TOWARD A COOPERATIVE NETWORK OF TIME-SHARED COMPUTERS

Thomas Marill

Computer Corporation of America, Cambridge, Massachusetts

and

Lawrence G. Roberts

MIT, Lincoln Laboratory, Lexington, Massachusetts*

INTRODUCTION: NETWORKS AND THE PROBLEM OF COMPUTER INCOMPATIBILITY

Incompatible machines represent an old problem in the computer field. Very often, because of computer incompatibility, programs developed at one installation are not available to users of other installations. The same program may therefore have to be rewritten dozens of times.

Assume, for example, that a program which performs a syntactic analysis of natural English sentences has been developed for a certain computer, Y. Such a program would be of interest to workers in the fields of natural-language inputs to computers, mechanical translation and linguistics, information retrieval, command and control, and a number of other ancillary disciplines. Unfortunately, the program would be available only to those who had direct access to Y (or to a computer compatible with Y); users of other installations would have to recode the program for their own computers, at a cost per computer comparable to the cost of the original programming development. Viewed on a nationwide scale, such inefficiencies can be enormously expensive.

The time-honored remedies for computer incompatibility have been the following.

1. Use identical computers. Thus, for example, the use of two identically modified IBM 7094's at Project MAC and at the MIT Computation Center made possible the development of the Compatible Time-Sharing System (CTSS), allowing programs written for one system to be run on the other.
2. Write programs in a high-level language for which compilers exist on different machines. Thus, a given FORTRAN source program can be compiled for and run on a large number of different computers.

* This work was supported in part under a subcontract of MIT, Lincoln Laboratory, which is operated with support from the Advanced Research Projects Agency.

Unfortunately, these remedies have worked quite badly in the past and will probably work as badly in future time-sharing environments. Regarding remedy (1), there is no indication that the proliferation of hardware is ending. It is not difficult to find examples of situations where, even within a single organization, incompatible new computers are being added to existing ones, requiring duplication of programming efforts. Regarding remedy (2), one finds that new computer languages are developed daily. It has been estimated that the number of time-sharing installations is roughly equal to the number of languages offered among them.

Thus we see no particular reason to believe that the old remedies will work better in the future than in the past. The possibility exists, however, that the technique of computer networking will contribute to the solution of the problem. Since time-sharing systems, by their nature, are designed to be operated remotely on a real-time basis, we can envision the possibility of the various time-shared computers communicating directly with one another, as well as with their users, so as to cooperate on the solution of problems. To return to our earlier example, if a user of machine X wanted to use, as part of a larger program, the syntactic-analysis routine running on computer Y, this larger program would communicate the sentence to be analyzed to Y, cause the syntactic-analysis program to run on Y, accept the results, and go on running at X.

Such an approach would circumvent the problem of incompatible machines by allowing all programs to run on their home computer.

A program which ran on computer Y would not need to be rewritten or recompiled for computer X in order to be part of a system running on X; the program would run on Y, as always, and its output would be communicated to X at the proper time.

Within a computer network, a user of any cooperating installation would have access to programs running at other cooperating installations, even though the programs were written in different languages for different computers. This forms the principal motivation for considering the implementation of a network.

The establishment of a network may lead to a certain amount of specialization among the cooperating installations. If a given installation, X, by reason of special software or hardware, is particularly adept at matrix inversion, for example, one may expect that users at other installations in the network will exploit this capability by inverting their matrices at X in preference to doing so on their home computers.

Carrying this train of thought one step further, it may develop that small time-shared computers will be found to be efficiently utilized when employed only as communication equipment for relaying the users' requests to some larger remote machine on which the substantive work is done. Such smaller installations might then be considered to be "retail outlets" for the "wholesale computer power" provided by the giant machines.*

SOFTWARE CONSIDERATIONS

The Elementary Approach

We will first discuss an elementary approach to the problem of forming computer networks. This approach certainly does not represent the best alternative, but it has the advantage as a point of departure of being by far the simplest, since it allows a network of existing time-sharing systems to be formed without any appreciable change to hardware or monitor software.

