Network Working Group
Request for Comments: 492
NIC: 15357
E. Meyer
MIT-Multics
18 April 1973

RESPONSE TO RFC 467

Jerry Burchfiel and Ray Tomlinson of Bolt, Beranek, and Newman, Inc, have issued a Network Request for Comments (#467) which proposes a solution to two problems which have been annoying to Network users. This document will briefly describe the problems and proposed solutions, and offer comments and alternative suggestions.

BACKGROUND

To establish a data connection between two hosts through the network, the Host-Host protocol requires that one host send a Request for Connection and that the second Host reply affirmatively. If the desired socket("port") at the target host is already in use, the target host replies negatively. Once a connection is established, data transmission may proceed, controlled by data allocation messages dispatched by the host at the read end of the connection. The host on the write side is constrained by protocol to send only as much data as has been permitted by the read side. If it exhausts the allocation it must wait until a new data allocation control message is received. Then it can send more.

One of the problems arises from the fact that messages apparently are lost somewhere in the transmission path with a low but regular frequency. If an allocate control message concerning an open connection is lost, a situation can occur in which data transmission over the connection ceases permanently. This can happen because the host at the send side believes it has exhausted its allocation, and sits holding back data to end because it is waiting for a new data allocation message to come from the read side. However, the read side has actually sent out the allocation, but it was lost. It thinks that the send side may proceed and sits waiting for data to come in over the connection. This is known as the "lost allocate" phenomenon. However, similar symptoms can occur if a data message is lost and the send side exhausts its allocation before a new allocation is given by the read side. The send side waits for a new allocation, but the read side has not received one of the data messages and believes there is still some allocation left. In either case, the result is a permanently blocked connection. This appears to happen with enough regularity to be annoying to users who connect typewriters to foreign hosts through the Network. When it happens, the only current solution is to disconnect and to establish a new connection.

The solution to this problem which RFC 467 proposes is to establish a pair of allocation-resetting control messages, one for use by the send side (RCS) and the other for the read side (RCR). Whenever it wishes, either side may initiate the allocation-resetting sequence by setting its own allocation counter to zero and dispatching an RCS or RCR control message to the other side. The host receiving it will set its own allocation counter for that connection to zero and send an RCR or RCS in reply. Now the allocations for both sides are in synchronization (they are zero), and data transmission can begin again when a new allocation is sent by the receive side. This procedure is intended to be initiated whenever either side thinks the connection has been quiescent for a suspiciously long time. The actual specification of this control message pair in RFC 467 is more complex in that the pipeline between the two sides must be empty of data messages before the send side may dispatch an RCS control message.

The second problem arises when the host at one side of an open connection crashes and purges its tables when it comes back up, while the host at the other end of the connection does not notice that anything has happened. (A similar situation occurs when the Network path temporarily fails between the two hosts, but only one host notices the failure and closes the connection.) If the host which crashed attempts to re-establish the connection, the host at the other end refuses to do so because the socket to which the connection request is targeted is seemingly already involved in an open connection. Given the idiosyncrasies of the terminal support software of some systems, users at some consoles may be unable to reconnect to the distant system they were connected with when the local system supporting his terminal crashed. This can continue indefinitely until the system which believes the original connections to be still open resets its internal state. This is call the "half-closed" phenomenon, and a solution is proposed in RFC 467. The basic principle of the RFC 467 proposal is that the side which has the open connection is able to detect an inconsistency whenever either side performs communication regarding this connection. When it does, it is supposed to silently (without regard to normal protocol) close the connection and be ready to handle connection requests to the previously connected port.

There are two types of interactions in which "half-closed" inconsistency is uncovered. The first case occurs when the connected side sends a message over a write connection. The side which has lost the connection receives this as a data message which does not correspond to an open connection and replies with an Error Report control message. When the connected side receives it, it realizes that the connection actually no longer exists and deletes it from its own tables. The second case occurs when the host which has lost the

connection sends a connection request to the other host specifying the same sockets as were involved in the previous connection. The host receiving this request recognizes the inconsistency, because not only is the local socket already connected, it is connected to the same foreign socket as specified in the connection request. It internally deletes its record of the connection, making the local socket free, and responds to the connection request normally.

