4687

Title: Host Software

Author: Steve Crocker

Installation: UCLA

Date: 7 April 1969

Network Working Group Request for Comment: 1

CONTENTS

Introduction

The software for the ARPA Network exists partly in the IMPs and partly in the respective HOSTs. BB&N has specified the software of the IMPs and it is the responsibility of the HOST groups to agree on HOST software.

During the summer of 1968, representatives from the initial four sites met several times to discuss the HOST software and initial experiments on the network. There emerged from these meetings a working group of three, Steve Carr from Utah, Jeff Rulifson from SRI, and Steve Crocker of UCLA, who met during the fall and winter. The most recent meeting was in the last week of March in Utah. Also present was Bill Duvall of SRI who has recently started working with Jeff Rulifson.

Somewhat independently, Gerard DeLoche of UCLA has been working on the HOST-IMP interface.

I present here some of the tentative agreements reached and some of the open questions encountered. Very little of what is here is firm and reactions are expected.

I. A Summary of the IMP Software

Messages

Information is transmitted from HOST to HOST in bundles called messages. A message is any stream of not more than 8080 bits, together with its header. The header is 16 bits and contains the following information:

Destination	5 bits
Link	8 bits
Trace	1 bit
Spare	2 bits

The destination is the numerical code for the HOST to which the message should be sent. The trace bit signals the IMPs to record status information about the message and send the information back to the NMC (Network Measurement Center, i.e., UCLA). The spare bits are unused.

Links

The link field is a special device used by the IMPs to limit certain kinds of congestion. They function as follows. Between every pair of HOSTs there are 32 logical full-duplex connections over which messages may be passed in either direction. The IMPs place the restriction on these links that no HOST can send two successive messages over the same link before the IMP at the destination has sent back a special message called an RFNM (Request for Next Message). This arrangement limits the congestion one HOST can cause another if the sending HOST is attempting to send too much over one link. We note, however, that since the IMP at the destination does not have enough capacity to handle all 32 links simultaneously, the links serve their purpose only if the overload is coming from one or two links. It is necessary for the HOSTs to cooperate in this respect.

The links have the following primitive characteristics. They are always functioning and there are always 32 of them.

By "always functioning," we mean that the IMPs are always prepared to transmit another message over them. No notion of beginning or ending a conversation is contained in the IMP software. It is thus not possible to query an IMP about the state of a link (although it might be possible to query an IMP about the recent history of a link -- quite a different matter!).

The other primitive characteristic of the links is that there are always 32 of them, whether they are in use or not. This means that each IMP must maintain 18 tables, each with 32 entries, regardless of the actual traffic.

The objections to the link structure notwithstanding, the links are easily programmed within the IMPs and are probably a better alternative to more complex arrangements just because of their simplicity.

IMP Transmission and Error Checking

After receiving a message from a HOST, an IMP partitions the message into one or more packets. Packets are not more than 1010 bits long and are the unit of data transmission from IMP to IMP. A 24 bit cyclic checksum is computed by the transmission hardware and is appended to an outgoing packet. The checksum is recomputed by the receiving hardware and is checked against the transmitted checksum. Packets are reassembled into messages at the destination IMP.

Open Questions on the IMP Software

1. An 8 bit field is provided for link specification, but only 32 links are provided, why?
2. The HOST is supposed to be able to send messages to its IMP. How does it do this?
3. Can a HOST, as opposed to its IMP, control RFNMs?
4 Will the IMPs perform code conversion? How is it to be controlled?

II. Some Requirements Upon the Host-to-Host Software

Simple Use

As with any new facility, there will be a period of very light usage until the community of users experiments with the network and begins to depend upon it. One of our goals must be to stimulate the immediate and easy use by a wide class of users. With this goal, it seems natural to provide the ability to use any remote HOST as if it had been dialed up from a TTY (teletype) terminal. Additionally, we would like some ability to transmit a file in a somewhat different manner perhaps than simulating a teletype.

Deep Use

One of the inherent problems in the network is the fact that all responses from a remote HOST will require on the order of a half-second or so, no matter how simple. For teletype use, we could shift to a half-duplex local-echo arrangement, but this would destroy some of the usefulness of the network. The 940 Systems, for example, have a very specialized echo.

When we consider using graphics stations or other sophisticated terminals under the control of a remote HOST, the problem becomes more severe. We must look for some method which allows us to use our most sophisticated equipment as much as possible as if we were connected directly to the remote computer.

Error Checking

The point is made by Jeff Rulifson at SRI that error checking at major software interfaces is always a good thing. He points to some experience at SRI where it has saved much dispute and wasted effort. On these grounds, we would like to see some HOST to HOST checking. Besides checking the software interface, it would also check the HOST-IMP transmission hardware. (BB&N claims the HOST-IMP hardware will be as reliable as the internal registers of the HOST. We believe them, but we still want the error checking.)

III. The Host Software

Establishment of a Connection

The simplest connection we can imagine is where the local HOST acts as if it is a TTY and has dialed up the remote HOST. After some consideration of the problems of initiating and terminating such a connection, it has been decided to reserve link 0 for communication between HOST operating systems. The remaining 31 links are thus to be used as dial-up lines.

Each HOST operating system must provide to its user level programs a primitive to establish a connection with a remote HOST and a primitive to break the connection. When these primitives are invoked, the operating system must select a free link and send a message over link 0 to the remote HOST requesting a connection on the selected link. The operating system in the remote HOST must agree and send back an accepting message over link 0. In the event both HOSTs select the same link to initiate a connection and both send request messages at essentially the same time, a simple priority scheme will be invoked in which the HOST of lower priority gives way and selects another free link. One usable priority scheme is simply the ranking of HOSTS by their identification numbers. Note that both HOSTs are aware that simultaneous requests have been made, but they take complementary actions: The higher priority HOST disregards the request while the lower priority HOST sends both an acceptance and another request.

The connection so established is a TTY-like connection in the pre-log-in state. This means the remote HOST operating system will initially treat the link as if a TTY had just called up. The remote HOST will generate the same echos, expect the same log-in sequence and look for the same interrupt characters.

High Volume Transmission

Teletypes acting as terminals have two special drawbacks when we consider the transmission of a large file. The first is that some characters are special interrupt characters. The second is that special buffering techniques are often employed, and these are appropriate only for low-speed character at time transmission.

We therefore define another class of connection to be used for the transmission of files or other large volumes of data. To initiate this class of link, user level programs at both ends of an established TTY-like link must request the establishment of a file-like connection parallel to the TTY-like link. Again the priority scheme comes into play, for the higher priority HOST sends a message over link 0 while the lower priority HOST waits for it. The user level programs are, of course, not concerned with this. Selection of the free link is done by the higher priority HOST.

File-like links are distinguished by the fact that no searching for interrupt characters takes place and buffering techniques appropriate for the higher data rates takes place.

A Summary of Primitives

Each HOST operating system must provide at least the following primitives to its users. This list knows not to be necessary but not sufficient.

a) Initiate TTY-like connection with HOST x.

b) Terminate connection.

c) Send/Receive character(s) over TTY-like connection.

d) Initiate file-like connection parallel to TTY-like connection.

e) Terminate file-like connection.

f) Send/Receive over file-like connection.

Error Checking

We propose that each message carry a message number, bit count, and a checksum in its body, that is transparent to the IMP. For a checksum we suggest a 16-bit end-around-carry sum computed on 1152 bits and then circularly shifted right one bit. The right circular shift every 1152 bits is designed to catch errors in message reassembly by the IMPs.

Closer Interaction

The above described primitives suggest how a user can make simple use of a remote facility. They shed no light on how much more intricate use of the network is to be carried out. Specifically, we are concerned with the fact that at some sites a great deal of work has gone into making the computer highly responsive to a sophisticated console. Culler's consoles at UCSB and Englebart's at SRI are at least two examples. It is clear that delays of a half-second or so for trivial echo-like responses degrade the interaction to the point of making the sophistication of the console irrelevant.

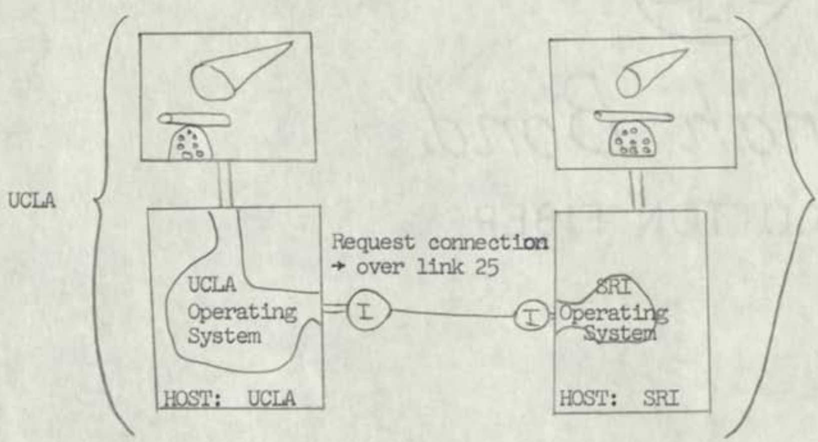
We believe that most console interaction can be divided into two parts, an essentially local, immediate and trivial part and a remote, more lengthy and significant part. As a simple example, consider a user at a console consisting of a keyboard and refreshing display screen. The program the user is talking typing into accumulates a string of characters until a carriage return is encountered and then it processes the string. While characters are being typed, it displays the characters on the screen. When a rubout character is typed, it deletes the previous non-rubout character. If the user types H E L L O ← ← P CR where ← is rubout and CR is carriage-return, he has made nine keystrokes. If each of these keystrokes causes a message to be sent which in return invokes instructions to our display station we will quickly become bored.

A better solution would be to have the front-end of the remote program -- that is the part scanning for ← and CR -- be resident in our computer. In that case, only one five character message would be sent, i.e., H E L P CR , and the screen would be managed locally.

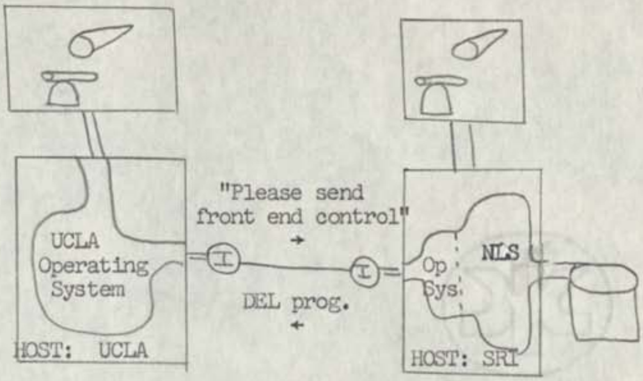
We propose to implement this solution by creating a language for console control. This language, current named DEL, would be used by subsystem designers to specify what components are needed in a terminal and how the terminal is to respond to inputs from its keyboard, Lincoln Wand, etc. Then, as a part of the initial protocol, the remote HOST would send to the local HOST, the source language text of the program which controls the console. This program would have been by the subsystem designer in DEL, but will be compiled locally.

The specifications of DEL are under discussion. The following diagrams show the sequence of actions.

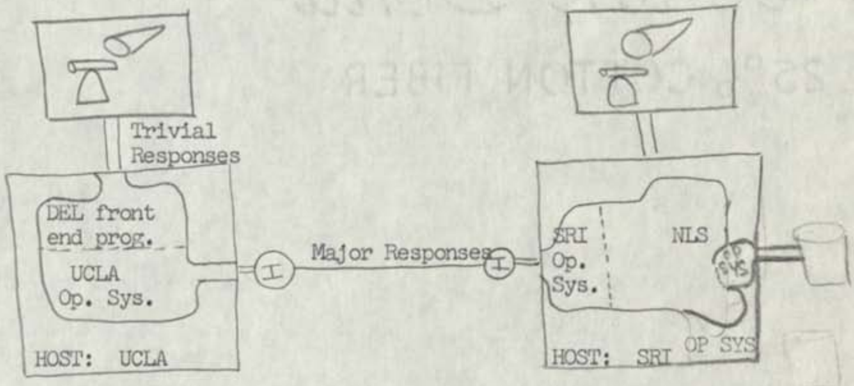
A. Before Link Establishment



B. After Link Establishment and Log-in



C. After Receipt and Compilation of the DEL program



Open Questions

1. If the IMPs do code conversion, the checksum will not be correct.

2. The procedure for requesting the DEL front end is not yet specified.

IV. Initial Experiments

Experiment One

SRI is currently modifying their on-line retrieval system which will be the major software component of the Network Documentation Center so that it can be operated with model 35 teletypes. The control of the teletypes will be written in DEL. All sites will write DEL compilers and use NLS through the DEL program.

Experiment Two

SRI will write a DEL front end for full NLS, graphics included. UCLA and UTAH will use NLS with graphics.

Title: Host Software

Author: Bill Duvall

Installation: Stanford Research Institute

Date: 9 April 1969

Network Working Group Request for Comment: 2

I LINKS

Ia Control Links

Ia1 Logical link 0 will be a control link between any two HOSTs on the network

Ia1a Only one control link may exist between any two HOSTs on the network. Thus, if there are n HOSTs on the network, there are n-1 control links from each HOST.

Ia2 It will be primarily used for communication between HOSTs for the purposes of:

Ia2a Establishing user links

Ia2b Breaking user links

Ia2c Passing interrupts regarding the status of links and/or programs using the links

Ia2d Monitor communication

Ia3 Imps in the network may automatically trace all messages sent on link 0.

Ib Primary Links

Ib1 A user at a given HOST may have exactly 1 primary link to each of the other HOSTs on the network.

Ib1a The primary link must be the first link established between a HOST user and another HOST.

Ib1b Primary links are global to a user, i.e. a user program may open a primary link, and that link remains open until it is specifically closed.

Ib1c The primary link is treated like a teletype connected over a normal data-phone or direct line by the remote HOST, i.e. the remote HOST considers a primary link to be a normal teletype user.

Ib1d The primary link is used for passing (user) control information to the remote HOST, e.g. it will be used for logging in to the remote host (using the remote hosts standard login procedure).

Ic Auxilliary Links

Ic1 A user program may establish any number of auxilliary links between itself and a user program in a connected HOST.

1c1a These links may be used for either binary or character transmission.

1c1b Auxilliary links are local to the sub-system which establishes them, and therefore are closed when that subsystem is left.

2 MANIPULATION OF LINKS

2a Control Links

2a1 The control link is established at system load time.

2a2 The status of a control link may be active or inactive

2a2a The status of the control lnk should reflect the relationship between the HOSTs.

2b Primary Links

2b1 Primary links are established by a user or executive call to the monitor

2b1a The network identification number of the HOST to be linked to must be included in the call

2b1b An attempt to establish more than one primary link to a particular HOST will be regarded as an error, and the request will be defaulted

2b1c Standard Transmission Character Set

2b1c1 There will be a standard character set for transmission of data over the primary links and control links.

2b1c1a This will be full (8 bit) ASCII.

2b1d (getlink) The protocal for establishing a link to HOST B from HOST A is as follows

2b1d1 A selects a currently unused link to HOST B from its allocation tables

2b1d2 A transmits a link connect message to B over link 0.

2b1d3 A then waits for:

2b1d3a A communication regarding that link from B

2b1d3b A certain amount of time to elapse

2b1d4 If a communication regarding the link is recieved from

B. it is examined to see if it is:

2bld4a A verification of the link from B.

2bld4a1 This results in a successful return from the monitor to the requestor. The link number is returned to the requestor, and the link is established.

2bld4b A request from B to establish the link. this means that B is trying to establish the same link as A independently of A.

2bld4b1 If the network ID number of A(Na) is greater than that of B(Nb), then A ignores the request, and continues to await confirmation of the link from B.

2bld4b2 If, on the other hand, Na<Nb, A:

2bld4b2a Honors the request from B to establish the link,

2bld4b2b Sends verification as required,

2bld4b2c Aborts its own request, and repeats the allocation process.

2bld4c Some other communication from B regarding the link.

