technical memorandum Daresbury Laboratory

DL/CSE/TM35

REMOTE PROCEDURE CALL FOR ETHERNET PROTOCOL SPECIFICATION.
VERSION 1.0

by

P.S. KUMMER and R. TASKER, Daresbury Laboratory

NOVEMBER, 1984

Science & Engineering Research Council

Daresbury Laboratory

Daresbury, Warrington WA4 4AD

SCIENCE AND ENGINEERING RESEARCH COUNCIL

DARESBURY LABORATORY
WARRINGTON
WA4 4AD

Remote Procedure Call for Ethernet Protocol Specification. Version 1.0

P.S.Kummer and R.Tasker
1-Nov-1984

CONTENTS

1. Introduction.

This document defines a Remote Procedure Call (RPC) protocol for use on an Ethernet. The protocol is deliberately kept simple in order to allow as much bandwidth as possible to be extracted from the Ethernet. Provision is made for the detection of lost Ethernet packets and a "no response" mode is included to provide the equivalent of "window size" in connection-oriented protocols (e.g. X25).

2. Overview.

The basis of the RPC mechanism is that a process in one network machine may call a specified routine in another network machine. This would normally involve the exchange of two messages on the network. The first makes the request and passes parameters for the requested routine. The second is a response which may return parameters from the routine. The underlying network is assumed to be of the datagram type (i.e. no virtual circuit need be established).

This specification defines two extensions to the basic mechanism. The first of these allows a remote procedure to be invoked but does not request a response. This allows for the transmission of bulk data at a higher rate than the simple request-response procedure. The second extension requires that before any RPC interchange the requestor should send a multicast message containing either the name of the target machine or the generic name of the service required. The response to this will be the physical address (on the Ether) of the required server machine. This allows for the easy exchange of interface boards which may have fixed Ethernet addresses. It also allows for several machine to provide equivalent services and for automatic hunting for a free server.

Two basic types of interaction are possible between requestors and servers. The first involves a sequence of interactions (e.g. a file transfer) and mechanisms exist within RPC to detect lost interactions in this sequence. The second is a simple "single-shot" mechanism which allows for single request-response transactions.

3. Implementation.

It is assumed in this specification that within a given machine there will be a "manager" process for RPCs which communicates with the Ether, with one, or more, "server" processes which service RPC requests and with one, or more, requestor processes which make RPC requests (Figure 1). No recommendations are made as to the implementation of the manager-server/requestor structure as this is operating system dependent. Indeed, in some operating systems it may be necessary to combine all functions into a single process. However, it is necessary for at least a logical distinction to exist as the manager and servers/requestors are addressed independently by the RPC protocol.

4. Carrier Service.

This protocol is defined to use a connectionless carrier service. In the case of Ethernet this will be IEEE 802.2 LLC 1.

The carrier service may lose data but any data delivered to the RPC manager is assumed to be correct.

The carrier service must support the delivery of data to a specific address and also to a specified multicast address.

5. Byte order.

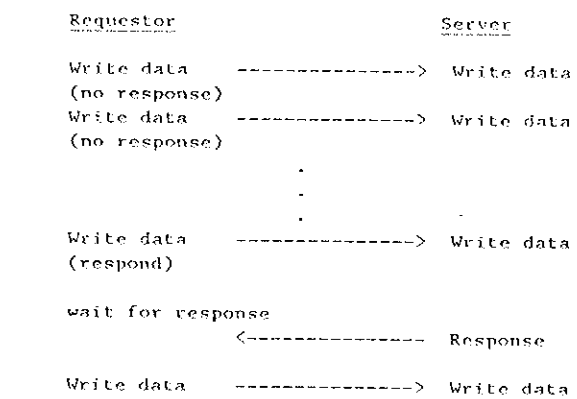
The protocol is defined in terms of 16-bit words.

To allow for the transmission of byte (text) data between machines with different byte ordering within this word a flag is provided within the transmission block header to determine the byte ordering in the transmitted data. It is the responsibility of the receiving machine to correct any incompatability, however, it is possible to provide procedures that will allow two machines to determine each others byte ordering requirement and agree on which machine should resolve any incompatability. This will allow the best suited machine to do the work.

6. Use of the "No response" option.

The no response option allows higher throughput when transferring bulk data. It should only be used in the data phase of a file transfer.

