RFC 636

NIC 30490

J. Burchfiel  - BBN-TENEX
B. Cosell     - BBN-NET
R. Tomlinson  - BBN-TENEX
D. Walden     - BBN-NET

10 June 1974

# TIP/TENEX Reliability Improvements

During the past months we have felt strong pressure to improve the reliability of TIP/TENEX network connection as improvement in the reliability of users' connections between TENEXs and TIPs would have major impact on the appearance of overall network reliability due to the large number and high visibility of TENEXs and TIPs.  Despite the emphasis on TIP/TENEX interaction, all work done applies equally well to interactions between Hosts of any type.

The remainder of this RFC gives a sketch of our plan for improving the reliability of connections bettween TIPs and TENEXs.  Major portions of this plan have already been implemented (TIP version 322; TENEX version 1.32) and are now undergoing final test prior to release throughout the network.  Completion of the implementation of the plan is expected in the next quarter.

Our plan for improving the reliability of TIP/TENEX connections is concerned with obtaining and maintaining TIP/TENEX connections, gracefully recovering from lost connections, and providing clear messages to the user whenever the state of his connection changes.

When a TIP user attempts to open a connection to any Host, the Host may be down.  In this case it would be helpful to provide the user with information about the extent of the Host's unavailability. To facilitate this, we modified the IMP program to accept and utilize information from a Host about when the Host will be back up and for what reason it is down.  TENEX is to be modified to supply such information before it goes down, or through manual means, after it has gone down.  When the TIP user then attempts to connect to the down TENEX, the IMP local to the TENEX returns the information about why and for how long TENEX will be down.  The TIP is to be modified to report this sort of information to the user; e.g., "Host unavailable because of hardware maintenance -- expected available Tuesday at 16:30 GMT".

The TIP's logger is presently not reentrant.  Thus, no single TIP user can be allowed to tie up the logger for too long at a time; and the TIP therefore enforces a timeout of arbitrary length (about 60 seconds) on logger use. However, a heavily loaded Host cannot be guaranteed always to respond within 60 seconds to a TIP login request, and at present TIP users sometimes cannot get connected to a heavily loaded TENEX. To correct this problem, the TIP logger will be made reentrant and the timeout on logger use will be eliminated.

One notorious soft spot in the Host/Host protocol which degrades the reliability of connections is the Host/Host protocol incremental allocate mechanism. Low frequency software bugs, intermittant hardware bugs, etc., can lead to the incremental allocates associated with a connection getting out of synchronization. When this happens it usually appears to the user as if the connection just "hung up". A slight addiition to the Host/Host protocol to allow connection allocates to be resynchronized has been designed and implemented for both the TIP and TENEX.

TENEX has a number of internal consistency checks (called "bughalts") which occasionally cause TENEX to halt. Frequently, after diagnosis by system personnel, TENEX can be made to proceed without loss from the viewpoint of local users. A mechanism is being provided which allows TENEX to proceed in this case from the point of view of TIP users of TENEX.

The appropriate mechanism entails the following: TENEX will not drop its ready line during a bughalt (from which TENEX can usually proceed successfully), nor will it clear its NCP tables and abort all connections. Instead, after a bughalt TENEX will: discard the message it is currently receiving, as the IMP has returned an Incomplete Transmission to the source for this message; reinitialize the interface to the IMP; and resynchronize, on all connections possible, Host/Host protocol allocate inconsistencies due to lost messages, RFNMs etc. The latter is done with the same mechanism described above. This procedure is not guaranteed to save all data -- a tiny bit may be lost -- but this is of secondary importance to maintaining the connection over the TENEX bughalt.

The TIP user must be kept fully informed as TENEX halts and then continues. Therefore, the TIP has been modified to report "Host not responding -- connection suspended" when it senses that TENEX has halted (it does this by properly interpreting messages returned by the destination IMP). When TENEX resumes service after proceeding from a bughalt, the above procedure notifies the TIP that service is restored, and the TIP has been modified to report "Service resumed" to all users of that Host.

On the other hand, the service interruption may not be proceedable and

TENEX may have to do a total system reload and restart. In this case TENEX will clear its NCP connection tables and send a Host/Host protocol reset command to all other Hosts. On receiving this reset command, the TIP will report "Host reset -- connection closed" to all users of that Host with suspended connections. The TIP user can then re-login to the TENEX or to some other Host.

Of couse, the user may not have the patience to wait for service to resume after a TENEX bughalt. Instead, he may unilaterally choose to connect to some other Host, ignoring the previously suspended connection. If TENEX is then able to proceed, its NCP will still think its connection to the TIP is good and suitable for use. Thus, we have a connection which the TIP thinks is closed and TENEX thinks is open, a phenomenon known as the "half-closed connection". An automatic procedure for cleanly completing the closing of such a connection has been specified and implemented for the TIP and TENEX.

