Host/Host Protocol for the ARPA Network

Abstract

   This document reproduces the Host/Host Protocol developed by the ARPA
   Network Working Group during 1969, 1970, and 1971. It describes a
   protocol used to manage communication between processes residing on
   independent Hosts. It addresses issues of multiplexing multiple
   streams of communication (including addressing, flow control,
   connection establishment/disestablishment, and other signaling) over
   a single hardware interface. It was the official protocol of the
   ARPA Network from January 1972 until the switch to TCP/IP in January
   1983. It is offered as an RFC at this late date to help complete the
   historical record available through the RFC series.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

   The Host/Host Protocol for the ARPA Network was created during 1969,
   1970, and 1971 by the Network Working Group, chaired by Steve
   Crocker, a graduate student at UCLA. Many of the RFCs with numbers
   less than 72, plus RFCs 102, 107, 111, 124, 132, 154, and 179 dealt
   with the development of this protocol. The first official document
   defining the protocol was issued by Crocker on August 3, 1970 as
   "Host-Host Protocol Document No. 1" (see citation in RFC 65), which
   was based on RFC 54 by Crocker, Postel, Newkirk, and Kraley.
   Revision of Document No. 1 began in mid-February 1971, as discussed
   in RFC 102. Although McKenzie is listed as the author of the January
   1972 document, which superseded Document No. 1, it is more correct to
   say McKenzie was the person who compiled and edited the document.
   Most or all of the ideas in the document originated with others.

   At the time "Host-Host Protocol Document No. 1" was issued it was not
   given an RFC number because it was not to be viewed as a "request for
   comments" but as a standard for implementation. It was one of a set
   of such standards maintained as a separate set of documentation by
   the Network Information Center (NIC) at Stanford Research Institute
   (SRI). The January 1972 version (NIC 8246) reproduced here also
   followed that approach. It has been noted by many that all
   subsequent standards were issued as RFCs, and the absence of the
   Host/Host Protocol specification from the RFC series creates a
   curious gap in the historical record. It is to fill that gap that
   this RFC is offered.

   In 1972, most ARPA Network documents, RFCs and others, were prepared
   and distributed in hard copy. The Host/Host Protocol document was
   typed on a typewriter (probably an IBM Selectric), which had
   interchangeable print elements, and used both italic and boldface
   fonts in addition to the regular font. Diagrams were drawn by a
   graphic artist and pasted into the typed document. Since RFCs are
   constrained to use a single typeface, we have tried to indicate
   boldface by the use of either all capitals or by a double underline,
   and to indicate italics by the use of underscores around words in
   place of spaces. The resulting document is a bit more difficult to
   read, but preserves the emphases of the original. Of course, the
   pagination has changed, and we hope we have correctly modified all of
   the page numbers. There were three footnotes in the original
   document and we have moved these into the text, set off by
   indentation and square brackets. A .pdf image of the original
   document can be found at
   http://www.cbi.umn.edu/hostedpublications/pdf/McKenzieNCP1972.pdf.

2. A Few Comments on Nomenclature and Key Concepts

   In the protocol definition, "RFC" is used to mean "Request for
   Connection", which refers to either a "Sender to Receiver" or a
   "Receiver to Sender" request to initiate a connection. In
   retrospect, this seems like an unnecessarily confusing choice of
   terminology.

   At the time this protocol was defined, it was given the
   undistinguished name "Host-Host Protocol." The acronym "NCP" meant
   "Network Control Program" and referred to the code that had to be
   added to the operating system within each host to enable it to
   interact with its Interface Message Processor (IMP) and manage
   multiple connections. Over time, and particularly in the context of
   the change from this protocol to TCP/IP, this protocol was commonly
   called "NCP" and the expansion changed to "Network Control Protocol."

   This protocol was superseded by TCP. In this document, the protocol
   is referred to as a second layer (or "level") protocol, whereas in
   current writings TCP is usually referred to as a layer 4 protocol.
   When this protocol was created, it was expected that over time new
   layers would be created on top of, below, and even in between
   existing layers.

   This protocol used a separate channel (the control link) to manage
   connections. This was abandoned in future protocols.

   In this design, there was no checksum or other form of error control
   except for the RST. There had been in earlier versions, but it was
   removed at the insistence of the IMP designers who argued vigorously
   that the underlying network of IMPs would never lose a packet or
   deliver one with errors. Although the IMP network was generally
   quite reliable, there were instances where the interface between the
   IMP and the host could drop bits, and, of course, experience with
   congestion control as the network was more heavily used made it clear
   that the host layer would have to deal with occasional losses in
   transmission. These changes were built into TCP.

   Uncertainty about timing constraints in the design of protocols is
   evident in this document and remains a source of ambiguity,
   limitation, and error in today's design processes.

3. Host/Host Protocol Document

                        Host/Host Protocol
                             for the
                          ARPA Network

        Prepared for the Network Working Group by
            Alex McKenzie
            BBN
            January 1972

                              PREFACE

   This document specifies a protocol for use in communication between
   Host computers on the ARPA Network. In particular, it provides for
   connection of independent processes in different Hosts, control of
   the flow of data over established connections, and several ancillary
   functions. Although basically self-contained, this document
   specifies only one of several ARPA Network protocols; all protocol
   specifications are collected in the document
   _Current_Network_Protocols,_ NIC #7104.

   This document supersedes NIC #7147 of the same title. Principal
   differences between the documents include:

      - prohibition of spontaneous RET, ERP, and RRP commands
      - a discussion of the problem of unanswered CLS commands (page 16)
      - a discussion of the implications of queueing and not queueing
        RFCs (page 14)
      - the strong recommendation that received ERR commands be logged,
        and some additional ERR specifications.

   In addition to the above, several minor editorial changes have been
   made.

   Although there are many individuals associated with the network who
   are knowledgeable about protocol issues, individuals with questions
   pertaining to Network protocols should initially contact one of the
   following:

      Steve Crocker
      Advanced Research Projects Agency
      1400 Wilson Boulevard
      Arlington, Virginia 22209
      (202) 694-5921 or 5922

      Alex McKenzie
      Bolt Beranek and Newman Inc.
      50 Moulton Street
      Cambridge, Massachusetts 02133
      (617) 491-1350 ext. 441

      Jon Postel
      University of California at Los Angeles
      Computer Science Department
      3732 Boelter Hall
      Los Angeles, California 90024
      (213) 325-2363

TABLE OF CONTENTS

I. INTRODUCTION

The ARPA Network provides a capability for geographically separated
computers, called Hosts, to communicate with each other. The Host
computers typically differ from one another in type, speed, word
length, operating system, etc. Each Host computer is connected into
the network through a local small computer called an _Interface_
_Message_Processor_(IMP)._ The complete network is formed by
interconnecting these IMPs, all of which are virtually identical,
through wideband communications lines supplied by the telephone
company. Each IMP is programmed to store and forward messages to the
neighboring IMPs in the network. During a typical operation, a Host
passes a message to its local IMP; the first 32 bits of this message
include the "network address" of a destination Host. The message is
passed from IMP to IMP through the Network until it finally arrives
at the destination IMP, which in turn passes it along to the
destination Host.

