JULY/AUGUST 1981
Volume 9/Number 4

MAJOR NETWORK CHANGE

Roger Moore, Toronto

During May a major change was made in network software: the method used to route data between a terminal and an APL T-task has been altered. The change will allow a larger and a more complex network. Complex interconnection provides more protection for the user against *LINES DOWN,* with more than one route from a terminal to APL. At the same time, protection against network errors caused by very badly delayed packets has been incorporated into the system.

The network topology has already been modified to take advantage of the new software. Links have been installed between Seattle and Vancouver and between San Francisco and Newport Beach. These new links provide extra paths to APL for Californian and western Canadian nodes. The chance of getting a continuing *LINES DOWN* condition is reduced by the new software and the new links.

2005 Preface

The preceding preface was written in 1981 and appears on the front page of the newsletter (below the lead paragraph of an article by Paul Berry describing the June 1981 release of SHARP APL). My article appeared in the Technical Supplement to the newsletter. The newsletter by distributed to employees, customers, competitors and friends. Many of the readers possessed a journeyman knowledge of APL. Therefore I was comfortable using APL expressions in places where mathematic rigour was required. Perhaps I was wrong because no-one ever complained about three minor errors in the APL expressions.

When preparing this paper for republication, I realized that many potential readers are not familiar with the APL language. To that end I have prepared an Appendix which attempts to explain the APL expressions used in the paper. It also notes the incorrect expressions and supplies more rigourous versions. I have tried to avoid correcting any errors in the 1981 paper with one exception. The abbreviated network diagram has been enhanced to indicate the nodes which are loop members.

Roger D Moore
Etobicoke, Canada
July 2005

NETWORK
JULY/AUGUST 1981

technical supplement 33

NETWORK SOFTWARE CHANGES

Roger Moore, Toronto

A node has from one to six links which connect it to the rest of the network. Every node which has more than one network link requires some routing protocol to dispose of packets addressed to other nodes. A routing protocol takes as input some address information from the packet and some tables stored in the node. These tables may be fixed or may change as a result of commands or advice from other nodes in the network.

Both the old and the new routing protocol start by numbering the links of a node from one to six. The usual output of the routing algorithm is a link number along which the packet can be transported to some other node. Other possible outputs are an indication that the packet is destined for this node, or that there is no link which can be used to forward the packet. The old routing algorithm used a fixed length vector which had one entry for every node number in the network. The 16-bit address field of a packet was decoded by NODELINE← 256 256 ⊤ ADDRESS. The expression RT[NODELINE[0]] gave a link number for the destination node. For packets such that OWNNODE=NODELINE[0], NODELINE[1] addressed a particular terminal (or T-task) attached to the node.

The original network routing protocol was designed for a network with star connectivity. In a star network there is only one path between any pair of nodes. If some link in that path fails, the two nodes cannot communicate. In June 1977 the restriction was slightly relaxed to tolerate simple loops. With considerable awkwardness, traffic flow could be routed to avoid a failed link. This was a little cumbersome as the software imposed some consistency requirements on routes. The route from node A to node B can be represented as a vector of node numbers A2B.

To be meaningful this vector must have the following properties: A=1↑A2B; B=¯1↑A2B and all rows of A2B[(¯1↓⍳⍴A2B) ∘.+0 1] must represent physical links in the network. The old routing protocol imposed two additional constraints:

1] If traffic from A to B was using route A2B then
traffic from B to A was required to use the route ⌽A2B.

2] If traffic from A to B was using route A2B and C∈A2B held, then traffic from A to C had to use route (A2B⍳C)↑A2B.

These constraints required some care calculating and altering the routing vectors of ninety network nodes. An APL workspace was developed to maintain a matrix with the route tables for all nodes. The workspace expanded a manually specified loop solution and performed the obvious calculation on the star connected portions of network. One minor motivation in eliminating this routing system was that a WORKSPACE FULL appeared in May when the function attempted make a third copy of the matrix with rho 126 126.

The current routing protocol uses a scheme which is biased towards timesharing applications. Communication within the network is assumed to be between two points. The most common example is a terminal which is logically connected to a single APL T-task. (This T-task may be sharing variables with other tasks but this is beyond the scope of the network.) This logical connection between a terminal and an APL T-task is referred to as a 'virtual call'. The name is derived by analogy with the public telephone network which usually establishes a physical connection between two telephones. Every call in the network is signed a unique number in the zero to 32767 range. Packets are routed by call number rather than by destination node number. The original bit address field is decoded by `32768 2 ⊤ ADDRESS`. The first element is the number and the second element is the direction of travel (towards the zero or one end). The routing algorithm is:

```
ORGDEST←VCT[VCT[;2]⍳⌊ADDRESS÷2;0 1=2|ADDRESS]]
```

If the packet is legitimate, no index error will result and the packet will have been received from `ORGDEST[0]`. An illegitimate packet is discarded (with a warning message to the network logging system). A legitimate packet is forwarded to `ORGDEST[1]`. `ORGDEST[1]` when `≤6` is assumed be the number of a network link within the node. When `ORGDEST[1]≥8`, the packet is destined for the current node. `ORGDEST[1]-8` is the line number `256|(2 ⎕WS 3)[9+⎕IO]` within the node (assuming destination is a terminal). A node begins operation with `0 3←→ρVCT`. As calls are established from, to or through the node, rows are added to the `VCT` matrix. Call termination via `)OFF`, `LINES DOWN` etc. will remove the row from the matrix. Termination of a call within a node involves waiting until all data packets for that call to or from the adjacent nodes have been destroyed. This avoids a problem with the original routing protocol where a packet from an old call could be delayed by retransmission and later inserted into a subsequent call.

