PATHALIAS

or

The Care and Feeding of Relative Addresses

Peter Honeyman

Computer Science Department
Princeton University
Princeton, New Jersey 08544
`princeton!honey`

*Steven M. Bellovin**

AT&T Bell Laboratories
Murray Hill, New Jersey 07974
`ulysses!smb`

ABSTRACT

Pathalias computes electronic mail routes in environments that mix explicit and implicit routing, as well as syntax styles. We describe the history of *pathalias*, its algorithms and data structures, and our design decisions and compromises.

Pathalias is guided by a simple philosophy: get the mail through, reliably and efficiently. We discuss the principles of routing in heterogeneous environments necessary to make this philosophy a reality.

HISTORY AND OVERVIEW

UUCP,[1] the basic networking component of UNIX,[2] is the backbone of a widespread store-and-forward network. Because setting up new connections is easy, and does not require the intervention of a central administrator, the network has no regular topology. Mail routing is explicitly specified by users. That is, a user who wishes to send mail to `hostb` using `hosta` as a relay would write

```
mail hosta!hostb!user
```

When the UUCP network was small and the average connectivity was high, explicit routing was a minor annoyance at worst. Most paths were direct, and only a tiny fraction involved more than one or two hops, so remembering proper paths was easy.

Then came USENET.[3] For several reasons, UUCP routes soon became a major headache. First, many of the universities on USENET had a low degree of connectivity to other UNIX sites, typically with only two or three long-distance links. Second, USENET readers tended to reply along the USENET paths; these were rarely optimal, and were sometimes unusable. Third, as other networks were used for USENET transport,

* Much of the work was performed at the Department of Computer Science, University of North Carolina at Chapel Hill.

1 D.A. Nowitz and M.E. Lesk, ‘‘A Dial-Up Network of UNIX Systems,’’ in *UNIX Programmer’s Manual*, Seventh Ed., 1979.

2 UNIX is a trademark of AT&T Bell Laboratories.

3 S.M. Bellovin and M. Horton, ‘‘USENET — A Distributed, Decentralized News System,’’ unpublished manuscript, 1986.

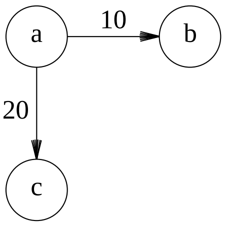
mail reply syntax became complicated by the variety of standards in use.

As USENET grew, it became clear that the UUCP network needed a routing tool, one that took as input a network connectivity graph and generated usable paths to every known destination. *Pathalias* is such a tool. Given a description of the connectivity of the UUCP network, it produces a ‘‘least cost’’ path to every known site.

Of course, any such effort relies heavily on the quality of the connectivity data. At first, gathering such data was a difficult administrative problem. Very few system administrators were willing to spend time compiling lists of neighbors and associated cost data. Some connections could be inferred from USENET maps, but these data were unreliable and lacked cost estimates. Worse, they tended to understate the connectivity of the network, putting more load on coöperative sites. Because the data were often contradictory and error-filled, it was necessary to inspect and edit the data manually. Thanks to the USENIX Association's UUCP-mapping project,[4] the picture is much brighter today, with timely and accurate data widely available on USENET.

INPUT

The input to *pathalias* is a description of a directed graph that models the connection topology of electronic networks. Each edge is assigned a non-negative cost value and a ‘‘routing operator’’; the latter shows what character is used for mail-routing, and whether it appears to the right or left of the destination host. For example, the graph



is described as follows, assuming that host `a` uses the UUCP syntax convention of `host!user` for network mail:

```
a      b(10), c(20)
```

If host `a` uses the ARPANET syntax of `user@host`, the description is

```
a      @b(10), @c(20)
```

The `@` itself indicates the character used for building an address; its position indicates that the host name is on the right. Thus, the default case may be written explicitly:

```
a      b!(10), c!(20)
```

Many sets of hosts are fully connected, *i.e.*, every host in the set talks to every other. To avoid the necessity of explicitly listing every connection, *pathalias* supports a network notation. Thus,

```
dopey   grumpy(10), sleepy(10)
grumpy  dopey(10), sleepy(10)
sleepy  grumpy(10), dopey(10)
```

can be written as

```
UNC-dwarf    = {dopey, grumpy, sleepy}(10)
```

[4] M.R. Horton, K. Summers-Horton, and B. Kercheval, ‘‘Proposal for a UUCP/USENET Registry Host,’’ in *Proc. Summer USENIX Conference*, Salt Lake City, 1984.

where `UNC-dwarf` is the name given to that network.

As is common on UNIX, data for *pathalias* may be read from the standard input or from a list of input files. Typically, each input file represents the data for a given machine or site; file boundaries have semantic implications in the treatment of ‘‘private’’ names and in resolving duplicate connection data.

The definition of a satisfactory cost metric proves troublesome. Possible metrics include the actual telephone cost of a connection, the nominal frequency of contact (many academic sites are passive — rather than calling out, they wait for other sites to call them), or the transmission speed of a connection. We adopt pragmatic approach: given a choice of paths, we attempt to choose the one that experienced users prefer, as exemplified by existing network traffic. The cost measure is tuned so that *pathalias* produces routes agreeing with these choices.

Using this metric helps balance conflicting concerns. For example, long distance costs often matter less to large corporations than to universities. On the other hand, sites with autodialers have far more freedom to connect with whom they wish. A pragmatic metric also accounts for differences in the reliability of sites; UUCP was not always as reliable as it is today. Actual transmission speed is less important than one might assume; call setup time and the time between calls tend to be the dominant factors, at least for mail messages.

With the basis for a metric in hand, symbolic names like **HOURLY**, **DAILY**, *etc.* are assigned numeric values. Early on, these numbers were juggled until, in the estimation of experienced users, the paths produced were reasonable.

