TCP-IP

The Transport Control Protocol - Internet Protocol (TCP-IP) family of networking protocols was developed under the auspices of the Department of Defense to standardize communication mechanisms within Department of Defense networks such as the ARPANET.

The protocols are documented in a collection of working papers known as Requests for Comments (RFCs). Appropriate RFC numbers appear throughout this document as new protocols are introduced.

Requirements

TCP-IP has both hardware and software requirements.

Hardware

- Ethernet
- Cooperating host (yours or theirs)
- 110X/118X with an Ethernet controller (usually co-resident on an otherwise inhabited module)
- XCVR interface cable
- XCVR installed on an Ethernet with a logical (direct or internet) connection to the cooperating host.

Software

You need the files enumerated in the section titled "Interlisp Files." Files loaded by the high-level modules TCPFTP, TCPFTPSRV, TCPCHAT, and TCPTFTP automatically load their dependencies. If you load files from floppy, you must load their dependencies first:

File	Dependencies
TCP	TCPLLIP
TCPCHAT	TCP, CHAT
TCPCONFIG	None
TCPDEBUG	TCP
TCPDOMAIN	TCPUDP
TCPFTP	TCPNAMES, TCP
TCPFTPSRV	TCPFTP
TCPHTE	None
TCPLLAR	None
TCPLLICMP	None
TCPLLIP	TCPHTE, TCPLLICMP, TCPLLAR
TCPNAMES	None
TCPTFTP	TCPUDP
TCPUDP	TCPLLIP

User Interface

TCP does not have a user interface module of its own. Its functions and variables are accessible via an Interlisp Executive, and you can direct some of its debugging information to a window.

As a network protocol module, it extends capability to other programs which may have their own window interfaces, for example, Chat and FileBrowser.

Installation

The first step in installing TCP-IP is to add your workstation to a network supporting TCP-IP and communications with others on the net. The rest of this section contains a step-by-step set of directions for this installation.

After you are on the network, load the required .LCOM modules for the type of service you want. For a full description of these modules, see the Interlisp Files section.

Module	Implementation
TCPFTP	TCP-based file transfer protocol
TCPFTPSRV	TCP-based FTP server
TCPCHAT	TELNET protocol for the Chat system
TCPTFTP	TFTP protocol

Obtaining Network Addresses

The first thing you need to do is to get a TCP-IP address assigned to each of your workstations from your network administrator. If your site supports Domains, get the name of your local domain and the addresses of your domain server(s) from your network administrator. You will also need to know the network addresses and operating system of the hosts you want to communicate with and the addresses of any network gateways you have.

Note: The maximum length of the domain and organization fields is 20 characters each.

Be sure to find out whether your net is a true Class A, B or C network and is not broken up into subnets. If it is broken up into subnets, be sure to read the discussion on SUBNETMASKs in the Primer on IP Networks section.

Differences between TCP and the Medley File Systems

When running TCP-IP to a Sun from an 11xx, directory enumeration on an unmatched directory path returns a listing for the top-level directory of the logged-in user. The TCPFTP protocol does not support directory creation.

Warning: When running TCP-IP to a Sun, a file which is still open on the Sun can be deleted.

Creating HOST.TXT File

Create a HOSTS.TXT file containing entries for the TCP-IP hosts needed by the user community and place a copy of the file on either a directory contained in the DIRECTORIES search path of each workstation on the net or the local disk of each Interlisp workstation.

The following is a sample HOSTS.TXT file:

```
; Hosts.txt,
; Internet Hosts Table for Networks 192.20.10.0 and 174.23.0.0
; 12-Dec-86
;
; The format of this file is documented in RFC 810, "DoD Internet
; Host Table Specification", which is available online at SRI-NIC
; as the file
;              [SRI-NIC]<RFC>RFC952.TXT
;
; It may be retrieved via FTP using username ANONYMOUS with
; any password.
;
; or as the file
;                                          [INDIGO]<RFC>RFC952.TXT
; Read access to GV World. Valid GV credentials required.
;
; The format for entries is:
;
; GATEWAY: ADDR, ADDR : NAME : CPUTYPE : OPSYS : PROTOCOLS :
; HOST: ADDR, ALTERNATE-ADDR (if any): HOSTNAME,NICKNAME : CPUTYPE :
;   OPSYS : PROTOCOLS :
;
; Where:
;;  ADDR = internet address in decimal, e.g., 26.0.0.73
;;  CPUTYPE = machine type (Xerox-11xx, VAX-11/780, SUN, etc.)
;;  OPSYS = operating system (UNIX, TOPS20, TENEX, VMS, Interlisp, etc.)
;;  PROTOCOLS = transport/service (TCP/TELNET, TCP/FTP, etc.)
;;  : (colon) = field delimiter
;;  :: (2 colons, NO space between) = null field
;
HOST : 192.20.10.1 : Bach : Xerox-1108 : Interlisp : TCP/TELNET, TCP/FTP
:
HOST : 192.20.10.3 : PARC-VAXC : VAX-11/780 : UNIX : TCP/TELNET, TCP/FTP
:
HOST : 192.20.10.15 : Oberon : VAX-11/780 : VMS : TCP/TELNET, TCP/FTP :
HOST : 192.20.10.71 : Explorer : TI-EXPLORER : TOPS-20 : TCP/TELNET,
TCP/FTP :
HOST : 174.23.77.22 : Sunrise : SUN : UNIX : TCP/TELNET, TCP/FTP :
HOST : 174.23.30.21 : Rutgers : VAX-11/780 : TOPS-20 : TCP/TELNET,
TCP/FTP :
HOST : 174.23.76.21 : Simba : SYMBOLICS : SYMBOLICS-3600 : TCP/TELNET,
TFP/FTP :
GATEWAY : 192.20.10.240, 174.23.77.250 : Hellsgate : VMS : IP/GW :
```

This example shows a host table that indicates that there are four hosts (Bach, PARC-VAXC, Oberon, and Explorer) on net 192.20.10.0, three hosts (Sunrise, Rutgers, and Simba) on net 174.23.0.0 and a gateway (Hellsgate) that connects the two.

In regard to the OPSYS field in the HOSTS.TXT file, it is preferable to use values recognized by the Lisp variable NETWORKOSTYPES. Interlisp is the default value if a host's OSType is not declared.

Note that if any host is accessible via another network protocol (for example, PUP or NS), you may desire to call the host by an unambiguous name when it is accessed via TCP. You can do this by giving it an unambiguous name in the HOSTS.TXT file.

If you ever modify the HOSTS.TXT table after TCP.LCOM has been loaded, use the function (\HTE.READ.FILE 'HOSTTABLE) to reread the file.

