PATHALIAS

or

The Care and Feeding of Relative Addresses

Peter Honeyman

Computer Science Department
Princeton University
Princeton, New Jersey 08544

Steven M. Bellovin

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

TR-030-86

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

In many computer networks, such as the ARPANET, individual users do not have to worry about routing. Whether the service is mail transfer or remote login, the task of moving data to its destination is automated. On the other hand, the UUCP network[1] relies on dial-up, point-to-point connections; each host needs to know exactly how to reach every other to which it wishes to talk. A UUCP host's set of network neighbors is defined solely by its configuration files, rather than by centrally administered network management; many hosts are unable to talk directly to one another. Accordingly, mail (and other such file transfers) must be relayed by intermediate hosts. Under the decentralized philosophy of UUCP, such relaying is explicitly specified by users. That is, a user sends mail to `hostb` using `hosta` as a relay with the command

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

Once upon a time, the UUCP network was small and the average connectivity was high, so explicit routing was a minor annoyance at worst. Most paths were direct, and only a tiny fraction

---

* 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.

involved more than one or two hops, so remembering proper paths was easy.

Then came USENET[2]. 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, 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 least-cost paths to every known destination. The most interesting technical question was how to define "cost." Possible metrics included the actual telephone cost of a connection, the nominal frequency of contact (many university sites were passive — rather than calling out, they waited for other sites to call them), or the transmission speed of a connection. Ultimately, a pragmatic metric was adopted: given a choice of paths, which did experienced users prefer?

Using this metric helped deal with 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 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.

Once the basis for a metric was adopted, symbolic names like **HOURLY**, **DAILY**, *etc.* were assigned numeric values. 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. Costs can be expressed as arbitrary arithmetic expressions, mixing numbers and symbolic values. For example, **HOURLY*3** indicates 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.

At first, gathering data on network connectivity 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,[3] the picture is much brighter today, with timely and accurate data widely available on USENET.

DATA STRUCTURES

Pathalias runs in three phases: parse the input, build a shortest path tree, and print the routes.

---

2 S.M. Bellovin and M. Horton, "USENET — A Distributed, Decentralized News System," unpublished manuscript, 1986.

3 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.

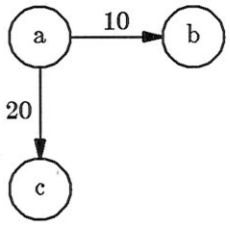
Each phase manipulates an in-memory representation of a directed graph. Before describing these computations, we describe the basic data structures.

Graph representation

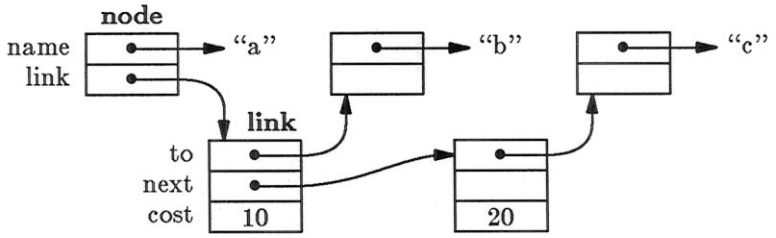
We assume that the world can be modeled as a set of hosts and networks, called *nodes*, with communication links among them. This in turn is modeled by a directed graph, with vertices representing hosts and networks, and edges representing communication links. Edges are weighted with a non-negative cost, 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 n^2 edges, so with over 2,000 hosts in the ARPANET we are presented 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.

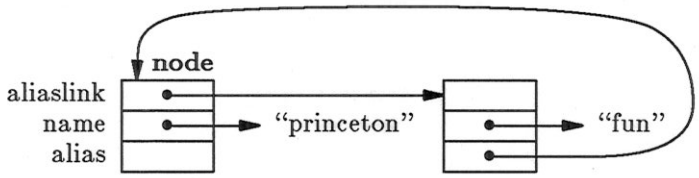
Edge weights and labels for networks are given as input. The weight pertains only to the edges originating at network members; the cost associated with an edge from a network to one of its members is zero. As an analogy, consider automobile toll collection by the Port Authority of New York and New Jersey: you pay to get into the City, but you get back to Jersey for free.

