLCHAIN (\IP.FIND.PROTOCOL \TCP.PROTOCOL \IP.PROTOCOLS))
          (IPSOCKET (fetch (IPSOCKET IPSLINK) of PROTOCOLCHAIN)))
          (while IPSOCKET do [COND
                                ((\TCP.PORTCOMPARE IP IPSOCKET T)
                                 (APPLY* (ffetch (IPSOCKET IPSINPUTFN) of IPSOCKET)
                                         IP IPSOCKET)
                                 (RETURN))
                                (T (SETQ IPSOCKET (fetch (IPSOCKET IPSLINK) of IPSOCKET]
            finally (COND
                       [(NOT (BITTEST (fetch TCP.CTRL of IP)
                                      \TCP.CTRL.RST))
                        (COND
                           ((BITTEST (fetch TCP.CTRL of IP)
                                     \TCP.CTRL.ACK)
                            (\TCP.SEND.RESET IP (fetch TCP.ACK of IP)
                                   0 \TCP.CTRL.RST))
                           (T (\TCP.SEND.RESET IP 0 (IPLUS (fetch TCP.SEQ of IP)
                                                           (fetch TCP.DATA.LENGTH of IP))
                                      (LOGOR \TCP.CTRL.ACK \TCP.CTRL.RST]
                       (T (\RELEASE.ETHERPACKET IP])
```

(\TCP.PORTCOMPARE

```
[LAMBDA (SEGMENT IPSOCKET WILDCARDFLG)                    (* ejs%: "13-Apr-85 17:44")

        (* Find a TCB that matches the specified addresses. If NOWILDCARDFLG is non-NIL we match against a partially specified
        TCB if no fully specified one was found.)
```

```
    (WITH.FAST.MONITOR \TCP.LOCK (PROG ((DST.HOST (fetch (TCPSEGMENT TCP.SRC.ADDR) of SEGMENT))
                                        (DST.PORT (fetch (TCPSEGMENT TCP.SRC.PORT) of SEGMENT))
                                        (SRC.PORT (fetch (TCPSEGMENT TCP.DST.PORT) of SEGMENT))
                                        (TCB (fetch (IPSOCKET IPSOCKET) of IPSOCKET)))
                                       (COND
                                          ((AND TCB (EQ SRC.PORT (fetch TCB.SRC.PORT of TCB)))
                                                                  (* only check further if the local ports match)
                                           (COND
                                              ((AND (IEQP DST.HOST (fetch TCB.DST.HOST of TCB))
                                                    (EQ DST.PORT (fetch TCB.DST.PORT of TCB)))
                                                                  (* a full match)
                                               (RETURN IPSOCKET))
                                              ([AND WILDCARDFLG (OR (ZEROP (fetch TCB.DST.HOST of TCB))
                                                                    (IEQP DST.HOST (fetch TCB.DST.HOST
                                                                                      of TCB)))
                                                    (OR (ZEROP (fetch TCB.DST.PORT of TCB))
                                                        (EQ DST.PORT (fetch TCB.DST.PORT of TCB]
                                                                  (* a wildcard match)
                                               (RETURN IPSOCKET])

)
```

;; TCP checksums

```
(DECLARE%: EVAL@COMPILE DONTCOPY

(DECLARE%: EVAL@COMPILE

(DATATYPE TCP.PSEUDOHEADER ((PH.SRC.ADDR FIXP)
                            (PH.DST.ADDR FIXP)
                            (NIL BYTE)
                            (PH.PROTOCOL BYTE)
                            (PH.LENGTH WORD))
       PH.PROTOCOL _ \TCP.PROTOCOL)
)

(/DECLAREDATATYPE 'TCP.PSEUDOHEADER '(FIXP FIXP BYTE BYTE WORD)
       ;; ---field descriptor list elided by lister---
       '6)

(DECLARE%: EVAL@COMPILE

(RPAQQ \TCP.PSEUDOHEADER.LENGTH 12)

(CONSTANTS \TCP.PSEUDOHEADER.LENGTH)
)

(DECLARE%: EVAL@COMPILE

(PUTPROPS \16BIT.COMPLEMENT MACRO ((X)
                                   (LOGXOR X (MASK.1'S 0 16))))

(PUTPROPS \16BIT.1C.PLUS MACRO [(X Y)

          (* compute the one's complement sum of X and Y without creating FIXP boxes --
          the sum modulo |2^16| plus an end-around carry)

                                (PROG ((DELTA (IDIFFERENCE MAX.SMALLP Y)))
                                      (RETURN (if (ILEQ X DELTA)
                                                  then (IPLUS X Y)
                                                  else (IDIFFERENCE X DELTA])
)
)

(/DECLAREDATATYPE 'TCP.PSEUDOHEADER '(FIXP FIXP BYTE BYTE WORD)
       ;; ---field descriptor list elided by lister---
       '6)

(RPAQ? \TCP.PSEUDOHEADER NIL)
```

;; this variable controls whether checksums are performed on incoming segments

```
(RPAQ? \TCP.CHECKSUMS.ON NIL)
```

;; checksum routines

```
(DEFINEQ
```

(\COMPUTE.CHECKSUM

```
  [LAMBDA (BASE LENGTH DONTCOMPLEMENTFLG)                    (* ecc "25-May-84 18:47")

          (* TCP/IP protocol checksum is the 16-bit 1's complement of the 1's complement sum of the 16-bit words)

    (PROG [(CHECKSUM 0)
```

```
          (N (SUB1 (LRSH LENGTH 1]
         [for I from 0 to N do (SETQ CHECKSUM (\16BIT.1C.PLUS CHECKSUM (\GETBASE BASE I]
         [if (ODDP LENGTH)
             then

         (* if LENGTH is odd, the last byte must be padded on the right by a zero byte)

                 (SETQ CHECKSUM (\16BIT.1C.PLUS CHECKSUM (LLSH (\GETBASEBYTE BASE (SUB1 LENGTH))
                                                                8]
         (RETURN (if DONTCOMPLEMENTFLG
                     then                                       (* if DONTCOMPLEMENTFLG is non-NIL just return the 1's
                                                                complement sum)
                          CHECKSUM
                   else (\16BIT.COMPLEMENT CHECKSUM])
```

(\TCP.CHECKSUM.INCOMING

```
  [LAMBDA (SEGMENT)                                             (* ecc "16-May-84 11:53")

         (* computes the TCP checksum and returns T or NIL depending on whether it matches the checksum in the header)

    (PROG ((LENGTH (IPLUS (fetch TCP.HEADER.LENGTH of SEGMENT)
                          (\TCP.DATA.LENGTH SEGMENT)))
           (SEGMENT.CHECKSUM (fetch TCP.CHECKSUM of SEGMENT))
           CHECKSUM OK)
          [WITH.FAST.MONITOR \TCP.LOCK                          (* need to lock this because we're using
                                                                \TCP.PSEUDOHEADER)
                 (replace PH.SRC.ADDR of \TCP.PSEUDOHEADER with (fetch TCP.SRC.ADDR of SEGMENT))
                 (replace PH.DST.ADDR of \TCP.PSEUDOHEADER with (fetch TCP.DST.ADDR of SEGMENT))
                 (replace PH.LENGTH of \TCP.PSEUDOHEADER with LENGTH)
                 (replace TCP.CHECKSUM of SEGMENT with 0)       (* checksum field must be 0 while we are computing checksum)
                 (SETQ CHECKSUM (\16BIT.COMPLEMENT (\16BIT.1C.PLUS (\COMPUTE.CHECKSUM \TCP.PSEUDOHEADER
                                                                    \TCP.PSEUDOHEADER.LENGTH T)
                                                                (\COMPUTE.CHECKSUM (fetch TCPHEADER of SEGMENT)
                                                                    LENGTH T]
          (SETQ OK (EQ CHECKSUM SEGMENT.CHECKSUM))
          (if (AND (NOT OK)
                   (MEMB 'CHECKSUM TCPTRACEFLG))
              then (printout TCPTRACEFILE .TAB0 0 "[bad checksum " CHECKSUM "]" T))
          (RETURN OK])
```

(\TCP.CHECKSUM.OUTGOING

```
  [LAMBDA (TCB SEGMENT)                                         (* ecc "16-May-84 11:53")
                                                                (* compute checksum and place in header)
    (PROG ((LENGTH (IPLUS (fetch TCP.HEADER.LENGTH of SEGMENT)
                          (\TCP.DATA.LENGTH SEGMENT)))
           (PH (if TCB
                   then (fetch TCB.PH of TCB)
                 else \TCP.PSEUDOHEADER)))
          (WITH.FAST.MONITOR \TCP.LOCK                          (* need to lock this in case we're using \TCP.PSEUDOHEADER)
                 (replace PH.SRC.ADDR of PH with (fetch TCP.SRC.ADDR of SEGMENT))
                 (replace PH.DST.ADDR of PH with (fetch TCP.DST.ADDR of SEGMENT))
                 (replace PH.LENGTH of PH with LENGTH)
                 (replace TCP.CHECKSUM of SEGMENT with 0)       (* checksum field must be 0 while we are computing checksum)
                 (replace TCP.CHECKSUM of SEGMENT with (\16BIT.COMPLEMENT (\16BIT.1C.PLUS (\COMPUTE.CHECKSUM
                                                                                           PH
                                                                                           \TCP.PSEUDOHEADER.LENGTH
                                                                                           T)
                                                                                       (\COMPUTE.CHECKSUM (fetch
                                                                                                             TCPHEADER
                                                                                                           of SEGMENT
                                                                                                             )
                                                                                           LENGTH T])

)

(DECLARE%: EVAL@COMPILE DONTCOPY

(DECLARE%: EVAL@COMPILE

(RPAQQ \TCP.INITIAL.RTO 1000)

(CONSTANTS \TCP.INITIAL.RTO)
)

(DECLARE%: EVAL@COMPILE

(RPAQQ \TCP.UBOUND 5000)

(RPAQQ \TCP.LBOUND 1000)

(CONSTANTS (\TCP.UBOUND 5000)
       (\TCP.LBOUND 1000))
)
)
```

```
;; maximum segment lifetime

(RPAQ? \TCP.MSL 5000)

(RPAQ? \TCP.DEFAULT.USER.TIMEOUT 60000)

(RPAQ? \TCP.DEFAULT.RECEIVE.WINDOW 4096)

(RPAQ? \TCP.DEVICE NIL)

;; TCP protocol routines

(DEFINEQ

(\TCP.ACK#
  [LAMBDA (TCB)                                          (* ejs%: " 7-Jun-85 13:18")

          (* * Returns the byte id for the next ACK)

                                                         (* (LET* ((STREAM (fetch TCB.RCV.STREAM of TCB))
                                                         (BUFFER (fetch TCB.RCV.SEGMENT of TCB)))
                                                         (COND (BUFFER (IPLUS (fetch TCP.SEQ of BUFFER)
                                                         (fetch (STREAM COFFSET) of STREAM)))
                                                         ((SETQ BUFFER (\QUEUEHEAD
                                                         (fetch TCB.INPUT.QUEUE of TCB)))
                                                         (IMIN (fetch TCB.RCV.NXT of TCB)
                                                         (fetch TCP.SEQ of BUFFER))) (T
                                                         (fetch TCB.RCV.NXT of TCB)))))
    (fetch TCB.RCV.NXT of TCB])

(\TCP.PACKET.FILTER
  [LAMBDA (SEGMENT PROTOCOL)                             (* ecc " 7-May-84 17:27")
                                                         (* packet filter used by IP code to dispatch packets by protocol)
    (SELECTC PROTOCOL
         (\TCP.PROTOCOL
              (ERSETQ (\TCP.INPUT SEGMENT))
              T)
         (\ICMP.PROTOCOL
              (ERSETQ (\TCP.HANDLE.ICMP SEGMENT))
              T)
        NIL])

(\TCP.SETUP.SEGMENT
  [LAMBDA (SRC.HOST SRC.PORT DST.HOST DST.PORT)          (* ejs%: " 1-Jan-01 10:28")
                                                         (* allocate a new TCP segment and set up its header)
    (PROG [(SEGMENT (\IP.SETUPIP (\ALLOCATE.ETHERPACKET)
                          DST.HOST NIL \TCP.MASTER.SOCKET 'FREE]
          (add (fetch (IP IPTOTALLENGTH) of SEGMENT)
               \TCP.HEADER.LENGTH)
          (replace TCP.SRC.PORT of SEGMENT with SRC.PORT)
          (replace TCP.DST.PORT of SEGMENT with DST.PORT)
          (replace TCP.DATA.OFFSET of SEGMENT with \TCP.MIN.DATA.OFFSET)
          (replace TCP.MBZ of SEGMENT with 0)
          (RETURN SEGMENT])

(\TCP.RELEASE.SEGMENT
  [LAMBDA (SEGMENT)                                      (* ecc " 7-May-84 17:28")
                                                         (* release a TCP segment -- it had better not be on anyone's
                                                         queue)
    (CHECK (OR (NULL (fetch QLINK of SEGMENT))
               (SHOULDNT "releasing queued segment")))
    (\RELEASE.ETHERPACKET SEGMENT])

(\TCP.CONNECTION
  [LAMBDA (DST.HOST DST.PORT SRC.PORT MODE OPTIONS)      ; Edited 23-May-88 19:14 by Snow
    ;; open a TCP connection and return the TCB or NIL if the connection fails
    (PROG (SPECIFIED TCB ISS TCP.PROCESS)
          (SELECTQ MODE
               (ACTIVE)
               (PASSIVE)
               (ERROR "TCP open mode must be ACTIVE or PASSIVE"))
          (if (NULL DST.HOST)
              then (SETQ DST.HOST 0))
          (if (NULL DST.PORT)
              then (SETQ DST.PORT 0))
          (if (NULL SRC.PORT)
              then (SETQ SRC.PORT 0))
          [SETQ SPECIFIED (NOT (OR (ZEROP DST.HOST)
                                   (ZEROP DST.PORT]
          (if (AND (EQ MODE 'ACTIVE)
                   (NOT SPECIFIED))
              then (ERROR "foreign socket unspecified"))
```