The basic procedure is as follows.



Note that after sending a "write data" request with response requested the requestor must wait for the response before issuing any more requests.

The number of requests issued before a reponse is requested is application specific. However, it may be possible to overrun the input data buffering of the receiving machine and, therefore, lose data blocks. At the least this will involve retransmitting some data and will adversely affect throughput. It may be necessary for the requestor to agree with the remote manager on the number of outstanding no-response requests so as to reduce data loss to an acceptable level.

To ensure correct recovery after an error the following rule needs to be obeyed.

The last data block in a transfer must request a response.

7. Functions of the Manager Process.

The manager process handles all communications with the network and routes data blocks to/from server/requestor processes.

The manager process may create new server processes on request.

A requestor process may connect to the manager to request service from a remote server process.

The manager process handles the no response option. All communications between the manager and its servers are of the request-response type.

The manager checks for lost data blocks (using the sequence number on received data blocks) and reports this at the earliest possible time to the requestor process (but not the server). The requestor process is thus responsible for error recovery. When returning a last valid sequence number the manager should always refer to the last request that it knows has been actioned by the server.

The manager process should validate the network address and process ID for all non-single-shot requests.

There should be no special internal routing within the manager process (i.e. all requests should be transmitted even though they may be actioned locally).

8. Rules for Error Recovery.

The requestor is responsible for all error recovery.

To aid in error recovery the following rules should be followed. (Note that rule b has already been stated in section 7.)

a) All requests should be classified as one of two types. The first type includes those requests invoking a non-repeatable action (e.g. to increment a counter, or to open a file). The second type includes those requests invoking a repeatable action (e.g. return status, or

close a file).

b) No-response requests should only be used for data transfer. The last data block in a transfer must request a response.

c) When a non-repeatable request fails the error recovery should be to invoke a repeatable request. When a repeatable request fails it should be repeated up to some specified (finite) limit.

9. Addressing and use of Multicast.

At the requestor-manager process interface it is recommended that system names or generic service names are used to identify the remote machine. These should then be mapped by the manager process into a physical Ethernet address. This mapping is performed by use of the multicast address feature of the Ether. Although this involves a longer setup time for an interaction there are some significant benefits.

a) The manager does not have to keep name-to-address translation tables.

b) The use of generic service names becomes available.

c) Most Ethernet boards have fixed physical addresses which remain with the board (not the system). Therefore, when a faulty board is changed the physical address of the system will also change. This problem is avoided by the use of multicast to perform name-to-address translation.

When an requestor makes a new service request (e.g. a request for a single-shot process or a request to create a new server) it provides the name of the remote service it requires. The manager process then issues a multicast message containing the name asking for any services with that name to respond. The manager builds an address list (up to a maximum length specified by the requestor) containing the physical address (i.e. the Ethernet source address) of all services that respond. After a specified time the address list, plus a count of the total number of responses, is returned to the requestor which may then decide on which address to use. (If subsequently the requestor finds that the selected system cannot actually perform its requests then it

may select another address from the list).

The multicast packet contains a sequence number which provides an identification by which the manager may relate responses to requests. This allows the manager to have several multicasts outstanding at any one time.

10. Error Detection.

The requestor process should supply a timeout for the execution of a request. The timeout period should be supplied to the local manager as part of the request. The local manager will implement the timeout and complete the requestors request with an error indicated (in SYSRESP) if the specified period is exceeded.

The timeout period specified by the requestor must allow for network proporgation delays and for the execution of the request by the server.

To provide compatibility across implementations a resolution of one second should be provided when specifying the timeout period. The manager may use a more coarse resolution than this if necessary provided that the specified period in a request is lengthened.

The sequence number in the transmission block should be provided by the requestor process. It is a 16-bit, positive integer (MOD 65536) which should be set to 0 in the first request sent to the server and should be incremented by one in each succeeding request.

The remote manager uses the sequence number to check for lost blocks.

The local manager uses the sequence number to check for delayed responses which arrive after the timeout period has expired.

The following procedure should be followed after a timeout.

After detecting the timeout the local manager completes the request with an error code in SYSRESP indicating timeout. The requestor should then send a "check-status" request to

the remote manager.

The remote manager replies with the sequence number of the last data block it received which it knows has been processed by the server. It also indicates whether the server is currently processing a request and whether there are requests waiting to be processed by the server (this should only be possible if the no-response option is being used).

