Efficient and Robust Policy Routing Using Multiple Hierarchical Addresses

Paul F. Tsuchiya
Bellcore
tsuchiya@thumper.bellcore.com

Abstract

One of the most compelling long term problems facing the IP and emerging OSI Internet is growth. At the same time, policy routing—the ability of a packet source to determine the type of path its packets will take, or the ability of a transit network to restrict usage—is increasingly required. A hierarchical addressing format reflecting the hierarchical backbone structure reduces routing overhead, but restricts paths to the backbones implied by the address, thus limiting policy and reducing robustness. To overcome this restriction, yet keep the efficiency of hierarchical addressing, this paper proposes the use of multiple addresses. In this scheme, directory service would return multiple addresses, and the host would choose the ones appropriate for its policies, any of which could be used during a connection, as network conditions deem appropriate. This paper establishes a framework for the new technique by exploring the fundamentals of routing and addressing, describes the new technique, and discusses its role as a policy mechanism.

1.0 Introduction

One of the most compelling long term problems facing the IP and emerging OSI Internet is growth.[1] The current IP address and resulting routing algorithms scale poorly. This is because there is no routing hierarchy above the network level, and many routers must keep and maintain entries for all other *N* networks [Ro]. Exacerbating this is the fact that policy routing—the ability of a packet source to determine the type of path its packets will take, or the ability of a transit network to restrict usage—is increasingly required. Of course, the introduction of scaling and policy should not decrease robustness (the ability of routing to find paths around failures).

Unfortunately, the current approach to network layer addressing in TCP/IP and OSI protocols does not easily allow for both scaling and policy. The reason is as follows. To achieve scaling, hierarchical addresses are required. However, hierarchical addresses tend to restrict the path found by routing to the hierarchical components listed in the address. Say for instance, in order to achieve scaling, Internet hosts that obtained their primary connectivity through the NSFNET had an address indicating this affiliation (i.e., the address read NSFNET.Network.Host). Then packets going to these hosts would go through the NSFNET. It would not be possible to send a packet to the host through say MILNET, because routers would have no way of knowing that the destination was reachable via MILNET[2]. Therefore, policy control is limited. In order to send packets through either MILNET or NSFNET, routers would need to keep entries for individual networks, not just backbones, which limits scaling, therefore defeating the purpose of the hierarchical address.

This problem goes away if the host has multiple addresses, in this case, one from NSFNET and one from MILNET. The source host picks one address or the other depending on whether the connection should go through NSFNET or MILNET, thus achieving some policy control. The routers still need only keep entries for NSFNET and MILNET, thus achieving scaling.

While this notion seems simple enough, it is foreign to the Internetworking protocols (especially the TCP/IP

1. From September 1989 to April 1990, the number of networks announced by the NSFNET backbone grew from 750 to 1100 [Ka].

2. NSFNET and MILNET are national backbones in the USA that support different but overlapping communities of users.

protocols) and by and large to the Internetworking community. There seems to be a well entrenched notion that routers and their routing algorithms should have the sole responsibility for finding various paths through the Internetwork, and that hosts and their addresses should have as little to do with it as possible. For instance, Shoch's widely referenced and generally accepted paper on naming, addressing, and routing clearly puts the burden of path finding on routing [Sh].

This paper argues that multiple, dynamically chosen addresses (in addition to current naming and routing techniques) is a simple way to achieve the disparate goals of scaling, policy, and robustness. It makes this argument by breaking down the philosophy that names, addresses, and routes are three separate functions, and instead argues that there are in fact only two functions—naming and routing. Addressing contain elements of both naming and routing. This paper also discusses various aspects of policy routing—in particular, which policies can and cannot be achieved through the use of multiple addresses.

This paper uses the terms *router* and *host* to refer to transit nodes and end nodes respectively. Also, while this paper is applicable to addressing and routing issues in all kinds of networks, its major focus is on the TCP/IP and OSI Internetwork protocols.

2.0 What are Addresses?

In his paper on Internetworking architecture [Sh], Shoch suggests the following taxonomy:

- **Name**: <u>What</u> the object is
- **Address**: <u>Where</u> the object is
- **Route**: <u>How</u> to get to the object (the path to the object)

The idea is that when one wants to communicate with an object, one starts with the name (usually something user-friendly), which is used to find an address, which then is in turn used to find a path. In spite of the wide acceptance of this taxonomy, I believe that it does more harm than good in that it implies three separate domains of operation (naming an object, addressing an object, routing to an object) when in fact there are only two: **identifying** (what) an object, and **routing** (how) to an object.

I agree that there are names (what the user normally types into his workstation or looks-up in a phone book), addresses (what is given to the network in the packet header or by dialing a phone), and routes (what is in the routing tables), but in fact these three things together only serve two purposes. The name identifies, and the route routes, but the address identifies <u>and</u> routes[3].

Briefly stated, the problem with this "where" (addressing) notion is that 1) we tend to think of things as being in only one place at a time, implying that we need only one address at a time, and 2) since we relatively rarely move our computers or telephones, we tend to think of addresses as rather static. I explain later how these completely artificial restrictions on addresses place unnecessary restrictions on the scalability, robustness, and policy richness of routing. Now, however, I would like to explore some of the ambiguities of Shoch's taxonomy.