PARSING

Parsing is done with *yacc*.[4] We use syntax-directed translation to support a rich syntax, with edge weights and labels, aliases, networks, and modest support for host name collisions. We experimented with *lex*[5] 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 cut the run time by 40%.

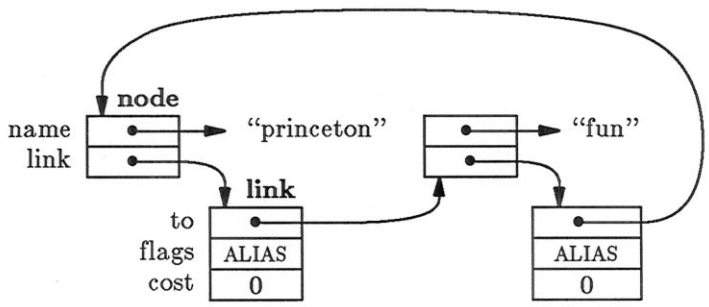
Aliases

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. The graph representation required two pointers in each host: one to link all the aliases of a node on a list, and one to indicate the primary host name for the node. For example, a host `princeton` with nickname `fun` had this graph representation.



This treatment did 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 required predicting the shortest path *a priori*.

To address this problem, we discarded 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 a possible 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.

---

[4] S.C. Johnson, "YACC — Yet Another Compiler Compiler," in *UNIX Programmer's Manual,* Seventh Ed., 1979.

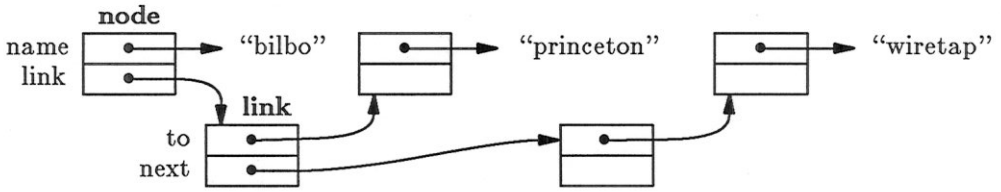
[5] M.E. Lesk and E. Schmidt, "LEX — Lexical Analyzer Generator," in *UNIX Programmer's Manual,* Seventh Ed., 1979.

Host name collisions

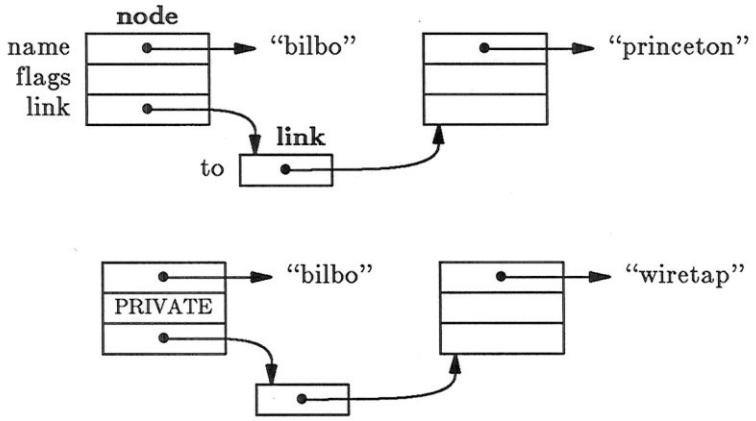
Among the hosts that rely on UUCP as their only means of communication, there is no authority empowered to prevent host name conflicts, and they do occur. Consequently, *pathalias* may mistakenly merge connection information for distinct hosts with identical names.

We address this difficulty with the "private" declaration, which narrows the scope of connection declarations. 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 a number of techniques[6] 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

[6] 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.

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 segmented machines 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 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 $1+(k \bmod T-2)$,[7] as this results in anomalous behavior (that we cannot explain); rather, we use the inverse $T-2-(k \bmod T-2)$.[8]

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 installed 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.[9]

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 $\delta=2$,[10] an excessive amount of space is wasted when the total number of hosts happens to be slightly more than $\alpha_H T$. 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 would satisfy $\alpha<\alpha_L$. 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 simply maintain a Fibonacci sequence of primes (more or less), which also follows the golden ratio. (Besides, we like Fibonacci numbers.)

