CHAPTER 21

ETHERNET

Interlisp was first developed on large timesharing machines which provided each user with access to large amounts of disk storage, printers, mail systems, etc. Interlisp-D, however, was designed to run on smaller, single-user machines without these facilities. In order to provide Interlisp-D users with access to all of these services, Interlisp-D supports the Ethernet communications network, which allows multiple Interlisp-D machines to share common printers, file servers, etc.

Interlisp-D supports the Experimental Ethernet (3 Megabits per second) and the Ethernet (10 Megabits per second) local communications networks. These networks may be used for accessing file servers, remote printers, mail servers, or other machines. This chapter is divided into three sections: First, an overview of the various Ethernet and Experimental Ethernet protocols is presented. Then follow sections documenting the functions used for implementing PUP and NS protocols at various levels.

21.1 ETHERNET PROTOCOLS

The members of the Xerox 1100 family (1100, 1108, 1132), Xerox file servers and laser xerographic printers, along with machines made by other manufacturers (most notably DEC) have the capability of communicating over 3 Megabit per second Experimental Ethernets, 10 Megabit per second Ethernets and telephone lines.

Xerox pioneered its work with Ethernet using a set of protocols known as PARC Universal Packet (PUP) computer communication protocols. The architecture has evolved into the newer Network Systems (NS) protocols developed for use in Xerox office products. All of the members of the Xerox 1100 family can use both NS and PUP protocols.

21.1.1 Protocol Layering

The communication protocols used by the members of the Xerox 1100 family are implemented in a ''layered'' fashion, which means that different levels of communication are implemented as different protocol layers. Protocol Layering allows implementations of specific layers to be changed without requiring changes to any other layers. The layering also allows use of the same higher level software with different lower levels of protocols. Protocol designers can implement new types of protocols at the correct protocol level for their specific application in a layered system.

At the bottom level, level zero, there is a need to physically transmit data from one point to another. This level is highly dependent on the particular transmission medium involved. There are many different level zero protocols, and some of them may contain several internal levels. At level one, there is a need to decide where the data should go. This level is concerned with how to address a source and destination, and how to choose the correct transmission medium to use in order to route the packet towards its destination. A level one packet is transmitted by *encapsulating* it in the level zero packet appropriate for

the transmission medium selected. For each independent communication protocol system, a single level one protocol is de ned. The rule for delivery of a level one packet is that the communication system must only make a best e ort to deliver the packet. There is no guarantee that the packet is delivered, that the packet is not duplicated and delivered twice, or that the packets will be delivered in the same order as they were sent.

The addresses used in level zero and level one packets are not necessarily the same. Level zero packets are speci c to a particular transmission medium. For example, the destination address of a level zero packet transmitted on one of the two kinds of Ethernet is the Ethernet address (host number) of a machine on the particular network. Level one packets specify addresses meaningful to the particular class of protocols being implemented. For the PUP and NS protocols, the destination address comprises a network number, host number (not necessarily the same as the level zero host number), and a socket number. The socket number is a higher- level protocol concept, used to multiplex packets arriving at a single machine destined for separate logical processes on the machine.

Protocols in level two add order and reliability to the level one facilities. They suppress duplicate packets, and are responsible for retransmission of packets for which acknowledgement has not been received. The protocol layers above level two add conventions for data structuring, and implement application speci c protocols.

21.1.2 Level Zero Protocols

Level zero protocols are used to physically connect computers. The addresses used in level zero protocols are protocol speci c. The Ethernet and Experimental Ethernet level zero protocols use host numbers, but level zero phone line protocols contain less addressing information since there are only two hosts connected to the telephone line, one at each end. As noted above, a level zero protocol does not include network numbers.

The 3MB Experimental Ethernet [1] was developed at PARC. Each Experimental Ethernet packet includes a source and destination host address of eight bits. The Experimental Ethernet standard is used by any machine attached to an Experimental Ethernet.

The 10MB Ethernet [2] was jointly developed and standardized by Digital, Intel, and Xerox. Each Ethernet level zero packet includes a source and destination host address that is 48 bits long. The Ethernet standard is used by any machine attached to an Ethernet.

Both of the level one protocols described later (PUP and NS) can be transported on any of the level zero protocols described above.

The Ethernet and Experimental Ethernet protocols are broadcast mediums. Data packets can be sent on these networks to every host attached to the net. A packet directed at every host on a network is a broadcast packet.

Other Level 0 protocols in use in industry include X.25, broadband networks, and Chaosnet. In addition, by using the notion of ‘‘mutual encapsulation’’, it is possible to treat a higher- level protocol (e.g. ARPANET) as if it were a Level Zero Protocol.

21.1.3 Level One Protocols

Two Level One Protocols are used in the Xerox 1100 Family, the PUP and the NS protocols. With the proper software, computers attached to Ethernets or Experimental Ethernets can send PUPs and NS packets to other computers on the same network, and to computers attached to other Ethernets or Experimental Ethernets.

The PUP protocols [3] were designed by Xerox computer scientists at the Palo Alto Research Center. The destination and source addresses in a PUP packet are specified using an 8-bit network number, an 8-bit host number, and a 32-bit socket number. The 8-bit network number allows an absolute maximum of 256 PUP networks in an internet. The 8-bit host number is network relative. That is, there may be many host number ''1''s,but only one per network. 8 bits for the host number limits the number of hosts per network to 256. The socket number is used for further levels of addressing within a specific machine.

The Network Systems (NS) protocols [4, 5] were developed by the Xerox Office Products Division. Each NS packet address includes a 32-bit network number, a 48-bit host number, and a 16-bit socket number. The NS host and network numbers are unique through all space and time. A specific NS host number is generally assigned to a machine when it is manufactured, and is never changed. In the same fashion, all networks (including those sold by Xerox and those used within Xerox) use the same network numbering space there is only one network ''74''.

21.1.4 Higher Level Protocols

The higher level PUP protocols include the File Transfer Protocol (FTP) and the Leaf Protocol used to send and retrieve files from Interim File Servers (IFSs) and DEC File Servers, the Telnet protocol implemented by ''Chat'' windows and servers, and the EFTP protocol used to communicate with the laser xerographic printers developed by PARC (''Dovers'' and ''Penguins'').

The higher level NS protocols include the Filing Protocol which allows workstations to access the product File Services sold by Xerox, the Clearinghouse Protocol used to access product Clearinghouse Services, and the TelePress Protocol used to communicate with the Xerox model 8044 Print Server.

21.1.5 Connecting Networks: Routers and Gateways

When a level one packet is sent from one machine to another, and the two machines are not on the same network, the packet must be passed between networks. Computers that are connected to two or more level zero mediums are used for this function. In the PUP world, these machines have been historically called ''Gateways.'' In the NS world these machines are called Internetwork Routers (Routers), and the function is packaged and sold by Xerox as the Internetwork Routing Service (IRS).

Every host that uses the PUP protocols requires a PUP address; NS Hosts require NS addresses. An address consists of two parts: the host number and the network number. A computer learns its network number by communicating with a Router or Gateway that is attached to the same network. Host number determination is dependent on the hardware and the type of host number, PUP or NS.

21.1.6 Addressing Con icts with Level Zero Mediums

For convenience in the respective protocols, a level one PUP (8-bit) host number is the same as a level zero Experimental Ethernet host number; i.e., when a PUP level one packet is transported by an Experimental Ethernet to another host on the same network, the level zero packet speci es the same host number as the level one packet. Similarly, a level one NS (48-bit) host number is the same as a level zero Ethernet host number.