The requestor can now decide on its error recovery action. Note that to ensure correct recovery after a timeout the response from the remote manager should indicate that the server is not processing a request and that there are no waiting requests.

If the request for status times-out then it may be retried several times before the requestor decides that there is an unrecoverable error.

11. Single-shot.

The single-shot procedure allows requests to be made which are not related to any other request. All requests to the remote manager process (SERVID=0) and the single-shot process (SERVID=1) are of this type.

For single-shot requests the remote manager bypasses all validity checking (i.e. sequence number, address).

Single-shot requests must always be repeatable.

The "check-status" manager request is invalid when asked about SERVID=0 or 1.

Single-shot requests need to include a sequence number but this is only used by the local manager to relate a response to the original request and to discard responses that arrive after a timeout.

12. Transmitted Block Format.

Two type of data transmission block are defined, one for data and one for multicast. These are shown below.

Note that all addressing is in terms of 16-bit words. Within the block header the least significant 8-bits of a 16-bit word should be transmitted first to the Ether followed by the most significant 8-bits.

The same format block is used for the response. Note that the RPC IDs are not changed around in the response.

It is possible to reserve a parameter field for a response but not a parameter data area by coding a zero offset.

Alternatively, the parameter offset may be set but the transmission block truncated after the last input parameter for the remote routine. In this way the requestor may force alignment on the returned data.

12.1. Common Header Format.

Both types of transmission block have a common header which is defined as follows.

Offset dec	Offset hex	Length dec	Length hex	Field Name	Description
0	0	1	1	DEVWRK	Device driver workarea
1	1	1	1	LENGTH	Frame length (in bytes) (Excludes DEVWRK and LENGTH).
2	2	3	3	REMOTE	Remote Ether address
5	5	3	3	LOCAL	Local Ether address
8	8	2	2	LLC1	LLC1 Header field. (3 octets padded with a trailing null octet)
10	A	1	1	TIMEOUT	Timeout in seconds.
11	B	var	var	----	Start of information field. See below.

The requestor should set the remote Ethernet address (REMOTE) but may leave the local Ethernet address (LOCAL) undefined. The local manager or the local ether interface will insert the local address.

In some cases the local manager may need to reformat the start of the transmission block to suit its ether interface. It is not expected that all the data in the block will be moved in this case but that the local manager will start I/O requests to its ether interface from addresses other than the start of the block.

The server is responsible for inverting the addresses in a response.

12.2. Information field for data blocks.

Offset dec	Offset hex	Length dec	Length hex	Field Name	Description
11	B	1	1	MYID	ID of the requestor process. (A 16-bit integer)
12	C	1	1	SERVID	ID of the server process. (A 16-bit integer)
13	D	1	1	FLAGS	A flag field. See below. (A 16-bit binary field)
14	E	1	1	SEQNO	Sequence number used when there is a sequence of interactions between the requestor and server processes. (A 16-bit positive integer. Mod-65536)
15	F	1	1	SYSRESP	System response. Generated by the manager process.
16	10	1	1	SRVRESP	Server response. Generated by the server process.
17	11	4	4	NAME	Name of the requested procedure. A text field left justified and padded with blanks. Subject to byte ordering.
21	15	1	1	PCOUNT	Number of parameters.
22	16	1	1	P1-OFF	Offset (from start of data block) to start of parameter 1. (A 16-bit integer)
23	17	1	1	P1-LEN	Length of parameter 1 in bytes. (A 16-bit integer).
.	.	.	.	.	.
.	.	.	.	.	.
x	x	1	1	Pn-OFF	Offset to start of parameter n.
x+1	x+1	1	1	Pn-LEN	Length of parameter n.
x+2	x+2	var	var	P1	Parameter 1.
.	.	.	.	.	.
.	.	.	.	.	.
y	y	var	var	P2	Parameter 2.
.	.	.	.	.	.
.	.	.	.	.	.
z	z	var	var	Pn	Parameter n.
.	.	.	.	.	.
.	.	.	.	.	.
End of block.					

The flag field is defined as follows.