2bld4c1 This is an error condition, meaning that either:

2bld4c1a A has faulted by selecting a previously allocated link for allocation,

2bld4c1b B is transmitting information over an un-allocated link,

2bld4c1c Or a message regarding allocation from B to A has been garbled in transmission.

2bld4c2 In this case, A's action is to:

2bld4c2a Send a link disconnect message to B concerning the attempted connection

2bld4c2b Consider the state of HOST B to be in error and initiate entry to a panic routine(error).

2bld5 If no communication regarding the link is recieved from B in the prescribed amount of time, HOST B is considered to be in an error state.

2b1d5a A link disconnect message is sent to B from A.

2b1d5b A panic routine is called(error).

2c Auxilliary Links

2c1 Auxilliary links are established by a call to the monitor from a user program.

2c1a The request must specify pertinent data about the desired link to the monitor

2c1a1 The number of the primary link to B.

2c1b The request for an auxilliary link must be made by a user program in each of the HOSTs (A and B).

2c1c If Na > Nb, then HOST A proceeds to establish a link to HOST B in the manner outlined above (getlink).

2c1d If Na<Nb, then A waits:

2c1d1 For HOST B to establish the link (after looking to see if B has already established the corresponding link).

2c1d2 For a specified amount of time to elapse.

2c1d2a This means that HOST B did not respond to the request of HOST A.

2c1d2b The program in HOST A and B should be able to specifiy the amount of time to wait for the timeout.

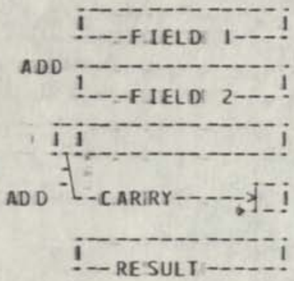
3 ERROR CHECKING

3a All messages sent over the network will be error checked initally so as to help isolate software and hardware bugs.

3b A checksum will be associated with each message, which is order dependent.

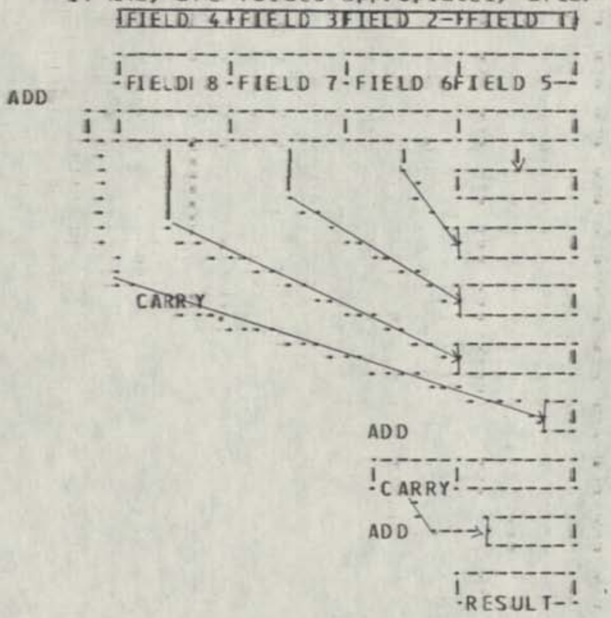
3b1 The following algorithm is one which might be used:

3b1a A checksum of length l may be formed by adding successive fields in the string to be checked serially, and adding the carry bit into the lowest bit position of the sum.



3b1a1 This process is known as folding.

3b1a2 Several fields may be added and folded in parallel, if they are folded appropiately after the addition.



3b1a2a Using this scheme, it is assumed that, if there are n fields, the carries from the first n-1 fields are automatically added into the low order position of the next higher field, so that in folding, one need only add the n result fields to the carry from the nth field, and then add in an appropiately sized carry from that addition (and repeat the desired number of times) to achieve the result.

3b1a3 A checksum computed in this manner has the advantage that, the word lengths of different machines may each be used optimally.

3b1a3a If a string of suitable length is chosen for computing the checksum, and a suitable checksum field length is selected, the checksum technique for each of the machines will be relatively optimal.

3b1a3a1 Field length: 288 bits (lowest common denomenatior of (24,32,36)

3b1a3a2 Checksum length: 8 bits (convenient field size for all machines)

3b1b If a message is divided into groups of fields, and each group is checksummed in this manner, an order dependent checksum may be got by shifting the checksum for each group, and adding it in (successively) to the checksum of the next group

3c A facility will be provided where two HOSTs may enter a mode which requires positive verification of all messages. This verification is sent over the control link.

4 MONITOR FUNCTIONS

4a Network I/O drivers

4a1 Input

4a1a Input message from IMP.

4a1b Do error checking on message.

4a1b1 Verify checksum

4a1b2 Send "message recieved" aknowledgement over control link if aknowledge mode is in effect.

4a1c (trans)character translation.

4a1c1 There is a strong possibility that the character translation may be done in the IMP.

4a1c2 This needs to be explored further with BBN.

4a1c3 There are two main considerations

4a1c3a Should the transltaion be done by table or algorithm?

4a1c3a1 Initially it seems as though the best way to go is table.

4a1c3b How should we decide which messages should be translated, - i.e. is it desirable to not translate everything (YES!!) and by what means can we use to differentiate?

4a1d Decode header, and pass message to correct recipient as identified by source, and link.

4a2 Output

4a2a Build header

4a2b Character translation

4a2b1 See remarks under the section on output translation(trans).

4a2c Create checksum

4a2d Check status of link

4a2d1 If there has not been a RFNM since the last message transmitted out the link, wait for it.

4a2e Transmit message to IMP

4a2f If aknowledge mode is in effect, wait for

4a2f1 RFNM from destination IMP.

4a2f2 Response from destination HOST over control line 0.

4b Network status

4b1 Maintain status of other HOSTs on network

4b1a If an IMP is down, then his HOST is considered to be down.

4b2 Maintain status of control lines.

4b3 Answer status queries from other HOSTs.

4b4 Inform other HOSTs as to status of primary and auxilliary links on an interrupt basis..

4b5 Inform other HOSTs as to status of programs using primary and secondary links

5 EXECUTIVE PRIMITIVES

5a Primary Links

5a1 These require the HOST number as a parameter.

5a1a Establish primary link

5a1b Connect controlling teletype to primary link

5a1c INPUT/OUTPUT over primary link

5a1d Interrogate status of primary link

5a1d1 don't know what, exactly, this should do, but it seems as though it might be useful.

5a1e Disconnect controlling teletype from primary link

5a1f Kill primary link

5b Auxilliary Links.

5b1 Establish auxilliary link.

5b1a requires the HOST number as a parameter

5b1b It returns a logical link number which is similar to a file index. It is this number which is passed to all of the other Auxilliary routines as a parameter.

5b2 INPUT/OUTPUT over auxilliary link

5b3 Interrogate status auxilliary link.

5b3a don't know what, exactly, this should do, but it seems as though it might be useful.

5b4 Kill auxilliary link.

5c Special executive functions

5c1 Transparent INPUT/OUTPUT over link

5c1a This may be used to do block I/O transfers over a link

5c1b The function of the monitor in this instance is to transfer a buffer directly to its IMP

5c1c At does not modify it in any way

5c1c1 This means that the header and other control information must be in the buffer.

5c1d The intended use of this is for network debugging.

6 INITIAL CHECKOUT

6a The network will be initially checked out using the links in a simulated data-phone mode.

6a1 All messages will be one character in length.

6a2 Links will be transparent to the monitor, and controlled by user program via a special executive primitive..

6a2a The initial test will be run from two user programs in different HOSTs, e.g. DDT to DDT.

6a2b It will be paralleled by a telephone link or similar.

Network Working Group 4689
RFC-3 April 1969
Steve Crocker
UCLA

DOCUMENTATION CONVENTIONS

The Network Working Group seems to consist of Steve Carr of Utah, Jeff Rulifson and Bill Duvall at SRI, and Steve Crocker and Gerard Deloche at UCLA. Membership is not closed.

The Network Working Group (NWG) is concerned with the HOST software, the strategies for using the network, and initial experiments with the network.

Documentation of the NWG's effort is through notes such as this. Notes may be produced at any site by anybody and included in this series.

CONTENT

The content of a NWG note may be any thought, suggestion, etc. related to the HOST software or other aspect of the network. Notes are encouraged to be timely rather than polished. Philosophical positions without examples or other specifics, specific suggestions or implementation techniques without introductory or background explication, and explicit questions without any attempted answers are all acceptable. The minimum length for a NWG note is one sentence.

These standards (or lack of them) are stated explicitly for two reasons. First, there is a tendency to view a written statement as ipso facto authoritative, and we hope to promote the exchange and discussion of considerably less than authoritative ideas. Second, there is a natural hesitancy to publish something unpolished, and we hope to ease this inhibition.

FORM

Every NWG note should bear the following information:

1. "Network Working Group"
   "Request for Comments:" x
   where x is a serial number.
   Serial numbers are assigned by Bill Duvall at SRI

2. Author and affiliation

3. Date

4. Title. The title need not be unique.

DISTRIBUTION

One copy only will be sent from the author's site to"

1. Bob Kahn, BB&N
2. Larry Roberts, ARPA
3. Steve Carr, UCLA

4. Jeff Rulifson, UTAH
5. Ron Stoughton, UCSB
6. Steve Crocker, UCLA

Reproduction if desired may be handled locally.

OTHER NOTES

Two notes (1 & 2) have been written so far. These are both titled HOST Software and are by Steve Crocker and Bill Duvall, separately.

Other notes planned are on

1. Network Timetable
2. The Philosophy of NIL
3. Specifications for NIL
4. Deeper Documentation of HOST Software.

4690

Title: Network Timetable

Author: Elmer B. Shapiro

Installation: Stanford Research Institute

Date: 24 March 1969

Network Working Group Request for Comment: 4

:N10, 03/24/69 1342:42 EBS ;

1 (n10) network checkout

2 Installation of communication gear 8/1/69

2a From AT&T and/or BBN need dimensional, power and cabling specifications

2b Need to establish SRI desired alternate locations so as to determine maximum telco cable lengths

2c Need to establish location and drops on voice coordination circuits

2d Need circuit information on voice drops for tie to intercom system

2e Need to order and instal a.c. power (coordinate with 4b)

2f See 16

3 Design and construct host-Imp interface 9/1/69

3a Need specifications from BBN

3b Develop trial design

3c Review with system programmers

3d Establish final design

3e Biuld and design hardware

3f Debug trial software with hardware loop test

4 Imp installation 9/15/69

4a From BBN get dimensional, power and cabling specifications

4b SRI orders and installs a.c. power (coordinate with 2e)

5 Debug host-Imp interface 10/1/69

5a Get debug specifications and procedures from BBN

5b Write programs to debug with BBN

5b1 Transfers of test messages

5b2 Test procedures for crash and recovery

5b3 Check message fil and stripping procedures

5c Try own transfer tests

5c1 VErify transfers to Imp

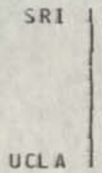
5c2 Verify transfers from Imp

5c3 Verify transfers looped with Imp

5d Work out Imp reload and restart procedures

6 Test messages between UCLA-SRI 10/15/69

6a Network configuration



6b Agree with UCLA on nature of test messages

6b1 Formats

6b2 Sequences

6b3 Checks

6b4 Test procedures

6b5 Fault reporting

6c Test integrity of messages

6d Test sequence of delivery

6e Measure delays

6f Loop with UCLA

6g System response to invalid and abnormal conditions

6h Lose and restore facilities

6h1 Communication link

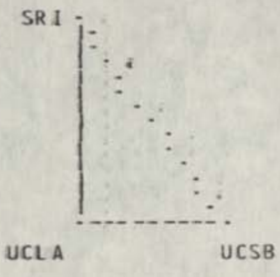
6h2 Imps

6h3 Hosts

6i Develop net trouble reporting scheme

7 Test messages between UCSB-SRI 11/15/69

7a Network configuration



7b All of 6

7c Load network for alternate routing to be effective

7d Develop voice coordination scheme

7d1 Three way conference

7d2 Design and build conference gear

7d3 Deliver conference gear to UCLA and UCSB

7e Route messages around ring

7e1 Via Imps

7e2 Via hosts

7e3 Six tests

7e3a UCLA-I, UCSB-I

7e3b UCLA-H, UCSB-I

7e3c UCLA-H, UCSB-H

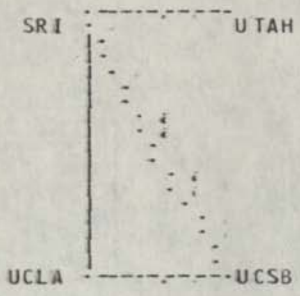
7e3d UCSB-I, UCLA-I

7e3e UCSB-H, UCLA-I

7e3f UCSB-H, UCLA-H

8 Test messages between UTAH-SRI 12/15/69

8a Network configuration



8b Selected group of previous tests

8b1 All of 6

8b2 7b

8c Expand voice coordination scheme

8c1 UTAH has access to UCLA and UCSB via SRI

8c2 with BBN and ARPA

9 Run simple TTY systems

9a Single user access

9a1 On a serving host

9a1a A to B

9a2 From a using host

9a2a A to B

9b Multiple user access

9b1 On a serving host

9b1a A,C to B

9b2 From a using host

9b2a A,A to B

9b3 Various combinations

9c Login, logout, in and out of subsystems

9d Handling of error messages, crashes, recoveries

9e Establish message formats

9f Establish protocols

9g File storage and retrieval

9h Need user's guides for each site

9i Need to establish usage schedules

9j Need to set user names

9k Design and build comm exec or its equivalent

10 Run simple typewriter systems

10a Same as 9c - 9g

10b How define when in half or full duplex mode

10c How to set "break" characters

11 Run arbitrary terminals without local feedback

12 Run arbitrary terminals

13 Move files

14 Develop debugging techniques

14a Fault detection

14a1 Conformance to manual

14a2 "REasonableness" of result

14a3 Comparison with alternate form of use

14b Cause localization

14b1 Comm-imp complex

14b2 Serving host

14b3 Using host

14b4 Try other programs

14b5 Monitor subsystem via "link" procedures, where possible

14b5a Use dialup Dataphone

14b5b Use voice coordination channel

14b6 Move canned messages

14c Cause determination

14d Cause correction

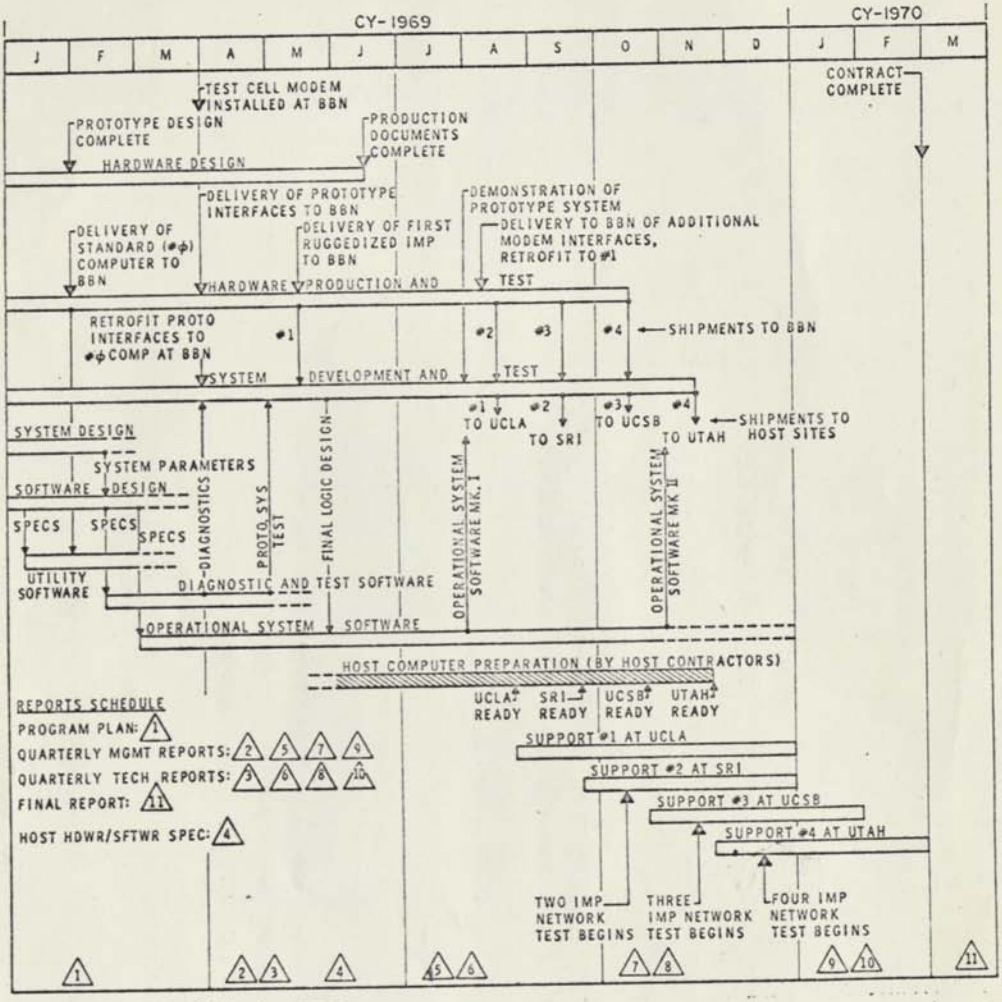


FIG. 1 IMP PROGRAM SCHEDULE

Network Working Group
RFC-5

4691
Jeff Rulifson
June 2, 1969
NIC 4691

DEL

:DEL, 02/06/69 1010:58 JFR ; .DSN=1; .LSP=0; ['=] AND NOT SP ; ['?];
dual transmission?