For example,

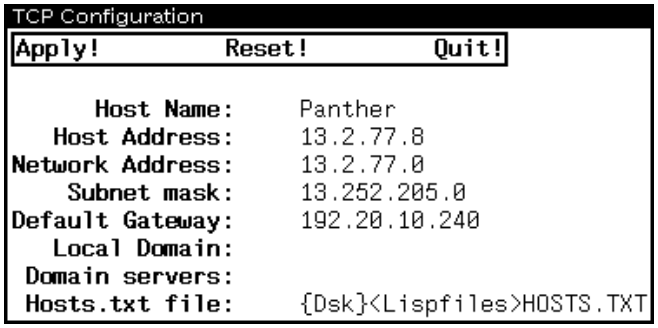
```
(\HTE.READ.FILE '{DSK}<LISPFILES>HOSTS.TXT)
```

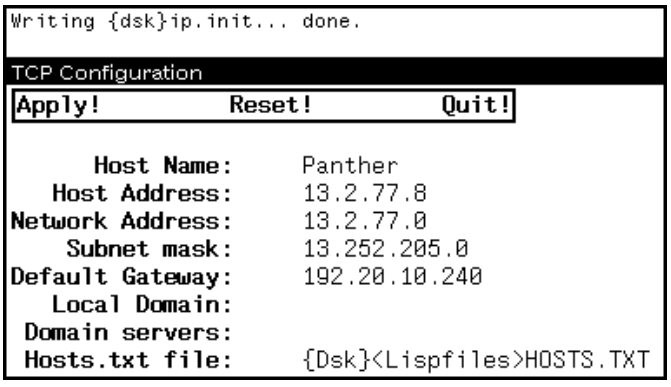
TCP.ALWAYS.READ.HOSTS.FILE [Variable]

Initially set to T. Setting it to NIL causes the system to parse the HOSTS.TXT file only when the filename (stored in the configuration file) is different from the previously read filename, or the write date of the file has changed. The HOSTS.TXT file will always be read at least once when loading the software into a clean sysout.

Creating the Local IP.INIT File

TCP.CONFIGURE brings up a menu that you complete.





If any field does not apply to your site, leave it blank.

Selecting **Apply!** writes the file {DSK}<LISPFILES>IP.INIT to the local disk.

Note: The file {DSK}<LISPFILES>IP.INIT must exist on each Interlisp machine before TCP.LCOM is loaded. And this file *must* remain on the workstation and must not be copied to other workstations. Also, the font GACHA 12 MRR must be available.

Selecting **Reset!** resets the menu to the original state.

Selecting **Quit!** closes the window.

You must perform the TCP.CONFIGURE step individually on each workstation, but you need to perform it only once. As long as there is an IP.INIT file on the workstation, the TCP-IP module will be configured automatically whenever it is loaded or initialized.

If you change your IP.INIT file while TCP-IP is running, you will be prompted to confirm **Restarting TCP**. In most cases, you should confirm the restart.

Adding Host and Operating System Names to NETWORKOSTYPES

The variable NETWORKOSTYPES is used during Chat to determine the sequence of characters to send when performing auto-login. There should be an entry in NETWORKOSTYPES for each TCP host that you want to communicate with in the form (TCPHOSTNAME . OSTYPE).

For example:

```
((SUNRISE . UNIX)(RUTGERS . TOPS-20) etc)
```

End-of-Line Conventions

To ensure correct line spacing, set the TCPFTP.EOL.CONVENTION switch. The associated function takes one argument, TYPE, which sets it correctly. TYPE can be one of the following:

CR	Set EOL to CR
LF	Set EOL to LF
CRLF	Set EOL to CRLF
OS	Set it to something based on the OS
Other	Set it to the default (CRLF)

Loading TCP

Make sure the variables DIRECTORIES and LISPUSERSDIRECTORIES point to the location of the .LCOM files of TCP-IP, or that they are in the connected directory.

You can then load TCP.LCOM which in turn loads its dependent files.

If you plan to do TCP file transfers, load TCPFTP.

If you plan to use the 11xx Lisp workstation as a TCPFTP Server host, load TCPFTPSRV. To start the server, evaluate (TCPFTP.SERVER). An Interlisp machine running the TCPFTP server should be identified as a TOPS-20 machine in the other Interlisp machines' HOSTS.TXT table. It will thus masquesrade as a TOPS-20 server.

The rest is automatic. You can treat an Interlisp host running the server just like any other TCPFTP server. The default path for resolving filenames is {DSK}<LISPFILES>, but you can change or override it.

For example, assume {ERIC} is a machine running the FTP server. From another machine which has TCPFTP loaded, you can do SEE {ERIC}{FLOPPY}FOO, which will type out the file FOO located on the floppy drive of {ERIC}.

If you plan to Chat to a TCP host, load CHAT, CHATTERMINAL, DMCHAT and then TCPCHAT.LCOM. Be sure that hosts with which you wish to chat have their NETWORKOSTYPES set.

If you plan to use the TCP Trivial File Transfer Protocol, load TCPTFTP.

Interlisp's TCPTFTP also provides a TCPTFTP server. Load TCPTFTP.LCOM and evaluate (TFTP.SERVER). You can then use the appropriate TFTP commands to copy files from the Interlisp machine; for example TFTP.PUT and TFTP.GET.

Verifying TCP Connections

Load TCPDEBUG. Execute (TCPTRACE T) and you will be prompted to open a window to show TCP packets. Select INCOMING, OUTGOING and CONTENTS from the window's menu. If the host that you are communicating with has a TCP echoserver process you can then try (TCP.ECHOTEST 'HOSTNAME 3) . For example, using the above HOSTS.TXT file this would be (TCP.ECHOTEST 'SIMBA 3).

You will be prompted to open a window for the echo test and should see text, for example:

```
This is byte number 21
This is byte number 45
This is byte number 69
```

You should also see packets being sent and received in the TCPTRACE window.

Note: If a remote host is not running a TCP echo server process you will not get this response.

Connecting, Transferring Files, and Chatting to a Host

First log in to the host by typing **(LOGIN 'HOST)**; for example, **(LOGIN 'SUNRISE)**. You will then be prompted for a user name and password. This will be sent to the host when you attempt to CONNect or Chat.

Use the command CONN {HOST}<DIRECTORY>SUBDIR> to connect to a particular host. The local directory delimiters < and > can be used when connecting or file transferring. When communicating with a remote host you can specify the directory path as <DIRECTORY> SUBDIRECTORY> SUBDIRECTORY. . . , and the appropriate delimiters are presented to the remote host. Determination of what delimiter is presented depends upon the value of the OSTYPE field in the HOSTS.TXT file. If the field is empty, OSTYPE = 'Interlisp' is the default.

Using the above HOST.TXT file as an example you can do the following:

```
                UNIX  CONN {SUNRISE}<DIR>SUBDIR>SUBDIR>
                 VMS  CONN {OBERON}<DIR>SUBDIR>SUBDIR>
             TOPS-20  CONN {RUTGERS}<DIR>SUBDIR>
      SYMBOLICS-3600  CONN {SIMBA}<DIR>SUBDIR>
```

You can then do a DIR of the remote host, COPYFILE files to and from the host, assuming TCPFTP is loaded, MAKEFILE, etc.