FLAGS

Bit setting	Name	Description
....0	RESP	A response is required to this request.
....1	NORESP	No response is required to this request.
....0.	BYTED	Byte ordering is as for DEC machines.
....1.	BYTEND	Byte ordering opposite to DEC machines.
.... .0..	REQUEST	This block is a request
.... .1..	RESPONSE	This block is a response
.... xxxx x...		Reserved
xxxx xxxx		Available for communications between a requestor/server and the manager process. (Implementation specific). Set to zero in the transmitted block.

The server ID field is defines as follows.

Value (dec)	Use
0	Addresses the manager process
1	Addresses the single-shot process
2 - n	Addresses a specific server process

To reserve a parameter position for returned data but to avoid sending an undefined (and redundant) data area in the request the parameter offset may be set to zero. The server process may then place the data to be returned at the end of the transmission block (thus increasing its length) and set the parameter offset accordingly.

The server may reduce the size of the response data block by removing unwanted data from the block and setting the corresponding parameter offset to zero. To facilitate this it is recommended that large data parameters sent from the requestor to the server are placed at the end of the transmission block.

12.3. Information field for multicast.

Offset dec	Offset hex	Length dec	Length hex	Field Name	Description
11	B	4	4	SERVNAME	Name of the requested service.
15	F	1	1	SERLEVEL	Service level. (A 16-bit integer)
16	10	1	1	SEQNUM	Sequence number. Used by the local manager to match responses with requests.
17	11	1	1	TOTRESP	Total number of responses received. (A 16-bit integer)
18	12	1	1	MAXRESP	Maximum number of responses to be put in list. (A 16-bit integer)
19	13	var	var	ADDLIST	Address list work area. Each address requires 3 words and should be followed by a word containing the service level for that address.

The actual transmission block for multicast may be terminated after SEQNUM. However the full block needs to be passed from the requestor to the local manager. Fields after SEQNUM are used by the local manager to record the multicast responses and return them to the requestor.

A response to the multicast should only be returned by systems containing the named service. The response packet should be formatted as a normal data block (see section 12.2) as follows:

MYID = 0
SERVID = 0
FLAGS = RESPONSE
SEQNO = sequence number of multicast packet
SYSRESP = (see section 14)
SRVRESP = 0
NAME = "MULTICST"

13. Requests to the remote manager.

This section details the parameter fields for requests to the remote manager.

13.1. Check-status.

Check-status requests the state of data transfers to a specific server. The request is invalid for the manager process itself and for the single-shot server.

The request should be directed to SERVID=0 (the remote manager) and should have NAME=CHKSTAT.

Five parameter fields are required. The first supplies the server ID to the remote manager and the remaining 4 are for the remote managers response.

Parameter	Contents	Notes
1	Server-ID	The server for which status is being requested.
2	Sequence Number	The sequence number of the last block actioned by the server.
3	Queue Length	The number of blocks waiting to be actioned by the server.
4	Server Status	0 Server not processing a request 1 Server processing a request -1 No server with specified ID
5	Manager Sequence Number	The sequence number the remote manager expects in the next transmission block to the server.

13.2. Create Server.

Create-server requests the remote manager to create a new server process of the specified type.

The request should be directed to SERVID=0 (the remote manager) and should have NAME=CSERVER.

There is one parameter which contains the type of server to be created.

Parameter	Contents	Notes
1	Server Type	The type of server to be created. (An 8-character text field, left justified and padded with blanks)

(Values for Server-Type to be defined)

13.3. Delete Server.

Delete-server requests the remote manager to delete the specified server.

The request should be directed to SERVID=0 (the remote manager) and should have NAME=DSERVER.

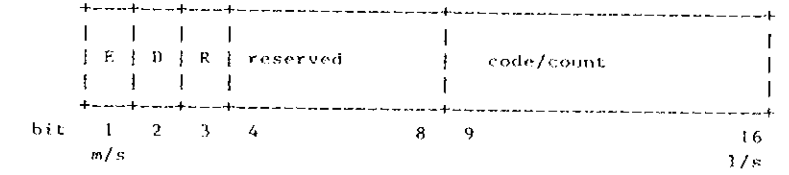
There is one parameter which contains the ID of the server to be deleted.

Parameter	Contents	Notes
1	Server ID	The ID of server to be deleted.

14. SYSRESP. System response codes.

This section defines the possible values for SYSRESP, the system response code.