ABSTRACT

The Decode-Encode Language (DEL) is a machine independent language tailored to two specific computer network tasks:

accepting input codes from interactive consoles, giving immediate feedback, and packing the resulting information into message packets for network transmissin.

and accepting message packets from another computer, unpacking them, building trees of display information, and sending other information to the user at his interactive station.

This is a working document for the evolution of the DEL language. Comments should be made through Jeff Rulifson at SRI.

FORWARD

The initial ARPA network working group met at SRI on October 25-26, 1968.

It was generally agreed beforehand that the runmning of interactive programs across the network was the first problem that would be faced.

This group, already in agreement about the underlaying notions of a DEL-like approach, set down some terminology, expectations for DEL programs, and lists of proposed semantic capability.

At the meeting were Andrews, Baray, Carr, Crocker, Rulifson, and Stoughton.

A second round of meetings was then held in a piecemeal way.

Crocker meet with Rulifson at SRI on November 18, 1968. This resulted in the incorporation of formal co-routines.

and Stoughton meet with Rulifson at SRI on Decembeer 12, 1968. It was decided to meet again, as a group, probably at UTAH, in late January 1969.

The first public release of this paper was at the BBN NET meeting in Cambridge on February 13, 1969.

NET STANDARD TRANSLATORS

NST The NST library is the set of programs necessary to mesh efficiently with the code compiled at the user sites from the DEL programs it receives. The NST-DEL approach to NET interactive system communication is intended to operate over a broad spectrum.

The lowest level of NST-DEL usage is direct transmission to the server-host, information in the same format that user programs would receive at the user-host.

In this mode, the NST defaults to inaction. The DEL program does not receive universal hardware representation input but input in the normal fashion for the user-host.

And the DEL 1 program becomes merely a message builder and sender.

A more intermediate use of NST-DEL is to have echo tables for a TTY at the user-host.

In this mode, the DEL program would run a full duplex TTY for the user.

It would echo characters, translate them to the character set of the server-host, pack the translated characters in messages, and on appropriate break characters send the messages.

When messages come from the server-host, the DEL program would translate them to the user-host character set and print them on his TTY.

A more ambitious task for DEL is the operation of large, display-oriented systems from remote consoles over the NET.

Large interactive systems usually offer a lot of feedback to the user. The unusual nature of the feedback make it impossible to model with echo table, and thus a user program must be activated in a TSS each time a button state is changed.

This puts an unnecessarily large load on a TSS, and if the system is being run through the NET it could easily load two systems.

To avoid this double overloading of TSS, a DEL program will run on the user-host. It will handle all the immediate feedback, much like a complicated echo table. At appropriate button pushes, message will be sent to the server-host and display updates received in return.

One of the more difficult, and often neglected, problems is the effective simulation of one nonstandard console on another non-standard console.

We attempt to offer a means of solving this problem through the co-routine structure of DEL programs. For the complicated interactive systems, part of the DEL programs will be constructed by the server-host programmers. Interfaces between this program and the input stream may easily be inserted by programmers at the user-host site.

To minimize the number of translators needed to map any facility's user codes to any other facility, there is a universal hardware representation.

This is simply a way of talking, in general terms, about all the hardware devices at all the interactive display stations in the initial network.

For example, a display is thought of as being a square, the mid-point has coordinates (0.0), the range is -1 to 1 on both axes. A point may now be specified to any accuracy, regardless of the particular number of density of rastor points on a display.

The representation is discussed in the semantic explanations accompanying the formal description of DEL.

INTRODUCTION TO THE NETWORK STANDARD TRANSLATOR (NST)

Suppose that a user at a remote site, say Utah, is entered in the AHI system and wants to run NLS.

The first step is to enter NLS in the normal way. At that time the Utah system will request a symbolic program from NLS.

REP This program is written in DEL. It is called the NLS Remote Encode Program (REP).

The program accepts input in the Universal Hardware Representation and translates it to a form usable by NLS.

It may pack characters in a buffer, also do some local feedback.

When the program is first received at Utah it is compiled and loaded to be run in conjunction with a standard library.

All input from the Utah console first goes to the NLS NEP. It is processed, parsed, blocked, translated, etc. When NEP receives a character appropriate to its state it may finally initiate transfers to the 940. The bits transferred are in a form acceptable to the 940, and maybe in a standard form so that the NLSW need not differentiate between Utah and other NET users.

ADVANTAGES OF NST

After each node has implemented the library part of the NST, it need only write one program for each subsystem, namely the symbolic file it sends to each user that maps the NET hardware representation into its own special bit formats.

This is the minimum programming that can be expected if console is used to its fullest extent.

Since the NST which runs the encode translation is coded at the user site, it can take advantage of hardware at its consoles to the fullest extent. It can also add or remove hardware features without requiring new or different translation tables from the host.

Local users are also kept up to date on any changes in the system offered at the host site. As new features are added, the host programmers change the symbolic encode program. When this new program is compiled and used at the user site, the new features are automatically included.

The advantages of having the encode translation programs transferred symbolically should be obvious.

Each site can translate any way it sees fit. Thus machine code for each site can be produced to fit that site; faster run times and greater code density will be the result.

Moreover, extra symbolic programs, coded at the user site, may be easily interfaced between the user's monitor system and the DEL program from the host machine. This should ease the problem of console extension (e.g. accommodating unusual keys and buttons) without loss of the flexibility needed for man-machine interaction.

It is expected that when there is matching hardware, the symbolic programs will take this into account and avoid any unnecessary computing. This is immediately possible through the code translation constructs of DEL. It may someday be possible through program composition (when Crocker tells us how??)

AHI NLS - USER CONSOLE COMMUNICATION - AN EXAMPLE

BLOCK DIAGRAM

The right side of the picture represents functions done at the user's main computer; the left side represents those done at the host computer.

Each label in the picture corresponds to a statement with the same name.

There are four trails associated with this picture. The first links (in a forward direction) the labels which are concerned only with network information. The second links the total information flow (again in a forward direction). The last two are equivalent to the first two but in a backward direction. They may be set with pointers t1 through t4 respectively.

[">tif:] OR I" >nif"]; ["<tif:] OR ["<nif"];

USER-TO-HOST TRANSMISSION

Keyboard is the set of input devices at the user's console. Input bits from stations, after drifting through levels of monitor and interrupt handlers, eventually come to the encode translator. [>nif(encode)]

Encode maps the semi-raw input bits into an input stream in a form suited to the serving-host subsystem which will process the input. [>nif(hrt)<nif(keyboard)]

The Encode program was supplied by the server-host subsystem when the subsystem was first requested. It is sent to the user machine in symbolic form and is compiled at the user machine into code particularly suited to that machine.

It may pack to break characters, map multiple characters to single characters and vice versa, do character translation, and give immediate feedback to the user.

1 dm Immediate feedback from the encode translator first goes to local display management, where it is mapped from the NET standard to the local display hardware.

A wide range of echo output may come from the encode translator. Simple character echoes would be a minimum, while command and machine-state feedback will be common.

It is reasonable to expect control and feedback functions not even done at the server-host user stations to be done in local display control. For example, people with high-speed displays may want to selectively clear curves on a Culler display, a function which is impossible on a storage tube.

Output from the encode translator for the server-host goes to the invisible IMP, is broken into appropriate sizes and labeled by the encode translator, and then goes to the NET-to-host translator.

Output from the user may be more than on-line input. It may be larger items such as computer-generated data, or files generated and used exclusively at the server-host site but stored at the user-host site.

Information of this kind may avoid translation, if it is already in server-host format, or it may undergo yet another kind of translation if it is a block of data.

hrp It finally gets to the host, and must then go through the host reception program. This maps and reorders the standard transmission-style packets of bits sent by the encode programs into messages acceptable to the host. This program may well be part of the monitor of the host machine. [>tif(net mode)<nif(code)]

HOST-TO-USER TRANSMISSION

decode Output from the server-host initially goes through decode, a translation map similar to, and perhaps more complicated than, the encode map. [>nif(urt)>tif(imp ctrl)<tif(net mode)]

This map at least formats display output into a simplified logical-entity output stream, of which meaningful pieces may be dealt with in various ways at the user site.

The Decode program was sent to the host machine at the same time that the Encode program was sent to the user machine. The program is initially in symbolic form and is compiled for efficient running at the host machine.

Lines of charaters should be logically identified so that different line widths can be handled at the user site.

Some form of logical line identification must also be made. For example, if a straight line is to be drawn across the display this fact should be transmitted, rather than a series of 500 short vectors.

As things firm up, more and more complicated structural display information (in the manner of LEAP) should be sent and accommodated at user sites so that the responsibility for real-time display manipulation may shift closer to the user.

imp ctrl The server-host may also want to send control information to IMPs. Formatting of this information is done by the host decoder. [>tif(urt) <tif(decode)]

The other control information supplied by the host decoder is message break up and identification so that proper assembly and sorting can be done at the user site.

From the host decoder, information does to the invisible IMP, and directly to the NET-to-user translator. The only operation done on the messages is that they may be shuffled.

urt The user reception translator accepts messages from the user-site IMP 1 and fixes them up for user-site display. [>nif(d ctrl)>tif(prgm ctrl)<tif(imp ctrl)<nif(decode)]

The minimal action is a reordering of the message pieces.

dctrl For display output, however, more needs to be done. The NET logical display information must be put in the format of the user site. Display control does this job. Since it coordinates between (encode) and (decode) it is able to offer features of display management local to the user site. [>nif(display)<nif(urt)]

prgmctrl Another action may be the selective translation and routing of information to particular user-site subsystems. [>tif(dctrl)<tif(urt)]

For example, blocks of floating-point information may be converted to user-style words and sent, in block form, to a subsystem for processing or storage.

The styles and translation of this information may well be a compact binary format suitable for quick translation, rather than a print-image-oriented format.

(display) is the output to the user. [<nif(d ctrl)]

USER-TO-HOST INDIRECT TRANSMISSION

(net mode) This is the mode where a remote user can link to a node indirectly through another node. [<nif(decode)<tif(hrt)]

DEL SYNTAX

NOTES FOR NLS USERS

All statements in this branch which are not part of the compiler must end with a period.

To compile the DEL compiler:

Set this pattern for the content analyzer ((symbol for up arrow)P1 SE(P1) <-"-;). The pointer "del" is on the first character of pattern.

Jump to the first statement of the compiler. The pointer "c" is on this statement.

And output the compiler to file ('/A-DEL'). The pointer "f" is on the name of the file for the compiler output -

PROGRAMS

SYNTAX

```
-meta file (k=100.m=300,n=20,s=900)

file = mesdecl $declaration $procedure "FINISH";

procedure =

  procname (

      (

          type "FUNCTION" /

          "PROCEDURE" ) .id (type .id / -empty)) /

      "CO-ROUTINE") ' /

  $declaration labeledst $(labeledst ';) "endp.";

labeledst = ((left arrow symbol).id ': / .empty) statement;

type = "INTEGER" / "REAL" ;

procname = .id;
```

Functions are differentiated from procedures to aid compilers in better code production and run time checks.

Functions return values.

Procedures do not return values.

Co-routines do not have names or arguments. Their initial envocation points are given the pipe declaration.

It is not clear just how global declarations are to be??

DECLARATIONS

SYNTAX

```
declaration = numbertype / structuredtype / label / lcl2uhr /
```

```
uhr2rmt / pipetype;

numbertype = : ("REAL" / "INTEGER") ("CONSTANT" conlist /
varlist);

conlist =

    .id '(left arrow symbol)constant

    $('. .id '(left arrow symbol)constant);

varlist =

    .id ('(left arrow symbol)constant / .empty)

    $('. .id('(left arrow symbol)constant / .empty));

idlist = .id $('. .id);

structuredtype = (tree" / "pointer" / "buffer" ) idlist;

label = "LABEL1" idlist;

pipetype = PIPE" pairedids $(', pairedids);

pairedids = .id .id;

procname = .id;

integerv = .id;

pipename = .id;

labelv = .id;
```

Variables which are declared to be constant, may be put in read-only memory at run time.

The label declaration is to declare cells which may contain the machine addresses of labels in the program as their values. This is not the B5500 label declaration.

In the pipe declaration the first .ID of each pair is the name of the pipe, the second is thke initial starting point for the pipe.

ARITHMETIC

SYNTAX

```
exp = "IF" conjunct "THEN" exp "ELSE" exp;

sum = term (

    '+ sum /

    '- sum /

    -empty);

term = factor (
```

```
        '* term /
        '/ term /
        '(up arrow symbol) term /
        .empty);
    factor = '- factor / bitop;
    bitop = compliment (
        '/' bitop /
        '/'\ bitop /
        '& bitop / (
        .empty);
    compliment = "--" primary / primary;
```

(symbol for up arrow) means mod. and /\ means exclusive or.

Notice that the uniary minus is allowable, and parsed so you can write x*-y.

Since there is no standard convention with bitwise operators, they all have the same precedence, and parentheses must be used for grouping.

Compliment is the 1's compliment.

It is assumed that all arithmetic and bit operations take place in the mode and style of the machine running the code. Anyone who takes advantage of word lengths, two's compliment arithmetic, etc. will eventually have problems.

PRIMARY

SYNTAX

```
    primary =
        constant /
        builtin /
        variable / (
        block /
        '( exp ');
    variable = .id (
        '(symbol for left arrow) exp /
        '( block ') /
```

```
        .empty);

    constant =  integer / real / string;

    builtin =

        mesinfo /

        cortnin /

        ("MIN" / "MAX") exp $('. exp) '/ ;
```

parenthesized expressions may be a series of expressions. The value of a series is the value of the last one executed at run time.

Subroutines may have one call by name argument.

Expressions may be mixed. Strings are a big problem? Rulifson also wants to get rid of real numbers!!

CONJUNCTIVE EXPRESSION

SYNTAX

```
    conjunct = disjunct ("AND" conjunct / .empty);

    disjunct = negation ("OR" negation / .empty);

    negation = "NOT" relation / relation;

    relation =

        '( conjunct ') /

        sum (

          "<=" sum /

          ">=" sum /

          '< sum /

          '> sum /

          '= sum /

          '" sum /

          .empty);
```

The conjunct construct is rigged in such a way that a conjunct which is not a sum need not have a value, and may be evaluated using jumps in the code. Reference to the conjunct is made only in places where a logical decision is called for (e.g. if and while statements).