First, consider the Ethernet address. It is a flat address, meaning that the address contains no hierarchical content in-so-far as routing is concerned.[4] Shoch allows that addresses can be flat. However, a flat address says nothing about "where" it is in a network. If I unplug my Ethernet board from my Ethernet in New Jersey and plug it into a different Ethernet in Tokyo, it keeps the same address. Clearly there is no notion of "where" in a flat address. A flat address is purely an identifier, or name.

3. In another paper on naming, addressing, and routing [Ha], Hauzeur also argues that there is only naming and routing. However, Hauzeur classifies an address as nothing more than a special kind of name, and fails to recognize the crucial fact that an address is also a route.

4. In fact, the Ethernet address has hierarchical content with respect to the address assignment process. Organizations are given 3 byte blocks of address space from which they assign individual addresses. Routing, however, ignores this structure.

Next, consider a telephone number, which is an address. It is hierarchical, not flat. Therefore, it seems to convey a notion of "where". (Where is the phone numbered 201-829-4484? In area 201, switch 829.) But it is more accurate and useful to say that it conveys a notion of path, or "how". (How do I route to 201-829-4484? First, route to area 201, then to switch 829.) Clearly, the address 201-829-4484 doesn't specify the complete route. There will be multiple paths to 201 and to 829. Rather, 201-829-4484 is an incomplete route, but a route just the same.

One might argue that the hierarchical telephone number is in fact not conveying a notion of "how", because a telephone switch can, and often does, choose to route directly to 829, and not first to 201. My counter-argument to this is that if a switch routes directly to 829, then it is treating the string 201829 as an identifier, and is disregarding the hierarchical content altogether (that is, it is ignoring any "where" or "how" that might otherwise be there). Further, if a telephone switch chooses to take advantage of the hierarchical content in 201-829, it has no choice but to route to 201. In other words, to the extent that a router chooses to take advantage of hierarchical information in an address, it is treating the address as a route. To the extent that router chooses to ignore the hierarchical information in an address, it is treating the address as an identifier.

Now, consider a source route. This is the set of fields in for example an IP or ISO 8473 header that specifies part ("loose" or "partial" source route) or all ("complete" source route) of the path that the source routed packet should take. Shoch calls this a route, not an address. However, there is very little difference really between a source route (especially a partial source route) and a hierarchical address. The main difference is that a hierarchical address always concerns only the part of the path from the top of the hierarchy on down, while the source route can specify any part of the path. In this sense, a hierarchical address is nothing more than a special type of source route.

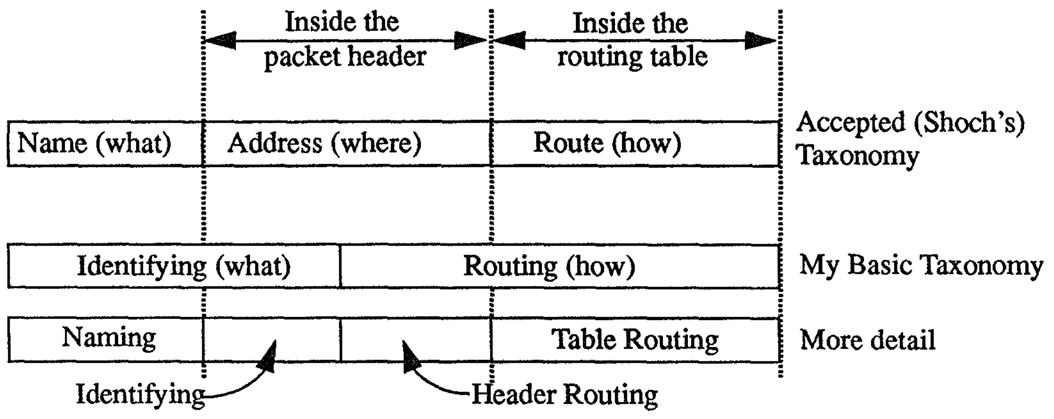
Figure 1 shows a comparison of the accepted (Shoch's) taxonomy and my taxonomy. We see that Shoch chose to classify according to what is inside the packet header (or call setup or out-of-band signalling) and what is inside the routing table. He gets into trouble because he doesn't account for the dual functionality of the packet header. I believe it is more appropriate in this case to classify according to function (identifying and routing). This more clearly illuminates what is being done where. That being said, however, I think the practical considerations of routing indeed have to do with what is done in the packet header and what is done in the routing tables. Therefore, I choose to further classify along those lines, and partition the routing function into **header routing** and **table routing**.

Table routing then is what is conventionally thought of as routing, and header routing is, for example, source routing, or the use of hierarchical addresses (which makes table routing scale better). Table routing and header routing are two realizations of the same function. People who champion source routing can be said to prefer putting the routing functionality into header routing. People who champion flat routing prefer putting the routing functionality into table routing. Hierarchical routing is a combination of both.[5]

The reader might ask why I make such a big fuss about taxonomy if in the end I classify along header/table boundaries as Shoch does. The reason has to do with where one perceives that the flexibility and dynamics of the system lie. Shoch and I believe that the "how" should be flexible and dynamic. Accordingly, Shoch says that routing tables should be able to hold multiple paths between a source and destination, and that the routing tables should be able to pick new paths quickly, even during the course of a connection. I agree with this.

