THE NETWORK SERVER

December 7, 1984

2600 Liberty Avenue
P. O. Box 2600
Pittsburgh, PA 15230
(412) 355-0900

Accent is a trademark of Carnegie-Mellon University.

Accent and many of its subsystems and support programs were originally developed by the CMU Computer Science Department as part of its Spice Project.

The information in this document is subject to change without notice and should not be construed as a commitment by PERQ Systems Corporation. The company assumes no responsibility for any errors that may appear in this document.

PERQ Systems Corporation will make every effort to keep customers apprised of all documentation changes as quickly as possible. The Reader's Comments card is distributed with this document to request users' critical evaluation to assist us in preparing future documentation.

PERQ, PERQ2, LINQ, and Qnix are trademarks of PERQ Systems Corporation.

Table of Contents Page

1. Theory

This document describes the user interface to the Network Server process. The Network Server provides access to the 10 megabit network for all processes running on a given PERQ workstation. All user access to the Ethernet is through the Network Server.

Access to the Network Server is through a special Matchmaker-generated interface. This interface provides the user with a set of routines that seem to perform networking functions. Actually, the routines act as intermediaries that send messages to the Network Server requesting its services, wait for its reply, and provide the user with the results of the request (success, failure, or other specific information).

1.1. Basic Functions

The Network Server interface routines provide four basic functions:

- Acquiring specific information about the network;
- Sending packets across the network;
- Registering requests for receiving packets of a specific type; and
- Providing conventions for notifying the user of the arrival of those packets.

1.2. Theory of Operation

The procedures for receiving and sending packets to and from the Network Server are described in detail in Section 2.3. The following is a brief introduction.

When a user process wants to send a packet over the Ethernet, it must first specify the source address, destination address, and packet typein the Ethernet packet header. The packet is then sent using the E10Send routine (described in Section 2.2.8).

When a user process wants to receive packets from the Ethernet, it first must install a filter in the Network Server. A filter is an association of send rights to a port and an Ethernet packet type. When the Network Server receives an Ethernet packet, it checks to see if there are any ports associated with that packet's type in a filter. If there is, it sends the received packet to the port specified in the filter in an IPC message. The user process receives the IPC message and calls the Net10MBAsynch routine (described in Section 2.2.10). This routine extracts the packet and its lengthfrom the IPC message (if possible) and passes them to the user process. Ways of receiving IPC messages are discussed in Section 2.3.

1.3. Byte Swapping

Note that the DMA (Direct Memory Access) unit of the PERQ workstation passes words to the network low order byte first. However, the network controller assumes that words are passed high order byte first. Because of this difference, ALL words going to and from the network must have their bytes swapped so each side correctly interprets the data. Byte swapping can be accomplished in any convenient way; an example of a Pascal byte swapping routine looks like this:

```
Function SwapByte (Wrd : integer): integer;

Begin

  SwapByte := lor(shIFt(wrd, -8), shIFt(wrd, 8));
End;
```

2. Use

This chapter has three sections. Section 2.1 contains constant and data type definitions that are used in the Network Server interface routines. Section 2.2 contains descriptions of the interface routines. Section 2.3 gives a detailed explanation of how to send and receive Ethernet packets.

The definitions and programming examples throughout this document are given in Pascal. If you are programming in the C language, please refer also to the document "C System Interfaces" in the *Accent Languages Manual*. If you are programming in the Lisp language, see the document "Lisp Interaction with the Accent Operating System" in the *Accent Lisp Manual*. When FORTRAN becomes available under Accent, the type and routine definitions will be the same as in the C language.

