Network Working Group
Request for Comments: 981

D. L. Mills
M/A-COM Linkabit
March 1986

An Experimental Multiple-Path Routing Algorithm

Status of This Memo

This RFC describes an experimental, multiple-path routing algorithm designed for a packet-radio broadcast channel and discusses the design and testing of a prototype implementation. It is presented as an example of a class of routing algorithms and data-base management techniques that may find wider application in the Internet community. Of particular interest may be the mechanisms to compute, select and rank a potentially large number of speculative routes with respect to the limited cumputational resources available. Discussion and suggestions for improvements are welcomed. Distribution of this memo is unlimited.

Abstract

This document introduces wiretap algorithms, which are a class of routing algorithms that compute quasi-optimum routes for stations sharing a broadcast channel, but with some stations hidden from others. The wiretapper observes the paths (source routes) used by other stations sending traffic on the channel and, using a heuristic set of factors and weights, constructs speculative paths for its own traffic. A prototype algorithm, called here the Wiretap Algorithm, has been designed for the AX.25 packet-radio channel. Its design is similar in many respects to the shortest-path-first (spf) algorithm used in the ARPANET and elsewhere, and is in fact a variation in the class of algorithms, including the Viterbi Algorithm, that construct optimum paths on a graph according to a distance computed as a weighted sum of factors assigned to the nodes and edges.

The Wiretap Algorithm differs from conventional algorithms in that it computes not only the primary route (a minimum-distance path), but also additional paths ordered by distance, which serve as alternate routes should the primary route fail. This feature is also useful for the discovery of new paths not previously observed on the channel.

Since the amateur AX.25 packet-radio channel is very active in the Washington, DC, area and carries a good deal of traffic under punishing conditions, it was considered a sufficiently heroic environment for a convincing demonstration of the prototype algorithm. It was implemented as part of an IP/TCP driver for the LSI-11 processor running the "fuzzball" operating system. The driver is connected via serial line to a 6809-based TAPR-1 processor running the WA8DED firmware, which controls the radio equipmnet in both

virtual-circuit and datagram modes. The prototype implementation provides primary and alternate routes, can route around congested areas and can change routes during a connection. This document describes the design, implementation and initial testing of the algorithm.

1. Introduction

This document describes the design, implementation and initial testing of the Wiretap Algorithm, a dynamic routing algorithm for the AX.25 packet-radio channel [4]. The AX.25 channel operates in CSMA contention mode at VHF frequencies using AFSK/FM modulation at 1200 bps. The AX.25 protocol itself is similar to X.25 link-layer protocol LAPB, but with an extended frame header consisting of a string of radio callsigns representing a path, usually selected by the operator, between two end stations, possibly via one or more intermediate packet repeaters or digipeaters. Most stations can operate simultaneously as intermediate systems digipeaters) and as end systems with respect to the ISO model.

Wiretap uses passive monitoring of frames transmitted on the channel in order to build a dynamic data base which can be used to determine optimum routes. The algorithm operates in real time and generates a set of paths ordered by increasing total distance, as determined by a shortest-path-first procedure similar to that used now in the ARPANET and planned for use in the new Internet gateway system [2]. The implementation provides optimum routes (with respect to the factors and weights selected) at initial-connection time for virtual circuits, as well as for each datagram transmission. This document is an initial status report and overview of the prototype implementation for the LSI-11 processor running the "fuzzball" operating system.

The principal advantage in the use of routing algorithms like Wiretap is that digipeater paths can be avoided when direct paths are available, with digipeaters used only when necessary and also to discover hidden stations. In the present exploratory stage of evolution, the scope of Wiretap has been intentionally restricted to passive monitoring. In a later stage the scope may be extended to include the use of active probes to discover hidden stations and the use of clustering techniques to manage the distribution of large quantities of routing information.

The AX.25 channel interface is the 6809-based TAPR-1 processor running the WA8DED firmware (version 1.0) and connected to the LSI-11 by a 4800-bps serial line. The WA8DED firmware produces as an option a monitor report for each received frame of a selected type,

including U, I and S frames. Wiretap processes each of these to extract routing information and (optionally) saves them in the system log file. Following is a typical report:

```
fm KS3Q to W4CQI via WB4JFI-5* WB4APR-6 ctl I11 pid F0
```

The originating station is KS3Q and the destination is W4CQI. The frame has been digipeated first by WB4JFI-5 and then WB4APR-6, is an I frame (sequence numbers follow the I indicator) and has protocol identifier F0 (hex). The asterisk "*" indicates the report was received from that station. If no asterisk appears, the report was received from the originator.

2. Design Principles

A path is a concatenation of directed links originating at one station, extending through one or more digipeaters and terminating at another station. Each link is characterized by a set of factors such as cost, delay or throughput that can be computed or estimated. Wiretap computes several intrinsic factors for each link and updates the routing data base, consisting of node and link tables. The weighted sum of these factors for each link is the distance of that link, while the sum of the distances for each link in the path is the distance of that path.