Discarding the old hash table is one of the few opportunities for re-using space during the parsing phase. Rather than freeing the old tables, which can range in size from 4 kbytes to 32 kbytes, they are placed on a list and made available to the 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*[11] which runs in time proportional to v^2 on a graph with v vertices. However, 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 even 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

---

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

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

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

10 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.

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

described in the USENET data as members of a network, such as the AT&T Network Action Central nets. Here again, the compact representation of cliques lends sparseness.

Accordingly, we employ a variant of Dijkstra's algorithm suitable for sparse graphs.[12]

Simply stated, we perform a breadth-first search of the graph, starting at a designated vertex, usually the local host. 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[13] and extract vertices in increasing order of weight. (The weight of a node in the queue is the sum of the edge weights along a path beginning at the source, subject to heuristics described below.)

For the priority queue itself, we use a binary heap.[14] This requires a contiguous array, so we slide the entries in the hash table to one end, and use the other end for the heap. In this way, we avoid allocating an array of several thousand pointers.

Time complexity

A vertex can be inserted into or deleted from a binary heap in time proportional to log k for a queue of size k. The heap can never have size greater than v, and we insert and delete each edge at most once, so the running time for the mapping phase is proportional to $e \log v$, *i.e.*, $v \log v$, since e is proportional to v. This is a clear winner over the standard version of Dijkstra's algorithm.

Back links

After one iteration of the shortest-path algorithm, it's common to find that some nodes are unreachable. To minimize error output, we examine the connections out of an unreachable host, invent links from its neighbors' nodes, 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,[15] this approach is not widespread: most mailers rigidly adhere to "UUCP syntax" or to "RFC822 syntax."[16] 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, a fraction of a percent of the generated routes are ambiguous, so the penalty is rarely applied. A more significant factor in avoiding ambiguous routes is the ability of gateway hosts to accept addresses that merge domain[17] and

---

12 *Ibid*, p. 207.
13 *Ibid*, p. 135.
14 Sedgewick, *op cit*, p. 130.
15 P. Honeyman and P.E. Parseghian, "Parsing Ambiguous Addresses for Electronic Services," *Software — Practice and Experience*, to appear.
16 D.H. Crocker, "Standard for the Format of ARPA Internet Text Messages," RFC822, Network Information Center, SRI International, Menlo Park CA, 1982.
17 P. Mockapetris, "Domain Names — Concepts and Facilities," RFC882, Network Information Center, SRI International, Menlo Park CA, 1983.

UUCP conventions. For example, where a route such as `seismo!postel@f.isi.usc.edu` was once unavoidable, it is now permissible to use `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 non-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 concerning use of the network as a relay.

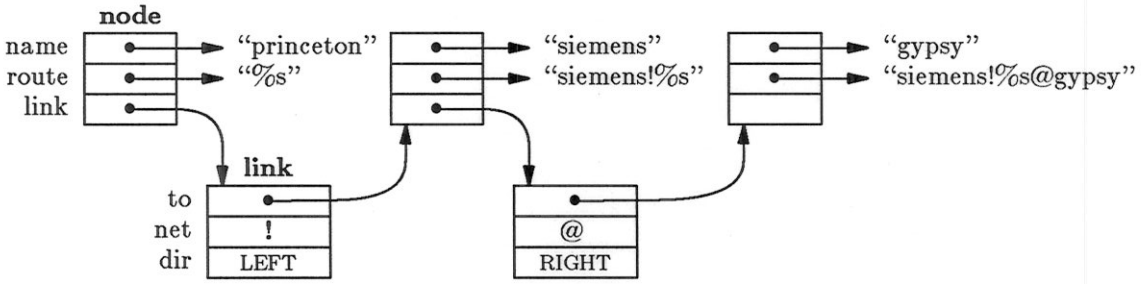
PRINTING THE ROUTES

With the shortest path tree constructed, 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 in the form of *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 recursively as local variables.