SYSRESP



E	Error Indication	1 = error 0 = no error
D	Detected by	1 = local manager 0 = remote manager
R	Retry (of block)	1 = possible 0 - not possible

If E=1 then an error is being reported and the DR bits are valid. The code field will contain an error code as detailed below. The D bit indicates where the error was detected and the R bit indicates whether a retry of the transmission of the block is wothwhile (i.e. the error may have been caused by a temporary condition).

EDR	Code (dec)	Meaning
100	1	Unable to create requested server - type not known.
100	2	Unable to create requested server - operator disabled.
100	3	Unable to create requested server - insufficient resources.
100	4	Server has aborted.
101	5	Sequence number error.
100	6	Queue of blocks waiting to be actioned by server too large.
110	7	Block failed address validity checks.
111	8	Timeout (generated by the local manager).
110	9	Local hardware error.
100	10	Service known but unavailable. (e.g. Inhibited by operator command) (Multicast response)
100	11	Service known but busy. (Multicast response)

100	14	RPC protocol error.
101	15	Server create error.

If E=0 no error is being reported and the DR bits are meaningless and should be set to zero. The count value provides a recommendation on how many no-response requests to send before requiring a response (for normal data blocks) or an indication of the current service level (for multicast blocks).

For responses to normal data blocks the recommended number of no-response requests may be set by the remote manager to indicate its current service level and this aids the requestor to avoid lost data blocks. Count=0 provides no recommendations on the number of no-response requests to send.

For responses to multicast blocks the service level should be in the range 0-9 with 9 implying better service than 0.

15. Recommendations on timeouts.

This sections provides some recommendations on the value to be used for timeouts.

The basic unit of time defined by the protocol is 1 second. As this may be implemented by a "tick timer" then the actual value of the timeout period supplied to the local manager should be one greater than the minimum timeout required.

For requests to the remote manager a minimum timeout of 2 seconds is recommended.

For multicast requests a minimum timeout of 1 second is recommended. This is applicable to use of the single-shot process. For less stringent requirements a minimum timeout of 2 seconds is recommended.

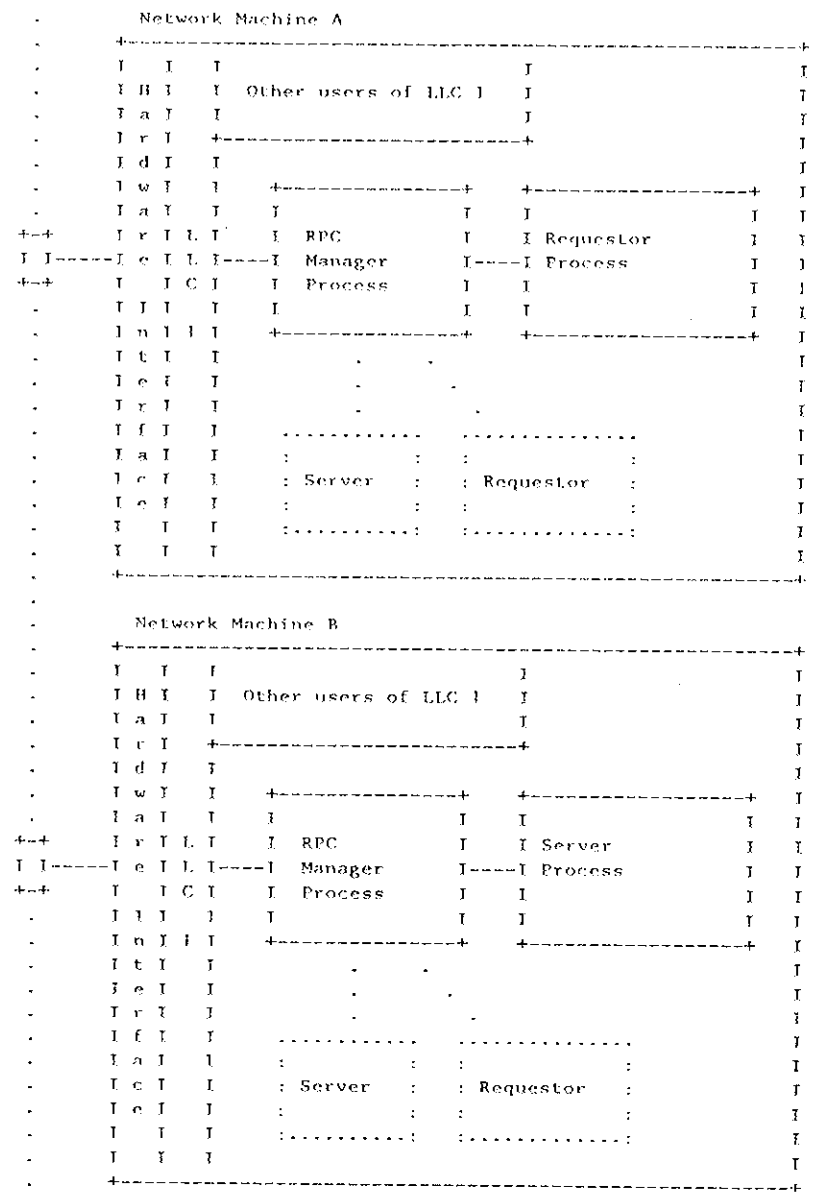
For requests requiring execution of a procedure by the server a minimum timeout of 10 seconds is recommended.