Specifications for the physical and logical message transfer between
a Host and its local IMP are contained in Bolt Beranek and Newman
(BBN) Report No. 1822. These specifications are generally called the
_first_level_protocol_ or Host/IMP Protocol. This protocol is not by
itself, however, sufficient to specify meaningful communication
between processes running in two dissimilar Hosts. Rather, the
processes must have some agreement as to the method of initiating
communication, the interpretation of transmitted data, and so forth.
Although it would be possible for such agreements to be reached by
each pair of Hosts (or processes) interested in communication, a more
general arrangement is desirable in order to minimize the amount of
implementation necessary for Network-wide communication.
Accordingly, the Host organizations formed a Network Working Group
(NWG) to facilitate an exchange of ideas and to formulate additional
specifications for Host-to-Host communications.

The NWG has adopted a "layered" approach to the specification of
communications protocol. The inner layer is the Host/IMP protocol.
The next layer specifies methods of establishing communications
paths, managing buffer space at each end of a communications path,
and providing a method of "interrupting" a communications path. This
protocol, which will be used by all higher-level protocols, is known
as the _second_level_protocol,_ or Host/Host protocol. (It is worth
noting that, although the IMP sub-network provides a capability for
_message_switching,_ the Host/Host protocol is based on the concept
of _line_switching._) Examples of further layers of protocol
currently developed or anticipated include:

1) An _Initial_Connection_Protocol_ (ICP) which provides a convenient
   standard method for several processes to gain simultaneous access
   to some specific process (such as the "logger") at another Host.

2) A _Telecommunication_Network_ (TELNET) protocol which provides for
   the "mapping" of an arbitrary keyboard-printer terminal into a
   Network Virtual Terminal (NVT), to facilitate communication
   between a terminal user at one Host site and a terminal-serving
   process at some other site which "expects" to be connected to a
   (local) terminal logically different from the (remote) terminal
   actually in use. The TELNET protocol specifies use of the ICP to
   establish the communication path between the terminal user and the
   terminal-service process.

3) A _Data_Transfer_ protocol to specify standard methods of
   formatting data for shipment through the network.

4) A _File_Transfer_ protocol to specify methods for reading,
   writing, and updating files stored at a remote Host. The File
   Transfer protocol specifies that the actual transmission of data
   should be performed in accordance with the Data Transfer protocol.

5) A _Graphics_ protocol to specify the means for exchanging graphics
   display information.

6) A _Remote_Job_Service_ (RJS) protocol to specify methods for
   submitting input to, obtaining output from, and exercising control
   over Hosts which provide batch processing facilities.

The remainder of this document describes and specifies the Host/Host,
or second level, protocol as formulated by the Network Working Group.

II. COMMUNICATION CONCEPTS

The IMP sub-network imposes a number of physical restrictions on
communications between Hosts; these restrictions are presented in BBN
Report Number 1822. In particular, the concepts of leaders,
messages, padding, links, and message types are of interest to the
design of Host/Host protocol. The following discussion assumes that
the reader is familiar with these concepts.

Although there is little uniformity among the Hosts in either
hardware or operating systems, the notion of multiprogramming
dominates most of the systems. These Hosts can each concurrently
support several users, with each user running one or more processes.
Many of these processes may want to use the network concurrently, and
thus a fundamental requirement of the Host/Host protocol is to
provide for process-to-process communication over the network. Since
the first level protocol only takes cognizance of Hosts, and since
the several processes in execution within a Host are usually
independent, it is necessary for the second level protocol to provide
a richer addressing structure.

Another factor which influenced the Host/Host protocol design is the
expectation that typical process-to-process communication will be
based, not on a solitary message, but rather upon a sequence of
messages. One example is the sending of a large body of information,
such as a data base, from one process to another. Another example is
an interactive conversation between two processes, with many
exchanges.

These considerations led to the introduction of the notions of
connections, a Network Control Program, a "control link", "control
commands", connection byte size, message headers, and sockets.

A _connection_ is an extension of a link. A connection couples two
processes so that output from one process is input to the other.
Connections are defined to be simplex (i.e., unidirectional), so two
connections are necessary if a pair of processes are to converse in
both directions.

Processes within a Host are envisioned as communicating with the rest
of the network through a _Network_Control_Program_ (NCP), resident in
that Host, which implements the second level protocol. The primary
function of the NCP is to establish connections, break connections,
and control data flow over the connections. We will describe the NCP
as though it were part of the operating system of a Host supporting
multiprogramming, although the actual method of implementing the NCP
may be different in some Hosts.

In order to accomplish its tasks, the NCP of one Host must
communicate with the NCPs of other Hosts. To this end, a particular
link between each pair of Hosts has been designated as the
_control_link._   Messages transmitted over the control link are
called _control_messages_*, and must always be interpreted by an NCP
as a sequence of one or more _control_commands_. For example, one
kind of control command is used to initiate a connection, while
another kind carries notification that a connection has been
terminated.

   [*Note that in BBN Report Number 1822, messages of non-zero type
   are called control messages, and are used to control the flow of
   information between a Host and its IMP. In this document, the
   term "control message" is used for a message of type zero
   transmitted over the control link. The IMPs take no special
   notice of these messages.]

The concept of a message, as used above, is an artifact of the IMP
sub-network; network message boundaries may have little intrinsic
meaning to communicating processes. Accordingly, it has been decided
that the NCP (rather than each transmitting process) should be
responsible for segmenting interprocess communication into network
messages. Therefore, it is a principal of the second level protocol
that no significance may be inferred from message boundaries by a
receiving process. _The_only_exception_to_this_principle_is_in_
_control_messages,_each_of_which_must_contain_an_integral_number_of_
_control_commands._

Since message boundaries are selected by the transmitting NCP, the
receiving NCP must be prepared to concatenate successive messages
from the network into a single (or differently divided) transmission
for delivery to the receiving process. The fact that Hosts have
different word sizes means that a message from the network might end
in the middle of a word at the receiving end, and thus the
concatenation of the next message might require the receiving Host to
carry out extensive bit-shifting. Because bit-shifting is typically
very costly in terms of computer processing time, the protocol
includes the notions of connection byte size and message headers.

As part of the process of establishing a connection, the processes
involved must agree on a _connection_byte_size._ Each message sent
over the connection must then contain an integral number of bytes of
this size. Thus the pair of processes involved in a connection can
choose a mutually convenient byte size, for example, the least common
multiple of their Host word lengths. It is important to note that
the ability to choose a byte size _must_ be available to the
processes involved in the connection; an NCP is prohibited from
imposing an arbitrary byte size on any process running in its own

Host. In particular, an outer layer of protocol may specify a byte
size to be used by that protocol. If some NCP is unable to handle
that byte size, then the outer layer of protocol will not be
implementable on that Host.

