Wiretap: An Experimental Multiple-Path Routing Algorithm[1 2]

David L. Mills
Electrical Engineering Department
University of Delaware

Abstract

This paper introduces Wiretap, an experimental routing algorithm which computes maximum-likelihood diversity routes for packet-radio stations sharing a common 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. The algorithm is presented as an example of maximum-likelihood routing and database management techniques useful for richly connected networks of mobile stations. Of particular interest are the mechanisms to compute, select, rank and cache a potentially large number of speculative routes when only limited computational resources are available.

A prototype implementation has been constructed and tested for the AX.25 packet-radio channel now in widespread use in the amateur-radio community. Its design is similar in many respects to the SPF algorithm used in the ARPANET and NSFNET backbone networks, and is in fact a variation of the Viterbi algorithm, which constructs maximum-likelihood paths on a graph according to a weighted sum of factors assigned to the nodes and edges.

Keywords: adaptive routing, diversity routing, packet radio, Viterbi algorithm

1. Introduction

This paper describes the design, implementation and initial testing of the Wiretap algorithm, which computes maximum-likelihood diversity routes for the amateur AX.25 packet-radio channel [3]. Wiretap operates in real time using passive monitoring of AX.25 frames transmitted on the channel and builds a dynamic database which can be used to construct a set of paths ordered by decreasing likelihood, as determined by a technique based on the Viterbi algorithm [2]. The Wiretap algorithm is similar in function to the shortest-path-first (SPF) routing algorithms used in the ARPANET and NSFNET backbone networks [4] and to the routing algorithms used in other packet-radio systems [1], but is specifically intended to produce multiple paths based on real-time measured characteristics of the channel itself.

The principal advantage in the use of Wiretap is that packet-repeater (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 active probes to discover quiescent stations and clustering techniques to manage the size of the database.

The AX.25 channel operates in CSMA contention mode at HF and VHF radio frequencies using AFSK/FM modulation at 300 or 1200 bps. The AX.25 protocol itself is similar to the X.25 link-layer protocol LAPB, but with an extended frame header including a string of radio callsigns, selected by the originator, designating the complete source route between two end stations, possibly via one or more intermediate digipeaters. Most AX.25 implementations can operate simultaneously as end stations and as digipeaters in datagram or multiple virtual-circuit mode, but have no provisions to specify the source route other than manually.

Since the 145.01-MHz AX.25 packet-radio channel in the Washington, D.C., area is very active and carries a good deal of traffic under punishing conditions, it was considered a sufficiently heroic environment for a convincing demonstration of a prototype Wiretap algorithm. The implementation provides primary and alternate diversity routes for both virtual circuits and datagrams, can route around congested areas and can change routes during a connection. This paper, which is an updated and condensed version of [5], presents a status report and overview of the prototype implementation.

1. Reprinted from: Mills, D.L. Wiretap: an experimental multiple-path routing algorithm. *ACM Computer Communication Review 19, 1* (January 1989), 85-98.

2. Sponsored by: Defense Advanced Research Projects Agency contract number N00140-87-C-8901 and by National Science Foundation grant number NCR-86-12015.

The prototype implementation is part of a TCP/IP driver for the LSI-11 processor running the Fuzzball operating system [7] and is connected via 4800-bps serial line to a terminal node controller (TNC) which controls the radio equipment in both AX.25 virtual-circuit and datagram modes. The TNC firmware produces as an option a monitor report for each received frame of a selected type, including AX.25 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 station is W4CQI. The frame has been repeated 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 an end station, extending through one or more digipeaters and terminating at another end station. Each link is characterized by a set of factors such as delay, throughput or reliability that can be computed or estimated. Wiretap computes several intrinsic factors for each link and updates the routing database, 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 frame 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 [2], it is convenient to represent distance as a logarithmic transformation of probability; however, in Wiretap the underlying probabilities are not determined directly, but estimated on a heuristic basis.