Dial-up service is specified as **DEMAND** for a site that is called whenever there is traffic, or its high-grade kin **DIRECT**, for local phone calls. In the early days, the odds of completing a call were disappointingly low; port contention, line noise, and difficulties with UNIX’s baud-rate switching were common problems. (This judgement may have been unduly colored by local experience. Several important sites had few dial-in lines, or were served by antiquated telephone switching equipment.) **DEDICATED** connections — two machines hard-wired together — are considered much higher-grade. A complete table is shown below.

Symbol	*Value*
LOCAL	25
DEDICATED	95
DIRECT	200
DEMAND	300
HOURLY	500
EVENING	1800
POLLED	5000
DAILY	5000
WEEKLY	30000

Costs can be expressed as arbitrary arithmetic expressions, mixing numbers and symbolic values. For example, **HOURLY***3 describes a connection that is completed once every three hours.

In theory, factors that influence cost are additive; in practice, experience shows that the per-hop overhead in time and reliability is so high that it is important to keep paths short. Thus, for example, **DAILY** is 10 times greater than **HOURLY**, instead of 24.

OUTPUT

Although it would be convenient to compute the path to a destination as needed, the cost of the calculation is prohibitively expensive. Consequently, *pathalias* precomputes paths to all destinations listed in the input data; these paths are retrieved as needed. The string `%s` is included in output paths as a marker to indicate where the user name should be inserted. Use of such a marker enables the generated path to be used directly as a format string for *printf*. Consider the following input data (a simplified portion of the map from 1981):

```
unc       duke(HOURLY), phs(HOURLY*4)
duke      unc(DEMAND), research(DAILY/2), phs(DEMAND)
phs       unc(HOURLY*4), duke(HOURLY)
research  duke(DEMAND), ucbvax(DEMAND)
ucbvax    research(DAILY)
ARPA      = @{mit-ai, ucbvax, stanford}(DEDICATED)
```

If run from `unc`, the following output is produced by *pathalias*:

```
0       unc             %s
500     duke            duke!%s
800     phs             duke!phs!%s
3000    research        duke!research!%s
3300    ucbvax          duke!research!ucbvax!%s
3395    mit-ai          duke!research!ucbvax!%s@mit-ai
3395    stanford        duke!research!ucbvax!%s@stanford
```

There are several points worth noting about the output. First, all generated paths route mail through `duke`, despite the presence of a direct connection to `phs` from `unc`. The reason for this is obvious, given the cost difference on the links to `duke` and `phs`.

Second, mail to ARPANET sites employs mixed-syntax addressing; the path to `ucbvax` uses UUCP conventions (i.e., the host name on the left, delimited by an '!'), while the ARPANET portion has the host name on the right, delimited by an '@'.

Finally, output from *pathalias* is a simple linear file, in the UNIX tradition. If desired, a separate program may be used to convert this file into a format appropriate for rapid database retrieval.

DATA STRUCTURES

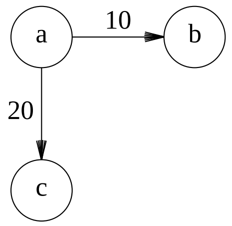
Pathalias runs in three phases: parse the input, build a shortest path tree, and print the routes. Each phase manipulates an in-memory representation of a directed graph. Before describing these computations, we describe the basic data structures.

Graph representation

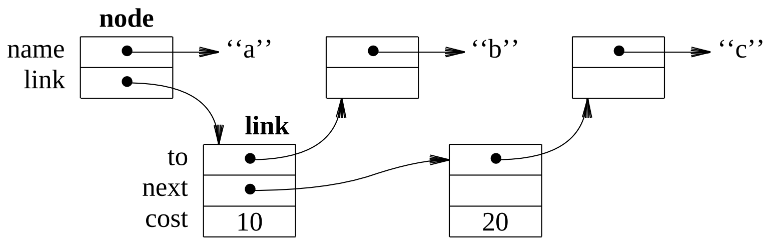
We assume that the world can be modeled as a set of hosts and networks, called *nodes*, with communication links among them. This in turn is modeled by a directed graph, with vertices representing hosts and networks, and edges representing communication links. Edges are weighted with a non-negative cost, and labeled with information describing the syntax used in building addresses from a host to its neighbor. In the input phase, *pathalias* builds an adjacency list representation of the host connectivity graph.

A node is represented by a structure consisting mostly of pointers and flags. One of the fields in a node is a pointer to a singly-linked list of adjacent hosts. A list element, called a *link*, contains a pointer to the next link on the list, a pointer to the destination host on the edge it represents, a non-negative cost, and some flags.

Thus we represent the graph



with these data structures



Networks

An accurate model must take into account various networks, ranging from the ARPANET to local Ethernets. A network has the property that its member hosts share a common name space. We model this with a fully connected graph, or *clique*.

A clique with n vertices contains about n^2 edges, so with over 2,000 hosts in the ARPANET we are faced with millions of edges. To avoid a quadratic explosion in time and space complexity, we represent a network as a single node, with a pair of edges between the network and each member. The following figure depicts this representation (on an undirected graph).

While network declarations also show edge weights and labels, the weight applies only to the edges originating at network members; the weight of edges from the network node to its members is zero. As an analogy, consider automobile toll collection by the Port Authority of New York and New Jersey: you pay to get into the City, but you get back to Jersey for free. Here, you pay to get onto a network, but you get off for free. This preserves the cost structure of the clique representation.