The IMP sub-network requires that the first 32 bits of each message
(called the leader) contain addressing information, including
destination Host address and link number. The second level protocol
extends the required information at the beginning of each message to
a total of 72 bits; these 72 bits are called the _message_header._ A
length of 72 bits is chosen since most Hosts either can work
conveniently with 8-bit units of data or have word lengths of 18 or
36 bits; 72 is the least common multiple of these lengths. Thus, the
length chosen for the message header should reduce bit-shifting
problems for many Hosts. In addition to the leader, the message
header includes a field giving the byte size used in the message, a
field giving the number of bytes in the message, and "filler" fields.
The format of the message header is fully described in Section IV.

Another major concern of the second level protocol is a method for
reference to processes in other Hosts. Each Host has some internal
scheme for naming processes, but these various schemes are typically
different and may even be incompatible. Since it is not practical to
impose a common internal process naming scheme, a standard
intermediate name space is used, with a separate portion of the name
space allocated to each Host. Each Host must have the ability to map
internal process identifiers into its portion of this name space.

The elements of the name space are called _sockets._ A socket forms
one end of a connection, and a connection is fully specified by a
pair of sockets. A socket is identified by a Host number and a
32-bit socket number. The same 32-bit number in different Hosts
represents different sockets.

A socket is either a _receive_socket_ or a _send_socket,_ and is so
marked by its low-order bit (0 = receive; 1 = send). This property
is called the socket's _gender._ The sockets at either end of a
connection must be of opposite gender. Except for the gender, second
level protocol places no constraints on the assignment of socket
numbers within a Host.

III. NCP FUNCTIONS

The functions of the NCP are to establish connections, terminate
connections, control flow, transmit interrupts, and respond to test
inquiries. These functions are explained in this section, and
control commands are introduced as needed. In Section IV the formats
of all control commands are presented together.

Connection Establishment
========================

The commands used to establish a connection are STR (sender-to-
receiver) and RTS (receiver- to-sender).

```
        8*           32                  32              8
   +----------------------------------------------------+
   | STR |   send socket  | receive socket | size |
   +----------------------------------------------------+
```
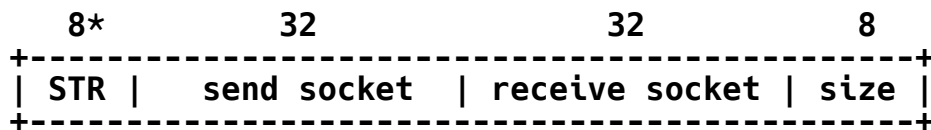
   [*The number shown above each control command field is the length
   of that field in bits.]

```
        8            32                  32              8
   +----------------------------------------------------+
   | RTS | receive socket |   send socket   | link |
   +----------------------------------------------------+
```
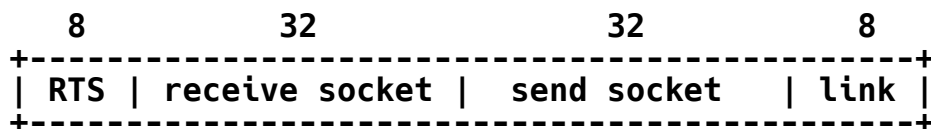
The STR command is sent from a prospective sender to a prospective
receiver, and the RTS from a prospective receiver to a prospective
sender. The send socket field names a socket local to the
prospective sender; the receive socket field names a socket local to
the prospective receiver. In the STR command, the "size" field
contains an unsigned binary number (in the range 1 to 255; zero is
prohibited) specifying the byte size to be used for all messages over
the connection. In the RTS command, the "link" field specifies a
link number; all messages over the connection must be sent over the
link specified by this number. These two commands are referred to as
requests-for-connection (RFCs). An STR and an RTS match if the
receive socket fields match and the send socket fields match. A
connection is established when a matching pair of RFCs have been
exchanged. _Hosts_are_prohibited_from_establishing_more_than_one_
_connection_to_any_local_socket._

With respect to a particular connection, the Host containing the send
socket is called the _sending_Host_ and the Host containing the
receive socket is called the _receiving_Host._ A Host may connect
one of its receive sockets to one of its send sockets, thus becoming
both the sending Host and the receiving Host for that connection.

These terms apply only to data flow; control messages will, in
general, be transmitted in both directions.

A Host sends an RFC either to request a connection, or to accept a
foreign Host's request. Since RFC commands are used both for
requesting and for accepting the establishment of a connection, it is
possible for either of two cooperating processes to initiate
connection establishment. As a consequence, a family of processes
may be created with connection-initiating actions built-in, and the
processes within this family may be started up (in different Hosts)
in arbitrary order provided that appropriate queueing is performed by
the Hosts involved (see below).

_There_is_no_prescribed_lifetime_for_an_RFC._   A Host is permitted to
queue incoming RFCs and withhold a response for an arbitrarily long
time, or, alternatively, to reject requests (see Connection
Termination below) immediately if it does not have a matching RFC
outstanding. It may be reasonable, for example, for an NCP to queue
an RFC that refers to some currently unused socket until a local
process takes control of that socket number and tells the NCP to
accept or reject the request. Of course, the Host which sent the RFC
may be unwilling to wait for an arbitrarily long time, so it may
abort the request. On the other hand, some NCP implementations may
not include any space for queueing RFCs, and thus can be expected to
reject RFCs unless the RFC sequence was initiated locally.

_Queueing_Considerations_

The decision to queue, or not queue, incoming RFCs has important
implications which NCP implementers must not ignore. Each RFC which
is queued, of course, requires a small amount of memory in the Host
doing the queueing. If each incoming RFC is queued until a local
process seizes the local socket and accepts (or rejects) the RFC, but
no local process ever seizes the socket, the RFC must be queued
"forever." Theoretically this could occur infinitely many times
(there is no reason not to queue several RFCs for a single local
socket, letting the local process decide which, if any, to accept)
thus requiring infinite storage for the RFC queue. On the other
hand, if no queueing is performed the cooperating processes described
above will be able to establish a desired connection only by accident
(when they are started up such that one issues its RFC while the RFC
of the other is in transit in the network -- clearly an unlikely
occurrence).

Perhaps the most reasonable solution to the problems posed above is
for _each_ NCP to give processes running in its own Host two options
for attempting to initiate connections. The first option would allow
a process to cause an RFC to be sent to a specified remote socket;

with the NCP notifying the process as to whether the RFC were
accepted or rejected by the remote Host. The second option would
allow a process to tell _its_own_ NCP to "listen" for an RFC to a
specified local socket from some remote socket (the process might
also specify the particular remote socket and/or Host it wishes to
communicate with) and to accept the RFC (i.e., return a matching RFC)
if and when it arrives. Note that this also involves queueing (of
"listen" requests), but it is internal queueing which is susceptible
to reasonable management by the local Host. If this implementation
were available, one of two cooperating processes could "listen" while
the other process caused a series of RFCs to be sent to the
"listening" socket until one was accepted. Thus, no queueing of
incoming RFCs would be required, although it would do no harm.

_It_is_the_intent_of_the_protocol_that_each_NCP_should_provide_
_either_the_"listen"_option_described_above_or_a_SUBSTANTIAL_
_queueing_facility._ This is not, however, an absolute requirement
of the protocol.

