Landmark Routing: Architecture, Algorithms, and Issues

Paul F. Tsuchiya

May 1988

MTR-87W00174

SPONSOR:
Defense Communications Agency
CONTRACT NO.:
F19628-86-C-0001

The MITRE Corporation
Washington C^{3}I Division
7525 Colshire Drive
McLean, Virginia 22102-3481

ABSTRACT

This paper is the second in a series of papers that document the research, development, specification, implementation, and deployment of a new routing technique called Landmark Routing. Landmark Routing is a distributed and adaptive hierarchical routing protocol for use in networks and internets of any size. Its primary features are that it is robust and durable in the face of rapid topological changes, that it is easy to administer, and that it provides full name-based addressing. The reason for these advantages is that Landmark Routing dynamically establishes its own hierarchy and modifies it as the network undergoes changes. This paper gives a medium to high-level design of all of the components of Landmark Routing—the routing algorithms, the hierarchy maintenance algorithms, the administrative zones, and the name-to-address binding algorithms. This paper also discusses ancillary issues such as transition from existing routing schemes and implementation design decisions. The paper concludes that Landmark Routing is a workable solution to a host of large network routing problems.

Suggested Keywords: Routing, Hierarchical networks, Hierarchies, Landmark routing, Landmark hierarchy, Addressing, Naming, Address binding, Packet-switching data communications

ACKNOWLEDGMENTS

I would like to extend my appreciation to the many people in the DARPA and ISO networking communities who provided critical feedback during the early stages of this work. In addition, thanks are due to Shannon Bleakley, Mary Pageau, and Richard Wilmer for their painstaking editing. Finally, I extend special thanks to Worth Kirkman, Robert Stine Jr., and John Weidner, for their original and creative technical contributions.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Concluded)

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

INTRODUCTION

This paper is the second in a series of papers that document the research, development, specification, implementation, and deployment of a new routing technique called Landmark Routing. Landmark Routing is a distributed and adaptive hierarchical routing protocol for use in arbitrarily large networks and internets. Its primary features are that it is robust and durable in the face of rapid topological changes, that it is easy to administer, and that it provides name-based addressing. The reason for these advantages is that Landmark Routing dynamically establishes its own hierarchy and modifies it as the network undergoes changes. This is the first routing protocol with this capability.

In addition, Landmark Routing has features designed to facilitate its operation in the existing Department of Defense (DoD) Advanced Research Projects Agency (DARPA) and emerging International Organization for Standardization (ISO) internet environments. In particular, it embraces the concept of separately administered networks which require some level of autonomy and protection from each other.

This work is supported by the Defense Communications Agency (DCA), and specifically by the Defense Communications System Data Systems organization, which manages the Defense Data Network (DDN). This work is motivated by limitations in current routing protocols in handling the large networks and internetworks that are emerging today.

The first paper in this series, *The Landmark Hierarchy: Description and Analysis*, analyzed in detail the efficiency of the Landmark Hierarchy in its static state. It showed that the Landmark Hierarchy exhibits routing table sizes and path lengths similar to those seen in the traditional area hierarchies.

This paper provides a medium to high-level design covering all aspects of Landmark Routing—dynamic management of the hierarchy, the routing algorithms, name-to-address binding, administrative boundaries, and implementation in existing networks. This paper explores the feasibility of accomplishing all goals set for Landmark Routing by stating how each goal can be accomplished. Second, it documents the intended design of Landmark Routing so that a larger community of experts can become involved and provide critical feedback. Third, it provides a basis for future work. This paper does not, however, analyze or simulate the designs. That will be the next phase of this project.

THE LANDMARK HIERARCHY

Landmark Routing is a hierarchical routing scheme based on the Landmark Hierarchy. The Landmark Hierarchy is different from the traditional area hierarchy. In the area hierarchy, routers are grouped into areas so that 1) all routers in an area have a routing entry for all other routers in the area, and 2) there is a path that does not leave the area between any two routers in the area. Areas are then grouped into super-areas, and so on to form a hierarchy (the telephone network is a well-

known example of an area hierarchy). Routers outside of an area view the area as a single entity, thus reducing the amount of routing information needed to address routers in that area.