5. Note that I also partition the identifying function into the part that is done in the network-layer packet header (identifying) and the part that is not (naming). The notion here is that naming is primarily there for the convenience of humans, while identifying is mainly a machine function. Since this paper is not about naming, I don't further discuss this point.

Figure 1: Comparison of Taxonomies



However, since Shoch doesn't view the packet header as contributing to "how", he (and the majority of the networking community) believes that addresses should not be nearly as dynamic as routes. In fact, perhaps because addresses mean "where", and because our hosts are in one (physical) place at a time and don't move often, the networking community believes that addresses should be nearly as static as names.

My assertion here is supported by the protocols that have been developed by the networking community. Consider TCP and ISO 8073, Class 4 (TP4). Once a connection is established, the address used at connection establishment time cannot change, even though the route can. Or, consider how difficult it is to change the address assigned to a host, or to a group of hosts. There are no protocols that explicitly support the assignment and reassignment of addresses to hosts, particularly groups of hosts. This is a reflection of the fact that addresses are not considered dynamic entities.

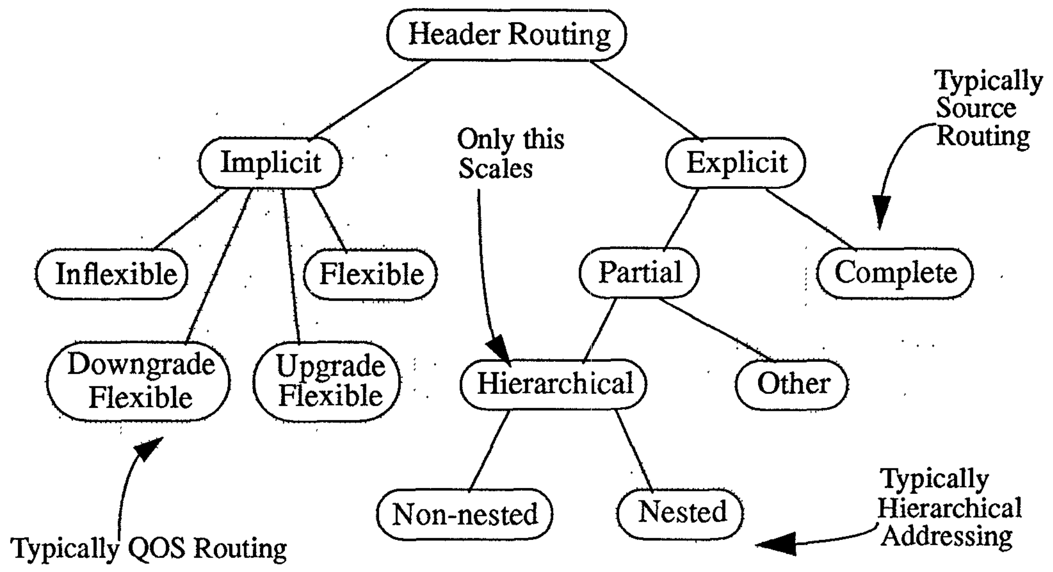
Since I do view the packet header as contributing to "how", I believe that, in certain respects, header routing, specifically the address, should be as flexible and dynamic as table routing. This flexibility and dynamicity is definitely constrained by the fact that the address also does the "what" job, which must be stable. However, as I will show, it is possible and beneficial to make addresses dynamic in some ways, and static in others. In fact, I argue that if we do not make addresses more flexible and dynamic, we cannot achieve the disparate goals of scaling, robustness, and policy in routing. That is the point of this paper.

3.0 Further Classification of Header Routing

Notice that in the last paragraph, I specifically point out that the address, rather than other types of header routing, should be more dynamic. Indeed, other proposals, such as Inter Domain Policy Routing (IDPR) [BE], call for dynamic source routing, which is another form of header routing. In addition, Quality-of-Service (QOS) routing, which is yet another form of header routing, can also be dynamic even using current IP and OSI protocols (although currently nobody uses them that way). However, of the types of header routing in common use, only hierarchical addressing scales well.

In order to shed some light on the various types of header routing, consider the taxonomy of Figure 2. The first categorization is between implicit and explicit header routing. In explicit header routing, the components of the path are explicitly called out, such as in a source route. The components do not have to be routers. They can be higher level entities such as backbones, but they must be representable as nodes connected to other nodes. In implicit header routing, on the other hand, actual components of the path are not called out. Instead, the header route implicitly influences table routing. An example is QOS routing, where a type of service is specified (for instance, high speed), but table routing chooses the actual components in the path.

Figure 2: Taxonomy for Header Routing



With implicit header routing, it is possible that the requested QOS is not available. The routers may then route over a different QOS (flexible), or may instead not deliver the packets (inflexible). Or, the routers may be willing to route along a better QOS, but not a worse QOS (upgrade flexible), and vice versa (downgrade flexible).

With explicit header routing, either the entire path (complete), or part of the path (partial) may be specified. If complete explicit header routing is used, then there is no table routing. All routing decisions are dictated by the information in the header. With partial explicit header routing, table routing is required to determine the part of the path not specified by the header route. As discussed in section 2, hierarchical routing is one form of partial explicit header routing. With hierarchical routing, the hierarchical components in the header may be nested or non-nested. If a hierarchical component is nested, then it is unique only with respect to the higher component. If it is non-nested, then it is unique with respect to all components at its level.

