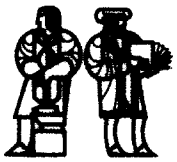
LABORATORY FOR COMPUTER SCIENCE
(formerly Project MAC)



MASSACHUSETTS INSTITUTE OF TECHNOLOGY

MIT/LCS/TR-184

FACILITATING INTERPROCESS COMMUNICATION IN A HETEROGENEOUS NETWORK ENVIRONMENT

Paul H. Levine

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Office of Naval Research under contract no. N00014-75-C-0661

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

This blank page was inserted to preserve pagination.

FACILITATING INTERPROCESS COMMUNICATION IN A HETEROGENEOUS NETWORK ENVIRONMENT *

by

Paul Howard Levine

ABSTRACT

Passing information among processors with different internal data formatting schemes has proven to be a major complication to computer networking efforts. Data format translation is necessary to support information exchange in a heterogeneous network environment. Three strategies for performing this translation for communications between a message sender and receiver are: translation by the receiver, translation by an intermediate translator, and the use of a standard intermediate format. The standard format is shown to be the most responsive to a set of general network design principles.

The implementation of an intermediate format based interprocess communications scheme requires a mechanism for passing the semantic description of each string of data bits. Two alternative mechanisms for passing this information are discussed, and data "tagging" is selected as the more flexible. Other implementation considerations are examined, including possible problems in performing translation and the relationship formal translation has to other network message handling functions.

THESIS SUPERVISOR: Prof. Liba Svobodova

COMPANY SUPERVISOR: Dr. Robert L. Gordon

* This report is based upon a thesis of the same title submitted to the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, on March 25, 1977 in partial fulfillment of the requirements for the degrees of Bachelor of Science and Master of Science.

This empty page was substituted for a blank page in the original document.

ACKNOWLEDGEMENTS

I would like to extend my thanks to my thesis advisor, Professor Liba Svobodova. Her substantive suggestions, tireless editing and constant encouragement contributed greatly to the final form and content of this thesis. My sincerest thanks also go to Dr. Robert L. Gordon for having suggested this area for research and for his valuable insight and support. I am also indebted to the many other people, particularly W. Allan Clearwaters, who contributed to my understanding of the issues through countless hours of discussion. Then there are my greatest supporters, Allan M. and Gloria Levine of Quincy, Massachusetts, whose perpetual support and encouragement made this thesis possible.

This research was funded in part by Naval Underwater Systems Center IRAD funds, project number A75030, W. A. Clearwaters principal investigator.

TABLE OF CONTENTS

CHAPTER I

Heterogeneous Computer Networks

A recent trend in computer systems research has been towards the investigation of and experimentation with computer networks. Besides the extensive work on ARPANET <Frank, Heart, Metcalfe1, Crocker, ARPA> and other geographically distributed computer networks <Pouzin1, Wood>, the possible implementations and applications of local computer networks is also being researched at an ever-increasing number of laboratories across the country <Fraser1, Farber1, Metcalfe2, Mills, Binder, MRG, Hirt, Chen, Wulf, Swan>. Passing information among processors with different internal data formats has proven to be a major complication to these computer networking efforts <Farber2, Millstein, VanDam2>.

1.1 Networking

The definition of a computer network can be phrased in terms of a network's form and function. One such definition asserts that

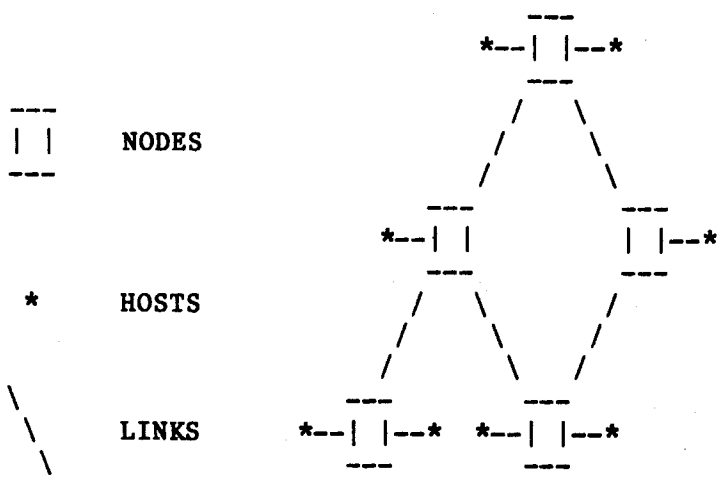
> ...a computer network is defined to be a set of autonomous, independent computer systems [the form], interconnected so as to permit interactive resource sharing between any pair of systems [the function]. [<Roberts> p. 543.]

Such a network is embodied as:

> 1) a collection of hosts (computers) providing service to a user (either an end user or another host computer), and 2) a subnetwork providing communication among host computers, users, or both. [<Kimbleton2> p. 129.]

A typical network is depicted in Figure 1-1. The subnetwork is built from nodes and the communications links that serve as the data paths between the nodes. The nodes interface the network hosts to the subnetwork <Crowther>. As shown, it may be possible for a single node to support the network demands of more than one host.

Study of techniques for supporting general inter-computer information exchange is motivated by the proposed uses of computer networks. The most often cited rationale for computer networking is probably the facilitation of "resource sharing." <Chen, Farber1, Mills, Roberts, Thomas2> However, especially in the case of geographically local networks, much attention is now being focused on computer networks as the hardware/firmware base for distributed systems <Kimbleton1, MRG, Rowe, Swan, Thomas1>.



TYPICAL COMPUTER NETWORK

FIGURE 1-1.

Resource sharing networks are those computer communications systems that provide access to remote hardware and software services. This may include the use of standard peripherals, special hardware devices, information or software utilities through the network. The advantages are primarily economic. With the cost of the processing unit becoming a smaller percentage of the total system cost for an installation, concern has shifted to the cost of providing system services. By increasing access to high cost software, large data bases, or expensive peripherals, the need for redundant facilities can be minimized.

A distributed computer network has been described as a system that supports the execution of a user task by using multiple components throughout the network, each component performing some part of the required

task <VanDaml, Wecker>. The subtasks communicate over the network to accomplish the complete assignment. The principle distinction between this and a resource sharing network is that a distributed system offers the end user an interface to a single coherent system and yet employs a network of computers to process his request <Elovitz, Enslow>. Networks supporting distributed systems can transparently offer a user the performance advantages of load sharing and parallel processing as well as the reliability feature of hardware modularity and modular redundancy.

1.2 Interprocess communication

The transfer of information between computers in a network can accurately be described as data exchange between distinct processes active on different processors. This view is a natural one for network based distributed systems. One model of such a system consists of several procedures for each task, running on different processors and performing the required interprocess communications across the network. However, viewing network message passing as a case of interprocess communications is also appropriate for resource sharing networks.

> It is useful to think of resources as being associated with processes and available only through communication with these processes. This is a viewpoint that has been successfully applied to time-sharing systems and has been more recently been suggested to be an appropriate view for computer networks. Consistent with this view, the fundamental problem of resource sharing is...the problem of interprocess communication....The view

> is also held that interprocess communication over a network is a subcase of general interprocess communication in a multiprogrammed environment. [<Walden> pp. 221-222.]

Considering the messages passed between network hosts to be instances of interprocess communication provides insight into the mechanisms needed to support inter-host network communications. Specifically, any message passing scheme must support the transfer of the kinds of messages that are the units of communication between processes. Communicating processes may need to exchange only boolean values or entire data files. The ALGOL-like languages allow the interprocess exchange of the primitive data types (INTEGER, CHARACTER) as well as more complex structures (STRINGS, ARRAYS). To facilitate inter-host communications, then, a network message passing strategy must support the transfer of both simple and composite data types.

The problem of passing information across a network can be broken down into two stages. First, regardless of the information being passed, a protocol must be established that assures the bit integrity of exchanged messages. Schemes for this level of hand-shaking usually employ a three part structure, including a header, the data bits to be passed and a trailer (Figure 1-2).

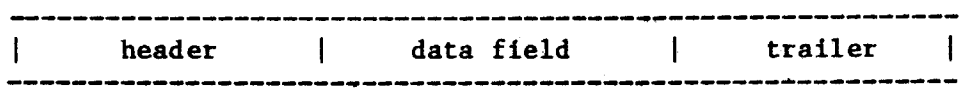


Figure 1-2.

The header contains a destination field as well as some, possibly complex, message control information. The data field is usually transparent to the message passing hardware and protocols. The trailer contains the error-checking codes and status information.

This level of interprocessor communication has been examined extensively in the literature <Bhushan, Metcalfe2, WhiteG> and is not addressed in this study.

The second stage of information transfer over a network is the interpretation of the bits in the data field. Because the internal representation of data is different across products of different computer manufacturers and even computer products from the same manufacturer, some reformatting of the information is necessary to support information transfer in networks.

1.3 Heterogeneity

Little has emerged in the way of techniques for allowing different kinds of processors in a heterogeneous computing environment to exchange information in a general way. Rather than concentrating on the semantic content of interprocessor messages, much of the effort has been directed towards simply getting one host computer in a network to accept unexamined binary data from another. To this end, several topologies for computer interconnection have appeared, as well as schemes for insuring delivery of a binary packet from a sending host to its

intended receiver. Yet to be addressed is the problem of also transmitting the semantic content or "meaning" of the bits in a general way. Passing the bits themselves is only the first step towards interprocessor communications. At this point, a mechanism is needed to support the passing of information.

> Insuring the integrity of a binary string as it is moved from the memory of one processor to another, has not been easy. Many complex issues concerning error detection and recovery, message routing, system response and component loading have been faced, only to uncover the next set of problems, that of providing adequate semantics for the transferred bit strings...Suppose the text file of one system requires a carriage return and line feed as a line separator, while another system requires a carriage return. Who should be responsible for the inclusion or exclusion of the line feed? Worse yet, what do we do about incompatible integers, character sets, and floating point data types? Current solutions are worked out by cooperative programmers, not processors, and severly limit solutions to dynamic reconfiguration and load sharing among connected processors. [<Gordon> p. 4]

These data formatting problems have been essentially avoided in some networks by inter-connecting strictly machines that use similar internal data representations <Fredericksen, Mills, Thomas1, Swam, Wulf>. Processors in such a homogeneous computing environment require no data format translation to exchange information. They are assured by common hardware and software design that the semantic content of their passed data will be correctly understood by their intended receiver if the bit content is delivered correctly.

This approach to distributed computing, although attractive, is not sufficient to support the growing demand for connected computers.

Clearly, homogeneous machines provide a processing environment more hospitable for inter-computer message transfer. Unfortunately, many organizations have discovered too late the advantages of inter-connection, having already acquired machines of different manufacture for separate computing requirements. The capital investment represented by these computers, in both hardware and software, often prohibits their replacement with more compatible counterparts. Ignoring the data translation problem because it can be avoided in homogeneous environments is being unresponsive to the real needs of a large segment of the computing community.

Conveying meaning of transmitted bits in a heterogeneous environment is not as simple as it may first appear. The difficulty arises because of the total lack of an industry standard for the internal representation of information in computers. The market is filled with machines of every description: they support sign-magnitude, or one's or two's complement arithmetic, 12, 16, 24, 32, 36, 48, or 60 bit word lengths, and unique floating point number representations. At the software level, there are different ways to represent complex numbers, vectors, arrays and other data structures. Discrepancies exist even in the case of character data. Although the ASCII character set has become an industry standard, different machines still ascribe different contextual meanings to control characters such as form feed, line feed, tab and carriage return.

1.4 Summary

There are several methods for facilitating general information exchange between two dissimilar processors. This report examines these methods in light of a set of design considerations for computer networks. There are problems inherent to any scheme for presenting data in different formats to processors with different requirements and these, too, are examined. The report does not claim to solve the problem of inhomogeneity. Rather its intent is to examine the alternatives, and to offer an adaptable and extensible scheme for facilitating interprocessor communication in heterogeneous environments.

Chapter I has attempted to review computer networking and the relationship between host-to-host message passing and interprocess communication. The problem of moving information between heterogeneous processors is introduced, and the intent of the research stated.

Chapter II proposes design considerations for networks and network supporting functions. Section 2.2 presents three mechanisms for data translation that can facilitate data transfer in a heterogeneous environment. The summary evaluates these strategies with respect to the stated design considerations.