2.1. Type Definitions

The data type and constant definitions are in module Net10MBDefs in Net10MBDefs.Pas in LibPascal.

```
Module Net10MBDefs;

Exports

Imports AccentType from AccentType;
const
   El00K = 3000;
   El0TimeOut = 3001;
   El0ByteCount = 3002;
   El0NoNet = 3003;
   El0SendError = 3004;
   El0CRC = 3005;
   El0BytesInHeader = 14;
   El0WordsInHeader = 7;
   El0MinDataBytes = 46;
   El0MaxDataBytes = 1500;

type
   El0Port = port;
```

```
    El0Address = record
                  High: integer;
                  Mid: integer;
                  Low: integer;
                 end;
    El0RetVal = integer;
    El0Type = integer;
    El0FilterType = integer;
    El0ByteData = packed array [1 .. 1500] of 0 .. 255;
    El0Packet = packed record
                 Dest:   El0Address;
                 Src:    El0Address;
                 PType:  El0Type;
                 BData:  El0ByteData
                end;
    pEl0Packet = ^El0Packet;
    El0StatsRec = record
                   CRC: long;
                   Collision: long;
                   Runt: long;
                   Dribble: long;
                   Drabble: long;
                   Snt: long;
                   Rcvd: long;
                   Retry: long;
                  end;
    A_String = string[81];
    El0MsgArray = array [1 .. 124] of integer;
    El0Message = record
                  Head: Msg;
                  Body: El0MsgArray;
                 end;
    pEl0Message = ^El0Message;

Private

Procedure Bug;
begin end.
```

2.2. Routine Definitions

The routine definitions are in module Net10MB in Net10MBUser.Pas in LibPascal.

2.2.1. Initializing Network Server interface

```
Procedure InitNet10MB( RPort : Port );
```

Abstract:

Initializes the Matchmaker Network Server interface.

Parameters:

RPort One of:

NullPort If RPort is NullPort, Net10MBUser will try to allocate its own port for replies. This is the normal value

ReplyPort User-allocated port for replies

2.2.2. Returning the server's version number

```
Function Net_Version( ServPort: El0Port ): String;
```

Abstract:

Obtains the server's version number.

Parameters:

ServPort The 10MBit network server port

Returns:

Version number

Version number of the Net Server, in string form

2.2.3. Getting the Ethernet address of a machine

```
Function El0GetAdd(
                ServPort : El0Port;
            var Addr     : El0Address
        ):GeneralReturn;
```

Abstract:

Gets the address of the machine. Note that this address will have to be swapped to be inspected.

Parameters:

ServPort The 10MBit network server port

Addr Will be set to the address of the machine. The address is in a form

that can be placed into Ethernet headers. If it is to be printed to users the bytes of each word MUST BE SWAPPED.

Results:

E10OK Indicates that there is network hardware in the machine and that the routine has executed successfully. In this case Addr will contain the address of the machine

E10NoNet Indicates that there is not network hardware in the machine. In this case Addr is not valid

2.2.4. Setting a filter

```
Function E10SetFilter(
                ServPort    : E10Port;
                PacketPort  : E10Port;
                Which       : E10FilterType
        ):GeneralReturn;
```

Abstract:

This procedure is used to Informs the Network Server that a user wishes to receive Ethernet packets of a specific type. It sets up a filter, which is a port/packet type pairing.

Parameters:

ServPort The port of the Ethernet server

PacketPort
A port that the server has send rights to. Packets of the requested type will be sent to the user on this port

Which The Ethernet packet type in which the user is interested

Results:

E10OK Indicates that all went well

E10NoNet No network hardware in the machine

Exceptions:

E10NoFreeStructures
Raised if can't associate any more ports and packet types. Note that a handler for this exception must be written by the user

2.2.5. Clearing packet types associated with a given port

```
Function ElOPortClear(
                ServPort:   ElOPort;
                PacketPort: ElOPort
        ):GeneralReturn;
```

Abstract:

Clears a set of packet types that a process has associated with a given port.

Parameters:

ServPort The 10MBit network server port

PacketPort
The port that is to be cleared. All filters that were set with PacketPort as the port will be removed

Returns:

E10OK

2.2.6. Clearing ports associated with a given packet type

```
Function ElOTypeClear(
                ServPort : ElOPort;
                Which    : ElOFilterType
        ):GeneralReturn;
```

Abstract:

Clears a set of ports that a process has associated with a given packet type.

