A Programmer's Guide to the EUCSD Ethernet

Introduction

CLIENT, STATION, PACKET, PORT

Ethernet "Stations" are devices connected to "Clients" (computers). It is the job of stations to forward messages across the ethernet between and on behalf of clients or processes within clients.

Physically, messages are transmitted in "packets" which consist of a 15-byte header, the data part containing the actual message (up to 532 bytes) and a 4-byte trailer. Client software has very little control over the actual scheduling of transmissions, or of what exactly goes into the header and trailer. These details are left up to "firmware" running within the station.

The interface presented to the client is of a number (32) of "ports" which are endpoints of transmissions. These ports are addressable objects, each having a six-byte address. Within the EUCSD ethernet, the first address byte (usually) uniquely identifies a station, the second byte (usually) identifies a port within a station, and the remaining four bytes must all be zero. Client software treats ports number 1 to 31 of its station as independent general-purpose bi-directional streams through which messages of between 0 and 532 bytes in length may be sent or received. Port number 0 is rather more special-purpose and is discussed later.

The services offered by the station fall into one of three categories:

CONNECTION, DATAGRAM, BROADCAST

(1) The virtual connection provides a reliable bi-directional link between a port in one station and a port in another station. The connection is reliable in the sense that the station deals with any requirement for re-transmitting packets which have collided or have been lost for some other reason (usually congestion at the destination station). Packets are guaranteed to arrive at their destination in the same order in which they were transmitted and receiving stations guarantee to suppress the forwarding of duplicated packets. The transmitting client is notified when a packet just sent has been accepted by the receiving station (not the receiving client). Alternatively, when the transmitting station reckons that further retransmissions due to absence of acknowledgements would be futile, the transmitting client is notified of the transmission failure. Transmissions of datagrams and broadcasts are not reliable in the sense that connections are; packets merely arrive at their destination with a high probability and are duplicated with low probability.

In a virtual connection there is always a unique "other end" associated with each end-point (port). So there is the concept of a port being "closed", i.e. having no other end associated with it, or "open to station S port P", i.e. having a particular endpoint associated with it. A port which is closed will not accept any packets addressed to it, and a port which is open will only accept packets coming from the particular other end in which it is

interested.

Once various bits of software running in different clients have established that they wish to communicate they would normally seek to establish a reliable connection by agreeing which of their respective ports to open. Such agreement can be established by the programmer beforehand, if he knows which physical machines are going to be involved.

For example, this approach, rather inelegant but dictated at the time for various reasons, is taken in the way the Fred-machines talk to the 1976 Departmental filestore. The machines know that the filestore has address 16■70, the filestore knows that the Fred-machines have addresses in the range 16■11 to 16■2F, so the filestore starts up by opening its port I to machine 16■10+I, port F, for I between 1 and 16■1F, and each Fred-machine (with address J) opens its port F to machine 16■70, port J-16■10.

(2) The datagram is a message sent through port 0 of any machine. Port 0 is in a sense deemed permanently open to any other end, and so always accepts any packet addressed to it. Because there is no permanent remote address information associated with port 0, any message sent through port 0 must be prefixed by six address bytes, i.e. the first six bytes of a message (which must be between 6 and 538 bytes in length) are interpreted by the station as a destination address, and any message received through port 0 will have the six-byte source address prefixed to it by the station. This allows receiving clients to determine where the datagram came from and transmitting clients to instruct the station where a message is to go to.

For example, the 1982 mini-filestore and the 1983 replacement filestore use datagrams from a prospective customer machine to the filestore (the station address of which is known to the prospective customer) to convey a request to establish a connection. The filestore responds with a datagram saying which port it has allocated, and opened, to the prospective customer.

(3) The broadcast is a message sent to (port 0 of) all machines on the ethernet. Address filtering hardware in the ether receiver of every station makes the station deaf to all packets other than those which are either addressed specifically to that station (the first byte of the header is in fact the destination station address), or else addressed to pseudo-station number 0 (deemed the broadcast address). Since packets are always addressed to a remote object (port) with a six-byte (effectively two-byte) address, and the first byte being zero designates a broadcast, and since broadcasts are always accepted by port zero, the meaning of the second address byte of a broadcast message cannot be a port number. Instead it is taken to be a notional broadcast "channel" number. Receiving stations may be configured to tune in to all, no, or an arbitrary selection of these broadcast channels, currently numbered 0 to 63.

