Network Working Group
Request for Comments #333
NIC #9926
Category: C9 (experimentation)
Obsoletes: 62
Updates: none

Bob Bressler, MIT/Dynamic Modeling
Dan Murphy, BBN/TENEX
Dave Walden, BBN/IMP

15 May 1972

A PROPOSED EXPERIMENT WITH A MESSAGE SWITCHING PROTOCOL

CONTENTS

Introduction

A message switching protocol (MSP) is a system whose function is to switch messages among its ports.

For example, there is an implementation of an MSP in each Interface Message Processor. We believe that the effective utilization of communications networks by computer operating systems will require a better understanding of MSPs. In particular, we feel that Network Control Programs (NCPs), as they have been implemented on the ARPA Computer Network (ARPANET), do not adequately emphasize the communications aspects of networking--i.e., they reflect a certain reluctance on the part of systems people to move away from what we term "the stream orientation". We propose, as an aside to network development using the current NCPs, to rethink the design of NCP-level software beginning with a consideration of MSPs.

The thrust of this note is to sketch how one would organize the lowest level host-host protocol in the ARPANET around MSPs and how this organization would affect the implementation of host software.

Some Background

Over the past several weeks there has been considerable informal discussion about the possibility of implementing, on an experimental basis, in several of the ARPA Network Host Computers, NCPs which follow a protocol based on the concept of message switching rather than the concept of line switching (see the parenthetical sentence in the first paragraph of page 6 of NIC document 8246, Host/Host Protocol for the ARPA Network). Party to this discussion have been Bob Bressler (MIT/Dynamic Modeling), Steve Crocker (ARPA), Will Crowther (BBN/IMP), Tom Knight (MIT/AI), Alex McKenzie (BBN/IMP), Bob Metcalfe (MIT/Dynamic Modeling), Dan Murphy (BBN/TENEX), Jon Postel (UCLA/NMC), and Dave Walden (BBN/IMP).

Several interesting points and conclusions have been made during this discussion:

1. Bressler has implemented a message switched interprocess communication system for the Dynamic Modeling PDP-10 and has extended it so it could be used for interprocess communication between processes in the Dynamic Modeling PDP-10 and the AI PDP-10. He reports that it is something like an order of magnitude smaller than his NCP.

2. Murphy has noted that a Host/Host protocol based on message switching could be implemented experimentally and run in parallel with the real Host/Host protocol using some of the links set aside for experimentation. Further, Murphy has noted that if this experimental message switching protocol were implemented in TENEX, a number of (TENEX) sites could easily participate in the experiment.

3. It is the consensus of the discussants that Bressler should take a crack at specifying a message switching protocol* and that

*This note fulfills any obligation Bressler may have incurred to produce an MSP specification.

if this specification looked relatively easy to implement, a serious attempt should be made by Murphy and Bressler to find the resources to implement the experimental protocol on the two BBN TENEX and the MIT Dynamic Modeling and AI machines.

4. MSP was chosen as the acronym for Message Switching Protocol, and links 192-195 were reserved for use in an MSP experiment.

We solicit comments and suggestions from the Network Working Group with regard to this experiment. However, although we will very much appreciate comments and suggestions, because this is a limited experiment and not an attempt to specify a protocol to supersede the present Host/Host protocol for the ARPA Network, we may arbitrarily reject suggestions.

References

Familiarity with the following references will be helpful to the reading of the rest of this note.

1) NIC document 8246, Host/Host Protocol for the ARPA Network;

2) NIC document 9348 on the Telnet Protocol;

3) NIC document 7101, Official Initial Connection Protocol, Document #2;

4) A System of Interprocess Communication in a Resource Sharing Computer Network, CACM, April, 1972.

Reference 4 is a revision of RFC 62. We strongly suggest the reader be familiar with reference 4 before he attempts to read the present RFC; a reprint of reference 4 is attached as an appendix.

MSP Specification

Our MSP is essentially a generalization of the interprocess communication system outlined in Section 3 of the fourth reference. (Henceforth, if we are required to mention the interprocess communication system presented in Section 3 of reference 4, we shall call it "the IPC".) For two processes to communicate using the MSP, the process desiring to send must in some sense execute a SEND and the process desiring to receive must in some sense execute a RECEIVE. The SEND and RECEIVE, in effect, rendezvous somewhere and transmission is allowed to take place. With the RECEIVE are specified (among other things) a from-port-id, a to-port-id, and a rendezvous Host. With SEND are specified a from-port-id, a to-port-id, a rendezvous Host, and (possibly) some data to be transmitted. Using SEND and RECEIVE, sending a message from a sender process to a receiver process takes place as follows. The sender process executes a SEND which causes an OUT-message plus the specified data to be transmitted to the Host specified as the rendezvous Host in the SEND. Concurrently (although not necessarily simultaneously) the receiver process executes a RECEIVE which causes an IN-message to be sent to the Host specified as the rendezvous Host in the RECEIVE. At the rendezvous Host, OUT-messages and IN-messages are entered in a table called the rendezvous table. When an OUT-message and an IN-message are detected with matching to-port-id, from-port-id, and rendezvous Host, three things are done: 1) the OUT-message plus the data is forwarded to the Host which was the source of the IN-message, 2) the IN-message is forwarded to the Host which was the source of the OUT-message, and 3) the IN-message and OUT-message plus the data are deleted from the rendezvous table in the rendezvous Host.

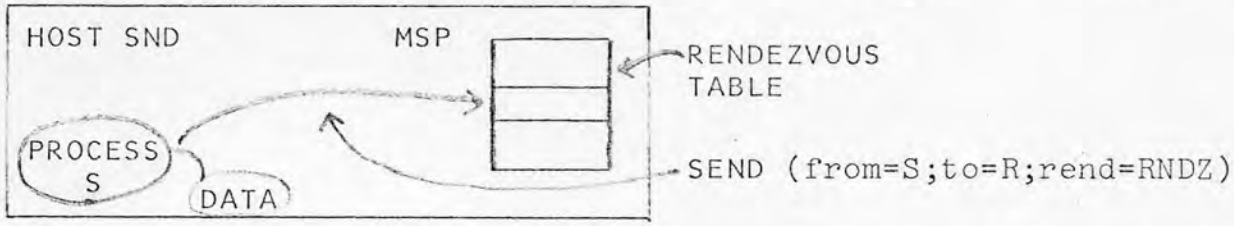
The process is greatly simplified if the rendezvous Host is also either the send Host or receive Host. Specific algorithms enumerating these sequences appear later in this note.

To clarify the basic concepts, let us look at a case involving three Hosts, to which we shall give the names SND, RCV, and RNDZ. At Host SND, process S is doing a send, and at Host RCV, process R is doing a receive. Both specify rendezvous at Host RNDZ.

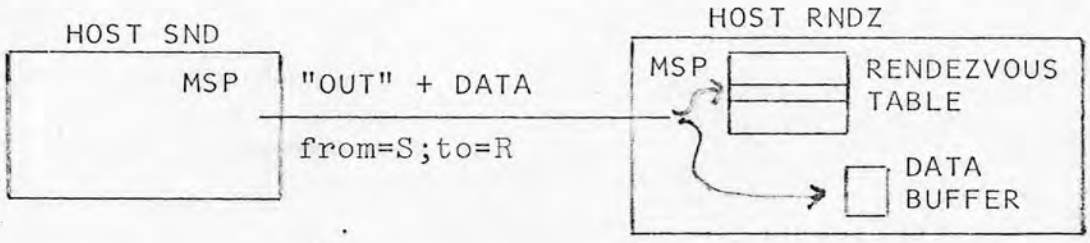
Process S now executes a SEND with

from-port-id = S, to-port-id = R, and rendezvous-Host = RNDZ.

Host SND then creates a table entry in its rendezvous table.