Parameters:

ServPort The 10MBit network server port

Which The packet type that is to be cleared. All filters that were set with Which as the packet type will be removed

Returns:

E10OK

2.2.7. Clearing a specific packet type / port pair

```
Function E10BothClear(
                ServPort    : E10Port;
                PacketPort  : E10Port;
                Which       : E10FilterType
        ):GeneralReturn;
```

Abstract:

Clears a specific filter that has been set by a process.

Parameters:

ServPort The 10MBit network server port.

PacketPort The port of the filter that is to be cleared.

Which The packet type of the filter that is to be cleared.

Returns:

E10OK

2.2.8. Sending a packet

```
Function E10Send(
                ServPort : E10Port;
                Buff     : pE10Packet;
                NumBytes : long
        ):GeneralReturn;
```

Abstract:

Sends a packet on the network.

Parameters:

ServPort The 10MBit network server port

Buff A pointer to the buffer that is to be sent. The Src, Dest, and Type fields of the packet must have been filled in by the user

NumBytes The TOTAL number of bytes that are in the packet to be sent, INCLUDING the 14 bytes that are in the header

Returns:

E10OK

E10SendError
Timed out on sending the packet

E10ByteCount
Indicates that the byte count was not between 46 and 1500

E10NoNet No network hardware on the machine

2.2.9. Getting network statistics

```
Procedure E10Stats(
              ServPort : E10Port;
            var Stats  : E10StatsRec
        );
```

Abstract:

Returns a record containing seven statistics about traffic at the network server. Each field in the record is a long, and a list of each field name and a description of the value is given below:

FIELD	DESCRIPTION
CRC	# of CRC errors found by hardware
Collision	# of collisions detected on receives
Runt	# of packets under minimum length
Dribble	# of packets of odd number of bytes
Drabble	# of packets exceeding maximum length
Snt	# of packets successfully sent
Retry	# of retries for sending a packet
Rcvd	# of packets successfully received

Parameters:

ServPort The 10MBit network server port.

Stats Record into which the various statistics are to be placed.

2.2.10. Handling a received packet

```
Function Net10MBAsynch(
              RequestMsg : pE10Message
        ):Boolean;
```

Abstract:

Net10MBAsynch is a routine that is called right after receiving an IPC message that may be from the Network Server. It checks the message ID to make sure it

is from the Network Server, then strips the packet and number of bytes in the packet from the message and hands them to the user process by raising the exception E10Receive with them as parameters. The user must write a handler for this exception.

Parameters:

RequestMsg
Pointer to the received message

2.3. Using the Network Server

This section provides a detailed, step-by-step description of how to use the Network Server to send and receive Ethernet packets.

2.3.1. Finding the server

All communication with the Network Server is done through IPC messages. To gain access to the Network Server itself, a user process must have access to a port that can be used to send requests to the server. This port must be obtained from the Name Server. When the Network Server is started it registers itself with the Name Server under the name "EtherServer." The user can then use the LookUp routine to get the Network Server port.

The following is a sample piece of code to obtain the Network Server port. Note that in this and all other programming samples, failure of any function call will be reported to the user process and a reason will be given. These reasons are defined in the appropriate definitions files in LibPascal.

```
Import MsgN from MsgNUser;
Import PascalInit from PascalInit;

VAR
   servPort: E10Port;
   gr: GeneralReturn;
    .
    .
    .
InitMsgN(NullPort);  { must be called first
                     { for LookUp to proceed }
gr := LookUp(NameServerPort,
```

```
      'EtherServer', servPort);
  IF gr <> Success THEN
     Writeln('Could not find server, Reason # ',gr);
```

