                    Proposed Change To Host-Host Protocol
                    Resynchronization Of Connection Status

I. Introduction

   The current Host-Host protocol (NIC #8246) contains no provisions for
   resynchronizing the status information kept at the two ends of each
   connection. In particular, if either host suffers a service
   interruption, or if a control message is lost or corrupted in an
   interface or in the subnet, the status information at the two ends of
   the connection will be inconsistent.

   Since the current protocol provides no way to correct this condition,
   the NCP's at the two ends stay "confused" forever. A frequent and
   frustrating symptom of this effect is the "lost allocate" phenomenon,
   where the receiving NCP believes that it has bit and message
   allocations outstanding, while the sending NCP believes that it does
   not have any allocation. As a result, information flow over that
   connection can never be restarted.

   Use of the Host-Host RST (reset) command is inappropriate here, as it
   destroys all connections between the two hosts. What is needed is a
   way to reset only the affected connection without disturbing any
   others.

   A second troublesome symptom of inconsistency in status information
   is the "half-closed" connection: after a service interruption or
   network partitioning, one NCP may believe that a connection is still
   open, while the other believes that the connection is closed. (Does
   not exist.) When such an inconsistency is discovered, the "open" end
   of the connection should be closed.

II. The RCR and RCS Commands

   To achieve resynchronization of allocation, we propose the addition
   of the following two commands to the host-host protocol.

```
        8          8
  +-----------+-----------+
  |   RCS     |   link    |    Reset connection by sender
  +-----------+-----------+


        8          8
  +-----------+-----------+
  |   RCR     |   link    |    Reset connection by receiver
  +-----------+-----------+
```
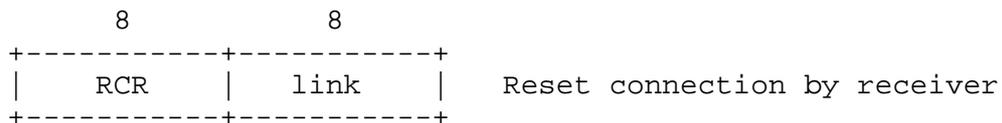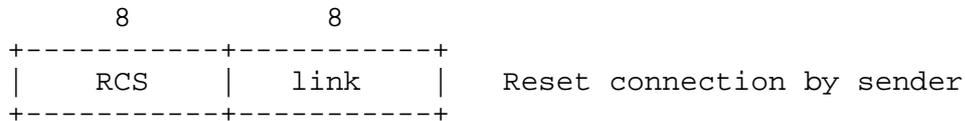
   The RCS command is sent from the host sending on "link" to the host
   receiving on "link". This command may be sent whenever the sending
   host desires to re-synch the status information associated with the
   connection. Some circumstances in which the sending Host may choose
   to do this are:

      1.) After a timeout when there is traffic to move but no
      allocation. (Assumes that an allocation has been lost)

      2.) When an inconsistent event occurs associated with that
      connection (e.g. an outstanding allocation in excess of 2^{32} bits
      or 2^{16} messages.

   The mechanics of re-synchronizing the allocations is simply:

      1.) Empty all messages and allocates from the "pipeline".

      2.) Zero the variables at both ends indicating bit and message
      allocation.

      3.) Restart allocate/message exchanges in the normal way.

   This resynchronization scheme is race-free because the RCS and RCR
   commands are used as a positive acknowledgement pair.

III. Resynchronization by Sender

   To initiate resynchronization, the sending NCP should:

      1.) Put the connection in a "waiting-for-RCR-reply" state. No
      more regular messages may be transmitted over this connection
      until the RCR reply is received.

2.) Wait until the message pipeline is empty, i.e. until a RFNM
has been received for each regular message sent over this
connection. This synchronizes the control and data activity, and
also assures that the data stream will not be corrupted during the
control re-synchronization exchange.

3.) Send the RCS command.

4.) Continue to process allocates normally, updating the variables
which indicate outstanding bit and message allocation.

When the receiving NCP receives the RCS, it should:

1.) Zero the variables indicating outstanding bit and message
allocation.

2.) Reset the connection to the state which indicates readiness to
accept a message.

3.) Confirm the re-synchronization by sending the RCR reply.

4.) Reconsider bit and message allocation, and send an ALL command
for any allocation it cares to do.

When the sending host receives the RCR reply, it should:

1.) Zero the variables indicating outstanding bit and message
allocate.

2.) Put the connection into the "ready-to-send-message" state in
preparation for any forthcoming ALL commands.

At this point, the "pipeline" contains no messages and no allocates,
and the outstanding allocation variables at both ends are in
agreement. (With value zero)

IV. Resynchronization By Receiver

The re-synchronization sequence may be triggered by the receiving
NCP. Such resynchronization could be initiated manually by TIP and
TELNET users who are expecting output but receiving none. Again
assuming that allocation has been lost, the appropriate action is to
reset the connection by sending an RCR command. This action is also
appropriate if an inconsistent event occurs with respect to the
connection. (e.g. arrival of a message which exceeds allocation).

To initiate re-synchronization, the receiving NCP should:

   1.) Put the connection into a "waiting-for-RCS-reply" state. No
   more allocates may be transmitted for this connection until the
   RCS reply is received.

   2.) Send the RCR command.

   3.) Continue to process regular messages normally, updating the
   variables which indicate outstanding bit and message allocation.