Since the TCPFTP specification does not specify file type conventions, the variable TCP.DEFAULT.FILETYPES is used to associate a file's extension with the type of file it is. It is a list in the form (extension . type); for example,

```
((LCOM . BINARY)  (TXT . TEXT) etc)
```

Since UNIX systems are case-sensitive, you should also have the lower case version of the file extensions on this list. If a file extension is not found on this list, the variable TCP.DEFAULTFILETYPE is used as the default file type during file transfers.

To Chat to a remote host, select Chat from the background menu and enter the host name when prompted. You will be prompted for a Chat window and should then be able to chat to the host. If you have problems opening the Chat connection, try (CHAT 'HOST 'NONE). This will suppress the automatic login.

Warning: If no username or password has been provided for the TCP host, the default ID and password will be sent. This may create a security hazard.

Making a Sysout that Contains TCP-IP

1. Load Medley sysout.
2. Create TCP host table.
3. Load TCPCONFIG.LCOM and run TCP.CONFIGURE if there is no IP.INIT file on local disk.
4. Load TCP.LCOM.TCPFTP.LCOM, TCPCHAT.LCOM.
5. Load TCPDEBUG.LCOM if you always want to have trace and echo facilities available.
6. Evaluate (TCP.STOP)
7. Evaluate (STOPIP)
8. Evaluate

   ```
   (SETQ RESTARTETHERFNS (LIST '(LAMBDA NIL (AND \IPFLG
   (\IPINIT)))))
   ```

9. Load any other files that you want in this sysout.

10. Evaluate SYSOUT to the device of your choice. Evaluate `(\TCP.INIT)` to re-enable TCP.
11. Load TCP sysout on other machine.
12. Create TCP host table.
13. Evaluate `(TCP.CONFIGURE)` and identify the new machine.
14. Evaluate `(\TCP.INIT)` to re-enable TCP.
15. Evaluate (\IPINIT) to restart the IPLISTENER process.

TCP-IP Protocol Layers

The TCP-IP family consists of four principal protocol layers: the link layer, the internet layer, the transport layer, and the application layer.

Link Layer

The physical link layer, the medium for transferring packets between hosts, is assumed to be any medium capable of transporting packets of data between hosts. Common link layers in this family include the Ethernet and the ARPANET.

The Address Resolution (AR) Protocol enables hosts to map between internet addresses and link layer addresses.

For example, the internet layer protocol IP (see below) uses a 32-bit combined unique host and network address; the host address field is of variable size and depends on the pattern encoded in the high-order bits of the address. On the other hand, the 10 MB Ethernet uses a fixed-size 48-bit unique host address. The Address Resolution protocol, documented in RFC826, allows hosts to discover dynamically the link layer address equivalents of other internet hosts.

Internet Layer

The internet layer is responsible for routing packets between hosts. Unlike the link layer, the internet layer is capable of moving packets between hosts that are not connected to the same network. The term IP in TCP-IP refers to the Internet Protocol, the protocol that performs this task in the TCP-IP family. IP is documented in RFC791. IP is not assumed to be error-free; packets may be lost or duplicated while moving through the internet. It is the responsibility of the transport layer (see below) to guarantee perfect delivery, should the client require it.

IP also depends on an associated protocol called the Internet Control Message Protocol (ICMP). ICMP is responsible for handling exception conditions that arise between hosts using IP. Such conditions include the inability to deliver packets, errors in packet formats, etc. ICMP is documented in RFC792.

Transport Layer

The transport layer is responsible for assuring error-free, duplicate-free, sequenced delivery of packets between communicating processes. The most common transport layer is TCP, the Transport Control Protocol. TCP maintains the appearance of a perfect byte stream between processes. TCP is documented in RFC793.

An unreliable transport layer called the User Datagram Protocol (UDP) allows for packet exchange between communicating processes, but makes no attempt to guarantee delivery, suppress duplication, etc. Clients of UDP must provide their own error-recovery mechanisms if necessary. UDP is documented in RFC768.

Application Layer

Many applications exist in the TCP-IP family. The most common applications are file transfer, virtual terminal interaction, and mail delivery.

File Transfer

Two principal file transfer applications are in use: FTP, based on TCP and documented in RFC765; and TFTP (the Trivial File Transfer Protocol), based on UDP and documented in RFC783. Both are implemented in Interlisp, and are discussed at greater length below.

Virtual Terminal Interaction

The TELNET protocol, documented in RFC854, specifies the protocol for virtual terminal interaction between a user and a remote system. The Chat module will use the TELNET protocol to connect to TCP-only hosts.

Mail Delivery

The Simple Mail Transfer Protocol (SMTP) enables the delivery of mail between system elements using TCP. It is not currently implemented in Interlisp. SMTP is documented in RFC821. The format of messages is described in RFC822.

Primer on IP Networks

The Internet Protocol internetwork is a collection of IP networks, a subset of which may communicate with each other. Each network is assigned an IP address, which is composed of a network number and a host number. No two hosts in the internetwork have the same network and host number combination; the composition of the network and host number for a particular host unambiguously identifies that host within the internetwork.

Network Addresses

The address space of the internetwork is formed of the concatenated network and host numbers of its constituent hosts, and is 32 bits long. This 32-bit address space is currently partitioned into three classes of network addresses, known as class-A, class-B, and class-C:

Class-A addresses consist of 7 bits of network number and 24 bits of host number.

Class-B addresses consist of 14 bits of network number and 16 bits of host number.

Class-C addresses consist of 21 bits of network number and 8 bits of host number.

Thus, there may be 128 class-A networks, 16,384 class-B networks, and over two million class-C networks. In addition, a single class-A network has the capacity to address over 16 million hosts, while a class-C network can address only 255 hosts. The class to which a particular IP network belongs may be determined by examining the most significant bits of its address.

Network number assignments are strictly controlled by a central authority. Institutions requesting network assignments are given class-A, -B, or -C networks

depending on their estimated eventual size (numbers of hosts). Sites without assigned network numbers may request an assigned number by contacting:

Joyce Reynolds
USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, California 90292-6695
Phone: (213) 822-1511
ARPANET: JKREYNOLDS@USC-ISIF.ARPA

IP addresses are normally stored or exchanged as single 32-bit numbers. The printed representation of an IP address takes the form W.X.Y.Z, where W through Z are the decimal equivalents of each of the 8--bit bytes that constitute the address. Class-A addresses are of the form N.H.H.H; class-B addresses are of the form N.N.H.H; and class-C addresses are of the form N.N.N.H, where N indicates a byte of the network number, and H indicates a byte of the host number.

Class-A: N.H.H.H The first number is between 0-127 (for example, 122.0.2.1)

Class-B: N.N.H.H The first number is between 128-191 (for example, 153.4.23.5)

Class-C: N.N.N.H The first number is between 192-255 (for example, 194.5.67.3)

For example, 36.47.0.12 is an address on network 36, a class-A network; and 192.10.200.1 is an address on network 192.10.200, a class-C address.

Broadcast Address

The Internet Protocol defines an address in which the host field contains all ones to be a broadcast address for its network. Thus, the address 36.255.255.255 is the broadcast address on network 36, and 192.10.200.255 is the broadcast address on network 192.10.200.