Connection Termination
======================

The command used to terminate a connection is CLS (close).

```
      8          32              32
   +-----+-------------+-------------+
   | CLS |  my socket  | your socket |
   +-----+-------------+-------------+
```

The "my socket" field contains the socket local to the sender of the
CLS command. The "your socket" field contains the socket local to
the receiver of the CLS command. _Each_side_must_send_and_receive_a_
_CLS_command_before_connection_termination_is_completed_and_the_
_sockets_are_free_to_participate_in_other_connections._

It is not necessary for a connection to be established (i.e., for
both RFCs to be exchanged) before connection termination begins.
For example, if a Host wishes to refuse a request for connection, it
sends back a CLS instead of a matching RFC. The refusing Host then
waits for the initiating Host to acknowledge the refusal by returning
a CLS. Similarly, if a Host wishes to abort its outstanding request
for a connection, it sends a CLS command. The foreign Host is
obliged to acknowledge the CLS with its own CLS. Note that even
though the connection was never established, CLS commands must be
exchanged before the sockets are free for other use.

After a connection is established, CLS commands sent by the receiver

and sender have slightly different effects. CLS commands sent by the
sender indicate that no more messages will be sent over the
connection. _This_command_must_not_be_sent_if_there_is_a_message_
_in_transit_over_the_connection._ A CLS command sent by the receiver
acts as a demand on the sender to terminate transmission. However,
since there is a delay in getting the CLS command to the sender, the
receiver must expect more input.

A Host should "quickly" acknowledge an incoming CLS so the foreign
Host can purge its tables. However, _there_is_no_prescribed_time_
_period_in_which_a_CLS_must_be_acknowledged._

Because the CLS command is used both to initiate closing, aborting
and refusing a connection, and to acknowledge closing, aborting and
refusing a connection, race conditions can occur. However, they do
not lead to ambiguous or erroneous results, as illustrated in the
following examples.

     EXAMPLE 1: Suppose that Host A sends Host B a request for
     connection, and then A sends a CLS to Host B because it is tired
     of waiting for a reply. However, just when A sends its CLS to B,
     B sends a CLS to A to refuse the connection. A will "believe" B
     is acknowledging the abort, and B will "believe" A is
     acknowledging its refusal, but the outcome will be correct.

     EXAMPLE 2: Suppose that Host A sends Host B an RFC followed by a
     CLS as in example 1. In this case, however, B sends a matching
     RFC to A just when A sends its CLS. Host A may "believe" that the
     RFC is an attempt (on the part of B) to establish a new connection
     or may understand the race condition; in either case it can
     discard the RFC since its socket is not yet free. Host B will
     "believe" that the CLS is breaking an _established_ connection,
     but the outcome is correct since a matching CLS is the required
     response, and both A and B will then terminate the connection.

Every NCP implementation is faced with the problem of what to do if a
matching CLS is not returned "quickly" by a foreign Host (i.e., if
the foreign Host appears to be violating protocol in this respect).
One naive answer is to hold the connection in a partially closed
state "forever" waiting for a matching CLS. There are two
difficulties with this solution. First, the socket involved may be a
"scarce resource" such as the "logger" socket specified by an Initial
Connection Protocol (see NIC # 7101) which the local Host cannot
afford to tie up indefinitely. Second, a partially broken (or
malicious) process in a foreign Host may send an unending stream of
RFCs which the local Host wishes to refuse by sending CLS commands
and waiting for a match. This could, in worst cases, require 2^{32} !
socket pairs to be stored before duplicates began to appear.

Clearly, no Host is prepared to store (or search) this much
information.

A second possibility sometimes suggested is for the Host which is
waiting for matching CLS commands (Host A) to send a RST (see page
20) to the offending Host (Host B), thus allowing all tables to be
reinitialized at both ends. This would be rather unsatisfactory to
any user at Host A who happened to be performing useful work on Host
B via network connections, since these connections would also be
broken by the RST.

Most implementers, recognizing these problems, have adopted some
unofficial timeout period after which they "forget" a connection even
if a matching CLS has not been received. The danger with such an
arrangement is that if a second connection between the same pair of
sockets is later established, and a CLS finally arrives for the first
connection, the second connection is likely to be closed. This
situation can only arise, however, if one Host violates protocol in
two ways; first by failing to respond quickly to an incoming CLS, and
second by permitting establishment of a connection involving a socket
which it believes is already in use. It has been suggested that the
network adopt some standard timeout period, but the NWG has been
unable to arrive at a period which is both short enough to be useful
and long enough to be acceptable to every Host. Timeout periods in
current use seem to range between approximately one minute and
approximately five minutes. _It_must_be_emphasized_that_all_timeout_
_periods,_although_they_are_relatively_common,_reasonably_safe,_and_
_quite_useful,_are_in_violation_of_the_protocol_since_their_use_can_
_lead_to_connection_ambiguities._

Flow Control
============

After a connection is established, the sending Host sends messages
over the agreed-upon link to the receiving Host. The receiving NCP
accepts messages from its IMP and queues them for its various
processes. Since it may happen that the messages arrive faster than
they can be processed, some mechanism is required which permits the
receiving Host to quench the flow from the sending Host.

The flow control mechanism requires the receiving Host to allocate
buffer space for each connection and to notify the sending Host of
how much space is available. The sending Host keeps track of how
much room is available and never sends more data than it believes the
receiving Host can accept.

To implement this mechanism, the sending Host keeps two counters

associated with each connection, a _message_counter_ and a
_bit_counter._ Each counter is initialized to zero when the
connection is established and is increased by allocate (ALL) control
commands sent from the receiving Host as described below. When
sending a message, the NCP of the sending Host subtracts one from the
message counter and the _text_length_ (defined below) from the bit
counter. The sender is prohibited from sending if either counter
would be decremented below zero. The sending Host may also return
all or part of the message or bit space allocation with a return
(RET) command upon receiving a give-back (GVB) command from the
receiving Host (see below).

The _text_length_ of a message is defined as the product of the
connection byte size and the byte count for the message; both of
these quantities appear in the message header. Messages with a zero
byte count, hence a zero text length, are specifically permitted.
Messages with zero text length do not use bit space allocation, but
do use message space allocation. The flow control mechanisms do not
pertain to the control link, since connections are never explicitly
established over this link.

The control command used to increase the sender's bit counter and
message counter is ALL (allocate).

```
        8        8       16              32
     +-----------------------------------+
     | ALL | link | msg space | bit space |
     +-----------------------------------+
```

This command is sent only from the receiving Host to the sending
Host, and is legal only when a connection using the link number
appearing in the "link" field is established. The "msg space" field
and the "bit space" field are defined to be unsigned binary integers
specifying the amounts by which the sender's message counter and bit
counter (respectively) are to be incremented. The receiver is
prohibited from incrementing the sender's counter above $(2^{16} - 1)$,
or the sender's bit counter above $(2^{32} - 1)$. In general, this rule
will require the receiver to maintain counters which are incremented
and decremented according to the same rules as the sender's counters.

