Network Working Group R. Kalin
Request for Comments: 60 MIT
Category: Experimental 13 July 1970

# A Simplified NCP Protocol

## Status of this Memo

## Abstract

This RFC defines a new NCP protocol that is simple enough to be implemented on a very small computer, yet can be extended for efficient operation on large timesharing machines. Because worst case storage requirements can be predicted, a conservative implementation can be freed of complicated resource allocation and storage control procedures. A general error recovery procedure is also defined.

## Overview and Rational

The central premise of this proposal is an insistence that all user-to-user connections be bi-directional. For those familiar with communication theory, this appears most reasonable. All communication requires a cyclical flow of information. To deny a simple association between a message and its reply makes protocol unnecessarily complicated and turns simple mechanisms of flow control into nightmares.

It is proposed that a bi-directional connection, or duplex link, be identified by a pair of socket numbers, one for each end. This is half the number presently required. Associated with the connection are some number of "crates" or message containers. These crates travel back and forth over the link carrying network messages from one side to the other. Buffers are allocated at each end of the link to hold crates and the messages that they carry. Worst case buffer requirements are equal to the number of crates in circulation, or the "capacity" of the link.

## Details

A message buffer has four states which follow one another cyclically. They are:

1) empty,

2) filled with a message-laden crate to be unloaded,

3) filled with an empty crate, and

4) filled with a message-laden crate to be sent.

Normally state transitions correspond to message arrival, message removal, message insertion and message transmission.

For a process to be an NCP it must:

1) be able to make initial contact with foreign hosts via the control link and, if necessary, delete user-to-user links left over from the previous system incarnation.

2) be able to create user-to-user links.

3) be able to interface users with these links.

4) be able to delete user-to-user links.

The first of the four functions shall not be discussed here except to point out that it contains critical races that can not be resolved without making assumptions about maximum message propagation delays. Since within the ARPA network, bounds on message turnaround time do not exist, the approach chosen must necessarily be tender. The other three functions are discussed first from the viewpoint of one interested in implementing a minimal NCP. Then extensions and improvements are proposed that are suitable for larger machines.

Any NCP must be capable of creating a duplex link between a local user process and a remote one. The current protocol accomplishes this by queuing a potentially unbounded number of RFC's and waiting for the user to examine the queue to determine with whom he wishes to talk. There is no guarantee that the user will ever look at the queue and there is no way to limit the size of the queue. The overflow error message suggested fails in the respect because it admits that the RFC will only be sent again. The picture need not be this bleak. The following network conversation demonstrates how connections can be made without using queues or relying on user process attention.

Suppose that a local user process and a remote user process wish to establish a new connection. The remote process asks its NCP to listen for a connection request and gives it the socket identifier for its end. Optionally it can give both socket identifiers. The user process at the local end asks its NCP to send a request for a duplex link

(RFDL). It specifies both socket identifiers of the proposed link. The local NCP sends a RFDL over the control link with the following format:

```
RFDL <my socket> <your socket> <max number buffers> <spare>
```

The third argument is normally supplied by the local NCP and indicates the maximum number of buffers the NCP will consider allocating to this duplex link. If buffers are in user storage the count may be given by the user in a call made to the NCP.

The RFDL is received at the remote host and the remote NCP compares <my socket> and <your socket> against the socket identifiers supplied by unmatched listens issued to it. For listens in which just a single identifier was given only <your socket> must match. If both socket identifiers were given, they both must match. If a match is found an acknowledgement message with the following format is sent back by the NCP:

```
ACDL <your socket> <my socket> <number buffers> <spare>
```

The <number buffers> parameter is equal to the smaller of <max number buffers> as specified in the RFDL and the number of message buffers agreeable to the remote NCP. If no match is found the error message returned is an ACDL in which <number buffers> equals zero. Note that the RFDL mechanism is similar to a RFC mechanism in which the bound on queue size is one and connection acceptance is done entirely by the NCP.

The two varieties of a listen correspond to two modes of channel operation. The single parameter variety, as typified by a LOGIN process, is to be used by programs that will "talk with anyone who happens to dial their number". Screening of contacts for appropriateness is left to the user process. The double parameter listen is used by user programs who know with whom they will communicate and do not wish to be bothered by random RFDL’s from other sources. Given the way in which socket name space is partitioned, it is impossible to get a matching RFDL from any process but the one intended.