The mechanism found most responsive to the design goals in

Chapter II is the subject of Chapter III. The mechanics of the scheme are presented and alternative designs argued.

Chapter IV discusses problems that are inherent to any data translation mechanism, and suggests practical ways to deal with these problems. Chapter V introduces implementation considerations for format translators, and Chapter VI presents some areas for future study.

CHAPTER II

Facilitating Interprocessor Communications

The development of any technique acceptable for providing communications in a heterogeneous network environment must be guided by the anticipated operating requirements such a facility may face. Any such scheme must be flexible, extensible, provide enough functionality to compensate for the cost and overhead it incurs, and also be easy to use. From its beginning, the research reported herein has used a set of design principles as a basis for the evaluation of strategies to provide general interprocessor communications. The following section describes those design principles.

2.1 Design principles

Process Addressing -- Almost all of the documented computer networks in operation today support node to node message transfer. In the header of the message being sent, the transmitting node designates a second node on the network as the target for that message. Implicit in this method of information exchange is the binding of each communicating process to a network node.

To send data, a process at one network site builds a message and addresses it to the network node that represents the process receiving the

data. The sending process then passes the message to the network to be delivered. When it is accepted at its destination, the message is passed to a process running in a host at that site for which the information in the message was intended. It has been suggested <Farber1, Walden> that rather than addressing messages to a receiving network node, the transmitting process address messages directly to the receiving process. The subnetwork interface at each node is then responsible for finding and accepting messages addressed to any processes currently active at its node.

Process addressing has several inherent advantages compared to the more standard technique of node or processor addressing.

> The most attractive feature of this approach is that it allows a uniform conceptual point of view. The processor oriented view requires a rather continual translation from process name to the process that supplied the service. This continual translation is required for reliability and flexibility. [<Farber1> p. 7.]

The added flexibility offered by process addressing is a result of having the physical location of the receiving process be transparent to the sender. This transparency facilitates the dynamic relocation of running processes on a network for purposes of load leveling or in the event of partial node failure.

Since a message is not directed to a particular processor it can

reach several instances of a process, each running under the same process name at different network nodes. By allowing the duplication of names, the communications system facilitates broadcast announcements to several or all network nodes. It further supports identical processes on different processors for increased reliability through parallel redundancy.

Process addressing is not without a serious technical problem. Besides depending on a network-wide process naming scheme, inherent in the concept of location transparency is the requirement that every node be allowed to examine every message. Each must compare the name of the process being addressed with the names of the processes active at its location. While reasonable for some network topologies, such as a simple bus <Metcalfe2>, ring <Farber1>, or star, it is out of the question for others. The advantage of a tree network <MRG>, for example, is its ability to favor communications paths between certain processors. This advantage is meaningless when every message must be circulated to every node. In the case of multiply connected store and forward packet switching networks <ARPA,\ Frank, Heart, Metcalfe1, Pouzin>, a mechanism would be necessary to insure that every packet travelled through every node.

However, because some network structures are suited to it, sending messages by process name is a legitimate operating feature for a mechanism that supports interprocessor communications. Facilitating

process addressing is, therefore, a design consideration for such a mechanism.

Easy Expansion -- Short of special purpose networks designed with particular components and applications in mind, extensibility is an important consideration in network facility design. In the case of a general interprocessor communications scheme, the issue takes two forms.

First is the expansion of the network itself. The ability to add nodes to a network with a minimum of disruption to the operation of already existing network nodes has been a design consideration for and been achieved by many networking efforts <Binder, Farber1, Frank, Fraser1, Mills, Metcalfe2, Pouzin1>. It is equally essential that incremental expansion of a network should cause minimal disruption to a mechanism that provides data representation compatibility between processors.

The second concern is for the addition or modification of data formats in use by the processors at nodes already part of a network. The design of a network-wide communications scheme must anticipate the need for such changes and provide the means for handling them with a minimum of effort.

Functional Sophistication -- General interprocess information exchange requires the support of a communications facility that allows for the trans-network movement of a wide range of data types. The size of the subset of data types handled by a mechanism and the flexibility with which the supported data types can be manipulated are a measure of that mechanism's sophistication.

Strategies for relatively simple information exchange between processors in a heterogeneous environment have already appeared. An example of such a strategy that handles a single data type is the ARPANET TELNET <ARPA2> described in a later section. TELNET provides a protocol for sending text across the ARPA network between host processors. Each processor may store text in any of the several commercially used internal representations for characters.

The single data type provided by TELNET does not offer the sophistication required to support general interprocess communications. Although characters are the most often considered data type, they are only a very small subset of types used to transfer information between processes. Textual information is more easily handled because of the industry-wide recognition of the USASCII character set <Bhushan>. However, providing data transfer in a heterogeneous processing environment requires provisions for handling data types without a standard format as well.

Application-level Transparency — Removing the programmer from the details of machine operation has become a generally accepted notion among members of the computer field. An example of this kind of thinking is found in <Corbato>. In line with this philosophy is the logical separation of the internal and external data formats used in ordinary data processing. In this sense, internal data formats are hardware dependent and external formats are those conceptual items with which the applications programmer and the human end-user must deal.

The importance of this distinction was noted as early as 1968 by the National Bureau of Standards.

> ...the internal representation of data is concerned with the manner in which particular computers and other hardware store and move the data around inside the system. This is not the user's province, and there should be no imposition on him as to how it is done. For example, it should be of no concern to him whether the hardware represents his data by means of 6-bit, 8-bit, or 64-bit units within the computer. He should have no concern with "packing" and "unpacking" of characters within the computer words. He should not be troubled with physical file units. These are all aspects of the supporting technology... [<Little> p. 93]

Applications of this view of data maintenance to the problems of general process communications in a network demands that the necessary data reformatting be transparent to the applications programmer. He should

have a consistent view of data items regardless of their actual representation or where in a network they may originate.

Minimal Host Overhead -- Providing an interprocess data communications service among heterogeneous processors requires some level of data reformatting. In large part, the amount of overhead required to perform data translation depends on the strategy used to achieve format compatibility. A general scheme for facilitating such communications and an associated implementation, however, should not presume on the sophistication of processors connected to the networks as hosts.

This concern is slightly apart from the development of a data communications scheme, as it is more an issue of a scheme's implementation. The distribution of network related functions, such as message formatting, between the subnetwork communications components and their associated hosts is not a settled question for networking in general. The issues are discussed in Section 5.2. Nonetheless, in order to be responsive to the needs of those network environments that include hosts with limited processing power, a mechanism for providing interprocessor communications must be designed with the demands it places on host processors in mind.

Minimal Message Overhead -- The limiting of overhead added to the network hosts is required to support the use of devices with limited or inflexible processing capabilities as hosts. Minimizing message overhead is a consideration that addresses the total cost of message processing. These two design goals combine to minimize the overhead caused by message passing, and then force as much of the remaining overhead as possible into the node. that remains into the network hosts.

This total overhead includes the processing required at each network site (host, subnetwork node, or special network processing modules) and traffic on the communications links. Overhead is represented at each network site by the maintenance of a software/hardware base plus the processing time to perform the message reformatting. At the communications level, message overhead appears in the form of header and information-describing bits that increase the length of data messages being carried. Reducing the overhead at these levels increases the effective thruput of each message transmission and reduces the message processing required at each network node.

Reliability -- The reliability of a network based system is a function of several different aspects of design and implementation. The

categories of areas that must be considered include: failure of a communications link, failure of the node hardware supporting message handling functions, surfacing of hardware/software design errors, and system fault-tolerance for each of these categories. These issues are involved, however, not all of them are relevant to a discussion of facilitating meaningful host-to-host information exchange. Thus, a simplified view of reliability is adopted.

The interprocess communication facility considered here is a functional improvement to a rudimentary bit-passing communications scheme. However, increasing the functionality of a network communications system involves increasing the number and/or complexity of required system modules; both the size and the sophistication of a mechanism are directly related to failure through error in design. It follows, therefore, that an important consideration in the design of a strategy to increase system function is to limit the number and complexity of additionally required hardware and software modules. It is with this limited view of reliability in mind that the following strategies are evaluated.

2.2 Possible strategies

Several schemes for facilitating communications in a heterogeneous environment are presented in this section. The common

basis for each of these strategies is their need for a format translation from the internal data format used by the transmitter to that used by the intended receiver.

> It is well recognized that hosts in a heterogeneous network use different bit patterns for encoding information. Data translation is the basic capability which permits hosts to communicate with each other in spite of their differences. It follows that a data translation capability is central to any effective capability to communicate among heterogeneous computers. <Kimbletonl> p. 555.]

The differences in the schemes discussed below lie in the steps that each requires to perform that translation.

2.2.1 User translation

The simplest, and so the most often adopted, attitude towards providing formatting for data transmitted over a network attempts to avoid the issue completely. In some cases, networks consist of a totally homogeneous collection of processors and software environments, and so never require any data translation <Haverty, Mills, Swan, Wulf>. However, the majority of currently operating networks that have adopted this approach do not fall into this category. Networks such as ARPANET, DCS, CYCLADES, and ETHERNET are designed to support heterogeneous processor environments, yet leave the data translation necessary to facilitate general interprocessor communications

totally to the applications programmers. (ARPANET does provide some format translation for specific types of process to process communication. These will be discussed later.)

For network user communities for which the coordinated use of more than one network processor is an infrequent requirement, handling data format incompatibilities at the applications level on a special case basis may be sufficient. This seems inappropriate, however, for heterogeneous networking efforts investigating the issues relevant to distributed data bases and distributed operating systems. It is these functions especially that require a high degree of interprocessor interprocess information exchange.

DCS is an example of such a network project. The installation of a distributed operating system on a fully heterogeneous DCS-type network has already been discussed <Rowe>. The DCS project head agrees that a data reformatting mechanism would be an important addition to his research efforts, however, the problems of general format translation are too complicated to be addressed by his researchers at this time <Farber2>.

2.2.2 Receiver translation

One approach to actually facilitating data communications is to provide data translators at each node eligible to receive messages from the network. In such a scheme, the transmitter performs no data reformatting. To send a message, a process only forms a data block to be transferred using the internal format native to the processor on which it is running. The data traverses the network in its original format, but carries with it, in some network-wide format, a description of the processor at which the message originated. The transmitting process can always know the nature of its supporting host and insert this information into the message being sent.

When a message is accepted at a network node, the receiver reads the message field that identifies the transmitting processor type. It then performs any conversion necessary to translate the sender's internal format into the internal format appropriate to the receiving host. The identity of the transmitter needs to (and can) be known to the receiver, while the receiving node remains unknown to the transmitter. This condition supports the node independent (or process) addressing previously discussed.

A major disadvantage of such a scheme is that for each processor to be able to interpret messages from every other, it must have access to a

translator that can resolve each possible dissimilar processor pairing. At every node there must be a translator to convert each of the internal formats used on the network to the internal format used at that node. In the case of 'n' different types of processor, this method of operation requires that the network support 'n(n-1)' translators to be completely general, since each processor must be able to communicate with all of the other types of processor on the network. For diverse environments, this number quickly becomes prohibitive.

This technique also hampers incremental system expansion. In order for a new type of processor to communicate in the system environment, it must be supported by a translator that translates from every existing format into the format of the node being added. Conversely, a translator that translates the new machine's format must be added to every node already in the system. To continue to support every possible communication path in the environment, every host requires some modification when the 'n+1'th host is introduced into the system. '2n' translators must be developed -- 'n' to reside at the new node to allow its neighbors communicate with it, and one new translator for each host already in the environment to allow them to receive communications from the added host.

A slight variant of this scheme is to have each transmitter perform all of the data formatting for its intended receiver.

However, this offers no relief to the need for a large number of translators. Further, it interferes with process-to-process message transfer and with network-wide message broadcasting by forcing the sender of a message to anticipate the internal data format requirements of its receiver.

2.2.3 Intermediate translator

A topologically different mechanism places a third party between two communicating processes solely to perform any needed data conversions. An experimental project on the ARPA network provides access to such an intermediate translator for specific applications. The project is the data reconfiguration service (DRS) <ARPA5>.

The DRS offers a solution to the problem of data format incompatibility between a particular applications program and its intended users. Through a predefined translation mapping, the DRS acts as an interpreter between the program and its user, permitting each to communicate in its own format.