When a PUP level one packet is transported by an Ethernet, or an NS level one packet is sent on Experimental Ethernet, the level one host number cannot be used as the level zero address, but rather some means must be provided to determine the correct level zero address. Xerox solved this problem by specifying another level-one protocol called *translation* to allow hosts on an Experimental Ethernet to announce their NS host numbers, or hosts on an Ethernet to announce their PUP host numbers. Thus, both the Ethernet and Experimental Ethernet Level Zero Protocols totally support both families of higher level protocols.

21.1.7 References

[1] Robert M. Metcalfe and David R. Boggs, Ethernet: Distributed Packet Switching for Local Computer Networks, *Communications of the ACM* , vol. 19 no. 7, July 1976.

[2] Digital Equipment Corporation, Intel Corporation, Xerox Corporation. The Ethernet, A Local Area Network: Data Link Layer and Physical Layer Speci cations. September 30, 1980, Version 1.0

[3] D. R. Boggs, J. F. Shoch, E. A. Taft, and R. M. Metcalfe, PUP: An Internetwork Architecture, *IEEE Transactions on Communications* , com-28:4, April 1980.

[4] Xerox Corporation. Courier: The Remote Procedure Call Protocol. Xerox System Integration Standard. Stamford, Connecticut, December, 1981, XSIS 038112.

[5] Xerox Corporation. Internet Transport Protocols. Xerox System Integration Standard. Stamford, Connecticut, December, 1981, XSIS 028112.

21.2 HIGHER-LEVEL PUP PROTOCOL FUNCTIONS

This section describes some of the functions provided in Interlisp-D to perform protocols above Level One. Level One functions are described in a later section, for the bene t of those users who wish to program new protocols.

The following functions provide assorted network services.

`(ETHERHOSTNUMBER NAME )` [Function]

Returns the number of the named host. The number is 16-bit quantity, the high 8 bits designating the net and the low 8 bits the host. If `NAME` is `NIL`, returns the number of the local host.

(ETHERPORT NAME ERRORFLG MULTFLG) [Function]

Returns a port corresponding to NAME. A ''port'' is a network address that represents (potentially) one end of a network connection, and includes a socket number in addition to the network and host numbers. Most network functions that take a port as argument allow the socket to be zero, in which case a well-known socket is supplied. A port is currently represented as a dotted pair (NETHOST . SOCKET).

NAME may be a litatom, in which case its address is looked up, or a port, which is just returned directly. If ERRORFLG is true, generates an error ''host not found'' if the address lookup fails, else it returns NIL. If MULTFLG is true, returns a list of alternative port specifications for NAME, rather than a single port (this is provided because it is possible for a single name in the name database to have multiple addresses). If MULTFLG is NIL and NAME has more than one address, the currently nearest one is returned. ETHERPORT caches its results.

The SOCKET of a port is usually zero, unless the name explicitly contains a socket designation, a number or symbolic name following a + in NAME, e.g., PHYLUM+LEAF. A port can also be specified in the form ''net#host#socket'', where each of net, host and socket is a sequence of octal digits; the socket, but not the terminating #, can be omitted, in which case the socket is zero.

(ETHERHOSTNAME PORT USE.OCTAL.DEFAULT) [Function]

Looks up the name of the host at address PORT. PORT may be a numeric address, a (NETHOST . SOCKET) pair returned from ETHERPORT, or a numeric designation in string form, ''net#host#socket'', as described above. In the first case, the net defaults to the local net. If PORT is NIL, returns the name of the local host. If there is no name for the given port, but USE.OCTAL.DEFAULT is true, the function returns a string specifying the port in octal digits, in the form "NET#HOST#SOCKET", with SOCKET omitted if it is zero. Most functions that take a port argument will also accept ports in this octal format.

(PRINTERSTATUS PRINTERNAME) [Function]

Returns status of PRINTERNAME, the name of a Press Printer, in the form (CODE . "readable string"). Returns NIL if the printer does not respond in a reasonable time, which can occur if the printer is very busy, or does not implement the printer status service. CODE is interpreted as follows:

1 Printer is not spooling (down for servicing)

2 Printer is idle

3 Printer is busy (printing or accepting a file)

(EFTP HOST FILE PRINTERFLG #SIDES) [Function]

Transmits FILE to HOST using the EFTP protocol. The FILE need not be open on entry, but in any case is closed on exit. The principal use of the EFTP protocol is for transmitting Press files to a printer. If PRINTERFLG is non-NIL, assumes that HOST is a printer and FILE is a press file, and takes additional action: it performs a PRINTERSTATUS for HOST and prints this information to the prompt window; and it fills in the ''printed-by'' field on the last page of the press file with USERNAME, and the ''copies'' field with (OR (FIXP PRINTERFLG) 1). For printers capable of duplex printing, #SIDES may be 1 or 2, meaning print one- or

two-sided, respectively; NIL means use the printer's default. EFTP returns only on success; if HOST does not respond, it keeps trying.

21.3 HIGHER-LEVEL NS PROTOCOL FUNCTIONS

The following is a description of the Interlisp-D facilities for using Xerox SPP and Courier protocols and the services based on them.

21.3.1 SPP Stream Interface

This section describes the stream interface to the Sequenced Packet Protocol.

(SPP.OPEN HOST SOCKET PROBEP NAME) [Function]

This function is used to open an SPP stream. If HOST is specified, an SPP connection is initiated to HOST with remote socket SOCKET. If both HOST and PROBEP are specified, then the connection is probed for a response before returning the stream; NIL is returned if HOST doesn't respond. If HOST is NIL, a passive connection is created which listens for an incoming connection to local socket SOCKET. NAME is a mnemonic name for the connection process, mainly useful for debugging. The function returns an SPP stream, for which the standard stream operations BIN, BOUT, CLOSEF, and EOFP are defined. In particular, COPYBYTES may be used on SPP streams.

The SPP stream that is returned is open for both input and output, since SPP connections are bidirectional. However, the underlying stream I/O functions use only a single buffer. Some care must therefore be exercised to insure that any buffered output data is forced out before any new data is read, and that all data up to a message boundary has been read before any new data is written. Functions described below are used for this purpose. While these restrictions may seem severe, in practice most use of SPP streams is done by the Courier remote procedure call facility, rather than directly by the programmer. Courier conforms to the model of alternating exchanges of messages quite well.

SPP.USER.TIMEOUT [Variable]

Specifies the time, in milliseconds, to wait before deciding that a host isn't responding.

(SPP.FLUSH STREAM) [Function]

This function forces any buffered output data to be transmitted.

(SPP.SENDEOM STREAM) [Function]

This function forces out any buffered data and causes an End of Message indication to be sent.

(SPP.CLOSE STREAM ABORT?) [Function]

This function closes an SPP stream using the reliable termination protocol. If ABORT? is not NIL, the stream is closed even if there is an outstanding bulk data

transfer in progress.

(SPP.DSTYPE STREAM DSTYPE) [Function]
This function gets or sets the current datastream type. If DSTYPE is specified, all subsequent packets that are sent will be of this datastream type, until the next call to SPP.DSTYPE. Since this affects the current partially-filled packet, the stream should probably be flushed (via SPP.FLUSH) before this function is called. If DSTYPE is not specified, this function returns the datastream type of the current packet being read.

(SPP.READP STREAM) [Function]
This function returns T or NIL depending on whether or not there is data to be read without waiting.