Usually, hierarchical header routes are nested. For instance, in the example of section 2, the 829 of 201-829 is only unique under 201. There are 829's in other area codes as well. However, the telephone system offers an example of non-nested hierarchical header routing. In the USA, one can choose among several long distance carriers by dialing an access code before dialing the usual 10-digit number.[6] Therefore, the complete hierarchical header route within the USA is <long distance carrier, area code, office code, telephone>. Since each area code is unique among all other area codes, regardless of the long distance carrier, that part of the header route is non-nested.

Interestingly, the long distance access codes concatenated with the rest of the telephone number is essentially multiple hierarchical addressing. However, this fact is largely hidden because 1) we rarely actually dial the access code, and 2) directory service does not publish multiple telephone numbers for each subscriber. This second point is possible only because with high probability any telephone can be reached through any of the major long distance carriers. Therefore, publishing each access code for each subscriber would be redundant.

This same style of multiple hierarchical addresses (that is, non-nested) is not appropriate for data networks. This is because one cannot assume that any data terminal will be reachable through any backbone.

6. Normally, a default long distance carrier is chosen, without requiring the access code. However, the default can be over-ridden by dialing the access code.

Indeed, even if any data terminal could be reached through any backbone, the service offered by the backbone may not be appropriate. Therefore, in data networks multiple hierarchical addresses must be explicitly called out, not assumed as in the USA telephone network case.

It is also interesting to note that multiple hierarchical addressing already exists in data networking, but only by virtue of there being multiple address spaces for different networking technologies—IP, NSAP, X.121, and E.164 as used for ISDN[7], to name a few. As internetworking spreads, the need for a single host to utilize more than one of these technologies, and therefore addressing spaces, grows. Indeed, NSAP addresses encompass X.121 and E.164 addresses. Also, even within one addressing space, multiple addresses are useful for distinguishing backbones of the same technology but different service or price. The use of hierarchical addresses must therefore be made explicit and intentional, rather than ad hoc and accidental.

There are three types of header routing information fields found in an IP or ISO 8473 (ISO's IP equivalent, called CLNP) header—address, QOS, and source route. Usually, they are encoded as indicated by the shading in Figure 2. However, it is possible that hierarchical header routing could be encoded in the source route or QOS fields. Indeed, since the "what" function is done in the address, and must therefore have certain static properties, it would make sense to put hierarchical header routing in a field other than the address. However, current IP and CLNP encodings of source routing are not efficient (with respect to header size), and neither source routing nor QOS are commonly implemented. Furthermore, both DNS and X.500 directory service currently return addresses [Mo], not QOS or source routing fields.

Therefore, to implement flexible and dynamic hierarchical header routing using the QOS or source routing fields, significant changes to current systems and protocols are required. However, relatively few changes are required to current systems and protocols to make addresses more flexible and dynamic. So, while fundamentally this paper argues for dynamic hierarchical *header routing*, using *addresses* for that purpose is the most expedient way to do it.

7. NSAP means Network Service Access Point. It is the address used for OSI internetworking. ISDN is Integrated Services Digital Network.

4.0 Scaling and Directory Service

In this section, the role of directory service as a necessary function for scaling is discussed.

Of the types of header routing, only hierarchical header routing scales well.[8] At first glance, the reason seems obvious—because hierarchical addresses result in smaller routing tables. However, source routing, as specified by IP or CLNP, eliminates the need for routing tables in routers altogether, but nobody is under the illusion that (non-hierarchical) source routing scales well.

Consider a simplistic (and unrealistic) implementation of non-hierarchical source routing where the routing tables are eliminated, and instead directory service returns an entire source route. In this case, directory service would require R^2 entries, one for each path from each of R routers to all other routers.

This is the same overhead (R^2) as in non-hierarchical table routing, where each of R routers must keep a next hop entry for all other routers.[9] (With non-hierarchical table routing, directory service is not needed because the address serves as the name.) Indeed, in a sensible implementation of source routing where the router calculates the source route (rather than directory

8. Of course, if a network is small enough with respect to the memory, bandwidth, processing, and human resources (to operate and maintain routing), then the overhead of flat (non-scaling) routing may be perfectly acceptable. This paper is only concerned with networks where the overhead of flat routing is not acceptable.

9. By overhead, I simply mean number of entries in memory. While the true overhead of routing also involves messages sent, CPU processing, and human resources for maintenance, the number of entries as an indication of overhead suffices for this discussion.

service), each router would obtain a complete topology database (just as is done with a link-state version of flat routing), and from this derive the full path to every destination router (rather than derive just the next hop as would be done with link-state flat routing [MRR]).

With hierarchical routing, both directory service and routers get involved. However, the overhead of routing is (optimally) $RHR^{1/H}$, because each of R routers must maintain $R^{1/H}$ entries for each of H hierarchy levels [KK]. The overhead of directory service is R, because directory service must store one address for each router. So, the total overhead is $R(1+HR^{1/H})$, a significant reduction compared to R^2.