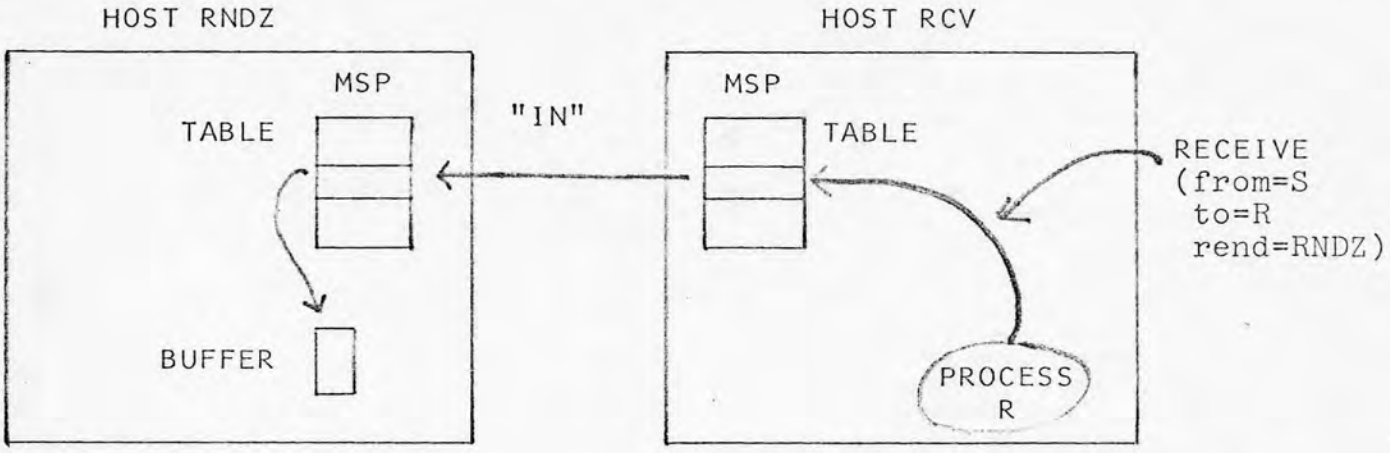


Host SND now sends an "OUT" message with S's data to Host RNDZ.



Concurrently process R at Host RCV executes a RECEIVE with from-port-id = S, to-port-id = R, and rendezvous-Host = RNDZ. As above, Host RCV creates a table entry in its rendezvous table and sends an "IN" message to Host RNDZ (see following figure).*

*Don't panic now about buffering in an intermediate Host. The time to panic is after you've read and understood the rest of our arguments.

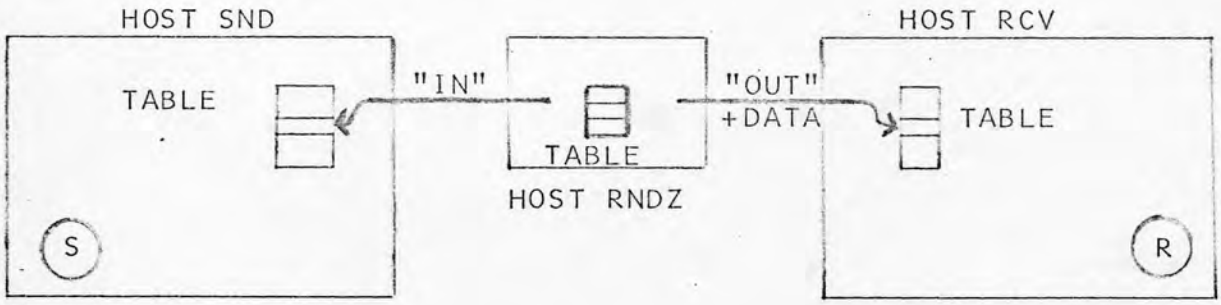


Host RNDZ now notices that the "OUT" from Host SND and the "IN" from R at RCV match one another and thus Host RNDZ takes three actions:

1. Sends an "IN to Host SND (from-port-id = S, to-port-id = R, rendezvous-Host = RNDZ).

2. Sends an "OUT" and the buffered data to Host RCV (from-port-id = S, to-port-id = R, rendezvous-Host = RNDZ)

3. Clears the entry from its table.



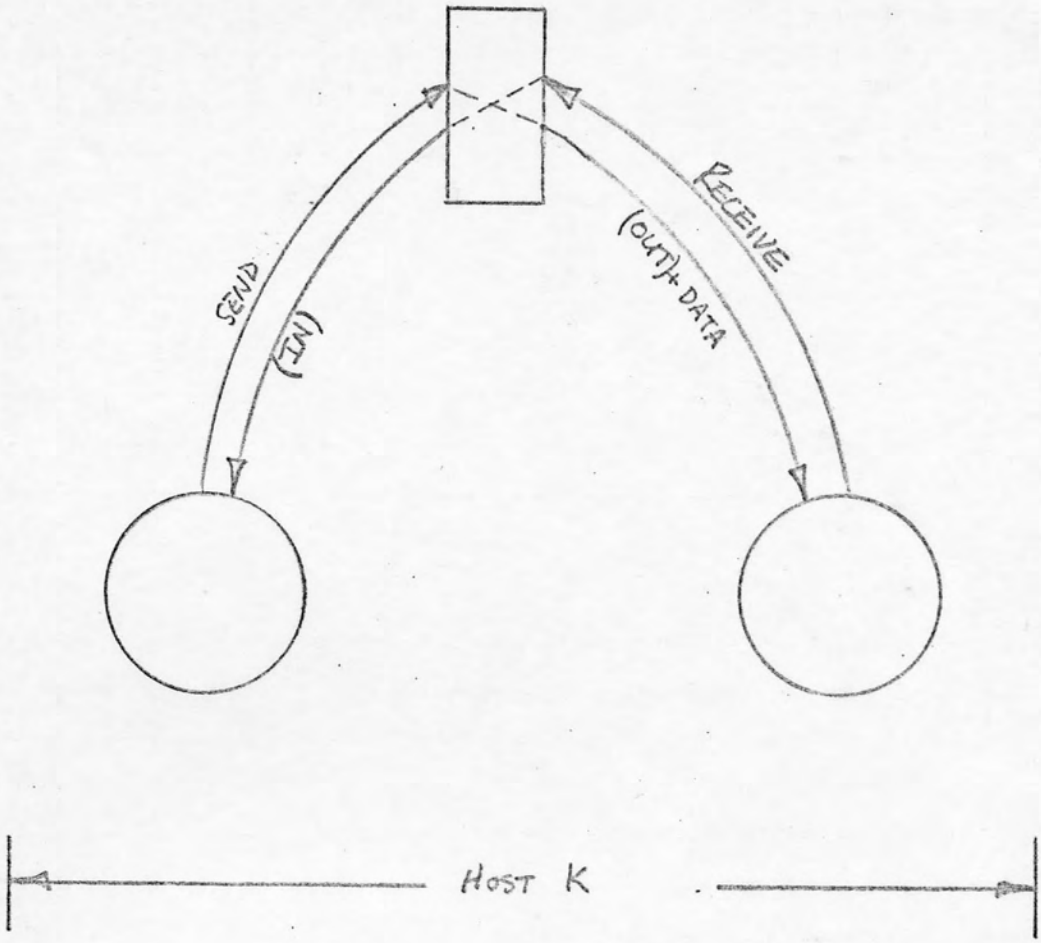
Host RCV gets the "OUT" and DATA and finds the matching entry in its table. It gives the DATA to process R and clears the entry from its table.

Host SND gets an "IN" which matches an entry in his table and clears that entry. This message serves as a combined acknowledgment and go ahead which can be passed along to process S.