Whereas an area is a group of routers *all of which have a routing table entry for each other*, a Landmark Vicinity is a group of routers *all of which have a routing table entry for a single router*, namely, the Landmark. The Landmark, then, is at the center of a Landmark Vicinity, and every router r hops away from the Landmark has a routing table entry for that Landmark. A hierarchy of Landmarks is formed by having all routers be Landmarks with small Landmark Vicinities, a portion of those routers be Landmarks with larger Landmark Vicinities, a portion of those be Landmarks with still larger Landmark Vicinities, and so on until there are a few routers network-wide whose Landmark Vicinity covers the whole network. Whereas in the area hierarchy a router is addressed by its membership in areas, a router in a Landmark Hierarchy is addressed by its proximity to Landmarks. Figure 2 gives an example.

Here, Router a is the lowest-level Landmark; its vicinity is shown by the circle defined by radius r_0. Router b is the next level Landmark, and so on. The Landmark Address of Router a is therefore $c.b.a$. This is because Router a is closer to b than to any other level 1 Landmark, and because b is closer to c than any other level 2 Landmark. We call a a child of b, because a has chosen b as part of its Landmark Address. Likewise, we call b a parent of a. The individual components of the Landmark Address are Landmark Labels.

Assume we wish to find a path from the router labeled *Source* to Router a. *Source* will look in its routing tables and find an entry for c because *Source* is within the Landmark Vicinity of c. *Source* will not, however, find entries for either b or a, because *Source* is outside the Vicinity of those Landmarks. *Source* will choose a path towards c. The next router will make the same decision as *Source*, and the next, until the path reaches a router which is within the radius of b. When this router looks in its routing tables, it will find an entry for b as well as for c. Since b is finer resolution, the router will choose a path towards b. This continues until a router on the path is within the radius of a, at which time a path will be chosen directly to a. This path is shown as the solid arrow in Figure 2.

There are two important things to note about this path. First, it is, in general, not the shortest possible path. The shortest path would be represented by a straight line directly from *Source* to a. This increase in path length is the penalty paid for the savings in network resources which the Landmark hierarchy provides.

The other thing to note is that often the path does not necessarily go through the Landmarks listed in a Landmark Address. This is an important reliability consideration in that a Landmark may be heavily congested or down, and yet a usable path may be found using that Landmark (or, more literally, using previous updates received from that Landmark).

ROUTING IN A LANDMARK HIERARCHY

Given that a router needs to know how to route to any Landmark it sees, we must find a routing algorithm appropriate for deriving these routes. There are two fundamental types of distributive-adaptive routing algorithms—link-state (or New ARPANET) and distance-vector (or

Old ARPANET). Of these two, only distance-vector routing can work in the Landmark Hierarchy. This is because link-state requires a full topology map to calculate routes. Because routers only know of individual Landmarks, and not of every router and link between it and the Landmark, link-state cannot be used to calculate routes to a Landmark. (Note that because an area in the area hierarchy can be abstracted down to a single router, link-state routing can work in an area hierarchy.)

Distance-vector routing, on the other hand, is appropriate for Landmark Routing, since it only requires knowledge of the destination router itself (i.e., the Landmark). A long-standing problem with distance-vector routing, however, is the count-to-infinity problem. Explained briefly, this problem arises where routing loops form (for instance, where Router a routes to Router b, b to c, c to d, and d back to a). These loops are not fixed for a period of time during which routing updates are incrementing the distance seen to the destination to some pre-established value meaning infinity.

This problem has been vigorously attacked over the last fifteen years, most notably by McQuillan, Jaffe-Moss, Hagouel, and as recently as this year (1987) by Garcia-Luna. All of the proposed solutions result in a period of time after the distance increase (which we call the hold-down time) during which the destination is labeled as unreachable (whether it is or not). The latter three solutions, in addition, generate significant routing traffic during this time (although the end result is that the time is shorter).

Not being happy with any of these solutions, we have developed a solution, called Alternate-path Distance-vector Routing (ADR), which exhibits no hold-down time. Discovery of a new route after an old route has increased in distance is usually instant. This is because valid alternate routes are pre-established at the same time (or nearly so) that primary routes are found. This solution requires 1) two or three times the memory of the latter three count-to-infinity solutions and 2) roughly the same link bandwidth.