(SPP.EOFP STREAM) [Function]
This function returns T or NIL depending on whether or not the connection has been closed.

(SPP.EOMP STREAM) [Function]
This function returns T or NIL depending on whether or not an End of Message indication has been reached. This will only be true after the last byte of data in the message has been read.

21.3.2 Courier Remote Procedure Call Protocol

(COURIER.OPEN HOSTNAME SERVER TYPE NOERR ORFLG NAME) [Function]
This function opens a Courier connection to the specified HOST and returns an SPP stream. If HOST is a LITATOM, string, or list representation of a Clearinghouse name, SERVER TYPE should specify what type of server HOST is, so that the name may be looked up in the Clearinghouse database. Currently, SERVER TYPE must be one of PRINTSERVER or FILESERVER. Normally, this function will retry the connection \MAXETHERTRIES times before generating an error. If NOERR ORFLG is specified, NIL will be returned if the connection fails. The Courier connection will be given NAME , if specified.

(COURIERPROGRAM NAME) [NLambda NoSpread Function]
This function is used to define Courier programs. The syntax is

```
(COURIERPROGRAM name (programNumber versionNumber)
   TYPES
   ((typeName typeDefinition)
    ...)
   PROCEDURES
   ((procedureName ARGS (argType ...)
                   RESULTS (resultType ...)
                   ERRORS (errorName ...)
                   procedureNumber)
    ...)
   ERRORS
   ((errorName ARGS (argType ...) errorNumber)
```

```
    ...))
)
```

Type de nitions are written in the Courier template language, described below. Courier types may either be type names that are de ned in the current Courier program, quali ed names of the form `(otherCourierProgram . typeName)`, or explicit de nitions in the template language.

21.3.2.1 Courier Template Language

This section describes how Courier types are described in Interlisp, and how corresponding values are represented. (See also the Courier protocol de nition.)

Prede ned types:

`BOOLEAN` is represented by `T` and `NIL`; `STRING` is represented by strings; `CARDINAL`, `INTEGER`, `LONGCARDINAL`, `LONGINTEGER`, and `UNSPECIFIED` are represented by integers.

Constructed types:

```
(ENUMERATION (NAME VALUE) ... (NAME VALUE))
(ARRAY LENGTH TYPE)
(SEQUENCE TYPE)
(RECORD (NAME TYPE) ... (NAME TYPE))
(CHOICE (NAME VALUE TYPE) ... (NAME VALUE TYPE))
```

Representation of constructed types in Lisp:

Objects of Courier type `(ENUMERATION (UNKNOWN 0) (RED 1) (BLUE 2))` are represented by the litatoms `UNKNOWN`, `RED`, and `BLUE`.

Objects of Courier type `(ARRAY 3 INTEGER)` are represented by lists of three integers, such as `(10 1 59)`.

Objects of Courier type `(SEQUENCE BOOLEAN)` are represented by arbitrary- length lists of `T` and `NIL`, such as `(NIL T T NIL T)`.

Objects of Courier type

```
(RECORD (NETWORK LONGCARDINAL)
        (HOST (ARRAY 3 CARDINAL))
        (SOCKET CARDINAL))
```

are represented by lists like `((NETWORK 174) (HOST (100 24 363)) (SOCKET 20))`.

Objects of Courier type

```
(CHOICE (STATUS 0 (ENUMERATION (BUSY 0) (COMPLETE 1)))
        (MESSAGE 1 STRING))
```

are represented by lists like `(STATUS COMPLETE)` or `(MESSAGE "Your request has completed.")`.

(COURIER.CALL STREAM PROGRAM PROCEDURE ARG$_1$ ARG$_N$ NOERR ORFLG)

[NoSpread Function]

This function calls the remote procedure PROCEDURE of the Courier program PROGRAM. STREAM is the SPP stream returned by COURIER.OPEN. The arguments should be Lisp values appropriate for the Courier types of the corresponding formal parameters of the procedure (defined under the ARGS property for the procedure). Returns results of the Courier types defined under the RESULTS property. If there is only a single result, it is returned, otherwise a list of results is returned. The NOERR ORFLG argument controls the treatment of remote errors. If NOERR ORFLG is NIL, a Lisp error will be generated. If NOERR ORFLG is T, NIL will be returned as the result of the call. If NOERR ORFLG is RETURNERRORS, the result of the call will be a list consisting of the atom ERROR followed by the Courier name of the error and any arguments.

Examples:

```
(COURIERPROGRAM EXAMPLEPROGRAM (17 1)
   TYPES
   ((PERSON.NAME (RECORD (FIRST.NAME STRING)
                         (MIDDLE (CHOICE
                                   (NAME 0 STRING)
                                   (INITIAL 1 STRING)))
                         (LAST.NAME STRING)))
    (BIRTHDAY (RECORD (YEAR CARDINAL)
                      (MONTH STRING)
                      (DAY CARDINAL))))
   PROCEDURES
   ((GETBIRTHDAY ARGS (PERSON.NAME)
                 RESULTS (BIRTHDAY)
                 3))
)
```

Defines EXAMPLEPROGRAM to be Courier program number 17, version number 1. The example defines two types, PERSON.NAME and BIRTHDAY, and one procedure, GETBIRTHDAY, whose procedure number is 3. The following code could be used to call the remote GETBIRTHDAY procedure on the host with address HOSTADDRESS.

```
(SETQ STREAM (COURIER.OPEN HOSTADDRESS))
(COURIER.CALL STREAM
              (QUOTE EXAMPLEPROGRAM)
              (QUOTE GETBIRTHDAY)
              (QUOTE ((FIRST.NAME "Eric")
                      (MIDDLE (INITIAL "C"))
                      (LAST.NAME "Cooper"))))
```

COURIER.CALL in this example will return a value such as

```
((YEAR 1959) (MONTH "January") (DAY 10))
```

21.3.2.2 Manipulating Courier Representations

Several Courier programs use values of type (SEQUENCE UNSPECIFIED) to handle user-defined or otherwise extensible object types. Often it is necessary to convert between a list of 16 bit words (the sequence of UNSPECIFIEDs) and a Courier value. The following function should be used for this purpose.

(COURIER.READ.REP LIST.OF.WORDS PROGRAM TYPE) [Function]
This function returns the Lisp representation of the Courier object of type TYPE defined in the Courier program PROGRAM whose underlying Courier representation is LIST.OF.WORDS .

21.3.2.3 Using Bulk Data Transfer with Courier

Two Courier types are treated specially when they appear in the argument list of a procedure. They are BULK.DATA.SINK and BULK.DATA.SOURCE. A Courier procedure may have at most one such sink or source parameter. The result of a COURIER.CALL on such a procedure is an SPP stream, open for input or output according to whether the bulk data paramter is a sink or a source. The client uses this stream to receive or send the appropriate bulk data object. If the object consists of bytes, this may be done with the usual stream I/O functions such as COPYBYTES. If the data is a stream of Courier objects, the following function should be used.

(COURIER.READ.BULKDATA STREAM PROGRAM TYPE) [Function]
STREAM is the bulk data stream returned from COURIER.CALL. TYPE is the type of each Courier object in the stream. PROGRAM is the Courier program in which TYPE is defined. A list of objects of Courier type TYPE will be returned.

The observant reader may wonder what happens if the Courier procedure returns one or more results, in addition to taking a bulk data parameter. If a bulk data stream is returned to the caller, what happens to the results? The answer is that the results are collected when the bulk data stream is closed, after the client has transferred the bulk data. The disposition of these results depends on what actual parameter is supplied for the formal bulk data parameter at the time of the call. If it is NIL, the results, if any, will be ignored. Otherwise, the value is assumed to be a function which to be applied to the results. A FUNARG may be used for full generality.