Using this approach, the only requirement a system must meet to be eligible for membership in the network is the following: the time-sharing monitor must allow user programs to communicate with *two* terminals. If this requirement is uniformly fulfilled, then the network can be implemented without change to the monitor at any installation, by the simple expedient of letting each computer in the network look upon all the others as though they were its own remote-user terminals.

Figuratively speaking, we may think of the computer-to-computer link in such a network as being the result of removing a user terminal from its cable on computer X, removing a user terminal from its cable on computer Y, and splicing the two cable-ends together.

* It may be pointed out that the type of network described above is not the only possible type. One might alternately consider a network in which programs are shipped from one computer to another. Such a program-shipping network could be used for load-sharing: When the queue of programs waiting to run on a given computer becomes too long, certain of the programs are shipped off to another computer where quicker service is available. The establishment of a program-shipping network seems to us to be an exceedingly difficult undertaking and will *not* be considered in the present paper. Instead, we restrict ourselves entirely to the type of network in which all programs run exclusively on their home computer, that is, on the computer they were programmed for.

Such a network operates as follows. The user dials up his home computer, C_H, from a console. He logs in normally by transmitting characters from his console to the monitor M_H. He sets up his user program P_H and starts this program. P_H, through the second channel available to it, logs in at the remote computer C_R, by transmitting the correct sequence of symbols to the remote monitor M_R. (Note that it is the user's program P_H, not the monitor M_H, which has the responsibility for doing this.) The remote monitor M_R takes the proper actions and communicates with P_H that the actions have been taken. P_H accepts these messages. P_H then communicates with M_R to set up the desired program P_R at the remote computer; it then runs P_R and transmits and accepts data from it, until it is done. P_H then logs out by communicating with M_R. P_H continues on its own until done. The user logs out by communicating with M_H.

Note that neither M_H nor M_R was required to behave in an unusual fashion. The monitors did what they always do. The only requirement, as stated earlier, was that the user program P_H be allowed to communicate with two terminals, its own user terminal and the remote computer. Most present-day monitors provide for such a capability.

There are a number of problems with this elementary approach. First, while it is true that most present-day monitors allow the user program to communicate with two terminals, they typically allow this communication to occur only at very low data-rates. This is because the multiple remote-access lines which monitors are designed to service are teletypewriter lines, operating at teletypewriter data-rates, i.e., at rates on the order of 100 bits per second. In the following section we discuss alternative approaches to circumvent this difficulty.

Second, the elementary approach leads to a network which in many circumstances is quite inconvenient. One reason is that the responsibility for making the network operate rests on the calling program P_H. This program must handle the communication with the remote monitor, possible code conversion, error checks, etc., without any assistance from the monitor. Another reason is that there is no way of "reaching" remote programs other than those which take all inputs from, and give all outputs to, a user terminal. Ways of dealing with this second class of problems will be discussed below under the headings of Message Protocol, Auxiliary Software, and The Problem of Displays.

Achieving Higher Data-Rates

We have seen how an elementary network could be achieved without changing existing hardware or monitor software. This elementary network operates at teletype data-rates, i.e., at rates on the order of 100 bits per second. We consider next how to achieve higher data-rates while still preserving, as much as possible, the hardware and software of existing time-sharing systems. Two possibilities are open to us:

1. The limitation on the rate at which bits are transmitted to and from teletypewriter stations is imposed by the speed of the teletypewriters, not by the capabilities of the hardware interfacing the remote lines to the computer. Trivial modifications to this communications interface will frequently allow the bit-rate to be increased considerably over the teletype rate. An improvement by a factor of 10, leading to a rate of approximately 1000 bits per second, should not strain any existing hardware. Such a change in the hardware does not require any change in the monitor software, provided of course, that we do not exceed the monitor's capacity for accepting characters.

2. The channels considered up to now have been command-plus-data channels which carry commands (from user to monitor) as well as data (from user to user-program and vice versa). Since commands come at unpredictable times in the data-stream, each character transferred over a command-plus-data channel must be analyzed by the monitor. Not so for the case of data-only channels, such as those which transfer information between a user program and a disc file or a magnetic tape. The data-flow over a data-only channel contains no information addressed to the monitor itself; hence the information need not be analyzed by the monitor and can be transferred at higher rates.