```
;; Check for conflict with existing connections.  ACTIVE open only conflicts with other fully specified connections.  PASSIVE open conflicts with
;; fully specified connections if the open is fully specifed, and with partially specified connections if the open is partially specified
      [if (SETQ TCB (OR (AND (OR (EQ MODE 'ACTIVE)
                                 SPECIFIED)
                             (\TCP.LOOKUP.TCB DST.HOST DST.PORT SRC.PORT T))
                        (AND (EQ MODE 'PASSIVE)
                             (NOT SPECIFIED)
                             (SETQ TCB (\TCP.LOOKUP.TCB DST.HOST DST.PORT SRC.PORT NIL))
                             (OR (ZEROP (fetch TCB.DST.HOST of TCB))
                                 (ZEROP (fetch TCB.DST.PORT of TCB)))
                             TCB)))
          then (COND
                  [(type? TCP.CONTROL.BLOCK TCB)
                   (COND
                      ((FMEMB (fetch TCB.STATE of TCB)
                              '(CLOSED CLOSE.WAIT TIME.WAIT FIN.WAIT.1 FIN.WAIT.2))
                       (\TCP.DELETE.TCB TCB))
                      (T (ERROR "TCP connection already exists"]
                  (T (ERROR "TCP connection already exists"]
      (SETQ TCB (\TCP.CREATE.TCB DST.HOST DST.PORT SRC.PORT MODE (OR (LISTGET OPTIONS 'MAXSEG)
                                                                     \TCP.DEFAULT.MAXSEG)))
      (replace TCB.NO.IDLE.PROBING of TCB with (LISTGET OPTIONS 'NO.IDLE.PROBING))
      (SELECTQ MODE
          (ACTIVE (WITH.MONITOR \TCP.LOCK
                      (SETQ ISS (\TCP.SELECT.ISS))
                      (replace TCB.ISS of TCB with ISS))
                  [\TCP.TEMPLATE TCB (COND
                                        ((LISTGET OPTIONS 'MAXSEG)
                                         OPTIONS)
                                        (T (APPEND OPTIONS '(MAXSEG %, \TCP.DEFAULT.MAXSEG]
                  (replace TCB.SND.UNA of TCB with ISS)
                  (replace TCB.SND.NXT of TCB with ISS)
                  (replace TCB.SND.UP of TCB with ISS)
                  (\TCP.TRACE.TRANSITION TCB 'SYN.SENT)
                  (replace TCB.STATE of TCB with 'SYN.SENT)
                  [SETQ TCP.PROCESS (ADD.PROCESS '(\TCP.PROCESS %, TCB)
                                                 'NAME
                                                 'TCP
                                                 'WHENCLOSEDFN
                                                 (LISTGET OPTIONS 'WHENCLOSEDFN]
                                                                  ; initiate the three-way handshake to establish the connection
                  (\TCP.FLUSH (fetch TCB.SND.STREAM of TCB)
                              \TCP.CTRL.SYN)                      ; wait until established
                  (WITH.MONITOR (fetch TCB.LOCK of TCB)
                                (RESETLST
                                    [RESETSAVE NIL '(AND RESETSTATE (DEL.PROCESS %, TCP.PROCESS]
                                    (until (NEQ (fetch TCB.STATE of TCB)
                                                'SYN.SENT)
                                       do (MONITOR.AWAIT.EVENT (fetch TCB.LOCK of TCB)
                                                               (fetch TCB.ESTABLISHED of TCB))))))
          (PASSIVE (\TCP.TRACE.TRANSITION TCB 'LISTEN)
                   (replace TCB.STATE of TCB with 'LISTEN)
                   [SETQ TCP.PROCESS (ADD.PROCESS '(\TCP.PROCESS %, TCB)
                                                  'NAME
                                                  'TCP
                                                  'WHENCLOSEDFN
                                                  (LISTGET OPTIONS 'WHENCLOSEDFN]
                                                                  ; wait until established
                   (WITH.MONITOR (fetch TCB.LOCK of TCB)
                                 (RESETLST
                                     [RESETSAVE NIL '(AND RESETSTATE (DEL.PROCESS %, TCP.PROCESS]
                                     (until (NEQ (fetch TCB.STATE of TCB)
                                                 'LISTEN)
                                        do (MONITOR.AWAIT.EVENT (fetch TCB.LOCK of TCB)
                                                                (fetch TCB.ESTABLISHED of TCB))))))
          (SHOULDNT))
      (RETURN (if (NEQ (fetch TCB.STATE of TCB)
                        'CLOSED)
                  then TCB
                else NIL])
```

(\TCP.FIX.INCOMING.SEGMENT

```
  [LAMBDA (SEGMENT FLAGS)                                             (* ecc "16-May-84 11:56")
    (PROG NIL
      (if (AND (BITTEST FLAGS \TCP.CTRL.SYN)
               (BITTEST FLAGS \TCP.CTRL.FIN))
          then (RETURN NIL))

      (* calculate the length of the segment data and place it in a fixed position in the header for fast access --
      note that the TCP.DATA.LENGTH field isn't a true part of the TCP header;
      it overlays the IP level checksum which is no longer needed)

      (replace TCP.DATA.LENGTH of SEGMENT with (\TCP.DATA.LENGTH SEGMENT))
                                                                      (* return T or NIL depending on whether checksum is correct)
      (RETURN (OR (NOT \TCP.CHECKSUMS.ON)
```

(\TCP.CHECKSUM.INCOMING SEGMENT])

(\TCP.DATA.LENGTH

```
[LAMBDA (SEGMENT)                                          (* ejs%: "21-Jun-85 17:04")
                                                           (* data length = total segment length -
                                                           (IP header length + TCP header length))
    (IDIFFERENCE (fetch (IP IPTOTALLENGTH) of SEGMENT)
           (IPLUS (UNFOLD (fetch (IP IPHEADERLENGTH) of SEGMENT)
                          BYTESPERCELL)
                  (UNFOLD (fetch TCP.DATA.OFFSET of SEGMENT)
                          BYTESPERCELL])
```

(\TCP.SYN.OR.FIN

```
[LAMBDA (FLAGS NOERRORFLG)                                 (* ecc " 1-May-84 17:10")

          (* SYN and FIN occupy sequence number space so we have to include them in the "length" of the segment)

    (SELECTC (LOGAND FLAGS (LOGOR \TCP.CTRL.SYN \TCP.CTRL.FIN))
         (0 0)
         (\TCP.CTRL.SYN
            1)
         (\TCP.CTRL.FIN
            1)
         (if NOERRORFLG
             then 0
           else (SHOULDNT "both SYN and FIN"])
```

(\TCP.INPUT

```
[LAMBDA (SEGMENT TCB)                                      (* ejs%: "20-Jun-85 13:06")
                                                           (* handle an incoming TCP segment --
                                                           pages 65-76 of RFC 793)
    (PROG ((SEQ (fetch TCP.SEQ of SEGMENT))
           (ACK (fetch TCP.ACK of SEGMENT))
           (FLAGS (fetch TCP.CTRL of SEGMENT))
           UNA QUEUEDFLG)
          (if (NOT (\TCP.INPUT.INITIAL TCB SEGMENT SEQ ACK FLAGS))
              then (\TCP.RELEASE.SEGMENT SEGMENT)
                   (RETURN))
          (WITH.MONITOR (fetch TCB.LOCK of TCB)
              (PROG NIL                                    (* handle unsynchronized states)
                    (if (NOT (\TCP.INPUT.UNSYNC TCB SEGMENT SEQ ACK FLAGS))
                        then (GO DROPIT))                  (* first check sequence number)
                    (if (NOT (\TCP.CHECK.WINDOW TCB SEGMENT FLAGS))
                        then (GO DROPIT))                  (* second check the RST bit)
                    (if (NOT (\TCP.CHECK.RESET TCB SEGMENT SEQ ACK FLAGS))
                        then (GO DROPIT))                  (* third check security and precedence)
                    (if (NOT (\TCP.CHECK.SECURITY TCB SEGMENT FLAGS))
                        then (GO DROPIT))                  (* fourth check the SYN bit)
                    (if (NOT (\TCP.CHECK.NO.SYN TCB SEGMENT FLAGS))
                        then (GO DROPIT))
                    (if (NOT (\TCP.CHECK.OPTIONS TCB SEGMENT FLAGS))
                        then (GO DROPIT))                  (* fifth check the ACK field)
                    (if (NOT (\TCP.CHECK.ACK TCB SEGMENT FLAGS))
                        then (GO DROPIT))
                    (if (EQ (fetch TCB.STATE of TCB)
                            'SYN.RECEIVED)
                        then (if (AND (\32BIT.LEQ (fetch TCB.SND.UNA of TCB)
                                                  ACK)
                                      (\32BIT.LEQ ACK (fetch TCB.SND.NXT of TCB)))
                                 then                      (* our SYN has been acked)
                                      (\TCP.TRACE.TRANSITION TCB 'ESTABLISHED)
                                      (replace TCB.STATE of TCB with 'ESTABLISHED)
                                      (replace TCB.DST.HOST of TCB with (fetch (TCPSEGMENT TCP.SRC.ADDR)
                                                                               of SEGMENT))
                                      (replace TCB.DST.PORT of TCB with (fetch (TCPSEGMENT TCP.SRC.PORT)
                                                                               of SEGMENT))
                                      (NOTIFY.EVENT (fetch TCB.ESTABLISHED of TCB))
                                                           (* continue processing in ESTABLISHED state)
                               else (\TCP.SEND.CONTROL TCB ACK NIL \TCP.CTRL.RST)
                                    (GO DROPIT)))
                    (if (NOT (\TCP.HANDLE.ACK TCB SEGMENT SEQ ACK FLAGS))
                        then (GO DROPIT))
                    (SELECTQ (fetch TCB.STATE of TCB)
                         (FIN.WAIT.1 (if (\TCP.OUR.FIN.IS.ACKED TCB)
                                         then (\TCP.TRACE.TRANSITION TCB 'FIN.WAIT.2)
                                              (replace TCB.STATE of TCB with 'FIN.WAIT.2)
                                              (NOTIFY.EVENT (fetch TCB.FINACKED.EVENT of TCB))))
                         ((ESTABLISHED FIN.WAIT.2 CLOSE.WAIT)
                             NIL)
                         (CLOSING (if (\TCP.OUR.FIN.IS.ACKED TCB)
                                      then (\TCP.START.TIME.WAIT TCB)
                                           (NOTIFY.EVENT (fetch TCB.FINACKED.EVENT of TCB))
                                    else (GO DROPIT)))
                         (LAST.ACK (if (\TCP.OUR.FIN.IS.ACKED TCB)
```

```
                                    then (\TCP.TRACE.TRANSITION TCB 'CLOSED)
                                         (replace TCB.STATE of TCB with 'CLOSED)
                                         (NOTIFY.EVENT (fetch TCB.FINACKED.EVENT of TCB))
                                         (RETURN)
                                  else (GO DROPIT)))
                             (TIME.WAIT (\TCP.SEND.ACK TCB)
                                        (GO DROPIT))
                             (SHOULDNT))                                (* sixth check the URG bit)
                        (\TCP.HANDLE.URG TCB SEGMENT SEQ ACK FLAGS)     (* seventh process the segment text)
                        (SELECTQ (fetch TCB.STATE of TCB)
                             ((ESTABLISHED FIN.WAIT.1 FIN.WAIT.2)
                                 (SETQ QUEUEDFLG (\TCP.QUEUE.INPUT TCB SEGMENT SEQ)))
                             ((CLOSE.WAIT CLOSING LAST.ACK TIME.WAIT))
                             (SHOULDNT))                                (* eighth check the FIN bit)
                        (\TCP.HANDLE.FIN TCB SEGMENT SEQ ACK FLAGS)
                        (if QUEUEDFLG
                            then (RETURN))
                   DROPIT
                        (\TCP.RELEASE.SEGMENT SEGMENT)))])
```

(\TCP.INPUT.INITIAL

```
[LAMBDA (TCB SEGMENT SEQ ACK FLAGS)                                     (* ecc "16-May-84 17:27")
                                                                        (* handle segment for non-existent TCB --
                                                                        page 65 of RFC 793)
  (PROG NIL
        (\TCP.TRACE.SEGMENT 'RECV SEGMENT)
        (if (NOT (\TCP.FIX.INCOMING.SEGMENT SEGMENT FLAGS))
            then                                                        (* bad checksum)
                 (RETURN NIL))
        (if (OR (NULL TCB)
                (EQ (fetch TCB.STATE of TCB)
                    'CLOSED))
            then

        (* an incoming segment not containing a RST causes a RST to be sent in response)

                 (if TCPTRACEFLG
                     then (printout TCPTRACEFILE .TAB0 0 "[no such TCP connection]"))
                 [if (NOT (BITTEST FLAGS \TCP.CTRL.RST))
                     then                                               (* send a RST)
                          (if (BITTEST FLAGS \TCP.CTRL.ACK)
                              then (\TCP.SEND.RESET SEGMENT ACK)
                            else (\TCP.SEND.RESET SEGMENT 0 (IPLUS SEQ (fetch TCP.DATA.LENGTH of SEGMENT)
                                                                   (\TCP.SYN.OR.FIN FLAGS]

                 (RETURN NIL))
        (RETURN T])
```

(\TCP.INPUT.UNSYNC

```
[LAMBDA (TCB SEGMENT SEQ ACK FLAGS)                                     (* ejs%: "21-Mar-86 20:03")

        (* handle segment for TCB in LISTEN or SYN.SENT state -- pages 65-68 of RFC 793)

  (SELECTQ (fetch TCB.STATE of TCB)
       (LISTEN (\TCP.INPUT.LISTEN TCB SEGMENT SEQ ACK FLAGS)
               NIL)
       (SYN.SENT (\TCP.INPUT.SYN.SENT TCB SEGMENT SEQ ACK FLAGS)
                 (\TCP.CHECK.OPTIONS TCB SEGMENT FLAGS)
                 NIL)
       T])
```

(\TCP.INPUT.LISTEN