For example, the Courier procedure to print an Interpress master uses a bulk data source to transfer the master, and also returns a request identifier. The Lisp function which performs the COURIER.CALL passes a functional to be called on this request identifier after the stream is closed and printing begins; this functional in turn spawns a process which monitors the progress of the job.

(COURIERTRACE FLG REGION) [Function]
This function controls the tracing of Courier remote procedure calls. It is similar to PUPTRACE and XIPTRACE, but operates at the call/return level rather than the packet level.

21.3.3 NS Printing

This section describes the facilities that are available for printing Interpress masters on NS printservers.

NS.DEFAULT.PRINTER [Variable]

The value of this variable is used whenever no printserver is specified for the functions described below. If its value is a LITATOM, string, or Clearinghouse name, the Clearinghouse is queried to find the address of the printserver with that name. If its value is NIL, it will be set automatically to some printserver in the local Clearinghouse domain. In environments where there is no Clearinghouse, the value of NS.DEFAULT.PRINTER must be an appropriate NSADDRESS record.

(OPEN.NS.PRINTING.STREAM PRINTER DOCUMENT.NAME DOCUMENT.CREATION.DATE SENDER.NAME RECIPIENT.NAME qCOPIES MEDIUM PRIORITY STAPLE? TWOSIDED? NOWATCHDOG?) [Function]

This function returns a stream for printing an Interpress master on PRINTER or on NS.DEFAULT.PRINTER as mentioned above. The caller should write the Interpress data to the stream and then close it using CLOSEF. Printing begins after the stream is closed.

DOCUMENT.NAME is the document name to appear on the header page (a string).

DOCUMENT.CREATION.DATE is the creation date to appear on the header page (a Lisp integer date). The default value is the time of the call.

SENDER.NAME is the name of the sender to appear on the header page (a string). The default value is the name of the user.

RECIPIENT.NAME is the name of the recipient to appear on the header page (a string). The default value is the name of the user.

qCOPIES is the number of copies to be printed. The default value is 1.

MEDIUM is the medium on which the master is to be printed. This must be a Courier value of type MEDIUM, which is a list of the form (PAPER (KNOWN.SIZE NAME)), where NAME is one of the LITATOMs US.LETTER, US.LEGAL, A0 through A10, ISO.B0 through ISO.B10, and JIS.B0 through JIS.B10. The default value is determined by the printer.

PRIORITY is the priority of this print request (LOW, NORMAL, or HIGH). The default value is NORMAL.

STAPLE? is T or NIL depending on whether the document should be stapled. The default value is NIL.

TWOSIDED? is T or NIL depending on whether the document should be printed on two sides. The default value is NIL.

NOWATCHDOG? is non-NIL if the client does not want a watchdog process to monitor the status of the printing job.

(NSPRINT PRINTER FILE.NAME DOCUMENT.NAME DOCUMENT.CREATION.DATE SENDER.NAME RECIPIENT.NAME qCOPIES MEDIUM PRIORITY STAPLE? TWOSIDED?) [Function]

This function prints an Interpress master on PRINTER or on NS.DEFAULT.PRINTER as mentioned above. FILE.NAME should be the name of an Interpress file to be printed. The remaining arguments are all optional, and are as described for OPEN.NS.PRINTING.STREAM above. DOCUMENT.NAME defaults to the full name of the file, and DOCUMENT.CREATION.DATE defaults to the creation date of

the file.

(NSPRINTER.STATUS PRINTER) [Function]
This function returns the Courier value resulting from the GET.PRINTER.STATUS call.

(NSPRINTER.PROPERTIES PRINTER) [Function]
This function returns the Courier value resulting from the GET.PRINTER.PROPERTIES call.

21.3.4 Clearinghouse

This section describes functions that may be used to access Clearinghouse servers. Note that these functions are used by the NS printing functions if the printserver is specified by name rather than address.

(START.CLEARINGHOUSE REST AR TFL G) [Function]
This function enables Clearinghouse access. It performs an expanding ring broadcast in order to find the first Clearinghouse server. If REST AR TFL G is non-NIL, the cache of Clearinghouse information is invalidated and a new broadcast is done. This may be necessary if the local Clearinghouse server goes down.

CH.NET.HINT [Variable]
Hint as to which network the local Clearinghouse server is on, for use by START.CLEARINGHOUSE above. If CH.NET.HINT is bound to a network number, that network will be tried first, followed by the others in the routing table. If the local Clearinghouse server is not on the directly connected network, setting CH.NET.HINT to the proper network number in the local INIT file will speed up START.CLEARINGHOUSE considerably.

(SHOW.CLEARINGHOUSE) [Function]
This function displays the structure of the cached Clearinghouse information in a window. Once created, it will be redisplayed whenever the cache is updated. The structure is shown using GRAPHER‘‘.

(SHOW.ENTIRE.CLEARINGHOUSE) [Function]
This function attempts to cache information about all the Clearinghouse domains, so that the Clearinghouse structure window will show the entire database.

CH.DEFAULT.DOMAIN [Variable]
This is a string specifying the default Clearinghouse domain. If it is NIL, it will be set automatically by START.CLEARINGHOUSE. Otherwise, it should be set in an INIT file.

CH.DEFAULT.ORGANIZATION [Variable]
This is a string specifying the default Clearinghouse organization. If it is NIL, it will be set automatically by START.CLEARINGHOUSE. Otherwise, it should be set in an INIT file.

(CH.ORGANIZATIONS OR GANIZA TIONP ATTERN) [Function]
This function returns the list of organization names in the Clearinghouse database matching OR GANIZA TIONP ATTERN. The default pattern is "*", which matches

anything.

(CH.DOMAINS DOMAINP ATTERN) [Function]
This function returns the list of domain names in the Clearinghouse database matching DOMAINP ATTERN . The default pattern is "*", which matches anything.

(CH.ENUMERATE OBJECTP ATTERN PR OPER TY) [Function]
This function returns the list of object names matching OBJECTP ATTERN and having the property PR OPER TY . Currently, PR OPER TY must be one of USER, PRINTSERVER, FILESERVER, and ALL. For example,

```
(CH.ENUMERATE "*:PARC:Xerox" (QUOTE USER))
```

will return a list of the names of users at Xerox PARC.

(CH.LOOKUP.USER NAME) [Function]
This function returns the user information for the rst user whose name matches NAME .

(LOOKUP.NS.SERVER NAME TYPE) [Function]
This function returns the NSADDRESS for the rst server whose name matches NAME and has the property TYPE , which must be PRINTSERVER or FILESERVER.

21.3.5 NS Filing

This section describes functions that may be used to access NS leservers.

21.3.5.1 Pathnames and NS Fileservers

The NS Filing protocol does not support conventional le system pathnames directly. However, the Interlisp- D software that supports access to NS leservers uses IFS- style pathnames and does the appropriate mapping in software. One important di erence, however, is that leserver, directory, and le names may have spaces in them, each of which must be preceded by a percent sign. The name of an NS leserver is required to have a colon in it. Thus, even if the leserver is in the local Clearinghouse domain, a trailing colon should be appended to the name. Case is not signi cant. For example,

```
{LISPFILE:}<LISPDRAWER>XYZ;3
```

is a valid name for a le on the NS leserver "LispFile:Parc Place:Xerox".

