MEMORANDUM

To: P.M. Wexelblat

From: B.P. Cosell

Subject: Skeleton for an NCP for the PDP-1

Date: 30 June 1972

There are four major constraints governing the design of the NCP: (1) NCP activities may not interfere with non-protocol use of the net; (2) the NCP must be able to handle many connections simultaneously; (3) buggy user programs must be prevented from screwing up the network; and (4) the NCP must allow network initiated connections. This memo blocks out some ideas which could form a basis for such an NCP.

As a starting point, all non-protocol transactions will take place as previously described.* In particular, all such messages from the net will be queued through a block of invariant numbers indexed by *Host* (this is different from Ed's description) and all such messages for the net will pass through a single output queue. For protocol traffic, however, the net handler will do a great deal more work.

Protocol users will never see connections, allocates, or any of that garbage. The net handler will provide a "virtual file system" for such users. In a quiescent system, the user will pass a Host #, a socket #, and "ICP" or "OPEN" to the net handler. The handler will then do whatever is necessary and return a "file number" to the user. At this point data may begin to flow.

*Although it occasionally lies, see Ed Belove's memo on *PDP [illegible] Us[illegible] Access to the Network*.

For "to the net" connections, when the handler determines that the network is ready to accept data, it will notify the user, who may then pass a single item of data to the handler, along with the "file" it is to go out to. The handler will worry about breaking up the message to conform with the allocates it has. When the item is completely sent off, the handler will, again, notify the user to provide more data.

For "from the net" connections, when the user program is prepared to accept data, it will send the handler a READY along with the file number. Upon receipt of a READY, the handler will obtain one message's worth of data from the network and pass it to the user. When the data has been passed to the user, the handler will, again, begin waiting for a READY.

The user may also give the handler a CLOSE with a file number, or a CLOSE ALL. The user will have to handle his own sub-protocol, be it Telnet or data transfer or whatever.

In summary, the primitives that the user will see are:

A) User → handler

1) OPEN
2) ICP
3) CLEAR TO RECEIVE (READY)
4) HERE'S DATA
5) CLOSE
6) CLOSE ALL

B) Handler → user

1) HERE'S YOUR FILE NUMBER
2) CAN'T CONNECT
3) CLEAR TO SEND
4) HERE'S DATA

These mechanisms will, I believe, adequately handle all problems wi
PDP-1 initiated transactions. Network initiated connections are no
very difficult. The handler, when it receives an unsolicited RFC,
will use the socket specified as an index into a block of invariant
numbers. If the invariant number is empty, the RFC will be refused
Otherwise, the program found there will be PSUed, and the RFC will
be saved up. The program, which should know which socket it is
built-in to, issues to the handler an "OPEN SOCKET". The handler
will find the matching RFC and complete opening the connection,
returning the program a file number. At this point, the program
may do as it wishes over its connection as for PDP-1 initiated con-
nections. Note that this allows several "network logins" to be
happening simultaneously.

This rounds out the coarse design, and I would like to digress ont
some implementation considerations. While the handler uses a block
of queues for forwarding general data to users, and individual
channels for forwarding protocol data, users, both general and
protocol, use only one queue for forwarding all data and control in-
formation to the handler. The block of queues should, in effect, a
as a hash table and keep general users out of each others way, but
the single user-to-handler queue will certainly be a problem if it
is kept on the Fastrand. To alleviate this we probably ought to
reinstate some sort of ADDLP and GETLP. This will let users com-
municate to the handler directly, through the Exec. We can implem
either the simple ADDLP and GETLP, requiring the user to set up th
code (ICP, HERE'S DATA, etc.), or we could implement each code as
a separate IOT which would pass the appropriate command and data t
the handler. The latter is nicer if there is enough core and IOT-
address space available. In either case, NETPOK can then be remov
and the equivalent code run through when an ADDLP (or whatever) s
done. This is the way I intended NETPOK to be used, anyway

We could modify SPP so that the net handler is allowed to write into another user's core with SS5 down. This makes the protocol handler-to-user communication easier for everybody. I think this is a good thing to do, and we get the equivalent system integrity by protecting GETNET with SS5, which should have been done in the first place to preserve the network's integrity.

As long as we are putting in new IOTs all over the place, another idea occurs to me. If the net handler were augmented to accept an "OPEN DRUM FILE" command, users could get convenient access to programming system files. The same protocol commands could be used for reading and writing locally to the programming system. Few people need the generality available with my access package, and just this simple addition would help out almost everybody almost all the time.

BPC/nlg

cc R. Alter
 D. Walden