Subnets

It is quite common for class-B networks to be partitioned into a set of smaller subnetworks, which are really class-C networks, but have the wrong network number to be recognized as class-C networks. This is just as common is partitioning a class-A network into many class-B subnetworks. An implementation of TCP-IP that is not prepared to handle this violation of the IP standard will not be able to communicate with hosts on the same network but different subnetworks. Fortunately, extending an IP implementation to support subnetworks is straightforward.

`SUBNETMASK` is a 32-bit parameter that resembles an IP address. The purpose of the mask is to enable a host to determine when a destination IP address is or is not on the same subnet as the sending host itself.

The `SUBNETMASK` has the following properties:

- The bitwise-AND of a source host's address (for example, this machine) and the `SUBNETMASK` must be equal to the bitwise-AND of a destination host's address and the `SUBNETMASK` if and only if the two hosts are on the same subnetwork.
- The bitwise-AND of a source host's address and the `SUBNETMASK` must not be equal to the bitwise-AND of a destination host's address and the `SUBNETMASK` if and only if the two hosts are on different subnetworks.

As an example, consider network 39.0.0.0. This is a class-A network. Suppose this network consists of a number of subnetworks; for example, subnetworks with numbers like 39.47.*.* and 39.9.*.*. According to the IP specification, these subnetworks should

really be one monolithic network, such that a host desiring to communicate with any other host whose address begins with 39... should have to take no special action with regard to routing packets to that host. Let us assume that this is not the case. The only way a machine has of telling which hosts are on different networks is to compare the masked version of the address with the masked version of its own address.

To continue the example further, assume the following:

Host A has address 39.9.0.6. Host A's SUBNETMASK is 39.255.0.0.

Host B has address 39.9.0.7. Host B's SUBNETMASK is also 39.255.0.0.

Host C has address 39.47.0.6. Host C's SUBNETMASK is also 39.255.0.0.

When host A sends to host B, it compares its masked address with host B's masked address, and finds them equal:

39.9.0.6 AND 39.255.0.0 = 39.9.0.0; 39.9.0.7 AND 39.255.0.0 = 39.9.0.0

However, when host A sends to host C, it finds the masked comparison does not match:

39.9.0.6 AND 39.255.0.0 = 39.9.0.0; 39.47.0.6 AND 39.255.0.0 = 36.47.0.0

Class-A networks that are subdivided into class-B subnetworks have SUBNETMASKs that look like X.255.0.0, where X is the class-A network number. Likewise, class-B networks subdivided into class-C subnetworks have SUBNETMASKs that look like X.Y.255.0, where X.Y is the class-B network number. Finally, networks in which subnet routing is not in use have SUBNETMASKs identical to their network addresses. For example, if network 36 did not use subnet routing, its SUBNETMASK would be 36.0.0.0.

The definitive document on this approach to subnetwork routing is RFC940.

Interlisp Files

The files that implement the TCP-IP protocol suite are divided into two classes: those that implement low-level functionality, normally not of interest to general users, and those that implement higher-level functionality for user programs (either application or transport layer protocols).

The higher-level functions reside in the files TCP, TCPDEBUG, TCPFTP, TCPCHAT, TCPNAMES, TCPPUDP, and TCPFTP.

TCP	The TCP layer. Implements TCP streams, based on the buffered TCP device (for example , BIN runs in microcode).
TCPDEBUG	Contains routines to help debug TCP and TCP-based applications.
TCPFTP	Contains the TCP-based file transfer protocol. Creates a new virtual I/O device, allowing transparent filing operations with TCP-only hosts.
TCPFTPSRV	Contains the TCP-based FTP server program. When the server program is running on a Xerox 1100-series workstation, other TCP-based hosts may transfer files to and from the workstation.
TCPNAMES	Implements translation of file name formats between operating system types.
TCPCHAT	Implements the TELNET protocol for the Chat system.
TCPUDP	Contains the UDP layer.

TCPTFTP Implements the TFTP protocol. Creates a buffered TFTP device to allow efficient bulk transfer between hosts.

The low-level functions reside in the files TCPLLIP, TCPLLICMP, TCPLLAR, TCPHTE, and TCPCONFIG.

TCPLLIP Implements the IP layer.

TCPLLICMP Implements ICMP for IP.

TCPLLAR Implements ARP for the 3- and 10-megabyte Ethernets.

TCPHTE Implements the functionality necessary to parse RFC810-style HOSTS.TXT files. This allows name-to-address translation within the Interlisp host.

TCPCONFIG Provides a function to carry on a configuration dialog when TCP-IP is first installed on a machine. This file needs to be loaded only once, to produce the file {DSK}IP.INIT. Thereafter, TCPCONFIG is needed only to reestablish or modify IP parameters.

TCPDOMAIN Implements a domain host address lookup client.

TCP

TCP implements the transport control protocol for Interlisp. After TCP is loaded, Interlisp supports a TCP stream capable of bidirectional I/O to a remote system element. The following functions are intended for use by applications programs.

(TCP.OPEN *DST.HOST DST.PORT SRC.PORT MODE ACCESS NOERRORFLG OPTIONS*) [Function]

Opens a TCP stream to *DST.PORT* on *DST.HOST* from *SRC.PORT*.

DST.HOST can be a host name, an IP host address in text format (such as 192.10.200.1), or the 32-bit integer representation of an IP host address as returned by the function DODIP.HOSTP (which is documented under TCPLLIP).

DST.PORT is a 16-bit number representing a TCP port open in LISTENING mode on the remote system.

SRC.PORT is also a 16-bit number, but may be supplied as NIL to obtain a defaulted unique local port number.

MODE is either ACTIVE, meaning to act as initiator of the connection, or PASSIVE, meaning to wait for a remote system element to initiate the connection.

ACCESS is either INPUT, OUTPUT, or APPEND (OUTPUT and APPEND are treated in the same manner).

If *NOERRORFLG* is non-NIL, TCP.OPEN will return NIL if the connection fails; otherwise, TCP.OPEN will call ERROR to signal failure.

OPTIONS is an optional parameter which allows the application program to control some of the characteristics of the TCP connection. *OPTIONS* is supplied in property-list format. Currently, the only recognized option is MAXSEG, whose value should be the number of data bytes the remote TCP sender is allowed to place into a single TCP segment (Ethernet packet). The maximum value of MAXSEG is 536.

If TCP.OPEN succeeds, it returns a STREAM open as specified by *ACCESS*. The generic operations BIN, BOUT, PEEKBIN, BINS, BOUTS, READP, EOFP, OPENP, GETFILEPTR, FORCEOUTPUT, and CLOSEF may be performed on streams opened for suitable access.

(TCP.OTHER.STREAM *STREAM*) [Function]

Returns the *STREAM* open in the other direction with respect to *STREAM* (for example, if *STREAM* is open for INPUT, TCP.OTHER.STREAM returns a *STREAM* open for OUTPUT, and vice versa).

(TCP.URGENT.EVENT *STREAM*) [Function]

Returns an event upon which a user process may wait for URGENT data to arrive on *STREAM*.

