A DISTRIBUTED ROUTING ALGORITHM FOR UNIDIRECTIONAL NETWORKS

M. Gerla, L. Kleinrock and Y. Afek

Reprinted from IEEE Global Telecommunications Conference,
GLOBECOM '83, November 28-December 1, 1983, San Diego, CA

A DISTRIBUTED ROUTING ALGORITHM FOR UNIDIRECTIONAL NETWORKS

M. Gerla
L. Kleinrock
Y. Afek

Computer Science Department
University of California, Los Angeles

ABSTRACT

A unidirectional communications network is defined to be a distributed packet network in which some (or all) of the links are unidirectional (instead of bidirectional). That is, the presence of a channel from A to B does not necessarily imply the presence of a channel from B to A. Examples of unidirectional networks are found in packet radio and packet satellite environments.

Conventional, distributed routing algorithms generally cannot be applied to unidirectional networks. In this paper we present an efficient distributed algorithm to compute shortest paths between two-way connected node pairs in a unidirectional network. As a byproduct, the algorithm builds at each node the list of two-way connected nodes, i.e., nodes for which there is a direct path to and from that node. The computational overhead of this algorithm is shown to be comparable to that of conventional, bidirectional routing algorithms.

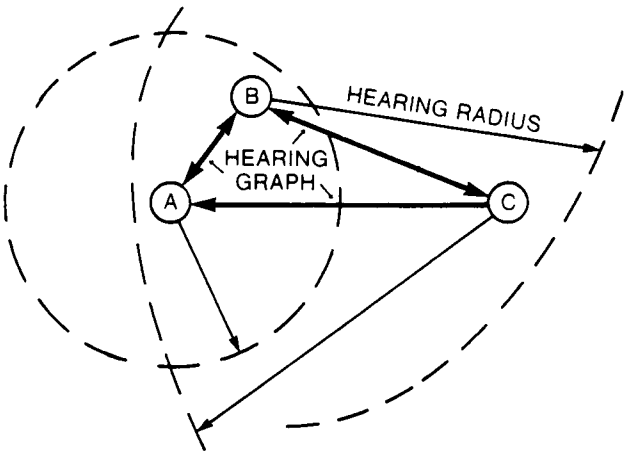


Figure 1. Non symmetric hearing configuration in a Packet Radio system

1. INTRODUCTION

A unidirectional communications network is defined here to be a packet switched network in which the communications processors are connected by unidirectional (instead of bidirectional) channels. That is, the presence of a channel from node A to node B does not necessarily imply the presence of another channel in the opposite direction. Using communications systems terminology, all channels are simplex channels (as opposed to duplex channels). Note that bidirectional channels are permitted and will be implemented by two simplex channels in opposite directions.

Most of the protocol development work in the past has been devoted to bidirectional communications networks. Yet, some network environments can be more properly modeled with unidirectional networks. For example, in packet radio networks it often happens that, because of transmission power and antenna gain imbalance, the "hearing" matrix is not symmetric (see fig 1) [KAHN 78].

Another cause of asymmetry in radio networks using frequency diversity (i.e., different frequencies at different repeaters) is the dependency of signal attenuation on frequency due to atmospheric conditions and multipath interference. Thus, the channel from node A to node B (on one frequency) may be temporarily cut off, while the B to A channel is operative (on another frequency).

Other examples of unidirectional networks are packet radio networks used in a hostile environment, where selective unidirectionality may be chosen to prevent the enemy from tracking the geographical location of some of the Packet Radio Units; and mixed terrestrial and satellite networks in which all the nodes can receive from the satellite, but only a few can also transmit to it.

Networks in which some or all the links are unidirectional require different protocols than bidirectional networks. For example, the link level protocol on a unidirectional link provides error detection (or, at most forward error correction), but not error recovery via retransmission [CARL 80]. Thus, full error protection must be implemented at the transport level (i.e. ISO level 4) [ZIMM 80]. At the network layer, a special routing algorithm must be developed to account for the fact that the path from A to B is generally different from the B to A path, and for the fact that routing updates can be transmitted only to "downstream" neighbors.

In this paper we propose a distributed algorithm for the computation of minimum hop routes in a unidirectional network. As a byproduct, the algorithm provides at each

83CH1956-2/83/0000-0654 $1.00

node, say, v, a list of nodes with which v is two-way connected, that is, with which v has both a directed path to and a directed path from itself. The knowledge of two-way connectivity is essential to determine whether two-way communication is possible with a remote node.