It is the intent of the Wiretap design that the distance of a link reflect the a-priori probability that a packet will successfully negotiate that link relative to the other choices possible at the sending node. Thus, the probability of a non-looping path is the product of the probabilities of its links. Following the technique of Viterbi [1], it is convenient to represent distance as a logarithmic transformation of probability, which then becomes a metric. However, in the following the underlying probabilities are not considered directly, since the distances are estimated on a heuristic basis.

Wiretap incorporates an algorithm which constructs a set of paths, ordered by distance, between given end stations according to the factors and weights contained in the routing data base. Such paths can be considered optimum routes between these stations with respect to the given assignment of factors and weights. In the prototype implementation one of the end stations must be the Wiretap station itself; however, in principle, the Wiretap station can generate routes for other stations subject to the applicability of the information in its data base.

Note that Wiretap in effect constructs minimum-distance paths in the

direction from the destination station to the Wiretap station and, based on that information, then computes the optimum reciprocal routes from the Wiretap station to the destination station. The expectation is that the destination station also runs its own routing algorithm, which then computes its own optimum reciprocal routes (i.e. the optimum direct routes from the Wiretap station). However, the routing data bases at the two stations may diverge due to congestion or hidden stations, so that the computed routes may not coincide.

In principle, Wiretap-computed routes can be fine-tuned using information provided not only by its directly communicating stations but others that may hear them as well. The most interesting scenario would be for all stations to exchange Wiretap information using a suitable distributed protocol, but this is at the moment beyond the scope of the prototype implementation. Nevertheless, suboptimum but useful paths can be obtained in the traditional and simple way with one station using a Wiretap-computed route and the other its reciprocal, as determined from the received frame header. Thus, Wiretap is compatible with existing channel procedures and protocols.

3. Implementation Overview

The prototype Wiretap implementation for the LSI-11 includes two routines, the wiretap routine, which extracts information from received monitor headers and builds the routing data base, and the routing routine, which calculates paths using the information in the data base. The data base consists of three tables, the channel table, node table and link table. The channel table includes an entry for each channel (virtual circuit) supported by the TAPR-1 processor running the WA8DED firmware, five in the present configuration. The structure and use of this table are only incidental to the algorithm and will not be discussed further.

The node table includes an entry for each distinct callsign (which may be a collective or beacon identifier) heard on the channel, together with node-related routing information, the latest computed route and other miscellaneous information. The table is indexed by node ID (NID), which is used in the computed route and in other tables instead of the awkward callsign string. The link table contains an entry for each distinct (unordered) node pair observed in a monitor header. Each entry includes the from-NID and to-NID of the first instance found, together with link-related routing information and other miscellaneous information. Both tables are dynamically managed using a cache algorithm based on a weighted least-recently-used replacement mechanism described later.

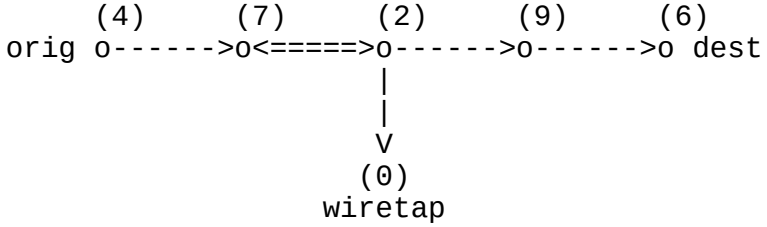
The example discussed in Appendix A includes candidate node and link tables for illustration. These tables were constructed in real time by the prototype implementation from off-the-air monitor headers collected over a typical 24-hour period. Each node table entry requires 26 bytes and each link table entry four bytes. The maximum size of the node table is presently 75 entries, while that of the link table is 150 entries. Once the cache algorithm has stabilized for a day or two, it is normal to have about 60 entries in the node table and 100 entries in the link table.

The node table and link table together contain all the information necessary to construct a network graph, as well as calculate paths on that graph between any two end stations, not just those involving the Wiretap station. Note, however, that the Wiretap station does not in general hear all other stations on the channel, so may choose suboptimum routes. However, in the Washington, DC, area most stations use one of several digipeaters, which are in general heard reliably by other stations in the area. Thus, a Wiretap station can eventually capture routes to almost all other stations using the above tables and the routing algorithm described later.

4. The Wiretap Routine

The wiretap routine is called to process each monitor header. It extracts each callsign from the header in turn and searches the node table for corresponding NID, making a new entry and NID if not already there. The result is a string of NIDs, starting at the originating station, extending through a maximum of eight digipeaters and ending at the destination station. For each pair of NIDs along this string the link table is searched for either the direct link, as indicated in the string, or its reciprocal; that is, the direction towards the originator.