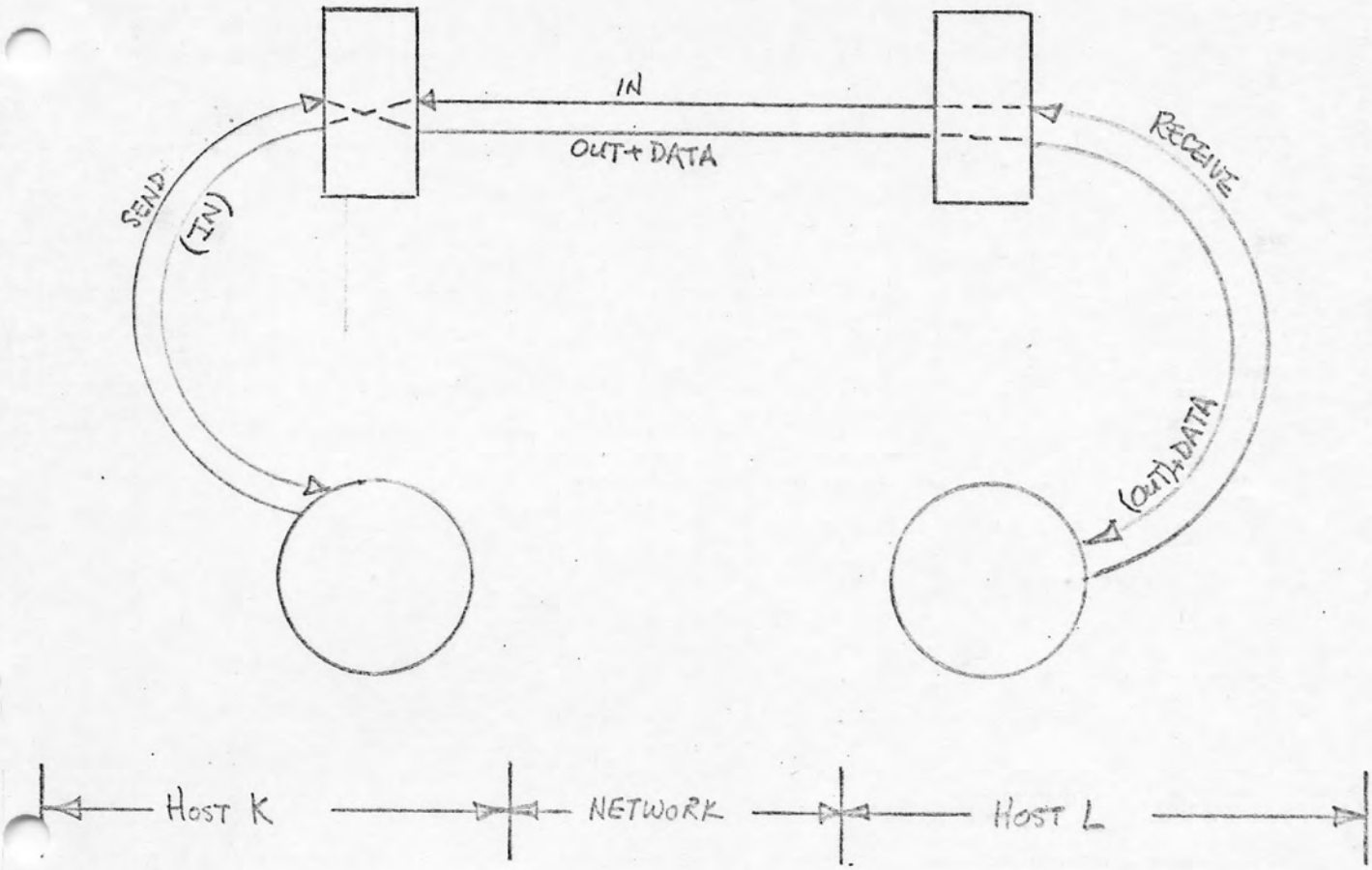
The transmission is now complete.

By both, one, or neither of the sender and receiver processes specifying a remote rendezvous Host, four important different kinds of transmissions can be made to take place. These are illustrated in the following four figures. In the figures crossed or parallel dotted lines are used to indicate rendezvous. The site of the "crossed rendezvous" is the important difference between types of transmission illustrated in the figures. Circles indicate processes. Rectangles are rendezvous tables.

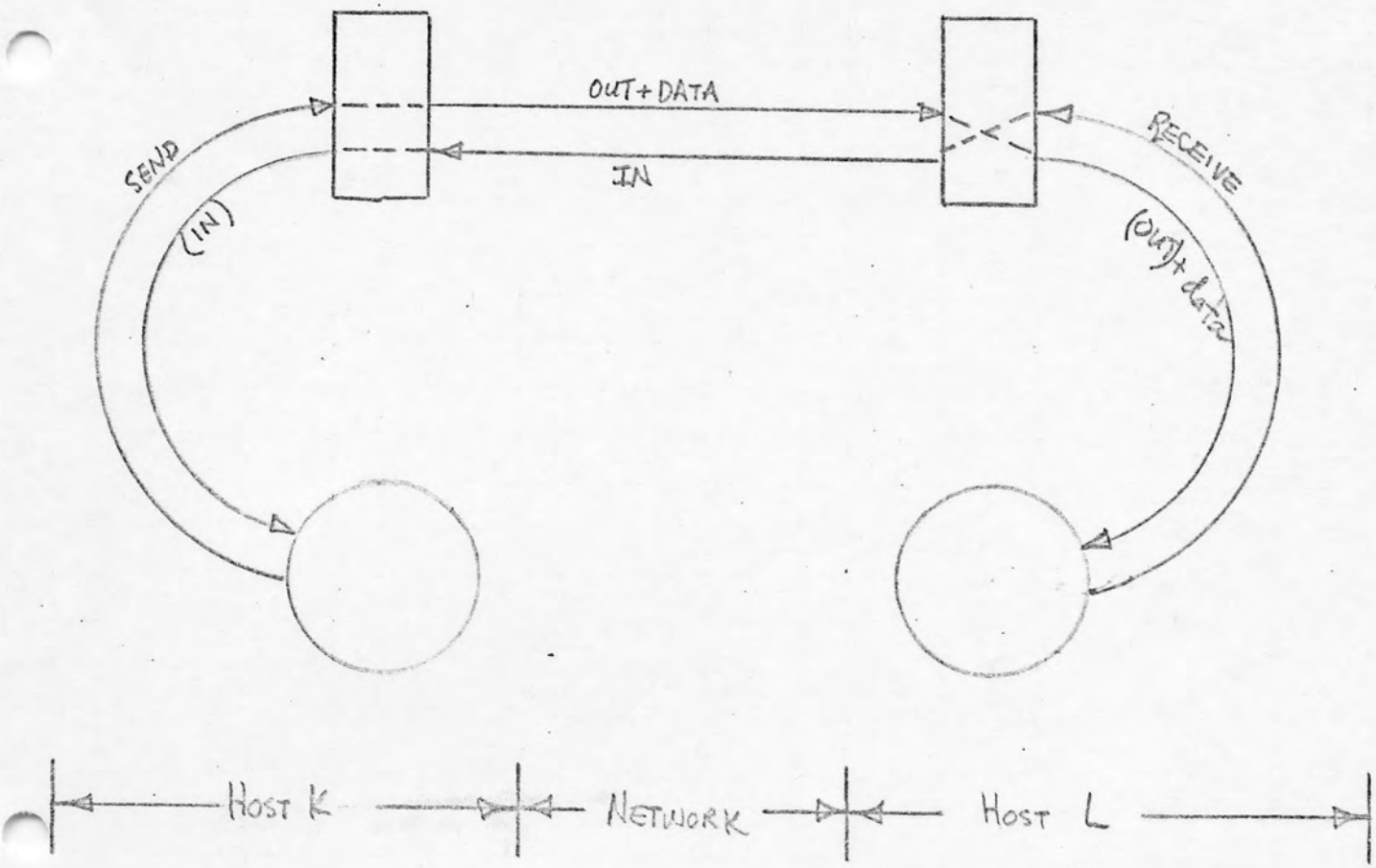
The figures also show "(IN)" and "(OUT)" messages being passed into the processes. The parentheses are used to indicate that the "IN" and "OUT" are only conceptually passed into the processes. What actually happens is implementation dependent. The process might be awakened and be given no further information if it blocked when issuing the SEND or RECEIVE. The process might be interrupted and passed some information such as the to-port-id from the IN or the from-port-id of the OUT. The process might actually be passed the complete IN or OUT message.