Message buffers for the connection are allocated in the remote host before it sends the ACDL and in the local host at the time the ACDL is received. The number of buffers at each end is equal to the <number buffers> parameter in the ACDL. The state of all remote buffers is "empty" and of all local buffers "filled with empty crate". After buffers are allocated the local user process is notified that it is able to start sending messages.

The type of interface presented by the NCP between the user process and the newly created duplex link is a decision local to that host. A simple but complete interface would provide two calls to be made to the NCP. GETMESSAGE would return the next message from the link complete with marking, text and padding. PUTMESSAGE would take a message, marking and text only, and buffer it for transmission. The obvious logical errors would be reported.

We suggest that message alignment be left to the user. On most machines it is a simple but time consuming operation. If done in the NCP there is no guarantee that the user will not have to readjust it himself. It is usually not possible to know a priori whether the text portion should be right adjusted to a word boundary, left adjusted to a word boundary, aligned to the end of the last message, or fragmented in some exotic way.

Within this protocol message boundaries are used to provide storage allocation information. If not required by the user this information can be forgotten and the user interface can be made to appear as a bit stream. Though welcomed by purists, such a strategy may produce complications when attempting to synchronize both ends of a link.

Links are deleted by removing empty crates from them and reclaiming the buffers allocated to the crates removed. Only buffers with crates in can be reclaimed; empty buffers must remain available to receive messages that may arrive. When no crates are left, no buffers remain, and the socket identifiers can be forgotten. When empty crates are removed, a decrement size message is sent to the foreign NCP to allow it to reduce its buffer allocation:

DEC <my socket> <your socket> <number of buffers dropped>

A reply is solicited from the foreign NCP to affirm the deletions or to complain of an error. Possible errors include "no such link" and "impossible number of buffers dropped".

The option to close a link can be given to a user process by providing either of two system calls. NOMOREOUTPUT declares that no more messages will be sent by the local user process. All local buffers for the link that contain empty crates are reclaimed by the NCP. DEC messages are sent to the foreign NCP. As crates are emptied, via GETMESSAGE calls, their buffers are reclaimed too. As an alternative, the call KILLMESSAGE can be implemented. This call can be used in place of a PUTMESSAGE. Instead of filling an empty crate with a message to be sent, KILLMESSAGE will cause the crate to be reclaimed and a DEC control message sent.

In situations where the user process has died, or for some other reason can not close the link, more drastic measures must be taken. For these situations, the ABEND control message is defined:

ABEND <my socket> <your socket>

After sending an ABEND the issuing NCP starts to close the link. All buffers containing input are destroyed. A DEC is issued for these and the previously empty buffers. If messages arrive on the link, they are destroyed and a DEC is issued. Any ABEND received for the link is ignored.

When the remote NCP receives the ABEND, it stops sending messages over the link and refuses new messages from the user process at its end. Empty buffers are reclaimed. Pending output messages are destroyed and their buffers reclaimed. Input messages are fed to the user process as long as it will accept them. Buffers are reclaimed as input is accepted. DEC’s are issued to cover all buffers reclaimed. When the user process will take no more input, input messages are destroyed and their buffers reclaimed. Eventually all buffers will be reclaimed at both ends of the link. At such time the connection can be considered closed and the socket numbers used can be reassigned without ambiguity.

Under this proposed protocol the above four functions constitute all that must be part of a network NCP. If buffers are allocated only when free ones exist there can be no "overflow" errors nor is there any need to place further constraints on message flow. For any user message that arrives buffer room is guaranteed. All control messages can be processed without requiring additional storage to be allocated. Attempts by a user process to issue too many listens can be thwarted by local control procedures.

Inefficiencies in storage will result when the number of outstanding connections gets large. One price of coding simplicity is a fifty percent utilization of buffer space. On large hosts it may prove advantageous to implement some of the following NCP extensions. With more complicated flow control procedures, it becomes possible for an NCP to allocate more buffer space than actually exists and still not to get into trouble. Other extensions provide message compression, improved throughput and user transparent error recovery.