(TCP.URGENTP *STREAM*) [Function]

Returns T if *STREAM* is currently reading URGENT data.

(TCP.URGENT.MARK *STREAM*) [Function]

Marks the current point in *STREAM* as the end of URGENT data. *STREAM* must be open for OUTPUT.

(TCP.CLOSE.SENDER *STREAM*) [Function]

Closes the output side of *STREAM*, which may be either the INPUT or OUTPUT stream for the connection. This function differs from CLOSEF in that the INPUT side of the connection is not closed (although the remote system element may close the connection once the local output side of the connection is closed).

(TCP.STOP) [Function]

Disables the TCP protocol, closing all open TCP streams.

(\TCP.INIT) [Function]

(Re)initializes the TCP module.

\TCP.DEFAULT.RECEIVE.WINDOW [Variable]

Is the default number of bytes allowed outstanding from the remote system. It is initially 4,096.

\TCP.DEFAULT.USER.TIMEOUT [Variable]

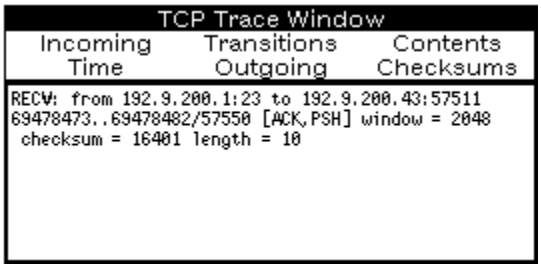
Is the default number of milliseconds a remote system element is allowed to remain silent before the TCP connection is declared broken. It is initially 60,000.

TCPDEBUG

TCPDEBUG implements tracing and test functions used to debug TCP and TCP-based applications.

(TCPTRACE) [Function]

Opens a trace window and attaches a menu to the window's top.



The menu entries represent state changes or data elements to be traced; each entry is a toggle. Clicking on the toggle once will activate the trace of the particular element and will gray-over the entry; clicking a second time will deactivate the tracing and ungray the menu item. The following data elements/transitions may be displayed:

Contents	Displays a line's worth of packet contents. The Incoming or Outgoing switch must be on.
Incoming	Displays incoming data.
Outgoing	Displays outgoing data.
Checksums	Displays checksums for each TCP segment.
Time	Displays the time interval since the last action on the connection.
Transitions	Displays state transitions on the TCP state machine.

(PPTCB *TCB FILE*) [Function]

Prints the state of a TCP connection. PPTCB is normally the INFO function for the process that monitors a connection; thus, selecting INFO in the process status window will cause a window to pop up containing a report on the status of the associated connection.

(TCP.ECHOTEST *HOST NLINES*) [Function]

Opens a TCP connection to the TCP echo port on *HOST* and sends *NLINES* of random text. The echo responses are displayed in a window. If *NLINES* is NIL, the echo test will run forever.

(TCP.ECHO.SERVER *PORT*) [Function]

Starts a TCP echo server on *PORT* (defaults to the TCP echo port). It is usually more useful to start the echo server as a process by doing (ADD.PROCESS '(TCP.ECHO.SERVER *PORT*)).

(TCP.SINK.SERVER *PORT*) [Function]

Starts a TCP sink server on *PORT* (defaults to the TCP sink port). Any data sent to this port will be acknowledged and discarded. As with the TCP echo server, it is usually more useful to start this server as an independent process.

(TCP.FAUCET *HOST PORT NLINES*) [Function]

If *HOST* is non-NIL, this function opens a connection to *PORT* on *HOST* and sends *NLINES* of text (the default is to send lines of text forever). *PORT* defaults to the TCP sink port. If *HOST* is NIL, this function waits for a remote system to connect to the TCP faucet port and then sends out *NLINES* of random text.

TCPFTP

TCPFTP implements a virtual I/O device that performs Lisp filing operations transparently using the RFC765 FTP protocol. The standard filing operations of reading, writing, renaming, deleting, and directory enumeration are supported by the TCPFTP device. However, neither random access filing nor GETFILEINFO are supported, as there is no protocol specification for performing these operations on files. Interlisp operations such as RECOMPILE will not work when files are stored on TCPFTP file servers.

Once TCPFTP is loaded, filing operations should be transparent to users; no additional initialization need be performed. There are, however, three important global variables:

TCPFTP.EOL.CONVENTION [Variable]

This variable controls the end–of–line convention used with files accessed via TCP. Generally, you should set it to match the convention on the system where the files reside. The value can be one of those shown below.

(TCPFTP.EOL.CONVENTION *TYPE*) [Function]

Sets the variable TCP.EOL.CONVENTION to *TYPE*. *TYPE* can be one of the following:

CR	Set EOL to CR
LF	Set EOL to LF
CRLF	Set EOL to CRLF
OS	Set it to something based on the OS
Other	Set it to the default (CRLF)

TCPFTP.DEFAULT.FILETYPES [Variable]

This variable is an association list, keyed by common extensions of file names, and contains appropriate file types (for example, TEXT or BINARY) for such files. The TCPFTP protocol provides no mechanism for determining the type of a file about to be retrieved. The file type is usually known in the case of output operations (for example, COPYFILE or MAKEFILE to a file server). However, in the case of COPYFILE from a file server, the TCPFTP module has to infer the file type from other knowledge. The module tries to match the extension of the file name with an entry on the list TCPFTP.DEFAULT.FILETYPES. If it finds a match, it uses the value of the entry in the list as the file type of the file; if it doesn't find a match, it uses the value of TCP.DEFAULTFILETYPE for the file type of the file.

TCP.DEFAULTFILETYPE [Variable]

If no matching extension is found for the file being opened, the TCPFTP module uses the value of TCP.DEFAULTFILETYPE as the file type of the remote file. The initial value of TCP.DEFAULTFILETYPE is BINARY.

The following functions are available for debugging broken file server connections.

(FTPDEBUG *FLG*) [Function]

If *FLG* is T, this function opens a scrolling trace window that displays FTP commands as they are issued. PUPFTP commands will also be displayed in this window (the window is the value of FTPDEBUGLOG).

(\TCP.BYE *HOST*) [Function]

Breaks an FTP connection to *HOST*.

(\TCPFTP.INIT) [Function]

(Re)initializes the TCPFTP module.

TCPFTPSRV

The TCPFTPSRV module contains a program which implements an FTP service for Interlisp. When this program is running on a workstation, other hosts are able to store and retrieve files from the workstation.

(TCPFTP.SERVER *PORT DEFAULT.FILE.PATH*) [Function]

To start the server program, evaluate the form (TCPFTP.SERVER). If *PORT* is supplied, the FTP server program will listen for connections on the TCP port specified by *PORT*; otherwise, the server will listen on the default FTP server port, port 21.

If *DEFAULT.FILE.PATH* is supplied, the initial path for resolving file names will be relative to *DEFAULT.FILE.PATH*; the default value of this variable is {DSK}<LISPFILES>.