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

Note that Wiretap in effect constructs maximum-likelihood paths in the direction from the destination station to the Wiretap station, then computes the reciprocal routes from the Wiretap station to the destination station. The expectation is that the destination station also runs a routing algorithm which computes its own reciprocal routes (i.e. the direct routes from the Wiretap station). However, the routing databases 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, suboptimal 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 includes two procedures: the wiretap procedure, which extracts information from received monitor headers and builds the routing database, and the routing procedure, which calculates paths using the contents of the database. The database includes two tables: the node table and link table. 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 routes and other miscellaneous information. Each entry is indexed by node ID (NID), which is used elsewhere in the database 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. Both tables are dynamically managed using a cache algorithm based on a weighted least-recently-used replacement mechanism described later.

The example discussed in the Appendix 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

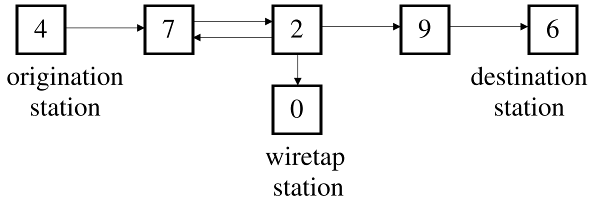
60 entries in the node table and 100 entries in the link table.

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 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 suboptimal 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 Procedure

The wiretap procedure is called to process each monitor header. It extracts each callsign from the header in turn and searches the node table for the corresponding callsign, making a new entry 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 indicated in the string or its reciprocal, making a new entry if not already there.

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 stations 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 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 repeat them. Station 4 is observed to originate frames and station 7 to repeat them, but station 9 is only a presumptive repeater 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 "repeated."

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 procedure and stored in the node table as it adds entries to the link table. The second, not yet completely implemented, is a dynamic measure of node congestion which tallies the number of NID references during the most recent time interval of specified length.

5. Factor Computations and Weights

The data items produced by the wiretap procedure are processed to produce a set of factors that can be used by the routing procedure to develop maximum-likelihood 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 must have the following properties:

1. All link and node factors must be positive, monotone functions which increase in value as system performance degrades from optimum.

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

3. The criteria used to determine node factors must 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 necessary 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 maximum-likelihood routes over all choices in factor computations and weights is unchanged. Recall that the wiretap procedure 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 procedure to generate the factors shown below in Table 1 and Table 2

Factor	Weight	Name	How determined
f0	3	hop	1 for each link
f1	50	unverified	1 if not heard either direction
f2	5	non-reciprocal	1 if not heard both directions

Table 1. Link Factors

In the case of 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 link is not marked "heard" (heard in either direction), while the "non-reciprocal" factor is assigned as one if the link is not marked "reciprocal" (heard in both directions). The "unsynchronized" factor is assigned as one if the link is not marked "synchronized" is (no I or S frames observed in either direction).

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

Table 2. Node Factors

In the case of intermediate-node factors, the "complexity" factor is computed as the number of links incident at the node plus one, while the "congestion" factor will be computed as the number of frames heard in the last minute. The "repeated" factor is assigned as one if the node is only a source (i.e. no repeated frames have been heard from it). For the purposes of path-distance calculations, the end-node factors are taken as zero, since their contribution to any path would be the same.

6. The Routing Procedure

The dynamic database built by the wiretap procedure is used by the routing procedure to compute routes as required. Ordinarily, this needs to be done only when the first frame to a new destination is sent and at regular intervals thereafter (in future the intervals may be modulated by congestion thresholds, etc.). The technique used is a variation of the Viterbi algorithm [2], which operates by constructing a set of candidate paths (survivors) 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 as in [1] the primary route is selected from among the minimum-distance paths and the alternate routes selected in order of increasing distance of the remaining paths.

The routing procedure operates using a linked list of entries derived from the link table. Each list entry includes the NID of the current node, a pointer to the preceding node on the path to the root plus the total hop count and distance from the node to the root:

[NID, pointer, hop, distance].

The procedure starts with the list containing only the root entry [root-NID, 0, 0, 0], where root-NID represents the final destination station, 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+link-distance],

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 points to the old entry, while the link-distance is computed from the factors and weights as described previously. The procedure continues to generate new entries until no further entries remain to be processed or the maximum hop count or distance are exceeded, as explained below.