COMMENTS AND ALTERNATIVE PROPOSALS

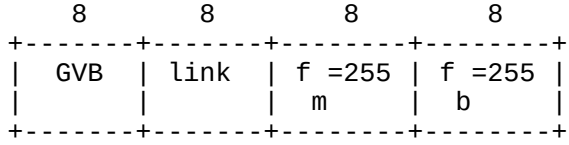
The Project MAC Computer Systems Research Division opposes both protocol change proposals in this RFC. We have moderate opposition to the proposal to handle half-closed connections because it fails to consider all aspects of the problem and it further complicates the protocol, but very strong opposition to the proposal for allocation resynchronization because it attacks a symptom, not the disease, and furthermore tends to mask diagnosis of a potentially very serious network problem.

RFC 467 proposes the addition of two control messages, Reset Connection by Sender (RCS) and Reset Connection by Receiver (RCR) whose sole purpose is to resynchronize the allocation counters at both ends of a connection. In this way the "lost allocate" phenomenon, in which allocate (ALL) control messages somehow are lost in transmission so that the sending side is unable to continue transmitting data is solved. If it were truly a "lost allocate" problem, this would be viable solution. However, I feel that this is really a "lost message" problem, in which messages of all kinds are being lost in transmission, which is much more serious. ALL messages may be very frequent in communications with some hosts and these may be the ones most often lost, but if messages are actually lost in the network, it may also be data messages that are being lost, which would provide similar symptoms. A lost message in a Telnet connection can be detected and overcome by the human user, but an undetected lost message from the middle of a transmitted file can have disastrous consequences, especially because the invalid file, if ever detected, can perhaps not be corrected. Because this "solution" tends to paper over the immediate problem and to propagate it to a point far removed in both space and time at which it appears as an incomprehensible disaster, it should be strongly opposed.

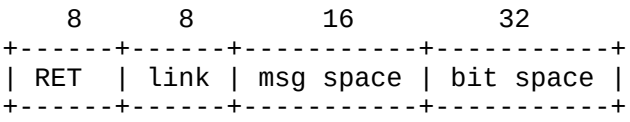
The real problem appears to be the random undetected loss of messages somewhere in the transmission path. A true solution to this problem is either a) to eliminate the cause of undetected loss of messages, or b) to move to a new protocol which is designed to cope with an unreliable physical transmission path. Either of these solutions is

some distance away. A proposed interim solution which modifies the existing GVB and RET commands and which has the additional feature of simplifying them somewhat is outlined below.

A receiving host may at an arbitrary time issue a Give-Back allocation (GVB) control message for a connection.



The format of this GVB message is the same as that currently defined, except that the fraction fields f(m) and f(b) are required to all 1s. This is designed to provide a measure of upward compatibility. A host operating under the modified protocol will ignore the fraction fields, but under the current protocol this message means return everything. A sending host which receives a GVB control message immediately ceases transmission on the specified link. When the RFNM from the last message transmitted is received (indicating an empty pipeline), the sending host issues a Return Allocation (RET) control message, returning the remaining allocation.



The modified RET command has the same format as that currently defined. The two differences are that it can not be sent until data transmission ceases and the last RFNM is received, and that it must return all remaining allocation for the send link (i.e., the allocation counters are set to zero).

When the host on the read side of the connection receives the RET message, the allocation counters at the send side are zero and the pipeline is empty. Therefore, if no error has occurred during the connection, the allocation returned in the RET message should be the same as the allocation in the counters of the read side of the connection. If so, the read side can proceed to send a new allocation secure in the knowledge that no message has been lost. If the two sets of values do not agree, some error in the transmitted data may have occurred. What to do in that case is a local host option. Some hosts may choose to close the connection, while others may choose to resume transmission by sending a new allocation to the

sending side. I feel that as a minimum a host should send a message indicating the error both to the user and to some human being at the host responsible for monitoring network performance.