2.3.2. Getting the Ethernet address of the machine

The process using the Network Server must place the Ethernet address of the current machine into the Source field of the Ethernet packet header before it can send that packet. The E10GetAdd routine returns the Ethernet address from the Ethernet hardware. The address that is returned is in a form that can be placed directly into the an address field of an Ethernet header; bytes do not need to be swapped first (note, however, that bytes within the address words must be swapped in order to properly display the Ethernet address). Here is an example of how to get the machine's Ethernet address:

```
VAR
  retVal: E10retVal;
  add: E10address;
  retVal := E10GetAdd(servPort, add);

  IF retVal <> E100K THEN
     Writeln
        ('Could not get address, Reason # ', retVal:1)
  ELSE
     Writeln
        ('My address is: ', SwapByte(add.High):1, ',',
                            SwapByte(add.Mid):1, ',',
                            SwapByte(add.Low):1);
              { swapping done for display only }
```

2.3.3. Sending an Ethernet packet

Before sending a packet on the Ethernet, the user must specify the following information in the Ethernet packet header: Destination address, Source address, and type of packet. After these fields have been sent, the user can send the packet.

The following code is an example of how to set the Ethernet header fields and send a packet:

```
VAR
  i, startByteOffset: integer;
  retVal: GeneralReturn;
```

```
  sPacket:  pEIOPacket;
  numBytes: Integer;
  packType : Integer;
      .
      .
      .
  new(sPacket);          { Get a packet }

{
{ Set source address (obtained in previous example)
{

  sPacket^.Src := add;

{
{ Set destination address according to user inquiry
{

  Writeln;
  Writeln('Enter Destination Ethernet Address.');
  Writeln;

  Write('   High:   ');
  Readln(sPacket^.Dest.High);
  sPacket^.Dest.High := SwapByte(sPacket^.Dest.High);

  Write('   Middle: ');
  Readln(sPacket^.Dest.Mid);
  sPacket^.Dest.Mid := SwapByte(sPacket^.Dest.Mid);

  Write('   Low:    ');
  Readln(sPacket^.Dest.Low);
  sPacket^.Dest.Low := SwapByte(sPacket^.Dest.Low);

{
{ Set Ethernet packet type field according to user
{ inquiry

  Writeln;
  Write('Enter Ethernet Packet Type: ');
  Readln(packType);
  Spacket^.PType := packType; { assume swapped }

{
{ Insert some data
{

  Writeln;
  Write(
    'Enter Offset in
     "For i=0 To n-1 Do Byte[i] = i + Offset" : ');
  Readln(startByteOffset);
  FOR i := 0 TO numBytes - 1 DO
    sPacket^.BData[i] :=
       land((i + startByteOffset), #377);
```

```
{
{ Send the packet
{

   Writeln;
   retVal := E10Send(servPort, sPacket, numBytes +
        E10BytesInHeader);
   IF retVal <> E10OK THEN
        Writeln('Could not do send.  Reason # ',
                    retVal:1)
   ELSE
        Writeln('Packet Successfully Sent.');
```

2.3.4. Connecting to an Ethernet type

Before receiving packets from the network, the user process must tell the Network Server which packet types in which the user is interested and on which port the packets are to be sent. This packet type / message port pair is called a filter.

It is possible to receive packets of more than one Ethernet type on the same port by making multiple calls to the E10SetFilter routine with different types but the same port.

The following programming example assumes that the application has only one port associated with the server. This need not be the case.

```
CONST BACKLOG = 5;   { For port allocation purposes }

VAR PacketType: integer;
    retVal: E10retVal;
    myPort: Port;
    gr:     GeneralReturn;
      .
      .
      .
{
{  Have to have a port to associate with the packet
{

  gr := AllocatePort(KernelPort, myPort, BACKLOG);
  IF gr <> Success THEN
     Writeln(
        'Could not allocate packet port, Reason # ',gr);

{
{  Get the packet type and set the filter
```

```
{

  Writeln;
  Write('Packet Type to receive: ');
  Readln(PacketType);      { swapped }
  retVal := E10SetFilter(servPort, myPort, PacketType);

  Writeln;
  IF retVal <> E100K THEN
     Writeln('Could not set filter, Reason # ',
         retVal:1)
  ELSE
     Writeln('Filter successfully set.');
```