The operations that occur at this point can be illustrated by the following diagram, which represents a monitor header with apparent path from station 4 to station 6 via digipeaters 7, 2 and 9 in sequence. It happens the header was heard by the Wiretap station (0) from station 2.



Presumably, the fact that the header was heard from station 2 indicates the path from station 4 to station 2 and then to station 0 is viable, so that each link along this path can be marked "heard" in that direction. However, the viability of the path from station 2 to station 6 can only be presumed, unless additional evidence is available. If in fact the header is from an AX.25 I or S frame (but not a U frame), an AX.25 virtual circuit has apparently been previously established between the end stations and the presumption is strengthened. In this case each link from 4 to 6 is marked "synchronized" (but not the link from 2 to 0).

Not all stations can both originate frames and digipeat them. Station 4 is observed to originate and station 7 to digipeat, but station 9 is only a presumptive digipeater and no evidence is available that the remaining stations can originate frames. Thus, the link from station 4 to station 7 is marked "source" and from station 7 to station 2 is marked "digipeated."

Depending on the presence of congestion and hidden stations, it may happen that the reciprocal path in the direction from station 6 to station 4 has quite different link characteristics; therefore, a link can be recognized as heard in each direction independently. In the above diagram the link between 2 and 7 has been heard in both directions and is marked "reciprocal". However, there is only one synchronized mark, which can be set in either direction. If a particular link is not marked either heard or synchronized, any presumption on its viability to carry traffic is highly speculative (the traffic is probably a beacon or "CQ"). If later marked synchronized the presumption is strengthened and if later marked heard in the reciprocal direction the presumption is confirmed.

Experience shows that a successful routing algorithm for any packet-radio channel must have provisions for congestion avoidance. There are two straightforward ways to cope with this. The first is a static measure of node congestion based on the number of links in the network graph incident at each node. This number is computed by the wiretap routine and stored in the node table as it adds entries to the link table.

The second, not yet implemented, is a dynamic measure of node congestion which tallies the number of link references during the most recent time interval (of specified length). The current plan was suggested by the reachability mechanism used in the ARPANET and the Exterior Gateway Protocol [3]. An eight-bit shift register for each node is shifted in the direction from high-order to low-order bits, with zero-bits preceeding the high-order bit, at the rate of one shift every ten seconds. If during the preceeding ten-second

period a header with a path involving that node is found, the high-order bit of the register is set to one. When a path is calculated the number of one-bits in the register is totalled and used as a measure of dynamic node congestion. Thus, the time interval specified is 80 seconds, which is believed appropriate for the AX.25 channel dynamics.

5. Factor Computations and Weights

The data items produced by the wiretap routine are processed to produce a set of factors that can be used by the routing routine to develop optimum routes. In order to insure a stable and reliable convergence as the routing algorithm constructs and discards candidate paths leading to these routes, the factor computations should have the following properties:

1. All factors should be positive, monotone functions which increase in value as system performance degrades from optimum.

2. The criteria used to estimate link factors should be symmetric; that is, their values should not depend on the particular direction the link is used.

3. The criteria used to estimate node factors should not depend on the particular links that traffic enters or leaves the node.

Each factor is associated with a weight assignment which reflects the contribution of the factor in the distance calculation, with larger weights indicating greater importance. For comparison with other common routing algorithms, as well as for effective control of the computational resources required, it may be desirable to impose additional restrictions on these computations, which may be a topic for further study. Obviously, the success of this routing algorithm depends on cleverly (i.e. experimentally) determined factor computations and weight assignments.

The particular choices used in the prototype implementation should be considered educated first guesses that might be changed, perhaps in dramatic ways, in later implementations. Nevertheless, the operation of the algorithm in finding optimum routes over all choices in factor computations and weights is unchanged. Recall that the wiretap routine generates data items for each node and link heard and saves them in the node and link tables. These items are processed by the routing routine to generate the factors shown below in Table 1 and Table 2.

Factor	Weight	Name	How Determined
f0	30	hop	1 for each link
f1	50	unverified	1 if not heard either direction
f2	5	non-reciprocal	1 if not heard both directions
f3	5	unsynchronized	1 if no I or S frame heard

Table 1. Link Factors

Factor	Weight	Name	How Determined
f4	5	complexity	1 for each incident link
f5	20	digipeated	1 if station does not digipeat
f6	-	congestion	(see text)

Table 2. Node Factors

With regard to link factors, the "hop" factor is assigned as one for each link and represents the bias found in other routing algorithms of this type. The intent is that the routing mechanism degenerate to minimum-hop in the absence of any other information. The "unverified" factor is assigned as one if the heard bit is not set (not heard in either direction), while the "non-reciprocal" factor is assigned as one if the reciprocal bit is not set (not heard in both directions). The "unsynchronized" factor is assigned as one if the synchronized bit is not set (no I or S frames observed in either direction).