Broadcasts are sent in the same way as datagrams, i.e. if they are sent through port 0 they are prefixed by the six-byte address in which the pseudo station number is zero and the pseudo port number is the channel number, if they are sent through a non-zero port then that port must have been opened to remote "station" zero and "port" <channel number>. Reception of broadcasts, however, is different from datagrams in that, though they both come through port 0, there must be some way of deciding whether an incoming packet was a datagram or a broadcast. Since of the six-byte prefix the first is the source

station address and must hence be non-zero, a broadcast is further prefixed by two bytes of which the first is zero and the second is the channel number. In other words, a datagram is prefixed by the six bytes <source station>,<source port>,<four zeroes>, and a broadcast is prefixed by the eight bytes <zero>, <channel number>,<source station>,<source port>,<four zeroes>.

For example, where a particular service (such as the CS4 X25 exercise "other end" machine) is provided by software running in some machine on the ethernet, but where it is not known in which particular machine that software happens to be running, it would be appropriate to allocate (a priori) a broadcast channel number to that service, and to expect customer machines to broadcast on that channel to establish communication with the server. Once the server responds, the customer will know which machine the server is running in and subsequent communication may be by datagrams or virtual connections.

High-Level Programming

Under the current (December 1983) software environment on the Fred-machines the following procedures are available for communicating across the ether:

routine etheropen(integer localport,remoteport)
LOCALPORT should be in the range 1 to 31, REMOTEPORT is the sum of a number in the range 0 to 31 (the remote port number) and another number (the remote station address) multiplied by 256 (shifted left 8).
Calling ETHEROPEN has the obvious effect of opening port LOCALPORT in this client's station to port REMOTEPORT&31 in remote station numbmer REMOTEPORT>>8.

routine etherwrite(integer port, bytename buffer, integer size)
PORT must be zero or the number of a local port which has been opened.
BUFFER points to the first byte of the message to be transmitted, but if PORT=0 it points to the first byte of the six-byte address field, to be immediately followed by the actual message.
SIZE conveys the size of the message (inclusive of address field if PORT=0). It must be in the range 0 to 532 if PORT#0 or 6 to 538 if PORT=0.
The effect is to wait until transmission of any packet previously sent on the same port has been acknowleged, then to transmit this packet.

integerfn etherread(integer port, bytename buffer, integer maxsize)
PORT must be zero or the number of a local port which has been opened.
BUFFER points to the first byte of a buffer into which a message is to be received.
MAXSIZE indicates how big the buffer is. Incoming messages longer than this value will be read but the caller's memory is not overwritten past the end of the buffer.
The result of the function is the actual size of the message received (inclusive of the six-or-eight-byte prefix if PORT=0), and may be larger than maxsize.
The effect is to wait until a packet arrives on this port, then to read it.

routine enable broadcasts(integer channel)
CHANNEL should be in the range 0 to 63.
The effect is to instruct the station to tune in to the specified channel. It does not affect the "tuned-in-ness" to any other channel, i.e. to tell the station to tune in to all channels, this procedure must be called 64 times with CHANNEL taking every value between 0 and 63.

routine disable broadcasts
There is no parameter and the effect is to instruct the station to disregard broadcasts on all channels.

integerfn etherstation
The result is the station address of this client's ether station.

** NOTE **

Unless you know what you're doing, steer clear of attempting to talk to stations 16■70, which is the filestore, and 16■72, which is VAX. Also steer clear of port 16■0F, which is used by the Fred-machine system to communicate with the filestore.

** NOTE **

The standard firmware in the EPROM of the ether station is not up to date and does not work properly with regard to datagrams and broadcasts. To use these services, it is necessary (until such time as the EPROMs can be changed on all machines) to load the up-to-date firmware into your local ether station. This is done by issuing the system command ETHER:LOAD, which will respond by saying "Done" followed by the address of your station. This will have to be repeated if for any reason your station has been reset, such as when you re-load the Fred-machine.