In the Viterbi algorithm when survivors merge at a node, all but one of the survivors are abandoned. If only one of the minimum-distance paths is required, the wiretap procedure does the same; however, in the more general case where alternate paths are required, all non-looping paths are potential survivors and must be retained. 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 root are suppressed and paths with hop counts exceeding (currently) eight or distances exceeding 255 are abandoned (pruned).

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 routing procedure remembers the minimum distance and minimum hop count of the complete paths found as it proceeds. When only the primary route is required, a survivor is pruned if the distance exceeds the minimum distance or the hop count exceeds the minimum hop count plus one. When alternate routes are required the hop-count test is used, but the minimum-distance test is not. The assignment of factor computations and weights is intended to favor minimum-hop paths under most conditions, but to allow a survivor to grow by no more than one additional hop under conditions of extreme congestion. Thus, the minimum-distance path may not be a minimum-hop path. Obviously, the resources required can escalate dramatically, unless an effective pruning technique such as this are used.

Some idea of the time and space required by the prototype implementation can be gathered from the primary and secondary routes for the example in the Appendix with 58 nodes and 98 links. The linked list uses about 30 entries on average, but occasionally exceeds 100 entries. The prototype procedure requires 316 milliseconds on an LSI-11/73 to calculate the 58 primary routes to all nodes and 1416 milliseconds to calculate the 201 combined primary and alternate routes to all nodes.

The Wiretap routing algorithm can be compared to the Tier algorithm developed for the DARPA Survivable Radio Network (SURAN) program [1], which is a variant of the Bellman-Ford algorithm described in [6]. The Tier algorithm is designed to operate in a distributed manner where each station broadcasts a routing vector to its neighbors on a periodic or event-triggered basis. The routing metric is based on hop count and station number (to avoid loops), and with primary and alternate routes ranked as in Wiretap. As described in [1] the Tier metric includes neither the link factor computations and weights nor the pruning techniques to control computational overheads as mentioned above. On the other hand, the Tier algorithm is designed to avoid loops caused by database inconsistencies due to hidden stations and unreliable links, while Wiretap has no overt provisions to avoid this.

The most important difference between the Tier and Wiretap approach is that the Tier algorithm assumes all stations will cooperate to broadcast routing vectors, which can consume considerable channel overhead and be heard by potential jammers. On the other hand, Wiretap stations operate independently in receive-only mode and do not reveal the database except indirectly by the routes they use.

7. Database 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 database. 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 database 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 passive monitoring in the management of the AX.25 packet-radio channel. While the experience using the prototype implementation is encouraging, considerable work needs to be done in the optimization of factor computations and weight assignments. For this to be done effectively, more experience needs to be gained in the day-to-day operation of the prototype during which various combinations of weight assignments can be evaluated.

As described in the Appendix, when attempting to compute a route to a previously unknown destination station, a simple but effective heuristic is to generate speculative paths by adding synthetic links between the destination and the Wiretap station and between the destination and the known digipeaters. This heuristic is used in the datagram mode to generate primary routes and in the initial-connection phase of virtual-circuit mode to generate both primary and alternate routes. While in practice this heuristic works very well, it requires significant computational resources, due to the large number of possible paths ranging from reasonable to outrageous, so that the pruning strategy outlined previously can be a critical performance determinant in both modes.

While there is a mechanism for the TNC provide notification that an AX.25 virtual circuit has failed, so that an alternate route 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 backward compatibility, with previous algorithms or 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.

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 node (digipeater) to be explicitly declared, while loose source routing would allow some or all forwarding nodes to be selected dynamically as the frame progresses along the route. One suggestion is to devise a special collective indicator or callsign that would signal a designated digipeater to insert a computed source route following its callsign in the AX.25 frame header.

A particularly difficult issue for any routing algorithm is the detection and response to congestion. Some hints on how the existing Wiretap mechanism can be improved are indicated in this paper. 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 designated 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. Belghith, A., and L. Kleinrock. A distributed routing scheme with mobility handling in stationless multi-hop packet-radio networks. *Proc. SIGCOMM Symposium* (March 1983), 101-108.