There are two stages to the use of DRS. First, the applications programmer must describe a mapping between the data formats native to the processor hosting his program and the formats native to the processors representing his program's users. This requires specific

knowledge of both of the formats involved and the I/O data requirements of the application. Once such a mapping is fully defined, the programmer prepares a description of the conversion in a DRS supported language and catalogs that description by a unique name with the DRS.

When a user process wishes to communicate with such an applications program, it makes a connection with the DRS. It requests, by name, the use of the appropriate format translation description prepared by the applications programmer. The DRS then makes a connection to the desired program and from then on the program and its user communicate through the DRS -- each data transfer being reformatted according to the specified reconfiguration scheme.

The result is that both the applications program and its users only handle messages in their own respective formats.

> The user process behaves as if it were connected directly to the server process, and vice versa. The DRS appears transparent to both processes; its function is to reconfigure data that pass in each direction between them into formats amenable to each of their processing requirements. [<Anderson> p. 3.]

The DRS is effectively transparent at the application level and yet extensive data translation may be taking place.

As implemented on ARPANET, an intermediate third party for data reformatting has only limited use. Each application requiring the service must catalog the appropriate format descriptors at the reconfiguration service site. Each DRS mapping description is specific to an application, as well as to the formats of the associated processors. These descriptors provide a syntactic structure which can be applied to incoming bit strings to delimit the separate data items for reformatting. Such a description is essential to the reformatting process.

2.2.4 Standard format

The most often cited network communications facility uses a standard intermediate format to exchange data between potentially dissimilar hosts. This facility is TELNET <ARPA3>. Running on ARPANET, TELNET and its companion protocol for file transfer, FTP (file transfer protocol) <ARPA4>, support the transfer of characters from one network host to another. These protocols are discussed below.

The TELNET protocol is intended to carry characters between a process representing a human user at a data terminal or a process expecting to communicate with a terminal. TELNET forces standardization of character formats by interposing the notion of a network virtual terminal (NVT) between the two communicating

processes. Each host maintains a resident translator that performs the data reformatting between the internal representation of character data used by the host and that of the NVT. Any host-resident process that either emulates or services a remote terminal must communicate with the network through an instance of such a translator.

The standard character format used by TELNET is seven-bit USASCII. The data representation and conventions adopted for NVT, as described in the TELNET specifications, were

> intended to strike a balance between being overly restricted (not providing hosts a rich enough vocabulary for mapping into their local character sets), and being overly inclusive (penalizing users with modest terminals). [<ARPA2> p. 1.]

This is the original TELNET protocol. However, a scheme for providing extensions to the NVT through the "principle of negotiated options" has been added. The principle of negotiated options allows two communicating processes to discuss and agree to the use of each available extension to the standard NVT format. Since not all options will be supported at all sites, the ability to decline as well as request and accept the use of options is provided. By using the hand-shaking protocol, two processes can find the maximal set of options that is appropriate for their use.

The options available are all extensions and enhancements of the NVT.

They include changing the disposition of control characters (carriage return, line feed, form feed, tab), extending the character set and altering the message format. As described in the TELNET option specifications, these extensions are provided

> to permit sites to obtain more elegant solutions to the problems of communication between dissimilar devices than is possible within the framework provided by the Network Virtual Terminal. [<ARPA3> p. 1.]

It is through the mechanism of negotiation that use of these options is controlled by the communicating hosts.

The file transfer protocol was designed to provide a mechanism for file movement across ARPANET. As with TELNET, the communicating FTP processes agree through negotiation on the data format for the information transfer. Each host performs the translation necessary to convert its internal representation into and out of the intermediate format being used for the data exchange.

The need for data reformatting in the hosts is discussed in the original specifications for FTP. While crossing the network, a text file can be represented in the character set used by the TELNET NVT.

> Data is transferred from a storage device in the sending host to a storage device in the receiving host. Often it is necessary to perform certain transformations on the data because data storage representations in the two systems are different. For example, NVT-ASCII has

> different data storage representations in different systems. PDP-10's generally store NVT-ASCII as five 7-bit ASCII characters left justified in a 36-bit word. 360's store NVT-ASCII as 8-bit EBCDIC codes. MULTICS stores NVT-ASCII as four 9-bit characters in a 36-bit word. It may be desirable to convert characters into the standard NVT-ASCII when transmitting text between dissimilar systems. The sending and receiving sites would have to perform the necessary transformations between the standard representations and their external representations. [<ARPA4> p. 9.]

For text files, two standard character representations (NVT-ASCII and EBCDIC) are supported by FTP. Options for specifying format control information are also available. The human FTP user sets up the appropriate options and then initiates the file transfer.

Non-text files may also be moved by FTP. These are transferred as unexamined blocks of bytes of a specified length. However, even uninterpreted binary data can cause a problem in representation between host systems with different internal word lengths.

> It is not always clear how the sender should send data, and the receiver should store it. For example, when transmitting 32-bit bytes form a 32-bit word-length system to a 36-bit word-length system, it may be desirable, (for reasons of efficiency and usefulness) to store the 32-bit bytes right-justified in a 36-bit word in the latter system. In any case, the user should have the option of specifying data representation and transformation functions. It should be noted that FTP provides for very limited data type representations. Transformations desired beyond this limited capability should be performed by the user directly or via the use of the data reconfiguration service. [<ARPA4> pp. 9-10.]

An option is provided to allow the human user to specify the logical byte size of the data being sent. Through this mechanism the user can force the receiver to block and pad the data for storage.

> This...is intended for the transfer of structured data. For example, a user sending 36-bit floating point numbers to a host with a 32-bit word could send his data...with a logical byte size of 36. The receiving host would then be expected to store the logical bytes so that they could be easily manipulated; in this example putting the 36-bit logical bytes into 64-bit double words should suffice. [<ARPA4> p. 13]

It is only through this option that any information on the intended format or use of binary data can be passed along with the bits in non-text files. The problem of non-character data types is only considered in this way by FTP.

Another ARPANET project that has had to deal with the problems of interprocessor communication in a heterogeneous networking environment is the National Software Works (NSW). The NSW project recognizes the existence of large software systems that can serve as "tools" for further software development. Presently these software systems are scattered across the ARPANET.

> ...the National Software Works will provide users with access to software development tools on whichever machine the tools happen to be. User's files are moved to the tools over the NET, so the tools do not have to be reprogrammed for each new computer. People building tools may select the machine which is best suited for the tool they are building... [<Crocker> p. 5.]

Two phases of NSW are specified to address the data translation issues. These are interprocess communication and the file transfer system.

Interprocess communication under NSW was originally to be supported by a system called the Procedure Call Protocol (PCP) which was later augmented and renamed the Distributed Processing System (DPS). DPS was designed to support information transfer between dissimilar network hosts. The protocol was to include data communication through the use of standard intermediate representations. <Kimbleton3, WhiteJ> The scheme was to handle most fundamental data types.

> Until mid-August 1975, NSW planned to provide for communication between most of its building blocks through the Distributed Processing System. In August, DPS was formally dropped from the NSW plan in favor of a much less complicated scheme called MSG. [<Kimbleton3> p. 1-58.]

In January 1976, the preliminary specifications for MSG were released. The report <MSG> deals with the data reformatting issue in a way different from that of DPS.

> Message exchange...is expected to be the most common mode of communication among NSW processes. To send a message, a process addresses it by specifying the address of the process to receive the message and then executes an MSG "send" primitive which requests MSG to deliver the message. [<MSG> p. I-8.]

> A message is a string of bits created in the local memory of a sending process. MSG sends the message to a receiving process by duplicating the bit string in a specified portion of the receiving process's local memory. MSG itself imposes no further structure on messages, nor does it interpret the contents of messages. [<MSG> p. 2-6.]

Plans for data format standardization to support interprocess communication were dropped in MSG.

The file transfer system was designed to perform file format translations on data files as they were moved by NSW across ARPANET. This facility, too, has been reconsidered.

> The file transfer system is heavily dependent on DPS. Since DPS has been discontinued, the initial NSW implementation is going to use FTP to move files. Later refinements may provide for the non-FTP supported features of the file system. [<Kimbleton3> p. a-68.]

In summary, then, although the original NSW design included an examination of the data formatting issues, the current project effort has, at least for now, laid those issues aside.

TELNET and FTP offer two examples of the use of a standard data format to support information transfer between dissimilar processors on a computer network. Negotiated options are an extension of the mechanism that allows flexibility in the selection of the intermediate representations two processors will use in a given exchange.

Both TELNET and FTP force each transmission between processors to conform to a universally observed intermediate data format (IDF). To transfer data, each processor reformats its information into the IDF, and then sends the translation. Upon receipt of a message, a processor must perform a format translation to change the IDF representation into its own. This mechanism provides processor independent interprocess communication, since the broadcast message data format is the same for every system host. The number of required translators is reduced, as well. Two translators for each type of processor are required -- one for translation into and one for translation out of the IDF. Again letting 'n' be the number of dissimilar machines, the number of translators needed here is only '2n'. That is, each processor in the environment must support exactly two translators.

A universal format also facilitates incremental system expansion. Since each new processor need only be able to understand the relationship between the IDF and its own format, its addition to the system does not require knowledge of the current configuration. Further, the processors already in the system will only communicate with the new entry in a format they already know. No modifications or additions to them are required. The burden for system expansion is solely on the processor being added, which is precisely where it belongs.

2.3 Summary

The previous sections have presented design alternatives for a mechanism to support interprocessor communications in a heterogeneous computer network. The selection of a design for implementation should be a direct result of measuring the proposed mechanisms against desired design characteristics. The design characteristics being considered are the following:

-- process addressing
-- easy expansion
-- functional sophistication
-- application-level transparency
-- minimal host overhead
-- minimal message overhead
-- reliability

These are applied to the three proposed mechanisms described in the preceding section:

-- receiver translation
-- intermediate translator
-- standard intermediate format

Process Addressing - The internal data formats used by a running process depend on the format employed by the processor on which that process is active. Only by delaying the binding of a target data format to each message until that message is accepted at the node on which the desired process is active can process addressing be facilitated.

Both translation by receiver and the use of an intermediate format delay the final stage of data reformatting until a message reaches its destination. These mechanisms do not interfere with addressing messages to processes. An intermediate translator, on the other hand, requires the specification of the target format, and therefore the receiving processor has to be identified before a message can be translated and retransmitted to its intended destination. A scheme based on such a translator, then, cannot support process addressing, while the other two strategies can.

Easy Expansion - Expansion includes both the addition of processor types to a network and the extension of the formats used by processors already supported. An intermediate translator acts as a central agency for all data reformatting. Under such a scheme, any revisions required for system expansion are localized at that translator, and so the modifications can be made easily. Similarly, because an intermediate format demands that communications only appear in the network standard, expansion of a system based on that mechanism impacts only the translator at the site being added or changed.

However, as described, in a network with 'n' processor types, the modifications required for the receiver translation mechanism increase as

'n'. ('n-1' and '2n' translators are affected by format extensions and additions respectively.) Compared with the two other schemes, receiver translation carries a high cost of expansion.

Functional Sophistication - Each strategy is logically sufficient to support a full range of data reformatting facilities.

Application-level Transparency - In large part, the impact felt by end-users of any network mechanism depends on the host or network operating system to which user software must interface. For example, TELNET offers almost complete application-level transparency while DRS requires a considerable amount of information from the applications programmer. TELNET is fully supported by systems software and DRS is not. The difference lies in the way the description of passed data is handled. This issue is discussed further in Chapter III. Transparency at the application level is less a function of the scheme used to handle messages and more a function of the chosen scheme's implementation. In this regard the three translation schemes each offer the same opportunity for application-level transparency.

Minimal Host Overhead - Both TELNET and FTP interpret the standard data format being employed through a translator process that runs in the network hosts. Because the actual data translation for an

application independent intermediate data format would be performed as a preprocessing step for all messages, and because the number of translators required for each host is small, translation of the intermediate format could be withdrawn from the host and placed at a lower functional level in the node. Receiver translation could also be performed at a node level below the receiving host, but the large number of translators required could force an extra degree of node component sophistication. An intermediate translator eliminates node resident reformatting overhead, but requires one or more nodes (and hosts) dedicated to data reconfiguration.