With hierarchical routing, part of the routing function is in routers (table routing) and part is in directory service (header routing). However, we have carefully chosen which functions go where in order to best take advantage of the characteristics of directory service and routers. Directory service works efficiently as long as the answer given by directory service is independent of the source of the query. This was not the case in the source routing example. This is the case, however, with hierarchical addresses, since the hierarchical address only gives the path from the top of the hierarchy down[10] and is therefore independent of the source. Table routing, then, finds the paths to the backbones, which is far more efficient than finding paths to all routers.

The above analysis considers only one path from source to destination. Therefore, it does not take policy routing into consideration. With policy routing, we want to find P paths between router pairs, not just one. However, similar arguments apply. With non-hierarchical routing (source or table), there are PR^2 entries (P paths between R^2 router pairs). With hierarchical routing, it is not sufficient for table routing alone to find P paths. There must be P addresses as well (that is, header routing must also find P paths). This is because with single addressing there will be P paths to the backbone, but routing will still only be able to route through one backbone (the one in the address) to reach the destination. The most common policy in policy routing, however, is to choose between multiple backbones (see section 6). Therefore, in addition to P paths in table routing, there must be P addresses. To reiterate the main point of this paper, in order to have multiple addresses, addressing (header routing) must be more dynamic and flexible than it is currently.

10. The hierarchical address can also determine the path from the bottom of the hierarchy on up, if the source address is used by routers. This is discussed in section 6.

5.0 Dynamic and Flexible Addresses

There are two well-known ways to implement dynamic routes in table routing, and both of those ways apply to header routing as well. The first way is to statically preassign a set of routes for each destination, and then allow choices from the preassigned set to be made dynamically. This is how routing is done in the telephone system. The second way is to dynamically calculate one or more routes from scratch, based on the current network conditions. This technique is common in connectionless data networks. In general, the latter technique is more robust, since it will find routes after any arbitrary set of failures, whereas the former technique can react only to limited failures. For networks with good reliability and stable topologies, however, the two techniques are for all practical purposes equally robust.

For dynamic header routing, only the statically-preassigned, dynamically-chosen technique is appropriate. One reason for this is simply that dynamic assignment of addresses is largely beyond the state of the art.[11]

11. The author has done work in the area of dynamic address assignment, called Landmark Routing [Ts]. However, this work is mainly applicable to networks where static administration of addresses has high cost or is not feasible, such as a rapidly deployed military network.

Another reason is that header routing is good for gross path control, whereas table routing is better for fine path control. For instance, header routing will determine which backbone to traverse, while table routing will determine exactly which routers to use to traverse the backbone. Therefore, header routes need only be re-assigned (to a router or host) when gross topology changes occur, such as establishing or breaking long-term connectivity with a major backbone (as opposed to short-term connectivity changes due for instance to equipment failure). Since this happens relatively infrequently, compared to the frequency of connection establishments, the assignment of header routes can be relatively infrequent without hampering the effectiveness of dynamic header routing.

Header routes, therefore, are dynamic in the following way:

1. **Get address set from directory service.** When a source wishes to establish communications, it obtains multiple header routes (addresses) from directory service (X.500, Domain Name System, the phone book, or whatever). This is the statically pre-assigned set of addresses for the destination.
2. **Prune address set based on policy.** Based on the type of service desired by the source, it will pick zero or more of the header routes appropriate for this particular communications. Note that the "source" can be the host or user, or can be a "policy server" of some kind.
3. **Negotiate address set with destination.** The chosen header routes will be conveyed to the destination, which can reject some or all of them.
4. **Establish communications using one address.** Communications is established using the first of the set of header routes.
5. **Change address if current address fails.** If a failure or other degradation in path quality occurs that table routing alone cannot resolve, then another of the addresses is chosen, and communications continues.

Scaling is provided by the hierarchical addresses. Policy is provided by the address choosing and negotiation in steps 1 through 3 (section 6 discusses this further). Robustness is provided by the modification of the header Route in step 5, in concert with dynamic table routing. The following paragraphs further discuss the above steps.

First notice that step 1, obtaining multiple addresses from directory service, does not significantly increase the overhead of directory service. The number of queries to directory service does not change, just the size of the response (several addresses instead of one). Also, multiple addresses does not change the way directory service currently operates. Both the Domain Name System and X.500 can return multiple addresses.

Steps 2 and 3, pruning the set of addresses based on policy, is a new function. However, it is a desirable function. If indeed the choice of one backbone (or set of backbones) vs. another will influence the quality or cost of communication, then the endpoints of that communication would like to be able to make the choice. If the choice of one backbone vs. another does not influence the quality or cost of communications, then the choice is trivial and no changes to current practice are required. Note that the choice does not necessarily have to be made at the host, but could be made by a policy server between the host and directory service. Policy issues are further discussed in section 6.

Steps 3 and 5 require modification to TCP and its ISO counterpart, TP4. Currently, TCP and TP4 use one and only one network address (IP address and NSAP address respectively) to identify a connection. If the network address changes during a connection, TCP and TP4 will not recognize the newly-addressed packets as belonging to the same connection. This is an example of where the address has not been recognized as serving two functions. The transport protocol assumes that the entire address is serving the "what" function only. Put another way, the transport protocol is perfectly happy to let table routing change the path a packet takes, but not header routing.