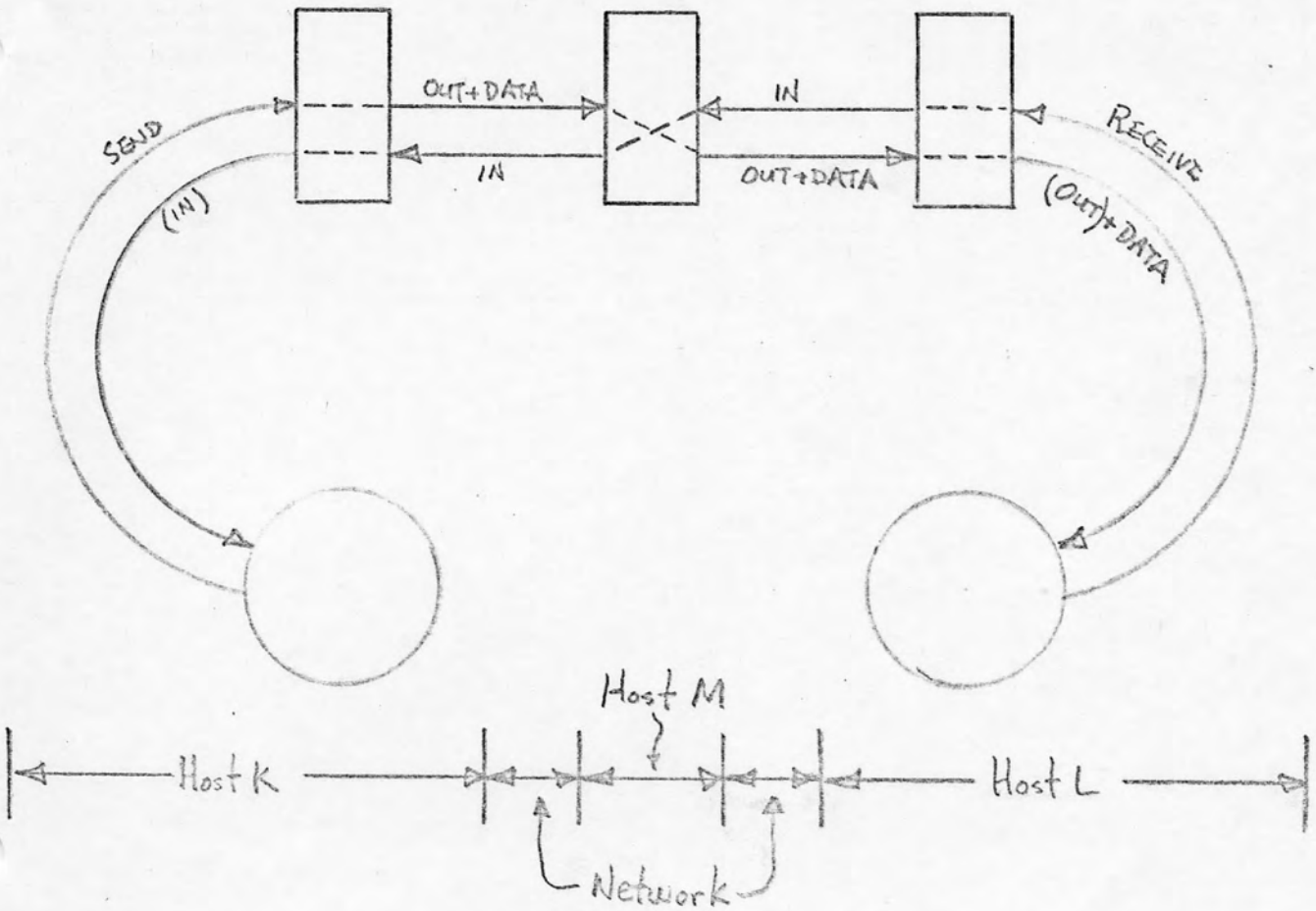
A completely local rendezvous



A rendezvous at the senders Host



A rendezvous at the receiver's Host



A rendezvous at an intermediate Host

Issues

Timeouts.

The issue of timeouts is a very sticky one. A coherent system of timeouts simplifies everything and does away with races. However, many Hosts are unwilling or unable to use timeouts, especially timeouts whose duration is specified.

Without these timeouts there is probably a need for a negative acknowledgment which goes back to the source of an IN or OUT when one is timed out. However, this now leads to races.

A negative acknowledgment (which we will refer to as a FLUSH message) could be employed by a Host to mean:

1. I have no room in my table

2. I have no more available buffer space, or

3. I no longer wish to retain the table entry/buffer.

In general, we believe that a Host should be allowed to throw away an IN or OUT+data whenever it is no longer convenient for the Host to hold the messages. This can be immediately on the arrival of a message; for instance, if the Host does not want to buffer traffic for which it does not have a user buffer. In lieu of timeouts, any time a process issues a SEND or RECEIVE, it can take it back by issuing the matching RECEIVE or SEND.

Blocking the process after a SEND or RECEIVE.

This is a question which is left implementation dependent. In general, we do not think it is a good idea to block the process after a SEND since it may want to do another to another port or even do a RECEIVE. In fact, we see nothing inherently wrong with a process doing two or more SENDs to the same port as long as

the communicating processes know what they are doing. Of course, some communicating processes will prohibit several simultaneous messages being in transit between the same ports, for instance the TELNETs may well prohibit this. However, for reasons of increasing bandwidth, etc., two processes may well want several simultaneous messages. In this case we think it is up to the processes to worry about the sequencing of messages; however, we refer users desiring their processes to take care of message sequencing to the method used in the IMP/Very Distant Host interface which is documented in Appendix F of BBN Report 1822.

Message Buffering

A few points are worth mentioning with regard to message buffering. First, most OUTs will probably be accompanied by data. Therefore, in general, since the receiver process may be swapped out, the receiver Host monitor must be prepared to buffer some data somewhere. To minimize the amount of buffering needed, the monitor could refuse further traffic from the IMP until the earlier traffic from the IMP has been written on a disk or drum. Or the monitor could have a small number of buffers in the monitor area of memory which it fills as traffic comes from the IMP, and which are swapped with buffers claimed earlier by the receiver processes as the receiver processes are swapped in. Note that the buffers may be less than the maximum subnet message size in length if the RECEIVEs never specify a longer message length--of course, this can be enforced. Finally, note that the message size, receive-port-id, etc. are available in the first 144 bits which come in from the IMP. It might be useful to read this before deciding into which buffer to read the rest of the message.

Positive Acknowledgments

Built into the system is a certain form of acknowledgment. The information is always available as to when the receiving process has done a RECEIVE. The sending Host is assured of receiving an "IN" when the receive call is issued.

Further forms of acknowledgment and validation can be implemented at the first user level, and advanced protocols will probably develop a library of such routines.

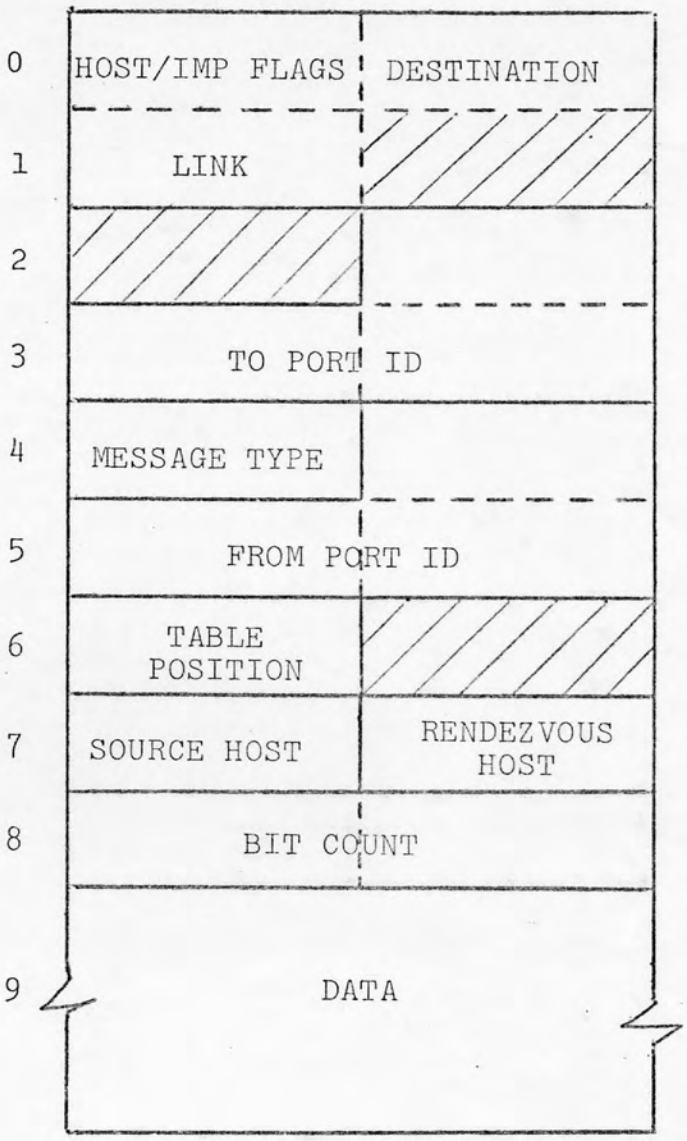
Message Header

The following section deals with the specific format of Host to Host messages and algorithms describing the proper response to a given message.