16. Conformance.

No conformance rules are applied to this protocol other than that systems which are required to interwork will, in fact, interwork sucessfully. This RPC is not designed as an Open Systems Interconnection protocol.

The authors of this paper are the "design authority" for the RPC and any comments, reports on inconsistences, or requests for changes should be sent to them. The design authority will keep an address list of people with an interest in the RPC and will circulate updates and any relevant user notes.

Figure 1. Process Overview.



APPENDIX I. Implementation Notes.

This appendix details the first implementation of RPC for the SRS Data Acquisition system. The notes describe which facilities of RPC will not be included in the initial release.

Further notes may be added to this appendix from time-to-time. To avoid a full update to the RPC document this section is dated seperately.

This version dated 12 July 1984.

1. 21/06/84 In section 14 the definition of SYSRESP (for responses to multicast blocks) allows for non-negative values to indicate service level. This is intended as an aid to the requestor when choosing a physical address from the list. In the initial implementation this will always be set to zero.

2. 21/06/84 In section 14 the definition of SYSRESP allows for a recommendation on how many no-response requests to send before requiring a response. In the initial implementation this will always be set to zero (i.e. no recommendation). Experience is required with a running system before it will be possible to decide on a suitable algorithm for evaluating this recommendation. In any case, the algorithm will be system/hardware specific.

3. 21/06/84 The Interlan Ethernet controllers to be used on the SRS Data Acquisition system allow for a "scatter/gather" operation on the data. The restriction is that the scattered/gathered blocks should be multiples of 8 bytes long.

   The specification of the experimental station interface to the RPC only requires scatter/gather in the parameter part of the transmission block. Thus, the alignment may be forced by the requestor by judicious setting of the parameter offsets.

   (Ref: Specification for the Experimental Station Interface to the Ethernet remote procedure call protocol. Frances Rake.)

4. 12/07/84 The zero parameter offset defined in section 12 will not be used in the first implementation. The alternative of a truncated block will be used.

EXAMPLE 1

Write a data file from a data station to the SRE VAX.

This example writes a data file from an SRS data station to the SRE VAX. The no-response option is used during the data transfer phase to maximise throughput.

The following server routine are needed.

OPEN file
WRITE data
CLOSE file

The following abbreviations are used in the example.

BDB	Build a Data Block (in format described in main text)
SDB	Send Data Block (requestor asks manager to transmit a previously constructed data block)
T/O	Timeout

A. Find physical address of the SRE VAX.

Step	Requestor	Local Manager	Remote Manager	Server
1	BDB (Find: SREVAX,5)			
2	SDB, T/O=x			
3		Send Multicast		
4			Receive Multicast	
5			Build Response	
6			Send response	
7		Receive response		
8		Put address into list		
9		Increment total count		
10		Loop 7 until T/O		
11		T/O		
12		Return list + total count		
13	Request complete			

B. Create a server process on the VAX.

Step	Requestor	Local Manager	Remote Manager	Server
14	Select physical address			
15	BDB (Create server: type)			
16	SDB, T/O=x			
17		Send block		
18			Receive block	
19			Create server of requested type	
20			Build OK response containing SERVID	
21			Send block	
22		Receive block		
23		Return to requestor		
24	Request complete			