We hope that most compilers will be smart enough to skip unnecessary evaluations at run time. I.e a conjunct in which the left part is false or a disjunct with the left part true need not have the corresponding right part evaluated.

ARITHMETIC EXPRESSION

SYNTAX

```
statement = conditional / unconditional;

unconditional = loopst / cases / cibtrikst / uist / treest /
block / null / exp;

conditional = "IF" conjunct "THEN" unconditional (

   "ELSE" conditional /

   .empty);

block = "begin" exp $('; exp) "end";
```

An expressions may be a statement. In conditional statements the else part is optional while in expressions it is mandatory. This is a side effect of the way the left part of the syntax rules are ordered.

SEMI-TREE MANIPULATION AND TESTING

SYNTAX

```
treest = setpntr / insertpntr / deletepntr;

setpntr = "set" "pointer" pntrname "to" pntrexp;

pntrexp = direction pntrexp / pntrname;

insertpntr = "insert" pntrexp "as"

   (("left" / "right") "brother") /

   (("first" / "last: ) "daughter") "of" pntrexp;

direction =

   "up" /

   "down" /

   "forward" /

   "backward: /

   "head" /

   "tail";

plantree = "replace" pntrname "with" pntrexp;

deletepntr = "delete: pntrname;

tree = '( tree1 ') ;
```

```
tree1 = nodename $nodename ;

nodename = terminal / '( tree1 ');

terminal = treename / buffername / point ername;

treename = id;

treedecl = "pointer" .id / "tree" .id;
```

Extra parentheses in tree building results in linear subcategorization, just as in LISP.

FLOW AND CONTROL

```
controlst = gost / subst / loopstr / casest;
```

GO TO STATEMENTS

```
gost = "GO" "TO" (labelv / .id);

   assignlabel = "ASSIGN" .id "TO" labelv;
```

SUBROUTINES

```
subst = callst / returnst / cortnout;

   callst = "CALL" procname (exp / .emptyu);

   returnst = "RETURN" (exp / .empty);

   cortnout = "STUFF" exp "IN" pipename;

cortnin = "FETCH" pipename;
```

FETCH is a builtin function whose value is computed by envoking the named co-routine.

LOOP STATEMENTS

SYNTAX

```
loopst = whilest / untilst / forst;

whilest = "WHILE" conjunct "DO" statement;

untilst = "UNTIL" conjunct "DO" statement;

forst = "FOR" integerv '- exp ("BY" exp / .empty) "TO" exp

"DO" statements;
```

The value of while and until statements is defined to be false and true (or 0 and non-zero) respectively.

For statements evaluate their initial exp, by part, and to part once, at initialization time. The running index of for statements is not available for change within the loop, it may only be read. If, some compilers can take advantage of this (say put it in a register) all the better. The increment and the to bound will both be rounded to integers during the initialization.

CASE STATEMENTS

SYNTAX

```
casest = ithcasest / condcasest;

ithcasest = "ITHCASE" exp "OF" "BEGIN" statement $(';
statement) "END";

condcasest = "CASE" exp "OF" "BEGIN" condcs $('; condcs)
"OTHERWISE" statement "END";


condcs = conjunct ': statement;
```

The value of a case statement is the value of the last case executed.

EXTRA STATEMENTS

```
null = "NULL";
```

I/O STATEMENTS

```
iost = messagest / dspyst ;
```

MESSAGES

SYNTAX

```
messagest = buildmes / demand;

    buildmest = startmes / appendmes / sendmes;

      startmes = "start" "message";

      appendmes = "append" "message" "byute" exp;

      sendmes = "send" "message";


   demandmes = "demand" "Message";

mesinfo =

   "get" "message" "byte"

   "message1" "length" /

   "message" empty: '?;

mesdecl = "message" "bytes" "are" ,byn "bits" long" '..
```

DISPLAY BUFFERS

SYNTAX

```
dspyst = startbuffer / bufappend / estab;
```

```
startbuffer - "start" "buffer";

bufappend = "append" bufstuff $('& bufstuff);

bufstuff = :

   "parameters" dspyparm $('. dspyparm) /

   "character" exp /

   "string"1 strilng /

   "vector" ("from" exp ':exp / .empty) "to" exp '. exp /

   "position" (onoff / .empty) "beam" "to" exp '= exp/

   curve" ;

dspyparm F :

   "intensity" "to" exp /

   "character" "width" "to" exp /

   "blink" onoff /

  "italics" onff;

onoff = "on" / "off";

estab = "establish" buffername;
```

LOGICAL SCREEN

The screen is taken to be a square. The coordinates are normalized from -1 to +1 on both axes.

Associated with the screen is a position register, called PREG. The register is a triple <x.y.r> where x and y specify a point on the screen and r is a rotation in radians, counter clockwise, from the x-axis.

The intensity, called INTENSITY, is a real number in the range from 0 to 1. 0 is black, 1 is as light as your display can go, and numbers in between specify the relative log of the intensity difference.

Character frame size.

Blink bit.

BUFFER BUILDING

The terminal nodes of semi-trees are either semi-tree names or display buffers. A display buffer is a series of logical entities, called bufstuff.

When the buffer is initilized, it is empty. If no parameters are initially appended, those in effect at the end of the display of the last node in the semi-tree will be in effect for the display of this node.

As the buffer is built, the logical entities are added to it. When it is established as a buffername, the buffer is closed, and further appends are prohibited. It is only a buffername has been established that it may be used in a tree building statement.

LOGICAL INPUT DEVICES

Wand

Joy Stick

Keyboard

Buttons

Light Pens

Mice

AUDIO OUTPUT DEVICES

.end

SAMPLE PROGRAMS

Program to run display and keyboard as tty.

to run NLS

input part

display part

DEMAND MESSAGE;

While LENGTH " O DO

ITHCASE GETBYTE OF Begin

ITHCASE GETBYTE OF %file area uipdate% BEGIN

%literal area%

%message area%

%name area%

%bug%

%sequence specs%

%filter specs%

%format specs%

%command feedback line%

%filer area%

%date time%

%echo register%

BEGIN %DEL control%

DISTRIBUTION LIST

Steve Carr
Department of Computer Science
University of Utah
Salt Lake City, Utah 84112
Phone 801-322-7211 X8224

Steve Crocker

Boelter Hall
University of California
Los Angeles, California
Phone 213-825-4864

Jeff Rulifson

Stanford Research Institute
333 Ravenswood
Menlo Park, California 94035
Phone 415-326-6200 X4116

Ron Stoughton

Computer Research Laboratory
University of California
Santa Barbara, California 93106
Phone 805-961-3221

Mehmet Baray

Corey Hall
University of California
Berkeley, California 94720
Phone 415-843-2621

4692

Title: Conversation with Bob Kahn

Author: Steve Crocker

Installation: University of California at Los Angeles

Date:10 April 1969

Network Working Group Request for Comment: 6

Network Working Note
RFC-6

Steve Crocker, UCLA
10 April 1969

CONVERSATION WITH BOB KAHN

I talked with Bob Kahn at BB&N yesterday. We talked about code conversion in the IMP's, IMP-HOST communication, and HOST software.

BB&N is prepared to convert 6, 7, 8, or 9 bit character codes into 8-bit ASCII for transmission and convert again upon assembly at the destination IMP. BB&N plans a one for one conversion scheme with tables unique to the HOST. I suggested that places with 6-bit codes may also want case shifting. Bob said this may result in overflow if too many case shifts are necessary. I suggested that this is rare and we could probably live with an overflow indication instead of a guarantee.

With respect to HOST-IMP communication, we now have a five bit link field and a bit to indicate conversion. Also possible is a 2-bit conversion indicator, one for converting before sending and one for converting after. This would allow another handle for checking or controlling the system.

The HOST can send messages or portions of a message to its IMP specifying

1. Tracing
2. Conversion
3. Whether message is for destination IMP or HOST
4. Send RFNM
5. HOST up or down
6. Synchronization
7. Format Error Messages
8. Master Link Clear
9. Status Requested

The IMP can send to its HOST information on

1. Conversion
2. REFNM Arrived
3. IMP up or down
4. Synchornization
5. Called HOST not Responding
6. Format Error
7. Status in IMP

I also summarized for Bob the contents of Network Notes 1, 2, and 3.

retyped by ARC
original attached 4693

HOST-IMP INTERFACE (NWG/RFC 7)
ARPA Network: Specification Outlines of the HOST-IMP (HI) Interface Programs

G. Deloche
May 1969; UCLA

Outline

I. Introduction

This paper is concerned with the preliminary software design of the HOST-IMP interface. Its main purpose is on the one hand to define functions that will be implemented, and on the other hand to provide a base for discussions and . . .(unreadable).

This study is based upon a study of the BBN Report No. 763.

II. Scope of the software organization.

The system is based upon two main programs: the Handler program that drives the channel hardware unit, and the Network program which carries out the user's transmission requests.

As the communication is full duplex, each of these programs can be viewed as divided into two parts: one is concerned with the output data, the other with the input. (See fig. 1)

These two programs exchange data through a pool of buffers, and logical information through an interface table.

In the following we only focus on the output part of each program (see fig. 2). The input part would be very similar.

II-1 Network Program

II-1-1 Multiplex function

This program multiplexes the outgoing messages (and distributes the incoming messages) - The multiplexing consists in stacking up all ther user's (or caller, or party) . . .(unreadable) and filling up the pool of buffers so as to keep the handler busy emitting

Multiplexing (and distribution) is based on the link identification numbers. (Link - logical connection between two users). The multiplexing problem is closely related to the interface between a user's program and the network program, that is in fact . . .(unreadable) operating system (see below: Questions)

II-1-2 Output message processing

When a user's program wants to send out text it should indicate the following information (through a macro, or as call p parameters): text location, text length in bytes, and destination-

Using these data the Network program:

* prepares a 16 bit Host Leading (1 bit: trace, 2 bits: spares, 8 bits: link identification no., 5 bits: destination host)

* inserts a 16 bit marking between the header and the text so as to start the text at a word boundary. This marking consists of a one preceding the first bit of the text and, in turn, preceded by fifteen zeros to fill up the gap.

* checks he length of the user's text - if it exceeds 1006 bytes [.xxx.] the program breaks down the text into a sequence of messages whose maximum length is 1006 bytes - Each of these messages is preceded by a heading as explained above.

Remark: in that case one of the heading spares could be used for indicating that several messages belong to the same text.

* transcodes the EBCDK characters constituting the messages into ASCII characters.

* fills the buffers of the pool with the contents of the messages.

* updates the content of the interface table and moves the filling pointer (see below)

II-2 Handler Program

This program is initiated either by the network program, or by the I/O interrupt

This program will be very snort. It will be coded in master mode (privileged instructions) and should be integrated in the I/O supervisor of the operating system.

This program:

* controls the channel hardware unit. It initiates the emission * controls the channel hardware unit. It initiates the emission, eventually provides data chaining between the buffers, tests the different device status upon receiving an interrupt

empties the buffers that are filled up by the network program.

* explores and updates the interface table (see below)

* can eventually insure a control transmission procedure with the IMP (See Questions)

II-3 Buffers and Interface Table.

II-3;1 Buffers.

They should be large enough for containing the maximum host message text + heading and marking (1006 + 4 = 1010 bytes)

Consequently the buffer size could be chosen equal to 256 words (1024) bytes). As for the buffer number it will determine by the link utilization frequency -

II-3-2 Interface Table

It is through this table that the network program informs the handler with the location and length of the emitting data.

This table could be a ring table with two pointers: one for filling, the other for extracting. They are respectively updated by the network and the handler programs.

*****

III. Questions

III-1 Why is there not a simple control procedure between the host and the IMP? What happens if a message, issued from the HOST, reaches the IMP with an error due to the transmission?

From the BBN specifications it appears that this error will be transmitted as far as the receiving HOST.

In that case must an HOST-HOST control procedure be provided?

III-2 Where will the special channel hardware unit be connected (MIOP/SIOP)?

How will this device be notified of an outgoing message end in order to start the padding?

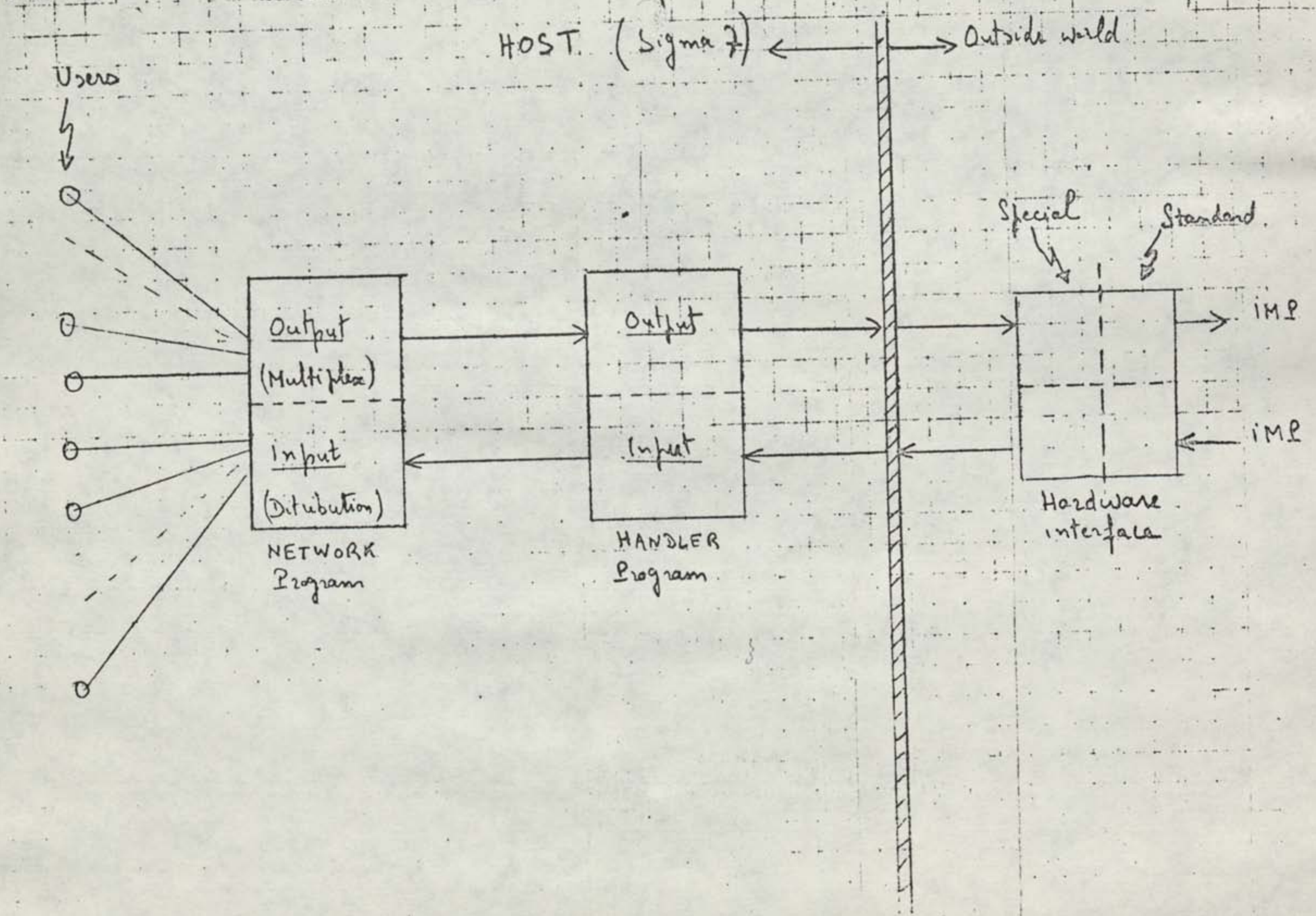
(The program will provide to the MIOP/SIOP the number of bytes of the outgoing message, and will receive back an interrupt when the last byte is sent out. Is it that signal which will be also sent to the special device?)