With regard to node factors, the "complexity" factor is computed as the number of links incident at the node, while the "congestion" factor is to be computed as the number of intervals in the eight ten-second intervals preceding the time of observation in which a frame was transmitted to or through the node. The "digipeated" factor is assigned as one if the node is only a source (i.e. no digipeated frames have been heard from it). For the purposes of path-distance calculations, the node factors are taken as zero for the endpoint nodes, since their contribution to any path would be the same.

6. The Routing Routine

The dynamic data base built by the wiretap routine is used by the routing routine to compute routes as required. Ordinarily, this needs to be done only when the first frame to a new destination is sent and at intervals thereafter, with the intervals perhaps modulated by retry count together with congestion thresholds, etc. The technique used is a variation of the Viterbi Algorithm [1], which is similar to the the shortest-path-first algorithm used in the ARPANET and elsewhere [2]. It operates by constructing a set of candidate paths on the network graph from the destination to the source in increasing number of hops. Construction continues until all the complete paths satisfying a specified condition are found, following which one with minimum distance is selected as the primary route and the others ranked as alternate routes.

There are a number of algorithms to determine the mimimum-distance path on a graph between two nodes with given metric. The prototype implementation operates using a dynamic path list of entries derived from the link table. Each list entry includes (a) the NID of the current node, (b) a pointer to the preceding node on the path and (c) the hop count and (d) distance from the node to the final destination node of the path:

[NID, pointer, hop, distance] .

The algorithm starts with the list containing only the entry [dest-NID, 0, 0, 0], where dest-NID is the final destination NID, and then scans the list starting at this entry. For each such entry it scans the link table for all links with either to-NID or from-NID matching NID and for each one found inserts a new entry:

[new-NID, new-pointer, hop + 1, distance + weight] ,

where the new-NID is the to-NID of the link if its from-NID matches the old NID and the from-NID of the link otherwise. The new-pointer is set at the address of the old entry and the weight is computed from the factors and weights as described previously. The algorithm coontinues to select succeeding entries and scan the link table until no further entries remain to be processed, the allocated list area is full or the maximum hop count or distance are exceeded, as explained below.

Note that in the Viterbi Algorithm, which operates in a similar manner, when paths merge at a single node, all except one of the minimum-distance paths (called survivors) are abandonded. If only one of the minimum-distance paths is required, Wiretap does the same;

however, in the more general case where alternate paths are required, all non-looping paths are potential survivors. In order to prevent a size explosion in the list, as well as to suppress loops, new list entries with new-NID matching the NID of an existing entry on the path to the final destination NID are suppressed and paths with hop counts exceeding (currently) eight or distances exceeding 255 are abandoned.

If the Wiretap station NID is found in the from-NID of an entry inserted in the list, a complete path has been found. The algorithm remembers the minimum distance and minimum hop count of the complete paths found as it proceeds. When only one of the minimum-distance paths (primary route) is required, then for any list entry where the distance exceeds the minimum distance or the hop count exceeds the maximum hop count (plus one), the path is abandoned and no further processing done for it. When alternate routes are required the hop-count test is used, but the minimum-distance test is not.

The above pruning mechanisms are designed so that the the algorithm always finds all complete paths with the minimum hop count and the minimum hop count (plus one), which are designated the alternate routes. The assignment of factor computations and weights is intended to favor minimum-hop paths under most conditions, but to allow the path length to grow by no more than one additional hop under conditions of extreme congestion. Thus, the minimum-distance path (primary route) must be found among the alternate paths, usually, but not always, one of the minimum-hop paths.

At the completion of processing the complete paths are ranked first by distance, then by the order of the final entry in the list, which is in hop-count order by construction, to establish a well-defined ordering. The first of these paths represents the primary route, while the remaining represent alternatives should all lower-ranked routes fail.

Some idea of the time and space complexity of the routing routine can be determined from the observation that the computations for all primary and secondary routes of the example in Appendix A with 58 nodes and 98 links requires a average of about 30 list entries, but occasionally overflows the maximum size, currently 100 entries. Each step requires a scan of all the links and a search (for loops) along the maximum path length, which in principle can add most of the links to the list for each new hop. Obviously, the resources required can escalate dramatically, unless effective pruning techniques such as the above are used.

The prototype implementation requires 316 milliseconds on an

LSI-11/73 to calculate the 58 primary routes to all 58 nodes for an average of about 5.4 milliseconds per route. The implementation requires 1416 milliseconds to calculate the 201 combined primary and alternate routes to all 58 nodes for an average of about 3.4 milliseconds per route.