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

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

4. McQuillan, J., I. Richer and E. Rosen. An overview of the new routing algorithm for the ARPANET. *Proc. ACM/IEEE Sixth Data Communications Symposium* (November 1979).

5. Mills, D.L. An experimental multiple-path routing algorithm. DARPA Network Working Group Report RFC-981. M/A-COM Linkabit, March 1986.

6. Bertsekas, D., and R. Gallager. Data Networks. Prentice-Hall, Englewood Cliffs, NJ, 1987.

7. Mills, D.L. The Fuzzball. *Proc. ACM SIGCOMM 88 Symposium* (Palo Alto, California, August 1988), 115-122.

10. Appendix. An Example

An example will illustrate how Wiretap constructs primary and alternate routes given candidate node and link tables resulting from normal traffic monitoring on the 145.01-MHz AX.25 packet-radio channel in the Washington, D.C., area during a typical 24-hour period.

Figure 1 illustrates a node table showing the node ID (NID), callsign and related information for each station. The bits in the Flags field (octal) are interpreted starting from the rightmost: "originating station," "repeater station," "station heard" and "station synchronized connection." The Links field shows the "complexity factor," which is the number of links incident at that node (plus one), the Last Rec field shows the time (UTC) the station was last heard, directly or indirectly and the Weight field shows the total distance of the primary route. Finally, the Route field shows the primary route (minimum-distance path), as a string of NIDs from the origination station W3HCF (NID 0) via the route shown to the destination station NID. The absence of a route indicates the station is directly reachable without the assistance of a digipeater.

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 repeated a synchronized connection and only one (DPTRID, actually a beacon), has not been heard to either originate or repeat traffic.

Figure 2 illustrates a 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 (octal) are interpreted starting from the rightmost: "source," "repeated," "heard on at least one direction," "synchronized" and "heard on both directions." The Age field increments in minutes until reaching 60, then in hours after that.

NID	Callsign	Flags	Links	Last Rec	Weight	Route
0	W3HCF	05	26	15:00:19	255	
1	WB4APR-5	17	18	16:10:38	30	
2	DPTRID	00	3	00:00:00	210	1
3	W9BVD	05	3	23:24:33	40	
4	W3IWI	15	5	16:15:30	35	
5	WB4JFI-5	17	34	16:15:30	35	
6	W3TMZ	15	2	01:00:49	150	1
7	WB4APR-6	17	14	14:56:06	35	
8	WB4FQR-4	17	4	06:35:15	40	
9	WD9ARW	15	3	14:56:04	115	11
10	WA4TSC	15	3	15:08:53	115	11
11	WA4TSC-1	17	9	15:49:15	35	
12	KJ3E	15	4	15:57:26	155	1
13	WB2RVX	17	3	09:19:46	135	7
14	AK3P	15	2	12:57:53	185	7 15
15	AK3P-5	16	4	12:57:53	135	7
16	KC2TN	17	3	04:01:17	135	7
17	WA4ZAJ	15	2	21:41:24	240	5
18	KB3DE	15	3	23:38:16	35	
19	K4CG	15	3	13:29:14	35	
20	WB2MNF	15	2	04:01:17	180	7 16
21	K4NGC	15	3	14:57:44	90	8
22	K3SLV	05	2	03:40:01	160	1
23	KA4USE-1	17	6	14:57:44	35	
24	K4AF	05	3	12:46:38	40	
25	WB4UNB	15	2	06:45:09	240	5
26	PK64	05	3	02:50:54	40	
27	N4JOG-2	15	3	13:24:53	35	
28	KX3C	15	4	02:57:29	35	
29	W3CSG	15	4	06:10:17	115	11

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

Figure 1. Node 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 Pointer column indexes the preceding step along the path to the root. The NID column identifies the station at each step, while the Hop and Distance columns show the total hop count and computed distance along the path to the root.

Following is a 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, and generates two routes of two hops and two of three hops, as shown below. Note that only the steps leading to complete paths are shown.