```
[LAMBDA (TCB SEGMENT SEQ ACK FLAGS)                                     (* ejs%: "22-Jun-85 03:14")
                                                                        (* handle segment for TCB in LISTEN state --
                                                                        pages 65-66 of RFC 793)
  (PROG (ISS)                                                           (* first check for a RST)
        (if (BITTEST FLAGS \TCP.CTRL.RST)
            then (RETURN NIL))                                          (* second check for an ACK)
        (if (BITTEST FLAGS \TCP.CTRL.ACK)
            then

        (* any acknowledgment is bad if it arrives on a connection still in the LISTEN state)

                 (\TCP.SEND.RESET SEGMENT ACK)
                 (RETURN NIL))                                          (* third check for a SYN)
        (if (BITTEST FLAGS \TCP.CTRL.SYN)
            then (if (NOT (\TCP.CHECK.SECURITY TCB SEGMENT FLAGS))
                     then (RETURN NIL))
                 (replace TCB.RCV.NXT of TCB with (ADD1 SEQ))
                 (replace TCB.IRS of TCB with SEQ)
                 (SETQ ISS (\TCP.SELECT.ISS))
                 (replace TCB.ISS of TCB with ISS)
                 (replace TCB.SND.NXT of TCB with ISS)
                 (replace TCB.SND.UNA of TCB with ISS)
                 (replace TCB.SND.UP of TCB with ISS)
```

```
                (\TCP.TRACE.TRANSITION TCB 'SYN.RECEIVED)
                (replace TCB.STATE of TCB with 'SYN.RECEIVED)    (* fill in foreign socket in case it was only partially specified)
                (replace TCB.DST.HOST of TCB with (fetch TCP.SRC.ADDR of SEGMENT))
                (replace TCB.DST.PORT of TCB with (fetch TCP.SRC.PORT of SEGMENT))
                (\TCP.TEMPLATE TCB)                              (* send a SYN, ACK segment using \TCP.FLUSH because SYN
                occupies sequence number space)
                (\TCP.FLUSH (fetch TCB.SND.STREAM of TCB)
                       \TCP.CTRL.SYN)

        (* NOTE%: we never queue data that arrives in a SYN segment, we just ACK the SYN and require the data to be
        retransmitted)

                )
        (RETURN NIL])
```

(\TCP.INPUT.SYN.SENT

```
  [LAMBDA (TCB SEGMENT SEQ ACK FLAGS)                            (* ecc "16-May-84 12:13")
                                                                 (* handle segment for TCB in SYN.SENT state --
                                                                 pages 66-68 of RFC 793)
    (PROG NIL                                                    (* first check the ACK bit)
          (if (BITTEST FLAGS \TCP.CTRL.ACK)
              then (if (OR (\32BIT.LEQ ACK (fetch TCB.ISS of TCB))
                           (\32BIT.GT ACK (fetch TCB.SND.NXT of TCB)))
                       then                                      (* ACK is unacceptable)
                            (if (NOT (BITTEST FLAGS \TCP.CTRL.RST))
                                then (\TCP.SEND.CONTROL TCB ACK NIL \TCP.CTRL.RST))
                            (RETURN NIL)))                       (* second check the RST bit)
          (if (BITTEST FLAGS \TCP.CTRL.RST)
              then (if (BITTEST FLAGS \TCP.CTRL.ACK)
                       then                                      (* if the ACK was acceptable then signal the user)
                            (\TCP.CONNECTION.DROPPED TCB "reset"))
                   (RETURN NIL))                                 (* third check the security and precedence)
          (if (NOT (\TCP.CHECK.SECURITY TCB SEGMENT FLAGS))
              then (RETURN NIL))                                 (* fourth check the SYN bit)
          (if (BITTEST FLAGS \TCP.CTRL.SYN)
              then (replace TCB.RCV.NXT of TCB with (ADD1 SEQ))
                   (replace TCB.IRS of TCB with SEQ)
                   (if (AND (BITTEST FLAGS \TCP.CTRL.ACK)
                            (\32BIT.GEQ ACK (fetch TCB.SND.UNA of TCB)))
                       then                                      (* new ACK information)
                            (replace TCB.SND.UNA of TCB with ACK))
                   (replace TCP.CTRL of SEGMENT with (SETQ FLAGS (BITCLEAR FLAGS \TCP.CTRL.SYN)))
                   (if (\32BIT.GT (fetch TCB.SND.UNA of TCB)
                                  (fetch TCB.ISS of TCB))
                       then                                      (* our SYN has been acked)
                            (\TCP.TRACE.TRANSITION TCB 'ESTABLISHED)
                            (replace TCB.STATE of TCB with 'ESTABLISHED)
                                                                 (* send an ACK segment)
                            (\TCP.SEND.ACK TCB 'NOW)
                            (NOTIFY.EVENT (fetch TCB.ESTABLISHED of TCB))
                     else                                        (* we can just let our original SYN segment be retransmitted)
                          (\TCP.TRACE.TRANSITION TCB 'SYN.RECEIVED)
                          (replace TCB.STATE of TCB with 'SYN.RECEIVED)
                                                                 (* send an ACK segment)
                          (\TCP.SEND.ACK TCB 'NOW))

        (* NOTE%: we never queue data that arrives in a SYN segment, we just ACK the SYN and require the data to be
        retransmitted)

                )                                                (* drop the segment and return)
        (RETURN NIL])
```

(\TCP.CHECK.WINDOW

```
  [LAMBDA (TCB SEGMENT FLAGS)                                    (* ecc "16-May-84 16:29")
                                                                 (* check segment length against receive window --
                                                                 page 69 of RFC 793)
    (PROG ((LEN (fetch TCP.DATA.LENGTH of SEGMENT))
           (SEQ (fetch TCP.SEQ of SEGMENT))
           (RCV.NXT (fetch TCB.RCV.NXT of TCB))
           (WND (fetch TCB.RCV.WND of TCB))
           TOP)
          (SETQ TOP (IPLUS SEQ LEN (\TCP.SYN.OR.FIN FLAGS)))
          [if (ZEROP LEN)
              then (if (ZEROP WND)
                       then (if (\32BIT.EQ SEQ RCV.NXT)
                                then (RETURN T))
                     else (if (AND (\32BIT.LEQ RCV.NXT SEQ)
                                   (\32BIT.LT SEQ (IPLUS RCV.NXT WND)))
                              then (RETURN T)))
            else (if (NOT (ZEROP WND))
                     then (if [OR (AND (\32BIT.LEQ RCV.NXT SEQ)
                                       (\32BIT.LT SEQ (IPLUS RCV.NXT WND)))
                                  (AND (\32BIT.LT RCV.NXT TOP)
                                       (\32BIT.LEQ TOP (IPLUS RCV.NXT WND]
                              then (RETURN T]
```

```
          (if (NOT (BITTEST FLAGS \TCP.CTRL.RST))
              then                                              (* send an ACK in reply)
                   (\TCP.SEND.ACK TCB 'NOW))
          (RETURN NIL])
```

(\TCP.CHECK.RESET

```
  [LAMBDA (TCB SEGMENT SEQ ACK FLAGS)                           (* ecc "16-May-84 12:07")
                                                                (* check the RST bit -- page 70 of RFC 793)
    (PROG NIL
          (if (BITTEST FLAGS \TCP.CTRL.RST)
              then (SELECTQ (fetch TCB.STATE of TCB)
                            (SYN.RECEIVED (if (EQ (fetch TCB.MODE of TCB)
                                                  'PASSIVE)
                                              then (\TCP.TRACE.TRANSITION TCB 'LISTEN)
                                                   (replace TCB.STATE of TCB with 'LISTEN)
                                            else (\TCP.CONNECTION.DROPPED TCB "refused"))
                                          (\FLUSH.PACKET.QUEUE (fetch TCB.REXMT.QUEUE of TCB))
                                          (\TCP.SEND.CONTROL TCB ACK NIL \TCP.CTRL.RST))
                            ((ESTABLISHED FIN.WAIT.1 FIN.WAIT.2 CLOSE.WAIT)
                                 (\TCP.CONNECTION.DROPPED TCB "reset"))
                            ((CLOSING LAST.ACK TIME.WAIT)
                                 (\TCP.TRACE.TRANSITION TCB 'CLOSED)
                                 (replace TCB.STATE of TCB with 'CLOSED))
                            (SHOULDNT))
                   (RETURN NIL)
            else (RETURN T])
```

(\TCP.CHECK.SECURITY

```
  [LAMBDA (TCB SEGMENT FLAGS)                                   (* ecc "16-May-84 12:06")

          (* returns T or NIL depending on whether security and precedence are OK;
          sends RST if necessary)
                                                                (* not implemented)
    T])
```

(\TCP.CHECK.NO.SYN

```
  [LAMBDA (TCB SEGMENT FLAGS)                                   (* ecc "16-May-84 12:07")
                                                                (* check the SYN bit -- page 71 of RFC 793)
    (PROG NIL
          (CHECK (OR (NOT (BITTEST FLAGS \TCP.CTRL.RST))
                     (SHOULDNT "RST bit set")))
          (if (NOT (BITTEST FLAGS \TCP.CTRL.SYN))
              then (RETURN T))
          (if (BITTEST FLAGS \TCP.CTRL.ACK)
              then (\TCP.SEND.CONTROL TCB (fetch TCP.ACK of SEGMENT)
                                      NIL \TCP.CTRL.RST)
            else (\TCP.SEND.CONTROL TCB 0 (IPLUS (fetch TCP.ACK of SEGMENT)
                                                 (fetch TCP.DATA.LENGTH of SEGMENT)
                                                 1)
                                      (LOGOR \TCP.CTRL.ACK \TCP.CTRL.RST)))
          (\TCP.CONNECTION.DROPPED TCB "reset")
          (RETURN NIL])
```

(\TCP.CHECK.ACK

```
  [LAMBDA (TCB SEGMENT FLAGS)                                   (* ecc "16-May-84 12:08")
                                                                (* check the ACK field -- page 72 of RFC 793)
    (PROG NIL
          (CHECK (OR (NOT (BITTEST FLAGS (LOGOR \TCP.CTRL.SYN \TCP.CTRL.RST)))
                     (SHOULDNT "SYN or RST bit set")))
          (RETURN (BITTEST FLAGS \TCP.CTRL.ACK])
```

(\TCP.HANDLE.ACK

```
  [LAMBDA (TCB SEGMENT SEQ ACK FLAGS)                           (* ejs%: "22-Jun-85 00:35")
                                                                (* ACK processing -- pages 72-73 of RFC 793)
    (PROG (EVENT)
          (if (\32BIT.GT ACK (fetch TCB.SND.NXT of TCB))
              then                                              (* this segment acks something we haven't sent yet)
                   (\TCP.SEND.ACK TCB 'NOW)
                   (RETURN NIL))
          [if (AND (fetch TCB.RTFLG of TCB)
                   (\32BIT.GT ACK (fetch TCB.RTSEQ of TCB)))
              then                                              (* calculate smoothed round trip time)
                   (replace TCB.RTFLG of TCB with NIL)
                   (replace TCB.SRTT of TCB with (FOLDLO (PLUS (ITIMES 7 (fetch TCB.SRTT of TCB))
                                                               (CLOCKDIFFERENCE (fetch TCB.RTTIMER of TCB)))
                                                         8))
                   (replace TCB.RTTIMER of TCB with (SETUPTIMER 0 (fetch TCB.RTTIMER of TCB)))
                   (replace TCB.RTO of TCB with (IMIN \TCP.UBOUND (IMAX \TCP.LBOUND
                                                                        (FOLDLO (ITIMES 3 (fetch TCB.SRTT
                                                                                                 of TCB))
                                                                                2]
          (if (\32BIT.GT ACK (fetch TCB.SND.UNA of TCB))
```

```
            then                                                    (* new ACK information)
                 (replace TCB.SND.UNA of TCB with ACK)
                 (SETQ EVENT T))
            (if [OR (\32BIT.GT SEQ (fetch TCB.SND.WL1 of TCB))
                    (AND (\32BIT.EQ SEQ (fetch TCB.SND.WL1 of TCB))
                         (\32BIT.GEQ ACK (fetch TCB.SND.WL2 of TCB]
                then                                                (* update send window)
                 (replace TCB.SND.WND of TCB with (fetch TCP.WINDOW of SEGMENT))
                 (replace TCB.SND.WL1 of TCB with SEQ)
                 (replace TCB.SND.WL2 of TCB with ACK)
                 (SETQ EVENT T))
            (if EVENT
                then (NOTIFY.EVENT (fetch TCB.SND.EVENT of TCB)))
            (RETURN T])
```

(\TCP.HANDLE.URG

```
  [LAMBDA (TCB SEGMENT SEQ ACK FLAGS)                               (* ecc "16-May-84 12:10")
                                                                    (* check the URG bit -- pages 73-74 of RFC 793)
    (PROG (UP)
          (if (BITTEST FLAGS \TCP.CTRL.URG)
              then (SELECTQ (fetch TCB.STATE of TCB)
                            ((ESTABLISHED FIN.WAIT.1 FIN.WAIT.2)
                                 (SETQ UP (IPLUS SEQ (fetch TCP.URG.PTR of SEGMENT)))
                                 (if (\32BIT.GT UP (fetch TCB.RCV.UP of TCB))
                                     then (replace TCB.RCV.UP of TCB with UP)
                                          (if (\32BIT.GT UP (fetch TCB.RCV.NXT of TCB))
                                              then                  (* urgent pointer is in advance of the data consumed)
                                                   (\TCP.SIGNAL.URGENT.DATA TCB))))
                            ((CLOSE.WAIT CLOSING LAST.ACK TIME.WAIT)
                                NIL)
                            (SHOULDNT])
```

(\TCP.QUEUE.INPUT

```
  [LAMBDA (TCB SEGMENT SEQ)                                         (* ejs%: "18-Dec-86 17:39")
```

(* Put the segment in its proper position in the input queue according to its sequence number range.
Returns T if the segment was queued, NIL if it was a duplicate.
Segments are queued by increasing left endpoint of their sequence number range.
If the entire sequence number range has been seen or is covered by segments already in the queue, the segment is a duplicate. Otherwise, it covers some gap in the queue, so it is placed in its proper position.
Note that a later segment that covers gaps on both sides will also be queued, resulting in duplicates in the queue. Therefore \TCP.GET.SEGMENT must be prepared to skip over duplicates.)