Vice versa how does the Handler know the length of the incoming message? From the contents of the previous one or should this program always be ready to receive a message of maximum length? (then an interrupt should be triggered when the real end is detected by the hardware).

III-3 When does the Gordo documentation will be available in order to design the user-network program interface. What are the mechanisms for program initiations, transferring parameters from one program to another etc.

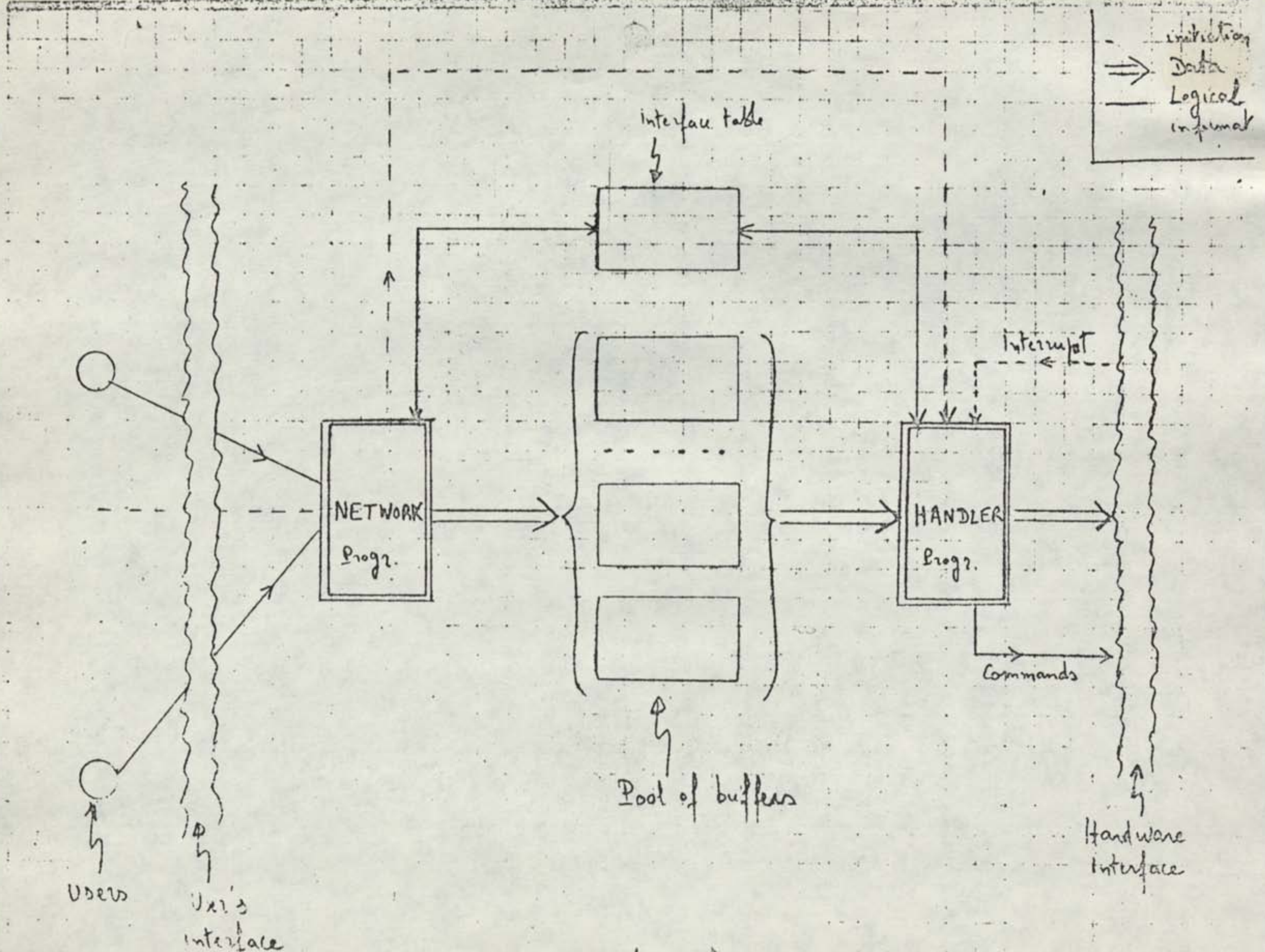
Fig. 1

Fig. 2



HOST (Sigma 7)
Outside world
Users
Output
(Multiplex)
Input
(Distribution)
NETWORK
Program
Output
Input
HANDLER
Program
Special
Standard
Hardware
interface
IMP
IMP

(Fig 1)


initiation
Data
Logical
informat
interface table
Interrupt
NETWORK
Progr.
HANDLER
Progr.
Commands
Pool of buffers
Users
User's
interface
Hardware
Interface

Title: Host-Imp Interface

Author: G. Deloche

Installation: University of California at Los Angeles

Date: May 1969

Network Working Group Request for Comment: 7

G. Deloche → Prof. J. Estrin
Prof L. Kleinrock
Prof B. Bussel
D. Mandell
S. Cracker
L. Bonamy

Object : Arpa Network _ Specification outlines : HOST-IMP (Hi) interface programs _

I Introduction

II Scope of the software organization

II-1 Network program

II-2 Handler program

III Questions

I Introduction

This paper is concerned with the [illegible] software design of the HOST IMP interface. [illegible] main purpose is [illegible] on one hand to [illegible] functions that will [illegible] implemented, and on [illegible] hand to provide a base [illegible] on.

This study is based upon a study of [illegible] report no. [illegible] 62 –

II Scope of the software organization.

The system is based upon two main programs: the Handler program that drives the channel hardware unit, and the Network program which carries out the user's transmission requests.

As the communication is full duplex, each of these programs can be viewed as divided into two parts: one concerned with the output data, the other with the input. (See fig. 1)

These two programs echange data through a pool of buffers, and logical information through an interface table.

In the following we only focus on the output part of each program (See fig 2). The input part would be very similar.

II-1 Network program.

II-1-1. Multiplex function

This program multiplexes the outgoing messages (and distributes the incoming message). The multiplexing consists in stacking up all the user's (or caller, or party) requ and filling up the pool of buffers so as to keep the handler busy emitting.

Multiplexing (and distribution) is based on the [illegible] identification numbers. (Link = logical connection between two users). The multiplexing problem is [illegible] related to the interface between an user's [illegible] and the the network program, that is in fact [illegible] operating system (See below Questions)

II-1-2 Output messages processing.

When an user's program wants to send out text it should indicate the following informat[illegible] (through a macro, or as call parameters) : text loc[illegible] text lenght in bytes, and destination.

Using those data the Network program:

* prepares a 16 bits Host Leading (1 bit: trace, [illegible] bits: spares, 8 bits: link identification n°, 5 bits: dest[illegible] host)

* inserts a 16 bits marking between the [illegible] and the text so as to start the text at a word boundary. This marking consists of a one preced[illegible] the first bit of the text and, in turn, preceded by fifteen zeros to fill up the gap.

* checks the lenght of the user's text. If it ex[illegible] 1006 byts $\left[\frac{\text{8080 (max Host message lenght)} - \text{32 (heading + marking)}}{\text{8 (byte = 8 bits)}}\right.$

the program breaks down the text into a sequence of mess[illegible] whose maximum lenght is 1006 byts. Each of these me[illegible] is preceded by a heading as explained above.

Remark : in that case one of the heading spare bits c be used for indicating that several messages belong the same test.

* transcodes the EBCDIC characters constituting th messages into ASCII characters.

* fills the buffers of the pool with the content of the messages.

* updates the content of the interface table and mov the filling pointer (see below)

II - 2 Handler program.

This program is initiated either by the network program, or by the I/O interrupt.

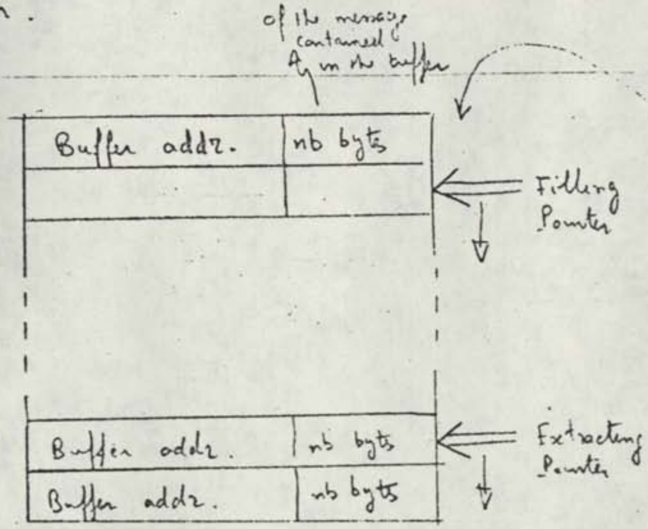
This program will be very short. It will be cod in master mode (privileged instructions) and should be integrated in the I/O supervisor of the operating sy

This program :

* controls the channel hardware unit. It init the emission, eventually provides data chaining between the buffe tests the different device status upon receiving an interru

* empties the buffers that are filled up by

This table could be a ring table with 2 pointers one for filling, the other for extracting - they are respectively updated by the network and the handler program.



III Questions

III-1. Why is there not a simple control procedure between the HOST and the IMP? What happens if a message, issued from the HOST, reach the IMP with an error due to the transmission?
From the BBN specifications it appears that the error will be transmitted as far the receiving HOST
In that case must an HOST-HOST control procedure be provided?

III-2. Where will the special channel hardware unit be connected (MIOP/SIOP)?
How will this device be notified of an outgoing message end in order to start the padding?
(The program will provide to the MIOP SIOP the number of bytes of the outgoing message, and will receive back an interrupt when the last byte is sent out.
Is it that signal which will be also sent to the special device?)
Vice versa how does the Handler know the length of the incoming message? From the content of the previous o
or should this program always ready to receive a mes
of maximum lenght? (then an interrupt should be trigg
when the real end is detected by the hardware)

III.3 When does the Cordo documentation will be available in order to design the user-network program interface. What are the mecanisms for program initiations, transfering parameters from one program to another etc..

4694

Title: ARPA Network Functional Specifications

Author: G. Deloche

Installation: University of California at Los Angeles

Date: 5 May 1969

Network Working Group Request for Comment: 8

ARPA network: Functional specif.

4-5

TABLE OF CONTENTS.

I Transmission features

I-1 Transmission checking

There exists two kind of transmission checking :

* IMP to IMP

It is a cyclic checksum computed and checked by the BBN hardware

* HOST to HOST

It is a special 16 bits checksum compiled and checked by the HOST programs.

For this purpose a HOST memory is broken down into 1152 bits pieces A, B, C ... (1152 = 2.24 # packet
For each of pieces, we calculate an end-around carry sum and form the checksum as follows :

Checksum = Sum of A + 2 * Sum of B + 4 * Sum of C etc..

This 16 bits checksum is located just after the marking of the HOST heading, that is at the beginning of a message text (See fig 1)

This checking procedure allows the verification of the right IMP to IMP procedure. It also protects against HOST to IMP (or IMP to HOST) bad transmission, and against IMP packet number inversion.

Remark : Example of an end around carry sum :

$$\begin{array}{r} 101 \\ +\ 101 \\ \hline 1010 \end{array}$$

Checksum = 011

I - 2 HOST (A) to HOST (B) links.

32 links are possible between two HOSTs. Each of those links are used as full duplex.

Link 0 is considered as a control link (request connection, status of any kind ....)

The 31 others are used either for "teletype like" connections or for file transmission connections.

A "TTY like" connection is one where :

- ASCII character are sent or received
- Echos are generated by the remote HOST
- The remote HOST looks for specific characters (break or interrupt control characters).
- The transmission is slow.

II Functional software specifications.

- See fig 2 -

II-1 User program - DEL language

Its an application program that exists within a HOST. For example the NLS program at SRI. For network purposes this program should be viewed as parted in two: The local part and the hard par. (the body).

- The hard part represents the user application.
- The local control part is the user interfa: It exerts immediate control of the terminal and provid specific responses to the man's inputs.

In order to facilitate and speed up remote interac the 'local control' program can be transmitted to another HOS Thanks to that capability an UCLA user, for example, will us its terminal exactly like the SRI user uses it own. Also only the program data are transmitte over the link (versus the user terminal dialogue) - See f

DEL language. (Decode Encode Language)

The "local control" program should be written in t DEL language - when it is transmitted over to a remot

[illegible]

II-2 Network program

This program should provide:

- The outgoing messages multiplexing (and incoming messages distribution)
- The link initiation procedure: see below.
- The HOST message Heading.
- The "HOST-HOST" checksum computation/checking.
- The receiving of the RFNM control messages.
- The supervisory control of the Handler program.

II-3 Transmission Handler program

This program is initiated either by the network program, or by the I/O interrupt. Its purpose is to control the channel hardware unit.

This program is very short and closely related to the Network program.

Remark: As the communication is full duplex the Net and Handler programs can be viewed as divided into 2 parts: one is concerned with the outgoing messages, the other with the incoming messages.

III Link establishment procedure

III-1 General procedure

* Establish link to HOST (x).
A "TTY like" connection is established to HOST(x). connection is in a pre-log-in state. Standard TTY c... are expected. The remote HOST provides the echo.

* Send/Receive characters over "TTY like" link.

* Establish file transmission link parallel to existing "TTY like" link. This must be executed by both HOST user programs.

* Send/Receive over "file like" link.

III-2 Example

Suppose that we, at UCLA, want to use NLS at SRI

a) Local arrangements
* Log in on local TTY to Sigma 7. We are now talking to the command level of the Sigma operating system.
* Select an user program to put in executi...

We start up a program we previously write. ..
our TTY and the transmission with SRI.

* Ω3 select the standard UCLA communicat
program. This is the standard option for any
control of a remote HOST.

b) Connection to SRI

* Initiate link to remote HOST

The previously selected program asks the UCLA Net program to initiate a link to SRI. The Network program:

- Selects an open link e.g 25
- Sends a message to SRI over link 0 re connection on link 25.
- Waits for an acceptance from the SRI network program. This acceptance is in the form of another message over link 0.
- If it should happen that both SRI a UCLA try to initiate a connection over 25, the one with the higher priority w prevail. (This is extremely rare) We suggest that the priority be exactly the HOST indentification number.
- This connection is teletype-like connec only a standard subset of ASCII charac is expected or accepted.
- The connection is a "pre-log-in" con The remote HOST expects its standard log-in sequence

* Log-in at SRI.

This may be done either by the UCLA user program if it knows how, or by the man at UCLA by typing the required sequence. We are now talking to the command level of the SRI operating system.