Step	NID	Ptr	Hop	Dist	Comments
0	29	0	0	0	Destination: W3CSG
1	5	0	1	30	
2	11	0	1	35	
3	39	0	1	35	
4	0	1	2	235	Complete path: 0
35	0	2	2	115	Complete path: 0
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
63	0	42	3	165	Complete path: 0

The algorithm ranks these routes first by distance and then by order in the list, so that the two-hop route at step 35 would be chosen first, followed by the three-hop route at step 63, the two-hop route at step 4 and, finally the three-hop route at step 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 in Figure 1.

From	To	Flags	Age
5	0	17	0
1	0	37	5
4	0	15	0
5	4	35	0
4	1	15	28
7	0	17	60
9	5	15	60
1	5	06	56
4	7	15	60
11	0	17	24
7	15	36	62
7	13	37	60
12	1	15	71
15	14	35	62
7	16	37	70
12	5	15	71
19	0	15	61
16	20	35	70
5	11	36	60
23	0	17	60
5	24	35	73
30	0	15	71
29	11	15	69
5	29	35	73
8	21	35	67

From	To	Flags	Age
8	5	17	67
31	0	15	72
31	5	15	72
32	0	15	74
32	5	15	69
40	5	15	17
40	0	15	19
34	7	15	70
35	5	15	62
1	40	35	74
38	7	15	71
5	36	35	72
45	5	15	0
36	0	15	72
5	30	35	14
37	1	15	70
44	5	16	14
12	44	15	17
46	1	15	69
34	1	15	72
44	1	16	70
5	23	36	60
9	11	15	79
10	11	15	60
1	6	35	72

From	To	Flags	Age
27	5	15	61
11	1	06	83
45	11	15	76
52	1	15	71
5	2	00	14
8	0	05	76
57	5	15	75
17	5	15	75
3	0	05	74
3	5	05	74
26	5	05	71
26	0	05	74
18	5	15	74
18	0	15	74
55	5	05	73
24	0	05	62
61	0	15	63
55	23	05	73
54	5	15	71
61	5	15	63
59	0	05	71
56	0	05	71
5	7	06	71
7	60	35	72
28	0	15	71

From	To	Flags	Age
62	5	15	69
1	7	36	70
28	5	15	71
7	41	35	70
28	1	15	71
58	0	05	62
1	22	05	70
33	7	05	70
33	0	05	70
64	15	15	69
25	5	15	67
39	10	35	68
11	39	36	68
43	13	15	65
29	39	15	68
40	7	15	62
47	5	05	62
19	23	15	61
27	0	15	61
42	1	05	23
23	21	35	60
1	2	00	5
42	44	15	14

Figure 2. Link Table

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 abandoned for the reasons given in the Comments column.

Step	NID	Ptr	Hop	Dist	Comments
0	13	0	0	0	Destination: WB2RVX
1	7	0	1	30	
2	43	0	1	35	No path
3	0	1	2	135	Complete path: 0
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
17	1	4	3	215	Max distance
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
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
34	0	14	3	215	Complete
35	5	14	3	215	Max distance 385
36	1	14	3	210	Max distance 300

The steps labelled "No path" are abandoned because no links could be found in Figure 2 with one end matching NID and the other end matching a NID not already on the path to the root. The steps labelled "Max distance" are abandoned because the total distance shown, computed as the sum of the Distance value plus the weighted node factors, exceeds 256. The steps labelled "Max hops" are abandoned because the total hop count shown exceeds the minimum hop count plus one.

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 abandoned. In this particular case path exploration would then terminate after only 14 steps.

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

For instance, in the case of Figure 1, the following links to a new station (NID 74) would be added to Figure 2: 0, 1, 5, 7, 8, 11, 13, 15, 16, 23, 39 and 44. The resulting primary route and five alternate routes are shown below (only the steps leading to complete paths are shown). Note that only five of the eleven digipeaters are used, since the remainder were either too distant or too heavily congested.

Step	NID	Ptr	Hop	Dist	Comments
0	74	0	0	0	Destination: CQ
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