PARSING

Parsing is done with *yacc*.[5] We use syntax-directed translation to support a rich syntax with edge weights and labels, aliases, networks, and accommodation of host name collisions. We experimented with *lex*[6] for transforming the raw input into lexical tokens, but were disappointed with its performance: half the run time was spent in the scanner. Since our input tokens are easy to recognize, we built a simple scanner and

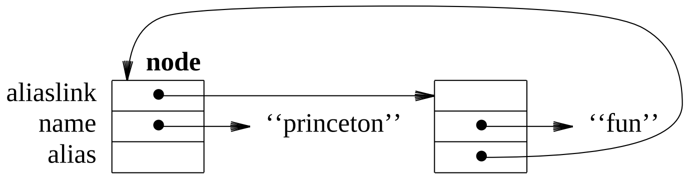
5 S.C. Johnson, ‘‘YACC — Yet Another Compiler Compiler,’’ in *UNIX Programmer’s Manual,* Seventh Ed., 1979.

6 M.E. Lesk and E. Schmidt, ‘‘LEX — Lexical Analyzer Generator,’’ in *UNIX Programmer’s Manual,* Seventh Ed., 1979.

cut the overall run time by 40%.

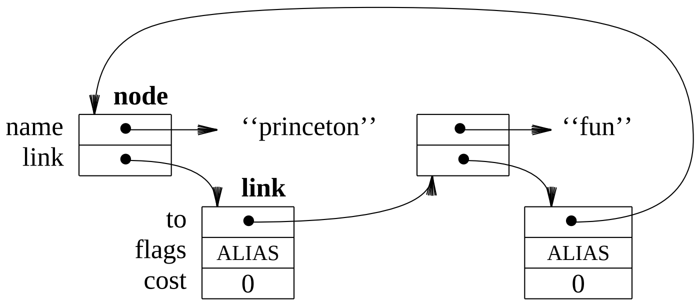
Aliases

Host name aliases are useful when a host is known by several names, or when a host name changes. Originally, aliases were used to assign a set of nicknames to a host. One name, called the *primary* host name, was used in routes; the other names were synonyms for the primary name. Under this model, the graph representation requires two pointers in each node structure: one to link all the aliases of a node onto a list, and one to show the primary host name for the node. For example, a host `princeton` with nickname `fun` has this graph representation.



This treatment does not adequately address the problem of hosts with different names on different networks. For example, the ARPANET host `nosc` has UUCP name `noscvax`. A route by way of the ARPANET must use the former, while a route by way of UUCP must use the latter. Selecting the primary host name requires predicting the shortest path *a priori*.

To address this problem, we discard the notion of a primary host name and treat all aliases as equal. A pair of zero cost edges connects aliases, giving the following as the representation of the above example:



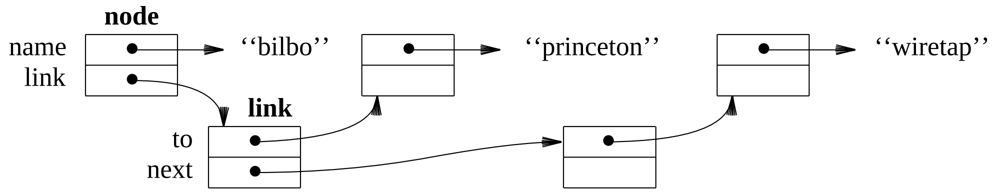
The critical point is that aliases are a property of edges, not vertices. When a host is known by several names, this alias mechanism assures that the name used in a path is the one understood to a host’s predecessor.

Host name collisions

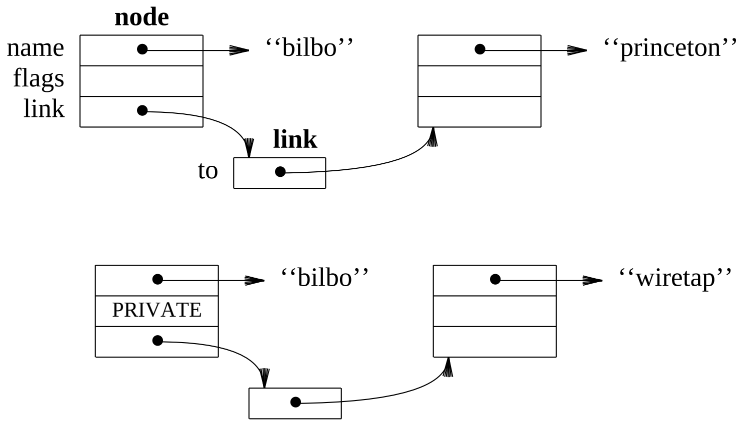
Among the hosts that rely on UUCP as their only means of communication, there is no way to prevent administrators from choosing a host name already in use, and such conflicts do occur. Consequently, *pathalias* may mistakenly merge connection information for distinct hosts with identical names.

We address this difficulty with the ‘‘private’’ declaration, which narrows the scope of connection declarations. Normally, the scope of a host name is global, ranging across all input files. The scope of a private declaration extends to the end of the file in which it is declared. If a host is declared private, it is assumed to be distinct from identically named hosts mentioned outside this scope.

For example, suppose there are two machines named `bilbo`. If the input shows a link from `bilbo` to `princeton` and another link from `bilbo` to `wiretap`, *pathalias* builds these data structures:



If the latter instance of `bilbo` is declared private, these data structures are built instead



Of course, the two declarations must be in different files for this to succeed.

Memory allocation woes

The volume of input data presented to *pathalias* can be overwhelming. USENET maps contain over 5,700 nodes and 20,000 links, while ARPANET, CSNET, and BITNET add another 2,800 nodes and 8,000 links. Since nodes and links are dynamically allocated, the selection of a memory allocator is critical to good performance.