Each message begins with a 144 bit header containing the following fields:

1. HOST-TO-IMP leader (32 bits) as specified in BBN Report 1822
2. to port ID (i.e., the id of the port receiving the message) (24 bits)
3. MSG TYPE (8 bits) IN, OUT, FLUSH, etc.
4. from port ID (i.e., id of the port sending the message) (24 bits)
5. initiating Host's table position (8 bits) see below.
6. HOST "sourcing" this message (8 bits) see below.
7. RENDEZVOUS HOST (8 bits)
8. bit count of data (16 bits)

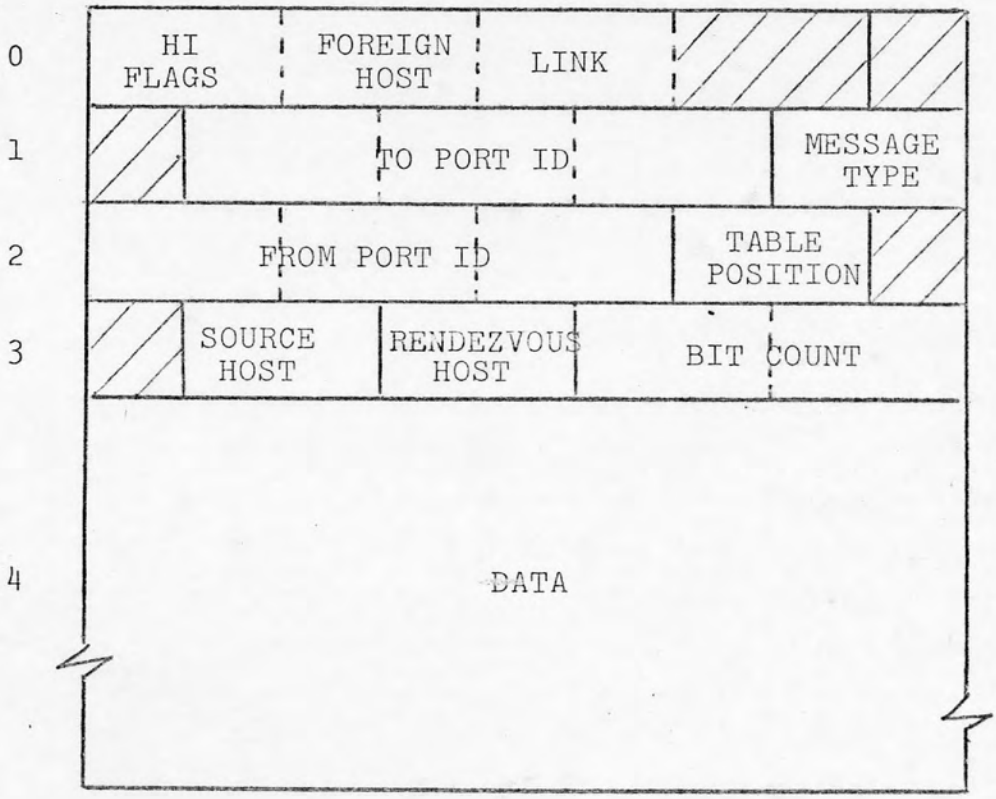
The header format has been arranged so that no data item will cross a word boundary on machines with 16, 32, and 36-bit words, except where the size of the item is greater than the word size. The actual arrangement of bytes within words is shown in the following figures for these three word sizes. For the benefit of 36-bit Hosts, bytes 4 and 13 (numbering from Ø) are unused. The 2 and 3-byte items do not cross word boundaries except for the port ID's on the 16-bit machines. This attention to packing and unpacking ease was given both for general convenience, and in particular because Hosts may wish to examine the header at interrupt level to determine where the rest of the message should go.



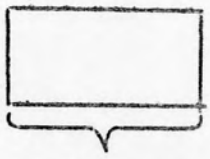
16-bit Host format

8 bits

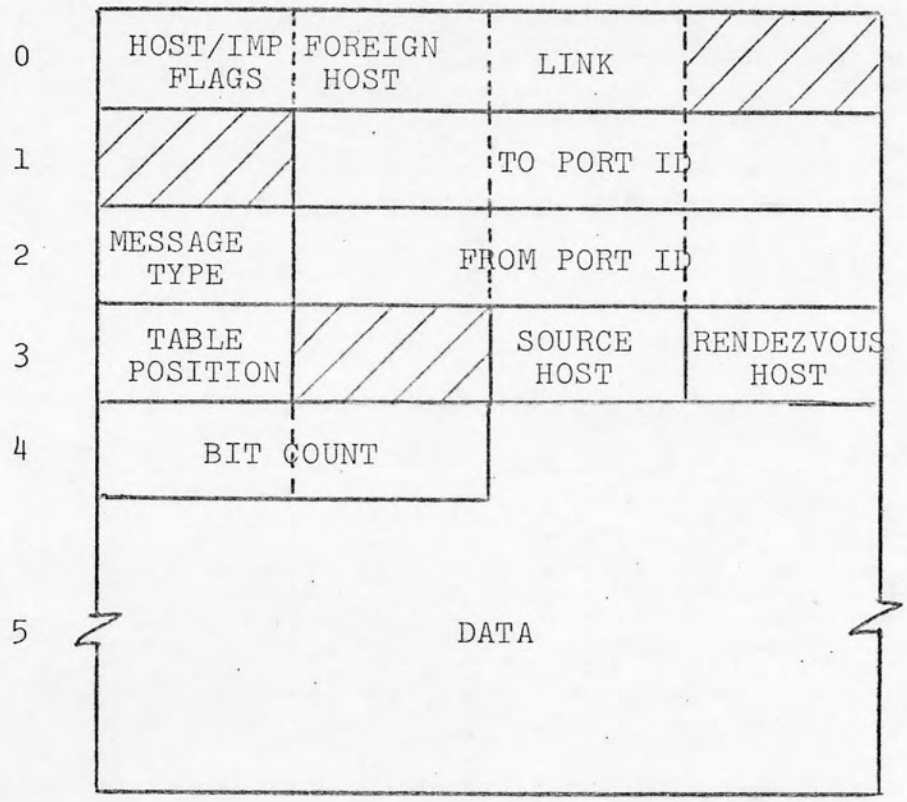
= unused



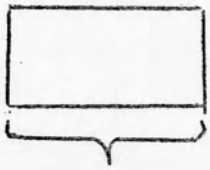
36-bit Host format



8 bits



32-bit Host format



8 bits

The fields within the Host/IMP leader are already familiar to NCP programmers; however, two points about these fields are worth mentioning. First, the destination field originally contains the number of the rendezvous Host. After rendezvous at an intermediate site, the destination field contains the source of the message rendezvoused with. Second, the link field for the MSP experiment can only contain link number 192-195. We have not taken the time to figure out a sensible allocation of these four links among all the messages which might be sent using the MSP. One alternative is to cycle over the links to increase the bandwidth of the "pipe" between any two Hosts. For the time being, until further consideration is given to this issue, we suggest each Host at a site using one (unique) link for all its communication.

The message types we have to represent in the message type field are few now: we suggest message type 2 for SEND or OUT messages and message type 3 for RECEIVE or IN messages. Message type 4 is the FLUSH message, if FLUSH is used.