```
    (CHECK (OR (NULL (fetch QLINK of SEGMENT))
               (SHOULDNT "input segment already queued")))
    (CHECK (\TCP.CHECK.INPUT.QUEUE TCB))
    (UNINTERRUPTABLY
        (PROG ((QUEUE (fetch TCB.INPUT.QUEUE of TCB))
               (RCV.NXT (fetch TCB.RCV.NXT of TCB))
               (LEN (fetch TCP.DATA.LENGTH of SEGMENT))
               (FLAGS (fetch TCP.CTRL of SEGMENT))
              TOP CURRENT CURSEQ NEXT)
              (if (EQ 0 LEN)
                  then                                              (* this segment has no data)
                       (GO DROPITANDPROBE))
              (SETQ TOP (IPLUS SEQ LEN))
              (if (\32BIT.LEQ TOP RCV.NXT)
                  then                                              (* this segment is a duplicate)
                       (GO DROPITANDPROBE))
              (SETQ CURRENT (fetch SYSQUEUEHEAD of QUEUE))
              (SETQ NEXT (fetch SYSQUEUETAIL of QUEUE))
              (if (OR (NULL CURRENT)
                      (\32BIT.GEQ SEQ (fetch TCP.SEQ of NEXT)))
                  then
```

(* the segment goes at the tail of the queue -- we check this first since this is the expected case)

```
                       (\ENQUEUE QUEUE SEGMENT)
                elseif (\32BIT.LT SEQ (SETQ CURSEQ (fetch TCP.SEQ of CURRENT)))
                  then                                              (* the segment goes at the head of the queue)
                       (replace QLINK of SEGMENT with CURRENT)
                       (replace SYSQUEUEHEAD of QUEUE with SEGMENT)
                else
```

(* * Search for this segment's proper position in the queue. The invariant upon entering this loop is%: segment.seq >= current.seq)

```
                     (do (if (\32BIT.LEQ TOP (IPLUS CURSEQ (fetch TCP.DATA.LENGTH of CURRENT)))
                             then
```

(* * segment.seq <= current.seq + current.length. The packet is totally subsumed by a previously received packet, and thus, is a duplicate and is dropped)

```
                                 (GO DROPITANDPROBE))
                         (SETQ NEXT (fetch QLINK of CURRENT))
```

```
                    (SETQ CURSEQ (fetch TCP.SEQ of NEXT))
                    (if (\32BIT.LT SEQ CURSEQ)
                        then

(* * current.seq <= segment.seq < next.seq. Insert the segment between current and next)

                            (replace QLINK of SEGMENT with NEXT)
                            (replace QLINK of CURRENT with SEGMENT)
                            (RETURN))
                    (SETQ CURRENT NEXT)))

(* * Note that we have a zero window allocation at this point. When we free up the window
(in \TCP.GET.SEGMENT)%, we'll know to send a gratuitous ACK to our partner to let it know the window's once again
open.)

    (replace TCB.RCV.WND of TCB with (IMAX 0 (IDIFFERENCE (fetch TCB.RCV.WND of TCB)
                                                          LEN)))
    (replace TCB.LAST.SENT.RCV.WND of TCB with (IMAX 0 (IDIFFERENCE (fetch TCB.LAST.SENT.RCV.WND
                                                                       of TCB)
                                                                    LEN)))
    (COND
       ((OR (EQ 0 (fetch TCB.LAST.SENT.RCV.WND of TCB))
            (EQ 0 (fetch TCB.RCV.WND of TCB)))
        (replace TCB.SENT.ZERO of TCB with T)))
    [while (AND (\32BIT.LEQ SEQ RCV.NXT)
                (\32BIT.LT RCV.NXT TOP))
       do                                                    (* advance RCV.NXT)
          (replace TCB.RCV.NXT of TCB with (SETQ RCV.NXT TOP))
          (if (SETQ SEGMENT (fetch QLINK of SEGMENT))
              then (SETQ TOP (IPLUS (SETQ SEQ (fetch TCP.SEQ of SEGMENT))
                                    (fetch TCP.DATA.LENGTH of SEGMENT]
    (if (BITTEST FLAGS \TCP.CTRL.PSH)
        then (\TCP.SEND.ACK TCB 'NOW)
      else (\TCP.SEND.ACK TCB))
    (NOTIFY.EVENT (fetch TCB.RCV.EVENT of TCB))
    (RETURN T)
DROPITANDPROBE


(* * Here when we think we should let the other side know immediately about our condition
(e.g. a duplicate packet was received))

    (\TCP.SEND.ACK TCB 'NOW)
DROPIT


(* * Here when we have nothing to do, but it's not worth informing our TCP partner)

    (RETURN NIL)))])
```

(\TCP.HANDLE.FIN

```
  [LAMBDA (TCB SEGMENT SEQ ACK FLAGS)                          (* ejs%: "11-Aug-86 22:29")
                                                               (* check the FIN bit -- pages 75-76 of RFC 793)
    (PROG (TOP)
          (if (BITTEST FLAGS \TCP.CTRL.FIN)
              then (SETQ TOP (IPLUS SEQ (fetch TCP.DATA.LENGTH of SEGMENT)))
                                                               (* check whether we've received all the data before the FIN)
                   (if (\32BIT.GEQ (fetch TCB.RCV.NXT of TCB)
                                   TOP)
                       then (if (\32BIT.EQ (fetch TCB.RCV.NXT of TCB)
                                           TOP)
                                then                           (* advance RCV.NXT over the FIN)
                                     (add (fetch TCB.RCV.NXT of TCB)
                                          1))
                            (SELECTQ (fetch TCB.STATE of TCB)
                                ((SYN.RECEIVED ESTABLISHED)
                                     (\TCP.TRACE.TRANSITION TCB 'CLOSE.WAIT)
                                     (replace TCB.STATE of TCB with 'CLOSE.WAIT))
                                (FIN.WAIT.1 (if (\TCP.OUR.FIN.IS.ACKED TCB)
                                                then (\TCP.START.TIME.WAIT TCB)
                                                     (NOTIFY.EVENT (fetch TCB.FINACKED.EVENT of TCB))
                                              else (\TCP.TRACE.TRANSITION TCB 'CLOSING)
                                                   (replace TCB.STATE of TCB with 'CLOSING)))
                                (FIN.WAIT.2 (\TCP.START.TIME.WAIT TCB))
                                ((CLOSE.WAIT CLOSING LAST.ACK)
                                    NIL)
                                (TIME.WAIT (\TCP.START.TIME.WAIT TCB))
                                (SHOULDNT))
                            (NOTIFY.EVENT (fetch TCB.RCV.EVENT of TCB)))
                                                               (* acknowledge the FIN)
                   (\TCP.SEND.ACK TCB 'NOW])
```

(\TCP.OUR.FIN.IS.ACKED

```
  [LAMBDA (TCB)                                                (* ecc "16-May-84 12:15")
                                                               (* check whether our FIN's sequence number
                                                               (recorded in the TCB.FINSEQ field) has been acknowledged)
```

```
    (\32BIT.GEQ (fetch TCB.SND.UNA of TCB)
           (OR (fetch TCB.FINSEQ of TCB)
               (SHOULDNT "FIN not sent"])
```

(\TCP.SIGNAL.URGENT.DATA

```
  [LAMBDA (TCB)                                                    (* ecc " 7-May-84 12:19")
    (NOTIFY.EVENT (fetch TCB.URGENT.EVENT of TCB))
    (if TCPTRACEFLG
        then (printout TCPTRACEFILE .TAB0 0 "[Urgent TCP data has arrived]" T])
```

(\TCP.PROCESS

```
  [LAMBDA (TCB)                                                    (* ejs%: "11-Aug-86 21:57")
                                                                   (* process to handle retransmission and timeouts for TCP
                                                                   connection)

    (RESETSAVE NIL (LIST (FUNCTION \TCP.DELETE.TCB)
                         TCB))
    [PROCESSPROP (THIS.PROCESS)
           'INFOHOOK
           (FUNCTION (LAMBDA NIL
                     (PPTCB TCB]
    (replace TCB.PROCESS of TCB with (THIS.PROCESS))
    (WITH.MONITOR (fetch TCB.LOCK of TCB)
        [bind SEGMENT PACKETQUEUE REXMTQUEUE EVENT (IPSOCKET _ (fetch TCB.IPSOCKET of TCB))
           first (SETQ PACKETQUEUE (fetch (IPSOCKET IPSQUEUE) of IPSOCKET))
                 (SETQ REXMTQUEUE (fetch TCB.REXMT.QUEUE of TCB))
                 (SETQ EVENT (fetch (IPSOCKET IPSEVENT) of IPSOCKET))
           while (NEQ (fetch TCB.STATE of TCB)
                      'CLOSED)
           do (COND
                 ((AND (fetch TCB.RTFLG of TCB)
                       (fetch TCB.PROBE.TIMER of TCB)
                       (IGREATERP (CLOCKDIFFERENCE (fetch TCB.RTTIMER of TCB))
                                  (fetch TCB.USER.TIMEOUT of TCB)))   (* timeout has expired without other end responding)
                  (\TCP.CONNECTION.DROPPED TCB "not responding"))
                 ((AND (EQ (fetch TCB.STATE of TCB)
                           'TIME.WAIT)
                       (TIMEREXPIRED? (fetch TCB.2MSL.TIMER of TCB)))
                                                                   (* 2MSL has expired)
                  (\TCP.TRACE.TRANSITION TCB 'CLOSED)
                  (replace TCB.STATE of TCB with 'CLOSED))
                 ((\TCP.RETRANSMIT TCB)
                  NIL)
                 ([OR (EQ (fetch TCB.ACKFLG of TCB)
                          'NOW)
                      (AND (EQ (fetch TCB.STATE of TCB)
                               'ESTABLISHED)
                           (fetch TCB.PROBE.TIMER of TCB)
                           (TIMEREXPIRED? (fetch TCB.PROBE.TIMER of TCB)))
                      (AND (\QUEUEHEAD (fetch TCB.INPUT.QUEUE of TCB))
                           (\32BIT.GT (fetch TCP.SEQ of (\QUEUEHEAD (fetch TCB.INPUT.QUEUE of TCB)))
                                  (fetch TCB.RCV.NXT of TCB]

          (* an ACK needs to be sent either because the protocol routines requested it or because we need to fill a gap in the input
          queue)

                  (\TCP.SEND.CONTROL TCB (fetch TCB.SND.NXT of TCB)
                         (\TCP.ACK# TCB)
                         \TCP.CTRL.ACK))
                 ((AND (\32BIT.GT (fetch TCB.SND.NXT of TCB)
                                  (IPLUS (fetch TCB.SND.WL1 of TCB)
                                         (fetch TCB.SND.WND of TCB)))
                       (fetch TCB.PROBE.TIMER of TCB)
                       (TIMEREXPIRED? (fetch TCB.PROBE.TIMER of TCB)))
                                                                   (* a probe needs to be sent to open the window)
                  (\TCP.SEND.CONTROL TCB (IPLUS (fetch TCB.SND.NXT of TCB)
                                                (fetch TCB.SND.WND of TCB))
                         (\TCP.ACK# TCB)
                         \TCP.CTRL.ACK)))
              (COND
                 ((SETQ SEGMENT (\DEQUEUE PACKETQUEUE))
                  (add (fetch (IPSOCKET IPSQUEUELENGTH) of IPSOCKET)
                       -1)
                  (\TCP.INPUT SEGMENT TCB))
                 (T (COND
                       ((EQ [COND
                              ([OR (fetch TCB.OUTPUT.HELD of TCB)
                                   (fetch SYSQUEUEHEAD of REXMTQUEUE)
                                   (\32BIT.GT (fetch TCB.SND.NXT of TCB)
                                          (IPLUS (fetch TCB.SND.WL1 of TCB)
                                                 (fetch TCB.SND.WND of TCB]
                                                                   (* Something on the retransmit queue.
                                                                   Be agressive.)
                            (MONITOR.AWAIT.EVENT (fetch TCB.LOCK of TCB)
                                   EVENT
                                   (fetch TCB.RTO of TCB)))
```

```
                                (T                                  (* Nothing to do. Be lazy)
                                   (MONITOR.AWAIT.EVENT (fetch TCB.LOCK of TCB)
                                          EVENT
                                          (fetch TCB.PROBE.TIMER of TCB)
                                          (NOT (NULL (fetch TCB.PROBE.TIMER of TCB]
                               EVENT)
                            (COND
                               ((SETQ SEGMENT (\DEQUEUE PACKETQUEUE))
                                (add (fetch (IPSOCKET IPSQUEUELENGTH) of IPSOCKET)
                                     -1)
                                (\TCP.INPUT SEGMENT TCB])])
```

(\TCP.TEMPLATE

```
  [LAMBDA (TCB OPTIONS)                                 (* ejs%: "21-Jun-85 16:40")
                                                        (* set up segment for sending control information and
                                                        pseudo-header for checksumming)

    (LET ((SEGMENT (fetch TCB.TEMPLATE of TCB)))
         [if SEGMENT
             then (replace TCP.DST.ADDR of SEGMENT with (fetch TCB.DST.HOST of TCB))
                  (replace TCP.DST.PORT of SEGMENT with (fetch TCB.DST.PORT of TCB))
           else (SETQ SEGMENT (\TCP.SETUP.SEGMENT (\LOCAL.IP.ADDRESS)
                                    (fetch TCB.SRC.PORT of TCB)
                                    (fetch TCB.DST.HOST of TCB)
                                    (fetch TCB.DST.PORT of TCB]
         (if OPTIONS
             then (\TCP.SETUP.SEGMENT.OPTIONS SEGMENT OPTIONS))
         (replace TCB.TEMPLATE of TCB with SEGMENT)
         (if (NULL (fetch TCB.PH of TCB))
             then (replace TCB.PH of TCB with (create TCP.PSEUDOHEADER)))
         SEGMENT])
```

(\TCP.SETUP.SEGMENT.OPTIONS

```
  [LAMBDA (SEGMENT OPTIONS)                             (* ejs%: "28-Jul-86 13:31")

          (* * Add options to a freshly setup segment. OPTIONS is in PLIST format)