TCPFTP.SERVER.USE.TOPS20.SYNTAX [Variable]

This variable controls whether file names sent back to FTP client programs are formatted in Tops-20 or Interlisp syntax. If the variable is true (the default), all file names will be formatted in Tops-20 syntax. This permits an Interlisp workstation to masquerade as a Tops-20 mainframe for the purposes of file transfer to and from other vendors' machines.

TCPNAMES

The TCPNAMES module provides a set of functions for translating among the file-naming conventions of different operating systems. This is needed by the TCPFTP module in order for it to convert between Interlisp format file names and the file name formats of other operating systems.

(REPACKFILENAME.STRING *NAME FOROSTYPE*) [Function]

NAME is a file name in some operating system's format. *FOROSTYPE* is the name of an operating system. REPACKFILENAME.STRING attempts to translate *NAME* into a format acceptable to the operating system named by *FOROSTYPE*. *NAME* may be a string or atom; the function always returns a string.

Currently acceptable operating system types are:

```
IFS
INTERLISP
MS-DOS
SYMBOLICS-3600
TENEX
TOPS-20 (also TOPS20)
UNIX
VMS
```

(TI-Explorers should use TOPS-20 as their operating system.)

The correspondence between the target operating system type and the file name translation function is maintained in an extensible hash table.

(\REPACKFILENAME.NEW.TRANSLATION *OSTYPE FUNCTION*) [Function]

This function adds a new file name translation function for a new operating system type. The function must be a LAMBDA-NOSPREAD function, and must be prepared to receive either a single property-list format argument, such as would be returned by UNPACKFILENAME, or an arbitrary number of arguments in property-list format.

File names in the above format will be passed to the translation function adhering to the conventions of many operating systems; the function must recognize the operating system type and produce the desired output format, which must be a string.

\REPACKFILENAME.OSTYPE.TABLE [Variable]

This variable is the hash table that stores the correspondence between operating system types and translation functions.

TCPCHAT

TCPCHAT implements the TELNET protocol for virtual terminal I/O between Interlisp and a remote system. Once loaded into Interlisp, the standard Chat system will use TCP TELNET to communicate with hosts that are believed to support the protocol.

No user-callable functions reside in this module, although the following variables may be of interest.

TCPCHAT.TELNET.TTY.TYPES [Variable]

This variable is an association list that maps internal names of Chat terminal emulators to official terminal names as specified in RFC884, the TELNET Terminal Type Option. This allows TCPCHAT to set the user's terminal type automatically when a connection is established.

TCPCHAT.TRACEFLG [Variable]

If this variable is non-NIL, TELNET negotiations will be printed to TCPCHAT.TRACEFILE (see below). This is sometimes useful in debugging negotiation problems.

TCPCHAT.TRACEFILE [Variable]

TELNET negotiations are printed to this file if TCPCHAT.TRACEFLG is non-NIL.

TCPUDP

UDP implements the user datagram protocol. The following functions are meant to be called by client applications.

(UDP.INIT) [Function]

Initializes the UDP module. This function is normally called when UDP is loaded and should not need to be called again under normal circumstances.

(UDP.STOP) [Function]

Disables the UDP module, closing any open UDP sockets.

(UDP.OPEN.SOCKET *SKT# IFCLASH*) [Function]

Opens a socket for UDP operations.

SKT#, if supplied, is a 16-bit number and will default to a number between 1,000 and 65,535.

IFCLASH specifies what to do if the requested socket is already open and is handled as in OPENPUPSOCKET and OPENNSOCKET (see the *IRM*).

It returns an instance of an IPSOCKET.

(UDP.CLOSE.SOCKET *IPSOCKET NOERRORFLG*) [Function]

Closes an open *IPSOCKET*. If *IPSOCKET* is not an open socket and *NOERRORFLG* is NIL, an error will occur; otherwise, NIL is returned if the socket is not active, and T is returned if the socket is active.

Any remaining packets on the socket's input queue are discarded when this function is called.

(UDP.SOCKET.EVENT *IPSOCKET*) [Function]

Returns an event that a process may use to wait for packet arrival on *IPSOCKET*.

(UDP.SOCKET.NUMBER *IPSOCKET*) [Function]

Returns the socket number of *IPSOCKET*.

(UDP.GET *IPSOCKET WAIT*) [Function]

Returns the next packet waiting on *IPSOCKET*. If no packets are waiting, does one of the following based on the value of *WAIT*.

NIL	Returns immediately.
T	Waits forever for a packet to arrive.
a number	*FIXP* waits up to *WAIT* milliseconds for a packet to arrive and returns NIL if none arrived during that time.

Thus, this function is like GETPUP and GETXIP.

(UDP.SETUP *UDP DESTHOST DESTSOCKET ID IPSOCKET REQUEUE*) [Function]

Initializes a fresh packet (as returned from \ALLOCATE.ETHERPACKET). The packet will be sent to *DESTSOCKET* on *DESTHOST*.

ID is a number to be placed in the IP header ID field (zero is fine).

REQUEUE specifies what to do with the packet after it is sent; NIL (the default) means no special treatment; FREE means to release the packet and return it to the free packet queue. Any instance of a SYSQUEUE will cause the packet to be queued on the tail of the specified queue.

UDP.SETUP initializes all IP and UDP fields and sets the packet up as a minimum-length UDP packet.

(UDP.SEND *IPSOCKET UDP*) [Function]

Sends *UDP*, a UDP-formatted packet, out from *IPSOCKET*.

(UDP.EXCHANGE *IPSOCKET OUTUDP TIMEOUT*) [Function]

Sends *OUTUDP* out from *IPSOCKET* and waits *TIMEOUT* milliseconds for a response; returns NIL if no response came in during the specified interval, or the packet that did come in during that time.

Clears the socket's input packet queue before waiting for a packet to arrive.

(UDP.APPEND.BYTE *UDP BYTE*) [Function]

Appends *BYTE* to the UDP data portion of *UDP* and increments the UDP and IP length fields by one.

(UDP.APPEND.WORD *UDP WORD*) [Function]

Appends *WORD* to the UDP data portion of *UDP* and increments the UDP and IP length fields by two.

(UDP.APPEND.CELL *UDP CELL*) [Function]

Appends *CELL* to the UDP data portion of *UDP* and increments the UDP and IP length fields by four.

(UDP.APPEND.STRING *UDP STRING*) [Function]

Appends *STRING* to the UDP data portion of *UDP* and increments the UDP and IP length fields by the length *STRING*.

TCPTFTP

TFTP implements the trivial file transfer protocol. This protocol is useful for transferring unimportant files rapidly (for example, between workstations and printers). The following user-callable functions exist.

(TFTP.PUT *FROM TO PARAMETERS*) [Function]

Sends a file to a TFTP host.

FROM may refer to any accessible file; *TO* must refer to a file accessible via TFTP.

No attempt is currently made to translate between Interlisp file name syntax and remote system file name syntax for *TO*.

For example, if *TO* resides on a UNIX host, it would take a syntax like {HOST}/DIRECTORY/SUBDIRECTORY/FILENAME.