The receiving Host may request that the sending Host return all or
part of its current allocation. The control command for this request
is GVB (give-back).

```
        8        8    8    8
     +----------------------+
     | GVB | link | fm | fb |
     +----------------------+
```

This command is sent only from the receiving Host to the sending
Host, and is legal only when a connection using the link number in
the "link" field is established. The fields fm and fb are defined as
the fraction (in 128ths) of the current message space allocation and
bit space allocation (respectively) to be returned. If either of the
fractions is equal to or greater than one, _all_ of the corresponding
allocation must be returned. Fractions are used since, with messages
in transit, the sender and receiver may not agree on the actual
allocation at every point in time.

Upon receiving a GVB command, the sending Host must return _at_
_least_* the requested portions of the message and bit space
allocations. (A sending Host is prohibited from spontaneously
returning portions of the message and bit space allocations.) The
control command for performing this function is RET (return).

    [*In particular, fractional returns must be rounded up, not
    truncated.]

```
       8       8       16           32
    +---------------------------------------+
    | RET | link | msg space | bit space |
    +---------------------------------------+
```

This command is sent only from the sending Host to the receiving
Host, and is legal only when a connection using the link number in
the "link" field is established and a GVB command has been received
from the receiving Host. The "msg space" field and the "bit space"
field are defined as unsigned binary integers specifying the amounts
by which the sender's message counter and bit counter (respectively)
have been decremented due to the RET activity (i.e., the amounts of
message and bit space allocation being returned). NCPs are obliged
to answer a GVB with a RET "quickly"; however, there is _no_
prescribed time period in which the answering RET must be sent.

Some Hosts will allocate only as much space as they can guarantee for
each link. These Hosts will tend to use the GVB command only to
reclaim space which is being filled very slowly or not at all. Other
Hosts will allocate more space than they have, so that they may use
their space more efficiently. Such a Host will then need to use the
GVB command when the input over a particular link comes faster than
it is being processed.

Interrupts
==========

The second level protocol has included a mechanism by which the
transmission over a connection may be "interrupted." The meaning of

the "interrupt" is not defined at this level, but is made available
for use by outer layers of protocol. The interrupt command sent from
the receiving Host to the sending Host is INR (interrupt-by-
receiver).

```
         8        8
    +------------+
    | INR | link |
    +------------+
```

The interrupt command sent from the sending Host to the receiving
Host is INS (interrupt-by-sender).

```
         8        8
    +------------+
    | INS | link |
    +------------+
```

The INR and INS commands are legal only when a connection using the
link number in the "link" field is established.

Test Inquiry
============

It may sometimes be useful for one Host to determine if some other
Host is capable of carrying on network conversations. The control
command to be used for this purpose is ECO (echo).

```
         8        8
    +------------+
    | ECO | data |
    +------------+
```

The "data" field may contain any bit configuration chosen by the Host
sending the ECO. Upon receiving an ECO command an NCP must respond
by returning the data to the sender in an ERP (echo-reply) command.

```
         8        8
    +------------+
    | ERP | data |
    +------------+
```

A Host should "quickly" respond (with an ERP command) to an incoming
ECO command. However, there is no prescribed time period, after the
receipt of an ECO, in which the ERP must be returned. A Host is
prohibited from sending an ERP when no ECO has been received, or from
sending an ECO to a Host while a previous ECO to that Host remains

"unanswered." Any of the following constitute an "answer" to an ECO:
information from the local IMP that the ECO was discarded by the
network (e.g., IMP/Host message type 7 - Destination Dead), ERP, RST,
or RRP (see below).

Reinitialization
================

Occasionally, due to lost control messages, system "crashes", NCP
errors, or other factors, communication between two NCPs will be
disrupted. One possible effect of any such disruption might be that
neither of the involved NCPs could be sure that its stored
information regarding connections with the other Host matched the
information stored by the NCP of the other Host. In this situation,
an NCP may wish to reinitialize its tables and request that the other
Host do likewise; for this purpose the protocol provides the pair of
control commands RST (reset) and RRP (reset-reply).

```
     8
  +-----+
  | RST |
  +-----+

     8
  +-----+
  | RRP |
  +-----+
```

The RST command is to be interpreted by the Host receiving it as a
signal to purge its NCP tables of any entries which arose from
communication with the Host which sent the RST. The Host sending the
RST should likewise purge its NCP tables of any entries which arise
from communication with the Host to which the RST was sent. The Host
receiving the RST should acknowledge receipt by returning an RRP.
_Once_the_first_Host_has_sent_an_RST_to_the_second_Host,_the_first_
_Host_is_not_obliged_to_communicate_with_the_second_Host_(except_for_
_responding_to_RST)_until_the_second_Host_returns_an_RRP._ In fact,
to avoid synchronization errors, the first Host _should_not_
communicate with the second until the RST is answered. Of course, if
the IMP subnetwork returns a "Destination Dead" (type 7) message in
response to the control message containing the RST, an RRP should not
be expected. If both NCPs decide to send RSTs at approximately the
same time, then each Host will receive an RST and each must answer
with an RRP, even though its own RST has not yet been answered.

Some Hosts may choose to "broadcast" RSTs to the entire network when
they "come up." One method of accomplishing this would be to send an

   RST command to each of the 256 possible Host addresses; the IMP
   subnetwork would return a "Destination Dead" (type 7) message for
   each non-existent Host, as well as for each Host actually "dead."
   _However,_no_Host_is_ever_obliged_to_transmit_an_RST_command._

   Hosts are prohibited from sending an RRP when no RST has been
   received. Further, Hosts may send only one RST in a single control
   message and should wait a "reasonable time" before sending another
   RST to the same Host. Under these conditions, a single RRP
   constitutes an "answer" to _all_ RSTs sent to that Host, and any
   other RRPs arriving from that Host should be discarded.

IV.   DECLARATIVE SPECIFICATIONS

Message Format
==============

All Host-to-Host messages (i.e., messages of type zero) shall have a
header 72 bits long consisting of the following fields (see Figure
1):

   Bits 1-32   Leader - The contents of this field must be
                constructed according to the specifications contained
                in BBN Report Number 1822.

   Bits 33-40   Field M1 - Must be zero.

   Bits 41-48   Field S - Connection byte size. This size must be
                identical to the byte size in the STR used in
                establishing the connection. If this message is being
                transmitted over the control link the connection byte
                size must be 8.

   Bits 49-64   Field C - Byte Count. This field specifies the number
                of bytes in the text portion of the message. A zero
                value in the C field is explicitly permitted.

   Bits 65-72   Field M2 - Must be zero.