We experimented with several techniques[7] covering a broad spectrum of time-space tradeoffs for memory allocation. We discovered that a buffered *sbrk* scheme for allocation, with no attempt to re-use freed space, gives superior performance in both time and space. This is due to the allocation/freeing pattern of *pathalias*. Most allocation takes place during the parsing phase, with very little space freed. After parsing, only minuscule amounts of space are allocated, while just about everything is freed. Thus memory allocators that attempt to coalesce when space is freed simply waste time (and space). For portability to segmented architectures, we use the C library *malloc* to obtain space for our memory allocator, since machines with small (64 kbyte) segments frequently impose severe constraints on the use of *sbrk*.

Hash table management

Host names are stored in a hash table that employs open addressing and double hashing. Given a host name, we calculate an integer key k using bit-level shifts and exclusive-ors. The primary hash function is the remainder of k divided by the (prime) hash table size, T. For the secondary hash function, we do not use the oft-suggested $1+(k \bmod T-2)$,[8] as this results in anomalous behavior (that we cannot explain); rather, we use the inverse $T-2-(k \bmod T-2)$.[9]

---

7 Implementations f, b, s, and d described in D.G. Korn and K-P Vo, ‘‘In Search of a Better Malloc,’’ in *Proc. Summer USENIX Conference*, Portland, 1985.

8 D.E. Knuth, *The Art of Computer Programming*, Vol. III, *Sorting and Searching*, Addison Wesley, Reading MA, 1973.

9 R. Sedgewick, *Algorithms*, Addison Wesley, Reading MA, 1983.

We cannot know *a priori* how many hosts n will be declared in the input, so we use a rehashing scheme, iteratively increasing T. When the load factor $\alpha = \frac{n}{T}$ exceeds a high-water mark α_H, a new table is allocated, the entries in the old table are inserted into the new table, and the old table is discarded. We use 0.79 for α_H, as this gives a predicted ratio of 2 probes per access when the table is full.[10]

The new table size can be chosen by increasing T by some factor δ, in which case the sequence of primes forms a geometric progression. If δ is too large, say $\delta=2$,[11] an excessive amount of space is wasted when the total number of hosts happens to be slightly more than $\alpha_H T$, for a given value of T. Since we want the new table to be ''large enough,'' but not ''too large,'' we experimented with an arithmetic sequence for the list of candidates for T, and employed a low-water value, α_L. When α exceeded α_H, we searched the list for a new value of T that satisfied $\alpha<\alpha_L$. This is equivalent to increasing the table size by a factor of $\delta = \frac{\alpha_H}{\alpha_L}$; for α_L we used 0.49, as this gave a value of δ close to the golden ratio. In the current implementation, we abandon α_L altogether and maintain a Fibonacci sequence of primes (more or less), which also follows the golden ratio. This simplifies the computation of table sizes without changing the behavior.

Discarding the old hash table is among the few opportunities for re-using space during the parsing phase. Rather than freeing the old tables, which can range from 4 kbytes to 32 kbytes (or more), they are placed on a list and made available to our memory allocator for later use.

CALCULATING SHORTEST PATHS

The obvious algorithm for computing shortest paths in a directed graph with non-negative edge weights is *Dijkstra's algorithm*.[12] The input is a weighted graph and a distinguished vertex, called the *source*. The output is a set of paths, one for each vertex, each starting at the source.

Dijkstra's algorithm computes minimal cost paths, where the cost of a path is the sum of the weights of the edges along the path. That is, for any vertex, the path computed by the algorithm has cost that is at least as small as any other path from the source to that vertex.

The graph described by the USENET data is sparse, *i.e.*, the number of edges e is proportional to v, not v^2. After including data describing other networks, the graph is yet more sparse. We identify two factors responsible for this:

- Our compact representation of cliques greatly diminishes the average degree of vertices in complete graphs like the ARPANET.
- It is difficult to manage, or even to collect, a large database of connection information, such as UUCP's Systems file. Also, hosts that do have large Systems files tend to be described in the USENET data as members of a network, such as the AT&T Network Action Central clients. Here again, the compact representation of cliques lends sparseness.

We therefore use a variant of Dijkstra's algorithm suitable for sparse graphs.[13]

Simply stated, we perform a modified breadth-first search of the graph, starting at the source. However, where breadth-first search normally inserts vertices into a queue and extracts them in first-in-first-out order, we use a priority queue[14] and extract vertices in increasing order of path cost. (The cost of a path is the sum of the weights of the edges along the path, subject to some heuristics described below.)

The algorithm maintains three sets of vertices: *mapped* vertices, to which optimal paths are known; *queued* vertices, for which a candidate path has been found; and *unmapped* vertices, which are not yet reachable.

---

10 G.H. Gonnet, *Handbook of Algorithms,* Addison Wesley, Reading MA, 1984.

11 As suggested in A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms,* Addison Wesley, Reading MA, 1974.

12 Described in A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *Algorithms and Data Structures,* Addison Wesley, Reading MA, 1983, p. 203.

13 *Ibid*, p. 207.

14 *Ibid*, p. 135.

A queued vertex v is extracted from the priority queue if it has the smallest candidate path cost, whereupon v is marked as mapped. The neighbors of v are then added to the priority queue and the edges that brought us these neighbors are marked as participating in optimal paths. If some neighbor of v is already queued, but the path through v is shorter, we reduce the cost to this neighbor, unmark the ‘‘old’’ edge, mark the ‘‘new’’ edge, and restore the heap property.

As it happens, when the mapping algorithm terminates, the marked edges form a directed tree, rooted at the source vertex. Later, we will show how to traverse this tree and generate the paths themselves.

For the priority queue itself, we use an implicit binary heap.[15] This requires a large contiguous array, but since the hash table is no longer needed and is guaranteed to be large enough, we use that space instead of allocating a new array.