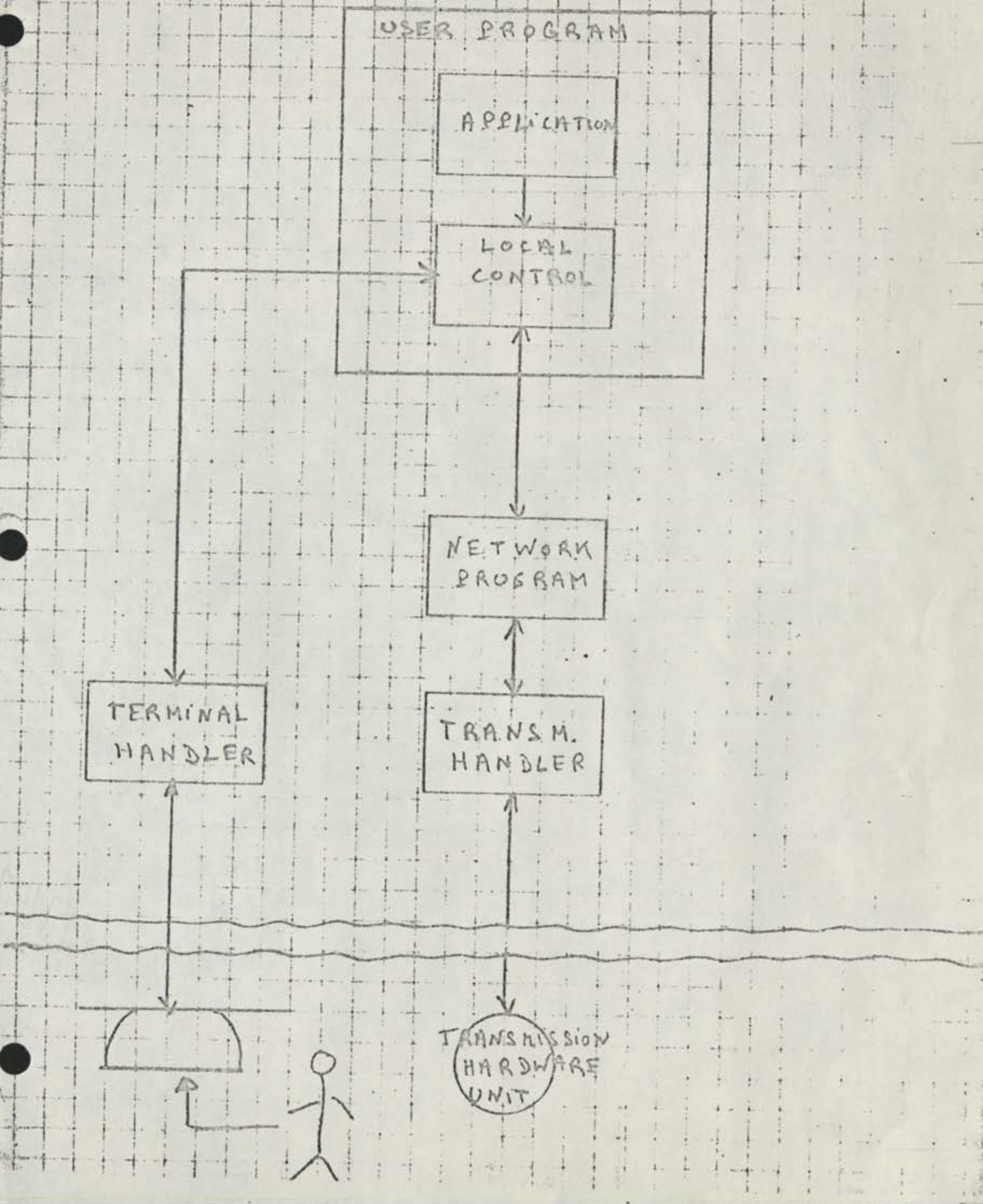
Get user prog at SRI into a state

c) Request 'local control' program for SRI.

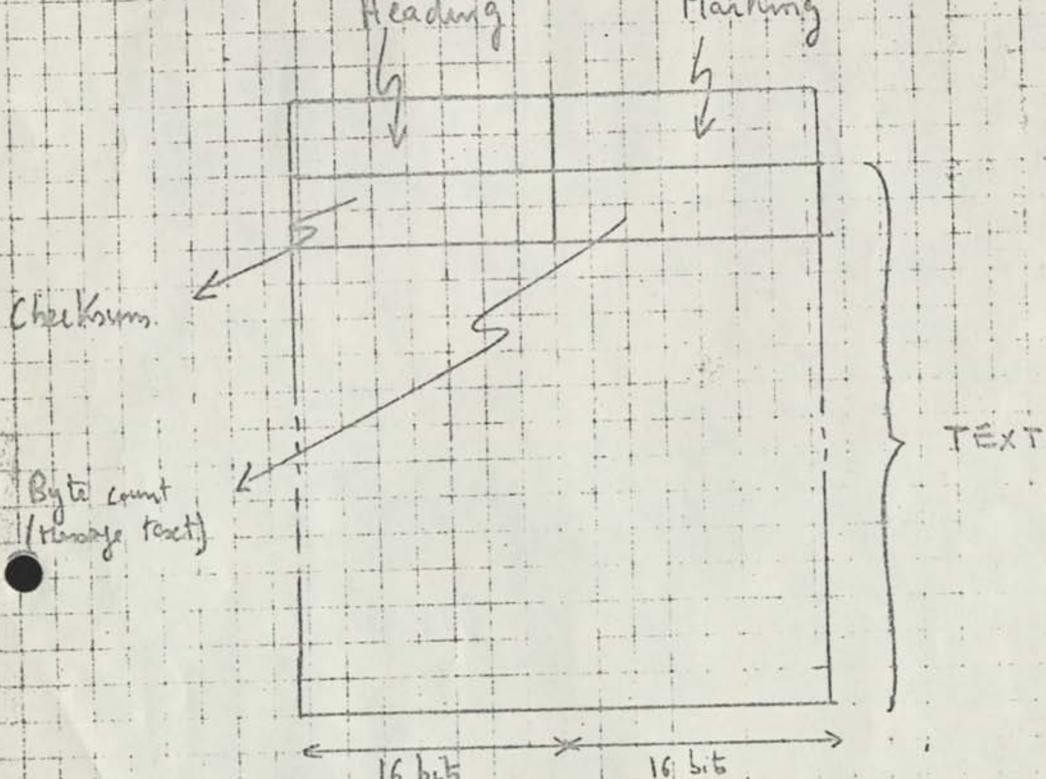
* The UCLA selected program sends a message over the link to the SRI user program. The message requests that SRI transmit to UCLA the 'local control' program which is written in the DEL language

* We compile this program through a compiler

* We turn control of the TTY link and terminal over the just compiled DEL program



USER PROGRAM
APPLICATION
LOCAL CONTROL
NETWORK PROGRAM
TERMINAL HANDLER
TRANSM. HANDLER
TRANSMISSION HARDWARE UNIT



(Fig 1) : UCLA HOST message

FUNCTIONAL SPECIFICATIONS FOR THE ARPA NETWORK (NWG/RFC 8)

G. Deloche
5 May 1969; UCLA

TABLE OF CONTENTS

I. Transmission Features

I-1 Transmission checking

There exists two kinds of transmission checking:

* IMP to IMP

It is a cyclic checksum computed and checked by the BBN hardware

* HOST to HOST

It is a special 16 bit checksum computed and checked by the HOST programs.

For this purpose a HOST message is broken down into 1152 bits pieces A, B, C . . . (1152 = 224 sup 2 # pocket . . .(unreadable)

For each of pieces, we calculate an end-around carrying sum and form the checksum as follows:

Checksum = Sum of A + 2* Sum of B + 4* Sum of C etc. . .

This 16 bits checksum is located just after the marking of the HOST heading, that is at the beginning of a message text (see Fig. 1)

This checking procedure allows the verification of the right IMP to IMP procedure. It also protects against HOST to IMP (or IMP to HOST) bad transmission and against IMP packet number inversion.

Remark: Example of an end-around carry sum

I-2 HOST(A) to HOST(B) links

32 links are possible between two HOSTS. Each of those links are viewed as full duplex. Link 0 is considered as a control link (request connection, status of any kind. . ..). The 31 others are used either for "teletype like" connections or for file transmission connections. A "TTY like" connection is one where:

- ASCII characters are sent or received.

- Echos are generated by the remote HOST

- The remote HOST looks for specific character (break or interrupt control characters).

- The transmission is slow.

II. Functional Software Specifications (See Fig. 2)

II-1 User Program - DEL language

It's an application program that exists within a HOST. For example the NLS program at SRI. For network purposes this program should be viewed as parted in two: The local part and the hard part (the body).

- The hard part represents the user application.

- The local control part is the user interface. It exerts immediate control of the terminal and provides specific responses to the man's inputs.

In order to facilitate and speed up remote interaction the 'local control' program can be transmitted to another Host. Thanks to that capability an UCLA user, for example, will use its terminal exactly like the SRI user uses its own. Also only the program data are transmitted over the link (versus the user- terminal dialogue) See fig. ?

DEL language. (Decode Encode Language)

The "local control" program should be written in the DEL language- when it is transmitted over to a remote HOST . . .(unreadable line).

II-2 Network Program

This program should provide:

- The outgoing messages multiphasing (and incoming message distribution)

- The link initiation procedure: see below

- The HOST message Heading.

- The "HOST-HOST" checksum computation/checking.

- The receiving of the RFNM control messages.

- The supervisory control of the Handler program.

II-3 Transmission Handler Program

This program is initiated either by the network program, or by the I/O interrupt. Its purpose is to control the channel hardware unit.

This program is very short and slosely related to the Network program.

Remark- As the communication is full duplex, the Network and Handler programs can be viewed as divided into 2 parts: one is concerned is the outgoing messages, the other with the incoming messages.

III. Link Establishment Procedure

III-1 General Procedure

* Establish link to HOST(X). A "TTY like" connection is established to HOST(X). The connection is in a pre-log-in state. Standard TTY . . .(unreadable ?codes) . . .are expected. The remote HOST provides the echo.

* Send/Receive characters over "TTY like" link.

* Establish file transmission link parallel to existing "TTY like" link. This must be executed by both HOST user programs.

* Send/Receive over "file like" link

III-2 Example

Suppose that we, at UCLA, want to use NLS at SRI

A. Local arrangements

* Log-in on local TTY to Sigma f. We are now talking to the command level of the Sigma operating system.

* Select an user program to put in executive on the Sigma f. We start up a program we previously wrote. It will cont??? our TTY and the transmission with SRI

* Or select the standard UCLA communication program. This is the standard option for simple control of a remote HOST.

B. Connection to SRI

* Initiate link to remote HOST. The previously selected program asks the UCLA Network program to initiate a link to SRI. The Network program:

- Selects an open link e.g. 25

- Sends a message to SRI over link 0 . . .(unreadable) connection on link 25.

- Waits for an acceptance from the SRI network program. This acceptance is in the form of another message over link 0.

- If it should happen that both SRI and UCLA try to initiate a connection over 25, the one with the higher priority would prevail. (This is extremely rare). We suggest that the priority be exactly the HOST identification number.

- This connection is teletype-like connection only a standard subset of ASCII characters is expected or accepted.

- The connection is a "pre-log-in" connection. The remote HOST expects its standard log-in sequence.

* Log-in at SRI

This may be done either by the ucla user program, if it knows how, or by the man at UCLA by typing the required sequence. We are now talking to the command level of the SRI operating system.

(in margin): Get user program at SRI into execution.

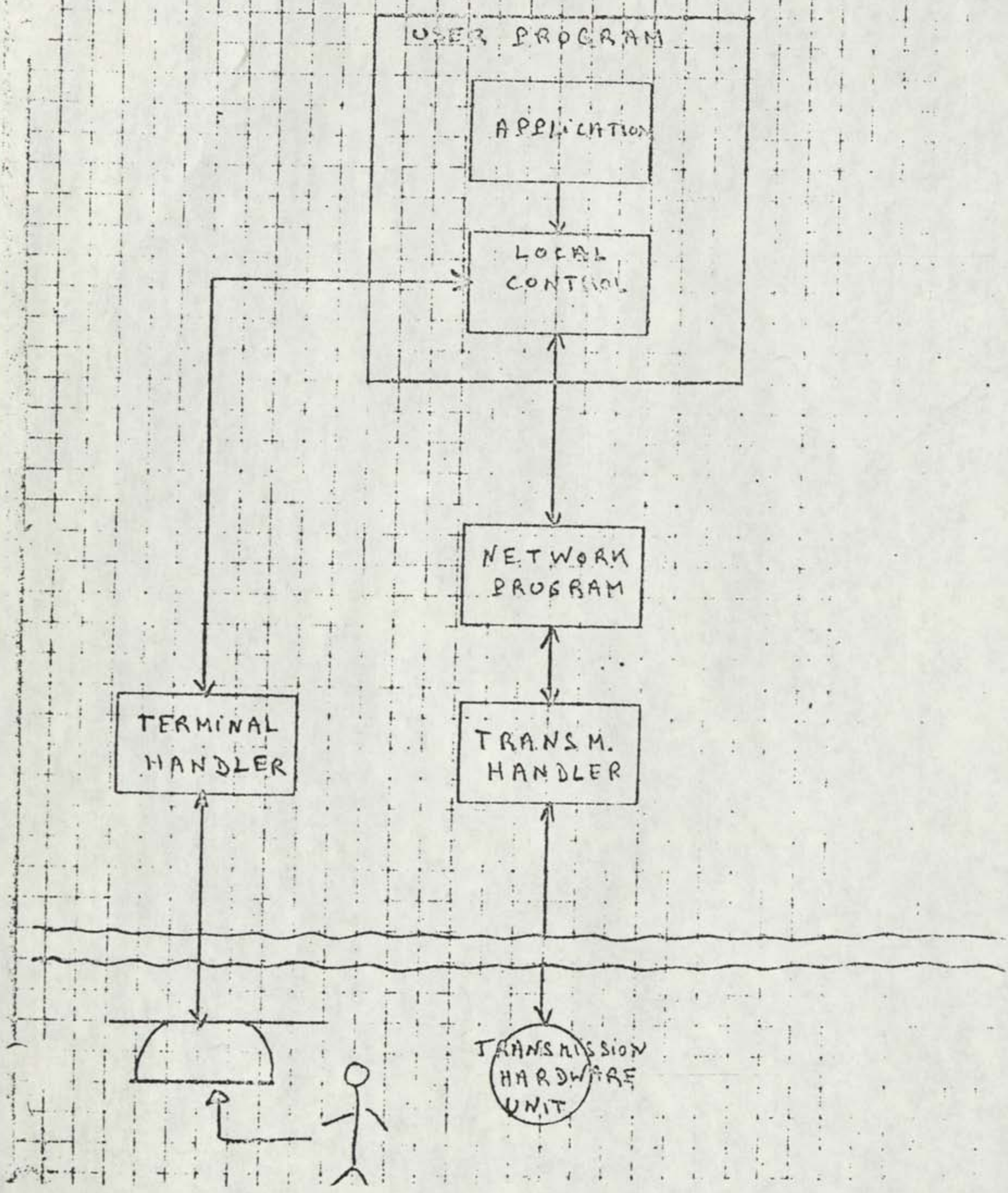
C. Request 'local control' program from SRI.

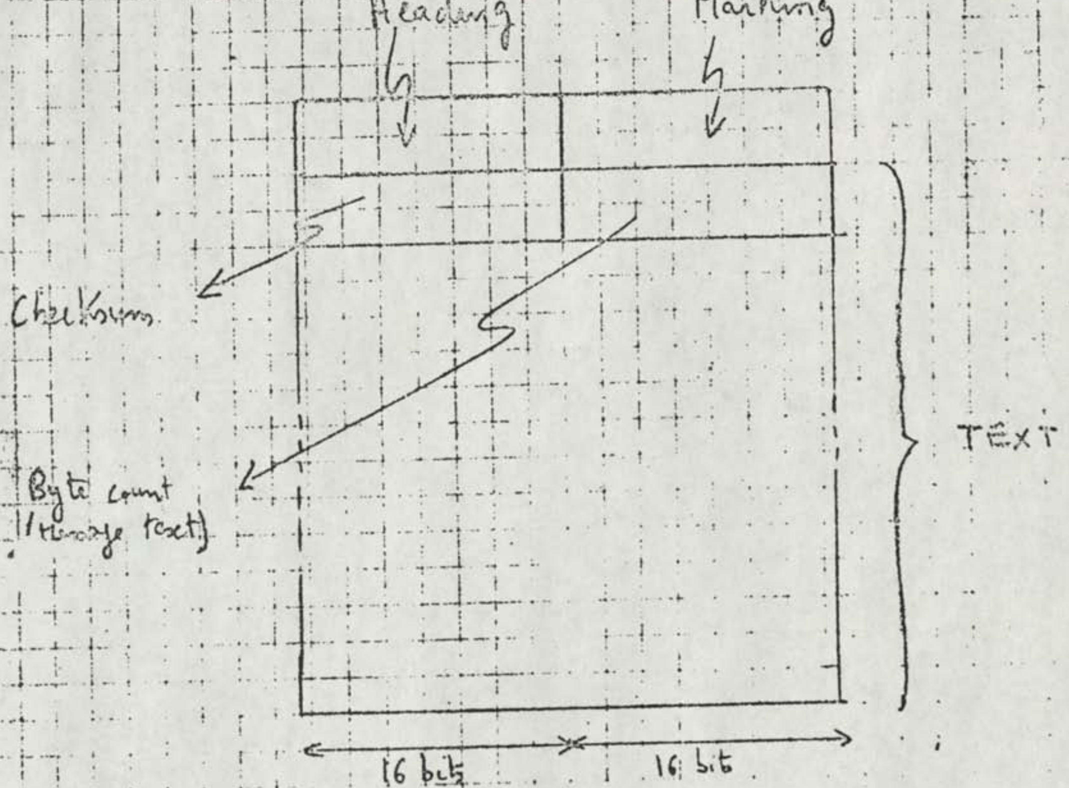
* The UCLA selected program sends a message over the link to the SRI user program. The message requests that SRI transmit to UCLA the 'local control' program which is written in the DEL language

* We compile this program through our . . .(unreadable ?local) compiler

* We turn control of the TTY link . .(unreadable). . . terminal over the just compiled DEL program.