The proposed algorithm is a fully distributed algorithm in which every node participates in the routing computation and periodically broadcasts its routing and distance information to its immediate neighbors. A node is not required to store the full topology map in its memory - it only needs to store routing and distance tables. In this respect the algorithm is reminiscent of the "old" ARPANET routing algorithm [McQU 72].

Other distributed approaches could be proposed to solve the routing problem in unidirectional networks. In particular, each node could be required to broadcast its local "inward" connectivity (i.e., the ID's of the neighbors from which it can hear) to the entire network, using a flooding algorithm. Each node then constructs a complete topological map of the network based on the information received from all other nodes and computes shortest routes accordingly. This approach would be very similar to the "new" ARPANET algorithm described in [McQU 80]. We chose not to follow this approach for a number of reasons. First, the flooding procedure is complicated to implement: in order to detect (and discard) duplicates each node must remember packet ID's for the packet lifetime. Secondly, the storage required to store the entire topology may be excessive if the network is large. Finally, an intruder could easily learn the entire network topology by eavesdropping - a very undesirable property in hostile environments.

2. PROBLEM STATEMENT

The unidirectional network routing problem can be stated as follows:

In a network with unidirectional links design a distributed algorithm which:

(a) constructs at each node a table of all nodes with which two-way communication is possible.

(b) computes the minimum hop distance and the best route to such nodes.

(c) is robust to link and node failures

Note: we assume a flooding approach (as in the "new" ARPANET routing algorithm [McQU 80]) is not permitted because of the above mentioned limitations (namely, each node must remember packet ID's for a while; high storage overhead is involved; and, the entire network topology can be easily discovered by an intruder).

3. SOLUTION APPROACH

Let the *upstream* neighbors (or, up-neighbors for short) of node i be the nodes which have directed links to node i. Likewise, down-neighbors are the nodes to which i is directly connected In the example in Figure 2, nodes 2 and 3 are upstream neighbors of node 1; node 4 is the down-neighbor of 1.

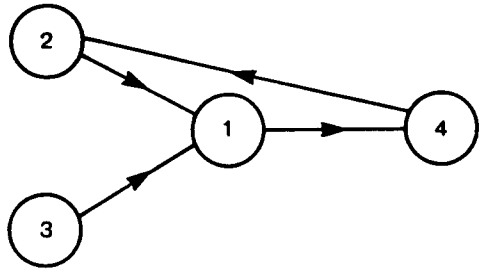


Figure 2. Upstream neighbors

The proposed approach consists of two phases, as follows:

- First, we develop a mechanism that permits each node to identify which up-neighbors it can talk back to, and along which path. In the example in Fig. 2, node 1 finds out that it can talk back to 2 via 4. This mechanism is described in Step 1 below.
- Next, we use the standard min hop routing algorithm [GERL 81] in which each node broadcasts to its up-neighbors its min hop estimates to all 2-way connected destinations. This procedure is described in Step 2 and Step 3 below.

4. ALGORITHM

Step 1

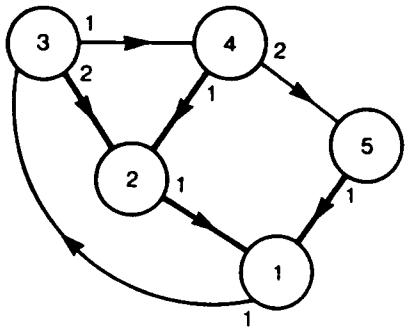
Each node v builds the inbound shortest tree, i.e., the shortest tree pointing to it as the root. This tree contains the shortest, directed path from each node in the network to v. In the shortest path computation, the length of each directed link is assumed to be unity. The tree thus includes all the nodes which have a directed path to v.

The inbound shortest tree is maintained at each node as a "parent tree" [KNUT 68]. That is, a vector P of size N (N = number of nodes) is kept at each node, where $P(i)$ is the ID of the "parent" of i in the inbound tree of v. In addition to the parent ID, for each node, the shortest distance $D(i)$ from i to the root (along the shortest path) is also stored, as is the ID link $L(i)$ from i to the parent. It is assumed that each node labels its outgoing links sequentially.

Parent, Distance and Link ID information are kept in the PDL Table. Figure 3 shows an example of PDL Table computed at node 1. The "thick" links define the inbound shortest tree into node 1.