Following the header, the message shall consist of a text field of C
bytes, where each byte is S bits in length. Following the text there
will be field M3 followed by padding. The M3 field is zero or more
bits long and must be all zero; this field may be used to fill out a
message to a word boundary.

```
|<---------------------------------32 bits-------------------------------->|
|<----8 bits--->|<----8 bits--->|<-----------16 bits----------->|

+------------------------------------------------------------------------+
|                                                                        |
|                              LEADER                                    |
|                                                                        |
+------------------------------------------------------------------------
|                |                |                                      |
|   FIELD M1     |    FIELD S     |             FIELD C                  |
|                |                |                                      |
+----------------+----------------+--------------------------------------+
|                |                ^
|   FIELD M2     |                |
|                |                |
+----------------+                |
|                                 |
|                                 |
|                                 |
|                                 |
|                                 |
|                                TEXT
|                                 |
|                                 |
|                                 |         +----------------------+
|                                 |         |                      |
|                                 |         |       FIELD M3       |
|                                 |         |                      |
|                                 V         |                      |
+---------------------------------+---------+----------------------+
|                                 |
|   10------------------0         | <-------PADDING
|                                 |
+---------------------------------+
```
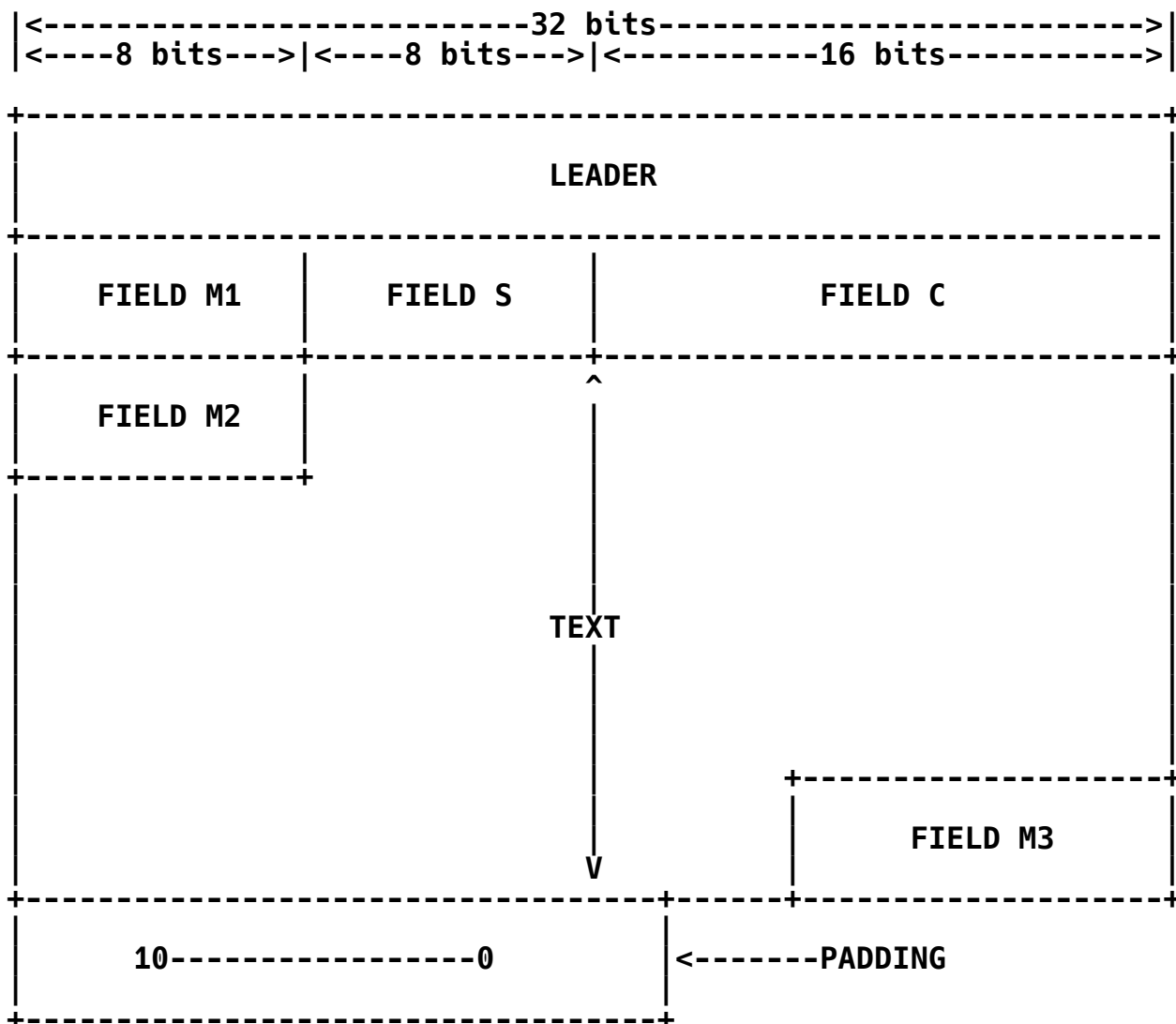
                            Figure 1
                            ========

The message header must, among other things, enable the NCP at the
receiving Host to identify correctly the connection over which the
message was sent. Given a set of messages from Host A to Host B, the
only field in the header under the control of the NCP at Host B is
the link number (assigned via the RTS control command). Therefore,
each NCP must insure that, at a given point in time, for each
connection for which it is the receiver, a unique link is assigned.
Recall that the link is specified by the sender's address and the
link number; thus a unique link number must be assigned to each
connection to a given Host.

Link Assignment

Links are assigned as follows:

Link number	Assignment
==========	==========
0	Control link
2-71	Available for connections
1, 72-190	Reserved - not for current use
191	To be used only for measurement work under the direction of the Network Measurement Center at UCLA
192-255	Available for private experimental use.

Control Messages

Messages sent over the control link have the same format as other
Host-to-Host messages. The connection byte size (Field S in the
message header) must be 8. Control messages may not contain more
than 120 bytes of text; thus the value of the byte count (Field C in
the message header) must be less than or equal to 120.

Control messages must contain an integral number of control commands.
A single control command may not be split into parts which are
transmitted in different control messages.

Control Commands

Each control command begins with an 8-bit _opcode._ These opcodes
have values of 0, 1, ... to permit table lookup upon receipt.
Private experimental protocols should be tested using opcodes of 255,
254, ... Most of the control commands are more fully explained in
Section III.

NOP - No operation
==================

```
        8
     +-----+
     | NOP |
     +-----+
```

The NOP command may be sent at any time and should be discarded by
the receiver. It may be useful for formatting control messages.

RST - Reset
===========

```
        8
     +-----+
     | RST |
     +-----+
```

The RST command is used by one Host to inform another that all
information regarding previously existing connections, including
partially terminated connections, between the two Hosts should be
purged from the NCP tables of the Host receiving the RST. Except for
responding to RSTs, the Host which sent the RST is not obliged to
communicate further with the other Host until an RRP is received in
response.

RRP - Reset reply
=================

```
        8
     +-----+
     | RRP |
     +-----+
```

The RRP command must be sent in reply to an RST command.

RTS - Request connection, receiver to sender
===

```
        8           32              32          8
     +-------------------------------------------------+
     | RTS | receive socket |  send socket   | link |
     +-------------------------------------------------+
```

The RTS command is used to establish a connection and is sent from
the Host containing the receive socket to the Host containing the
send socket. The link number for message transmission over the

connection is assigned with this command; the "link" field must be
between 2 and 71, inclusive.