Minimal Message Overhead - While an intermediate translator eliminates the need for any data translation at the communicating nodes, the installation and maintenance of and communications to special purpose reformatting nodes require additional overhead. Receiver translation requires exactly one data reformatting stage (that at the receiver), but demands the management of a large number of translators. An intermediate format requires exactly two data reformatting steps, but greatly reduces the number of translators that must be maintained over that for receiver translation, and so is the most preferable of the three strategies.

Reliability - The criteria for measuring reliability of the proposed mechanisms are the number and complexity of critical

components. Each of the three alternatives requires the integrity of the two nodes between which data is being exchanged. The use of an intermediate translator, also requires that a third entity to perform the data translation be functioning. This scheme has three critical modules while both receiver translation and the use of a standard intermediate format have only two.

The nodes in a receiver translation environment must maintain translators for every format used in the network. Therefore, the size and complexity of their network support software and hardware is greater than the package required in an intermediate format environment. Of the three, this rough measure of reliability favors the use of a standard format.

Figure 2-1 summarizes the evaluation of the three design stages with respect to the design considerations discussed. The figure includes examination of both the "ideal" implementation and, where applicable, an existing implementation of each strategy. The rating of the strategies for each consideration is qualitative. Where a strategy is decidedly more responsive to a design consideration than the alternatives, it is marked with a "plus" and the others are marked with "minuses" (e.g. reliability). Conversely, when one strategy is decidedly worse than the others in a particular category, it is marked with a "minus" and the others are marked "plus" (e.g. process addressing).

For some considerations, the specific implementation of a strategy is marked "minus" and the general use of that same strategy is marked with a "plus." These markings indicate that while the current implementation of the strategy is not responsive to that consideration, an extension/generalization of it would meet the designated design goal (e.g. sophistication).

	RECEIVER TRANSLATION	INTERMEDIATE TRANSLATOR		INTERMEDIATE FORMAT	
	GENERAL	DRS	GENERAL	TELNET	GENERAL
PROCESS ADDRESSING	+	-	-	+	+
EASY EXPANSION	-	+	+	+	+
SOPHISTICATION	+	+	+	-	+
APPLICATION-LEVEL TRANSPARENCY	+	-	+	+	+
MINIMAL HOST OVERHEAD	-	+	+	-	+
MINIMAL MESSAGE OVERHEAD	-	-	-	+	+
RELIABILITY	-	-	-	+	+

DESIGN CONSIDERATIONS
VS.
DESIGN STRATEGIES

FIGURE 2-1.

CHAPTER III

An Intermediate Data Format

The discussion in the preceding section motivates the use of an intermediate data format to facilitate interprocessor communication in a heterogeneous computer network. This section will address the mechanisms needed to support an IDF and the selection of standard data representations.

The conceptual view of interprocessor communications is depicted in Figure 3-1.

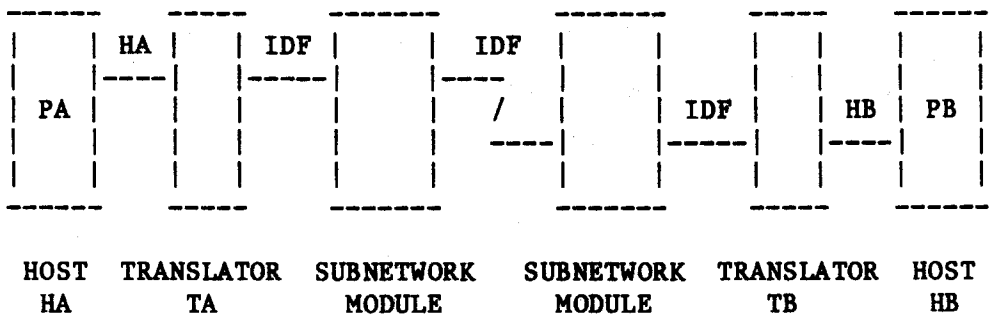


Figure 3-1.

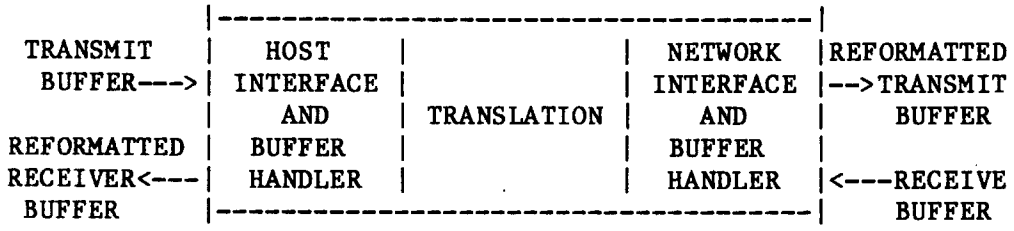
'PA' and 'PB' are processes residing in processors 'HA' and 'HB' respectively. The translators are responsible for any reformatting necessary between the data representation used by their corresponding host processors and the network standard. All data formatting is transparent to the subnetwork communication modules. Each data link between the modules in the figure has associated with it the data

format moved through that link.

There are six steps to the successful transfer of data from one process to another.

* the sending host passes data to its associated translator.
* the translator reformats the data to conform to the network standard representation.
* the translator passes the reformatted message to the subnetwork to be carried across the network.
* the subnetwork module at the receiving node passes the incoming message to its associated translator.
* the receiving translator reformats the message from the IDF to the representation appropriate for its host processor.
* the receiving translator passes the reformatted message to the receiving process.

To perform its function, the translator is passed a buffer in an input data format and builds a buffer containing the reformatted information. The translator can be broken down conceptually in the following way (Figure 3-2).



A TRANSLATOR
Figure 3-2.

The standardization of an intermediate network format in turn allows the standardization of the network interface section of the translator. Similarly, the portion of the translation section that is tuned to the IDF will be transportable among translators. The host interface and its half of the translation specifications will necessarily be host dependent.

3.1 Data description

There are two aspects of data description for a data item. The first is the definition of the data class or type of each item, such as character, integer, or instruction. In general, a string of bits carries no indication of the kind of data item it is intended to represent. This is because the overwhelming majority of currently available computer systems are based on the Von Neumann philosophy for storing digital information. These systems do not rely on any inherent distinction between the internal storage of different data types to manipulate information. Rather, the semantic meaning of a string of bits is derived solely from the context in which the bits are used.

> The Von Neumann form states that data and program are indistinguishable. This form assumes fixed size binary words or characters [bytes] which allow programs to be treated as data. These computational units

> are manipulable by a large, general purpose set of operations. Meaning is not inherently represented in the contents of these units; rather it is assigned to the contents of these units by the program manipulating them. [<Feustal2> p. 1.]

By example, an 8-bit string sent to a teletype may ring a bell, while that same string may be moved to an arithmetic unit to represent an integer value. Moving the same bits to the instruction register of a processor may cause yet another effect. It is the use of the bits that defines their data type, and not the bits themselves.

The second aspect of data description is the specification of the internal data format used to represent the value of an item of a particular data type. "Integer" is a data type. "Two's complement," on the other hand, is a data format used in many processors to represent integer values. Other formats used for integers include sign-magnitude, one's complement and decimal. Saying that a 16-bit data item is an integer specifies its type, but not the format used to encode the value it represents, and so is an incomplete description of the item. Describing the item completely requires the inclusion of both the data type and the internal format of the data item, i.e. the sixteen bit item is an integer represented in two's complement format.

Just as with the data type, determination of the data format used to encode the value of an item cannot be made by examining the data bits.

When two data items are added together in a processor that supports integer addition, the operation identifies them as integers and the architectural design of the "add" instruction identifies their format. PDP-11's support two's complement integer addition. S/370 supports addition both of two's complement and decimal format integers; in this machine, integers may be represented in either of these two formats.

The data reformatting function replaces the bits that reflect a value for a data type in one representation with the bits that denote the same value in a second representation. (For a discussion of how difficult this mapping can be, refer to Chapter IV.) To faithfully reconfigure an input data stream, a format translator must know how those bits were interpreted in the computing system environment from which they originated. That is, the translator must know whether to treat the incoming 8-bit bytes as EBCDIC characters or as quarters of 32-bit one's complement integers. As the data bits themselves carry no indication of their type or format, they alone cannot specify which data translation scheme must be applied to interpret them. A description of the type of data being moved and the internal representation scheme used for each type must be available to the translator to facilitate the proper reformatting.

The information needed to form such a description for each

message being sent across a network is known exclusively by the transmitting process. While the receiving process may be expecting a message containing particular data items, there is no assurance that an incoming message conforms to that expectation. The description of data items anticipated by the receiver is important to error checking and process synchronization. However, insuring a precise transfer of data items with their original values requires the sole use of a data description supplied by the sending process.

A semantic description of the data, then, as well as the data being transmitted must cross three communications links: from the sending process to its translator, from that translator to the translator for the receiving process, and from there to the receiving process itself. These links can be broken into two categories: the host-translator connection and the translator-translator connection. The distinction is important. Any protocol used to exchange information between a host and its associated translator must reflect the host's data formatting needs. Such a protocol is constrained by the specific characteristics of that host. While the type of information to be carried by a translator-to-translator protocol depends on the hosts included in the network, the form each type of data must take in the protocol is totally independent of all existing equipment. Strategies for transporting data descriptors must be evaluated in light of both of these connection categories. The rest of

this section discusses two alternative strategies for passing the necessary information.

3.1.1 Passing description by prearrangement

One strategy for passing the semantic description of a string of data bits is to rely on a prearrangement of the bit stream format. The mechanism is best explained by example. Two communicating modules may agree that the next stream of bits transmitted will form 16-bit two's complement integers. The sending module then transmits sixty-four data bits. The receiver breaks the incoming data stream into four 16-bit two's complement integers, assuming that to be the type and format of the data being sent. If the prearrangement mechanism has functioned correctly, the receiver has been successfully transmitted the context and substance of the four data items. In a more complex situation, two communicating modules may handle a bit stream that represents a specific mix of data items of different data types (e.g. the first item is a 16-bit two's complement integer, the second item is a 7-bit ASCII character, etc.).

Prearrangement of the type and format of the data items in a bit stream can be handled in two ways. First, the character of the stream can be fixed by system design and implementation. For example, one data terminal can only understand 7-bit ASCII characters, while another

can only understand 8-bit EBCDIC characters. A mismatch in design prevents the transmission of meaningful information from one to the other. Second, two communicating modules can select a format for the bit stream through format option negotiation. The semantic content of the passed bit stream is still described by prearrangement. However, format negotiation allows both the types of data items being moved, and the formats in which they are encoded to change dynamically as conditions warrant.

ARPANET implementation of the TELNET and FTP protocols are examples of the prearrangement technique. The original TELNET specifications required each inter-host TELNET transmission to format data in the 7-bit USASCII character set. For two communicating TELNET processes, a description of the trans-network data stream was specified by the system design. That is, by prearrangement, the data stream was a stream of USASCII characters.

The development of negotiated options offered a natural extension to the TELNET data description strategy. Rather than forcing two TELNET processes to communicate through a design determined format, negotiation allowed the selection of a data stream format to be delayed until just before each data transfer was to take place. This was a step towards increasing communications flexibility. Data description was still accomplished by prearrangement between two TELNET processes,

but each option (or combination of options) represented a different description that could be applied to the data stream being transmitted. Once two communicating processes agreed on a set of options, each could be certain that the receiving TELNET was applying the correct data description when interpreting the data bits being sent.

The TELNET strategy attacks the problem of carrying a data description across a network from one host to another. The data type of the items being passed is fixed as characters. This permanent requirement of a single data type represents data description prearrangement by system design. The data format representation of the characters being exchanged is also described through prearrangement. Instead of being fixed at design time, however, the format of the characters being transmitted can be respecified through option negotiation. The character formats currently available as TELNET negotiable options include standard USASCII, extended ASCII and binary. Under TELNET, renegotiation is allowed before every transmission and so the format agreed upon may change as often as every message.

Prearrangement with optional format negotiation has proven very successful for TELNET. However it seems untenable for a general communications scheme where it is necessary to communicate data of many different types. The rigidity imposed by total agreement through original

design is an unacceptable base for a general facility. The only flexible form of prearrangement appears to be option negotiation as implemented for TELNET. This, however, would be a costly mechanism for general interprocess communication. TELNET is already characterized by proliferation of options <ARPA>. The introduction of each new data type can be expected to be accompanied by its own set of options. Indeed, each data type would itself represent an option. Support for messages composed of items of mixed data types would require a further extension of the negotiation scheme to handle composite messages.