C. Open the file on the VAX.

Step	Requestor	Local Manager	Remote Manager	Server
25	BDB (Open file: write, filename)			
26	SDB, T/O=x			
27		Send block		
28			Receive block	
29			Route block to server (also check validity)	
30				Issue OPEN request
31				Build OK response
32				Return response
33			Add OK manager response	
34			Send block	
35		Receive response		
36		Return to requestor		
37	Request complete			

D. Write data to the VAX file using the no-response option.

Step	Requestor	Local Manager	Remote Manager	Server
38	RDB (Write: data, no-response)			
39	SDB, T/O=x			
40		Send block		
41			Receive block	
42			Route block to server (also check validity)	
43				Write data
44				Build OK response
45				Return response
46			Record OK response	
47	Loop 38 until 7 blocks sent			

(continued)

Step	Requestor	Local Manager	Remote Manager	Server
48	RDB (Write: data, with response)			
49	SDB, T/O=x			
50		Send block		
51			Receive block	
52			Route block to server (also check validity)	
53				Write data
54				Build OK response
55				Return response
56			Add OK manager response	
57			Add any previous error response from server	
58			Send block	
59		Receive block		
60		Return to requestor		
61	Loop 38 until end-of-data			
62	Write data complete			

E. Close the file on the VAX.

Step	Requestor	Local Manager	Remote Manager	Server
63	BDB (Close file:)			
64	SDB, T/O=x			
65		Send block		
66			Receive block	
67			Route block to server (also check validity)	
68				Issue CLOSE request
69				Build OK response
70				Return response
71			Add OK manager response	
72			Send block	
73		Receive response		
74		Return to requestor		
75	Request complete			

F. Delete the server process on the VAX.

Step	Requestor	Local Manager	Remote Manager	Server
76	BDB (Delete server:)			
77	SDB, T/O=x			
78		Send block		
79			Receive block	
80			Delete server	
81			Build OK response	
82			Send block	
83		Receive block		
84		Tidy local work areas		
85		Return to requestor		
86	Request complete			

Notes on Example 1.

These notes expand parts of the above example. The number on each note refers to the step number in the example.

1. The transmission block is built in the multicast format and includes space for 5 address responses.

2. The timeout should be sufficient for the remote managers to respond.

4. The remote manager should compare the service name in the multicast with its list of servers. If no match is found then no response should be sent.

8. In this case up to 5 responses may be put into the list. Any more responses must be discarded.

9. The total count is kept to inform the requestor of how many systems have responded. This may be more than the number of addresses returned in the list.

10. Any responses received after the timeout should be discarded.

14. The requestor should use the service level parameter (SERLEVEL) when selecting the address. As a minimum, all addresses with a negative service level should be discarded.

15. The transmission block should be addresses to the remote manager (SERVID=0). The request asks for a server of the specified type to be created.

19. If the remote manager is unable to create the requested server then it should return an error response. The requestor may then choose another address from its address list.

20. SERVID should be set to the remote managers identification of the new server.

25. The transmission block should be addressed to the server. The request is for the specified file to be opened for output.

29. The block is routed on SERVID. The validity check includes the requestor ID and Ether address, and that the sequence number is correct. Any error should cause the block to be discarded and an error response to be sent.

31. Any error in the open should be returned in SRVRESP

38. The transmission block should be addressed to the server. The request is to write the specified data to the file and no response is required.

39. The timeout should be set infinite (or close to!)

42. Any errors here should cause the remote manager to save the error code and discard the request. Also, any errors previously reported by the server but not yet returned to the requestor should cause the request to be discarded.

46. Any error reported by the server should be saved by the remote manager.

48. As step 38 except that a response is now requested.

52. Any error previously saved by the remote manager should inhibit routing of the request to the server. Instead, the remote manager should build the appropriate error response and send it to the requestor.

62. The last data block sent must request a response. For some systems that do not report EOF until an attempt to read past EOF it may be necessary to send a dummy data block which will be ignored by the server.

75. If the requestor has more files to transfer then it may return to step 25. It should not be necessary to delete and recreate a server in this case.

76. The "delete server" request should be addressed to the remote manager (SERVID=0).

EXAMPLE 2

Error recovery.

This example shows recovery from an error during the data transmission part of example 1.

The following abbreviations are used in the example.