Time complexity

A vertex can be inserted into or deleted from a binary heap of size k in time proportional to $\log k$. The heap can never have size greater than v, and we insert and delete vertices at most e times, where e is the number of edges, so the running time for the mapping phase is proportional to $e \log v$. Since the graph is sparse, *i.e.*, e is proportional to v, $e \log v$ is proportional to $v \log v$. Both asymptotically and pragmatically, the priority queue variant is a clear winner over the standard version of Dijkstra’s algorithm, which runs in time proportional to v^2. (Note, though, that if the graph is dense, our running time is proportional to $v^2 \log v$.)

Back links

After running of the shortest-path algorithm, it’s common to find that some hosts are unreachable. Before reporting these hosts on the error output, we examine the connections out of each unreachable host, invent links from its neighbors back to the host, and continue with Dijkstra’s algorithm. The resulting paths are thus generated by implication, rather than by explicit declaration.

Cost calculation

In calculating path costs, *pathalias* augments the edge weight sums with heuristics designed to avoid ambiguous routes and routes through networks that demand a gateway. Although this sullies our weighted graph model, it’s consistent with our pragmatic approach to cost measures.

Avoiding ambiguous routes

It is widely acknowledged that no simple measures suffice for disambiguating a route that contains both ‘@’ and ‘!’. Although effective heuristics have been proposed,[16] this approach is not widespread: most mailers rigidly adhere to ‘‘UUCP syntax’’ or to ‘‘RFC822 syntax.’’[17] As such, they consistently make the wrong choice on selected inputs. It’s clear, then, that we should be willing to pay a price if it results in fewer ambiguous routes, so *pathalias* adds a heavy penalty to paths that mix routing syntax.

As it happens, with our (atypically large) data set, this penalty is applied to only a fraction of a percent of the generated routes. A more significant factor in avoiding ambiguous routes is the ability of gateway hosts to accept addresses that merge domain[18] and UUCP conventions. For example, where a route such as `seismo!postel@f.isi.usc.edu` was once unavoidable, it is now permissible to use

15 Sedgewick, *op cit*, p. 130.

16 P. Honeyman and P.E. Parseghian, ‘‘Parsing Ambiguous Addresses for Electronic Services,’’ *Software — Practice and Experience*, to appear.

17 D.H. Crocker, ‘‘Standard for the Format of ARPA Internet Text Messages,’’ RFC822, Network Information Center, SRI International, Menlo Park CA, 1982.

18 P. Mockapetris, ‘‘Domain Names — Concepts and Facilities,’’ RFC882, Network Information Center, SRI International, Menlo Park CA, 1983.

`seismo!f.isi.usc.edu!postel`. We shall say more about domains a little later.

Gatewayed networks

Many hosts are situated on multiple networks and transports. For example, hundreds of ARPANET and CSNET hosts are also reachable via UUCP. However, for technical, administrative, and political reasons, only a (literal) handful provide gateway services. Therefore, we provide a way to declare networks that require explicit gateways, and a way to declare their gateways.

Any path that enters such a network through a host not declared as a gateway is severely penalized. Because hosts with domain addresses are by definition ARPANET hosts, domains and subdomains are assumed to require gateways. A similar constraint also affects links out of domains, so that once a path enters a domain, *pathalias* penalizes further links. This assures compliance with ARPANET restrictions on use of the network as a relay.

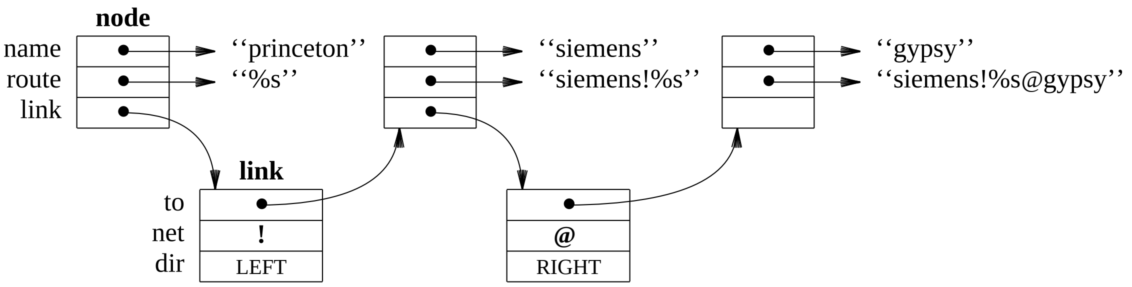
PRINTING THE ROUTES

With the shortest path tree identified, we now enter the third phase of *pathalias*. The goal is to print each host name followed by the route to that host. Routes are presented as *printf* format strings, *e.g.*, `ulysses!decvax!%s`. A mail user or delivery agent combines this route with a user name, producing a complete route.

Routes are computed by labeling nodes in the shortest path tree in a preorder traversal. We first label the root, which corresponds to the local host, with route `%s`. In the recursion step of the traversal, we calculate the route to a child node by combining the parent's route and the routing information in the parent-to-child edge.

Link routing information consists of a character to use as the network routing operator, and the specification of whether a host should appear on the left or right of the operator. For example, UUCP links use '!' and LEFT, while ARPANET links use '@' and RIGHT. As the traversal proceeds, the route to a node being visited is constructed by replacing `%s` in the parent's route with `host!%s` or `%s@host`.

For example, the following tree fragment, rooted at `princeton`, has been labeled with routes.



Note that the *net* and *dir* fields of the link structure determine the textual presentation of the route. Once we return from a level of recursion, the route to a host is no longer needed; thus routes are not actually stored as members of nodes, but are computed and passed as parameters in the recursion. Also note that if the mapping and printing phases were merged, we would be forced to keep intermediate paths in memory throughout the mapping phase, at a cost of hundreds of kbytes of extra space.