While prearrangement of data type and format is unwieldy for a system requiring generality, it may be adaptable to some instances of host-translator connections. An example of such a case is a network connection to an unintelligent peripheral device such as a data terminal. A terminal has neither the means nor the need to win format flexibility through option negotiation. It must send and receive bit streams that conform to a fixed data description. In this example, the translator must match the data type (character) and data format (ASCII, EBCDIC, etc.) of the terminal at all times. Such a binding is a natural application for using prearranged data description to pass the semantic content of data bits.

3.1.2 Passing description by data tagging

The second alternative for passing the description of a bit stream is to mark each data item in a message as to its type and format. In such a scheme, each datum in a transmitted message has a standard data descriptor associated with it. Embedding these descriptors in the actual data stream so that they preceed items which they describe allows the receiver to delimit and determine the type of each item separately as it is delivered. As long as it is paired with its data descriptor, each piece of data is a totally self-describing item.

The support of self-describing data insures against the separation of a message and its context.

> Precision in data transfer permits semantics and structural information which exists in the sender's instance of a datum to be reproduced in the receiver's image of the datum, even though it may be represented in the systems involved in entirely different fashions...Data of a given type should be recognizable as such [by a receiver] without the need for context...A particular service can achieve data precision by meticulous specification of the protocols by which data is transferred. This need is widespread enough, however, that it is appropriate to consider inclusion of a facility to provide data precision within the mechanism itself. [<Haverty> pp. 8-9.]

The technique of attaching descriptors to data items to make them self-describing has been called "tagging" <Feustal1, Feustal2, Illife>.

Tagging does not totally eliminate the need for data description by prearrangement; it only moves the agreement to a different level of data handling. Instead of requiring prearrangement of the content of each data stream passing between communicating modules, a scheme based on data tagging allows any message to contain any legal combination of data descriptors (tags) and data bits. The tags in the message describe the data being transmitted and no pre-message agreement of the items in the message is necessary. Required, however, is agreement of the meaning and form of the data tags. Every communicating module must understand and conform to the use of tags to describe data bits. Without common agreement at this level, messages built out of self-describing data would be unintelligible.

To send four 16-bit two's complement integers, a transmitting module builds a message out of the sixty-four data bits and the "16-bit two's complement" tag. For 8-bit tags, this would mean a message length of 96 bits. Then the module transmits the message. The receiver detects the tag that, through prearrangement, designates a 16-bit two's complement integer and uses the sixteen bits that follow it to

form the number. Likewise, the other three data items are delimited, and the transmission of the four integers is successful.

Self-description more readily facilitates general inter-processor communication by allowing data streams of mixed data types to appear in messages. The overhead associated with supporting the negotiation process is replaced with the overhead of encoding, decoding and moving the extra bits required for the descriptors.

Self-describing data involves a corruption of the data stream with the item descriptors. While it provides full flexibility of message format and content and is acceptable to intelligent communicating modules, the scheme is not at all appropriate for the data needs of a simple peripheral acting as a host. Although special equipment could be built, most currently available unintelligent devices cannot tolerate communications strategies that require modification of the data handling protocol. This includes interpreting or even simply discarding descriptors embedded in the data. For such devices, taging is an unacceptable proposition.

Because data tagging allows more flexibility than option negotiation, the use of data descriptors to build self-describing data is preferable for translator-to-translator connections. While this scheme offers the same advantage when applied to host-translator

connections, some potential hosts are unable to perform the message processing necessary to support it. The description passing strategies used in each host-translator link, therefore, must be selected to allow the translator to support its associated host in the most reasonable fashion.

3.2 The standard format

An intermediate data format (IDF) is intended to provide two processors using dissimilar internal data representations with a common ground for information exchange. The basis for their communications is the standard intermediate data format interpretable of data stream descriptors. This section discusses the selection of the data formats to act as the intermediate representation of each data type to be moved among the translators.

In the ideal sense, the choice of intermediate formats can be made arbitrarily. Messages with data items represented in a standard format pass only between network translators designed specifically to handle whatever format is picked. Only factors of economy constrain that selection.

3.2.1 ASCII character representation

Two alternative formatting schemes have been proposed. The first scheme involves the transmission of each data item in an ASCII character representation of its value <Kimbleton3, Teager>. Every datum has a human readable form that can be built as a character string. The scheme proposes this format as the intermediate representation of the item. To transmit a small floating point number, for example, a sending translator would broadcast the ASCII characters representing the sign and integer part of the data item, an ASCII period for the decimal point, and finally the ASCII characters representing the fractional part of the number being passed.

The most convincing argument for the use of an intermediate data format based on the ASCII character set is the already wide-spread use of this format for the internal representation of human readable information. By performing information input/output functions with ASCII characters to terminals and line printers, processors are already required to support translation between ASCII character strings and the machine-dependent internal representations for other data types. Choosing a standard format for which many processors already support software/firmware translation can greatly simplify the initial development and the continual maintenance of the IDF translator module associated with each network host.

A strategy using the character-based intermediate format mechanism for ARPANET has been suggested to facilitate the transfer of records to and from data files. This strategy assumes that the function to be supported is the transfer of data file records from the secondary storage of one system to another, and that such data files are associated with a data description of the records (type and format of the items in a record).

> Predicated upon the existence of a suitable logical data description of the file being accessed, the following four step approach to data translation in a networking environment seems reasonable...The four steps are:
>
> - using the access method originally used to write the file to retrieve the desired record at the source site,
>
> - using the logical data description of the record together with knowledge of the I/O routine originally [used to] write the file...to transform the record from the form in which it is internally stored to a character normal form analogous to that in which the record would be listed by a line printer,
>
> - using variants of existing ARPANET protocols to transmit the record from source to destination,
>
> - using at the destination, the record and its logical data description to reconvert from the character normal form to that used for internal storage of information (corresponds to the usual transformations performed in supporting data entry to go from the manner in which data is entered to that in which it is stored). [<Kimbleton3> pp. 2-7 to 2-8.]

Thus, this is an example of a situation where all data items are translated into the same format and the actual description of data types is passed by prearrangement. Similarly, it would be possible to use the tagging strategy, where the tag would describe the actual type of an item being transmitted as a string of characters.

3.2.2 Formats based on data types

The alternative to a character based intermediate format is the definition of a set of data representations with the formats best suited to each type of data item. An integer might be represented in a 32-bit two's complement format, as an example. Characters would probably use the standard ASCII character set.

An advantage to establishing a different intermediate format for every data type is that in many cases, the data translations can be relatively straightforward. Because most processors operate on two's complement numbers, for example, choosing a two's complement intermediate format for integers will necessitate only the simplest of data reformatting for many machines.

The selection of one of these two alternatives for the design of an intermediate format only impacts the implementation of the network

translators and the use of the communications links that connect them. The data passing strategy enforces transparency of the intermediate format at all other system levels. The two schemes must, therefore, be evaluated with respect to message processing overhead incurred at the translator module and the data transmission overhead required to carry information across a network.

Processing overhead at the translator is a direct function of the complexity of the required data translations. In general, converting from one machine readable form to another is simpler than manipulating character string representations. Translating a 24-bit one's complement integer into 32-bit two's complement representation is certainly easier than translating that same integer into as many as 64 bits (eight characters) to form the ASCII string. This overhead has prompted the support of a high-level language option to allow human user controlled specification of the internal format to be used to store application program data. The user is recommended to store data items predominantly used in calculations in binary or decimal format. Items that are required extensively in human readable I/O operations may be stored in 'picture' format to facilitate their conversion to character strings [<PLI> p. 222.].

There is no question but that the human readable form of most data requires a longer bit representation than common machine readable formats

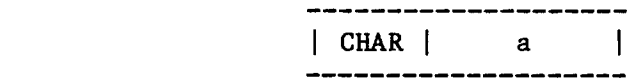
for the same data. The increase in message length through the use of bit-wise inefficient data formats is included in communications connection overhead. The use of inefficient formats increases network resource contention and decreases message thruput affecting both cost and performance.

While the existence of support for ASCII format translation in many processors encourages use of an ASCII-based scheme, simpler translations and the overhead issues weigh heavily in favor of data-type dependent intermediate formats. Were a scheme required solely to transmit file records from processor to processor, the character based formatting described above would be attractive. Instead of building a data descriptor for each record, the information passing utility could use the file descriptor already associated with each file record. The items in the data record would then have to be forced to conform to the format presented in the record descriptor, i.e. translated into their ASCII-character equivalents. However, inter-processor communications require more general message content. For these, the character-based format is more costly than a scheme based on selecting an applicable format for each type of data to be transmitted.

3.3 A possible IDF

This chapter has shown that the most reasonable choice for a standard intermediate data format is one that transfers data items that are tagged with their type and format. Further, for reasons of efficiency, the format used to transmit the value of each data item should be natural to that item's data type. Every item in an IDF of this form has two parts, a data type tag and a data value. For a simple data item such as an integer or a character, the "value" portion of the item can merely be the actual number or character. A distinction must be made, however, between primitive data types and composite data types.

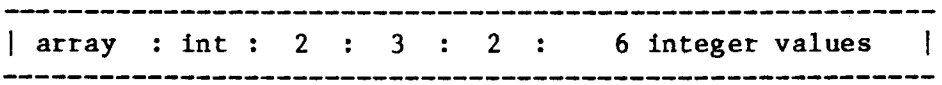
Figure 3-3 depicts an IDF representation of the letter "a". The left half of the item contains the type tag for character. There is exactly one such tag for every data type, and since their use is limited to the IDF translators the assignment of tag values to data types is totally arbitrary. The tag value is symbolized here by "CHAR". The right half of the item contains the character being represented in the intermediate format.



Primitive Data Type Format
Figure 3-3.

Besides integers and characters, the list of primitive data types includes items such as double precision integers, floating point numbers, and boolean values.

The tagging mechanism also supports the transfer of data structures. Although the set of compound data types that should be supported is not clearly defined, composite data types such as arrays or general data structures could be constructed from these primitive types. Figure 3-4 is an example of an IDF representation for an array. The tag value for an array is specified, and the "int" following it indicates that this is an array of 16-bit integers in two's complement format. The next three fields describe the number of dimensions and then the range of those dimensions. All of this is followed by the data values themselves.



3 x 2 array of integers
Figure 3-4.

The selection of the optimal intermediate format for each data item is a topic that requires further investigation. While there is some question as to the number of bits required in each item, the overwhelming use of two's complement arithmetic in commercial processors indicates that the intermediate formats for integers should be based on two's complement. The same is true for ASCII and the intermediate representation of characters.

CHAPTER IV

Problems of Data Translation

A data format is a scheme for representing the value of data items of a particular type in a convenient way. For example, both the familiar Arabic digits and Roman numerals are data formats for representing the values of the counting numbers. Translation is the process by which information in an input data format is mapped into its corresponding representation in an output or target data format. A simple example of this process is the conversion of 'XXV' to '25'.

Data translation can take place at two different levels. First is the mapping of a data item of a particular type and format into a different format for the same type. This is the case of translation between one's and two's complement integers or between Arabic and Roman numeral counting numbers. The second level of translation is mapping a data item in one representation into a representation for an item of a different data type. This process includes converting integers into floating point values, or converting numeric characters into numbers.

Both levels of data translation are required to support general interprocess communications in a heterogeneous environment. As a data item is moved from one processing environment to another, its representation must change to meet the data representation constraints of

its new host. Whether this involves mapping into a different format for the same data type, or translating into a representation for a completely different type, will depend on the data types and formats supported in the new environment.

As already stated in the section on data description in Chapter III, the data type and format of an item represented by a string of bits is established when those bits are used. Whether they appear as an operand to a one's complement addition or as an address loaded into the program counter, these bits can just as well be sent to a line printer as a character in the next instant. The conclusion to be drawn from this is that the software running on a processor gives semantic meaning to the data items. It is, therefore, the software, not the hardware, that determines the data representations supported in a processing environment.