    (LET ((OPTIONSBASE (fetch TCP.OPTIONS of SEGMENT))
          (OPTIONSOFFSET 0)
          DIDPLACEOPTION)
         (COND
            ((IGREATERP (fetch (IP IPTOTALLENGTH) of SEGMENT)
                        (CONSTANT (IPLUS \TCP.HEADER.LENGTH \IPOVLEN)))
             (ERROR "Tried to add options to a segment with TCP data already in place" SEGMENT)))
         (for OPTIONVALUETAIL on OPTIONS by (CDDR OPTIONVALUETAIL)
            do (SELECTQ (CAR OPTIONVALUETAIL)
                        (MAXSEG [LET ((VALUE (CADR OPTIONVALUETAIL)))
                                     (COND
                                        ((SMALLP VALUE)
                                         (\PUTBASEBYTE OPTIONSBASE OPTIONSOFFSET \TCPOPT.MAXSEG)
                                         (\PUTBASEBYTE OPTIONSBASE (add OPTIONSOFFSET 1)
                                                4)
                                         (\PUTBASEBYTE OPTIONSBASE (add OPTIONSOFFSET 1)
                                                (LOGAND (MASK.1'S 0 BITSPERBYTE)
                                                        (LRSH VALUE BITSPERBYTE)))
                                         (\PUTBASEBYTE OPTIONSBASE (add OPTIONSOFFSET 1)
                                                (LOGAND VALUE (MASK.1'S 0 BITSPERBYTE)))
                                         (SETQ DIDPLACEOPTION T])
                                NIL))
         (COND
            (DIDPLACEOPTION (\PUTBASEBYTE OPTIONSBASE (add OPTIONSOFFSET 1)
                                   \TCPOPT.END)))
         (until (EQ 0 (IMOD OPTIONSOFFSET 4)) do (\PUTBASEBYTE OPTIONSBASE (add OPTIONSOFFSET 1)
                                                        \TCPOPT.END))
         (add (fetch (IP IPTOTALLENGTH) of SEGMENT)
              OPTIONSOFFSET)
         (add (fetch TCP.DATA.OFFSET of SEGMENT)
              (FOLDHI OPTIONSOFFSET BYTESPERCELL])
```

(\TCP.SEND.CONTROL

```
  [LAMBDA (TCB SEQ ACK FLAGS)                           (* ejs%: "18-Dec-86 17:29")
                                                        (* send a control segment with the specified sequence number
                                                        and ACK information)

    (PROG [(SEGMENT (OR (fetch TCB.TEMPLATE of TCB)
                        (\TCP.NEW.TEMPLATE TCB]
          (if (NULL FLAGS)
              then (SETQ FLAGS 0))
          (CHECK (OR (NOT (BITTEST FLAGS (LOGOR \TCP.CTRL.SYN \TCP.CTRL.FIN)))
                     (SHOULDNT "SYN or FIN")))
          (while (fetch EPTRANSMITTING of SEGMENT) do (BLOCK))
          (replace TCP.SEQ of SEGMENT with SEQ)
          (if ACK
              then (replace TCP.ACK of SEGMENT with ACK)
                   (SETQ FLAGS (LOGOR FLAGS \TCP.CTRL.ACK))
            else (replace TCP.ACK of SEGMENT with 0))
```

```
        (replace TCP.CTRL of SEGMENT with FLAGS)
        [replace TCB.SENT.ZERO of TCB with (EQ 0 (replace TCP.WINDOW of SEGMENT
                                                  with (replace TCB.LAST.SENT.RCV.WND of TCB
                                                         with (fetch TCB.RCV.WND of TCB]
        (\TCP.SEND.SEGMENT TCB SEGMENT FLAGS)
        (\TCP.NEW.TEMPLATE TCB])
```

(\TCP.SEND.ACK

```
  [LAMBDA (TCB WHEN)                                                    (* ejs%: "17-Dec-86 16:43")

        (* set TCB.ACKFLG to tell the \TCP.PROCESS that an ACK needs to be sent --
        NOW means send the ack immediately, LATER means delay in the hope that it can be piggybacked on an outgoing data
        segment)

    (COND
       ((EQ WHEN 'NOW)
        (\TCP.SEND.CONTROL TCB (fetch TCB.SND.NXT of TCB)
               (\TCP.ACK# TCB)
               \TCP.CTRL.ACK))
       (T (replace TCB.ACKFLG of TCB with (OR WHEN 'LATER])
```

(\TCP.SEND.RESET

```
  [LAMBDA (ORIG SEQ ACK FLAGS)                                          (* ejs%: " 7-Jun-85 12:58")
                                                                        (* like \TCP.SEND.CONTROL but always sends RST and can
                                                                        be used without a TCB)
    (PROG (SEGMENT)
          (SETQ SEGMENT (\TCP.SETUP.SEGMENT (\LOCAL.IP.ADDRESS)
                                (fetch TCP.DST.PORT of ORIG)
                                (fetch TCP.SRC.ADDR of ORIG)
                                (fetch TCP.SRC.PORT of ORIG)))
          (replace TCP.SEQ of SEGMENT with SEQ)
          (if ACK
              then (replace TCP.ACK of SEGMENT with ACK)
                   (OR FLAGS (SETQ FLAGS (LOGOR \TCP.CTRL.RST \TCP.CTRL.ACK)))
            else (replace TCP.ACK of SEGMENT with 0)
                 (OR FLAGS (SETQ FLAGS \TCP.CTRL.RST)))
          (replace TCP.CTRL of SEGMENT with FLAGS)
          (replace TCP.WINDOW of SEGMENT with 0)
          (replace EPREQUEUE of SEGMENT with 'FREE)
          (\TCP.SEND.SEGMENT NIL SEGMENT FLAGS])
```

(\TCP.FIX.OUTGOING.SEGMENT

```
  [LAMBDA (TCB SEGMENT FLAGS)                                           (* ejs%: "18-Dec-86 17:29")
                                                                        (* fill in control bits, ACK and window information, and start
                                                                        round trip timer)
    (if (BITTEST FLAGS \TCP.CTRL.ACK)
        then (replace TCP.ACK of SEGMENT with (fetch TCB.RCV.NXT of TCB))
      else (replace TCP.ACK of SEGMENT with 0))
    (replace TCP.CTRL of SEGMENT with FLAGS)                            (* set control bits)
    (replace TCP.WINDOW of SEGMENT with (replace TCB.LAST.SENT.RCV.WND of TCB with (fetch TCB.RCV.WND of TCB)))
    (if (NULL (fetch TCB.RTFLG of TCB))
        then                                                            (* time round trip response to this segment)
             (replace TCB.RTFLG of TCB with T)
             (replace TCB.RTSEQ of TCB with (fetch TCP.SEQ of SEGMENT))
             (replace TCB.RTTIMER of TCB with (SETUPTIMER 0 (fetch TCB.RTTIMER of TCB])
```

(\TCP.SEND.DATA

```
  [LAMBDA (TCB SEGMENT LENGTH FLAGS)                                    (* wjy "13-Dec-85 14:30")

        (* * This function is used to send a TCP data segment for the first time.
        Subsequent retransmissions are done directly through \TCP.SEND.SEGMENT)

        (* * NOTE%: This function MUST be called with the TCB.LOCK already locked!)

    (PROG (SEQ TOP)
          (CHECK (OR (EQ LENGTH (\TCP.DATA.LENGTH SEGMENT))
                     (SHOULDNT "bad segment length")))
          (CHECK (OR (ILEQ LENGTH (fetch TCB.MAXSEG of TCB))
                     (SHOULDNT "segment > max segment size")))
          (if (NEQ (fetch TCB.STATE of TCB)
                   'SYN.SENT)
              then                                                      (* ACK in all synchronized states)
                   (SETQ FLAGS (LOGOR FLAGS \TCP.CTRL.ACK)))
          (SETQ SEQ (fetch TCB.SND.NXT of TCB))                         (* assign sequence number)
          (if (fetch TCB.ACKFLG of TCB)
              then (SETQ FLAGS (LOGOR FLAGS \TCP.CTRL.ACK)))
          (SETQ TOP (IPLUS SEQ LENGTH (\TCP.SYN.OR.FIN FLAGS)))
          (CHECK (OR (\32BIT.GEQ TOP (fetch TCB.SND.NXT of TCB))
                     (SHOULDNT "bad sequence numbers")))
          (replace TCP.SEQ of SEGMENT with SEQ)
          (if (BITTEST FLAGS \TCP.CTRL.URG)
              then (replace TCB.SND.UP of TCB with TOP))
          (if (\32BIT.GT (fetch TCB.SND.UP of TCB)
```

```
                SEQ)
        then                                                    (* there's urgent data to send)
             (SETQ FLAGS (LOGOR FLAGS \TCP.CTRL.URG))
             (replace TCP.URG.PTR of SEGMENT with (IDIFFERENCE (fetch TCB.SND.UP of TCB)
                                                               SEQ))
          else                                                  (* no urgent data)
                                                                (* drag the urgent pointer along at the left edge of the window)
               (replace TCB.SND.UP of TCB with (fetch TCB.SND.UNA of TCB)))
        (if (BITTEST FLAGS \TCP.CTRL.FIN)
            then

        (* remember the sequence number of the FIN so we can tell when it's been acked)

             (CHECK (OR (EQ (fetch TCB.STATE of TCB)
                            'FIN.WAIT.1)
                        (EQ (fetch TCB.STATE of TCB)
                            'LAST.ACK)
                        (SHOULDNT "bad state for FIN")))
             (replace TCB.FINSEQ of TCB with TOP))
        (replace TCB.SND.NXT of TCB with TOP)
        (do                                                     (* try to send segment)
            (SELECTQ (fetch TCB.STATE of TCB)
                 (LISTEN (ERROR "TCP connection not established"))
                 ((SYN.SENT SYN.RECEIVED ESTABLISHED FIN.WAIT.1 CLOSE.WAIT LAST.ACK)
                      (if (OR (ZEROP LENGTH)
                              (ZEROP (fetch TCB.SND.WL1 of TCB))
                              (\32BIT.LEQ TOP (IPLUS (fetch TCB.SND.UNA of TCB)
                                                     (fetch TCB.SND.WND of TCB)))
                              (\32BIT.GT (fetch TCB.SND.UP of TCB)
                                     (fetch TCB.SND.UNA of TCB)))
                          then                                  (* go ahead and send it)
                               [CHECK (OR (ZEROP LENGTH)
                                          (ZEROP (fetch TCB.SND.WL1 of TCB))
                                          (\32BIT.LEQ TOP (IPLUS (fetch TCB.SND.UNA of TCB)
                                                                 (fetch TCB.SND.WND of TCB]
                               (replace TCB.OUTPUT.HELD of TCB with NIL)
                                                                (* advance SND.NXT)
                               (\TCP.FIX.OUTGOING.SEGMENT TCB SEGMENT FLAGS)
                               (replace EPREQUEUE of SEGMENT with (fetch TCB.REXMT.QUEUE of TCB))
                               (replace EPUSERFIELD of SEGMENT with (CLOCK0 (CREATECELL \FIXP)))
                               (\TCP.SEND.SEGMENT TCB SEGMENT FLAGS)
                               (RETURN)
                            else                                (* block until we can send it)
                                 (replace TCB.OUTPUT.HELD of TCB with T)
                                 (MONITOR.AWAIT.EVENT (fetch TCB.LOCK of TCB)
                                        (fetch TCB.SND.EVENT of TCB))))
                 ((FIN.WAIT.2 CLOSING TIME.WAIT)
                      (ERROR "TCP connection closing"))
                 (CLOSED (ERROR "TCP connection closed"))
                 (SHOULDNT])
```

(\TCP.SEND.SEGMENT

```
  [LAMBDA (TCB SEGMENT FLAGS)                                   (* ejs%: "28-Dec-84 18:06")
                                                                (* common routine to transmit a TCP segment)
    (\TCP.CHECKSUM.OUTGOING TCB SEGMENT)
    (\TCP.TRACE.SEGMENT 'SEND SEGMENT)
    (if TCB
        then (if (BITTEST FLAGS \TCP.CTRL.ACK)
                 then (replace TCB.ACKFLG of TCB with NIL))
             (\TCP.START.PROBE.TIMER TCB))
    (\IP.TRANSMIT SEGMENT])
```

(\TCP.NEW.TEMPLATE

```
  [LAMBDA (TCB)                                                 (* ejs%: "29-Dec-84 13:05")
    (replace TCB.TEMPLATE of TCB with NIL)
    (\TCP.TEMPLATE TCB])
```

(\TCP.START.PROBE.TIMER

```
  [LAMBDA (TCB)                                                 (* ejs%: "12-Aug-86 10:35")
    (replace TCB.PROBE.TIMER of TCB with (COND
                                         ((AND (fetch TCB.NO.IDLE.PROBING of TCB)
                                               (EQ (fetch TCB.STATE of TCB)
                                                   'ESTABLISHED))
                                          NIL)
                                         (T [COND
                                               ((NULL (fetch TCB.PROBE.TIMER of TCB))
                                                (LET ((IPSOCKET (fetch TCB.IPSOCKET of TCB)))
                                                     (COND
                                                        (IPSOCKET (NOTIFY.EVENT (fetch (IPSOCKET IPSEVENT)
                                                                                   of IPSOCKET]
                                            (SETUPTIMER (ITIMES 4 (fetch TCB.RTO of TCB))
                                                   (fetch TCB.PROBE.TIMER of TCB])
```

(\TCP.RETRANSMIT

```
[LAMBDA (TCB)                                                    (* ejs%: " 3-Jun-85 07:58")
                                                                 (* find the first unacknowledged segment and retransmit it)
  (PROG ((QUEUE (fetch TCB.REXMT.QUEUE of TCB))
         (UNA (fetch TCB.SND.UNA of TCB))
         CURRENT CURSEQ NEXT PREV REST FIRSTSEG MINSEQ FLAGS)
        (UNINTERRUPTABLY

        (* detach the list of segments to be retransmitted so we don't interfere with the driver)