7. Data Base Housekeeping

In normal operation Wiretap tends to pick up a good deal of errors and random junk, since it can happen that a station may call any other station using ad-hoc heuristics and often counterproductive strategies. The result is that Wiretap may add speculative and erroneous links to the data base. In practice, this happens reasonably often as operators manually try various paths to stations that may be shut down, busy or blocked by congestion. Nevertheless, since Wiretap operates entirely by passive monitoring, speculative links may represent the principal means for discovery of new paths.

The number of nodes and links, speculative or not, can grow without limit as the Wiretap station continues to monitor the channel. As the size of the node table or link table approaches the maximum, a garbage-collection procedure is automatically invoked. The procedure used in the prototype implementation was suggested by virtual-memory storage-management techniques in which the oldest unreferenced page is replaced when a new page frame is required. Every link table entry includes an age field, which is incremented once each minute if its value is less than 60, once each hour otherwise and reset to zero when the link is found in a monitor header. When new space is required in the link table, the link with the largest product of age and distance, as determined by the factor computations and weights, is removed first.

Every node table entry includes the congestion factor mentioned above, which is a count of the number of links (plus one) incident at that node. As links are removed from the link table, these counts are decremented. If the count for some node decrements to one, that node is removed. Thus, if new space is required in the node table, links are removed as described above until the required space is reclaimed.

In addition to the above, and in order to avoid capture of the tables by occasional speculative spasms on one hand and stagnation due to excessively stale information on the other, if the age counter exceeds a predetermined threshold, currently fifteen minutes for a speculative link and 24 hours for other links, the link is removed

from the data base regardless of distance. It is expected that these procedures will be improved as experience with the implementation matures.

8. Summary and Directions for Further Development

Wiretap represents an initial experiment and evaluation of the effectiveness of passive monitoring in the management of the AX.25 packet-radio channel. While the results of initial experiments have been encouraging, considerable work needs to be done in the optimization effectively, some experience needs to be gained in the day-to-day operation of the prototype system during which various combinations of weight assignments can be tried.

The prototype implementation has been in use for about four months at this writing; however, a number of lessons were quickly learned. The implementation includes a finite-state automaton to manage initial connection requests, including the capability to retry SABM frames along alternate routes computed by Wiretap. A simple but effective heuristic is used to generate speculative paths by artificially adding links between the destination station and the Wiretap station together with all other stations in the node table identified as digipeaters. The algorithm then operates as described above to generate the primary and alternate routes. An example of this technique is given in the Appendix.

This technique works very well, at least in the initial-connection phase of virtual-circuit mode, although it requires significant computational resources, due to the large number of possible paths ranging from reasonable to outrageous. In the case of datagram mode only the primary route is computed. The heuristic path-abandonment strategy outlined above is a critical performance determinant in this area.

While there is a mechanism for the TAPR-1 processor to notify the prototype implementation that a lower-level AX.25 virtual circuit has failed, so that an alternate path can be tried, there is no intrinsic mechanism to signal the failure of an upper-level TCP connection, which uses IP datagrams wrapped in AX.25 I frames (connection mode) or UI frames (connectionless mode). This is a generic problem with any end-system protocol where the peers are located physically distant from the link-level entities. Experience indicates the value of providing a two-way conduit to share control information between protocol layers may be seriously underestimated.

The prototype implementation manages processor and storage demands in relatively simple ways, which can result in considerable

inefficiencies. It is apparent that in any widely distributed version of Wiretap these demands will have to be carefully managed. As suggested above, effective provisions to purge old information, especially speculative links, are vital, as well as provisions to control the intervals between route computations, for instance as a function of link state and traffic mode.

The next step in the evolution towards a fully distributed routing algorithm is the introduction of active probing techniques. This should considerably improve the capability to discover new paths, as well as to fine-tune existing ones. It should be possible to implement an active probing mechanism while maintaining compatibility with the passive-only Wiretap, as well as maintaining compatibilty with other stations using no routing algorithms at all. It does seem that judicious use of beacons to discover and renew paths in the absence of traffic will be required, as well as some kind of echo/reply mechanism similar to the ICMP Echo/Reply support required of Internet hosts.

In order to take advantage of the flexibility provided by routing algorithms like Wiretap, it will be necessary to revise the AX.25 specification to include "loose" source routing in addition to the present "strict" source routing. Strict source routing requires every forwarding stage (callsign) to be explicitly declared, while loose source routing would allow some or all stages to be left to the discretion of the local routing agent or digipeater. One suggestion would be to devise a special collective indicator or callsign that could signal a Wiretap digipeater to insert the computed route string following its callsign in the AX.25 frame header.

A particularly difficult area for any routing algorithm is in its detection and reponse to congestion. Some hints on how the existing Wiretap mechanism can be improved are indicated in this document. Additional work, especially with respect to the hidden-station problem, is necessary. Perhaps the most useful feature of all would be a link-quality indication derived from the radio, modem or frame-level procedures (checksum failures). Conceivably, this information could be included in beacon messages broadcast occasionally by the digipeaters.