The rendezvous Host field needs no comment. Except that the field is unnecessary after the rendezvous has taken place and could then be used for something else.

The bit count is a count of the data bits in an OUT message or the size of the input buffer (not including the header) in an IN message. Thus the sender process can tell from the IN message bit count when it receives the IN message how much of the data in the OUT message was accepted by the receiver process and can use this knowledge to retransmit the remainder of the message if so desired. After the rendezvous, we recommend that all of the data in the message be sent on the source of the IN message even if the OUT bit count was greater than the IN bit count. Thus, at the receiver Host the monitor has the option (if it wants to take it) of discarding the message for being too long, sending the number of bits the receiver process has done an in for into the receiver process and discarding the rest, or queueing the rest of the bits and somehow notify the receiver process that there are more bits which the receiver process can ask for.

The to- and from-port-id fields are 24-bit numbers. This size was chosen to help the TIPs. The first eight bits of a port id should be the number of the Host at which this port id was created. Note well, that this is not necessarily the Host at which the port is being used. This is necessary since rendezvous take place at intermediate sites and because ports may move from site to site. We suggest that all port ids with the first eight bits all zero be reserved for network-wide use. In particular, a port id with all 24 bits zero will be used to mean "ANY". This gives us the options of:

RECEIVE from ANY to SPECIFIC
RECEIVE from SPECIFIC to SPECIFIC
SEND from SPECIFIC to ANY
and SEND from SPECIFIC to SPECIFIC

Examples of the use of these options will be given below.

The other options (RECEIVE to ANY) and (SEND from ANY) we feel are kind of useless but would not prohibit them. We believe that in the absence of explicit specification of rendezvous Host, the use of an ANY port id in the user's system call should affect the default rendezvous site as follows:

RECEIVE from ANY -- rendezvous in receiver
RECEIVE from SPECIFIC -- rendezvous in sender
SEND to ANY -- rendezvous in sender
SEND to SPECIFIC -- rendezvous in sender

The less significant 16 bits of the port id can be used however a Host wants to. For instance, eight bits might be used as a process id and eight bits might be used as a channel specification within the specified process. We suggest that each Host reserve the port ids with the middle eight bits all zero for special uses as well known ports.

The table position field is included to help prevent costly table searches at interrupt level. Hosts sending INs and OUTs, put in the table position field the rendezvous table position of the SEND or

RECEIVE associated with the IN or OUT. At an intermediate Host rendezvous, the table position fields in the matching IN and OUT are swapped so that when the messages arrive at the opposite end, the matching SEND and RECEIVE can be found quickly. The MSP must do the swap at the rendezvous, but of course the MSPs need not fill in the table position field when first transmitting an IN or OUT in which case the information arriving in an IN or OUT will be meaningless. The general algorithm, then, is to check the table position as specified in this field and if that fails, search the whole table.

The source field is filled in in INs and OUTs by the MSP which originally sends these messages. At the rendezvous the source of each message is preserved in the message being forwarded to the final Host. When an IN or OUT arrives at a process, the process can use the source information to update its understanding of the rendezvous Host (e.g., when the destination Host and rendezvous Host are different).

Examples

The typical example.

We envision commmunication normally taking place using specific to and from ports and rendezvous at the sender. For instance, the TIP would probably send to other Hosts using this method and would certainly receive from other Hosts using this method, since no data can come from the Host until the TIP asks for it. In this "normal" method a monitor could even look at the bit count in the arriving IN-message, use that as an allocation and then simulate an OUT-message of the exact correct length.

The logging example.

Consider an example of SEND to SPECIFIC and RECEIVE from ANY with the rendezvous at the receiver. This method might be used by some logging receiver process with a well-known to-port. For instance, a measurements program to which statistics are sent from many processes throughout the net.

The program library example.

Suppose within a given time-sharing system there is a particular library routine which is available for use by any process in the network. The library process has a RECEIVE from ANY always pending at a well-known port. Eventually, some process sends a message to the library process' well-known port. This message includes the data to be processed, a port to use for sending the answer, and some money. The library process takes some of the money and sends it to the well-known port of the accounting process which itself has a RECEIVE from ANY pending. The library process then processes the data and sends the answer back to the process which requested the service using a SEND to SPECIFIC message which rendezvous at the destination where there is already a RECEIVE from SPECIFIC pending. Of course, in this message besides the answer, any change the requesting process has coming is returned.

A comment.

As can be seen from our examples, we think rendezvousing at an intermediate Host will seldom be done as the chief benefit of this comes when it is desirable to move a port (see reference 4 for a discussion of this). We would like to see all Hosts provide some (meager) amount of buffering for this purpose but would not require it. It shouldn't be too painful to provide a little of this kind of buffering—especially since a Host can throw away any message it can't handle.

TELNET

This telnet example is based on two points different from the current telnet implementation. One is a seperate communication path that is provided for non-syncronous control information (e. g. interrupts, allocations), and the other is the provision for the user telnet to specify all pertenent port IDs.

Let us identify 4 ports that each telnet will utilize;

1. A read control port - on which to do READs for interrupt, allocate, and other control information.

2. A read data port - on which to receive regular telnet communications.

3. A write control port - on which to SEND messages to the other telnet (control messages).

4. A write data port - on which to SEND data messages.

For the sake of hosts with limited storage, let us also assume that these four numbers (IDs) are all derivable from a single number, say the first. A scenario for the MSP equivilent of an ICP would be:

SERVER TELNET: receive_from_any,ID= some fixed no., say 1

USER TELNET: send_to_specific, to port_ID=1

from_port_ID= READ_CONTROL_PORT

The user's FROM_ID is then the first of the four necessary port IDs. The server will use two of them to receive messages on, and the others to address the messages to the user. The uniquexnexs of the servers read portds is aided by the fact that the high order bits of the port IDs contain the host identifier of the USER telnet.

The current telnet protocol will continue to be valid, however, it can be expanded to include an allocation mechanism, where by, for example, each telnet could specify the number of characters it is willing to accept in each message. This would enable small hosts to maintain negligable buffer space.

The Information Operator

The Message Switching Protocol itself imposes no fixed requirements on the use of the port ID's, and the problem of process identification is somewhat separated from the means used to effect communication. It is, however, very much a part of the overall issue of interprocess communication, and so we here specify a facility for handling process identification, the information operator.

One goal in a process identification scheme is to provide a means by which processes can select their own identifiers which can be guaranteed unique and can contain information meaningful to the user. Problems of efficiency prevent making the port ID's themselves large enough to accomplish this aim. Efficiency questions aside, it would appear to be ideal to allow processes to use character strings of arbitrary length to identify themselves. Uniqueness can then be easily ensured if, for example, users follow the convention of including their names in the process identification string. Further, the remainder of the name can be chosen to have some meaning related to its use with obvious advantages and convenience for users.

One solution is to establish a convention whereby the symbolic identifiers are used only during some initial phase of communication and not in every message. That is, processes identify each other initially using symbolic identifiers, but exchange local port identifiers at that time which are used for all ensuing messages.

The means of providing this facility is to establish a process at each of a number of Hosts (e.g., all server Hosts) called the "information operator". The function of this process is to associate symbolic identification strings and port ID's. A process can identify itself and/or a foreign process to the information

operator, and may request the port ID of the foreign process. The symbolic identification strings are chosen by the processes and are long enough to contain meaningful information, e.g., LOGGER, MURPHY-TESTPROG.

Communication with the information operator, whether by local or remote processes, is via the regular MSP functions. The information operator will always have a RECEIVE ANY outstanding on a well-known port. This could in general be the only well-known port in existence. A message received on this port contains the following parameters:

1. String identifying the foreign process with which communication is desired.

2. String identifying the calling process.

3. Calling process' port number.

4. A delay specification.

The format of these parameters is shown in Fig. 4. In some cases, one or more of the arguments would be null. Following receipt of a message, the information operator will, in some cases, do a SEND SPECIFIC to the calling process' port number providing the desired information or notice of failure.

The following two cases would appear to cover all functions of the information operator. They correspond to the SEND/RECEIVE SPECIFIC and SEND/RECEIVE ANY cases of the MSP.