            (SETQ NEXT (fetch SYSQUEUEHEAD of QUEUE))
            (replace SYSQUEUEHEAD of QUEUE with NIL)
            (replace SYSQUEUETAIL of QUEUE with NIL))
        (while (SETQ CURRENT NEXT) do (SETQ NEXT (fetch QLINK of CURRENT))
                                      (replace QLINK of CURRENT with NIL)
                                      (if (\32BIT.LEQ (IPLUS (SETQ CURSEQ (fetch TCP.SEQ of CURRENT))
                                                             (\TCP.DATA.LENGTH CURRENT)
                                                             (\TCP.SYN.OR.FIN (fetch TCP.CTRL of CURRENT)))
                                                      UNA)
                                          then                   (* this segment has already been acked)
                                               (\TCP.RELEASE.SEGMENT CURRENT)
                                        elseif (NULL FIRSTSEG)
                                          then                   (* this is the first unacked segment we've encountered)
                                               (SETQ FIRSTSEG CURRENT)
                                               (SETQ MINSEQ CURSEQ)
                                        elseif (\32BIT.LT CURSEQ MINSEQ)
                                          then

        (* this is the lowest sequence number seen so so far; put the previous contender back on the REST queue)

                                               (replace QLINK of FIRSTSEG with REST)
                                               (SETQ REST FIRSTSEG)
                                               (SETQ FIRSTSEG CURRENT)
                                               (SETQ MINSEQ CURSEQ)
                                      else

        (* this is an unacked segment but later than one we've already seen;
        just add it to the REST queue)

                                           (replace QLINK of CURRENT with REST)
                                           (SETQ REST CURRENT)))
        (UNINTERRUPTABLY                                         (* set the retransmit queue to be the REST queue we've
                                                                 accumulated)
            (if (SETQ CURRENT REST)
                then                                             (* find tail of REST queue)
                     (while (SETQ NEXT (fetch QLINK of CURRENT)) do (SETQ CURRENT NEXT)))
            (replace SYSQUEUEHEAD of QUEUE with REST)
            (replace SYSQUEUETAIL of QUEUE with CURRENT))
        (if FIRSTSEG
            then (if (IGEQ (CLOCKDIFFERENCE (fetch EPUSERFIELD of FIRSTSEG))
                           (fetch TCB.RTO of TCB))
                     then (SETQ FLAGS (fetch TCP.CTRL of FIRSTSEG))
                          (\TCP.FIX.OUTGOING.SEGMENT TCB FIRSTSEG FLAGS)
                          (replace EPREQUEUE of FIRSTSEG with (fetch TCB.REXMT.QUEUE of TCB))
                          (CLOCK0 (fetch EPUSERFIELD of FIRSTSEG))
                          (\TCP.SEND.SEGMENT TCB FIRSTSEG FLAGS)
                          (RETURN T)
                   else (\ENQUEUE (fetch TCB.REXMT.QUEUE of TCB)
                                  FIRSTSEG)
                        (RETURN NIL))
          else (RETURN NIL])
```

(\TCP.START.TIME.WAIT

```
[LAMBDA (TCB)                                                    (* ecc "16-Apr-84 17:58")
                                                                 (* start 2MSL timer)
  (replace TCB.2MSL.TIMER of TCB with (SETUPTIMER (ITIMES 2 \TCP.MSL)
                                                  (fetch TCB.2MSL.TIMER of TCB)))
  (\TCP.TRACE.TRANSITION TCB 'TIME.WAIT)
  (replace TCB.STATE of TCB with 'TIME.WAIT])
```

(\TCP.CONNECTION.DROPPED

```
[LAMBDA (TCB MSG)                                                (* ejs%: "29-Jan-85 16:06")
  (if TCPTRACEFLG
      then (printout TCPTRACEFILE .TAB0 0 "[TCP connection " (OR MSG "dropped")
                 "]" T))
  (\TCP.TRACE.TRANSITION TCB 'CLOSED)
  (replace TCB.STATE of TCB with 'CLOSED)
  (AND (OPENP (fetch TCB.RCV.STREAM of TCB)
              'INPUT)
       (CLOSEF (fetch TCB.RCV.STREAM of TCB)))
  (AND (OPENP (fetch TCB.SND.STREAM of TCB)
              'OUTPUT)
       (CLOSEF (fetch TCB.SND.STREAM of TCB)))
  (NOTIFY.EVENT (fetch TCB.RCV.EVENT of TCB])
```

(\TCP.CHECK.OPTIONS

```
  [LAMBDA (TCB SEGMENT FLAGS)                                    (* ejs%: "21-Mar-86 20:04")

          (* * Do TCP header options processing)

    (COND
       ((IGREATERP (fetch (TCPSEGMENT TCP.DATA.OFFSET) of SEGMENT)
              \TCP.MIN.DATA.OFFSET)
        (\TCP.PROCESS.OPTIONS TCB SEGMENT FLAGS))
       (T T])
```

(\TCP.PROCESS.OPTIONS

```
  [LAMBDA (TCB SEGMENT FLAGS)                                    (* ejs%: "20-Jun-85 16:08")

          (* * Process the options in a TCP header)

    (bind (OPTIONBASE _ (fetch (TCPSEGMENT TCP.OPTIONS) of SEGMENT))
          (OPTIONOFFSET _ 0)
          OPTION eachtime (SETQ OPTION (\GETBASEBYTE OPTIONBASE OPTIONOFFSET)) until (EQ OPTION \TCPOPT.END)
       do (SELECTC OPTION
              (\TCPOPT.END (HELP "Unexpected \TCPOPT.END processing TCP options"))
              (\TCPOPT.NOP (add OPTIONOFFSET 1))
              (\TCPOPT.MAXSEG
                   [COND
                      ((BITTEST FLAGS \TCP.CTRL.SYN)
                       (replace TCB.MAXSEG of TCB with (IMIN \TCP.DEFAULT.MAXSEG
                                                             (LOGOR (LLSH (\GETBASEBYTE OPTIONBASE
                                                                                        (IPLUS OPTIONOFFSET 2))
                                                                          BITSPERBYTE)
                                                                    (\GETBASEBYTE OPTIONBASE (IPLUS OPTIONOFFSET 3
                                                                                                    ]
                   (add OPTIONOFFSET (\GETBASEBYTE OPTIONBASE (ADD1 OPTIONOFFSET))))
              (RETURN)))
    T])

)
```

;; support for ICMP messages that affect TCP connections

```
(DECLARE%: EVAL@COMPILE DONTCOPY

(DECLARE%: EVAL@COMPILE

(RPAQQ \ICMP.PROTOCOL 1)

(CONSTANTS \ICMP.PROTOCOL)
)

(DECLARE%: EVAL@COMPILE

(RPAQQ \ICMP.32BIT.WORDS 2)

(CONSTANTS \ICMP.32BIT.WORDS)
)

(DECLARE%: EVAL@COMPILE

(RPAQQ \ICMP.DESTINATION.UNREACHABLE 3)

(RPAQQ \ICMP.SOURCE.QUENCH 4)

(CONSTANTS \ICMP.DESTINATION.UNREACHABLE \ICMP.SOURCE.QUENCH)
)
)

(DEFINEQ
```

(\TCP.HANDLE.ICMP

```
  [LAMBDA (ICMP SEGMENT)                                         (* ejs%: " 3-Jun-85 07:41")
                                                                 (* handle ICMP messages)
    (PROG (MSG TCB)
          (if (NEQ (fetch (ICMP ICMPTYPE) of ICMP)
                   \ICMP.DESTINATION.UNREACHABLE)
              then (RETURN))
          (SETQ MSG (SELECTQ (fetch (ICMP ICMPCODE) of ICMP)
                             (0 "net unreachable")
                             (1 "host unreachable")
                             (2 "protocol unreachable")
                             (3 "port unreachable")
                             (4 "fragmentation needed and DF set")
                             (5 "source route failed")
                             "destination unreachable (unknown code)"))
          (SETQ TCB (\TCP.LOOKUP.TCB (fetch TCP.DST.ADDR of SEGMENT)
                                     (fetch TCP.DST.PORT of SEGMENT)
                                     (fetch TCP.SRC.PORT of SEGMENT)))
```

```
          (if (OR (NULL TCB)
                  (EQ (fetch TCB.STATE of TCB)
                      'CLOSED))
              then (RETURN))
          (\RELEASE.ETHERPACKET ICMP)
          (\TCP.CONNECTION.DROPPED TCB MSG])

)

;; TCP stream routines

(DEFINEQ

(TCP.OPEN
  [LAMBDA (DST.HOST DST.PORT SRC.PORT MODE ACCESS NOERRORFLG OPTIONS)
                                                          (* ejs%: "21-Mar-86 17:38")
    (PROG (TCB DST.HOST.NUMBER)
          (SELECTQ ACCESS
              (INPUT)
              (APPEND)
              (OUTPUT (SETQ ACCESS 'APPEND))
              (LISPERROR "ILLEGAL ARG" ACCESS))
          (COND
             [(ATOM DST.HOST)
              (COND
                 ((AND (NOT (SETQ DST.HOST.NUMBER (DODIP.HOSTP DST.HOST)))
                       (EQ MODE 'ACTIVE))
                  (ERROR "Unknown TCP/IP host: " DST.HOST]
             ((FIXP DST.HOST)
              (SETQ DST.HOST.NUMBER DST.HOST))
             (T (ERROR "Illegal TCP/IP host: " DST.HOST)))
          (SETQ TCB (\TCP.CONNECTION DST.HOST.NUMBER DST.PORT SRC.PORT MODE OPTIONS))
          (RETURN (if (NULL TCB)
                      then (if NOERRORFLG
                               then NIL
                             else (ERROR "TCP connection failed"))
                    else (SELECTQ ACCESS
                              (INPUT (fetch TCB.RCV.STREAM of TCB))
                              (APPEND (fetch TCB.SND.STREAM of TCB))
                              (SHOULDNT])

(TCP.OTHER.STREAM
  [LAMBDA (STREAM)                                        (* ecc "14-May-84 16:52")
    (PROG ((TCB (fetch (TCPSTREAM TCB) of STREAM)))
          (if (NOT (type? TCP.CONTROL.BLOCK TCB))
              then (ERROR "no TCP control block"))
          (RETURN (SELECTQ (fetch (TCPSTREAM ACCESS) of STREAM)
                       (INPUT (fetch TCB.SND.STREAM of TCB))
                       (APPEND (fetch TCB.RCV.STREAM of TCB))
                       (SHOULDNT])

(\TCP.BOUTS
  [LAMBDA (STREAM BASE OFF NBYTES)                        (* ejs%: "27-May-86 15:09")
    (PROG ((TCB (fetch (TCPSTREAM TCB) of STREAM)))
          (RETURN (\BUFFERED.BOUTS (fetch (TCP.CONTROL.BLOCK TCB.SND.STREAM) of TCB)
                         BASE OFF NBYTES])

(\TCP.OTHER.BIN
  [LAMBDA (STREAM)                                        (* ejs%: "27-May-86 14:40")
    (\BIN (TCP.OTHER.STREAM STREAM])

(\TCP.OTHER.BOUT
  [LAMBDA (STREAM BYTE)                                   (* ejs%: "27-May-86 14:19")
    (BOUT (TCP.OTHER.STREAM STREAM)
          BYTE])

(\TCP.BIN
  [LAMBDA (STREAM)                                        (* ecc " 3-May-84 13:55")
    (do (if (ILESSP (fetch COFFSET of STREAM)
                    (fetch CBUFSIZE of STREAM))
            then [RETURN (\GETBASEBYTE (fetch CPPTR of STREAM)
                                       (PROG1 (fetch COFFSET of STREAM)
                                              (add (fetch COFFSET of STREAM)
                                                   1))]
          elseif (NULL (\TCP.GET.SEGMENT STREAM))
            then (RETURN (STREAMOP 'ENDOFSTREAMOP STREAM STREAM])

(\TCP.BACKFILEPTR
  [LAMBDA (STREAM)                                        (* ejs%: "15-Sep-85 23:25")
    (COND
       ((AND (fetch CPPTR of STREAM)
```

```
            (IGEQ (fetch COFFSET of STREAM)
                  (fetch (TCPSTREAM ORIGINAL.COFFSET) of STREAM)))
        (add (fetch COFFSET of STREAM)
             -1))
       (T (\IS.NOT.RANDACCESSP STREAM])
```

(\TCP.GETNEXTBUFFER

```
[LAMBDA (STREAM WHATFOR NOERRORFLG)                                    (* ejs%: "27-May-86 14:45")
  (BLOCK)
  (SELECTQ WHATFOR
      (READ [COND
                ((NEQ STREAM (fetch (TCP.CONTROL.BLOCK TCB.RCV.STREAM) of (fetch (TCPSTREAM TCB) of STREAM)))
                 (SETQ STREAM (TCP.OTHER.STREAM STREAM]
            (\TCP.GET.SEGMENT STREAM NOERRORFLG))
      (WRITE [COND
                ((NEQ STREAM (fetch (TCP.CONTROL.BLOCK TCB.SND.STREAM) of (fetch (TCPSTREAM TCB) of STREAM)))
                 (SETQ STREAM (TCP.OTHER.STREAM STREAM]
            (\TCP.FLUSH STREAM)
            (\TCP.FILL.IN.SEGMENT STREAM))
      (SHOULDNT])
```

(\TCP.GET.SEGMENT

```
[LAMBDA (STREAM NOERRORFLG)                                            (* ejs%: "18-Dec-86 17:33")
```

(* * Get the next segment from the input stream. Return T if successful;
otherwise, an error code. Call the user-specified error handler to get a code, if necessary)