The change required to TCP and TP4 is simple. The connection request packet, rather than carrying one address, needs to carry a set of addresses[12]. This is the set of addresses that the source is willing to use for the connection. The connection response packet would also carry a set of addresses, but pruned by the destination. This pruned set would be the allowable set of addresses for the connection.[13] Each endpoint would then establish connection state so that a packet received with any of the allowable addresses would be recognized as belonging to that connection. Notice that this does not affect fast path header processing, because the implementation can be optimized to assume that the incoming packets will use one particular address. Since the connection will normally change addresses only due to failure or severe congestion, this will normally be a good assumption. Notice also that the absence of these changes would not prevent the use of multiple addresses as a policy mechanism. Instead, it would reduce the potential robustness.

In order to do dynamic address changing during a connection, there must be some way for routers to inform hosts that no path can be found using the current address. This is done with the ICMP Network Unreachable message and its ISO 8473 equivalent error message. The only change to current practice is that, upon receiving this message, a host would try a new address rather than give up the connection. Again, without this change, only robustness suffers, not policy or scaling.

6.0 Policy Routing Considerations

In this section, policy routing is briefly discussed. Because of space limitations, this topic is not treated as thoroughly as it should be.

12. This set of addresses could be conveyed in a separate "out-of-band" packet as well.

13. Because the destination can prune the set of addresses, it is entirely possible that the data packets of the connection will take a different path from the connection establishment packets.

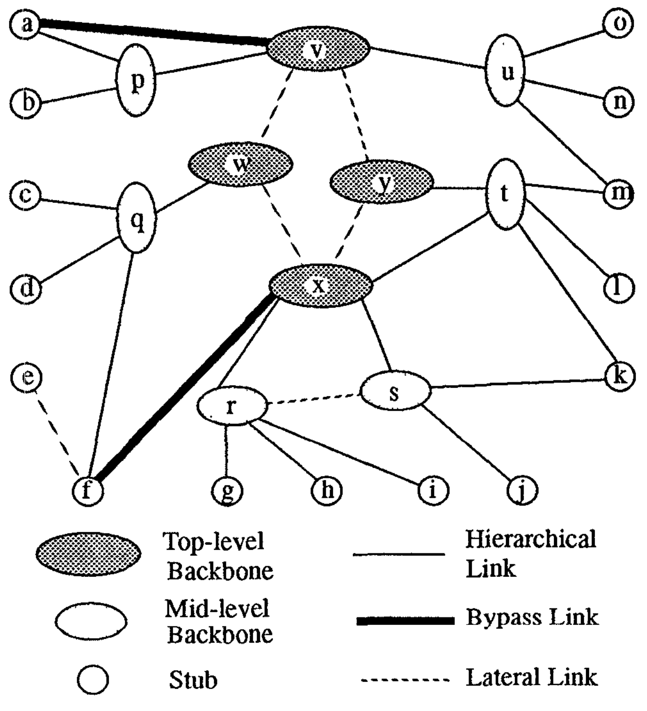
Simply put, the task of policy routing is to find a path from source to destination that:

1. satisfies the minimum performance requirements of the application,
2. satisfies any constraints placed on the path by source, destination, or backbones, and
3. gives the best price/performance (for whoever is paying—source, destination, both, or neither).

While in the abstract this policy statement allows for arbitrarily complex policies, in reality I believe policies will generally be limited to a relatively simple subset of all possible policies. This belief is substantiated below.

The Internet topology is and will continue to be predominantly hierarchical, but with a significant number of lateral and bypass connections [BE]. Figure 3 is an example Internet topology similar to that shown in [BE]. Given the hierarchical topology of the Internet, we can describe a typical path between two stubs, say b to k, as going up the hierarchy (b-p-v), going across the hierarchy (v-w-x), and going down the hierarchy (x-t-k).

Figure 3: Example Internet Topology



Generally, the destination hierarchical address determines the exit top-level backbone (x) and some or all of the down path from the exit top-level backbone to the destination stub (t-k). I say generally because there are several options as to how a hierarchical address can be composed.[14] If a router is able to look at both the source and destination addresses when making the forwarding decision, then the source hierarchical address generally determines the entry top-level backbone (v) and some or all of the up path from the source stub to the entry top-level backbone (b-p). The only part of the path not determined by the addresses then is the across path from the entry top-level backbone to the exit top-level backbone. This part is determined by table routing.

Because the across path is determined by table routing, the source and destination do not have explicit control over the across path, as they do the up and down paths. However, I argue that in fact the major policy decisions have to do with choosing the up and down paths. Once these are chosen, determining the across path is usually trivial.

First, consider that usually there is no across path at all. The top-level backbones are usually national or international backbones. Therefore, many paths will go to the top level backbone and come right back down, or will go directly from the entry top-level backbone to the exit top-level backbone without transiting a third top-level backbone. In these cases, the addresses alone determine the entire path.

For the remaining cases, where there are one or more top-level backbones between the entry and exit top-level backbones, the following simplifying case almost always applies. Almost all backbones are one of a relatively small number of backbone types, and groups of similar type backbones typically coordinate with each other to form a contiguous *backbone system*. For instance, there is a system of IP backbones (the Internet), of X.25 backbones, of telephony backbones, and in the future there will be systems of ISDN backbones and no doubt others. Also, there are systems of military backbones, commercial backbones, and research backbones, to name a few. In the large majority of cases, the entry and exit backbones denoted by the source and destination addresses will both belong to one of these systems. Since these systems typically form contiguous topologies, the across path will typically consist of backbones from the same system.