(NSDIRECTORY PATTERN) [Function]
This function returns a list of le names in PATTERN , which must be the NS pathname for a directory. (Any wildcards in the name eld of the pathname are ignored.)

(NSCREATEDIRECTORY HOST/DIR) [Function]
This function creates a new directory with pathname HOST/DIR . Top level directories (‘‘ le drawers’’) cannot be created in this way.

(CLOSE.NSFILING.CONNECTIONS) [Function]
This function closes any open connections to NS leservers.

21.4 LEVEL ONE ETHER PACKET FORMAT

The datatype ETHERPACKET is the vehicle for all kinds of packets transmitted on an Ethernet or Experimental Ethernet. An ETHERPACKET contains several elds for use by the Ethernet drivers and a large, contiguous data area making up the data of the level zero packet. The rst several words of the area are reserved for the level one to zero encapsulation, and the remainder (starting at eld EPBODY) make up the level one packet. Typically, each level one protocol de nes a BLOCKRECORD that overlays the ETHERPACKET starting at the EPBODY eld, describing the format of a packet for that particular protocol. For example, the records PUP and XIP de ne the format of level one packets in the PUP and NS protocols.

The extra elds in the beginning of an ETHERPACKET have mostly a xed interpretation over all protocols. Among the interesting ones are:

EPLINK — A pointer used to link packets, used by the SYSQUEUE mechanism (page 21.25). Since this eld is used by the system for maintaining the free packet queue and ether transmission queues, do not use this eld unless you understand it.

EPFLAGS — A byte eld that can be used for any purpose by the user.

EPUSERFIELD — A pointer eld that can be used for any purpose by the user. It is set to NIL when a packet is released.

EPTRANSMITTING — A ag that is true while the packet is ‘‘being transmitted’’, i.e., from the time that the user instructs the system to transmit the packet until the packet is gathered up from the transmitter’s nished queue. While this ag is true, the user must *not* modify the packet.

EPREQUEUE — A pointer eld that speci es the desired disposition of the packet after transmission. The possible values are: NIL means no special treatment; FREE means the packet is to be released after transmission; an instance of a SYSQUEUE means the packet is to be enqueued on the speci ed queue (page 21.25).

The normal life of an outgoing Ether packet is that a program obtains a blank packet, lls it in according to protocol, then sends the packet over the Ethernet. If the packet needs to be retained for possible retransmission, the EPREQUEUE eld is used to specify a queue to place the packet on after its transmission, or the caller hangs on to the packet explicitly.

There are rede nitions, or ‘‘overlays’’ of the ETHERPACKET record speci cally for use with the PUP and NS protocols. The following sections describe those records and the handling of the PUP and NS level one protocols, how to add new level one protocols, and the queueing mechanism associated with the EPREQUEUE eld.

21.5 PUP LEVEL ONE FUNCTIONS

The functions in this section are used to implement level two and higher PUP protocols. That is, they deal with sending and receiving PUP packets. It is assumed the reader is familiar with the format and use of pups, e.g., from reading reference [3] in section 21.1.7.

(RESTART.ETHER) [Function]

This function is intended to be invoked from the executive on those rare occasions when the Ethernet appears completely unresponsive, due to Lisp having gotten into a bad state. RESTART.ETHER reinitializes Lisp's Ethernet driver(s), just as when the Lisp system is started up following a LOGOUT, SYSOUT, etc. This aborts any Ethernet activity and clears several internal caches, including the routing table.

21.5.1 Creating and Managing Pups

There is a record PUP that overlays the data portion of an ETHERPACKET and describes the format of a pup. This record defines the following numeric fields: PUPLENGTH (16 bits), TCONTROL (transmit control, 8 bits, cleared when a PUP is transmitted), PUPTYPE (8 bits), PUPID (32 bits), PUPIDHI and PUPIDLO (16 bits each overlaying PUPID), PUPDEST (16 bits overlayed by 8-bit fields PUPDESTNET and PUPDESTHOST), PUPDESTSOCKET (32 bits, overlayed by 16-bit fields PUPDESTSOCKETHI and PUPDESTSOCKETLO), and PUPSOURCE, PUPSOURCENET, PUPSOURCEHOST, PUPSOURCESOCKET, PUPSOURCESOCKETHI, and PUPSOURCESOCKETLO, analagously. The field PUPCONTENTS is a pointer to the start of the data portion of the pup.

(ALLOCATE.PUP) [Function]

Returns a (possibly used) pup. Keeps a free pool, creating new pups only when necessary. The pup header fields of the pup returned are guaranteed to be zero, but there may be garbage in the data portion if the pup had been recycled, so the caller should clear the data if desired.

(CLEARPUP PUP) [Function]

Clears *all* information from PUP, including the pointer fields of the ETHERPACKET and the pup data portion.

(RELEASE.PUP PUP) [Function]

Releases PUP to the free pool.

21.5.2 Sockets

Pups are sent and received on a *socket*. Generally, for each ''conversation'' between one machine and another, there is a distinct socket. When a pup arrives at a machine, the low-level pup software examines the pup's destination socket number. If there is a socket on the machine with that number, the incoming pup is handed over to the socket; otherwise the incoming pup is discarded. When a *user* process initiates a conversation, it generally selects a large, random socket number different from any other in use on the machine. A *server* process, on the other hand, provides a specific service at a ''well-known'' socket, usually a fairly small number. In the PUP world, advertised sockets are in the range 0 to 100Q.

(OPENPUPSOCKET SKTQ IFCLASH) [Function]

Opens a new pup socket. If SKTQ is NIL (the normal case), a socket number is chosen automatically, guaranteed to be unique, and probably different from any socket opened this way in the last 18 hours (the low half of the time of day clock is sampled).

If a specific local socket is desired, as is typically the case when implementing a server, SKTQ is given, and must be a (up to 32-bit) number. IFCLASH indicates what to do in the case that the designated socket is already in use: if NIL, an error is generated; if ACCEPT, the socket is quietly returned; if FAIL, then OPENPUPSOCKET returns NIL without causing an error. Note that ''well-known'' socket numbers should be avoided unless the caller is actually implementing one of the services advertised as provided at the socket.

(CLOSEPUPSOCKET PUPSOC NOERRORFLG) [Function]

Closes and releases socket PUPSOC . If PUPSOC is T, closes all pup sockets (this must be used with caution, since it will also close system sockets!). If PUPSOC is already closed, an error is generated unless NOERRORFLG is true.

(PUPSOCKETNUMBER PUPSOC) [Function]

Returns the socket number (a 32-bit integer) of PUPSOC .

(PUPSOCKETEVENT PUPSOC) [Function]

Returns the EVENT of PUPSOC (page 18.30). This event is noti ed whenever a pup arrives on PUPSOC , so pup clients can perform an AWAIT.EVENT on this event if they have nothing else to do at the moment.

21.5.3 Sending and Receiving Pups

(SENDPUP PUPSOC PUP) [Function]

Sends PUP on socket PUPSOC . If any of the PUPSOURCESHOST, PUPSOURCENET, or PUPSOURCESOCKET elds is zero, SENDPUP lls them in using the pup address of this machine and/or the socket number of PUPSOC , as needed.

(GETPUP PUPSOC WAIT) [Function]

Returns the next pup that has arrived addressed to socket PUPSOC . If there are no pups waiting on PUPSOC , then GETPUP returns NIL, or waits for a pup to arrive if WAIT is T. If WAIT is an integer, GETPUP interprets it as a number of milliseconds to wait, nally returning NIL if a pup does not arrive within that time.