The current protocol has different consistency requirements than the original protocol. The representation assures that the route from the one end to the zero end is exactly the reverse of the zero to one route. Calls between a particular pair of nodes may follow different paths. This is a result of abandoning node number as a packet address.

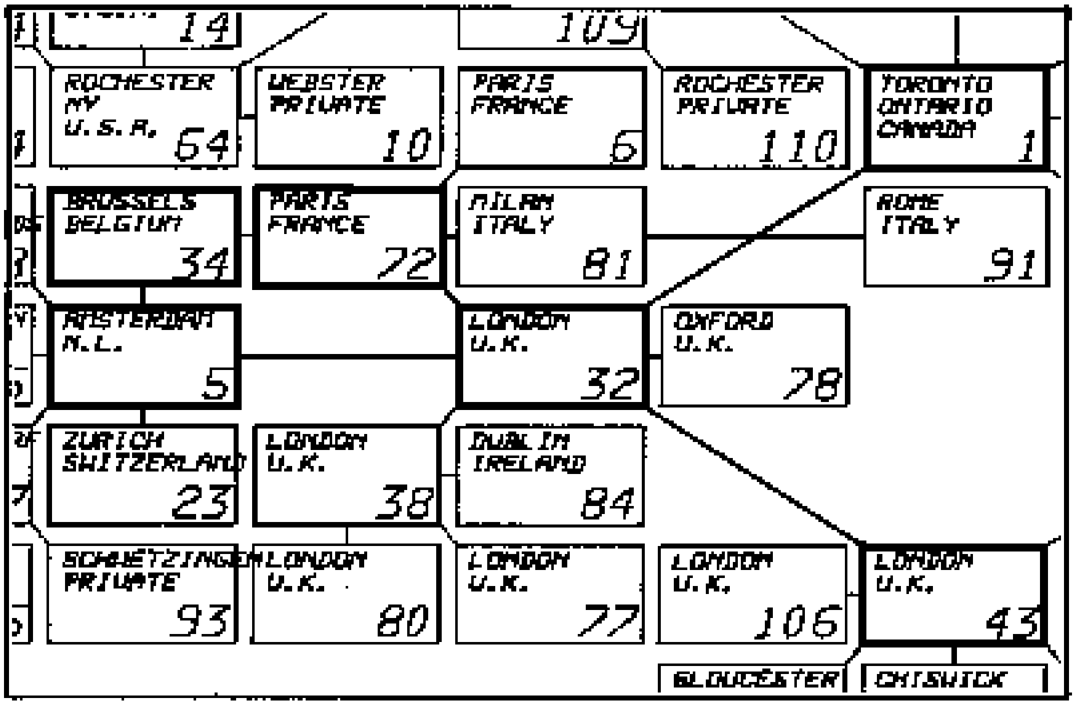
The important requirement introduced by the current protocol is the uniqueness of call numbers along a route. This is enforced by two methods. The first method is to avoid:

```
VCT←VCT,[0] ZERODIRECTION,ONEDIRECTION,NEWCALLNUM
```

when `NEWCALLNUM∊VCT[;2]` holds. The second method is to delay the deletion of a `VCT` row until the adjacent nodes in the zero and one directions have indicated that they have stopped forwarding data packets with the call number being deleted.

New Call Setup

Call setup in the current protocol is adaptive. A broadcast call setup packet is created in the originating node. This call setup packet gives some information about the terminal which originated the call and names the destination node. The broadcast call setup packet is sent across all of the operational links from the originating node. A node receiving a broadcast call setup packet checks to see if it is the destination node. If not, the call setup packet is again forwarded down all network links from the receiving node except the link from which it was received. If the broadcast call setup packet is received by the destination node, the destination node delays for half a second and then accepts the incoming call. It is possible that the destination node or some intermediate node has received more than one copy of the original call setup packet. This may indicate that there is more than one route from the originating node to the destination node. The choice between multiple routes is made by assigning a weight to every link in the network. For two routes `R1` and `R2` if `+/WEIGHTS[R1] < +/WEIGHTS[R2]` holds then `R1` will be selected. For the case where the weight sums are equal, the route which is seen first will be selected. The delay at the destination node is to allow time for a low weighted route to be selected even if the setup packet is slightly delayed. The scheme requires that the weights be >0. Thus the sum around a loop will be greater than zero. For example (see below) if an attempt is made to set up a route from node 43 to node 38, node 32 will receive setup packets from nodes 43 and 1. It will forward copies to nodes 72 and 5 as well as 38. Routes involving the 32 5 34 72 32 loop will have a greater weight sum than those which exclude it and so bubble routes are rejected.