Table routing alone, either of the link-state or distance-vector variety, can efficiently find homogeneous backbone paths by identifying the backbone types in the routing updates, and making routing decisions that maintain the backbone type of each path.

The type of policy not efficiently handled by table routing would be one where a stub desires an across path that does not match the policies of any backbone system. For example, assume that the top-level backbones in Figure 3 (v, w, x, and y) belonged to a single class, but stub b decided that it didn't want its messages going through backbone y (say because y was known to give poor service), although all other stubs were willing to go through y. This is inefficient because it forces backbone v to know the individual policy of stub b. If many stubs harbored such individualistic policies towards various backbones, especially those more than 1 or 2 backbone network hops away, then backbones would be required to maintain specific policy information for a large number of stubs.

I believe this "non-contiguous" type of policy to be rare. A non-contiguous policy is one where a network X imposes a policy on a network Y that is not above or below X, and is not a neighbor of X, and for which the networks between X and Y do not recognize the policy as belonging to a common policy type. Indeed, it is difficult to construct many plausible non-contiguous policies. (In the example above, if b didn't like y's service, it is likely that other stubs wouldn't either.) Yet, these are the only types of policies that aren't

14. The terms up, down, and across do not imply that one backbone is somehow subservient to the other or even smaller in either geographical scope or number of subscribers. The terms have only to do with the structure of their addresses. Indeed, two backbones could be part of each other's hierarchical address space, in which case up-ness or down-ness is determined only by which address is being carried in a packet.

efficiently handled by multiple addresses combined with table routing.

One might argue that billing policies can easily be non-contiguous. For instance, perhaps the reason that stub b wanted to avoid backbone y is because backbone y is expensive for b (but not to other stubs under backbone p). However, even billing policies are subject to the simplifying observation that billing relationships are generally formed between networks close to each other, usually neighbors, or at most networks above and below each other. In other words, billing policies tend not to be of the non-contiguous type.

Indeed, there are two factors that discourage far-away billing relationships. First, the cost of a single far-away billing relationship (for instance, in different states or different countries) is more than the cost of a single close-by billing relationship. Second, there are many more far-away things than close-by things. As a result, a stub will typically be billed by its mid-level backbone or at worst its top-level backbone, which will tell the stub how much it charges to reach various destinations. Since the charges usually depend only on the distance to the destination and the service provided, and not on how the destination was reached, the stub is not concerned with the across path, and therefore table routing is sufficient.

There are cases where simply choosing the backbone does not imply the across path. For instance, assume that a backbone X offers a high speed service for free to a particular group of stubs. If one of those stubs chooses an address indicating X, it is not clear whether the stub wants high speed service or free service. If table routing has two paths to the destination, one high speed but expensive, and another free but low speed, it will not know which path to choose.

There are at least two solutions. The first is to use the Quality-of-Service (QOS) parameter (implicit header routing). By indicating "high-speed" or "low-cost" in the QOS parameter, table routing knows how to route the packets. The second way is to assign multiple address spaces to the entry backbone, one for each of the characteristics that must be distinguished by table routing. Through table routing, X would advertise one of its address spaces to the high speed neighbor, and the other to the low cost neighbor. This way, the return path would be correct. X would have to look at the source address to correctly route on the forward path.

While this has the negative effect of proliferating addresses for the stubs, it has the positive effect of simplifying the forwarding function for routers, which would not have to look at the QOS field. I believe that the latter choice may very well be preferable. In any event, either solution appears satisfactory.

7.0 Other Aspects of Multiple Hierarchical Addresses

Perhaps the most serious disadvantage of multiple hierarchical addresses is the burden it places on the forwarding algorithm in routers. The forwarding algorithm is the one that searches the routing table for the appropriate next hop when a data packet or call setup is received. With multiple hierarchical addresses, the router must search multiple entries to determine even if the packet is destined for a destination within the router's private domain. With single addresses, a single compare will yield this result.

However, this problem may be minimized if most of the traffic in a private domain has its source and destination in the private domain. In this case, the search can be optimized for the case where the routing table entry for the "internal" address space is checked first. The internal address space is one that is not derived from any backbone, and can therefore be permanently assigned to the private domain. This address space is used for all internal communications, is not advertised externally, and provides address stability in an environment where "external" (backbone derived) addresses may change relatively often.

Another aspect of multiple hierarchical addresses is that address assignments to hosts may change often. This is mainly a disadvantage if there is no protocol for making these assignments easy. As network management becomes more advanced, address configuration (as well as the configuration of other information, such as policy databases, for instance) becomes easier. However, it may also make sense to incorporate address assignment into the intra-domain routing algorithm. This is because one aspect of intra-domain routing (both link-state and distance-vector) is database dissemination. With multiple hierarchical addresses, all hosts and routers in a private domain add or delete the same address prefix. Therefore, it would straight-forward to disseminate the addition or deletion using the intra-domain routing protocol. This would greatly simplify those aspects of address assignments peculiar to multiple hierarchical addresses.

