APPENDIX C : S.CROCKER's NWG Note

Network Working Group
Steve Crocker
Request for Comments: 1
UCLA

7

April 1969

Title: Host Software
Author: Steve Crocker
Installation: UCLA
Date: 7 April 1969
Network Working Group Request for Comment: 1

CONTENTS

A Summary of Primitives

Error Checking

Closer Interaction

Open Questions

IV. Initial Experiments

Experiment One

Experiment Two

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 systems 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 as 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.