The point is easily argued. A typical minicomputer has 16-bit registers and an assembly language instruction that performs two's complement addition on 16-bit operands. But with the proper software, this processor can be made to perform 18-bit one's complement or even 64-bit decimal floating point addition. True, the hardware facilitates the manipulation of 16-bit two's complement integer data items, but the hardware does not necessarily restrict the type and form of data that can be interpreted by a processing system it hosts.

All of the problems associated with data translation are the result of information moving between processing environments that support different data types and formats. These environments are shaped by software, and to say that a processor does not support a particular data type means only that there is no intelligence in place to handle data items of that form.

Performing precise data transfer requires the accurate and complete movement of all of the information in the items being transferred. When items must be exchanged by processors that support different sets of data types and formats, translation problems can occur. The rest of this chapter examines three of these problems, in particular, precision, format incompatibility, and data type incompatibility.

4.1 Precision

The precision of a data type format is a measure of the range of values that can be represented in that format. For binary data, the amount of information in an item is the number of significant bits the current value of that item contains. Precision problems exist when the input and target formats for a translation are formats for the same data type, but can represent different ranges of values of that data

type. If data items represented in a high precision format are translated into a format of lower precision, a loss of information can occur.

For computers to exchange data of a particular data type, translations may have to be performed between data formats based on different word lengths. Some values that can be represented in 36-bit words, for example, have no representation in 16-bit words. Items in the larger format that require more than sixteen bits of precision cannot be mapped into the sixteen bit format in a way that retains their value.

The problem of precision loss can manifest itself at any point in the communications system at which format translation is carried out. In the case of systems that are based on an intermediate standard format, these are the translations into and out of the IDF.

Precision problems can be avoided as items are translated into the standard format by selecting standard representations that are of as high a precision as the representations used in any of the host processors in the network. Of particular concern are numeric data. There is no single format of sufficient precision to represent the entire range of integers or real numbers, however, it is necessary for an intermediate format to be able to represent any number that may pass between

communicating processors. By considering the internal formats used by the hosts on a network, an IDF of sufficiently high precision can be chosen that contains representations for every value of each data type that may appear in the network.

The formats of the receiving hosts, however, cannot be altered to provide a representation for every required value. Their data format precision is fixed by hardware and the software that implements extensions of hardware defined formats. Data translators at this stage must address the possibility of receiving data that cannot be adequately represented in their associated hosts.

The responsibility of the receiving translator is to distinguish between data items that can and cannot be represented completely in the available target format. The translator can report incidence of problems in precision to the receiving host process through a predefined protocol. By convention, some attempt at representing the offending data item can be made. Once notified, it is the responsibility of the receiving process to respond to the problem through discussion with the process transmitting the datum.

A scenario typifying the problem consists of a sending host process transmitting 36-bit integers to a processor using 16-bit words. Since the standard format must be able to transfer all

possible integer data values, the intermediate format for integers may require 64 bits. Because of the IDF, neither of the two communicating processors has knowledge of the internal format used by the other. They each see only their own format.

Integers passed to the receiving translator are examined for their precision. Items whose value can be represented in the 16-bit integer format are translated and tagged appropriately in anticipation of movement into the host. If the data value can be represented as a double precision (32-bit) integer, the translation is performed and the item in the target format is marked with its description. When even double precision is not sufficient to represent the incoming data value (i.e. it carries more than 32 bits of precision) some information must be discarded. An algorithm can be applied to the incoming data to select 32 bits to form a double precision item. This item is then tagged with its description and a mark to indicate that the translation caused a loss of information.

The philosophy underlying the scheme to handle lost precision must be one of "make do." Firmware or software mechanisms to support multiple precision can offer an extended target format for the receiving translator. In general, though, processes communicating across a network need to anticipate the possibilities of sending or receiving messages with values that suffer precision problems and cannot be

understood. Until all processors support each data type with equal precision, the problem is unavoidable.

4.2 Format incompatibility

Another problem inherent to data translation is format incompatibility. As is the case with precision, format incompatibility problems occur when data items of a particular type and in a particular format must be translated into a different format for the same type. However, this incompatibility is strictly a function of formatting scheme, and is not related to the number of bits allowed for a value's representation.

The fractional values it is possible to represent "exactly" with a specific number of bits differ from one formatting scheme to another. The problem has been described in a warning that accompanies the discussion of the automatic format conversions that occur in PL/I.

> The rules for arithmetic conversion specify the way in which a value is transformed from one arithmetic representation to another. It can be that, as a result of the transformation, the value will change. For example, the number .2, which can be exactly represented as a decimal fixed point number, cannot be exactly represented in binary. [<PLI> p. 270.]

In discussing the problems that surround the use of fractional data, Kernighan and Plauger state:

> The reason is simple: "0.1" is not an exact fraction in a binary machine (in much the same way that 1/3 is not an exact fraction in a decimal world); its nearest representation in most machines happens to be slightly less than 0.1. [<Kernighan> p. 91.]

Another example is the format incompatibility that exists between different representations for binary integers. The "minus zero" value in the sign-magnitude and one's complement formats cannot be represented in two's complement. This is not a precision problem since increasing the number of bits allowed for the target (two's complement) format will not make a difference.

To the translator connected to a heterogeneous network, the rounding and truncation associated with reformatting fractional numeric data is unavoidable. The PL/I approach to format incompatibility is to print a warning in the language reference manual. The average application program (programmer) tolerates inaccuracies in the least significant digits of calculated results, and only when exactness is required is the issue raised. The problem cannot be circumvented, and perhaps the most reasonable alternative for a translator design is to issue a disclaimer and let the user beware.

4.3 Data type incompatibilities

The third problem of data translation is the resolution of data type incompatibilities. To perform its function, the translator associated with each network host must first determine the type of an incoming data item. That information is then used to map the input format representation into an output format representation of the same data type and value. Precision and format incompatibility problems exist when the input for a particular data type supports a higher precision or can represent different values that the output format for that same data type. A data type incompatibility, on the other hand, is the complete absence of an output format into which items of a data type being received can be mapped. While an example of a precision problem is mapping 36-bit integers into 16-bit integers, an incompatibility is trying to move 32-bit floating point numbers to a teletype. The information carried by the floating point item is lost to the teletype because it has no way to represent any data type but characters. In the context of a computer network supporting a diverse set of processing environments, data type incompatibilities can arise between hosts of different capabilities. This is particularly true in the case of special devices or unintelligent network hosts.

As with problems in precision, the translator must address data type incompatibilities on a case by case basis. In some

circumstances, the value of an item of a data type not supported by hardware may be marginally representable in the format for a second data type that is supported. Floating point numbers can sometimes be reasonably represented in an integer format. Boolean values can be represented as integer zeros and ones. In other situations, no intuitive alternative data type may exist. A hardware unit to perform fast Fourier transforms on arrays of floating point numbers may not be able to meaningfully handle any non-numerical data types.

Unlike precision problems, most instances of data type incompatibility will need to be handled by the translator. This is certainly true in the case of messages being sent to unintelligent hosts or special purpose devices with limited processing power.

When no alternative format is acceptable, the translator must initiate the appropriate fault recovery procedures. Again, especially in the case of unintelligent devices, the receiving host must be shielded from extraordinary conditions. Whether the necessary action is to dispatch a standard message to the transmitting process to inform it of the problem, or to just ignore the offending message, it is a task best left to the translator.

4.4 Summary

There are no elegant solutions to the problems of data format precision and format incompatibility. A translation scheme can only patch the environment. The inability to overcome some conditions of heterogeneity remains. These problems can only be handled by intelligently written applications software running on the appropriate network hosts.

An applications program requiring unusual data precision or using peculiar data types can announce its requirements to its correspondents. Difficulty in resolving representation problems may force communicating processes to resort to negotiation ala TELNET or to complete abandonment of the processing task at hand. Unless a receiving process can understand the type of data being sent to it, a standard intermediate format and format translators are of no use. When the type of the data is recognized, it is still necessary to consider the possible problems of precision and format incompatibility.

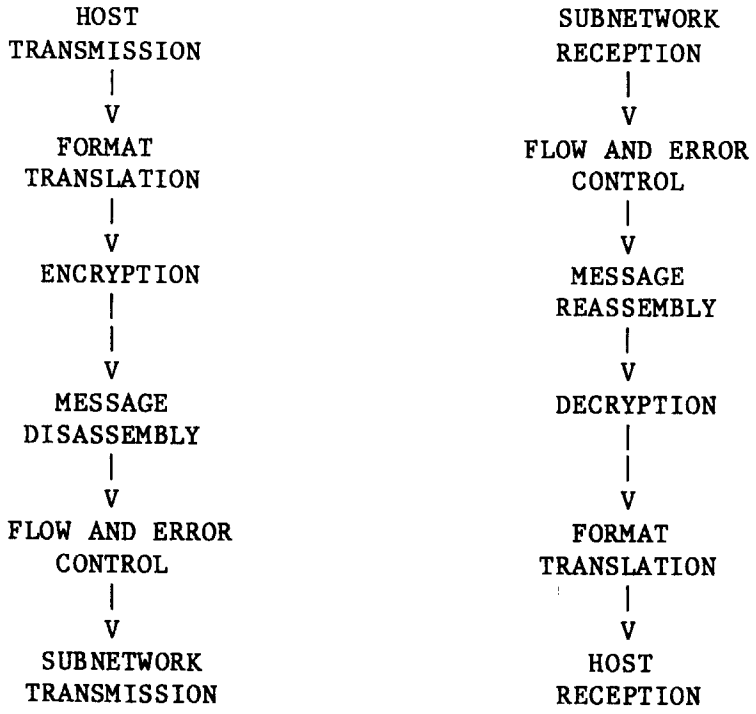
CHAPTER V

Translator Implementation Considerations

Examination of possible strategies for implementation of data translators raises issues that are transparent to the translation scheme itself. These issues include the interaction between the translator and other mechanisms that perform message processing, and the additional data describing functions required of the host.

5.1 Format translation and other message handling functions

Even without data format translation, successful transmission of information between machines on a network requires several message handling functions. The order in which these functions must be applied to each message is fixed, and this order is depicted in Figure 5-1. The following subsections briefly describe each function and its relationship to the implementation of format translation.



A. Message transmission B. Message reception

Figure 5-1.

5.1.1 Message packetizing

Since communications between processors may take the form of very large messages (up to millions of bits for file transfer), a great deal of consideration has been given to the advantages of sending long messages one portion at a time. To distinguish between messages and pieces of messages, the term "packet" is used to describe message fragments. While messages are the unit of communication between processes, these packets are the unit of data that moves through the communication subnetwork.

The motivating concern over packet size is subnetwork performance. There are tradeoffs to be examined.

> Large packets have a lower probability of successful transmission over an error-prone telephone line (and this drives packet size down), while overhead considerations (longer packets have lower percentage overhead) drive packet size up. [<Crowther> p. 170.]

Cases can be made for the optimum packet size in a particular network environment; the governing factors have different manifestations for local networks <Farber1, Fraser1, Metcalfe2> than for geographically distributed networks <Metcalfe1, Pouzin2>. (An especially good examination of the issues for packet switching networks (i.e. ARPANET) can be found in <Metcalfe1>.) It seems, however, that regardless of the network, the fragmentation of at least some messages into packets is necessary.

Figure 5-1 indicates that only fully assembled messages can undergo format translation. In order to make message fragmentation the simple-minded partitioning of messages into several packets of a fixed length, the semantic content of the message (Chapter III) must remain transparent to the disassembly process. The desire for this transparency imposes an ordering on the two functions of message packetizing and format translation.

Trying to perform data translation on message fragments to be transmitted causes two problems. First, data reformatting will likely, but unpredictably, change the size of the data items in a message, partially defeating the packetizing mechanism. Secondly, and also likely, the message may be split in the middle of a data item, severely complicating any mechanism attempting to reformat that item. This potential fragmentation of data items also discourages the translation of received information in any form but fully reassembled messages.

5.1.2 Flow and error control

Simply, flow control is the process of insuring that the receiving host does not lose information from its sender at any time due to too high a rate of data transfer. Mismatches of sender-receiver pairs can result in packets arriving at their destination faster than they can be ingested, overwhelming the receiver, and forcing packets to be discarded.