Loosely stated, ADR works as follows. To avoid the count-to-infinity problem, no latent routing information downtree from a destination (towards the leaves on the spanning tree formed from the destination by the routing tables) can be used to establish new routes after a distance increase to the destination is detected. Therefore, the only routers that can be trusted to provide new routing information are downtree routers that join other segments of the spanning tree. We call these routers juncture routers, because separate segments come together and form a juncture at these routers. The trick, then, is to identify juncture routers.

This is easily and efficiently done through what we call Juncture Configuration (JC) messages. JC messages follow normal routing messages downtree. JC messages label the various branches of the spanning tree, allowing juncture routers to identify themselves.

Juncture routers then use the parameters derived from the JC message to send an Alternate Path Priming (APP) message back downstream. This message provides downstream routers with their alternate routes. Because these routes were provided by juncture routers, they are free from the count-to-infinity problem. Because these routes are provided before any distance increases, any router knows instantly what its new route will be, or indeed if there is a new route at all, when it

sees a distance increase. After a distance increase, the old alternate route immediately becomes the primary route, and routing messages, JC messages, and APP messages are sent to establish new alternate routes.

Because each router has a notion of downtree and uptree, and because each router knows the distance to the destination via an alternate route, significant path splitting can be accomplished using the alternate routes. To do this, routing decisions are made not only based on the destination address, but also based on which link (downtree or uptree) the message arrived. A consistent alternate routing policy is enforced to prevent loops.

ADR is used by routers to both determine their distance in hops from a Landmark for the purpose of maintaining the Landmark Hierarchy, and to determine their distance in some other metric for the purposes of routing messages to that Landmark. As such, we may have multiple, completely separate instances of ADR running. The one using hops must be implemented by all routers participating in Landmark Routing. In the absence of any other routing metric, it can be used to route to a Landmark. As such, it also serves as the common routing metric for routers in different administrations that otherwise do not agree on a routing metric.

Any other instances of ADR are based on whatever metrics are appropriate, and can be confined to localities determined by what we call Administrative Zones. In our initial implementation of Landmark Routing, we plan to use a static metric that 1) is administratively assigned link-by-link and router-by-router, 2) is meant primarily to reflect bandwidth capacity of a link, and 3) fairly and and evenly spreads traffic over available resources.

Other metrics, however, are possible. Multiple metrics can be accommodated by running multiple instances of ADR.

DYNAMICALLY MANAGING THE LANDMARK HIERARCHY

In the area hierarchy, a partition occurs when some routers (or areas) in an area (or superarea) may not communicate without going outside the area. When this happens, communications cannot take place, because the routing function assumes that once a message has entered an area, it will be able to get to any destination in that area. There are known techniques for handling limited instances of the area partition. However, for the more general case, reorganization of the area hierarchy is required. Research has identified several major problems with dynamic reorganization of the area hierarchy:

1. **Potentially inefficient hierarchy structure.** The choice of clusters can effect hierarchy performance.

2. **Amount of coordinated decision making.** There are instances in the maintenance of an area hierarchy where a group of routers must synchronize to make a decision. This is a complex process.

3. **Contention problems.** The same resources can be required by different clusters, requiring contention resolution procedures.

4. **Completeness problems.** Routers can be left out in the cold because they have no cluster membership.

In the Landmark Hierarchy, a partition occurs when a parent Landmark does not see (have a routing table entry for) its child Landmark. This is because traffic will route to the parent Landmark, but the parent Landmark will not be able to route it further. There are three ways to repair the Landmark Hierarchy partition based on three types of partitions:

1. The child can still see its parent, but the parent can no longer see the child.

2. The child can no longer see its parent, but can see another Landmark with room for more children at the same level as the parent.

3. The child can see no valid Landmarks at the level of its parent.

The first two cases are easy to handle. In the first case, the child simply increases Landmark radius to encompass the parent. This is easy, because the radius is determined by a single field in the Landmark Update packet. In this case, no address change has taken place.