Since TENEX will maintain connections across service interruptions, the TIP user will be required to take the security procedure telling the TIP to "forget" his suspended connection before abandoning his terminal. The command @H 0 (for example) will guarantee that his connection will not be reestablished on resumpption of service. Otherwise, his job would be left at the mercy of anyone who acquires that terminal.

An appendix follows which describes the Host/Host protocol changes made. These changes are backward compatible (with the exception that Hosts which have not implemented these changes will sometimes receive unrecognizable Host/Host protocol commands which they presumably discard without suffering harm). These protocol changes are ad hoc in nature but in light of their backward compatibility and potential utility, ARPA okayed their addition to the TIP and TENEX NCPs without (we believe) any implication that other Hosts have to implement them (although we would encourage their widespread implementation).

## Appendix - Ad Hoc Change to Host-Host Protocol

### A.1 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 NCPs at the two ends stay "confused" forever. An occasional 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 resynchronize 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.

### A.2 The RAR, RAS and RAP commands

To achieve resynchronization of allocation, we add the following three commands to the host-host protocol.

```
      8 bits   8 bits
    -------------------
    !        !        !
 16 !  RAR   !  link  !
    !        !        !
    -------------------
 Reset Allocation by Receiver

      8 bits   8 bits
    -------------------
    !        !        !
```

```
17 !  RAS   !  link  !
   !        !        !
   -------------------
Reset Allocation by Sender

     8 bits   8 bits
   -------------------
   !        !        !
20 !  RAP   !  link  !
   !        !        !
   -------------------
Reset Allocation Please
```

The RAS 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 resynch the status information associated with the connection (and doesn't have a message in transit through the network). 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);

3) After the sending host has suffered an interruption of network service;

4) In response to a RAP (see below).

The RAR command is sent from the Host receiving on "link" to the Host sending on "link" in response to an RAS. It marks the completion of the connection resynchronization. When the RAR is returned the connection is in the known state of having no messages in transit in either direction and the allocations are zero. The receiving Host may then start afresh with a new allocation and normal message transmission can proceed. Since the RAR may be sent ONLY in response to an RAS, there are no races in the resynchronization. All of the initiative lies with the sending Host.

If the receiving Host detects an anomalous situation, however, there is no way to inform the sending Host that a resynchronization is desirable. For this purpose, the RAP command is provided. It constitutes a "suggestion" on the part of the receiving Host that the sending Host resynchronize; the sending Host is free to honor it or not as it sees fit. Since there is no obligatory response to a RAP, the receiving Host may send them as frequently as it chooses and no harm can occur. For example, if a message in excess of the allocate arrives, the receiving Host might send RAPs every few seconds until the sending Host replies with no fears of races if one or more RAPs pass a RAS in the network.

### A.3 Resynchronization Procedure

The resynchronization sequence below may be initiated only by the sender either for internally generated reasons or upon the receipt of a RAP.

a) Sender - decision to resynch

1) Set state to "Wait-for-RAR" (Defer transmission of message.)
2) Wait until no RFNM outstanding
3) Send RAS
4) Zero allocation
5) Ignore allocates until RAR received
6) Set state to "Open" (Resume normal message transmission subject to flow control.)

b) Receiver - receipt of RAS

1) Send RAR
2) Zero allocation
3) Send a new allocation

When the sender is in the "Wait-for-RAR" state it is not permitted to send new regular messages. (Note that steps 4 and 5 will insure this in the normal course of events.) With the return of the RAR the pipeline contains no messages and no allocates, the outstanding allocation variables at both ends are forced into agreement by setting them both to zero. The receiver will then reconsider bit and message allocation, and send an ALL command for any allocation it cares to do.

### A.4 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 ater 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 socket pair or same link.

The resynchronization needed here is to properly close the open end of the connection when the inconsistency is detected.  We will accomplish this by specifying consistency checks and adding a new pair of commands.

## A.5  The NXS and NXR Commands

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 closed end should respond with an NXS if the message referred to a non-existent transmit link (e.g. was an ALL) or NXR if the message referred to a non-existent receive link (e.g. a data message).  On receipt of such an NXS or NXR 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.

```
      8 bits   8 bits
    -------------------
    !        !        !
 21 !  NXR   !  link  !
    !        !        !
    -------------------
 Non-existent Receive Link

      8 bits   8 bits
```

```
            -------------------
            !        !        !
         22 !  NXS   !  link  !
            !        !        !
            -------------------
         Non-existent Send Link
```

### A.6  Consistency Checks

A 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 should detect the inconsistency when it receives such an RTS or STR command, because it specifies the same socket pair as an existing open connection, or, in the case of an RTS, the same link.  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.

### A.7  Conclusion

The scheme presented in Section A.2 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 resynchronization from either end at any time.  When in doubt, resynchronize.

The consistency checks for RTS and STR, and the NXR and NXS commands provide the synchronization needed to complete the closing of "half-closed" connections.

The protocol changes above