This fact suggests that data-rates higher than those discussed up to now could be achieved by using data-only channels as computer-to-computer links; such channels are already available in existing time-sharing systems, and could be modified for networking applications.

However, data-only links would not be sufficient for a network, since it *is* necessary to transmit commands to the monitors (in order to log in, for example); primary data-only links would have to be supplemented by secondary command-plus-data channels.

Thus, a possible alternative technique for achieving increased data-rates without greatly increasing the burden on the monitor would be to use high-rate data-only links, supplementing these by low-rate command-plus-data channels over which communication to the remote monitor could take place.

Message Protocol

We have seen how an elementary network, operating at low data-rates, could be implemented with virtually no changes to existing hardware or software, and how somewhat higher data-rates could also be achieved without radical redesign. However, the networks created by the principles discussed so far have great shortcomings, particularly since they require the user program initiating the call to the remote computer to do all the work associated with carrying out the transmissions. This can be a very great inconvenience; it would be much more desirable to have the monitor take over some of the chores.

The first step in that direction is the establishment of a message protocol, by which we mean a uniform agreed-upon manner of exchanging messages between two computers in the network.

The primary reasons for considering the establishment of a message protocol are the following:

1. By formatting transmissions into messages, and including a check-sum with each message, transmission errors can frequently be detected. If detected, the messages can automatically be retransmitted in accordance with the protocol.
2. If an existing command-plus-data link is used, certain characters or strings of characters are normally given special consideration by the monitor. Hence, binary data cannot efficiently be sent over such a link without some extra provision which specifies that a certain block is to be regarded as binary information. The protocol provides for such an extra provision and thereby allows binary information to be transmitted efficiently.

As will be seen below, work is proceeding on an experimental network between the TX-2 computer at Lincoln Laboratory and the Q-32 computer at System Development Corporation. The protocol to be used in this network is given in the Appendix. This protocol will be handled as an extension of the two monitors, which will automatically take care of error-checks, retransmissions, and acknowledgments.

While the implementation of a message protocol is a great convenience to the user, a certain cautionary remark may be in order. If a protocol is adhered to between two computers in the network, say A and B, it is not absolutely necessary that the same protocol be established between C and D, nor even between A and D. Since the motivation for the network is to overcome the problems of computer incompatibility without enforcing standardization, it would not do to require adherence to a standard protocol as a prerequisite of membership in the network. Instead, the network should be designed for maximum flexibility. If a protocol which is good enough to be put forward as a standard is designed, adherence to this standard should be encouraged but not required.

Auxiliary Software

Let us assume that we have a command-plus-data channel over which computers X and Y communicate, and that a message protocol has been arranged. Consider now the auxiliary software necessary to make use of the networking capability.

Let us say that X is the home computer (the computer on which the user is logged in) and that Y is the remote computer. There are two types of programs on Y that are of interest to the user of X:

1. "Total program packages," that is to say those programs all of whose inputs are typed in by the user and all of whose outputs are typed out to the user. An example of such a package is an on-line engineering-calculation program.
2. Subroutines, that is to say those programs which are called by other programs, whose arguments are transmitted at time of call, and which return control to the calling program together with the result of the calculation.

Consider now the auxiliary software that needs to be provided *at the remote computer Y* in order for the two types of programs to be usable remotely from X.

- To use a total program package remotely, no additional software is necessary at Y.
- To use a subroutine remotely, a *user program* running on Y must be provided. This user program is an interface between the link and the desired subroutine. The user program accepts type-in from the link, calls and runs the desired subroutine, and types out the answer to the link. A separate user program could be provided for each subroutine of interest; or else a common program could be written which is given the arguments and the name of the desired subroutine.

Consider next the auxiliary software that needs to be provided *at the home computer X*. For each remote program that is to be used, code needs to be written for calling up the remote computer, logging in, calling up the program, transferring data, and logging out. This code may as well be public, so as to be available to all users of X. Thus, if the programs ABLE and BAKER running on computer Y are to be used remotely from X, programs PSEUDOABLE and PSEUDOBAKER will be provided on X. These PSEUDO-programs will be called by the user-programs of X in the same manner as ordinary public programs at X. Since ABLE and BAKER run on the same computer, Y, PSEUDOABLE and PSEUDOBAKER will have a great deal of overlap (the dial-up and log-in routines for Y, for example). Hence PSEUDOABLE and PSEUDOBAKER will probably be written in such a way as to employ a common subroutine which handles all communications with Y. Given this routine, PSEUDOABLE and PSEUDOBAKER are trivial to write.