Because some extensions require the cooperation of foreign hosts and assume that they have implemented more than the minimal NCP it is proposed that an inquiry control message be used to find out what extensions the foreign host has implemented. The response to an INQ will be a control message defining a host profile. If an "undefined error" message is returned, the foreign host is assumed to have only a minimal NCP.

A simple extension is to define a control message that will replace user RFNM’s. A user RFNM is a null text message sent, for example, as a reply when a file is transferred via a duplex link. They are inefficient since they tie up an entry in the IMP’s link assignment table and degrade network throughput. A more efficient solution is to send a special message over the control link. In this way one short message can replace several user messages.

URFNM <my socket> <your socket> <number of user RFNM’s>

Because the control link is concurrent with the return side of the user link, URFNM’s can not be substituted for user RFNM’s when there are other messages to be sent on the return link. Otherwise ordering will be lost and with it user transparency.

Throughput can also be increased with a mechanism to add additional crates on a duplex link. This might be at user instigation or be a decision of the NCP.

INC <my socket> <your socket> <number buffers desired>

The foreign host replies to an increase request by returning an INCR.

INCR <my socket> <your socket> <number buffers to be added>

If the foreign NCP is unable to meet the additional buffer demand, <number buffers to be added> will be less than <number buffers desired> and possibly zero. The initial state of all local buffers added is "filled with empty crate" and of all foreign buffers "empty".

The spare argument in the RFDL and ACDL could be used to declare the maximum sized message that will actually be sent in that direction. A perceptive NCP could observe this information and allocate smaller buffers. A lesser NCP could ignore it and always assume maximum length messages. For example, if the field were zero then only user RFNM’s would be sent. A smart NCP would allocate no storage at all.

If the NCP retains a copy of each user message sent over the network until a reply is returned, an automatic error recovery procedure can be implemented. Because the capacity of the link is always known, an NCP can determine whether there are messages in transit. This is done by first sending a STOP message to the foreign NCP:

STOP <my socket> <your socket>

The STOP message tells the foreign NCP to temporarily stop transmitting messages over the selected link. Unlike CEASE-ON-LINK there is no guarantee as to how many messages will be sent before the STOP takes effect. The local NCP then sends a link inquiry message:

LINQ <my socket> <your socket>

The reply gives the number of crates at the foreign end of the link. The LINQ message is repeated until this number plus the number of local crates equals the capacity of the link. At this time no messages are in transit and the two ends of the link have been synchronized. Messages can now be identified relative to the synchronization point. Thus the local NCP can send a control message asking, for example, that the third to last message be retransmitted. The foreign NCP is able to identify which message this is and to retransmit it. Once all errors have been recovered a START control message, similar in format to the STOP, is sent to the foreign NCP and normal operation continues. The entire recovery procedure can be transparent to both user processes.

It is expected that the larger hosts will not adhere strictly to the worst case storage allocation requirements. Rather they will allocate more buffers than they have and reply on statistics to keep them out of trouble most of the time. Such conduct is perfectly permissible as long as it is transparent to foreign hosts. The protocol allows an NCP to lie about storage allocation as long as he is not caught. In situations where detection appears imminent some of the following control mechanisms will need to be applied. They are listed in increasing order of power.

a) Do not send out any user RFNM’s or other short messages. There is a good chance that they will be replaced by longer messages that will strain buffer capacity even more.

b) Try not to accept any new messages from the IMP. Block local processes attempting to issue messages.

c) Issue DEC’s to free up buffer space. Do not allocate more than one buffer to RFDL’s and refuse INC’s.

d) Fake errors in messages waiting for local user action. Do this only if the host that sent it has implemented error recovery. This will free buffer space and allow you to recover later. This final measure is admittedly a last resort, but it should be powerful enough to control any emergency.

It is the hope of the author that the above protocol presents an attractive alternative to that proposed by RFC 54 and its additions. Although it appears at a late date, it should not be more than a minor jolt to implementation efforts. It is simple enough to be implemented quickly. If adopted, a majority of the present sites could be talking intelligently with one another by the end of the summer.

## References

[1] Crocker, S.D., Postel, J., Newkirk, J. and Kraley, M., "Official protocol proffering", RFC 54, June 1970.

## Author’s Address

Richard Kalin
MIT Lincoln Laboratory