BDB	Build a Data Block (in format described in main text)
SDB	Send Data Block (requestor asks manager to transmit a previously constructed data block)
T/O	Timeout

A. Write data to the VAX file with error.

Step	Requestor	Local Manager	Remote Manager	Server
1	BDB (Write: data, no-response)			
2	SDB, T/O=x			
3		Send block		
4			Receive block	
5			Route block to server (also check validity)	
6				Write data
7				Build error response
8				Return response
9			Record error response	

(continued)

Step	Requestor	Local Manager	Remote Manager	Server
10	BDB (Write: data, no-response)			
11	SDB, T/O=x			
12		Send block		
13			Receive block	
14			Discard block	
15	Loop 1 until 7 blocks sent			

(continued)

Step	Requestor	Local Manager	Remote Manager	Server
16	BDB (Write:data, with response)			
17	SDB, T/O=x			
18		Send block		
19			Receive block	
20			Set error response	
21			Send block	
22		Receive block		
23		Return to requestor		
24	End of request			
25	Note error response			

B. Check present status.

Step	Requestor	Local Manager	Remote Manager	Server
26	BDB (Check status:)			
27	SDB, T/O=x			
28		Send block		
29			Receive block	
30			Build response (last actioned sequence number, queue status, server status, manager sequence number)	
31			Send block	
32		Receive block		
33		Return to requestor		
34	Request complete			
35	Restart data transfer			

Notes on Example 2.

These notes expand parts of the above example. The number on each note refers to the step number in the example.

1-6. Data transfer as in example 1.

7. An error is detected by the server and an error response built.

9. The remote manager notes the error response and saves it in a local work area. Note that it cannot report the error until it gets a request requiring a response.

10. The requestor continues sending data as it does not know yet of the error.

14. The remote manager discards the data as it has an unreported error and the no-response bit is set in the data block.

16. The requestor has sent its 7 blocks without response and is now sending a block requiring a response.

20. The remote manager receives the block requiring a response and notes its unreported error. It builds a response containing the saved error code.

25. The requestor notes the error and enters error recovery.

26. The requestor asks the remote manager for status.

30. The remote manager builds a response containing status information. This will include the sequence number of the last block actioned by the server without error (i.e. the one before the one generating the error), the next sequence number expected by the remote manager, and the state of any queues of data blocks waiting to go to the server. The status of the server (i.e. processing/not processing a request) is also returned.

In order to recover sucessfully from the error the queues must be empty and the server should not be processing a request.

35. The requestor backs-up its input data stream and restarts the data transfers from the block that gave the error.

Note that it may also be necessary for the requestor to ask for status from the server before it can restart data transmission. This will be application specific. However, it will always be necessary to perform the procedure described above in order to resynchronise the sequence number.

EXAMPLE 3

Request time and date.

This example requests the time and date from a "time server". It uses the single-shot server.

The following server routine is needed.

TIMEDATE

The following abbreviations are used in the example.

BDB	Build a Data Block (in format described in main text)
SDB	Send Data Block (requestor asks manager to transmit a previously constructed data block)
T/O	Timeout

A. Find physical address of a time/date server.

Step	Requestor	Local Manager	Remote Manager	Server
1	BDB (Find: TIMEDATE,5)			
2	SDB, T/O=x			
3		Send Multicast		
4			Receive Multicast	
5			Build Response	
6			Send response	
7		Receive response		
8		Put address into list		
9		Increment total count		
10		Loop 7 until T/O		
11		T/O		
12		Return list + total count		
13	Request complete			

B. Request the time and date.

Step	Requestor	Local Manager	Remote Manager	Server
14	Select physical address			
15	BDB (Timedate:)			
16	SDB, T/O=x			
17		Send block		
18			Receive block	
19			Route block to single-shot server	
20				Call TIMEDATE
21				Build OK response
22				Return response
23			Add OK manager response	
24			Send block	
25		Receive response		
26		Return to requestor		
27	Request complete			

Notes on Example 3.

These notes expand parts of the above example. The number on each note refers to the step number in the example.

1-14. As for Example 1.

15. This request is addressed to the single-shot server (SERVID=1).

19. No validity checking is required for requests to the single-shot server.

20. The required data is placed in a previously defined parameter area in the transmission block.

27. If an error response indicates that the single-shot server was not available then the requestor may return to step 14 to try another physical address.