The Problem of Displays

There are certain special problems associated with the remote use of display programs. The first of these deals with the fact that display programs fit into neither of the two program categories discussed above. Display programs are not total program packages, since they do not type out their results; nor are they subroutines in the sense defined, since they do not return their results to the calling program, but instead send these results to a display-generator.

In order for an *existing* display program to be used remotely, therefore, it is not sufficient to introduce auxiliary software as user programs. A change to the monitor is required.

What is required is (1) that the monitor recognize the situation in which the user who is calling up the display program is not an ordinary user but rather a remote computer and (2) that it take special action in this case regarding the display information. The special action required is to ship the display information out over the link rather than to the local display. If such a monitor modification is made, existing user programs involving displays can then be called from the remote computer, and the display information will be sent to the home computer.

The second problem arises from the fact that there is no agreed-upon language for transmitting displays. The handling of displays in a time-sharing system is usually built right into the time-sharing monitor. Unfortunately, every monitor handles displays differently, with the result that each computer in the network must be programmed to understand the display languages of the others. This can certainly be done, but it is inconvenient.*

It might be pointed out that if we solve the two problems stated above in a satisfactory manner, so that display programs can be used remotely in a network of time-shared computers, we will ipso facto have solved also another problem of current interest, namely the problem of small satellite computers used as remote display consoles. In fact, there is no reason why satellite computers and network computers need be treated differently. A satellite computer also must log in, run a program, and have display information transmitted back down the communication channel.

HARDWARE CONSIDERATIONS

There exists a multitude of hardware problems that must be considered when a computer network is planned. Decisions must be made regarding type and speed of communication channels, character size, serial versus parallel transmission, synchronous versus asynchronous transmission, full-duplex versus half-duplex, etc. Since these problems have been dis-

* As in the case of message protocol, adherence to a proposed standard should be encouraged but not made mandatory. In any event, the problem at the moment is not lack of adherence to a standard, but rather lack of any proposed standard.

cussed in detail elsewhere,[1] and will also be considered by other contributors to the present conference, no further review will be undertaken here.

One comment may be in order. Automatic calling units (devices which allow computers to place calls over common-carrier dial-up networks) are becoming available. These devices allow a computer to place a call to a remote computer at the time the connection becomes necessary, eliminating the need for permanent connections. In fact, if the remote computer is to perform a lengthy computation, it would be feasible to start the remote program, hang up, and have the remote program call back when it gets the answer.

NETWORK EXPERIMENTS

Work is proceeding on the implementation of an experimental network involving the APEX time-sharing system [2] running on the TX-2 computer at MIT Lincoln Laboratory in Lexington, Massachusetts, and the time-sharing system running on the Q-32/PDP-1 computer complex at System Development Corporation in Santa Monica, California.[3] Initially, a 4KC four-wire dial-up system will be used with 1200-bit-per-second asynchronous modems. The message protocol given in the Appendix will be used. In addition, a special display language is being developed, and suitable monitor changes are planned, so that display programs can also be used remotely.

As soon as possible, a series of demonstrations and experiments will be performed using the experimental network. The experience gained will be reported at the conference. If the outcome of the experiments supports the validity of the concepts, it is hoped that other time-sharing installations will join the experimental network.

APPENDIX

MESSAGE PROTOCOL FOR TX-2/Q-32 LINK

This Appendix describes the message protocol for use with the link between the Q-32 at System Development Corporation in Santa Monica, California, and the TX-2 at Lincoln Laboratory in Lexington, Massachusetts.

Each character consists of eight data-bits, sent least significant bit first, preceded by a zero start bit and followed by a one stop bit. When not transmitting a character, the link transmits a one continuously.