1. Two processes each knowing the specific identity of the other wish to communicate. Each does a SEND SPECIFIC to the information operator, giving parameters 1-3, the default delay spec in this case being WAIT. When the information operator receives the second of these and notes that a match exists, it sends to each process the port ID

of the other process and deletes both strings and both port ID's from its tables. The two processes, which have each done a RECEIVE SPECIFIC in anticipation of the foreign port number, can then communicate using just the port numbers and basic MSP functions.

2. A process is set up to provide some sort of general service or information, and its name and protocol advertised. This process intends to maintain an outstanding SEND or RECEIVE ANY for the first (and perhaps only) message transaction, e.g., the library process discussed earlier. Most such processes would be receivers initially, but there might be a few cases where a SEND could be left outstanding, and a foreign process could come along and pick up the information. In either case, the service process will do SEND SPECIFIC to the information operator giving the local symbolic ID and local port ID. The foreign symbolic ID would be null, and the default delay spec is NO-WAIT. That is,

INFO (-, local ID, local port)

The information operator will enter this information in its tables but return nothing to the caller. The caller would proceed to do its SEND/RECEIVE ANY to wait for business. When another process wishes to use the advertised service, it asks the logger for the port ID of the service process, i.e.,

INFO (service ID, -, local port)

The local symbolic ID need not be specified, and the default delay spec is NO-WAIT. The information operator would SEND the port ID of the service process to the

local port of the caller, and retain the table entry for future callers. Only the service process could request the entry be deleted. If the service ID was unknown to the information operator at the time of this call, it would immediately return a failure indication, i.e., Ø.

Communicating processes would normally use the information operator local to one or the other, and like the rendezvous Host in the MSP, this would be agreed upon in advance. Service processes would normally use the information operator at their local site, and correspondingly, user processes would call the information operator at the site where the service process was expected to be available. There is no restriction on using an information operator at some other site of course, and some small and/or lazy servers could use a different Host for their service process ID's. It presents no problem for two or more information operators to have entries for the same service process, and in fact, this may be very desirable for special types of service processes which exist only one place on the net and may move around from time to time.

Processes would specify their own local port numbers, and each system would have to provide some way to help user processes do this. In TENEX for example, one would probably use the job number concatenated with another number assigned within the job. The information operator cannot supply port numbers because it will be running on a different Host than one or both of the communicants and cannot know what is a unique number for that Host. In some cases, processes would ask the "unique number process" (described below) for their local port ID, and would make it known via the information operator.

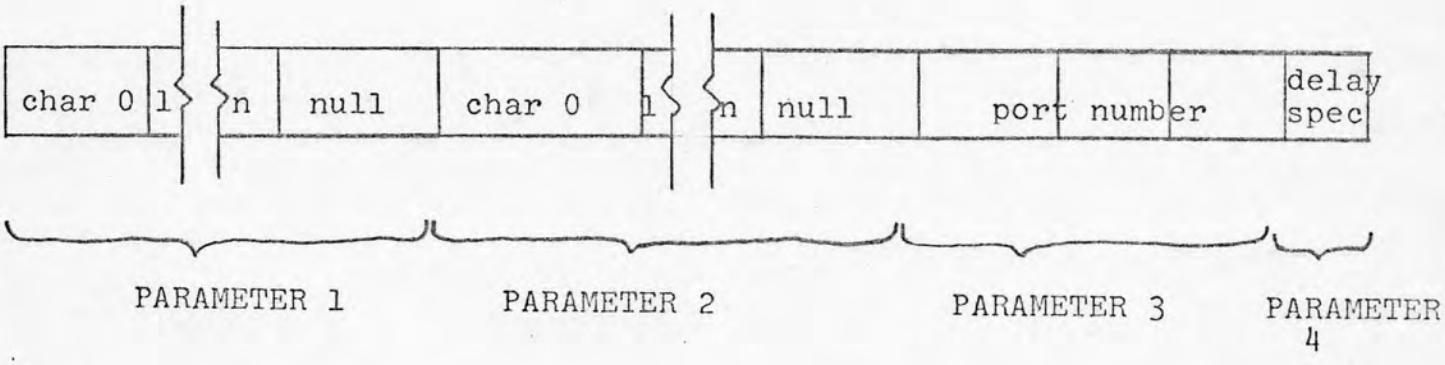
In actual practice, a few exceptions would be made to the rule that the only "well-known" port in the world is the information operator. Such exceptions would be processes common to many Hosts, e.g., LOGGER, or those in particularly frequent use. In such cases

the unique port numbers would be assigned by administrative fiat and recorded and published to all users.

The symbolic identification strings are specified to be from 1 to 39 (an arbitrary maximum) ASCII characters terminated by a null (byte of all Ø's). The characters will be 7-bit ASCII in 8-bit bytes with the high order bit set to Ø. A null string (first byte is null) is used where no argument is required.

Format of Information Operator Messages

To Information Operator: A stream of 8-bit bytes.



Parameters given:

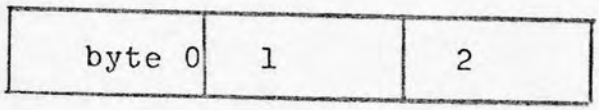
1. String identifying the foreign process with which communication is desired. (1 to 39 characters, or null)

2. String identifying the calling process. (1 to 39 characters, or null)

3. Calling process' port number.

4. Delay specification:
   0 = default
   1 = wait for match
   2 = don't wait for match

From Information Operator: 3 8-bit bytes.



Port number (24 bits) of requested foreign port if successful, 0 if unsuccessful.

Unique Port Numbers

The existence of unique port numbers is essential to the operation of the MSP. For instance, when two communicating processes specify message rendezvous at an intermediate site, the processes must be able to specify to- and from-ports which are not being used by other processes which have specified message rendezvous at the same site or else messages may be delivered to incorrect destinations. We have alluded to a method of providing unique port numbers earlier in this note. This method is to partition the 24-bit port number space into disjointed segments and give one segment to each Host in the network to distribute when it is called upon to "create" a unique port id. Thus each 24-bit Host number will consist of two major parts. The first 8 bits will be the number of the Host "creating" the port id and the next 16 bits can be used in any manner the creating Host desires. This gives each Host 2^{16} port numbers to distribute, and each Host will have the burden of distributing its segment of the port number space in a unique manner. We recommend the convention that the port numbers with the middle 8 bits equal to zero be reserved for well-known ports in the creating Host's system. We already recommended in an earlier section that port numbers with the first 8 bits equal to zero be reserved for network-wide use and in particular the port number with all 24 bits equal to zero be used to mean ANY.

Since each Host only has 2^{16} port numbers to distribute, in general port numbers will not be able to be held and used by processes for long periods of time (e.g., weeks and months). More typically, Hosts will probably implicitly "take back" all port numbers the Host has distributed each time the Host's system goes down and will redistribute the port numbers as required when the system comes back up. In other words, port numbers will not in general remain unique over the going down of the creating Hosts.

Of course, a given Host may see fit to give the same port numbers to a number of standard processes (such as the FORTRAN compiler) each time it comes up; port numbers registered with an information operator will frequently remain constant over system ups and downs.

In spite of the fact that each Host will probably not in general be able to distribute port numbers to arbitrary user processes which can be guaranteed to remain unique over a long period of time, there will still be demand for provision of long-term unique port numbers. To some, the procedure of going through the information operator smacks much too much of making a connection. These people will insist that for a variety of reasons their processes be allowed to communicate via ports whose identifiers remain constant for long periods of time. Therefore, it would be nice if at one or two places in the network, a long-term unique number service was provided. We'll call a process providing this service the Unique Number Process. The Unique Number Process would have assigned to it one segment of the unique port number space—all those port numbers, for instance, with the first 8-bits equal to 377_8. This process would have a SEND-to-ANY pending from a well-known port with local rendezvous specified. When any process wanted a unique number which it could depend on not to be used for all time or until the number is given back, it would send a RECEIVE-from-SPECIFIC specifying the well-known port of the Unique Number Process and rendezvous at the Unique Number Process' Host. The Unique Number Process' pending SEND-to-ANY would contain a unique number. Also, the Unique Number Process would have a RECEIVE-from-ANY always pending at another well-known port with local rendezvous specified. At this port the Unique Number Process would receive unique numbers which processes are giving back. The Unique Number Process would maintain a bit table 2^{16} bits long indicating the state of each of its unique numbers (free or in use) in some long-term storage medium such as in the file system. The Unique Number Process might also maintain some information about each process to which it gives a unique number

so that when the supply of unique number gets depleted, processes can be asked to return them.