When the sending NCP receives the RCR command, it should:

   1.) Wait until the message pipeline is empty, i.e. until the RFNM
   has been received for each regular message sent over the
   connection. This synchronizes the control and data activity, and
   also assures that the data stream will not be corrupted during the
   control re-synchronization exchange.

   2.) Zero the variables indicating outstanding bit and message
   allocation.

   3.) Put the connection into the "ready-to-send-message" state in
   preparation for any forthcoming ALL commands.

   4.) Confirm the re-synchronization by sending the RCS reply.

When the receiving host receives the RCS reply, it should:

   1.) Zero the variables indicating outstanding bit and message
   allocation.

   2.) Reset the connection to the state which indicates readiness to
   accept a message.

   3.) Reconsider bit and message allocation, and send an ALL command
   for any allocation it cares to do.

V. Simultaneous Resynchronization

   This specification for a re-synchronization exchange is guaranteed to
   restore the allocation information at the two ends to a consistent
   state. This happens correctly whether the re-synchronization is
   triggered by the sender, the receiver, or both at the same time.
   When both ends initiate a command at the same time, (the RCS and RCR
   commands cross in the pipeline) each interprets the other's command
   as a confirmation reply; thus, the resynchronization happens
   correctly independent of the relative timing.

The essential factor here is that when either end receives the reset request, it is sure that the other end will take no further actions which could affect the allocation variables. The activity which occurs during simultaneous resynchronization by both ends is as follows:

The sending NCP:

   1. Puts the connection into a "waiting-for-RCR-reply" state. No more regular messages may be transmitted over this connection until the RCR reply is received.

   2. Waits until the message pipeline is empty, i.e. until a RFNM has been received for each regular message sent over this connection. This synchronizes the control and data activity, and also assures that the data stream will not be corrupted during the control re-synchronization exchange.

   3. Sends the RCS command.

   4. Continues to process allocates normally, updating the variables which indicate outstanding bit and message allocation.

Concurrently with 1, 2, 3 and 4 above, the receiving NCP:

   5. Puts the connection into a "waiting-for-RCS-reply" state. No more allocates may be transmitted for this connection until the RCS reply is received.

   6. Sends the RCR command.

   7. Continues to process regular messages normally.

The RCS and RCR commands cross somewhere in the pipeline. When the sender receives the RCR command, it interprets it as a reply to its own RCS command. It then:

   8. Zeroes the variables indicating outstanding bit and message allocation.

   9. Puts the connection into the "ready-to-send-message" state in preparation for any forthcoming ALL commands.

Concurrently with 8 and 9 above, the receiving NCP will receive the RCS command. It will interpret it as a reply to its own RCR command. It then:

10. Zeroes the variables indicating outstanding bit and message allocation.

11. Resets the connection to the state which indicates readiness to accept a message.

12. Reconsiders bit and message allocation, and sends an ALL command for any allocation it cares to do.

VI. The Problem Of Half-closed Connections

The above procedures provide a way to resynchronize a connection after a brief lapse by a communications component, which results in lost messages or allocates for an open connection.

A longer and more severe interruption of communication may result from a partitioning of the subnet or from a service interruption on one of the communicating hosts. It is undesirable to tie up resources indefinitely under such circumstances, so the user is provided with the option of freeing up these resources (including himself) by unilaterally dissolving the connection. Here "unilaterally" means sending the CLS command and closing the connection without receiving the CLS acknowledgement. Note that this is legal only if the subnet indicates that the destination is dead.

When service is restored after such an interruption, the status information at the two ends of the connection is out of synchronization. One end believes that the connection is open, and may proceed to use the connection. The disconnecting end believes that the connection is closed (does not exist), and may proceed to re-initialize communication by opening a new connection (RTS or STR command) using the same local socket.

The re-synchronization needed here is to properly close the open end of the connection when the inconsistency is detected. We propose to accomplish this by changing the semantics of three existing host-host protocol commands.

VII. Redefinition of RTS, STR, ERR (link) to Handle Half-closed Connections

The "missing CLS" situation described above can manifest itself in two ways. The first way involves action taken by the NCP at the "open" end of the connection. It may continue to send regular messages on the link of the half-closed connection, or control messages referencing its link. The NCP at the "closed" end should respond with the ERR message, specifying that the link is unknown. (Error code = 5 does not correspond to an open connection). On

receipt of such an ERR message, the NCP at the "open" end should
close the connection by modifying its tables, (without sending any
CLS command) thereby bringing both ends into agreement.

The second way this inconsistency can show up involves actions
initiated by the NCP at the "closed" end. It may (thinking the
connection is closed) send an STR or RTS to reopen the connection.
The NCP at the "open" end will detect an inconsistency when it
receives such an RTS or STR command, because it specifies the same
foreign socket as an existing open connection. In this case, the NCP
at the "open" end should close the connection (without sending any
CLS command) to bring the two ends into agreement before responding
to the RTS/STR.

VIII. Conclusions

The scheme presented in Section II to resynchronize allocation has
one very important property: the data stream is preserved through the
exchange. Since no data is lost, it is safe to initiate re-
synchronization from either end at any time. When in doubt, re-
synchronize.

The changes in the semantics of RTS, STR, and ERR(code 5) commands
provide the synchronization needed to complete the closing of "half-
closed" connections.

The protocol changes above will make the host-host protocol far more
robust, in that useful work can continue in spite of lapses by the
communications components.

          [This RFC was put into machine readable form for entry]
             [into the online RFC archives by Via Genie 08/00]