Fig. 2


USER PROGRAM
APPLICATION
LOCAL CONTROL
NETWORK PROGRAM
TERMINAL HANDLER
TRANSM. HANDLER
TRANSMISSION HARDWARE UNIT



(Fig 1) : UCLA HOST message

4695

Title: Host Software

Author: G. Deloche

Installation: University of California at Los Angeles

Date: 1 May 1969

Network Working Group Request for Comment: 9

4695

Network Working Group
Requests for Comments: 9

HOST SOFTWARE

G. Deloche, U.C.L.A.
1 May 1969

TABLE OF CONTENTS

2. HOST-HOST Protocol

2.1 Logical Links (Figure 2)

Any IMP can be viewed as an interface between a local center and the trunk network. Locally, an IMP may serve up to four HOSTs; for each of them it provides 256 logical links to any remote HOST.

However, between an IMP and all the other IMPs no more than 64 links may be in use simultaneously. In other words, a HOST dialoguing with a remote HOST can consider its local IMP as a switching center offering 256 lines to the remote HOST, but only 64 can be activated at a time. (If a local center includes η HOSTs, 64 should be shared amount the η HOSTs).

The 256 logical links connecting two HOSTs can be distinguished as follows:

Link 0 has a special status. It is the control link (connection requests, status report of any kind...).

The 255 others can be used either as primary links, i.e., "teletype like" connections, or as auxiliary links for file transmission.

2.1.1 Primary Links Features

A primary link
* is the first link established for a HOST-HOST transmission.
* is a "TTY-like" connection that is:
  - ASCII characters are transmitted.
  - Echos are generated by the remote HOST.
  - The remote HOST scans for break character.
  - The transmission is slow (less than 20 characters per second).
* is mainly used for transmitting control commands, i.e., for log-in to the remote HOST operating system.
* provides special buffering techniques for slow, short transmission.

2.1.2 Auxiliary Links Features

An auxiliary link
* is used for transmission of large volumes of data.
* is established in parallel to the primary link
* can be established only if the following conditions are fulfilled: user programs, at the two extremities, must both require its opening.
* is used for either binary or character transmission.

1. Introduction

This paper concentrates upon the HOST-HOST dialogue procedure.

Chapter 2 describes the logical links connecting the HOST, and the way data are exchanged over these links.

The emphasis of Chapters 3, 4, and 5 is on software organization and data structure.

Figure 1 highlights the different programs involved in a HOST.

2.2 Link Establishment

2.2.1 General Procedures

Each HOST(X) user will respect the following procedure for communicating with HOST(Y).

(a) Establish a primary link to HOST(Y).
A primary link is established to HOST(Y) through the control link 0. The connection is then in a pre-log-in state, i.e., the remote HOST expects its standard log-in procedures.

(b) Log-in Sequence
Standard ASCII characters are sent/received over the primary link. In that way, the HOST(X) user signs in to remote HOST(Y) by using its standard log-in procedures.

(c) Establish an auxiliary link to HOST(Y)
This establishment must be executed by both extremities. As in (a), this is done by using the control link 0.

(d) Send/Receive Text over Auxiliary link

2.2.2 Example

Figure 3 focuses on the data exchanged over the links during a HOST(X)-HOST(Y) dialogue.
HOST(X) has the network identification 8.
HOST(Y) has the network identification 5.

Notations Used:
* Circled stuffs represent characters, e.g. ENQ
* Parenthesised numbers are used for cross referencing with further explanations, e.g. (2)

Explanations

* (1) and (2) constitute the primary link establishment
-HOST(X) sends the following message over link 0:
"ENQ PRIM 0 1 2 OPT"

ENQ : Enquiry for link establishment (ASCII character)
PRIM : Link type: primary (Special Character)
0 1 2 : Logical link identification number in decimal (3 ASCII characters)
OPT : Options: it is an alphanumerical character, e.g. 9.
Possible options could be: Full Echo, data type...

-HOST(Y) acknowledges by sending back:
"ACK ENQ PRIM 0 1 2 OPT"

ACK : postive acknowledgement (ASCII character)-Link 12 is now established.

ENQ PRIM 0 1 2 OPT : The previous message is returned to the requestor for security purpose.

* (3) and (4) constitute a trivial example of a log-in procedure -See remark 2 below-

*(5): HOST(X), talking to the operating system of HOST(Y), requests for URSA. URSA is supposed to be a user application program in HOST(Y).

*(6) and (7) constitute the auxiliary link establishment. After (5), an auxiliary link should be established. This is done by HOST(X) since it has the higher identification number in network. e.g., 8 against 5.
The procedure is very much like (1) and (2)

*(8): HOST(X) transmits a "file" to URSA. The transmission is done over link 25 which has just been established.

*(9): HOST(Y) answers back with a "file" over link 25.
And the dialogue goes on...

*(10): HOST(X) frees the links he has established
EOT : End of transmission (ASCII character).
002: Number of links wanted to be closed (3 ASCII character)
012025: Link identification number (ASCII characters)

*(11) HOST(Y) acknowledges back as in (2), (7).

Remark 1: The figure 3 doesn't show the heading of each message which are of course transmitted over these links. The characters represented on each line should be viewed inserted in the text zone of a message.

Remark 2: These characters -see (3) and (4)- can either be transmitted one at a time over the line (each character constitutes the text of a message) or be packed before transmission by the user communication program.

In either case, the remote HOST can consider the link as a normal teletype (Searchs breaking characters, provides echos...).

Remark 3: In (2), (7), or (11), HOST(Y) can answer back a negative acknowledgement character NAK instead of ACK. This, for many various reasons such as bad transmission, HOST(X) wants to open a link already established, and so forth. The message could be NAK IND where IND is a character indicating why the previous block has been refused. Upon receiving back such negative acknowledgements, HOST(X) will repeat its message until HOST(Y) accepts it. An emergency procedure will take place if too many successive NAK occur.

3. Network Service Calls

A user program accesses the network facilities (link establishment, data transmission...) through service calls. Under execution, a service call traps to a monitor service routine that interprets and executes the service. Control is then routed back to the user program.

3.1 List of service calls at user's disposal.

(a) Open Primary Link

OPENPRIM(PRIMID,HOSTID, BUFFADDR , INTRPT-CODE,[OPT])

PRIMID: User identification of the primary link.
HOSTID: Remote HOST identification.
BUFFADDR: Buffer address for the incoming messages.
INTRPT-CODE: Code that the network program should give to the user program when he is interrupted because a message has come back.
OPT: Options such as "full echo" (for testing purpose), message required after successful link establishment, etc....

Remark: [ ]: not required.

(b) Open auxiliary link

OPENAUX(AUXID,PRIMID,BUFFADDR,INTRPT-CODE,[OPT])

AUXID: User identification of the auxiliary link.
PRIMID: User identification of a primary link. Refers to an already established primary link.
BUFFADDR, INTRPT-CODE, OPT same meaning as above.

(c) Transmission over link

TRANSLINK (ID, BUFFADDR, N, [OPT])

ID: User link identification. Depending on which type of links we want to transmit, this identification number will be equal to a previously defined AUXID/PRIMID.
BUFFADDR: Data location address for transmission.
N: Data bytes number for transmission.
OPT: Options such as data type (character vs. binary), acknowledgements required (utilization of the auxiliary links in a half duplex mode), trace bit, etc....

(d) Modify link parameters

MODIFLINK (ID, OPT)

ID: User link identification (Equal to either AUXID/PRIMID)

(e) Close link

CLOSE LINK (ID, [OPT])

ID: Same meaning as above.
OPT: Can be used to close all the links in use by the user.

4. Data Structure

The allocation and the management of the links are carried out by means of three tables:

A Table Sorted By HOST.
A Table Sorted By LINK.
A Table Sorted by USER.

4.1 HOST Table (See Figure 4)

It is a bit-table indicating, for a given remote HOST, which links are free. (bit-0 means free link)

This table should provide 256 bits per HOST (256 logical links possible). At a given time no more than 64 bits can be set to 1 in the whole table.

4.2 LINK Table (See Figures 4 and 5)

This table contains as many sections as links in use. Figure 5 describes the structure of a section.
Starting and retrieval are carried out dynamically upon using a hashing technique based on the network link identifications.

4.3 USER Table (See Figure 4)

The table structure is given on Figure 4. These are as many sections as active users. Each section contains the user identification (given by the operating system) and the identifications of the links in use by this user. Notice that a link has two identifications: that of the user (given as a parameter in the OPEN service call) and that of the network (that is attributed by the network program).

This table is hashed by users.

5. Network Program

The emission functions of the network programs are fulfilled by monitor service routines. In that sense, this program can be viewed as belonging to the operating system.

These functions are concerned with the link establishments and data transmission; they are started by the service calls previously described.

Let's explain how these routines allocate and manage the links by describing the operations involved during the execution of the OPENPRIM routine.

Suppose that the value of the parameter HOSTID is equal to j.

(a) j is used as an index for the "HOST" table to reach the "HOST j"

(b) In "HOST j" section, we select the first free link (First bit=0) e.g., i^{th} bit.

(c) j and i determine respectively the HOST-IMP destination and the network link number.

(d) This j-i value is used as a hashing code to open a new section in the link table. e.g. section ℓ.

(e) In this section ℓ, the link ID zone is filled up with j-i, the "link opened by us" and "primary" bits are set to 1. (See Figure 5.)

(Remark: It is only when we receive back the acknowledgement message from the remote HOST - See Figure 3: (2)-that the link is considered completely established. Then we set to 1 the bit "link established".). Also in this section ℓ, we store the parameter BUFFADD Value in the "buffer address zone", and the user identification number, implicitly given, in "the user ID zone".

(f) Using the user identification number, we hash the USER Table to open (or find) the right m section.

We update this m section by storing the user link ID number (PRIMID) and the network link ID number (i).

(g) We prepare the message text:
ENQ PRIM 0 0 i OPT

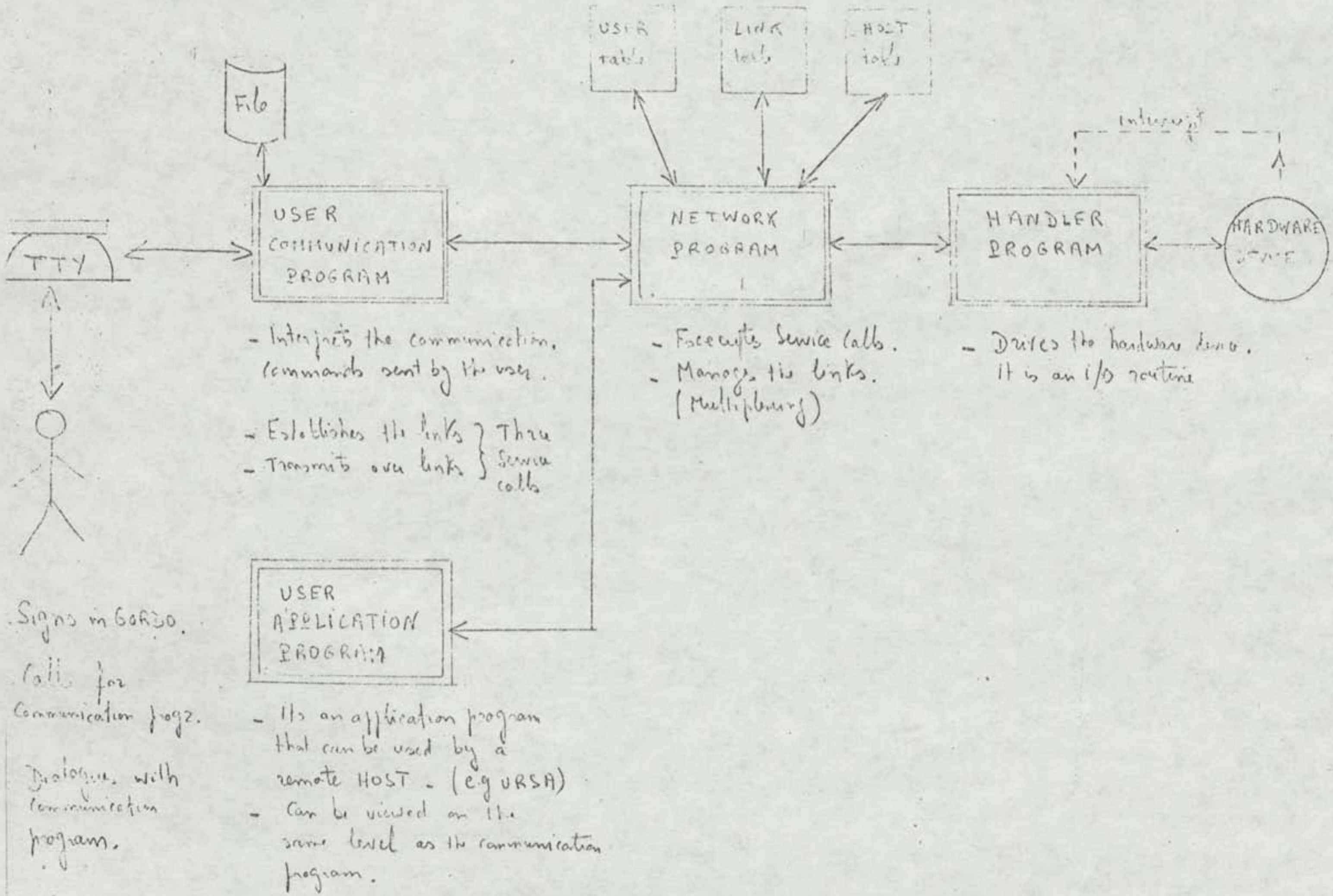
(h) We prepare a heading according to BBN specifications (in order to send the message over link 0).

(i) We calculate the HOST checksum.