PARAMETERS is currently a list of parameters in the same format used by OPENFILE in .PARAMETERS; for example ((EOLCONVENTION 1) (TYPE TEXT)).

Note: TFTP transfers between Xerox Lisp and UNIX hosts initiated from Xerox Lisp should have the PARAMETERS argument be '((EOLCONVENTION 10)).

(TFTP.GET *FROM TO PARAMETERS*) [Function]

Gets a file from a TFTP host. *FROM* must be a file accessible by TFTP; *TO* may be any file.

The file name syntax caveats for *FROM* are the same as for *TO* in TFTP.PUT. *PARAMETERS* is also as in TFTP.PUT.

(TFTP.SERVER *LOGSTREAM*) [Function]

Starts a TFTP server process.

LOGSTREAM may be left NIL, causing a new window to appear when the TFTP server is first invoked. Remote systems that support TFTP clients may store or retrieve files through any Interlisp workstation running the TFTP server.

The full Interlisp syntax for file names is supported; thus, requests to store files whose names include hosts will result in the Interlisp workstation's transparently storing the files on the designated hosts.

(\TFTP.OPENFILE *FILENAME ACCESS RECOG PARAMETERS*) [Function]

Returns a STREAM to open for *ACCESS* on *FILENAME*.

PARAMETERS is the usual format; TYPE is the only recognized parameter (BINARY opens a stream in *octet* format; TEXT, the default, opens a stream in NETASCII format; see RFC783).

BIN, BOUT, READP, EOFP, etc., may be used on this stream.

The stream is not RANDACCESSP.

(\TFTP.CLOSEFILE *STREAM*) [Function]

Closes the open stream. This is normally useful for streams open for OUTPUT; for INPUT streams, end-of-file will occur eventually.

TCPLLIP

For users planning implementations on top of IP, the following low-level TCP functions are available.

IP Socket Access

(\IPINIT) [Function]

Reinitializes the IP world; for example, after some catastrophe.

(STOPIP) [Function]

Disables IP.

(DODIP.HOSTP *NAME*) [Function]

If *NAME* is an integer, *NAME* is returned unaltered. If *NAME* is a text format IP host address (such as 192.10.200.1), DODIP.HOSTP returns its integer representation.

If *NAME* is a string or atom name, DODIP.HOSTP attempts to convert *NAME* to its IP host address integer value, using information supplied in the HOSTS.TXT file (see TCPHTE, below), followed by doing a domain query if TCPDOMAIN is loaded.

If *NAME* is unknown, DODIP.HOSTP returns NIL.

If *NAME* is known, it is cached with its corresponding address so that the function IPHOSTNAME may be used later to convert the address back to a name.

(IPHOSTNAME *IPADDRESS*) [Function]

Tries to convert *IPADDRESS* to a host name.

If *IPADDRESS* has no known name, it is converted to the text representation of an IP address (for example, 192.10.200.1).

(IPTRACE *MODE*) [Function]

Turns on tracing of IP activity. This function is like PUPTRACE and XIPTRACE, which are documented in the *IRM*.

If *MODE* is NIL, IP tracing is disabled.

If *MODE* is T, verbose IP tracing is enabled.

If *MODE* is PEEK, concise IP tracing is enabled. If *MODE* is either T or PEEK, the user is prompted for a window into which trace output will be printed.

(\IP.ADD.PROTOCOL *PROTOCOL SOCKETCOMPAREFN NOSOCKETFN INPUTFN ICMPFN*) [Function]

Defines a new IP-based protocol. The lowest-level IP functions maintain a list of active protocols and perform packet delivery based on the existence of open sockets for protocols of received packet types.

PROTOCOL is a protocol number, a number between 1 and 255. The following protocols are defined and should not be disturbed:

TCP	6
ICMP	1
UDP	17

SOCKETCOMPAREFN is a function with two arguments, an IP packet that has just been received and an open IPSOCKET. This function should return NIL if the packet does not belong to the supplied socket, or T if it does. The function will typically be interested in the IPSOCKET field of the IPSOCKET.

NOSOCKETFN is a function with one argument, an IP packet that has just been received. Its purpose is to handle received packets for which no socket can be found. If *NOSOCKETFN* is NIL, the default function, \IP.DEFAULT.NOSOCKETFN, will be used; this function simply returns an ICMP message indicating the socket is unreachable.

INPUTFN is a function with two arguments, a received IP packet and an open IPSOCKET. The *INPUTFN* is supposed to handle reception of packets when

their destination socket has been found. If *INPUTFN* is NIL, the default function, \IP.DEFAULT.INPUTFN, will be supplied.

INPUTFN enqueues the received packet on the IPSQUEUE field of the IPSOCKET if the current queue length (stored in the IPSQUEUELENGTH field) is less than the allocated length (stored in the IPSQUEUEALLOC field).

INPUTFN also increments the IPSQUEUELENGTH field, and notifies the event stored in the IPSEVENT field.

ICMPFN is a function with two arguments and is called when an ICMP packet referring to the protocol is received. The first argument is a pointer to the received ICMP packet. The second argument is a pointer that may be used as if pointed to the original outgoing packet included in the ICMP data. This allows the protocol functions to parse the data in the ICMP packet to determine which socket sent the offending packet. The *ICMPFN* must never attempt to deallocate the packet identified by the second argument; however, it is quite permissible (and expected) that the *ICMPFN* will release the packet identified by the first argument. The default *ICMPFN* simply releases the packet identified by the first argument.

\IP.ADD.PROTOCOL returns an IPSOCKET datum, which represents the active protocol; it is not in fact a useful IPSOCKET and may be safely ignored.

(\IP.DELETE.PROTOCOL *PROTOCOL*) [Function]

Deactivates a protocol with protocol number *PROTOCOL*. Any open sockets are closed.

(\IP.OPEN.SOCKET *PROTOCOL SOCKET NOERRORFLG SOCKETCOMPAREFN NOSOCKETFN INPUTFN*) [Function]

Attempts to open an IPSOCKET for protocol *PROTOCOL*.

SOCKET is the identifying information for this socket; this quantity will be EQUAL-compared with other sockets open on *PROTOCOL*. Should a match be found, an error will occur unless *NOERRORFLG* is T, in which case the existing socket will be returned.

SOCKETCOMPAREFN, *NOSOCKETFN*, and *INPUTFN* may be supplied to override the functions specified when the protocol was defined; they are not normally useful, however.

(\IP.CLOSE.SOCKET *SOCKET PROTOCOL NOERRORFLG*) [Function]

Closes a socket open on *PROTOCOL*. *SOCKET* is the same quantity passed to \IP.OPEN.SOCKET; it is currently not an instance of an IPSOCKET. If *NOERRORFLG* is T, an error will not occur if the socket is not found.

IP Packet Building

The following functions are useful for placing bytes into IP packets (as allocated by \ALLOCATE.ETHERPACKET).

Most applications will probably want to define a block record to overlay the data portion of an IP packet. Here is an example of such a block record.