It is quite likely that the most effective application of routing algorithms in general will be at the local-area digipeater sites. One reason for this is that these stations may have off-channel trunking facilities that connect different areas and may exchange wide-area routing information via these facilities. The routing information collected by the local-area Wiretap stations could then be exchanged directly with the wide-area sites.

9. References

[1] Forney, G.D., Jr. The Viterbi Algorithm. Proc IEEE 61, 3 (March 1973), 268-278.

[2] McQuillan, J., I. Richer and E. Rosen. An overview of the new routing algorithm for the ARPANET. Proc. ACM/IEEE Sixth Data Comm. Symp., November 1979.

[3] Mills, D.L. Exterior Gateway Protocol Formal Specification. DARPA Network Working Group Report RFC-904, M/A-COM Linkabit, April 1984.

[4] Fox, T.L., (Ed.). AX.25 amateur packet-radio link-layer protocol, Version 2.0. American Radio Relay League, October 1984.

Appendix A. An Example

An example will illustrate how Wiretap constructs primary and alternate routes given candidate node and link tables. The candidate tables resulted from a scenario monitoring normal traffic on the 145.01-MHz AX.25 packet-radio channel in the Washington, DC, area during a typical 24-hour period. The node and link tables illustrated below give an idea of what the constructed data base looks like, as well as provide the basis for the example.

Figure 1 illustrates a candidate node table showing the node ID (NID), callsign and related information for each station. The Route field contains the primary route (minimum-distance path), as a string of NIDs from the origination station (NID = 0) to the destination station shown, with the exception of the endpoint NIDs. The absence of a route string indicates the station is directly reachable without the assistance of a digipeater. Note that the originating station is always the first entry in the node table, in this case W3HCF, and is initialized with defaults before the algorithm is started.

NID	Callsign	Flags	Links	Last Rec	Wgt	Route
0	W3HCF	005	26	15:00:19	255	
1	WB4APR-5	017	18	16:10:38	30	
2	DPTRID	000	3	00:00:00	210	1
3	W9BVD	005	3	23:24:33	40	
4	W3IWI	015	5	16:15:30	35	
5	WB4JFI-5	017	34	16:15:30	35	
6	W3TMZ	015	2	01:00:49	150	1
7	WB4APR-6	017	14	14:56:06	35	
8	WB4FQR-4	017	4	06:35:15	40	
9	WD9ARW	015	3	14:56:04	115	11
10	WA4TSC	015	3	15:08:53	115	11
11	WA4TSC-1	017	9	15:49:15	35	
12	KJ3E	015	4	15:57:26	155	1
13	WB2RVX	017	3	09:19:46	135	7
14	AK3P	015	2	12:57:53	185	7 15
15	AK3P-5	016	4	12:57:53	135	7
16	KC2TN	017	3	04:01:17	135	7
17	WA4ZAJ	015	2	21:41:24	240	5
18	KB3DE	015	3	23:38:16	35	
19	K4CG	015	3	13:29:14	35	
20	WB2MNF	015	2	04:01:17	180	7 16
21	K4NGC	015	3	14:57:44	90	8
22	K3SLV	005	2	03:40:01	160	1

23	KA4USE-1	017	6	14:57:44	35	
24	K4AF	005	3	12:46:38	40	
25	WB4UNB	015	2	06:45:09	240	5
26	PK64	005	3	02:50:54	40	
27	N4JOG-2	015	3	13:24:53	35	
28	KX3C	015	4	02:57:29	35	
29	W3CSG	015	4	06:10:17	115	11
30	WD4SKQ	015	3	16:00:33	35	
31	WA7DPK	015	3	01:28:11	35	
32	N4JGQ	015	3	22:57:50	35	
33	K3AEE	005	3	03:52:43	40	
34	WB3ANQ	015	3	04:01:27	140	7
35	K2VPR	015	2	12:07:51	240	5
36	G4MZF	015	3	01:38:30	35	
37	KA3ERW	015	2	03:11:17	155	1
38	WB3ILO	015	2	02:10:34	140	7
39	KB3FN-5	016	4	06:10:17	110	11
40	KS3Q	015	5	15:54:57	35	
41	WA3WUL	015	2	03:36:18	135	7
42	N3EGE	015	3	15:58:01	160	1
43	N4JMQ	015	2	08:02:58	185	7 13
44	K3JYD-5	016	5	15:58:01	155	1
45	KA4TMB	015	3	16:15:23	115	11
46	KC3Y	015	2	04:14:36	155	1
47	W4CTT	005	2	12:21:33	245	5
52	K3JYD	015	2	02:16:52	155	1
54	WA5WTF	015	2	02:01:20	240	5
55	KA4USE	005	3	23:56:02	105	23
56	N3BRQ	005	2	02:00:36	40	
57	KC4B	015	2	22:10:37	240	5
58	WA5ZAI	005	2	12:44:03	40	
59	K4UW	005	2	02:36:05	40	
60	K3RH	015	2	01:20:47	135	7
61	N4KRR	015	3	10:56:50	35	
62	K4XY	015	2	04:53:16	240	5
64	WA6YBT	015	2	05:13:07	190	7 15