(DISCARDPUPS SOC) [Function]

Discards without examination any pups that have arrived on SOC and not yet been read by a GETPUP.

(EXCHANGEPUPS SOC OUTPUP DUMMY IDFILTER TIMEOUT) [Function]

Sends OUTPUP on SOC , then waits for a responding pup, which it returns. If IDFILTER is true, ignores pups whose PUPID is different from that of OUTPUP . TIMEOUT is the length of time (msecs) to wait for a response before giving up and returning NIL. TIMEOUT defaults to \ETHERTIMEOUT. EXCHANGEPUPS discards without examination any pups that are currently waiting on SOC before OUTPUP gets

sent. (DUMMY is ignored; it exists for compatibility with an earlier implementation).

21.5.4 Pup Routing Information

Ordinarily, a program calls SENDPUP and does not worry at all about the route taken to get the pup to its destination. There is an internet routing process in Lisp whose job it is to maintain information about the best routes to networks of interest. However, there are some algorithms for which routing information and/or the topology of the net are explicitly desired. To this end, the following functions are supplied:

(PUPNET.DISTANCE NET q) [Function]

Returns the ''hop count'' to network NET q, i.e., the number of gateways through which a pup must pass to reach NET q, according to the best routing information known at this point. The local (directly-connected) network is considered to be zero hops away. Current convention is that an inaccessible network is 16 hops away. PUPNET.DISTANCE may need to wait to obtain routing information from an Internetwork Router if NET q is not currently in its routing cache.

(SORT.PUPHOSTS.BY.DISTANCE HOSTLIST) [Function]

Sorts HOSTLIST by increasing distance, in the sense of PUPNET.DISTANCE. HOSTLIST is a list of lists, the CAR of each list being a 16-bit Net/Host address, such as returned by ETHERHOSTNUMBER. In particular, a list of ports ((nethost . socket) pairs) is in this format.

(PRINTROUTINGTABLE TABLE SORT FILE) [Function]

Prints to FILE the current routing cache. The table is sorted by network number if SORT is true. TABLE = PUP (the default) prints the PUP routing table; TABLE = NS prints the NS routing table..

21.5.5 Miscellaneous PUP Utilities

(SETUPPUP PUP DESTHOST DESTSOCKET TYPE ID SOC REQUEUE) [Function]

Fills in various elds in PUP's header: its length (the header overhead length; assumes data length of zero), TYPE, ID (if ID is NIL, generates a new one itself from an internal 16-bit counter), destination host and socket (DESTHOST may be anything that ETHERPORT accepts; an explicit nonzero socket in DESTHOST overrides DESTSOCKET). If SOC is not supplied, a new socket is opened. REQUEUE lls the packets EPREQUEUE eld (see above). Value of SETUPPUP is the socket.

(SWAPPUPPORTS PUP) [Function]

Swaps the source and destination addresses in PUP. This is useful in simple packet exchange protocols, where you want to respond to an input packet by diddling the data portion and then sending the pup back whence it came.

(GETPUPWORD PUP WORD q) [Function]

Returns as a 16-bit integer the contents of the WORD qth word of PUP's data portion, counting the rst word as word zero.

(PUTPUPWORD PUP WORD q VALUE) [Function]

Stores 16-bit integer VALUE in the WORD qth word of PUP's data portion.

(GETPUPBYTE PUP BYTE q) [Function]
Returns as an integer the contents of the BYTE q th 8-bit byte of PUP 's data portion, counting the rst byte as byte zero.

(PUTPUPBYTE PUP BYTE q VAL UE) [Function]
Stores VAL UE in the BYTE q th 8-bit byte of PUP 's data portion.

(GETPUPSTRING PUP OFFSET) [Function]
Returns a string consisting of the characters in PUP 's data portion starting at byte OFFSET (default zero) through the end of PUP .

(PUTPUPSTRING PUP STR) [Function]
Appends STR to the data portion of PUP , incrementing PUP 's length appropriately.

21.5.6 PUP Debugging Aids

Tracing facilities are provided to allow the user to see the pup tra c that passes through SENDPUP and GETPUP. The tracing can be verbose, displaying much information about each packet, or terse, which shows a concise ''picture'' of the tra c.

PUPTRACEFLG [Variable]
Controls tracing information provided by SENDPUP and GETPUP. Legal values:

NIL No tracing.

T Every SENDPUP and every successful GETPUP call PRINTPUP of the pup at hand (see below).

PEEK Allows a concise ''picture'' of the tra c. For normal, non-broadcast packets, SENDPUP prints ''!'', GETPUP prints ''+''. For broadcast packets, SENDPUP prints ''^'', GETPUP prints ''*''. In addition, for packets that arrive not addressed to any socket on this machine (e.g., broadcast packets for a service not implemented on this machine), a ''&'' is printed.

PUPIGNORETYPES [Variable]
A list of pup types (small integers). If the type of a pup is on this list, then GETPUP and SENDPUP will not print the pup verbosely, but treat it as though PUPTRACEFLG were PEEK. This allows the user to lter out ''uninteresting'' pups, e.g., routine routing information pups (type 201Q).

PUPONLYTYPES [Variable]
A list of pup types. If this variable is non-NIL, then GETPUP and SENDPUP print verbosely *only* pups whose types appear on the list, treating others as though PUPTRACEFLG were PEEK. This lets the tracing be con ned to only a certain class of pup tra c.

PUPTRACEFILE [Variable]
The le to which pup tracing output is sent by default. The le must be open. PUPTRACEFILE is initially T.

PUPTRACETIME [Variable]

If this variable is true, then each printout of a pup is accompanied by a relative timestamp (in seconds, with 2 decimal places) of the current time (i.e., when the SENDPUP or GETPUP was called; for incoming pups, this is not the same as when the pup actually arrived).

(PUPTRACE FLG REGION) [Function]

Creates a window for puptrac ing, and sets PUPTRACEFILE to it. If PUPTRACEFILE is currently a window and FLG is NIL, closes the window. Sets PUPTRACEFLG to be FLG. If REGION is supplied, the window is created with that region. The window's BUTTONEVENTFN is set to cycle PUPTRACEFLG through the values NIL, T, and PEEK when the mouse is clicked in the window.

(PRINTPUP PACKET CALLER FILE PRE.NOTE DOFIL TER) [Function]

Prints the information in the header and possibly data portions of pup PACKET to FILE. If CALLER is supplied, it identi es the direction of the pup (GET or PUT), and is printed in front of the header. FILE defaults to PUPTRACEFILE. If PRE.NOTE is non-NIL, it is PRIN1'ed rst. If DOFIL TER is true, then if PUP 's type fails the ltering criteria of PUPIGNORETYPES or PUPONLYTYPES, then PUP is printed ''tersely'', i.e., as a !, +, ^, or *, as described above.

GETPUP and SENDPUP, when PUPTRACEFLG is non-NIL, call (PRINTPUP PUP {'GET or 'PUT} NIL NIL T).

The form of printing provided by PRINTPUP can be in uenced by adding elements to PUPPRINTMACROS.

PUPPRINTMACROS [Variable]

An association list of elements (PUPTYPE . MA CR O) for printing pups. The MA CR O (CDR of each element) tells how to print the information in a pup of type PUPTYPE (CAR of the element). If MA CR O is a litatom, then it is a function of two arguments (PUP FILE) that is applied to the pup to do the printing. Otherwise, MA CR O is a list describing how to print the data portion of the pup (the header is printed in a standard way).