Note: Users who are developing new IP-based protocols will need to load EXPORTS.ALL from the library.

```
(ACCESSFNS UDP
    ((UDPBASE (\IPDATABASE DATUM)))
    (BLOCKRECORD UDPBASE
        ((UDPSOURCEPORT WORD)
        (UDPDESTPORT WORD)
        (UDPLENGTH WORD)
        (UDPCHECKSUM WORD)))
    (ACCESSFNS UDP
        ((UDPCONTENTS
        (\ADDBASE
            (\IPDATABASE DATUM)
            (FOLDHI \UDPOVLEN BYTESPERWORD))))))
```

(\IP.APPEND.BYTE *IP BYTE INHEADER*) [Function]

Appends *BYTE* to the IP data portion of *IP* and increments the IP length field by one. If *INHEADER* is T, the IPHEADERLENGTH field is appropriately incremented so that the bytes appear to have been appended to the options portion of the IP header. There must not be any data bytes in the data portion of the packet if this function is to work correctly.

(\IP.APPEND.WORD *IP WORD INHEADER*) [Function]

Appends *WORD* to the IP data portion of *IP* and increments the IP length field by two. *INHEADER* is as in \IP.APPEND.BYTE.

(\IP.APPEND.CELL *IP CELL INHEADER*) [Function]

Appends *CELL* to the IP data portion of *IP* and increments the IP length field by four. *INHEADER* is as in \IP.APPEND.BYTE.

(\IP.APPEND.STRING *IP STRING*) [Function]

Appends *STRING* to the IP data portion of *IP* and increments the IP length field by the length *STRING*.

IP Packet Sending

(\IP.SETUPIP *IP DESTHOST ID SOCKET REQUEUE*) [Function]

Initializes *IP*. This function should be called just after *IP* is obtained from \ALLOCATE.ETHERPACKET; if this is not done, the append functions above will fail.

DESTHOST is the 32-bit IP address to which this packet will be sent.

ID is an arbitrary 16-bit quantity that will become the IPID field of the packet.

SOCKET is the open IPSOCKET from which the packet will be sent.

REQUEUE defaults to FREE and controls the disposition of the packet after transmission (see the *IRM* for the documentation of SETUPPUP or FILLINXIP).

(\IP.TRANSMIT *IP*) [Function]

Tries to send *IP*. Performs IP checksum algorithm prior to sending. Returns NIL if successful, otherwise it returns a status indication, such as NoRouting or

AlreadyQueued. This function is like SENDPUP and SENDXIP, except that no socket argument is required.

TCPHTE

HTE provides functions for parsing HOSTS.TXT files as documented by RFC810. This file is loaded automatically by LLIP and is used by \IPINIT to read in the initial file, HOSTS.TXT. The following variable and function may be of interest.

HOSTS.TEXT.DIRECTORIES [Variable]

Is the search path for the file HOSTS.TXT. This variable is initialized to NIL; thus the search path to be used is by default DIRECTORIES.

(\HTE.READ.FILE *FILE WANTEDTYPES*) [Function]

Reads a HOSTS.TXT file.

WANTEDTYPES is a list of types drawn from the set *{HOST, NET, GATEWAY}*, to be read from the file; types not specified in *WANTEDTYPES* are ignored. *WANTEDTYPES* defaults to *(HOST)*.

TCPDOMAIN

TCPDOMAIN enhances the host address lookup to do queries to a domain name server as specified by RFC883. This allows the system to only keep in memory the addresses of hosts that are actually communicated with, resulting in considerable savings of space on large networks. There is a slight delay in establishinig communications with a host the first time. Since it is faster to initiate communications with hosts defined in the HOSTS.TEXT file, we recommend that frequently-communicated-with hosts be defined therein.

(DOMAIN.INIT) [Function]

(Re)initializes the TCPDOMAIN package.

(DOMAIN.TRACE) [Function]

Opens up a window for tracing domain queries. (The window is the value of DOMAIN.TRACE.FILE.

(DOMAIN.LOOKUP *NAME TYPE SERVER*) [Function]

Looks up a host named *NAME* and type *TYPE* from server *SERVER*. If *SERVER* is unspecified, uses the default server.

(DOMAIN.LOOKUP ADDRESS *NAME SERVER DONT.GET.OSTYPE*) [Function]

Looks up the address of host *NAME* from server *SERVER*. If *SERVER* is unspecified, uses the default server. If DONT.GET.OSTYPE is T, will not get the *OSTYPE* of the host.

(DOMAIN.LOOKUP.NAMESERVER *NAME SERVER*) [Function]

Looks up the server serving the domain of *NAME*. If *SERVER* is unspecified, chooses the "best" server based on its knowledge of the namespace.

(DOMAIN.GRAPH *WINDOW*) [Function]

Draws a graph of the namespace in window *WINDOW*. Opens a new one if *WINDOW* is unspecified.

TCP Debugging Aids

With TCPDEBUG loaded use `(TCPTRACE T)` to open up a trace window of TCP traffic. The appropriate items need to be selected from the windows menu in order for data to be seen.

`(SETQ TCPCHAT.TRACEFLG T)` will print TELNET negotiations to a file, which is what the variable `TCPCHAT.TRACEFILE` points to.

`(FTPDEBUG T)` opens a scrolling trace window that displays FTP commands as they are issued. You will see unencrypted passwords if they are issued.

Limitations

You must use the 1186 microcode in order for TCP-IP to work on an 1186 (microcode for the 1185 will not do).

TCP-IP will not work with UNIX systems that have trailer encapsulation enabled. Connections will hang and then eventually break.

Directory enumeration on a VMS system results in `NIL`.

Known Problems in TCPFTPSRV

It does not handle error conditions in the middle of file transfers.

Doing a DIR gives you only filename and version: no author, creation date, etc. This is because the TCPFTP protocol specification doesn't support author, creation date, etc.

If there are multiple files on the system, deleting a file without specifing a specific version deletes the most recent version. The workaround is to give the specific version to delete.

The subdirectory structure is not presented back to the client host. If you have a file on both the <lispfiles> directory and a subdirectory, when you do a DIR *.* you do not see the subdirectory listed, but you do see that there are two files on the host with the same version number.

References

Users with access to the ARPANET may retrieve any RFC from host SRI-NIC.ARPA with the file transfer protocol (FTP) anonymous log-in option. RFCs are stored under `<RFC>RFC`*nnn*`.TXT`, where *nnn* is replaced by the number of the particular RFC.

From points on the Xerox internet, the RFC files can be retrieved from {Indigo}<RFC>. {Indigo} is an IFS host. From Lisp, you can simply (LOGIN) and supply your GV credentials if you haven't already, open a FileBrowser on that directory, and retrieve the file to the local workstation environment.

The following RFCs are mentioned in this manual:

RFC765 (superseded by RFC 959)
RFC768
RFC783
RFC791
RFC792
RFC793
RFC810 (superseded by RFC 952)
RFC814
RFC821
RFC822
RFC826
RFC854
RFC894
RFC895
RFC903
RFC904
RFC940

[This page intentionally left blank]