Error control supercedes the normal error detection for packets between subnetwork nodes, such as parity and checksum verification. These simple types of errors are common when transmitting over potentially noisy communication lines, and must be handled totally at the subnetwork level. Rather, error control deals with the problems of lost or duplicated

packets. Packets may appear or dissappear either as a result of the simple checksum or parity errors (discarded by the subnetwork), or flow control deficiencies or hardware failure.

Both error and flow control are functions concerned with the movement of data packets over the communications subnetwork. These functions handle the blocks of data that move between subnetwork nodes, and so must be performed at a level between the actual transfer of message bits across the subnetwork and message packetizing.

5.1.3 Encryption

As the use of computers for the storage and manipulation of classified (military), proprietary (industrial) and confidential (personal) information increases, the need for mechanisms for secure data handling also increases. Particularly vulnerable to breaches of information security are the communications paths between the nodes of a computer network. Often these paths may be inter-laboratory, and so their physical security cannot be assured. Unauthorized access to information being carried in communications links can be thwarted by data encryption. The intent of this section is to relate encryption to other network message handling functions, in particular format translation. Encryption techniques and associated protocols are not discussed, and for these the reader is referred to <Kent>.

Message encryption can be broken up into two separate categories. First is the encoding of the data field (Figure 1-2) of the message. This field contains the information the sending process is trying to transmit to the receiving process. Presumably, this would be the primary target for unauthorized access. The other category is the encryption of the packet control information that must accompany the data as it traverses the network. Precisely what information must be passed with the text of a message and how it may be encrypted for a give network is partly dependent on the implementation of of the communications subnetwork. The protection of that information is not considered here. This section is only concerned with the encryption of the actual text of the message.

Protection modules that perform data encryption/decryption must be at the level following format translation for information transmission and conversely the level before format translation for information reception.

> With respect to functionality, protection modules are constrained to be below the portion of the communication system that engages in syntactic processing of message contents...With respect to output from the host, encryption can be performed only after such transformations as device-specific code conversion, white-space optimization, and formatting. With respect to input to the host, messages must be deciphered before such transformations as canonicalization, break character

> detection, erase-kill processing, translation, escape sequence processing, character echoing, and high priority message recognition can be performed. [<Kent> p. 65.]

A format translator must receive unencoded message text in order to perform the semantic analysis necessary for data reformatting. However, a mechanism for message packetizing must only be able to count and partition the bits in the data field of a message. It need not have access to that field in its unencrypted form. Similarly, flow and error control functions require access only to the unencoded header and trailer fields of packets. Figure 5-1 depicts the functional level of message handling appropriate for the encryption process.

5.2 Implementation of message processing functions

There are two schools of thought on the implementation of message handling functions. Both philosophies view each network site as having a host processor connected to a subnetwork node. Simply stated, one side argues that all message processing should be transparent to the communications subnetwork. The other argues that all message processing should be transparent to the hosts. As a result, networks have appeared that reflect both philosophies <Metcalfe1, Metcalfe2, Pouzin2>.

The principle advantage to performing all message processing operations in the host is the simplification of the subnetwork node.

Removing any host-specific functions from the subnetwork level permits each subnetwork node to be exactly like every other. This duplication facilitates subnetwork maintenance and enhancement. Perhaps more importantly however, limiting the subnetwork function to delivering bits accurately facilitates the interconnection of the subnetworks of different networks. The incorporation of network specific protocols into the subnetwork nodes necessarily complicates the mechanism that moves messages between subnetworks. This thought is expressed strongly in a paper about CIGALE, the subnetwork for the CYCLADES packet switching network.

> It is clear that the CIGALE transparency is its major trump to provide a communication service between existing systems. Any additional well-wishing function tied with the external world is likely to be incompatible and detrimental to a good service. in particular, communications networks studded with all sorts of bells and chimes will end up as one of a kind networks, unable to communicate, unless an ad hoc kludge be interposed so that they at last exchange packets. [<Pouzin2> p. 159.]

Another argument for performing all message handling functions in the host is based on data security. Performing encryption and decryption in the network hosts is necessary to insure that no unencoded data need ever leave the host. The importance of this consideration, however, is minimized if the hardware performing message processing is considered to be merely an extension of the host processor, as would be an I/O channel, for example. The host and the message processor can be

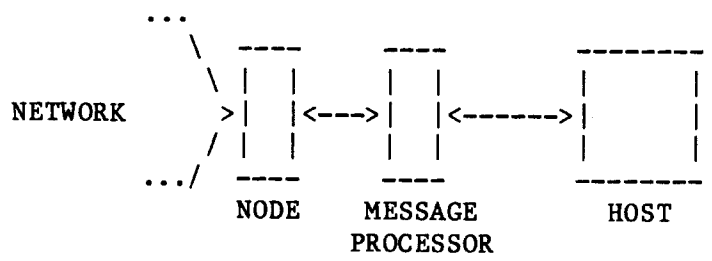
physically close, and so they and the communications link between them can enjoy the same level of physical security.

There are also two arguments for performing the message handling functions at the subnetwork level. First is that regardless of the host involved, the message processing operations are essentially the same for every site in a network. A programmable subnetwork node can be customized for each host, while the bulk of the software can be written one time in a single language compiled for the nodes. This eliminates the need to redevelop message handling routines at every host. It also greatly simplifies the addition of a new host to the network.

The second argument is that keeping network related functions out of the host minimizes the impact inclusion in a network may have on a host software system. This facilitates the use of the network by users at installations with either limited system expertise, or limited processing power or flexibility.

Although the arguments seem irreconcilable, a compromise that seems to be a natural conclusion to the controversy has been suggested <Manning>. Discussion to date has centered on the partitioning of functions between a subnetwork node and a host-resident network control program (NCP). Providing a separate hardware level expressly for message processing is responsive to both sides of the argument. Figure

5-2 depicts such a configuration. The additional hardware would host a format translator, message packetizer, flow and error control modules and, when required, a protection module for encryption/decryption.



DEDICATED MESSAGE PROCESSOR

FIGURE 5-2.

The introduction of a "message processor" does not render the network functions totally transparent to the processing systems of the network hosts. Each host must still exchange data description information with its associated format translator (Chapter III). This impact to the host system cannot be avoided. However, with all other message processing being performed at a separate level, the NCP required can be relatively small. An area requiring further investigation is how data description can best be supported by an existing host system. (See section 5.3.)

The subnetwork node, on the other hand, can be functionally limited to supporting bit-passing protocols. This requires no host-specific information and so each subnetwork node can be interchangeable with every other. Such a node has already been suggested for some local networks <Mockapetris>.

While not all of the hardware and software at the additional level can be standardized for all network hosts, it may be possible to limit host-dependent information to the data translator. It is the differences in the data rerepresentations of the hosts that force message handling functions to be specific to the host with which they are associated. These are the same differences that necessitate a network-wide data translation scheme in the first place. The discussions on data description (Chapter III) and on data format precision and incompatibility (Chapter IV) underscore this need for translator customization.

Simplifying the subnetwork node by separating the message handling functions from it satisfies an argument for moving all operations into the host. Moving all but a minimum of operations out of the host and into a separate module that can be mostly standardized, is responsive to the arguments for moving these functions into the subnetwork node.

5.3 Supporting data description in the host

If, as suggested in Section 5.2, the majority of the message handling functions are performed in a separate message processor, then a host need only be able to move messages back and forth between the network-using processes it supports and the message processor. In particular, a host-resident network control program (NCP) must interface applications programs with the format translator.

Section 3.1 discussed methods of passing data description between communicating modules. One of the two mechanisms described involved a prearrangement of the semantic meaning of a bit stream by either negotiation or by system design. The other was a data item tagging scheme, and this was suggested as the more flexible of the two in a general purpose processing environment.

An important consideration is the effect the implementation of a data tagging scheme would have on the host operating system and user community. Minimizing the impact on a community joining a network makes the network a more attractive resource. In line with this philosophy, a mechanism has been suggested to facilitate the implementation of data description through tagging between host and format translator.

Interprocess communication can be considered a case of data input and output <Hoare>. The destination process receives the output from a sending process in response to an input request. The implementation of such a mechanism could be modelled after the I/O routine packages currently available in single process environments for languages such as FORTRAN IV, ALGOL and PL/I. These languages perform I/O on a user specified transmission list and with a user specified format (i.e. the FORMAT statement in FORTRAN, and the EDIT option in PL/I). The user could request I/O as he would for a locally resident process with which he wanted to communicate. When some higher authority determines that the referenced process is active on some other network host, the network I/O control program could combine the user's description of the data with the data itself to form a tag-based data stream. This scheme has the advantages of being familiar to most application programmers and a straightforward "add-on" to existing systems.

CHAPTER VI

Conclusions

The intent of this report was to examine possible mechanisms for moving data between dissimilar processors and to identify the mechanism most responsive to the requirements of a heterogeneous computer network. Three data format translation schemes were reviewed, and from these, the use of an intermediate data format was selected. Alternatives for the intermediate formats were also discussed and one was proposed for general use.

Some problems are inherent to data translation and are independent of the translation scheme. Several of these were discussed including the passing of data description, data type and format incompatibilities and loss of data precision.

Although implementation considerations were presented, the results of a sample implementation were not. The unavailability of a suitable network testbed made such an implementation infeasible.

The effort involved in preparing this report will be justified by an implementation of the mechanism described. We hope this document will serve as the foundation for such implementations on both local and

geographically distributed networks. Several topics that require further investigation are discussed in the following section.

6.1 Areas for future study

Many of the problems surrounding the use of an intermediate data format based network communication scheme have not been solved. The last three chapters have pointed out areas that require further investigation. These include mechanisms for data description between hosts and network translators, the data format representation best suited for use in an IDF, and strategies for data format error recovery. The next subsections suggest other areas that still must be studied.

6.1.1 The contextual meaning of data

In certain processing environments, the appearance of particular data values can sometimes cause a special effect. That effect, while triggered by the data item, is a predetermined reaction of the environment to that value. In the case of such items, passing the type and form of the representation with the data bits to a second environment is not totally sufficient. The meaning of the item in the context of the transmitted environment must also be sent.

Chapter III discussed the problem of "passing the semantic description of a string of data bits." Mechanisms for describing data for this purpose were presented in Section 3.1. The contextual meaning of data is independent from issues of data type and format. It is separate, too, from the formatting problems presented in Chapter IV. The ability to move data values across the barrier of heterogeneity only uncovers the problem of passing the effect those values have on a processing environment.

The difficulty encountered when moving lines of text from one computer system to another is an example of a problem conveying contextual meaning. Two systems invariably disagree on the interpretation of format control characters. A specific instance is the use of the horizontal tab character. One system may take its appearance to mean pad the current line with spaces until the line character count is the next multiple of eight. Another system may space to the next multiple of five, while still a third may attribute no special meaning to it at all. Characters causing similar problems include form feed, vertical tab and carriage return.

The implementation of TELNET recognizes the problems of passing contextual meaning. The effect a special character has in the receiving environment can be established by prearrangement, and a TELNET negotiable option for the disposition of many such text formatting characters is described in <ARPA>.

It is important to understand that passing the tab character in itself is a different problem. The concern here is accurately passing the effect the tab had in the system in which it originated.

Special text characters, however, are only a small part of the set of data values that carry contextual meaning. Crucial to process coordination in distributed systems will be the meaningful transmission of process control and synchronization primitives.

> Standardization of data formats and control semantics will be essential for successful communication. While we do have standards at the very lowest levels of data communication, such as conventions for transparent binary and character codes, we do not see similar standards even for such primitives as floating point numbers, much less for records, files, or objects. The situation for control primitives is much worse. Description of processes, interrupts, and related mechanisms is presently very difficult to communicate across computer boundaries except by specialized, ad hoc methods. [<Levin> p. 16.]

Until a mechanism is provided to pass the contextual meaning of special data values, communication between cooperating processes will continue to be supported only on a case by case basis. The requirements for such a mechanism must be formally described, and the possibility of using an extension of an intermediate format based strategy investigated.

6.1.2 Passing pointers

One of the data types eligible for exchange between processes is address pointers. The use of pointers in certain data structures, such as linked lists, is indispensable, and a network data communications scheme must support their movement.