Advanced Facilities

It is not always convenient to call ETHERREAD or ETHERWRITE because they will wait until it is possible to proceed with the transfer. To allow user programs to determine whether such transfers can proceed there exist a number of integer variables which are treated as boolean arrays (0:31) which indicate whether a particular condition holds for each of the 32 ports.

Bit P of variable DTX indicates, when set, that a packet has arrived on port P, and that it is therefore possible to do an ETHERREAD on that port without waiting. In fact what ETHERREAD does is to wait until bit P of DTX becomes set, it then clears that bit and reads the packet. Setting of the bit is done by the interrupt handler in the system.

Bit P of variable ACK indicates, when set, that either an acknowledgement for the packet last sent on port P has been received (if bit P of variable NAK is clear) or that the station has given up re-transmitting (if bit P of variable NAK is set). In fact what ETHERWRITE does is to wait until bit P of ACK is set, it then clears it and sends the packet. It ignores NAK. Whenever the station sends an ACK control character to its client, the interrupt handler sets the appropriate bit in variable ACK; when the station sends a NAK character, the interrupt handler sets the appropriate bits in both NAK and ACK. Therefore, to determine whether your transmission has completed, you should wait until the bit in ACK sets, then look at NAK. If the NAK bit is set, the transmission failed (and you should then clear the NAK bit), otherwise the packet sent was accepted by the destination (except in the case of broadcasts, which are never acknowledged by receiving stations, but the transmitting station nevertheless sends a proforma ACK to its client to keep the protocol consistent).

In the above, "bit P of XXX is set" means that "XXX&(1<<P)#0", "bit P of XXX is clear" means "XXX&(1<<P)=0", in other words bit 0 is the least significant bit of the word.

All the above procedures and variables are specified in IMP Include-file "i:fs.inc". This file also contains references to variables STX and RDY, these are of relevance only to the ETHERREAD, ETHERWRITE, and ETHEROPEN routines and should not be used by high-level user programs running under the standard Fred-machine system.

Low-Level Programming

[This section is of interest to those writing programs which do not expect to run under the standard system. It has not been written yet.]

Rainer Thonnes, 16/12/83.

the ethernet routines and the (undocumented) facilities <> are contained in file ~~I:FS.INC~~ inc:fs.imp which is reproduced below. (27/5/84)

```
endoflist
! Ethernet and Filestore communication
@16■035c4 short USERNO
@16■0372c byte CYLOCK,FSPORT
@16■03fa8 byte LDTE,LSAP,RDTE,RSAP
@16■010f8 routine ETHEROPEN(integer port,remote)
@16■010fc routine ETHERCLOSE(integer port)
@16■01100 routine ETHERWRITE(integer port,bytename buf,integer size)
@16■01104 integerfn ETHERREAD(integer port,bytename buf,integer
max)
@16■01108 integerfn FCOMM(integer cn,string(255)s)
@16■0110c routine FCOMMW(integer c,string(255)p,bytename b,integer
len)
@16■01110 integerfn FCOMMR(integer c,string(255)p,bytename b,integer
max)
@16■03730 integer ERR
@16■03734 integer DTX
@16■03738 integer RDY
@16■0373c integer STX
@16■03740 integer ACK
@16■03744 integer NAK
@16■7fffc byte ETHS
@16■7fffd byte ETHD
@16■7ffff byte ETHC
routine TWAIT
  cycle; repeatuntil eths&8#0
end
routine DISABLE BROADCASTS
constinteger bof=10
  twait; ethc = bof; twait
end
routine ENABLE BROADCASTS(integer channel)
constinteger bon=9
  twait; ethc = bon; twait; ethd = channel; twait
end
integerfn ETHERSTATION
@16■372e short station
constinteger esa=5
  station = 0; twait; ethc = esa; twait
  cycle
    result = station unless station=0
  repeat
end
routine ETHERDTX(integer port)
integer bit
  bit = 1<<port
  if bit=0 or rdy&bit#0 or ack&bit=0 start
    selectoutput(0); printstring("*Naughty call on ETHERDTX(")
    write(port,0);          printstring(")          -          RDY=");
printsymbol(rdy>>port&1+'0')
    printstring(",  ACK="); printsymbol(Ack>>port&1+'0');  newline;
return
  finish
  twait; ethc = 16■30; twait; ethd = port; twait
end
list
```