Networks and private hosts

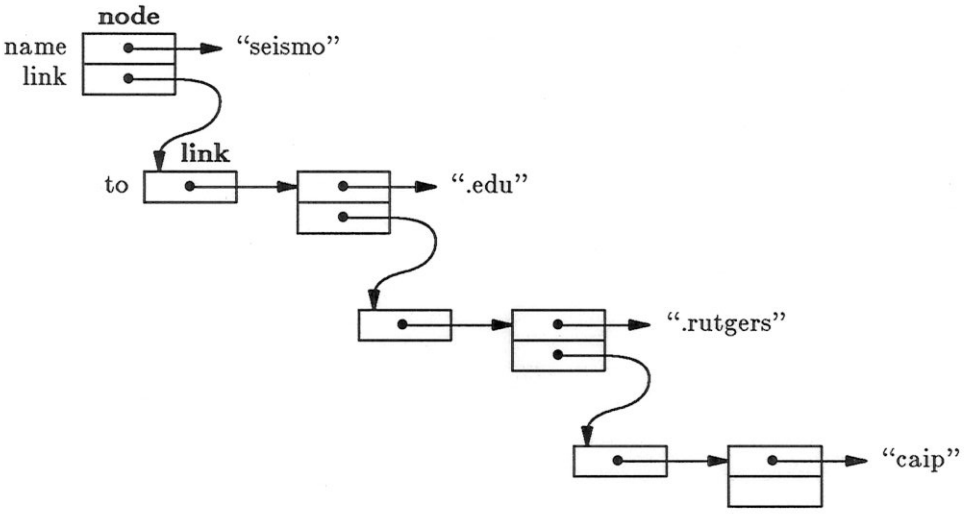
Pathalias gives networks special treatment: the route to a network is identical to the route to its parent. Except for domains, a network (and its route) never appears in the output, serving only as a placeholder for routes to network members.

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

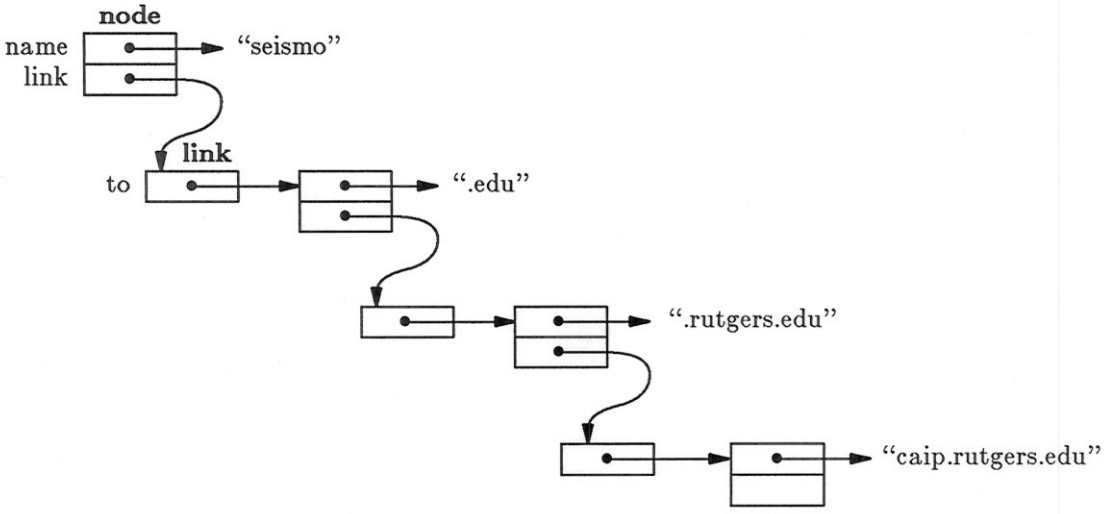
Private hosts are labeled no differently than other hosts, but no output is printed. However, a private host name appears in any route for which the host is a relay.

Domains

Domains, like networks, 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 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 be legal — we aren't interested).