The PDL table is constructed using a distributed algorithm as follows:

(a) Periodically (but asynchronously) each node (say node i) transmits on each of its outgoing links the PDL



AT NODE 1:

	P	D	L
1	0	0	0
2	1	1	1
3	2	2	2
4	2	2	1
5	1	1	1

Figure 3. PDL Table

table. Before transmission, the outgoing link ID is entered in the link field for i in the PDL table. Namely if the outgoing link p is chosen, the PDL table is set as follows:

$$L(i) \leftarrow p$$

(b) Upon receiving from each up-neighbor k the PDL table (with entries for a generic node s denoted $P_k(s)$, $D_k(s)$ and $L_k(s)$) node i updates its PDL table as follows:

$$D(s) \leftarrow \min [D_k(s) + 1], \text{ over all neighbors } k$$

$$L(s) \leftarrow L_{k^*}(s),$$

where k^* is the neighbor yielding the minimum distance. Ties are resolved by choosing the smallest k^* i.e., the neighbor with the smallest ID.

$$P(s) \leftarrow \begin{cases} P_k(s) & \text{if } s \neq k^* \\ i & \text{if } s = k^* \end{cases}$$

(c) Initially, node i sets its PDL table as follows

$$P(i) \leftarrow D(i) \leftarrow L(i) \leftarrow 0$$

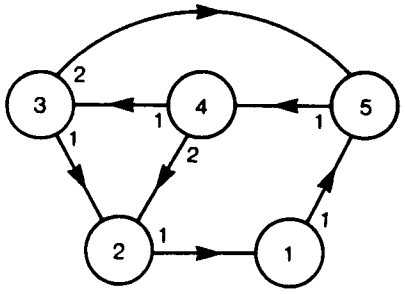
$$D(s) \leftarrow \infty, \text{ for } s \neq i$$

The Step 1 procedure converges yielding the inbound shortest path at each node. In fact, if the shortest distance from node s to node i is n, node i will have the correct entry in the PDL table after n updates. Thus, the entire Step 1 phase converges in d update periods where d is the network diameter. Convergence and overhead issues will be discussed in more detail in Sections 5 and 7.

The PDL table has the following practical significance. At steady state, each node knows that it can hear from all nodes whose distance is $\leq N$, where N is the total number of nodes in the network. It cannot hear from nodes at distance $> N$. In fact, the distance to the latter set of nodes is always infinity at steady state.

Step 2

After receiving the PDL table from up-neighbor k, and using such a table to update its own PDL table, node i proceeds to inspect $D_k(i)$.



2 SENDS TO 1 THE PDL TABLE BELOW:

	P	D	L
1	5	3	1
2	0	0	0
3	2	1	1
4	2	1	2
5	4	2	1

CYCLE FROM 1 THROUGH 2: (5,4,2,1)
LINK ID'S: (1,1,2,1)

Figure 4. Path tracing to up-neighbors

If $D_k(i) < N$, node i concludes that it has a directed path to k, i.e., k is a 2-way up-neighbor. Node i then generates and stores the shortest cycle through k by simply tracing the parents through the PDL table received from k. In the Example in Figure 4 node 1 has just received the PDL table from node 2. It then checks the vector P and determines that the shortest cycle through to 2 is (5, 4, 2). At the same time it determines (from vector L) the sequence of link ID's (in this case (1, 1, 2, 1)) associated with the cycle.

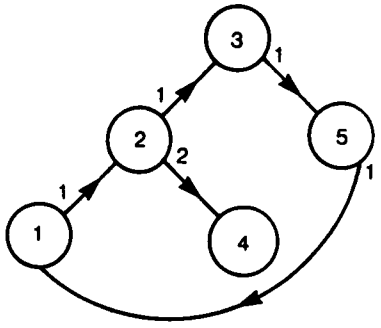
The path so traced allows node i to send messages to each 2-way up neighbor using a "path driven" routing procedure. That is, node i stamps in the packet header the sequence of link IDs which defines the cycle and transmits the packet on the first link in the sequence. The second node on the path will forward the packet on the second link, and so on until the packet reaches its destination. From the last entry in the header, the destination extracts the ID of its outgoing link pointing to the source node. This capability to communicate with up-neighbors is exploited in Step 3.

Step 3

Once each node has established a path to its 2-way upstream neighbors, the equivalent of the "old" ARPA routing algorithm can be carried out in the unidirectional network. Namely, Step 3 computes routing and hop vectors by mean of a distributed procedure in which every node updates its tables using the information received from the up-neighbors, and broadcasts its tables to the down-neighbors.

At each node, in addition to the PDL table, a routing vector (R-vector) and a hop vector (H-vector) are maintained. The R-vector stores the ID of the outgoing link on the shortest path to each 2-way connected destination. The H-vector stores the minimum hop distance to each such destination (see Figure 5 for an example). Note that $H < N$ implies that the destination is 2-way reachable, that is, there is a path from and a (generally different) path to that destination. If the destination is unreachable, its corresponding value of H is infinity .