If the communicating processes share a single address space, then passing pointers engenders no special problems. This approach is only appropriate for homogeneous computer networks, and has been implemented at CMU for local computer networks <Swan,Wulf>. Passing pointers does present a problem, however, in a heterogeneous environment. When two communicating processes exchange information, a representation of the data being transferred is moved from the address space of the sending process into the address space of the receiver. The position in the address space of data items to be referenced with pointers is important. Between the relocation of the data in the virtual memory of the receiving process and the potential differences in the addressing schemes involved, any pointers that move between processors will have to have their values adjusted. However, the necessary adjustment cannot merely be the calculation of a fixed offset within the virtual memory. Because different data types and formats are different lengths in different processing environments, the adjustment of the pointer value will depend on the kinds of items it references.

There appear to be two approaches to this problem. The sending process can pass entire data structures so that any pointer used would be "local" to the structure in a single message. By building pointers as offsets from the beginning of the structure, the receiving translator could map pointers into the values appropriate for the reformatted data.

A different approach is necessary for passing pointers that reference data structures too large to be feasible or practical to move. In this case, messages may contain pointers into the sender's address space for use by the receiver.

The principle difficulty in supporting this kind of passed pointer is the possible relocation of the sending process. The movement of absolute memory addresses can be avoided by passing offsets into the address space of the sending process. This will allow the relocation of the of the sending process within a single processor. However, if that process migrates to another processor, the appearance of its address space (ie. the length and format of the data items contained in it) will change. Maintaining passed pointers as simple offsets in this case will not be sufficient.

6.1.3 Passing programs

One of the proposed uses for network based systems is reliability through redundancy and increased performance through load sharing. Realization of either of these goals in a general way requires a mechanism to support the movement of program code through the network.

One strategy is to pass the source text of a high level language for which each host has a compiler. Code transportability has been attempted in this way by the specification of standard versions of FORTRAN and COBOL. Another possibility is the use of an intermediate programming language to bridge the gap between compilers and the object code of different processors. The application of these techniques to program passing needs to be considered.

6.1.4 Negotiating the IDF

The intermediate format to be used to represent data items as they move between IDF translators must be fully general, allowing any data values represented in one host to be transferred to another. In some cases, however, the use of general formats may be unnecessary and inefficient, and an ability to select format options for intermediate data formats may be useful.

Probably the most frequent use of such a mechanism would be to facilitate data transfer between similar processors in a network. The IDF is intended to serve as a common language for data transfer. When machines that use the same data formats for identical sets of data types wish to exchange information, no intermediate format translation need nor should occur.

One way to provide the additional flexibility offered by intermediate format options is through the use of option negotiation. By using a predetermined protocol, two communicating translators may agree to use a non-standard intermediate format or possibly to perform no data reformatting at all. The potential uses and implementation strategies for such a facility is yet one more area that requires further study.

REFERENCES

<Anderson> Anderson, R. H., et al, "The Data Reconfiguration Service -- An Experiment in Adaptable Process/Process Communication," The ACM/IEEE Second Symposium on Problems in the Optimization of Data Communications Systems, October 1971, pp. 1-9.

<ARPA> ARPANET Protocol Handbook, NIC #7104, April 1, 1976, in which can be found:

<ARPA1> McKenzie, A., "Host/Host Protocol for the ARPA Network," NIC #8246, January 1972.

<ARPA2> "TELNET Protocol Specification," NIC #18639, August 1973.

<ARPA3> "TELNET Option Specifications," NIC #18640, August 1973.

<ARPA4> Neigus, N. J., "File Transfer Protocol," NIC #17759, August 12, 1973.

<ARPA5> Anderson, R., et al, "Data reconfiguration Service -- Implementation Specification," NIC #6780, May 25, 1971.

<Bhushan> Bhushan, A. K. and R. H. Stotz, "Procedures and Standards for Inter-Computer Communications," SJCC 1968, pp. 95-104.

<Binder> Binder, R., N. Abramson, F. Kuo, A. Okinaka, and D. Wax, "ALOHA Packet Broadcasting - A Retrospect," NCC 1975, pp. 203-215.

<Chen> Chen, R. C., P. G. Jessel, and R. A. Patterson, "Mininet: a Microprocessor-controlled 'Mininetwork'," Proc. IEEE, vol. 64, no. 6, June 1976, pp. 988-993.

<Corbato> Corbato, F. J., "PL/I as a Tool for System Programming," Datamation May, 1969.

<Crocker1> Crocker, S. D., J. H. Heafner, R. M. Metcalfe, and J. B. Postel, "Function Oriented Protocols for the ARPA Computer Network," SJCC 1972, pp. 271-279.

<Crocker2> Crocker, S. D., "The National Software Works: A New Method for Providing Software Development Tools Using the ARPANET," Presented at Consiglio Nationale Delle Ricerche Istituto di Elaborazione Della Informazione meeting on 20 Years of Computer Science, Pisa, 16-19 June 1975.

<Crowther> Crowther, W. R., F. E. Heart, A. A. McKenzie, J. M. McQuillan, and D. C. Walden, "Issues in Packet Switching Network Design," NCC, 1975, pp. 161-175.

<Elovitz> Elovitz, H. S. and C. L. Heitmeyer, "What is a Computer Network?," Naval Research Laboratory Article NTC 74-1007.

<Enslow> Enslow, P. H. Jr., "What Does Distributed Processing Mean?" School of Info and Comp Sci Georgia Inst of Tech, presented at the Distributed Processing Workshop, Brown University Aug. 1976.

<Farber1> Farber, D. J. and K. C. Larsen, "The Systems Architecture of the Distributed Computing System," Proc. Symposium on Computer Networks, Brooklyn Polytechnic Institute, 1972.

<Farber2> David Farber, at the Distributed Processing Workshop, Brown University, Aug. 1976.

<Feustal1> Feustal, E. A., The Rice Research Computer - A Tagged Architecture, Rice University Laboratory for Computer Science and Engineering, ORO-4061-2, January 5, 1972.

<Feustal2> Feustal, E. A., Gedanken Machines, Rice University Laboratory for Computer Science and Engineering, ORO-4061-3, January 31, 1972.

<Frank> Frank, H., W. Chou and I. Frisch, "Topological Considerations in the Design of the ARPA Computer Network," SJCC 1970, pp. 581-587.

<Fraser1> Fraser, A. G., SPIDER -- A Data Communications Experiment, Bell Laboratories, Computing Science Technical Report #23.

<Fraser2> Fraser, A. G., "On the Interface Between Computers and Data Communications Systems," CACM July 1972, pp. 566-573.

<Fredericksen> Fredericksen, D. H., "Describing Data in Computer Networks," IBM Systems Journal, vol. 12, no. 3, 1973, pp. 257-282.

<Gordon> Gordon, R. L., "Distribution vs. Cooperation," Working paper presented at the Distributed Processing Workshop, Brown University, Aug. 1976.

<Haverty> Haverty, J., "Thoughts on Interactions in Distributed Services," NIC #36806, September 16, 1976.

<Heart> Heart, F. and R. Kahn et al., "The Interface Message Processor for the ARPA Computer Network," SJCC 1970, pp. 551-567.

<Hirt> Hirt, K. A., "A Prototype Ring-structured Computer Network Using Microcomputers," Naval Postgraduate School, NTIS AD772877, Dec. 1973.

<Hoare> Hoare, C. A. R., "Communicating Sequential Processes," Department of Computer Science, The Queen's University, Belfast, draft, August 1976.

<Illife> Illife, J. K., Basic Machine Principles, American Elsevier, New York, 1968.

<Kent> Kent, S. T., Encryption-based Protection Protocols for Interactive User-Computer Communication, MIT/LCS/TR-162, Massachusetts Institute of Technology, May 1976.

<Kernighan> Kernighan, B. W. and P. J. Plauger, The Elements of Programming Style, McGraw-Hill, New York, 1974.

<Kimbleton1> Kimbleton, S. R. and R. L. Mandell, "A Perspective on Network Operating Systems," NCC 1976, pp. 551-559.

<Kimbleton2> Kimbleton, Stephen R. and G. Michael Schneider, "Computer Communication Networks: Approaches, Objectives and Performance Considerations," Computing Surveys, vol. 7, no. 3, September 1975, pp. 129-173.

<Kimbleton3> Kimbleton, S. R. and R. L. Mandell, Distributed Computation Study, NTIS AD-A024670, April 1976.

<Levin> Levin, R., J. McQuillan, and R. Schantz, "Distributed Systems," Operating Systems Review, ACM SIGOPS, January 1977, pp. 14-19.

<Little> Little, J. L. and C. N. Mooers, "Standards for User Procedures and Data Formats in Automated Information Systems and Networks," SJCC 1968, pp. 89-94.

<Manning> Eric Manning, Private communication.

<MERIT> The MERIT Computer Network: Progress Report for the Period July 1969 - March 1971, Publication 0571-PR-4, NTIS PB 200 674.

<Metcalfe1> Metcalfe, R. M., Packet Communication, Massachusetts Institute of Technology Project MAC Report TR-114, December 1973.

<Metcalfe2> Metcalfe, R. M. and D. R. Boggs, "ETHERNET: Distributed Packet Switching for Local Computer Networks," CACM, July 1976, pp. 395-404.

<Mills> Mills, D. L., "An Overview of the Distributed Computer Network," NCC 1976, pp. 523-531.

<Millstein> Robert Millstein, at the Distributed Processing Workshop, Brown University, Aug. 1976.

<Mockapetris> Mockapetris, P. V., M. R. Lyle and D. J. Farber, "Design of Local Network Interfaces," Available from Dept. of Information and Computer Science, University of California, Irvine, CA 92717, 1977.

<MRG> MRG Progress Report, MIT Laboratory for Computer Science Domain Specific Systems Research - Microcomputer Research Group, May 1976.

<MSG> MSG: The Interprocess Communication Facility for the National Software Works, NSW #30, BBN Report #3237, January 23, 1976.

<PLI> PL/I (F) Language Reference Manual, Order no. GC28-8201-4, IBM, 1972.

<Pouzin1> Pouzin, L., "Presentation and Major Design Aspects of the CYCLADES Computer Network," Proc. 3rd Data Communications Symposium, 1974.

<Pouzin2> Pouzin, L., "CIGALE, The Packet Switching Machine of the CYCLADES Computer Network," Information Processing 74, North Holland Publishing Co., 1974, pp. 155-159.

<Roberts> Roberts, L. G., and B. D. Wessler, "Computer Network Development to Achieve Resource Sharing," SJCC 1970, pp. 543-549.

<Rowe> Rowe, L. A., "The Distributed Computing Operating System," University of Califiornia at Irvine, department of information and Computer Science Technical Report #66, June, 1975.

<Swan> Swan, R. J., S. H. Fuller, and D. P. Siewiorek, "The Structure and Architecture of Cm*: A Modular, Multi-microprocessor," Computer Science Dept., CMU, presented at the Distributed Processing Workshop, Brown University, Aug. 1976.

<Teager> Herbert M. Teager, Private communication.

<Thomas1> Thomas, R. H., D. A. Henderson, "MCROSS -- A Multi-computer Programming System," SJCC 1972, pp. 281-293.

<Thomas2> Thomas, R. H., "A Resource Sharing Executive for the ARPANET," NCC 1973, pp. 155-163.

<VanDam1> Van Dam, A., Memo #4 from the Distributed Processing Workshop held at Brown University, Aug. 1976, Computer Science Program, Brown University.

<VanDam2> Andries Van Dam, At the Distributed Processing Workshop, Brown University, Aug. 1976.

<Walden> Walden, D. C., "A System for Interprocess Communication in a Resource Sharing Computer Network," CACM, April 1972, pp. 221-230.

<Wecker> Wecker, S., "The Design of DECNET - A General Purpose Network Base," Presented at ELECTRO/76, May 1976, Boston, Ma.

<WhiteG> White, G. W., "Message Format Principles," The ACM/IEEE Second Symposium on Problems in the Optimization of Data Communications Systems, Oct. 1971, pp. 192-198.

<WhiteJ> White, J. E., "Elements of a Distributed Programming System," NWG/RFC #708, Menlo Park, CA, Jan. 5, 1976.

<Wood> Wood, D. C., "A Survey of the Capabilities of 8 Packet Switching Networks," Proc. 1975 Symposium on Computer Networks: Trends and Applications.

<Wulf> Wulf, W. A. and C. G. Bell, "C.mmp - A multi-mini-processor," AFIPS Conference Proceedings, vol. 41, part II, FJCC 1972, pp. 765-777.