Table 1. Special Characters for Message Protocol

Octal	*ASCII*	*Meaning*
HEADER		
201	SOH	characters for monitor
202	STX	characters for user
221	DC1	data for monitor
232	SS	data for user
END OF MESSAGE		
203	ETX	end of message
ACKNOWLEDGMENT		
225	NACK	message in error, repeat
234	FS	message OK, but wait
206	ACK	message OK, send next message
QUERY		
230	CNCL	resend last acknowledgment
SYNCHRONIZATION		
226	SYNC	ignore
SPECIAL FUNCTIONS		
220	DLE	help/break
233	ESC	panic.

All information transmitted is sent in the form of messages consisting of a header character, body, end-of-message character, and a checksum. All messages are acknowledged.

There are four types of messages. Each has a unique header character that determines both the destination of the message (user or monitor) and the mode of the message (character string or binary data). The specific characters used are listed in Table 1.

The body of the message has a maximum length of 119 characters if the message is a character string and 118 characters if the message is binary data. If the message consists of binary data, the first character of the body is a count character equal to the total number of characters in the body including the count character. Two through 118 are legal values.

The body of the message is followed by an end-of-message character. This is immediately followed by a checksum—the 8-bit ring-sum of the header character and all the characters in the body.

A message is acknowledged by sending one of three characters. One indicates an error, requesting retransmission. The second indicates that the message was received correctly, but requests that the transmitter wait before sending the next message, as the receiver buffers are full. The third type of acknowledgment indicates that the last message was received correctly and/or that the receiver is ready for the next message.

There are four other special characters. The first —query—requests the receiver to resend the last acknowledgment character that it sent, or to indicate an error if it is currently receiving a message or has received garbage since the last correct message.

The second—sync—will be used by another (synchronous) link attached to TX-2. As far as the SDC/TX-2 link is concerned, sync characters are ignored.

The other two—help and panic—are both treated as a break at SDC or help request at Lincoln. Eventually panic will be used for a higher-level interrupt at Lincoln.

As described herein, the system is capable of transmitting and receiving messages simultaneously. To speed up text transmission, acknowledgments and queries may be interjected in the middle of character strings (not binary data) as the receiver will always be looking for special characters when in the character mode. Interjected characters are not included in the checksum.

Since SDC desires not to transmit and receive simultaneously, programs using this system should be arranged to alternate messages or groups of messages.

To avoid "hung" conditions, the transmitter is responsible for getting the message through and acknowledged. If the transmitter does not receive an expected acknowledgment within one second, a query is sent to see if the message or acknowledgment was lost. Similarly, if a ready condition (ACK) is not received within 30 seconds of a "wait" (FS), a query is sent to determine if the ready was lost.

All special characters have the high order bit set to one. This leaves all 128 characters whose high order bit is a zero available to transmit the full 7-bit ASCII code in the character mode. Care must be exercised by programs using this system, since it is possible to send characters that could not originate from a teletype. When in the character mode, characters whose high order bit is a one are ignored if they are not special characters.

ACKNOWLEDGMENTS

The authors wish to thank the many individuals who have generously taken the time to discuss computer networks with them: at System Development Corporation, C. Fox, D. Kemper, L. Gallenson, J. Schwartz, R. von Buelow, C. Weissman; at MIT Project MAC, S. Dunton, D. Edwards, R. Stotz; at Lincoln Laboratory, J. Forgie, K. Konkle, J. Mitchell, J. Raffel; at Computer Corporation of America, W. Mann, H. Murray. In addition, the people at System Development Corporation, Lincoln Laboratory, and Computer Corporation of America are participating in setting up the experimental network, and their cooperation is very much appreciated.

REFERENCES

1. T. Marill, "A Cooperative Network of Time-Sharing Computers: Preliminary Study," Technical Report No. 11, Computer Corporation of America, Cambridge, Mass. (1966).

2. J. W. Forgie, "A Time- and Memory-Sharing Executive Program for Quick-Response On-Line Applications," *Proc. Fall Joint Computer Conf.*, vol. 27, part 1, Spartan Books, Washington, D.C., 1965, pp. 599–609.

3. J. I. Schwartz, E. G. Coffman, and C. Weissman, "A General-Purpose Time-Sharing System," Document SP-1499, System Development Corporation, Santa Monica, Calif. (1964).