Networks and private hosts

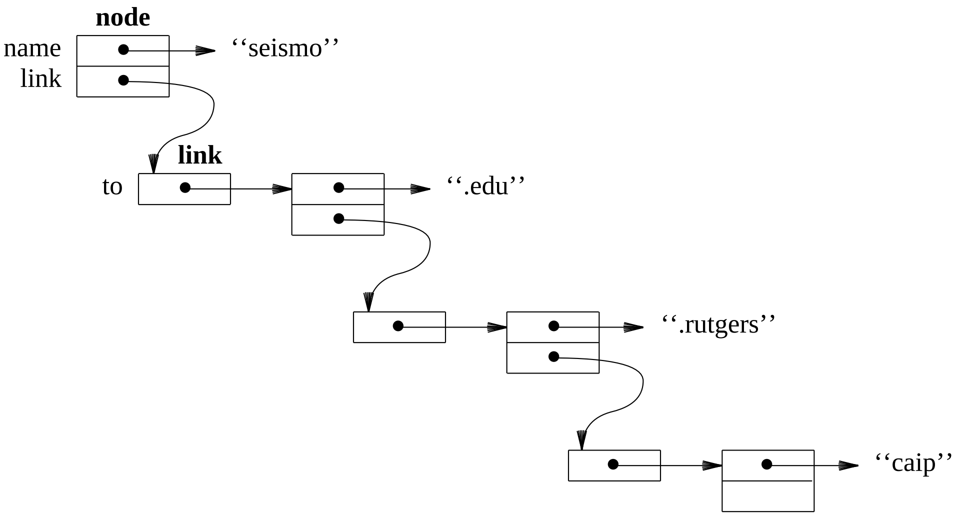
Pathalias gives networks special treatment: the route to a network is identical to the route to its parent. Except for domains, a network (and its route) never appears in the output, serving only as a placeholder for routes to network members.

When traversing a network-to-member edge, the routing character and direction are the ones encountered when entering the network. This allows different gateways between two networks to use different syntax.

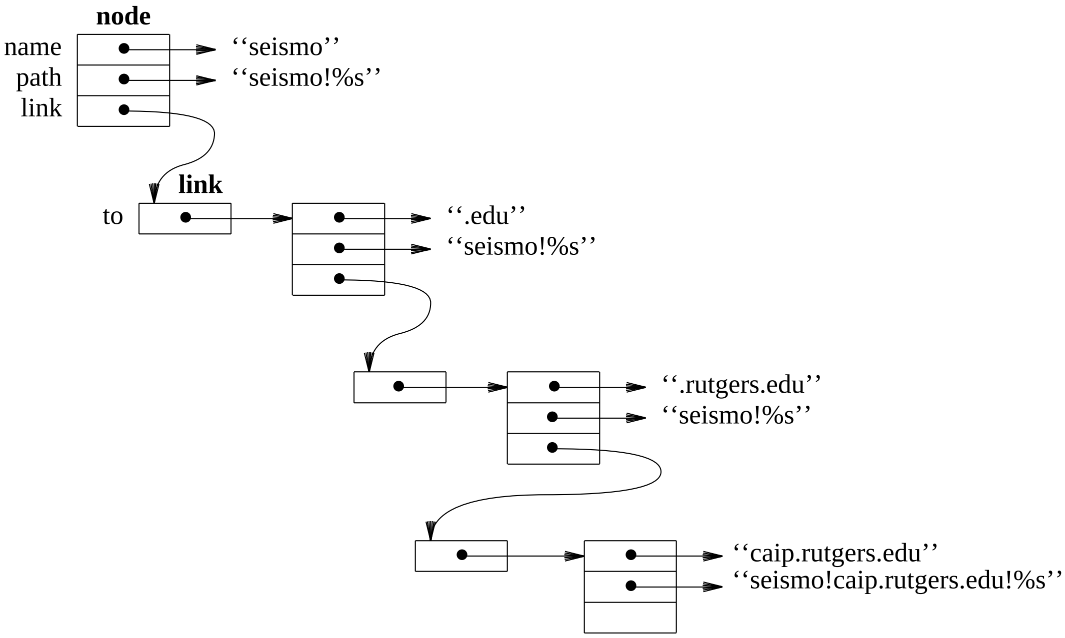
Private hosts are labeled just as other hosts, but no output is printed. However, a private host name might appear in the output as a relay to another, non-private host.

Domains

Like networks, domains act as placeholders for routes to domain members. But domains play a larger role in building up routes: upon encountering a domain in the tree traversal, the name of the domain is appended to the name of its successor. For example, given the following tree fragment



the traversal alters the host names and produces paths as shown.



As with networks, the cost from a domain to a subdomain is zero. However, the rule for building domain routes does not preclude traversals ‘‘up’’ the domain tree, if they happen to be ‘‘down’’ the shortest path tree. We therefore assume that all domains require gateways, and treat a parent domain as a gateway to its member domains. This imposes a heavy cost penalty, essentially infinite, on the edge from a subdomain to its parent, which prevents such absurdities as `caip!seismo.css.gov.edu.rutgers!%s` (which may or may not succeed — we don't care to experiment).

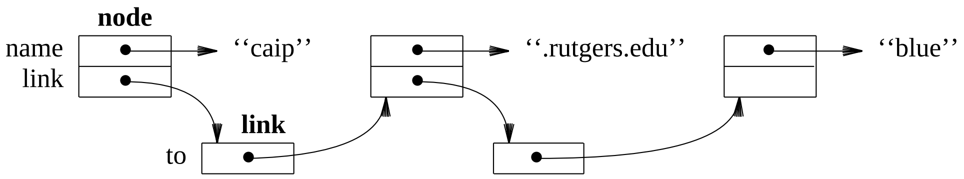
Unlike other networks, a top level domain, *i.e.*, a domain whose parent is not also a domain, is shown in the output. The route is given by the route to its parent (*i.e.*, its gateway).