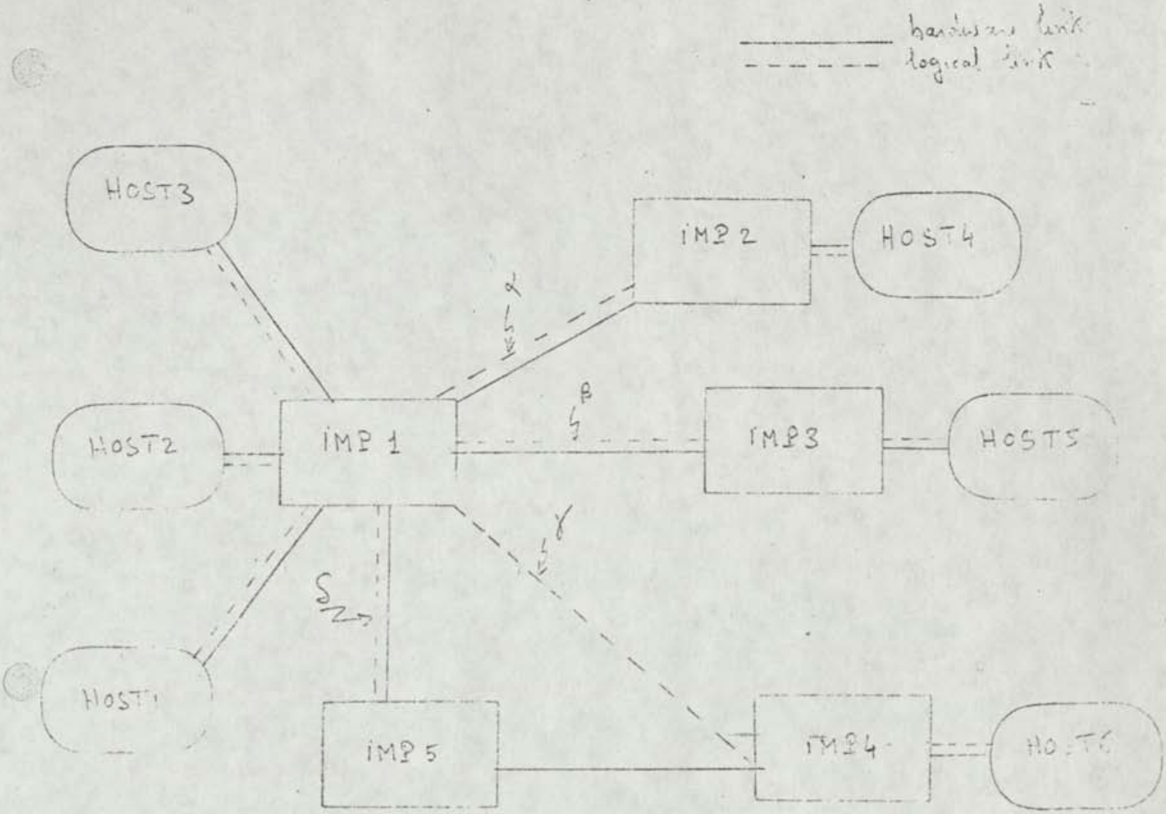
(j) We put together the heading, checksum, text by providing marking.

(k) We queue up this message for the handler.

The receiving functions will use these tables in a very similar way.


File
USER tables
LINK tables
HOST tables
Interrupt
TTY
USER COMMUNICATION PROGRAM
NETWORK PROGRAM
HANDLER PROGRAM
HARDWARE
- Interprets the communication commands sent by the user.
- Establishes the links
- Transmits over links
} Thru Service calls
- Executes Service Calls.
- Manages the links. (Multiplexing)
- Drives the hardware device. It is an i/o routine
Signs in GORDO.
Calls for Communication prog.
Dialogue with communication program.
USER APPLICATION PROGRAM
- It's an application program that can be used by a remote HOST - (e.g URSA)
- Can be viewed on the same level as the communication program.

(Fig 1)



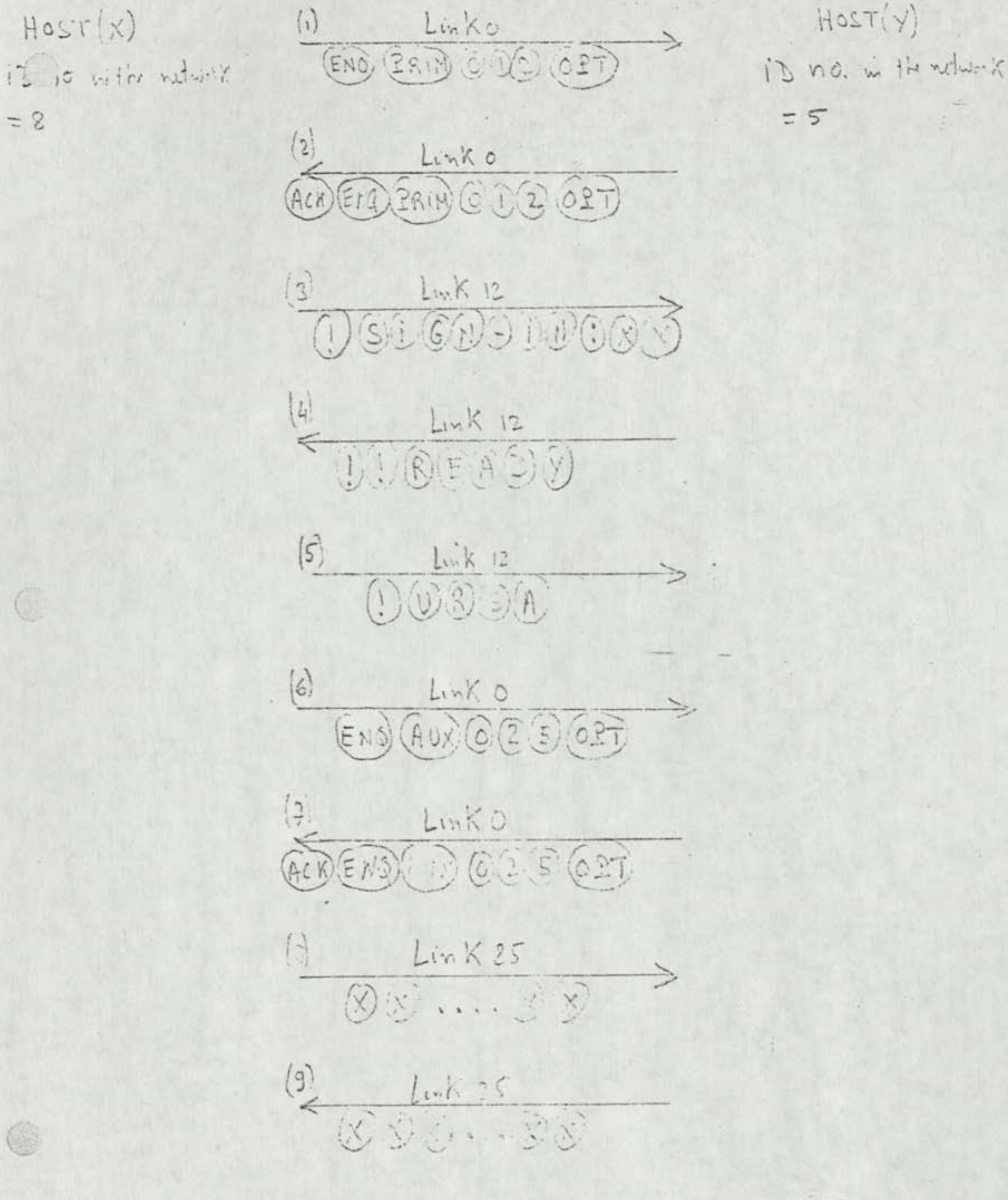
256 logical links between HOST1 and HOST5
256 logical links between HOST1 and HOST6
256 logical links between HOST1 and HOST4
" " " " HOST1 and HOST2

. . . .

256 logical links between HOST2 and HOST4
" " " " HOST2 and HOST6

. . . .

but $\alpha + \beta + \gamma + \varepsilon = 64$ at a given time (Fig 2)


HOST(X)
ID no. within network = 8
HOST(Y)
ID no. in the network = 5
(1) Link 0
ENQ PRIM 0 1 2 OPT
(2) Link 0
ACK ENQ PRIM 0 1 2 OPT
(3) Link 12
! S I G N - I N : X Y
(4) Link 12
! ! R E A D Y
(5) Link 12
! U R = A
(6) Link 0
ENS AUX 0 2 5 OPT
(7) Link 0
ACK ENS 0 2 5 OPT
Link 25
X X .... X X
(9) Link 25
X X .... X X

(Fig 3)

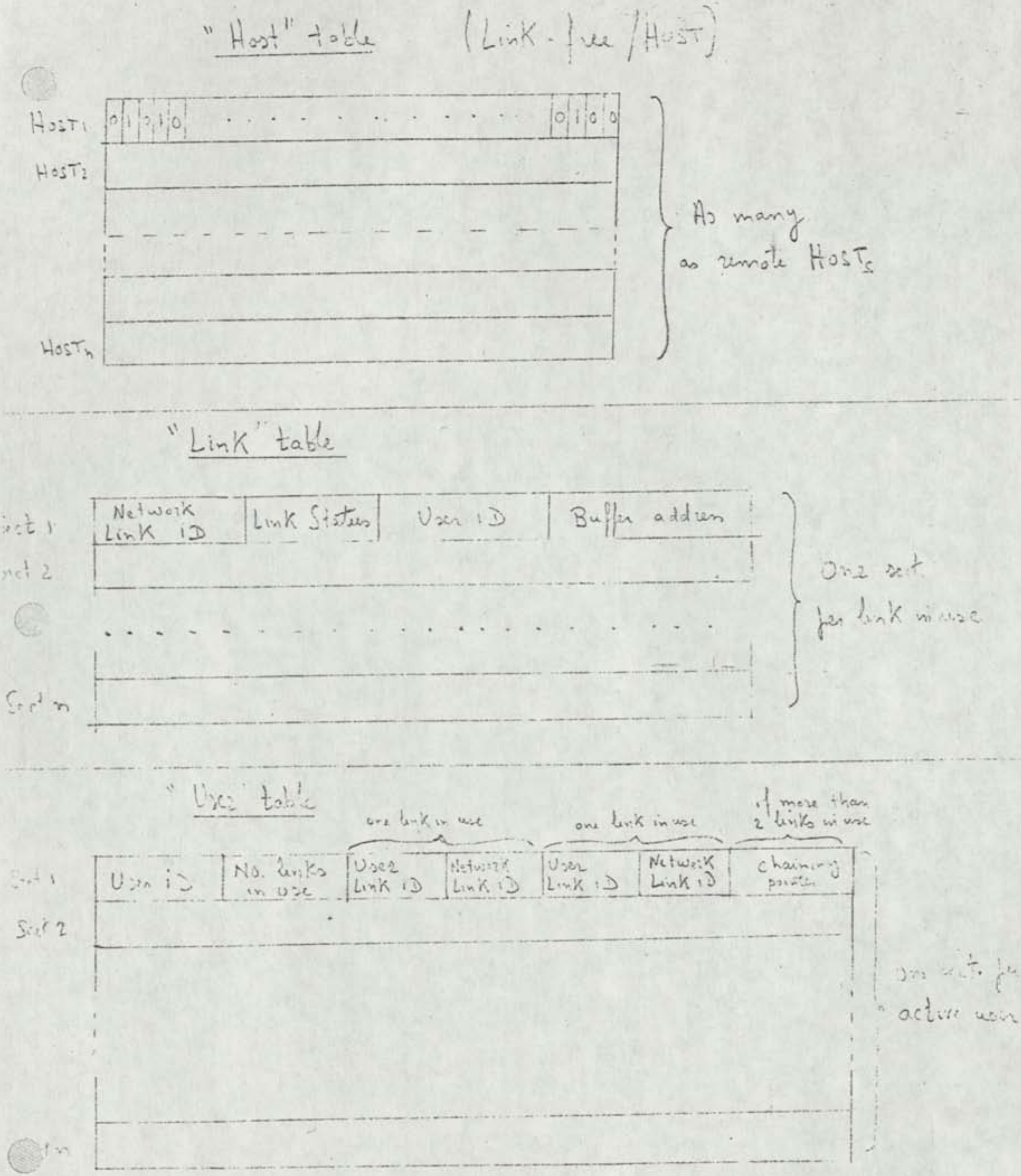
(10) $\xrightarrow{\text{Link 0}}$

EOT 0 0 2 0 1 2 0 2 5

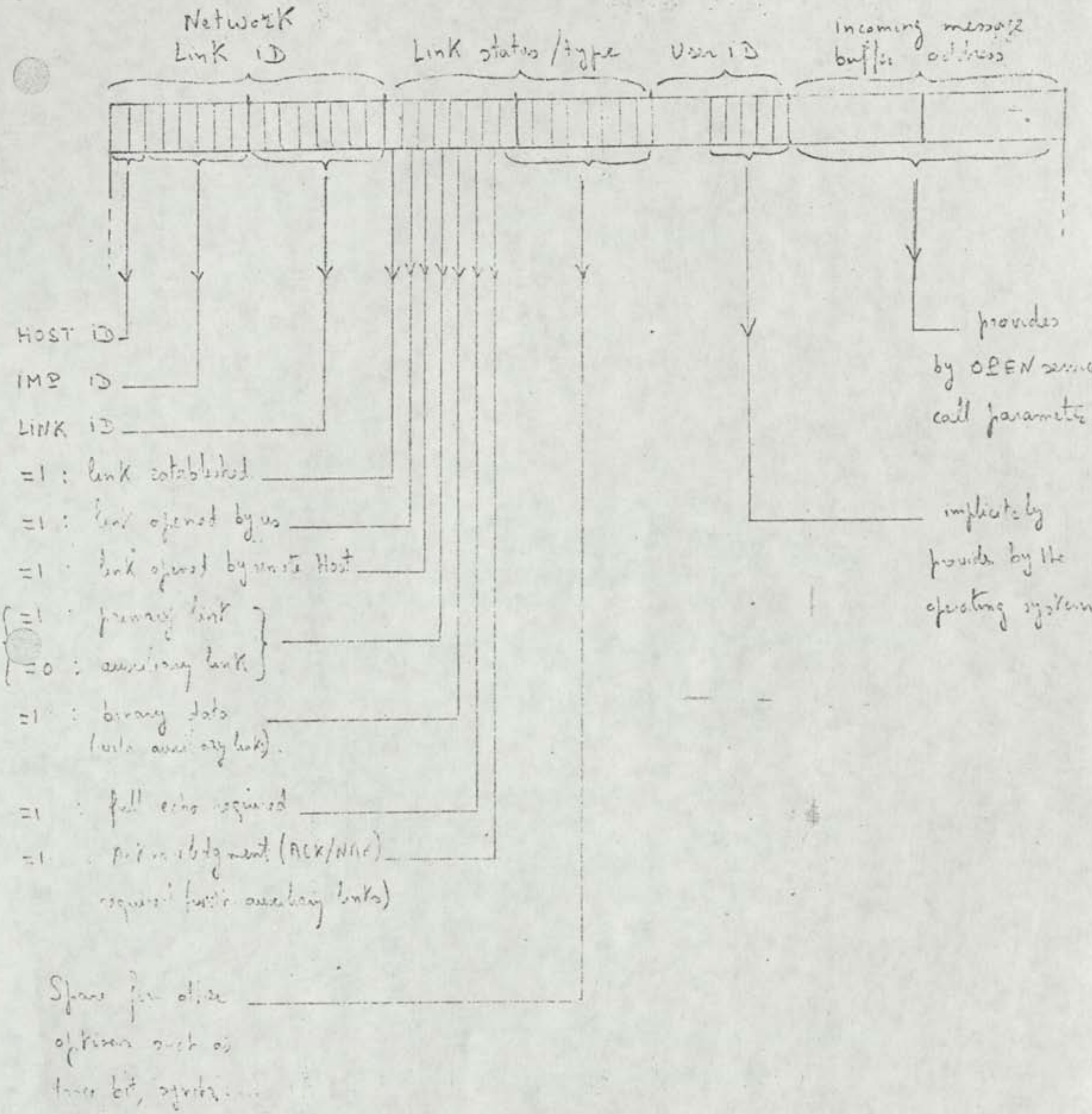
(11) $\xleftarrow{\text{Link 0}}$

ACK EOT 0 0 2 0 1 2 0 2 5

Fig 3)


"Host" table (Link - free / HOST)
HOST1
0 1 1 1 0 . . . . 0 1 0 0
HOST2
HOSTn
As many as remote HOSTs
"Link" table
set 1
set 2
Set n
Network Link ID
Link Status
User ID
Buffer address
One set per link in use
"User" table
Set 1
Set 2
Set n
User ID
No. links in use
one link in use
User Link ID
Network Link ID
one link in use
User Link ID
Network Link ID
if more than 2 links in use
Chaining pointer
one set per active user

(Fig 4)



Link table structure

(Fig 5)