STR - Request connection, sender to receiver
==

```
       8           32                32           8
    +-----------------------------------------------+
    | STR |   send socket  | receive socket | size |
    +-----------------------------------------------+
```

The STR command is used to establish a connection and is sent from
the Host containing the send socket to the Host containing the
receive socket. The connection byte size is assigned with this
command; the size must be between 1 and 255, inclusive.

CLS - Close
===========

```
       8          32              32
    +-----+-------------+-------------+
    | CLS |  my socket  | your socket |
    +-----+-------------+-------------+
```

The CLS command is used to terminate a connection. A connection need
not be completely established before a CLS is sent.

ALL - Allocate
==============

```
       8     8       16            32
    +----------------------------------+
    | ALL | link | msg space | bit space |
    +----------------------------------+
```

The ALL command is sent from a receiving Host to a sending Host to
increase the sending Host's space counters. This command may be sent
only while the connection is established. The receiving Host is
prohibited from incrementing the Host's message counter above
$(2^{16} - 1)$ or bit counter above $(2^{32} - 1)$.

GVB - Give back
===============

```
        8       8     8     8
    +----------------------+
    | GVB | link | fm | fb |
    +----------------------+
                  ^     ^
                  |     +--- bit fraction
                  +-------- message fraction
```
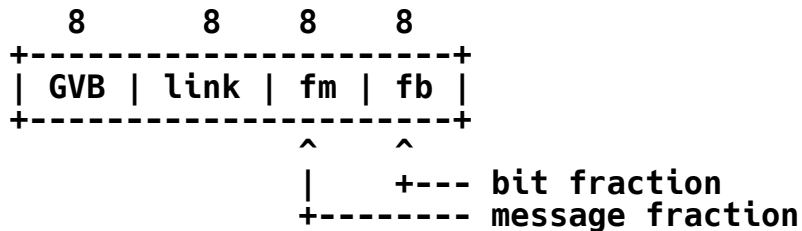
The GVB command is sent from a receiving Host to a sending Host to
request that the sending Host return all or part of its message space
and/or bit space allocations. The "fractions" specify what portion
(in 128ths) of each allocation must be returned. This command may be
sent only while the connection is established.

RET - Return
============

```
        8       8       16              32
    +---------------------------------------+
    | RET | link | msg space | bit space |
    +---------------------------------------+
```
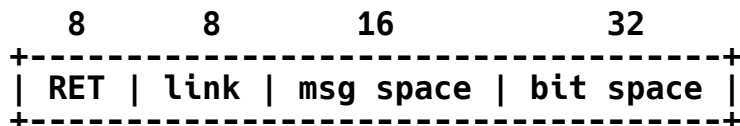
The RET command is sent from the sending Host to the receiving Host
to return all or a part of its message space and/or bit space
allocations in response to a GVB command. This command may be sent
only while the connection is established.

INR - Interrupt by receiver
===========================

```
        8       8
    +------------+
    | INR | link |
    +------------+
```
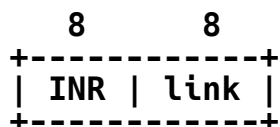
The INR command is sent from the receiving Host to the sending Host
when the receiving process wants to interrupt the sending process.
This command may be sent only while the connection is established.

INS - Interrupt by sender
=========================

```
        8        8
     +------------+
     | INS | link |
     +------------+
```

The INS command is sent from the sending Host to the receiving Host
when the sending process wants to interrupt the receiving process.
This command may be sent only while the connection is established.

ECO - Echo request
==================

```
        8        8
     +------------+
     | ECO | data |
     +------------+
```

The ECO command is used only for test purposes. The data field may
be any bit configuration convenient to the Host sending the ECO
command.

ERP - Echo reply
================

```
        8        8
     +------------+
     | ERP | data |
     +------------+
```

The ERP command must be sent in reply to an ECO command. The data
field must be identical to the data field in the incoming ECO
command.

ERR - Error detected
====================

```
        8        8                           80
     +-----+------+----------------------------- ~ -------------+
     | ERR | code |             data                           |
     +-----+------+----------------------------- ~ -------------+
```

The ERR command may be sent whenever a second level protocol error is
detected in the input from another Host. In the case that the error
condition has a predefined error code, the "code" field specifies the
specific error, and the data field gives parameters. For other

errors the code field is zero and the data field is idiosyncratic to
the sender. Implementers of Network Control Programs are expected to
publish timely information on their ERR commands.

The usefulness of the ERR command is compromised if it is merely
discarded by the receiver. Thus, sites are urged to record incoming
ERRs if possible, and to investigate their cause in conjunction with
the sending site. The following codes are defined. Additional codes
may be defined later.

   a. Undefined (Error code = 0)
      The "data" field is idiosyncratic to the sender.

   b. Illegal opcode (Error code = 1)
      An illegal opcode was detected in a control message. The "data"
      field contains the ten bytes of the control message beginning
      with the byte containing the illegal opcode. If the remainder
      of the control message contains less than ten bytes, fill will
      be necessary; the value of the fill is zeros.

   c. Short parameter space (Error code = 2)
      The end of a control message was encountered before all the
      required parameters of the control command being decoded were
      found. The "data" field contains the command in error; the
      value of any fill necessary is zeros.

   d. Bad parameters (Error code = 3)
      Erroneous parameters were found in a control command. For
      example, two receive or two send sockets in an STR, RTS, or CLS;
      a link number outside the range 2 to 71 (inclusive); an ALL
      containing a space allocation too large. The "data" field
      contains the command in error; the value of any fill necessary
      is zeros.

   e. Request on a non-existent socket (Error code = 4)
      A request other than STR or RTS was made for a socket (or link)
      for which no RFC has been transmitted in either direction. This
      code is meant to indicate to the NCP receiving it that functions
      are being performed out of order. The "data" field contains the
      command in error; the value of any fill necessary is zeros.

   f. Socket (link) not connected (Error code = 5)
      There are two cases:

      1. A control command other than STR or RTS refers to a socket
          (or link) which is not part of an established connection.
          This code would be used when one RFC had been transmitted,
          but the matching RFC had not. It is meant to indicate the

failure of the NCP receiving it to wait for a response to an
RFC. The "data" field contains the command in error; the
value of any fill necessary is zeros.

2. A message was received over a link which is not currently
being used for any connection. The contents of the "data"
field are the message header followed by the first eight bits
of text (if any) or zeros.