The R and H vectors are computed using the distributed algorithm outlined below:



	R	H
1	0	0
2	1	1
3	1	2
4	—	∞
5	1	3

R-H TABLE AT NODE 1

Figure 5. R and H vectors

(a) Periodically each node sends to its 2-way up-neighbors its H-vector. The H-vector is delivered along the path computed in Step 2 above using a "path driven" routing procedure. A special flag is set in the header to request that intermediate nodes use the path driven (rather than the destination driven) routing procedure. Note that at steady state path driven routing and destination driven routing coincide. During initialization, however, destination driven routing is not feasible since the R-vector has not been computed yet.

(b) Upon receiving all down-neighbor H-vectors, node i updates its H and R vectors as follows:

$$H(i) \leftarrow 0$$

For all $s \neq i$ do:

$$H(s) \leftarrow \min [H_k(s) + 1], \text{ over all neighbors } k$$

$$R(s) \leftarrow \text{Link}(i - k^*)$$

where k^* is the down-neighbor yielding minimum hop distance, and Link $(i - k^*)$ is the ID of the outgoing link to k^*. Recall that this link ID is obtained from the last entry of the header of the "path driven" update packet sent by k^*.

(c) Initially, the H-vector for node i is defined as follows:

$$H(s) = \infty, \text{ for } s \neq i$$

$$H(i) = 0$$

R is set to arbitrary values.

The procedure converges since a node learns the correct distance (say n) to a 2-way destination in n update periods. Convergence is discussed in more detail in Section 5.

At steady state all the nodes with hop distance $H(s) = \infty$ are not 2-way connected to node i. The remaining nodes are two-way connected. For the latter, $H(s) < N$, and the R-vector indicates the shortest path to them.

5. CONVERGENCE AND ADAPTIVITY TO FAILURES

The algorithm adjusts automatically (without operator intervention) to link and node failures and to the insertion of new nodes/links in the topology. Here we prove that the correct distances and routes to the two-way connected nodes are re-established in no more than $2d$ update periods after the occurrence of the topological change, where d is the network diameter. This also proves convergence at network initialization, since initialization can be viewed as a special case of network topology change. If a node becomes (two-way) disconnected because of a change, this condition is detected in no more than $2N$ updates, where N = total number of nodes in the network.

Immediately following a topological change, some of the entries in the PDL and routing tables are incorrect. To prove convergence after a topology change, it suffices to show that the algorithm converges to the correct solution regardless of the initial table values.

We first consider the convergence of the Step 1 procedure, that is, the inbound shortest tree computation. We assume that initially (at iteration 0) each node i has an arbitrary estimate $H_i^n(s)$ of the length of the shortest directed path from node s to itself. Obviously, $H_s^n(s) = 0$. It is also reasonable to assume that $H_i^n(s) \geq 1$ for $i \neq s$. For a node $v \neq s$ the estimate $H_v^n(s)$ at iteration n of Step 1 algorithm is simply given by the minimum over the neighbors' estimates at iteration n - 1, plus one. Applying this property recursively, we find:

$$H_v^n(s) = \begin{cases} \min_{i \in I} \{H_i^n(s) + n\}, & \text{for } n < h \\ \min \left\{ \min_{i \in I} \{H_i^n(s) + n\}, h \right\} & \text{for } n \geq h \end{cases}$$

where:

I = set on nodes having a directed path of length n (not necessarily the shortest path) to node v

h = length of shortest directed path from s to v ($h = \infty$, if there is no path from s to v).

Note that the term h in the r.h.s. of the above expression is due to the propagation of the estimate $H_s^n(s) = 0$ along the shortest path from s to v. This propagation requires exactly h iterations.

From the above expression it is now clear that $H_v^n(s) = h$ (i.e., the estimate at node v is correct) in at most h iterations. In fact $H_i^n(s) + n > h$ for $n > h$, for all $i \in I$, since $H_i^n(s) \Rightarrow 0$. Recalling that $h =< d$ by definition, the Step 1 procedure converges for the entire network in at most d iterations after the topological change.

If $h = \infty$ (i.e., there is no path from s to v), v detects this condition by verifying that its estimate $H_v(s) > N$ after N iterations.

It can be easily verified that the convergence of the shortest outgoing tree computation in Step 3 satisfies the same properties as the inbound tree computation. Thus, we conclude that the correct routing tables are re-established within $2d$ iterations after the topological change. If the change has caused a two-way disconnection, this fact is detected in no more than $2N$ iterations.