```
  (PROG ((TCB (fetch (TCPSTREAM TCB) of STREAM))
         SEGMENT SEQ LEN OLDSEGMENT OLDSEQ OLDLEN OLDTOP SUCCESS OFFSET LAST.BYTE)
        (if [OR (NULL TCB)
                (AND (NEQ (fetch TCB.STATE of TCB)
                          'CLOSED)
                     (NEQ STREAM (fetch TCB.RCV.STREAM of TCB]
            then (ERROR "not TCP input stream"))
        (WITH.MONITOR (fetch TCB.LOCK of TCB)
            (SETQ OLDSEGMENT (fetch TCB.RCV.SEGMENT of TCB))
            (CHECK (OR (NULL OLDSEGMENT)
                       (EQ (fetch TCP.DATA.LENGTH of OLDSEGMENT)
                           (fetch CBUFSIZE of STREAM))
                       (SHOULDNT "inconsistent stream buffer size")))
            (UNINTERRUPTABLY
                [COND
                   ((fetch CPPTR of STREAM)
                    (SETQ LAST.BYTE (\GETBASEBYTE (fetch CPPTR of STREAM)
                                                  (fetch COFFSET of STREAM]
                (replace TCB.RCV.SEGMENT of TCB with NIL)
                (replace CPPTR of STREAM with NIL)
                (replace CBUFSIZE of STREAM with 0)
                (replace COFFSET of STREAM with 0))
            (if OLDSEGMENT
                then                                                   (* remember sequence number range of previous segment so
                                                                       we can adjust for overlap)
                     [SETQ OLDTOP (IPLUS (SETQ OLDSEQ (fetch TCP.SEQ of OLDSEGMENT))
                                         (SETQ OLDLEN (fetch TCP.DATA.LENGTH of OLDSEGMENT]
                     (replace TCB.RCV.WND of TCB with (IMIN \TCP.DEFAULT.RECEIVE.WINDOW
                                                            (IPLUS (fetch TCB.RCV.WND of TCB)
                                                                   OLDLEN)))
                     (add (fetch (TCPSTREAM BYTECOUNT) of STREAM)
                          OLDLEN)
                     (\TCP.RELEASE.SEGMENT OLDSEGMENT)
                     (SETQ OLDSEGMENT T))
```

(* look at first segment in input queue to see if it overlaps the sequence number range we're expecting;
there may be duplicates that must be skipped over)

```
            [do ((CHECK (\TCP.CHECK.INPUT.QUEUE TCB))
                 (COND
                    [(AND (SETQ SEGMENT (\QUEUEHEAD (fetch TCB.INPUT.QUEUE of TCB)))
                          (\32BIT.LT (SETQ SEQ (fetch TCP.SEQ of SEGMENT))
                                     (fetch TCB.RCV.NXT of TCB)))
```

(* this segment is within the range of contiguous sequence numbers received so far, because its sequence number is less
than RCV.NXT)

```
                     (\DEQUEUE (fetch TCB.INPUT.QUEUE of TCB))
                     (SETQ LEN (fetch TCP.DATA.LENGTH of SEGMENT))
                     (COND
                        ((AND OLDSEGMENT (\32BIT.LEQ (IPLUS SEQ LEN)
                                                     OLDTOP))          (* this segment is a duplicate)
                         (\TCP.RELEASE.SEGMENT SEGMENT))
                        (T                                             (* this segment overlaps with the range of sequence numbers
                                                                       we're expecting)
                           (CHECK (OR (NOT OLDSEGMENT)
                                      (\32BIT.LEQ SEQ OLDTOP)
                                      (SHOULDNT "gap in input queue")))
```

```
                            (UNINTERRUPTABLY
                                (replace CPPTR of STREAM with (fetch TCP.CONTENTS of SEGMENT))
                                                                    (* eliminate overlap)
                                [SETQ OFFSET (replace (TCPSTREAM ORIGINAL.COFFSET) of STREAM
                                                  with (replace COFFSET of STREAM
                                                                with (COND
                                                                       (OLDSEGMENT (IDIFFERENCE OLDLEN (IDIFFERENCE
                                                                                                        SEQ OLDSEQ)))
                                                                       (T 0]
                                (COND
                                   (LAST.BYTE (\PUTBASEBYTE (fetch CPPTR of STREAM)
                                                            (SUB1 OFFSET)
                                                           LAST.BYTE)))
                                (add (fetch (TCPSTREAM BYTECOUNT) of STREAM)
                                     (IMINUS OFFSET))
                                (replace CBUFSIZE of STREAM with LEN)
                                (replace TCB.RCV.SEGMENT of TCB with SEGMENT))
                            (SETQ SUCCESS T)
                            (RETURN]
                        (T (SELECTQ (fetch TCB.STATE of TCB)
                                ((LISTEN SYN.SENT SYN.RECEIVED)     (* wait until established)
                                     (MONITOR.AWAIT.EVENT (fetch TCB.LOCK of TCB)
                                             (fetch TCB.ESTABLISHED of TCB)))
                                ((ESTABLISHED FIN.WAIT.1 FIN.WAIT.2)
                                                                    (* wait for next segment)
                                     (MONITOR.AWAIT.EVENT (fetch TCB.LOCK of TCB)
                                             (fetch TCB.RCV.EVENT of TCB))
                                     (SELECTQ (fetch TCB.STATE of TCB)
                                          ((CLOSED CLOSING LAST.ACK)
                                               (RELEASE.MONITORLOCK (fetch TCB.LOCK of TCB))
                                               [COND
                                                  (NOERRORFLG (RETURN NIL))
                                                  (T (RETURN (SETQ SUCCESS (\EOF.ACTION STREAM])
                                          NIL))
                                ((CLOSED CLOSE.WAIT CLOSING LAST.ACK TIME.WAIT)
                                                                    (* return NIL to punt to ENDOFSTREAMOP in \TCP.BIN)
                                     (RELEASE.MONITORLOCK (fetch TCB.LOCK of TCB))
                                     [COND
                                        (NOERRORFLG (RETURN NIL))
                                        (T (RETURN (SETQ SUCCESS (\EOF.ACTION STREAM])
                                (SHOULDNT])
                    (if (fetch TCB.SENT.ZERO of TCB)
                        then (\TCP.SEND.ACK TCB 'NOW)
                             (BLOCK))
                    (RETURN SUCCESS])
```

(\TCP.PEEKBIN

```
  [LAMBDA (STREAM NOERRORFLG)                                       (* ecc " 3-May-84 13:55")
    (do (if (ILESSP (fetch COFFSET of STREAM)
                    (fetch CBUFSIZE of STREAM))
            then (RETURN (\GETBASEBYTE (fetch CPPTR of STREAM)
                                       (fetch COFFSET of STREAM)))
          elseif (NULL (\TCP.GET.SEGMENT STREAM))
            then (RETURN (if NOERRORFLG
                             then NIL
                           else (STREAMOP 'ENDOFSTREAMOP STREAM STREAM])
```

(\TCP.GETFILEPTR

```
  [LAMBDA (STREAM)                                                  (* ejs%: "10-Jun-85 14:07")
    (IPLUS (fetch (STREAM COFFSET) of STREAM)
           (fetch (TCPSTREAM BYTECOUNT) of STREAM])
```

(\TCP.READP

```
  [LAMBDA (STREAM)                                                  (* ejs%: " 7-Jun-85 13:39")
    (PROG ((TCB (fetch (TCPSTREAM TCB) of STREAM)))
          (if [OR (NULL TCB)
                  (AND (NEQ (fetch TCB.STATE of TCB)
                            'CLOSED)
                       (NEQ STREAM (fetch TCB.RCV.STREAM of TCB]
              then (ERROR "not TCP input stream")
            else (RETURN (OR (ILESSP (fetch COFFSET of STREAM)
                                     (fetch CBUFSIZE of STREAM))
                             (AND (\QUEUEHEAD (fetch TCB.INPUT.QUEUE of TCB))
                                  T])
```

(\TCP.EOFP

```
  [LAMBDA (STREAM)                                                  (* ejs%: "13-Apr-85 16:15")

          (* check whether EOF has been reached on stream -- may block waiting for next segment)

    (PROG ((TCB (fetch (TCPSTREAM TCB) of STREAM)))
          (if (NULL TCB)
              then (ERROR "not TCP stream")
```

```
            elseif (AND (NEQ 'CLOSED (fetch TCB.STATE of TCB))
                        (EQ STREAM (fetch TCB.SND.STREAM of TCB)))
              then (RETURN T)                                    (* Always at EOF of outgoing stream.)
            elseif [OR (ILESSP (fetch COFFSET of STREAM)
                               (fetch CBUFSIZE of STREAM))
                       (NOT (NULL (\QUEUEHEAD (fetch TCB.INPUT.QUEUE of TCB]
              then                                               (* there is still data left to read)
                   (RETURN NIL)
            else (RETURN (SELECTQ (fetch TCB.STATE of TCB)
                                  (ESTABLISHED NIL)
                                  ((LISTEN SYN.SENT SYN.RECEIVED FIN.WAIT.1 FIN.WAIT.2)
                                                                 (* can't tell without waiting for next segment)
                                       (NULL (\TCP.GET.SEGMENT STREAM T)))
                                  ((CLOSE.WAIT CLOSING LAST.ACK TIME.WAIT CLOSED)
                                                                 (* no more data can be forthcoming)
                                       T)
                                  (SHOULDNT])
```

(TCP.URGENTP

```
  [LAMBDA (STREAM)                                               (* ecc " 7-May-84 14:27")
                                                                 (* check if current point in receive stream is before receive urgent pointer)
    (PROG ((TCB (fetch (TCPSTREAM TCB) of STREAM)))
          (if (OR (NULL TCB)
                  (NEQ STREAM (fetch TCB.RCV.STREAM of TCB)))
              then (ERROR "not TCP input stream"))
          (RETURN (AND (fetch TCB.RCV.SEGMENT of TCB)
                       (\32BIT.GT (fetch TCB.RCV.UP of TCB)
                                  (IPLUS (fetch TCP.SEQ of (fetch TCB.RCV.SEGMENT of TCB))
                                         (fetch COFFSET of STREAM])
```

(TCP.URGENT.EVENT

```
  [LAMBDA (STREAM)                                               (* edited%: "22-May-84 18:10")
                                                                 (* return the urgent data event so that a user process can wait for it)
    (fetch TCB.URGENT.EVENT of (fetch (TCPSTREAM TCB) of STREAM])
```

(\TCP.BOUT

```
  [LAMBDA (STREAM CHAR)                                          (* ecc " 3-May-84 13:55")
    (do (if (ILESSP (fetch COFFSET of STREAM)
                    (fetch CBUFSIZE of STREAM))
            then (\PUTBASEBYTE (fetch CPPTR of STREAM)
                               (fetch COFFSET of STREAM)
                               CHAR)
                 (add (fetch COFFSET of STREAM)
                      1)
                 (RETURN)
          else (\TCP.FLUSH STREAM)
               (\TCP.FILL.IN.SEGMENT STREAM])
```

(\TCP.FLUSH

```
  [LAMBDA (STREAM FLAGS)                                         ; Edited  4-Dec-87 12:11 by scp
```

(* Force out current output segment. If FLAGS is non-nil, send a segment with those flags even if we have to create a new one)

```
    (PROG ((TCB (fetch TCB of STREAM))
           SEGMENT LENGTH)
          (if [OR (NULL TCB)
                  (AND (NEQ (fetch TCB.STATE of TCB)
                            'CLOSED)
                       (NEQ STREAM (fetch TCB.SND.STREAM of TCB]
              then (ERROR "not TCP output stream"))
          (SETQ LENGTH (fetch COFFSET of STREAM))
          (WITH.FAST.MONITOR (fetch TCB.LOCK of TCB)
                  (if [OR (AND (SETQ SEGMENT (fetch TCB.SND.SEGMENT of TCB))
                               (NOT (ZEROP LENGTH)))
                          (AND FLAGS (SETQ SEGMENT (\TCP.FILL.IN.SEGMENT STREAM
                                                                         (COND
                                                                            [(EQ (fetch TCB.STATE of TCB)
                                                                                 'SYN.SENT)
                                                                             `(MAXSEG %, (OR (fetch TCB.OUR.MAXSEG of TCB)
                                                                                             \TCP.DEFAULT.MAXSEG]
                                                                            ((EQ (fetch TCB.STATE of TCB)
                                                                                 'SYN.RECEIVED)
                                                                             `(MAXSEG %, (OR (fetch TCB.OUR.MAXSEG of TCB)
                                                                                             \TCP.DEFAULT.MAXSEG]
                      then (if (NULL FLAGS)
                               then (SETQ FLAGS 0))
                           (CHECK (OR (NOT (ZEROP LENGTH))
                                      (NOT (ZEROP (\TCP.SYN.OR.FIN FLAGS)))
                                      (SHOULDNT "sending empty segment")))
                           (if (AND (IGREATERP LENGTH 0)
```

```
                    (ILESSP LENGTH (fetch TCB.OUR.MAXSEG of TCB)))
               then                                    (* PSH this segment to make sure it gets through to the remote
                    process)
                    (SETQ FLAGS (LOGOR FLAGS \TCP.CTRL.PSH)))
          (UNINTERRUPTABLY
              (replace TCB.SND.SEGMENT of TCB with NIL)
              (replace CBUFSIZE of STREAM with 0)
              (replace COFFSET of STREAM with 0)
              (replace CPPTR of STREAM with NIL)
              (add (fetch (TCPSTREAM BYTECOUNT) of STREAM)
                   LENGTH))
          (add (fetch (IP IPTOTALLENGTH) of SEGMENT)
               LENGTH)
          (\TCP.SEND.DATA TCB SEGMENT LENGTH FLAGS])
```

(\TCP.FORCEOUTPUT

```
[LAMBDA (STREAM WAITFLG)                                    (* ejs%: "27-May-86 14:36")

        (* just call \TCP.FLUSH with no flags -- to implement WAITFLG we should wait for SND.UNA to overtake the current
        SND.NXT)

   (COND
      [(NEQ STREAM (fetch (TCP.CONTROL.BLOCK TCB.SND.STREAM) of (fetch (TCPSTREAM TCB) of STREAM)))
       (\TCP.FLUSH (fetch (TCP.CONTROL.BLOCK TCB.SND.STREAM) of (fetch (TCPSTREAM TCB) of STREAM]
      (T (\TCP.FLUSH STREAM])
```

(TCP.URGENT.MARK

```
[LAMBDA (STREAM)                                            (* ecc " 7-May-84 14:17")
                                                            (* mark the current point in the output stream as the end of
                                                            urgent data)
   (\TCP.FLUSH STREAM \TCP.CTRL.URG])
```

(\TCP.FILL.IN.SEGMENT

```
[LAMBDA (STREAM OPTIONS)                                    (* ejs%: "22-Jun-85 03:18")