As with routes to hosts, domain routes are intended to be used by a mailer. While the argument for a host's format string is generally a user, the argument for a domain is a route relative to its gateway.

For example, in the figure above, if the route to seismo is `seismo!%s`, then so is the route to `.edu`. To route to `caip.rutgers.edu!pleasant`, a mailer first searches the route list for `caip.rutgers.edu`; if found, the mailer uses argument `pleasant`, producing `seismo!caip.rutgers.edu!pleasant`. Otherwise, a search for `.rutgers.edu`, followed by a search for `.edu`, produces `seismo!%s`, the route to the `.edu` gateway. The argument here is not `pleasant` (as it were), it is `caip.rutgers.edu!pleasant`, producing `seismo!caip.rutgers.edu!pleasant`, as before.

Since a subdomain has the same route as its parent, the rule for top-level domain routing subsumes subdomain routing. For brevity, then, routes to subdomains are not printed.

Note that our definition of a top-level domain, one whose parent is not a domain, allows a subdomain to masquerade as a top-level domain. This permits hosts to gateway into selected portions of a larger domain structure. For example, to augment the figure above with a top-level domain `.rutgers.edu` with gateway `caip`, we add the following fragment to the domain tree:



This makes `caip` a gateway for `.rutgers.edu`, but not for the ARPANET as a whole.

If a host has a direct path to `caip`, then the route to `caip` and `blue` become `caip!%s` and `caip!blue.rutgers.edu!%s`, respectively. Observe that `.rutgers.edu` is logically an alias of `.rutgers`, but such a declaration is superfluous.

INTEGRATING PATHALIAS WITH MAILERS

There are a number of ways to integrate *pathalias* with the rest of a mail system, each with advantages and disadvantages. Whatever course is taken, it is always advantageous to make the route database available for manual querying by users. In the simplest case, this constitutes the full extent to which *pathalias* is used in a mail system.

A second possibility is to modify user agents to make queries automatically. This allows addresses in mail headers to contain actual expanded paths, rather than eliding the information. However, many sites have more than one mailer; consistency requires that all be updated. Such changes incur a continuing maintenance burden as new releases are installed.

To avoid the extremes of full manual operation and large-scale modifications of existing software, a compromise choice is to bury the database query in a separate program that is executed by a delivery agent. This avoids complicated changes to existing code; changes that must be made tend to be simple, perhaps involving a configuration table rather than a program. If there is only one transmission channel, such as UUCP, this is certainly the method of choice. Unfortunately, this is not always the case: a host may have an SMTP[19] link to some hosts but not to others. Here the router must know how to speak to each possible

[19] J. Postel, ‘‘Simple Mail Transfer Protocol,’’ RFC 821, Network Information Center, SRI International, Menlo Park CA, 1982.

gateway, information that is more properly the responsibility of the delivery agent.

Which brings us to the last choice: integrating the query into the delivery agent. In some cases, this is straightforward: *upas*,[20] for example, permits an external program to be invoked to rewrite an address. In other cases, notably *sendmail*,[21] complicated changes to the code and configuration table are necessary. In any event, putting the database query into the mail delivery agent means that all mailers will receive its benefits, and any network may be used to transport the mail.

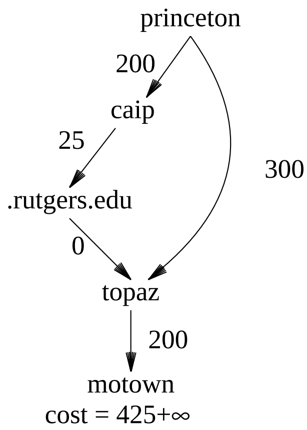
Another issue that must be settled is the extent to which *pathalias* data is allowed to override a user's selection of a path. In particular, given a hideously long UUCP path (such as one generated by a USENET reply), should the mailer simply find a route to the first site in the string, or should it search for the right-most host known to its database? The latter approach can result in significant savings; unfortunately, it can backfire if the user *wants* to use a circuitous route for some reason — say, to bypass a dead link. Indeed, it may be desirable to turn off optimization entirely. Loop tests are a time-honored UUCP tradition, and an overly-enthusiastic optimizer can eliminate them altogether.

Whatever choices are made, it is vital that any address visible in a locally-generated mail header must be acceptable if received in remote mail; if this rule is not heeded, replies may not work properly. Strictly speaking, this rules out using first-hop routing for remote mail while using ‘‘smart’’ routing for locally-generated mail; in practice, this causes trouble only if the first hop on such a path isn't known at all, but a later one is — a rare phenomenon.

PROBLEMS

We view our treatment of host name collisions with some skepticism. While we believe in individual hosts managing their own name spaces, the job of identifying private hosts is time-consuming and sometimes arbitrary. We would be pleased if, for example, the UUCP mapping project data either marked host name collisions with private declarations or simply excluded them. Even then, data collected from other sources must be perused to identify problem hosts. The only possible solution is a global absolute name space, but we see no viable candidate that can compete with the ease, low-cost, simplicity, and extensibility of UUCP's relative address space.

A more technical problem with *pathalias* is its insistence on using paths strictly out of the shortest path tree. Once *pathalias* has decided on a route, it is committed to that route for hosts beyond it. Consider, for example, the following connection graph.