6. ROUTING OPERATION

Two distinct routing procedures are implemented in the network, namely: path driven routing and destination driven routing

Path driven routing is used exclusively to send the H vector to an up-neighbor as explained in Section 4, Step 2. The originating node turns on the path routing flag and stamps the sequence of intermediate links in the header. The last link in the sequence is the link from the up-neighbor to the originating node itself. It is used by the up-neighbor to update its R vector, as explained in Section 4, Step 3.

Destination driven routing is used for all other packets (including data packets). Each node, upon receiving a packet directed to destination s, will test to see if $H(s) < N$, in which case the packet is forwarded to link $R(s)$; otherwise, it is dropped.

7. OVERHEAD CONSIDERATIONS

The CPU processing cost of the unidirectional routing algorithm is approximately 2.5 times the cost of the bidirectional algorithm. Instead of processing two vectors (R and H) per update period, we process five (R, H, P, D and L).

As discussed in Section 5, the algorithm converges in at most $2d$ update periods, where d is the network diameter. In fact, it takes d update periods to compute the PDL tables, and another d update periods (in the worst case) to compute the RH tables after the PDL tables have been computed. We assume that the update interval is larger than the time required to ship a set of RH tables from a node to its up-neighbor. In comparison, the bidirectional algorithm converges in d steps.

The line overhead can be measured in terms of messages exchanged during an update period. In the following we assume error free channels, i.e., no retransmissions. We have $|E|$ PDL messages, where E is the set of edges, and $(b-1)|E'|$ RH messages, where E' is the set of edges joining two-way connected neighbors (i.e., the edges in the strongly connected subgraphs) and b is the average length of the shortest cycles through the edges of E', i.e., the cycles obtained by constructing for each edge in E' the shortest cycle through it. The worst case is (among the strongly connected networks) the n-loop network, where $|E'| = |E|$ and $b = n$. The best case (among the strongly connected networks) is the bidirectional network, where $|E'| = |E|$ and $b = 2$. Thus, for bidirectional networks the line overhead caused by the unidirectional algorithm is only twice the overhead of the bidirectional algorithm.

8. CONCLUSION

We have presented a distributed algorithm for the computation of two-way connectivity and shortest paths in a unidirectional communications network. The unidirectional algorithm is robust to failures and, more generally, to topological changes. It is efficient in terms of memory, line traffic and processing overhead. Moreover, when the unidirectional algorithm is applied to a bidirectional network, it converges in the same number of steps as the bidirectional algorithm and produces only twice the overhead of the latter. Thus, the unidirectional algorithm can be efficiently applied to networks with a mix of unidirectional and bidirectional channels.

Several extensions to the basic algorithm are possible. Table updating, for example, can be carried out in an incremental fashion as soon as a PDL or H vector is received, rather than waiting until all the vectors have been received from all neighbors. The algorithm can also be applied to unidirectional packet radio networks. Some modifications, however, are required since the present version assumes point-to-point channels between nodes, while in a radio network a node broadcasts each message to all the down-neighbors. It is also possible to apply the algorithm to more general link cost functions and thus solve more general problems than hop minimization. In particular, work is now under way to solve the *minimum delay* routing problem in a unidirectional network using a modification of our basic algorithm.

REFERENCES

[CARL 80] Carlson, D. E., "Bit-Oriented Data Link Control Procedures," *IEEE Transactions on Communication*, April 1980, pp. 455-467.

[GERL 81] Gerla, M., "Routing and Flow Control," in *Protocols and Techniques for Data Communications Networks*, F. F. Kuo, Editor, Prentice Hall, 1981.

[KAHN 78] Kahn, R. E. et al, "Advances in Packet Radio Technology," *IEEE Proceedings*, Nov. 1978, pp. 1468-1496.

[KNUT 68] Knuth, D. E., *The Art of Computer Programming, Volume 1-Fundamental Algorithms*, Addison Wesley, Reading, MA, 1968.

[McQU 72] McQuillan, J. et al, "Improvements in the Design and Performance of ARPANET," *Fall Joint Conference Proceedings*, Fall 1972.

[McQU 80] McQuillan, J. et al, "The New Routing Algorithm for the ARPANET," *IEEE Transactions on Communication*, May 1980.

[ZIMM 80] Zimmermann, H., "OSI Reference Model," *IEEE Transactions on Communication*, April 1980, pp 425-432.

This research was support by DARPA contract MDA 903-82-C-0064.