Figure 1. Candidate Node Table

In the above table the Dist field shows the total distance of the primary route, the Links field shows the complexity factor, which is the number of links incident at that node (plus one), and the Last Rec field shows the time (UT) the station was last heard, directly or indirectly. The Flags field shows, among other things, which stations

have originated frames and which have digipeated them. The bits in this field, which is in octal format, are interpeted as follows (bit 0 is the rightmost bit):

Bit	Function
0	originating station
1	digipeater station
2	station heard (Last Rec column)
3	station synchronized connection

Among the 58 stations shown in Figure 1 are eleven digipeaters, all but three of which also originate traffic. All but twelve stations have either originated or digipeated a synchronized connection and only one "station" DPTRID, actually a beacon, has not been heard to either originate or digipeat traffic.

Figure 2 illustrates a candidate node table of 98 links showing the from-NID, to-NID, Flags and Age information for each link as collected. The bits in the Flags field, which is in octal format, are interpeted as follows (bit 0 is the rightmost bit):

Bit	Function
0	source
1	digipeated
2	heard
3	synchronized
4	reciprocal

From	To	Flags	Age	From	To	Flags	Age
5	0	017	0	1	0	037	5
4	0	015	0	5	4	035	0
4	1	015	28	7	0	017	60
9	5	015	60	1	5	006	56
4	7	015	60	11	0	017	24
7	15	036	62	7	13	037	60
12	1	015	71	15	14	035	62
7	16	037	70	12	5	015	71
19	0	015	61	16	20	035	70
5	11	036	60	23	0	017	60
5	24	035	73	30	0	015	71
29	11	015	69	5	29	035	73
8	21	035	67	8	5	017	67
31	0	015	72	31	5	015	72
32	0	015	74	32	5	015	69

40	5	015	17	40	0	015	19
34	7	015	70	35	5	015	62
1	40	035	74	38	7	015	71
5	36	035	72	45	5	015	0
36	0	015	72	5	30	035	14
37	1	015	70	44	5	016	14
12	44	015	17	46	1	015	69
34	1	015	72	44	1	016	70
5	23	036	60	9	11	015	79
10	11	015	60	1	6	035	72
27	5	015	61	11	1	006	83
45	11	015	76	52	1	015	71
5	2	000	14	8	0	005	76
57	5	015	75	17	5	015	75
3	0	005	74	3	5	005	74
26	5	005	71	26	0	005	74
18	5	015	74	18	0	015	74
55	5	005	73	24	0	005	62
61	0	015	63	55	23	005	73
54	5	015	71	61	5	015	63
59	0	005	71	56	0	005	71
5	7	006	71	7	60	035	72
28	0	015	71	62	5	015	69
1	7	036	70	28	5	015	71
7	41	035	70	28	1	015	71
58	0	005	62	1	22	005	70
33	7	005	70	33	0	005	70
64	15	015	69	25	5	015	67
39	10	035	68	11	39	036	68
43	13	015	65	29	39	015	68
40	7	015	62	47	5	005	62
19	23	015	61	27	0	015	61
42	1	005	23	23	21	035	60
1	2	000	5	42	44	015	14

Figure 2. Candidate Link Table

The following tables illustrate the operation of the routing algorithm in several typical scenarios. Each line in the table represents the step where an entry is extracted from the path list and new entries are determined. The "Step" column indexes each step, while the "To" column indicates the NID of the station at that step. The "Ptr" column is the index of the preceeding step along the path to the destination, while the "Hop" and "Dist" columns represent the total hop count and computed distance along that path.

Following is a fairly typical example where the destination station is not directly reachable, but several multiple-hop paths exist via various digipeaters. The algorithm finds four digipeaters: 1, 5, 11 and 39, all but the last of which are directly reachable from the originating station, to generate two routes of two hops and two of three hops, as shown below. Note that only the steps leading to complete paths are shown.