The list form of MA CR O consists of ''commands'' that specify a ''datatype'' to interpret the data, and an indication of how far that datatype extends in the packet. Each element of MA CR O is one of the following: (a) a byte o set (positive integer), indicating the byte at which the next element, if any, takes e ect; (b) a negative integer, the absolute value of which is the number of bytes until the next element, if any, takes e ect; or (c) an atom giving the format in which to print the data, one of the following:

BYTES — Print the data as 8-bit bytes, enclosed in brackets. This is the default format to start with.

CHARS — Print the data as (8-bit) characters. Non-printing characters are printed as if the format were BYTES, except that the sequence 15Q, 12Q is printed specially as [crlf].

WORDS — Print the data as 16-bit integers, separated by commas (or the current SEPR).

`INTEGERS` Print the data as 32-bit integers, separated by commas (or the current `SEPR`). Note: the singular `BYTE`, `CHAR`, `WORD`, `INTEGER` are accepted as synonyms for these four commands.

`SEPR` Set the separator for `WORDS` and `INTEGERS` to be the next element of the macro. The separator is initially the two characters, comma, space.

`IFSSTRING` Interprets the data as a 16-bit length followed by that many 8-bit bytes or characters. If the current datatype is `BYTES`, leaves it alone; otherwise, sets it to be `CHARS`.

`...` If there is still data left in the packet by the time processing reaches this command, prints ‘‘ ’’ and stops.

`FINALLY` The next element of the macro is printed when the end of the packet is reached (or printing stops because of a). This command does not alter the datatype, and can appear anywhere in the macro as long as it is encountered before the actual end of the packet.

`T` Perform a `TERPRI`.

`REPEAT` The remainder of the macro is itself treated as a macro to be applied over and over until the packet is exhausted. Note that the o sets speci ed in the macro must be in the relative form, i.e., negative integers. For example, the macro `(INTEGERS 4 REPEAT BYTES -2 WORDS -4)` says to print the rst 4 bytes of the data as one 32-bit integer, then print the rest of the data as sets of 2 8-bit bytes and 2 16-bit words.

Only as much of the macro is processed as is needed to print the data in the given packet. The default macro for printing a pup is `(BYTES 12 ...)`, meaning to print the rst up to 12 bytes as bytes, and then print ‘‘ ’’ if there is anything left.

The following functions are used by `PRINTPUP` and similar functions, and may be of interest in special cases.

`(PORTSTRING NETHOST SOCKET )` [Function]
Converts the pup address `NETHOST` , `SOCKET` into the following octal string format: net#host#socket. `NETHOST` may be a port (dotted pair of nethost and socket), in which case `SOCKET` is ignored, and the socket portion of `NETHOST` is omitted from the string if it is zero.

`(PRINTPUPROUTE PACKET CALLER FILE)` [Function]
Prints the source and destination addresses of pup `PACKET` to `FILE` in the `PORTSTRING` format, preceded by `CALLER` (interpreted as with `PRINTPUP`).

`(PRINTPACKETDATA BASE OFFSET MA CR O LENGTH FILE)` [Function]
Prints data according to `MA CR O`, which is a list interpreted as described under

PUPPRINTMACROS, to FILE. The data starts at BASE and extends for LENGTH bytes. The actual printing starts at the OFFSET th byte, which defaults to zero. For example, PRINTPUP ordinarily calls (PRINTPACKETDATA (fetch PUPCONTENTS of PUP) 0 MACRO (IDIFFERENCE (fetch PUPLENGTH of PUP) 20) FILE).

(PRINTCONSTANT VAR CONSTANTLIST FILE PREFIX) [Function]

CONSTANTLIST is a list of pairs (VARNAME VALUE), of the form given to the CONSTANTS File Package Command. PRINTCONSTANT prints VAR to FILE, followed in parentheses by the VARNAME out of CONSTANTLIST whose VALUE is EQ to VAR, or ? if it finds no such element. If PREFIX is non-NIL and is an initial substring of the selected VARNAME, then VARNAME is printed without the prefix.

For example, if FOOCONSTANTS is ((FOO.REQUEST 1) (FOO.ANSWER 2) (FOO.ERROR 3)), then (PRINTCONSTANT 2 FOOCONSTANTS T "FOO.") produces ''2 (ANSWER)''.

(OCTALSTRING N) [Function]

Returns a string of octal digits representing N in radix 8.

21.6 NS LEVEL ONE FUNCTIONS

The functions in this section are used to implement level two and higher NS protocols. The packets used in the NS protocol are termed Xerox Internet Packets (XIPs). The functions for manipulating XIPs are similar to those for managing PUPs, so will be described in less detail here. The major difference is that NS host addresses are 48-bit numbers. Since Interlisp-D cannot currently represent 48-bit numbers directly as integers, there is an interim form called NSHOSTNUMBER, which is defined as a TYPERECORD of three fields, each of them being a 16-bit portion of the 48-bit number.

21.6.1 Creating and Managing XIPs

There is a record XIP that overlays the data portion of an ETHERPACKET and describes the format of a XIP. This record defines the following fields: XIPLENGTH (16 bits), XIPTCONTROL (transmit control, 8 bits, cleared when a XIP is transmitted), XIPTYPE (8 bits), XIPDESTNET (32 bits), XIPDESTHOST (an NSHOSTNUMBER), XIPDESTSOCKET (16 bits), and XIPSOURCENET, XIPSOURCEHOST, and XIPSOURCESOCKET, analagously. The field XIPCONTENTS is a pointer to the start of the data portion of the XIP.

(ALLOCATE.XIP) [Function]

Returns a (possibly used) XIP. As with ALLOCATE.PUP, the header fields are guaranteed to be zero, but there may be garbage in the data portion if the pup had been recycled.

(RELEASE.XIP XIP) [Function]

Releases XIP to the free pool.

21.6.2 NS Sockets

As with pups, XIPs are sent and received on a *socket*. The same comments apply as with pup sockets (page 21.16), except that NS socket numbers are only 16 bits.

(OPENNSOCKET SKT q IFCLASH) [Function]
Opens a new NS socket. If SKT q is NIL (the normal case), a socket number is chosen automatically, guaranteed to be unique, and probably di erent from any socket opened this way in the last 18 hours. If a speci c local socket is desired, as is typically the case when implementing a server, SKT q is given, and must be a (up to 16-bit) number. IFCLASH governs what to do if SKT q is already in use, as with OPENPUPSOCKET.

(CLOSENSOCKET NSOC NOERR ORFL G) [Function]
Closes and releases socket NSOC . If NSOC is T, closes all NS sockets (this must be used with caution, since it will also close system sockets!). If NSOC is already closed, an error is generated unless NOERR ORFL G is true.

(NSOCKETNUMBER NSOC) [Function]
Returns the socket number (a 16-bit integer) of NSOC .

(NSOCKETEVENT NSOC) [Function]
Returns the EVENT of NSOC . This event is noti ed whenever a XIP arrives on NSOC .

21.6.3 Sending and Receiving XIPs

(SENDXIP NSOC XIP) [Function]
Sends XIP on socket NSOC . If any of the XIPSOURCESHOST, XIPSOURCENET, or XIPSOURCESOCKET elds is zero, SENDXIP lls them in using the NS address of this machine and/or the socket number of NSOC , as needed.