`LINES DOWN` is generated after trying every link in the network which is accessible from the originating node. Any endnode which is not the destination node rejects the call setup packet by sending a negative reply back up the link from which it received the setup packet. When a simple fanin node (such as 16 in the network diagram) receives negative replies from all its endnodes, it can send a negative reply up towards node 5 (assuming the call setup packet originally passed from 5 to 16). When a call setup packet is rejected by a loop member node because the link weight sum is not a new minimum, the loop member sends a negative reply along the link from which it received the rejected call setup packet. Failure of a link while a reply to call setup is expected implies a negative reply to any call setups along that link. Assuming all nodes which receive the call setup packet reject it within a finite time, the *LINES DOWN* message will eventually appear on the user's terminal. (see page 15)

Future Extensions

Some future extensions to the routing protocol are being considered. It is theoretically possible to change the route of an established call. This involves establishing *VCT* entries along the new route and eventually destroying the *VCT* entries in the old route. The operation must be performed in a consistent sequence to avoid loss of data. The usual reason for doing this would be in response to failure of a network link. Another reason might be to balance the load on the links of the network.

A somewhat simpler extension would be a change in the call setup method. The present APL system is served by two network nodes. Under normal circumstances, a T-task user would prefer to use the node which gives the best response. If the node which would normally give the best response is responding with some rude message such as *APL PROBABLY DOWN,* automatic selection of the alternate node would be desirable.

APPENDIX

This Appendix expounds on the meaning of some APL expressions which occur in the paper.

X=Y comparison (like C or JAVA ==)

VARIABLE ← <expression> assignment

```
NODELINE← 256 256 ⊤ ADDRESS
32768 2 ⊤ ADDRESS
```

Representation is used twice to decompose an integer scalar into a multi-element integer vector. The scalar right argument is divided by the last element of the right argument to obtain a remainder and quotient. The remainder becomes the last element of the result; the quotient is the dividend for the next divide. The quotient from the final division is discarded. Thus in the second example, the last element of the result is the odd/even bit of the argument and the first element is the fifteen bit number obtained after dividing the argument by two.

```
A=1↑A2B    B=¯1↑A2B
```

Uparrow takes elements from the beginning or end of a vector. Thus `1↑A2B` extracts the first element; `¯1↑A2B` extracts the final element.

```
A2B[(¯1↓⍳⍴A2B) ∘.+0 1]
```

`⍳⍴A2B` generates an index vector to select every element of *A2B*

`¯1↓X` discards the last element of `X`
`X∘.+Y` Cartesian product generates a matrix giving the sum of every element of `X` with every element of `Y`
`A2B[(¯1↓⍳⍴A2B) ∘.+0 1]` generates a matrix such that every row represent a pair of adjacent elements from `A2B`

`⌽A2B` reversal: `⌽1 2 3` is 3 2 1

`ARGUMENT ∊ TABLE` set membership returns 1 if `ARGUMENT` is a member of `TABLE`

`(A2B⍳C)↑A2B` extracts a prefix vector from `A2B` ending with `C`
{the 1981 expression is flawed as it fails when `⎕IO` is zero. `((~⎕IO)+A2B⍳C)↑A2B` is better ;-) }

`ORGDEST←VCT[VCT[;2]⍳⌊ADDRESS÷2;0 1=2|ADDRESS]`
The preceding expression extracts two elements from a row of `VCT`.
`VCT[;2]⍳⌊ADDRESS÷2` generates the row index
`0 1=2|ADDRESS` generates the column index: IF `ADDRESS` even THEN 1 0 ELSE 0 1 ENDIF

`(2 ⎕WS 3)`
"An integer vector that at present contains twelve elements, describing certain aspects of the active task and active workspace."
{SHARP APL REFERENCE MANUAL by Paul Berry 1981 p227}

`256 | X` gives the remainder after division by 256.

`0 3↔⍴VCT` impure APL which asserts that `VCT` is a three column matrix with zero rows.
{`VCT←0 3⍴0` assignment or the more modern `0 3≡⍴VCT` equivalence are better}

`VCT←VCT,[0] ZERODIRECTION,ONEDIRECTION,NEWCALLNUM` catenation adds a new row to `VCT`

`NEWCALLNUM∊VCT[;2]` membership again

`+/WEIGHTS[R1] < +/WEIGHTS[R2]`
`+/X` sums all the elements of `X`
{ `(+/WEIGHTS[R1]) < +/WEIGHTS[R2]` is correct APL as APL function precedence is strictly right to left.
This is unlike C/Java which share a lengthy precedence list.}