Destination: 29 Station: W3CSG

Step	NID	Ptr	Hop	Dist	Comments
0	29	0	0	0	
1	5	0	1	30	
2	11	0	1	35	
3	39	0	1	35	
4	0	1	2	235	Complete path: 0 5 29
35	0	2	2	115	Complete path: 0 11 29
37	9	2	2	115	
38	10	2	2	115	
39	1	2	2	120	
40	45	2	2	115	
41	39	2	2	110	
42	11	3	2	85	
43	10	3	2	85	
46	0	39	3	240	Complete path: 0 1 11 29
63	0	42	3	165	Complete path: 0 11 39 29

The algorithm ranks these routes first by distance and then by order in the list, so that the two-hop route at N = 35 would be chosen first, followed by the three-hop route at N = 63, the two-hop route at N = 4 and, finally the three-hop route at N = 46. The reason why the second choice is a three-hop route and the third a two-hop route is because of the extreme congestion at the digipeater station 5, which has 34 incident links.

Following is an example showing how the path-pruning mechanisms operate to limit the scope of exploration to those paths most likely to lead to useful routes. The algorithm finds one two-hop route and four three-hop routes. In this example the complete list is shown, including all the steps which are abandond for the reasons given.

Destination: 13 Station: WB2RVX

Step	NID	Ptr	Hop	Dist	Comments
0	13	0	0	0	
1	7	0	1	30	
2	43	0	1	35	No path
3	0	1	2	135	Complete path: 0 7 13
4	4	1	2	135	
5	15	1	2	130	
6	16	1	2	130	
7	34	1	2	135	
8	38	1	2	135	No path
9	60	1	2	130	No path
10	5	1	2	140	Max distance 310
11	1	1	2	130	
12	41	1	2	130	No path
13	33	1	2	140	
14	40	1	2	135	
15	5	4	3	210	Max distance 380
16	0	4	3	215	Complete path: 0 4 7 13
17	1	4	3	215	Max distance 305
18	14	5	3	180	Max hops 4
19	64	5	3	185	Max hops 4
20	20	6	3	175	Max hops 4
21	1	7	3	205	Max distance 295
22	0	11	3	250	Complete path: 0 1 7 13
23	4	11	3	255	Max distance 300
24	12	11	3	255	Max distance 295
25	40	11	3	250	Max distance 295
26	37	11	3	255	Max distance 285
27	46	11	3	255	Max distance 285
28	44	11	3	255	Max distance 280
29	34	11	3	255	Max distance 290
30	6	11	3	250	Max distance 280
31	52	11	3	255	Max distance 285
32	28	11	3	255	Max distance 295
33	0	13	3	215	Complete path: 0 33 7 13
34	0	14	3	215	Complete path: 0 40 7 13
35	5	14	3	215	Max distance 385
36	1	14	3	210	Max distance 300

The steps labelled "No path" are abandonded because no links could be found satisfying the constraints: (a) to-NID or from-NID matching the NID of the step, (b) loop-free or (c) total path distance less

than 256. The steps labelled "Max distance" are abandonded because the total distance, computed as the sum of the Dist value plus the weighted node factors, would exceed 256 as shown. The steps labelled "Max hops" are abandonded because the total hop count would exceed the minimum hop count (plus one) as shown.

Although this example shows the computations for all alternate routes, if only the primary route is required all steps with total distance greater than the minimum-distance (135) can be abandonded. In this particular case path exploration terminates after only 14 steps.

The following example shows a typical scenario involving a previously unknown station; that is, one not already in the data base. Although not strictly part of the algorithm itself, the strategy in the present system is to generate speculative paths consisting of an imputed direct link between the originating station and the destination station, together with imputed direct links between each digipeater in the data base and the destination station. The new links created will time out according to the cache-management mechanism in about fifteen minutes.

In the following example the destination station is 74, which results in the following additions to the link table:

fm-NID	To-NID	Flags	Node Type
0	74	000	Originator
1	74	000	Digipeater
5	74	000	Digipeater
7	74	000	Digipeater
8	74	000	Digipeater
11	74	000	Digipeater
13	74	000	Digipeater
15	74	000	Digipeater
16	74	000	Digipeater
23	74	000	Digipeater
39	74	000	Digipeater
44	74	000	Digipeater

There are eleven digipeaters involved, not all of which may be used. The resulting primary route and five alternate routes are shown below. Note that only five of the eleven digipeaters are used. The remainder were either too far away or too heavily congested. Note that only the list entries leading to complete paths are shown.

Destination: 74 Station: CQ

Step	NID	Ptr	Hop	Dist	Comments
0	74	0	0	0	
1	0	0	1	90	Complete path: 0 74
2	1	0	1	90	
4	7	0	1	90	
5	8	0	1	90	
6	11	0	1	90	
7	13	0	1	90	
8	15	0	1	90	
9	16	0	1	90	
10	23	0	1	90	
11	39	0	1	90	
12	44	0	1	90	
13	0	2	2	210	Complete path: 0 1 74
29	0	4	2	195	Complete path: 0 7 74
44	0	5	2	150	Complete path: 0 8 74
45	0	6	2	170	Complete path: 0 11 74
60	0	10	2	155	Complete path: 0 23 74