This modified control message pair is capable of both its originally intended function,and of detecting errors and resynchronizing allocations (if desired) when initiated by the receiving side. I feel that the inability of this scheme to initiate allocation checking from either side is only a minor disadvantage which is more than compensated for by its positive features: this scheme gives positive indication that an error has occurred (the proposed RCS/RCR method conceals errors), and this minor change to the protocol may mean a correspondingly minor change to NCP's.

I have negative feelings regarding the solution to the "half-closed" problem proposed in RFC 467. To put additional burden on the RTS and STR commands not only unduly complicates the protocol, but in some sense can make operation less fail-safe and problems more obscure. My main objection concerns the action to be taken when control messages are received which conflict with the current state of the receiving NCP. This proposal suggests that an NCP receiving an STR or RTS for a socket it believes to be connected assume something about the state of the foreign NCP (that the foreign NCP has closed the connection) and automatically change its own state to agree with the assumed state at the other end (close the connection at its end). This may work fine if the assumption is correct and the implementations are free from bugs. However, the following situations could cause problems that are perhaps hard to diagnose: 1) the foreign NCP has a bug which causes it to send an RTS or STR for a connected socket, 2) the foreign NCP chooses to interpret the queuing option of the current protocol as permitting RFC's to be sent for already connected sockets, or 3) the local NCP has a bug which erroneously causes it to regard RFC's coming from a different host or from the particular foreign host but concerning a different foreign socket as pertaining to the open connection attached to the target socket.

A second objection is that this proposal does not cover all possibilities. Two likely possibilities are: another socket (from any host) attempts to connect to the socket involved in the dead connection. Second, the host that lost a connection attached to one of its read sockets makes another connection with different sockets, but uses the same link number that implemented the previous connection. The second case can be handled by additional complications to the protocol. However, the first case is symptomatically identical to the situation in which an RFC is issued for a genuinely already-connected socket. It can not be handled using this approach.

I believe that a more rigorous use of the existing Reset Host (RST) control message would eliminate most of the causes of the "half-closed" phenomenon; viz. one of the hosts involved in a connection goes down without sending an RST when it comes back up; or the network between the two hosts partitions, and only one host notes it. If it were deemed necessary, a pair of Reset Link control commands to reset an individual link could be added to the protocol to cope with instance of the "half-closed" phenomenon due to other causes.

I'd like to set down here a number of principles which I think are at least peripherally concerned with alleviating the "half-closed" phenomenon. None of these is explicitly stated in the current Host-Host protocol document, but I believe that their enunciation would tend to alleviate confusion caused by network and host failures.

1. A NCP which receives an Imp-to-Host message type 7 (Host Dead) concerning a host should consider all connections or connection attempts with that host as dead and should purge them from its tables.

2. When after noting a foreign host as dead (by receiving a "Host Dead" Imp-to-Host message), an NCP receives any message from that host other than a Reset Host (RST) control message, it should delete the message and respond with an RST. It should respond normally to a received RST.

3. Two hosts must exchange the RST - RRP reset control message pair prior to any other form of communications. An RST must first be sent by an NCP wishing to start communications with a foreign host if that host pair has not been previously reset since the local NCP came up or it noted the foreign NCP as down. Note that this does not require an NCP to send resets to all other hosts each time it comes up.

4. An NCP which receives an Imp-to-Host message type 9 (Incomplete Transmission) concerning a write link implementing an open connection, may at its option make several tries to retransmit the last message until a RFNM is received or the NCP gives up. However, unless the message is eventually successfully transmitted to the foreign host the NCP must abort the connection, sending out a CLS control message. The successful implementation of retransmission depends on the retransmitting host to wait for a RFNM on a data link before sending a subsequent message and on all hosts to be able to discard messages which are not completely received.

5. An NCP which receives a message from a foreign host that seems inconsistent with its current state should take no action to modify that state. Rather it should send an ERR error control message specifying the type of inconsistency and discard the inconsistent message. An NCP receiving an ERR message should log it for human inspection and is then allowed to silently modify its internal state or send out control messages in order to remove the inconsistency. (This is an extension of the proposal in RFC 467 that an NCP should delete a connection when it receives an ERR message specifying that the link involved is unknown.)