Another concern is the proliferation of addresses due to complex topologies between a stub and its top-level backbones. For instance, if there are several levels of backbone between a stub and the top-level backbone, and each level has multiple backbones with rich connectivity above and below, then there may be a large number of backbone paths between the stub and the top-level backbones, and therefore a large number of addresses. However, in fact hierarchies tend to be shallow, thus preventing this address explosion. The telephone network and Internet have at most three levels, including stubs[15].

In addition, one does not have to have one address for every possible path. Since table routing handles what header routing does not, one can, by not encoding intermediate hierarchy levels in the address at all, always increase the overhead of table routing in order to keep header routing overhead within acceptable levels.

Finally, there remains the question of how a source or destination knows, when it is choosing an address, what type of backbone is represented by that address. The appropriate solution to this is for directory service to return backbone class information along with each address. Both the DNS [Mo] and X.500 standards are flexible as to what kind of information can be returned, so this is possible without modifying standards. Current implementations do not return this kind of information. However, even before backbone class information is available from directory service, since 1) many backbone systems all fall under a single address space (for instance, X.25 networks all use X.121), and 2) there should be a relatively small number of top-level backbones or backbone systems in the world (perhaps several hundred), it should not be difficult for sources to maintain tables of all backbone class information.

15. By "level", I mean levels of separately administered backbones, not levels of hierarchical address or levels of switching hierarchy.

8.0 Summary

This paper has concerned itself with issues of scaling, robustness, and policy routing. While the focus of this paper is internetworking, it applies to large-scale communications networking of any kind.

This paper argues that the only way to achieve scaling is through hierarchical addresses. However, hierarchical addresses constrain the paths found by routing to follow the backbones denoted by the hierarchical address. In other words, the hierarchical address acts like a partial source route. Since both policy and robustness require that multiple paths be discovered, the use of hierarchical addresses, and therefore scaling, is counter to the goals of policy and robustness.

Since policy and robustness require multiple paths, and since a hierarchical address by and large determines a path, it follows that to achieve policy and robustness and scaling, we need multiple hierarchical addresses. However, the notion of multiple hierarchical addresses goes against the grain of conventional thinking. This paper therefore attacks certain accepted fundamental principals—specifically, the notion that an address signifies "where" an addressed object is (while a name signifies "what" the object is, and a route signifies "how" to get to the object). In reality, there is no such thing as "where" in networking, only "what" and "how". The role of the hierarchical address is a combination of "what" and "how".

In particular, the problem with the notion of "where" is that it implies that an object is in one "place" at a time (meaning that it needs only one address), and that the address changes only when the object "moves" (meaning that addresses are rather static). The main point of this paper is that there should be multiple addresses, and the use of addresses should be dynamic, meaning that one should be able to change addresses easily, even during a connection.

Instead of the tri-functional taxonomy of naming, addressing, and routing, this paper argues for a bi-functional taxonomy—naming (or identifying), and routing. This paper then partitions routing into two types—header routing and table routing. Header routing is the routing information in the packet header (mainly the hierarchical address, but also QOS information and source routing), and table routing is what is more traditionally thought of as simply "routing". That is, the algorithms that establish and maintain the routing tables.

This paper then considers further aspects of scaling, and in particular that the function of directory service returning a hierarchical address is an integral part of routing, without which routing would not scale. Given then that directory service and hierarchical addresses are an integral part of routing, we propose that, to achieve scaling, policy, and robustness, 1) stub networks should have multiple addresses, typically one for each backbone they derive service from, 2) that directory service should return multiple hierarchical addresses, 3) that, as a policy decision, the source and destination choose from the multiple hierarchical addresses the most appropriate ones, and 4) that all the chosen addresses be used to identify a transport connection, and that the address can change during a connection to respond to failures. This method of operation adds little overhead to directory service, provides a powerful policy mechanism that fits in well with traditional networking protocols, and provides a mechanism for reacting to failures.

Finally, this paper considers in some detail how multiple hierarchical addresses, combined with table routing and QOS indicators, provides a rich and almost always adequate policy functionality.

REFERENCES

[BE] Breslau, L., Estrin, D., "Design of Inter-Administrative Domain Routing Protocols", proceedings of ACM SIGCOMM '90, September 1990, pp. 231-41.

[Ha] Hauzeur, B.M., "A Model for Naming, Addressing, and Routing", ACM Transactions on Office Information Systems, Vol. 4, No. 4, October 1986, pp. 293-311

[Ka] Katz, D., Merit Computer Network, Private Communications, April, 1990

[KK] Kamoun F., Kleinrock L., "Optimal Clustering Structures for Hierarchical Topological Design of Large Computer Networks," Computer Networks, Vol. 10, No. 3, 1980, pp. 221-248

[Mo] Mockapetris, P.V., "Domain names - implementation and specification", RFC-1035, USC/Information Sciences Institute, November 1987.

[Ro] Rosen, E.C. "Exterior Gateway Protocol (EGP)", RFC-827, USC/Information Sciences Institute, October 1982.

[Sh] Shoch, J. F., "Inter-Network Naming, Addressing, and Routing", Proc. 17th IEEE Computer Society International Conference, September 1978, pp. 72-79

[Ts] Tsuchiya, P.F., "The Landmark Hierarchy: A New Hierarchy for Routing in Very Large Networks", SIGCOMM'88, August 1988, pp. 35-42