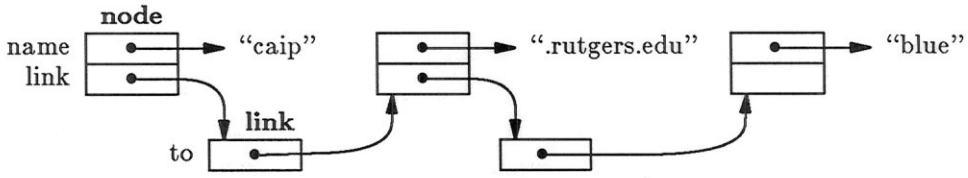
Unlike other networks, a top level domain, *i.e.*, a domain whose parent is not also a domain, is shown in the output. The route is given by the route to its parent (*i.e.*, its gateway).

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 tree fragment:



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 hiding 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.

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[18] link to some hosts but not to

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

others. In this case, the router must know how to speak to each possible 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*,[19] for example, permits an external program to be invoked to rewrite an address. In other cases, notably *sendmail*,[20] complicated hacking is 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 rightmost 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.

It is easier for local users to control this than it is for remote users; mailers can be modified to request different levels of service from a router, under user control. Indeed, since most such overrides are done by system and network administrators, it is not unreasonable (though it is perhaps undesirable) to limit such options to one or two mailers. But mail arriving from remote sites typically carries no such option selection; generally, only the destination address is given.

The best rule is that any address visible in a locally-generated mail header must be acceptable when received in remote mail; if this 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 situation only causes trouble if the first hop on such a path isn't known at all, but a subsequent one is — a rare situation.

PROBLEMS

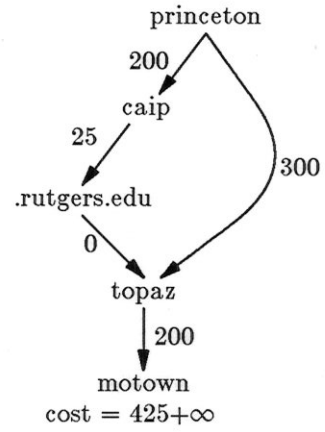
We view our treatment of host name collisions with some skepticism. While we believe in individual hosts managing their own name spaces, the job of identifying private hosts is time-consuming and sometimes arbitrary. We would be pleased if, for example, the USENET data either marked host name collisions 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.

---

[19] D. Presotto, "Upas — A Simpler Approach to Network Mail," in *Proc. Summer USENIX Conference,* Portland, 1985.

[20] E. Allman, "SENDMAIL — An Internetwork Mail Router," In *UNIX Programmer's Manual,* 4.2 Berkeley Software Distributuion, 1983.



At first glance, the best route to `motown` follows the left branch, with cost 425, instead of the right branch, with cost 500. However, the domain heuristic penalizes the former route, so that the right branch is actually the preferable one. (And in fact, the mailer at Rutgers rejects the left branch route.)

The problem lies with our shortest path computation: we are computing 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.

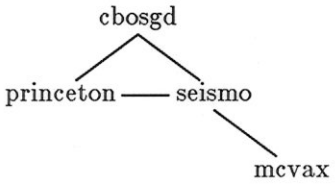
PERSPECTIVES ON RELATIVE ADDRESSING

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`.

This should come as no surprise: absolute addressing requires absolute compliance to standards promulgated by an absolute authority, yet it is impossible to bring absolutely every host under any authority. Furthermore, even RFC822 recognizes the necessity of explicit routing, and provides a (clumsy) syntax to support it. Finally, any network 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`.[21] 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.

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 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*,[22] in which headers are wantonly modified, 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 loosely, can make internetwork addressing 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.

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-mappers mailing list. Although we usually ignored their suggestions, and were occasionally positively insulting in response, they made a positive contribution by forcing us us elucidate our ideas. A special thanks to Karen Summers-Horton, who slaved over the early USENET maps.

Finally, we thank the thousands of people who use *pathalias*; their reliance on our ideas has been a constant encouragement.

---

[21] See Honeyman and Parseghian, *op cit*, for details.

[22] Allman, *op cit*.