It has already been mentioned that some of the process ID's registered along with their symbolic names at the information operator might be long-term unique numbers gotten from the Unique Number Process. It should also be mentioned that there would seem to be no reason, other than scarcity of storage space, that in addition to the port number through which primary access is gained to a process and which was called the process ID in the previous section, arbitrary port numbers along with their symbolic identifiers could not be registered with an information operator. For instance, rather than registering the name BBN-FORTRAN and a single port number, one could perhaps register the port numbers whose symbolic identifiers were BBN-FORTRAN-CONTROL-TELETYPE, BBN-FORTRAN-INPUT-FILE, BBN-FORTRAN-LISTING-FILE, and BBN-FORTRAN-BINARY-OUTPUT-FILE. This is perhaps at odds with standard practice within operating systems, but is consistent with the philosophy of reference 4 that communication is done with ports and not processes.

Let us now address an issue which has been ignored up to now and which was only alluded to in reference 4, the issue of port protection. We have not given this matter a great deal of thought; however, one mechanism for port protection seems quite straightforward. The heart of this mechanism is a process at each Host which we shall call (alliteratively) the Port Protection Process (PPP). The PPP maintains a list of all processes which exist at the Host and for each process the numbers of all ports which the process has "legally" obtained. Every time a process does a SEND or RECEIVE, the monitor checks with the PPP to see if the process has specified port numbers it has the right to use; i.e., those legally obtained. The PPP has some RECEIVEs always pending at well-known ports. When one process wants to pass a port to some other process, the first process sends a message

to the PPP specifying the number of the port to be sent, the Host number at which the second process resides, a port at which the second process is expecting to receive the port, etc. The PPP looks up in its tables whether the first process has the port it wants to send. If it does, it sends a message to the PPP at the destination site. The message contains the number of the port to be transferred and the RECEIVE port for the destination process. The destination PPP checks in its table whether the process has the RECEIVE port, and if so, passes the new port to the process and updates its tables to indicate the process now possesses the new port. The messages to a PPP will optionally be able to specify that a copy of a port be sent, a port be deleted, etc. The PPPs would probably have some built-in legal ports for each process, particularly the port's processes used to communicate with the PPP. The exact specification requires development but that should not be hard (see (3), (6), and (7) in reference 4). The main difficulty we see is efficient checking of the PPP's tables by the monitor for every RECEIVE or SEND without entirely supplanting the monitor's current protection system.

Flow Chart

The following section describes a flow chart for most of the MSP. A distinction is made between calls made by local processes called SEND and RECEIVE, and messages coming in over the NET called IN and OUT. An additional distinction is made between calls (or messages) with a local rendezvous and those with a foreign rendezvous Host.

Since the code is quite similar, the distinction need not be made, but will be included for the sake of clarity.

It is assumed that the MSP has table provisicns for the following items:

source of message
rendezvous Host
FROM-PORT-ID
TO-PORT-ID
table position
type of message
data size and location
data about the user process

User does a SEND or RECEIVE

A. Rendezvous is at a foreign host

    1. Store the appropriate table data

    2. send a message to the rendezvous host

        a. SEND: OUT + DATA

        b. RECEIVE: IN

B. Rendezvous is local - look for entry in table

    1. Entry NOT found: create entry with appropriate data

    2. A matching entry exists in the table:

        a. RECEIVE: give user the data

        b. Send a message to the other host (as specified by the source field of the original msg)

            1) SEND: OUT + DATA

            2) RECEIVE: IN

        c. Alert user to the fact that transaction is complete

        d. Clear table entry

An IN is received over the NET - search table for matching entry.

A. No matching entry create an entry with appropriate data.

B. A match exists

   1. Entry was cause by a local SEND

      a. Send "OUT _ DATA" to source of IN

      b. Inform user of transaction

      c. Clear table entry

C 2. Entry was caused by an OUT received over net - acting as third host.

      a. Send IN to site that created table entry

      b. Send OUT + DATA (previously buffered) to site sending the IN

      c. Clear table entry.

An OUT + DATA is received over the NET - search table for matching entry

A. No match is found
    1. buffer data
    2. create appropriate table information

B. A match is found
    1. Table entry was caused by locally executed RECEIVE
        a. give data to the user and alert him to its existence
        b. send a matching "IN" to the source of the "OUT"
        c. remove entry from table.
    2. Table entry was caused by the receipt of an "IN" over the NET, thus we are acting as a third party host
        a. send the "OUT + DATA" to the host stored in the table
        b. send an "IN" to the host from which the "OUT" had just arrived.

MSP Variations

It may be of interest to the reader to know of some of the other MSPs we have considered while arriving at the present one.

The simplest we considered is an MSP based on all rendezvous being done at the destination Host. The sender process sends an OUT-message plus the data to the destination Host. The receiver process does an IN which stays at the receiver's Host. The OUT and RECEIVE rendezvous and the data is passed to the receiver process. The transmission is now complete, except in some variations of this MSP an acknowledgment is sent to the sender process. This MSP has a couple of disadvantages: In the simplest formulation, the RECEIVE had to be waiting when the OUT+data arrived, otherwise the out and data were thrown away. This puts too tight a constraint on the timing of the SEND and RECEIVE, especially since the sender and receiver processes can be a continent apart. However, if the IN is allowed to arrive first and must be held until matched by a RECEIVE, the monitor must buffer an indeterminate amount of data in all cases including the normal one. Further, basing everything on rendezvous at the destination makes the process of moving a port difficult.

The next simplest MSP we considered was the IPC of reference 4. This works just the opposite of the above described MSP in that it is based on almost all rendezvous being done at the source Host with two special messages to handle the relatively uncommon cases when a rendezvous must be done at the destination or an intermediate Host. This system, its advantages, and disadvantages is discussed at very great length in the reference.

A third variation on the MSP, suggested by Crowther, is the same as the present MSP in that the OUT and IN rendezvous at a process specified rendezvous Host and the OUT is sent to the source of the IN and the IN to the source of the OUT, but the data is not sent along with the OUT. Instead, when the OUT finally reaches the

source of the IN, another message is sent from the receiver Host to the source Host requesting the data to be sent. The data finally is transmitted to the destination in response to this data request message. Our main objection to this system is its lack of symmetry, but we do recognize that it does not require any Host to buffer data for which a process has not set up an input buffer and perhaps for that reason it is a better system than the MSP we are presenting.

In the last MSP variation we considered, the difference between SEND or RECEIVE and OUT or IN was discarded. In this case only one message is used which we will call TRANSFER. When a process executes a TRANSFER it can specify an input buffer, an output buffer, both, or neither. Two processes wishing to communicate both execute TRANSFERs specifying the same to and from port ids and the same rendezvous Host. The TRANSFERs result in TRANSFER-messages plus data in the case that an output buffer was specified which rendezvous at the rendezvous Host. When the rendezvous occurs, the TRANSFER-messages plus their data cross and each is sent to the source of the other. The system allows processes not to know whether they must do a SEND, or RECEIVE and is (perhaps) a nice generalization of the MSP presented in this note. For instance, two processes can exchange data using this system, or two processes can kind of interrupt each other by sending dataless TRANSFERs. This variation of the MSP is a development of a suggestion of Steve Crocker. Its disadvantages are: (1) unintentional matches are more likely to occur, (2) rendezvous selection site is more complex, and (3) it's hard to think about.

Appendix

A System for Interprocess Communication in a Resource Sharing Computer Network. Communications of the ACM, April, 1972. Permission to reprint this paper was granted by permission of the Association for Computing Machinery.

N.B. The ideas of section 4 of the following paper are in no way critical to the ideas developed in section 3--DCW.