(GETXIP NSOC W AIT) [Function]
Returns the next XIP that has arrived addressed to socket NSOC . If there are no XIPs waiting on NSOC , then GETXIP returns NIL, or waits for a XIP to arrive if W AIT is T. If W AIT is an integer, GETXIP interprets it as a number of milliseconds to wait, nally returning NIL if a XIP does not arrive within that time.

(DISCARDXIPS NSOC) [Function]
Discards without examination any XIPs that have arrived on NSOC and not yet been read by a GETXIP.

(EXCHANGEXIPS SOC OUTXIP IDFILTER TIMEOUT) [Function]
Useful for simple NS packet exchange protocls. Sends OUTXIP on SOC , then waits for a responding XIP, which it returns. If IDFILTER is true, ignores XIPs whose packet exchange ID (the rst 32 bits of the data portion) is di erent from that of OUTXIP . TIMEOUT is the length of time (msecs) to wait for a response before giving up and returning NIL. TIMEOUT defaults to \ETHERTIMEOUT. EXCHANGEXIPS discards without examination any XIPs that are currently waiting on SOC before OUTXIP gets sent.

21.6.4 NS Debugging Aids

XIPs can be printed automatically by SENDXIP and GETXIP analogously to the way pups are. The following variables behave with respect to XIPs the same way that the corresponding PUP-named variables behave with respect to PUPs: XIPTRACEFLG, XIPTRACEFILE, XIPIGNORETYPES, XIPONLYTYPES, XIPPRINTMACROS. In addition, the functions PRINTXIP, PRINTXIPROUTE and XIPTRACE are directly analogous to PRINTPUP, PRINTPUPROUTE, and PUPTRACE.

21.7 SUPPORT FOR OTHER LEVEL ONE PROTOCOLS

Raw packets other than of type PUP or NS can also be sent and received. This section describes facilities to support such protocols. Many of these functions have a \ in their names to designate that they are system internal, not to be dealt with as casually as user-level functions.

(\ALLOCATE.ETHERPACKET) [Function]

Returns an ETHERPACKET datum. Enough of the packet is cleared so that if the packet represents a PUP or NS packet, that its header is all zeros; no guarantee is made about the remainder of the packet.

(\RELEASE.ETHERPACKET EPKT) [Function]

Returns EPKT to the pool of free packets. This operation is dangerous if the caller actually is still holding on to EPKT , e.g., in some queue, since this packet could be returned to someone else (via \ALLOCATE.ETHERPACKET) and su er the resulting contention.

From a logical standpoint, programs need never call \RELEASE.ETHERPACKET , since the packets are eventually garbage-collected after all pointers to them drop. However, since the packets are so large, normal garbage collections tend not to occur frequently enough. Thus, for best performance, a well-disciplined program should explicitly release packets when it knows it is nished with them.

A locally-connected network for the transmission and receipt of Ether packets is speci ed by a *network descriptor block*, an object of type NDB. There is one NDB for each directly-connected network; ordinarily there is only one. The NDB contains information speci c to the network, e.g., its PUP and NS network numbers, and information about how to send and receive packets on it.

\LOCALNDBS [Variable]

The rst NDB connected to this machine, or NIL if there is no network. Any other NDBs are linked to this rst one via the NDBNEXT eld of the NDB.

In order to transmit an Ether packet, a program must specify the packet's type and its immediate destination. The type is a 16-bit integer identifying the packet's protocol. There are preassigned types for PUP and NS. The destination is a host address on the local network, in whatever form the local network uses for addressing; it is not necessarily related to the logical ultimate destination of the packet. Determining the immediate destination of a packet is the task of *routing*. The functions SENDPUP and SENDXIP take care of this for the PUP and NS protocols, routing a packet directly to its destination if that host is on the local network, or routing it to a gateway if the host is on some other network accessible via the gateway. Of course, a gateway must know about the type (protocol) of a packet in order to be

able to forward it.

(ENCAPSULATE.ETHERPACKET NDB PACKET PDH NBYTES ETYPE) [Function]
Encapsulates PACKET for transmission on network NDB. PDH is the physical destination host (e.g., an 8-bit pup host number or a 48-bit NS host number); NBYTES is the length of the packet in bytes; ETYPE is the packet's encapsulation type (an integer).

(TRANSMIT.ETHERPACKET NDB PACKET) [Function]
Transmits PACKET, which must already have been encapsulated, on network NDB. Disposition of the packet after transmission is complete is determined by the value of PACKET 's EPREQUEUE eld.

In order to receive Ether packets of type other than PUP or NS, the programmer must specify what to do with incoming packets. Lisp maintains a set of *packet lters*, functions whose job it is to appropriately dispose of incoming packets of the kind they want. When a packet arrives, the Ethernet driver calls each lter function in turn until it nds one that accepts the packet. The lter function is called with two arguments: (PACKET TYPE), where PACKET is the actual packet, and TYPE is its Ethernet encapsulation type (a number). If a lter function accepts the packet, it should do what it wants to with it, and return T; else it should return NIL, allowing other packet lters to see the packet.

Since the lter function is run at interrupt level, it should keep its computation to a minimum. For example, if there is a lot to be done with the packet, the lter function can place it on a queue and notify another process of its arrival.

The system already supplies packet lters for packets of type PUP and NS; these lters enqueue the incoming packet on the input queue of the socket to which the packet is addressed, after checking that the packet is well-formed and indeed addressed to an existing socket on this machine.

Incoming packets have their EPNETWORK eld lled in with the NDB of the network on which the packet arrived.

(\ADD.PACKET.FILTER FILTER) [Function]
Adds function FILTER to the list of packet lters if it is not already there.

(\DEL.PACKET.FILTER FILTER) [Function]
Removes FILTER from the list of packet lters.

(\CHECKSUM BASE NW ORDS INITSUM) [Function]
Computes the one's complement add and cycle checksum for the NW ORDS words starting at address BASE. If INITSUM is supplied, it is treated as the accumulated checksum for some set of words preceding BASE; normally INITSUM is omitted (and thus treated as zero).

(PRINTPACKET PACKET CALLER FILE PRE.NOTE DOFIL TER) [Function]
Prints PACKET by invoking a function appropriate to PACKET 's type. See PRINTPUP for the intended meaning of the other arguments. In order for PRINTPACKET to work on a non-standard packet, there must be information on the list \PACKET.PRINTERS.

\PACKET.PRINTERS [Variable]
An association list mapping packet type into the name of a function for printing

that type of packet.

21.8 THE SYSQUEUE MECHANISM

The SYSQUEUE facility provides a low-level queueing facility. The functions described herein are all system internal: they can cause much confusion if misused.

A SYSQUEUE is a datum containing a pointer to the first element of the queue and a pointer to the last; each item in the queue points to the next via a pointer field located at offset 0 in the item (its QLINK field in the QABLEITEM record). A SYSQUEUE can be created by calling (NCREATE 'SYSQUEUE).

(\ENQUEUE Q ITEM) [Function]
Enqueues ITEM on Q, i.e., links it to the tail of the queue, updating Q's tail pointer appropriately.

(\DEQUEUE Q) [Function]
Removes the first item from Q and returns it, or returns NIL if Q is empty.

(\UNQUEUE Q ITEM NOERR ORFLG) [Function]
Removes the ITEM from Q, wherever it is located in the queue, and returns it. If ITEM is not in Q, causes an error, unless NOERR ORFLG is true, in which case it returns NIL.

(\QUEUELENGTH Q) [Function]
Returns the number of elements in Q.

(\ONQUEUE ITEM Q) [Function]
True if ITEM is an element of Q.