Opcode Assignment
=================

Opcodes are defined to be eight-bit unsigned binary numbers. The
values assigned to opcodes are:

     NOP = 0
     RTS = 1
     STR = 2
     CLS = 3
     ALL = 4
     GVB = 5
     RET = 6
     INR = 7
     INS = 8
     ECO = 9
     ERP = 10
     ERR = 11
     RST = 12
     RRP = 13

Control Command Summary
=======================

```
         8
     +-----+
     | NOP |
     +-----+

         8              32                  32              8
     +---------------------------------------------------------+
     | RTS | receive socket |  send socket  | link |
     +---------------------------------------------------------+

         8              32                  32              8
     +---------------------------------------------------------+
     | STR |   send socket  | receive socket | size |
     +---------------------------------------------------------+
```

```
      8           32                32
  +-----+-------------+-------------+
  | CLS |  my socket  | your socket |
  +-----+-------------+-------------+

      8       8        16              32
  +----------------------------------------+
  | ALL | link | msg space | bit space |
  +----------------------------------------+

      8       8     8     8
  +----------------------------+
  | GVB | link | fm | fb |
  +----------------------------+

      8       8        16              32
  +----------------------------------------+
  | RET | link | msg space | bit space |
  +----------------------------------------+

      8       8
  +-------------+
  | INR | link |
  +-------------+

      8       8
  +-------------+
  | INS | link |
  +-------------+

      8       8
  +-------------+
  | ECO | data |
  +-------------+

      8       8
  +-------------+
  | ERP | data |
  +-------------+

      8      8                        80
  +-----+------+--------------------------- ~ ------------+
  | ERR | code |             data                        |
  +-----+------+--------------------------- ~ ------------+
```
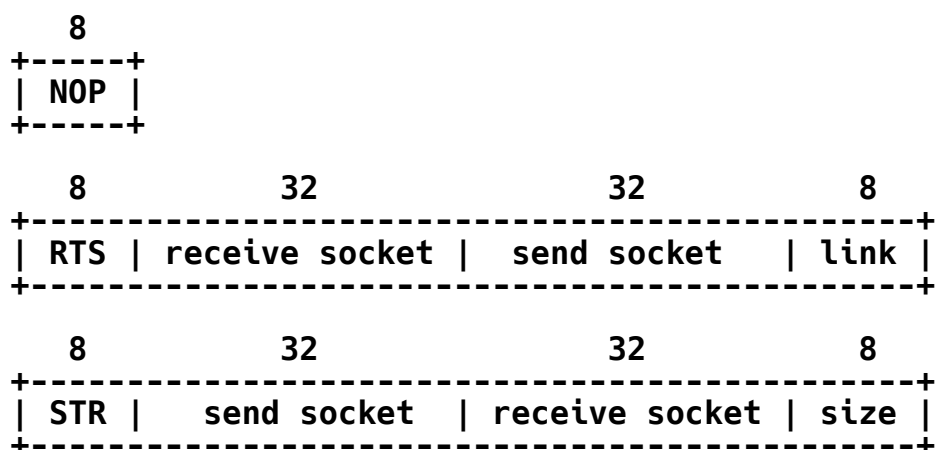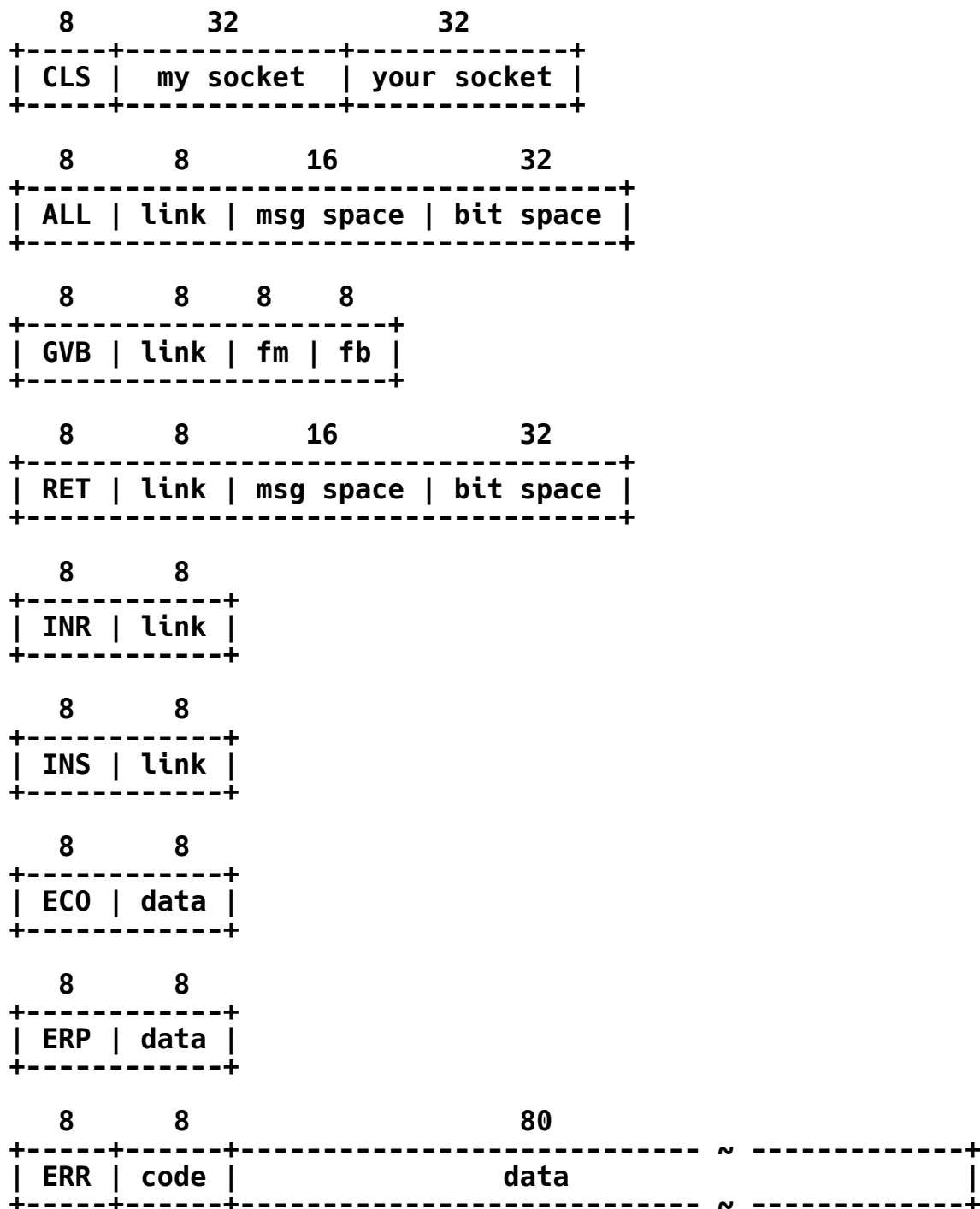
```
        8
     +-----+
     | RST |
     +-----+

        8
     +-----+
     | RRP |
     +-----+
```


     [This is the end of the January 1972 document.]

4. Security Considerations

   This document does not discuss any security considerations.

Authors' Addresses

   Alexander McKenzie
   PMB #4334, PO Box 2428
   Pensacola, FL 32513
   USA
   EMail: amckenzie3@yahoo.com

   Steve Crocker
   5110 Edgemoor Lane
   Bethesda, MD 20814
   USA
   EMail: steve@stevecrocker.com