        (* * set up a new segment to be filled by the output stream. OPTIONS, if supplied, is in PLIST format)

   (PROG ((TCB (fetch (TCPSTREAM TCB) of STREAM))
          SEGMENT)
         (SETQ SEGMENT (\TCP.SETUP.SEGMENT (\LOCAL.IP.ADDRESS)
                               (fetch TCB.SRC.PORT of TCB)
                               (fetch TCB.DST.HOST of TCB)
                               (fetch TCB.DST.PORT of TCB)))
         (COND
            (OPTIONS (\TCP.SETUP.SEGMENT.OPTIONS SEGMENT OPTIONS)))
         (UNINTERRUPTABLY
             (replace TCB.SND.SEGMENT of TCB with SEGMENT)
             (replace CPPTR of STREAM with (fetch TCP.CONTENTS of SEGMENT))
             (replace COFFSET of STREAM with 0)
             (replace CBUFSIZE of STREAM with (fetch TCB.MAXSEG of TCB))
             (replace CBUFMAXSIZE of STREAM with (fetch TCB.MAXSEG of TCB)))
         (RETURN SEGMENT])
```

(\TCP.CLOSE

```
[LAMBDA (STREAM)                                            (* ejs%: "29-Jan-85 17:19")
   (PROG ((TCB (fetch (TCPSTREAM TCB) of STREAM)))
         (if [OR (NULL TCB)
                 (FMEMB (fetch TCB.STATE of TCB)
                        '(CLOSED TIME.WAIT]
             then (RETURN))
         (if (NOT (fetch TCB.CLOSEDFLG of TCB))
             then (TCP.CLOSE.SENDER (fetch TCB.SND.STREAM of TCB)))
         (if (EQ STREAM (fetch TCB.RCV.STREAM of TCB))
             then (while (\QUEUEHEAD (fetch TCB.INPUT.QUEUE of TCB)) do
                                                            (* gobble remaining segments from remote end)
                         (\TCP.GET.SEGMENT STREAM])
```

(\TCP.RESETCLOSE

```
[LAMBDA (STREAM)                                            (* ejs%: "27-May-86 11:55")
   (\TCP.CLOSE STREAM])
```

(TCP.CLOSE.SENDER

```
[LAMBDA (STREAM)                                            (* ecc " 7-May-84 13:44")
   (PROG ((TCB (fetch (TCPSTREAM TCB) of STREAM)))
         (if (OR (NULL TCB)
                 (EQ (fetch TCB.STATE of TCB)
                     'CLOSED)
                 (fetch TCB.CLOSEDFLG of TCB))
             then (RETURN))
         (WITH.MONITOR (fetch TCB.LOCK of TCB)
             (replace TCB.CLOSEDFLG of TCB with T)
```

```
            (SELECTQ (fetch TCB.STATE of TCB)
                ((LISTEN SYN.SENT)
                    (\TCP.CONNECTION.DROPPED TCB "closed"))
                ((SYN.RECEIVED ESTABLISHED)
                    (\TCP.TRACE.TRANSITION TCB 'FIN.WAIT.1)
                    (replace TCB.STATE of TCB with 'FIN.WAIT.1)
                    (\TCP.FLUSH STREAM \TCP.CTRL.FIN))
                (CLOSE.WAIT (\TCP.TRACE.TRANSITION TCB 'LAST.ACK)
                            (replace TCB.STATE of TCB with 'LAST.ACK)
```

(* There is an inconsistency in the spec about this transition%: the description of the CLOSE operation says to go to the CLOSING state, while the diagram shows a transition to the LAST.ACK state.
Since the LAST.ACK state avoids the 2MSL wait, we use it.)

```
                            (\TCP.FLUSH STREAM \TCP.CTRL.FIN))
                NIL)
            (while (NOT (OR (EQ (fetch TCB.STATE of TCB)
                                'CLOSED)
                            (\TCP.OUR.FIN.IS.ACKED TCB)))
              do (MONITOR.AWAIT.EVENT (fetch TCB.LOCK of TCB)
                                      (fetch TCB.FINACKED.EVENT of TCB)))])
```

(TCP.DESTADDRESS

```
  [LAMBDA (STREAM)                                                    (* ejs%: "27-May-86 11:53")
    (\IP.ADDRESS.TO.STRING (fetch (TCP.CONTROL.BLOCK TCB.DST.HOST) of (fetch (TCPSTREAM TCB) of STREAM])
```

(TCP.STOP

```
  [LAMBDA NIL                                                         (* ejs%: "28-Dec-84 18:02")
    (MAPC \TCP.CONTROL.BLOCKS (FUNCTION \TCP.DELETE.TCB))
    (SETQ \TCP.CONTROL.BLOCKS NIL)
    (\IP.DELETE.PROTOCOL \TCP.PROTOCOL])

)
```

;; well-known ports for network standard functions

```
(RPAQQ \TCP.ASSIGNED.PORTS (\TCP.ECHO.PORT \TCP.SINK.PORT \TCP.SYSTAT.PORT \TCP.DAYTIME.PORT \TCP.NETSTAT.PORT
                                \TCP.FAUCET.PORT \TCP.FTP.PORT \TCP.TELNET.PORT \TCP.SMTP.PORT \TCP.TIME.PORT
                                \TCP.NAME.PORT \TCP.WHOIS.PORT \TCP.NAMESERVER.PORT \TCP.FINGER.PORT
                                \TCP.TTYLINK.PORT \TCP.SUPDUP.PORT \TCP.HOSTNAMES.PORT \TCP.UNIXEXEC.PORT
                                \TCP.UNIXLOGIN.PORT \TCP.UNIXSHELL.PORT))

(DECLARE%: EVAL@COMPILE

(RPAQQ \TCP.ECHO.PORT 7)

(RPAQQ \TCP.SINK.PORT 9)

(RPAQQ \TCP.SYSTAT.PORT 11)

(RPAQQ \TCP.DAYTIME.PORT 13)

(RPAQQ \TCP.NETSTAT.PORT 15)

(RPAQQ \TCP.FAUCET.PORT 19)

(RPAQQ \TCP.FTP.PORT 21)

(RPAQQ \TCP.TELNET.PORT 23)

(RPAQQ \TCP.SMTP.PORT 25)

(RPAQQ \TCP.TIME.PORT 37)

(RPAQQ \TCP.NAME.PORT 42)

(RPAQQ \TCP.WHOIS.PORT 43)

(RPAQQ \TCP.NAMESERVER.PORT 53)

(RPAQQ \TCP.FINGER.PORT 79)

(RPAQQ \TCP.TTYLINK.PORT 87)

(RPAQQ \TCP.SUPDUP.PORT 95)

(RPAQQ \TCP.HOSTNAMES.PORT 101)

(RPAQQ \TCP.UNIXEXEC.PORT 512)

(RPAQQ \TCP.UNIXLOGIN.PORT 513)

(RPAQQ \TCP.UNIXSHELL.PORT 514)

(CONSTANTS \TCP.ECHO.PORT \TCP.SINK.PORT \TCP.SYSTAT.PORT \TCP.DAYTIME.PORT \TCP.NETSTAT.PORT \TCP.FAUCET.PORT
```

```
       \TCP.FTP.PORT \TCP.TELNET.PORT \TCP.SMTP.PORT \TCP.TIME.PORT \TCP.NAME.PORT \TCP.WHOIS.PORT
       \TCP.NAMESERVER.PORT \TCP.FINGER.PORT \TCP.TTYLINK.PORT \TCP.SUPDUP.PORT \TCP.HOSTNAMES.PORT
       \TCP.UNIXEXEC.PORT \TCP.UNIXLOGIN.PORT \TCP.UNIXSHELL.PORT)
)
```

;; Stub for debugging

```
(RPAQ? \TCP.DEBUGGABLE )

(RPAQ? TCPTRACEFLG )

(DECLARE%: DOEVAL@COMPILE DONTCOPY

(GLOBALVARS \TCP.DEBUGGABLE TCPTRACEFLG)
)

(DEFINEQ
```

(PPTCB

```
  [LAMBDA (TCB FILE)                                          (* ejs%: " 5-Feb-85 16:47")
    (DECLARE (GLOBALVARS \TCP.DEBUGGABLE))
    (COND
       (\TCP.DEBUGGABLE (printout FILE "TCP connection from " %# (\IP.PRINT.ADDRESS (\LOCAL.IP.ADDRESS)
                                                                                FILE)
                                ":"
                                (fetch TCB.SRC.PORT of TCB)
                                " to " %# (\IP.PRINT.ADDRESS (fetch TCB.DST.HOST of TCB)
                                                  FILE)
                                ":"
                                (fetch TCB.DST.PORT of TCB)
                                " "
                                (fetch TCB.STATE of TCB)
                                T)
              (printout FILE " iss " (fetch TCB.ISS of TCB)
                     " window "
                     (fetch TCB.SND.UNA of TCB)
                     ".."
                     (IPLUS (fetch TCB.SND.UNA of TCB)
                            (fetch TCB.SND.WND of TCB))
                     " next "
                     (fetch TCB.SND.NXT of TCB))
              (if (fetch TCB.FINSEQ of TCB)
                  then (printout FILE " fin " (fetch TCB.FINSEQ of TCB)))
              (printout FILE " rto " (fetch TCB.RTO of TCB)
                     T)
              (printout FILE " irs " (fetch TCB.IRS of TCB)
                     " next "
                     (fetch TCB.RCV.NXT of TCB)
                     " window "
                     (fetch TCB.RCV.NXT of TCB)
                     ".."
                     (IPLUS (fetch TCB.RCV.NXT of TCB)
                            (fetch TCB.RCV.WND of TCB))
                     T)
              (\TCP.PRINT.SEGMENT.QUEUE "retransmit queue" (fetch TCB.REXMT.QUEUE of TCB)
                     FILE)
              (\TCP.PRINT.SEGMENT.QUEUE "input queue" (fetch TCB.INPUT.QUEUE of TCB)
                     FILE])
```

(\TCP.TRACE.SEGMENT

```
  [LAMBDA (CALLER SEGMENT)                                    (* ejs%: " 5-Feb-85 16:50")
    (DECLARE (GLOBALVARS \TCP.DEBUGGABLE TCPTRACEFLG))
    (if (AND \TCP.DEBUGGABLE (MEMB CALLER TCPTRACEFLG))
        then (printout TCPTRACEFILE .TAB0 0 %# (\TCP.PRINT.ELAPSED.TIME TCPTRACEFILE)
                    CALLER ": " %# (TCP.PRINT.SEGMENT SEGMENT TCPTRACEFILE NIL (MEMB 'CONTENTS TCPTRACEFLG])
```

(\TCP.TRACE.TRANSITION

```
  [LAMBDA (TCB NEWSTATE)                                      (* ejs%: " 5-Feb-85 16:51")
    (DECLARE (GLOBALVARS \TCP.DEBUGGABLE))
    (if (AND \TCP.DEBUGGABLE (MEMB 'TRANSITION TCPTRACEFLG)
             (NEQ (fetch TCB.STATE of TCB)
                  NEWSTATE))
        then (printout TCPTRACEFILE .TAB0 0 %# (\TCP.PRINT.ELAPSED.TIME TCPTRACEFILE)
                    (fetch TCB.SRC.PORT of TCB)
                    "/"
                    (fetch TCB.DST.PORT of TCB)
                    ": "
                    (fetch TCB.STATE of TCB)
                    " ---> " NEWSTATE])

)
```

;; TCP initialization

```
(DEFINEQ

(\TCP.INIT
  [LAMBDA NIL                                                    ; Edited 11-Aug-88 14:32 by atm
    (COND
       ((NULL \TCP.DEVICE)
        (SETQ \TCP.DEVICE (create FDEV
                                  FDBINABLE _ T
                                  FDBOUTABLE _ T
                                  BUFFERED _ T
                                  CLOSEFILE _ (FUNCTION \TCP.CLOSE)
                                  REGISTERFILE _ (FUNCTION \ADD-OPEN-STREAM)
                                  UNREGISTERFILE _ (FUNCTION \GENERIC-UNREGISTER-STREAM)
                                  OPENP _ (FUNCTION \GENERIC.OPENP)
                                  BIN _ (FUNCTION \BUFFERED.BIN)
                                  BOUT _ (FUNCTION \BUFFERED.BOUT)
                                  BLOCKIN _ (FUNCTION \BUFFERED.BINS)
                                  BLOCKOUT _ (FUNCTION \TCP.BOUTS)
                                  PEEKBIN _ (FUNCTION \BUFFERED.PEEKBIN)
                                  READP _ (FUNCTION \TCP.READP)
                                  FORCEOUTPUT _ (FUNCTION \TCP.FORCEOUTPUT)
                                  GETNEXTBUFFER _ (FUNCTION \TCP.GETNEXTBUFFER)
                                  BACKFILEPTR _ (FUNCTION \TCP.BACKFILEPTR)
                                  GETFILEPTR _ (FUNCTION \TCP.GETFILEPTR)
                                  EOFP _ (FUNCTION \TCP.EOFP)
                                  DEVICENAME _ 'TCP
                                  EVENTFN _ (FUNCTION NILL)))
        (\DEFINEDEVICE 'TCP \TCP.DEVICE)))
    (SETQ \TCP.LOCK (CREATE.MONITORLOCK))
    [COND
       ((NULL \TCP.PSEUDOHEADER)
        (SETQ \TCP.PSEUDOHEADER (create TCP.PSEUDOHEADER]
    (OR \IPFLG (\IPINIT))
    (\IP.ADD.PROTOCOL \TCP.PROTOCOL (FUNCTION \TCP.PORTCOMPARE)
           (FUNCTION \TCP.NOSOCKETFN)
           NIL
           (FUNCTION \TCP.HANDLE.ICMP))
    (SETQ \TCP.MASTER.SOCKET (\IP.FIND.PROTOCOL \TCP.PROTOCOL])

)

(\TCP.INIT)

(PUTPROPS TCP COPYRIGHT ("Xerox Corporation" 1983 1984 1985 1986 1901 1900 1987 1988 1989 1990))
```

FUNCTION INDEX

CONSTANT INDEX

VARIABLE INDEX

MACRO INDEX

RECORD INDEX