2.3.5. Receiving and processing Ethernet packets

This section describes how to receive an Ethernet packet. When the Network Server receives an Ethernet packet, it checks its users to see if a filter has been set for the received packet type. If so, the Network Server packages the Ethernet packet into an IPC message and sends it to the user process on the port provided in the filter.

The user process can either specifically attempt to receive the incoming message, or else can define a handler that will be invoked whenever a normal message is received for it. If the process is doing an explicit IPC receive on the filter port, the process must wait until the message is received before it can continue executing. By defining a handler, the user process can continue to execute, and the incoming message will be dealt with as an asynchronous event. In either case, once the IPC message is received, the Net10MBAsynch routine should be called to deal with it.

When Net10MBAsynch is called, it first checks to see if the message is from the Network Server; if it is not, Net10MBAsynch returns false. If the message is from the Network Server, Net10MBAsynch pulls the Ethernet packet out of the IPC message and raises the exception E10Receive.

E10Receive must be written by the user, and must be of the form:

```
Handler E10Receive(
            ServPort : E10Port;
            Buff     : pE10Packet;
            NumBytes : Long);
```

where Buff is a pointer to the packet and NumBytes is the total number of bytes in the packet; ServPort is usually ignored. Any processing of the packet can then be done within E10Receive or in other user-declared procedures called from E10Receive. Remember that the E10Receive exception MUST be provided by the user; there is no predefined E10Receive.

Below is an example of an E10Receive handler declaration that is consistent with the examples given so far. Note that in this section, the user process is performing an explicit receive on incoming messages and will wait until the receive procedure is finished before continuing execution. If incoming messages were being handled asynchronously, then similar code would be inserted into the normal message handler.

```
VAR
   rPacket : PE10Packet;
   numBytesRec: Integer;   {Number of data bytes}

HANDLER E10Receive(
            ServPort : E10Port;
            Buff     : pE10Packet;
            NumBytes : long
      );
{-------------------------------------------------}
{
{ Abstract:  Upon receipt of data from the netserver,
{    copy into our buffer and release message memory.
{
{ -------------------------------------------------}
VAR
  gr: GeneralReturn;
BEGIN
  loadadr(rPacket^); { This is inline code to copy the message }
  loadadr(Buff^);    { from message memory to our space }
  loadexpr(Shrink(NumBytes) div 2);
  inlinebyte(MovW);
```

```
  NumBytesRec := Shrink(NumBytes - E10BytesInHeader);
                 { This will get only number data bytes }

  { Note: important to invalidate the buffer }
  gr := InvalidateMemory(
          KernelPort,Recast(
               Buff, Virtual Address),NumBytes);
END;

VAR
  gr: GeneralReturn;
  requestMsg: pE10Message;
  replyMsg: pE10Message;
  .
  .
  .
 requestMsg^.Head.MsgSize := wordsize(E10Message) * 2;
 gr := Receive(requestMsg^.Head,0,AllPts,ReceiveIt);

 { here is the specific receive }
 IF gr <> Success THEN
    Writeln('Couldn't receive message, reason #',gr);

 IF Net10MBAsynch (requestMsg, replyMsg) THEN
     Writeln(
      'Packet received from Network Server.',
      ' Number of data bytes = ', numBytesRec)
 ELSE
     Writeln('Message not from the net server.');
```

Note two things about the above example. First, dynamic storage for "requestMsg," "replyMsg," and "rPacket" is assumed to have been allocated. Second, in the handler for E10Receive, the message memory allocated for the incoming packet is released (invalidated). It is very important to do this; otherwise the user's address space could grow unbounded.