At first glance, the best route to `motown` follows the left branch, with cost 425, instead of the right

---

20 D. Presotto, ‘‘Upas — A Simpler Approach to Network Mail,’’ in *Proc. Summer USENIX Conference,* Portland, 1985.

21 E. Allman, ‘‘SENDMAIL — An Internetwork Mail Router,’’ In *UNIX Programmer's Manual,* 4.2 Berkeley Software Distribution, 1983.

branch, with cost 500. However, the domain heuristic penalizes the former route, so that the right branch should be preferred. (In practice, the mailer at Rutgers rejects the left branch route.)

The problem lies with our shortest path computation: we compute a shortest path tree, but the routes we want to generate cannot be represented in a tree. We are currently experimenting with a modified algorithm that maintains the ‘‘second-best’’ path when the shortest path to a host goes by way of a domain.

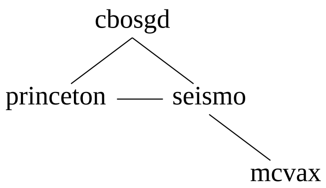
PERSPECTIVES ON RELATIVE ADDRESSING

Relative addressing is employed on most networks, even some that advertise otherwise. For example, while ARPANET requires compliance to RFC822 rules, member hosts stretch the rules with underground syntax: `user%host@relay`. Although this syntax is legal, it achieves neither the ARPANET goal of pure absolute addressing, nor the UUCP virtue of consistent syntax.

Absolute addressing requires absolute compliance to standards promulgated by an absolute authority, yet it is impossible to bring absolutely every host under any authority. This is especially true when crossing administrative boundaries, *e.g.*, from the ARPANET, administered by the Network Information Center, to UUCP, largely unadministered. Furthermore, even RFC822 recognizes the usefulness of explicit routing, and provides a (clumsy) syntax to support it. Finally, any host that hopes to provide gateway services to UUCP is forced to accommodate the UUCP addressing conventions.

With this as the framework for internetworking, *pathalias* is a tremendously useful tool. Internetwork addresses are built up by splicing together the name spaces of intermediate hosts and gateways. Without a routing tool, this places a heavy burden on users, in keystrokes and long-term memory. *Pathalias* frees users of these requirements. Yet this is a mixed blessing, as it can interfere with route translation. We illustrate by way of an example.

Consider the following fragment of the UUCP connection graph.



(All links are bidirectional.) Suppose a message is sent from `cbosgd!mark` to `princeton!honey`, with a copy to `seismo!mcvax!piet`. The message arrives on princeton as

```
From cbosgd!mark Sun Feb  9 13:14:58 EST 1986
To: princeton!honey
Cc: seismo!mcvax!piet
```

From the perspective of `princeton!honey`, the copy recipient is `seismo!mcvax!piet` relative to `cbosgd`, *i.e.*, `cbosgd!seismo!mcvax!piet`, but this can be safely shortened to `seismo!mcvax!piet`.[22] However, if `cbosgd` runs *pathalias*, the ‘‘carbon copy’’ header might be abbreviated `mcvax!piet`; the copy recipient is now `cbosgd!mcvax!piet`. This cannot be safely transformed without making assumptions about host name uniqueness.[23]

Pathalias thus warps the relative name space of UUCP, providing a false sense of absolute addressing. While *pathalias* works well as a router for the physical topology of a network, it can be abused when applied to the name space topology. In the above example, message headers clearly support the view that host `mcvax` is in the name space of host `cbosgd`. But it is folly to treat this as the case, as it makes

22 See Honeyman and Parseghian, *op cit*, for details.

23 Some UUCP hosts have adopted the ARPANET’s domain syntax, and are listing their names with a central registry. Among these hosts, it is reasonable to assume names are unique, and that full routing is supported. But the situation is in flux.

route optimization a complex problem in distributed computing, impossible to solve in a network of point-to-point, low bandwidth links.

For message headers to be useful, they must be accurate. This can be accomplished only if user or delivery agents expend some effort. While we do not advocate the approach taken by *sendmail*,[24] in which headers are subjected to deep analysis and complex transformations, we urge adherence to some simple principles:

- Message headers should be modified only as necessary to conform to network standards.
- Other message data should not be modified at all.
- A host must not generate a return path that would be rejected if used.
- Hosts that re-route mail from local users should show the modified routes in message headers.
- Relays within a network should not modify routes, nor translate to foreign addressing styles.
- Gateways should translate between addressing styles when providing gateway services.

Compliance with these guidelines, even loose compliance, can make internetwork addressing and routing a practical reality.

ACKNOWLEDGEMENTS

We thank Rick Adams, Steve North, Pat Parseghian, and Marc Stavely for many fruitful discussions. Pat also helped a great deal with this paper. Bob Sedgewick and Bob Tarjan helped with some of the fine points in algorithm design. Paul Haahr and Dave Hitz gave sound advice to their advisor. We thank Dennis Ritchie, Dave Presotto, and Mark Horton for comments on an early draft of this paper.

For the past four years, many of our USENET correspondents have played an important role in the development of *pathalias*. Among them are Paul Bame, Marc Brader, Piet Beertema, Scott Bradner, Robert Elz, Ray Essick, Erik Fair, Stephen Gildea, Jack Hagouel, Jordan Hayes, Johan Helsingius, Hokey, Mark Horton, Sam Kendall, Gary Murakami, Greg Noel, Mel Pleasant, Guy Riddle, Larry Rogers, Eric Roskos, Bill Sebok, Chris Seiwald, Alan Silverstein, Rob Warnock, and the UUCP map coördinators. A special thanks to Karen Summers-Horton, who slaved over the early UUCP and USENET maps.

Finally, we thank the thousands of people who use *pathalias*; their reliance on our ideas has been a constant encouragement, and their palpable confusion spurred us to write this paper.

---

[24] Allman, *op cit*.