In the second case, the child adopts a new parent, and adjusts its radius accordingly. This causes a change of address for all descendents of the child Landmark. The Assured Destination Binding technique is then used to establish the new addresses. Based on previous work, we know that these first two cases will constitute the large majority of partitions.

In the third situation, elections must take place to reestablish the hierarchy above the Landmark which sees no parents. This is the only situation where something more complex than the simple increase in radius and possible subsequent rebinding of addresses is required. Even so, the election process is straightforward. This is largely because the actual choice of Landmarks is not an important determination of the efficiency (routing table sizes and path lengths) of the Landmark Hierarchy. There is significant leeway with regards to the number of Landmarks at each level (a factor of 3 or 4) and the specific choice of Landmark. The efficiency of the Hierarchy is determined by the radii chosen by Landmarks. If Landmarks are far apart, the radii are larger; if they are close together, the radii are smaller. Because there is so much leeway in the choice of Landmarks, over-electing or under-electing by even one or two factors does not cause a problem. This frees the election process from any tight constraints on timing, thus making it simple.

In general, we try to avoid the second and third types of partitions. This is because 1) until the binding function completes, some nodes will not have communications, and 2) the binding function itself may generate a surge of traffic if it is to complete quickly. Therefore, adjustments are usually made in the hierarchy *while it is not partitioned.*

The purpose of these non-partitioned adjustments is to create as even a distribution of Landmarks as possible—not for the sake of efficiency, since it is the radii that determine efficiency, but to avoid partitions. Non-partitioned hierarchy adjustments can be made with a minimum of perturbation to the network, because while the adjustment is taking place, the old addresses can be kept valid while the new address bindings are happening. This way, there is no interruption of

communications, and there is no surge in binding traffic because the bindings can be spread over a large period of time.

There are three adjustments used in a non-partitioned hierarchy:

1. A Landmark adopts a new parent.

2. A level i Landmark demotes itself to a level $i-1$ Landmark.

3. A level i Landmark is promoted to a level $i+1$ Landmark via the election process.

The first adjustment occurs when a Landmark finds itself significantly closer to a potential new parent than its existing parent. The second adjustment occurs when a Landmark does not have enough children to justify being a Landmark. The third adjustment occurs when the Landmark is not close enough to any potential parents.

Finally, we note that there are procedures used to 1) minimize the number of nodes dependent on any single Landmark for their address, and 2) cope with merging Landmark Hierarchies. These procedures are discussed in the main body of this paper.

ADMINISTRATIVE BOUNDARIES AND AUTONOMY IN LANDMARK ROUTING

One of the disadvantages of the Landmark Hierarchy is that it does not, by nature, recognize boundaries—boundaries between different network types, between differently administered networks, between groups of nodes which communicate extensively, and so on.

It is therefore necessary to add boundaries to the Landmark Hierarchy in order to accommodate certain requirements. We classify those requirements into three categories:

1. A group of routers should be able to manipulate routes so that all traffic between two group members never transits routers outside of that group. A variation on this is that a group of routers should be able to still route traffic between group members in the face of routing failures in routers outside that group.

2. A group of routers should be able to manipulate routes so that all traffic between two non-group nodes will never transit routers within the group. A further refinement on this is that a group of routers should be able to select which non-group nodes can transit traffic through group routers.

3. A group of routers should be able to operate certain aspects of their routing protocols (metrics used, frequency of updates, types of service) differently than another group of routers.

More succinctly put, routers should be able to choose certain routes, prevent certain routes, and should be able to prioritize routing information. In addition, groups of routers should be able to

act with some limited autonomy from other groups. Finally, any router should be able to belong to several groups, whose relationships may be nested or overlapping.

We satisfy these requirements by creating what we call Administrative Zones (or just Zones for short). Zones are similar to areas in the area hierarchy in that all routers in a Zone must be able to communicate without leaving the Zone. Zones are different from areas, however, in that routing can still take place even if a Zone becomes partitioned.

A Zone can be auto-configured along with the rest of the Landmark Hierarchy. The only preconfiguration required is to the border routers of Zones (those routers that share a link with non-Zone routers). They must be told which links cross into other Zones. This way, border routers are always able to label Routing Updates and Landmark Updates (or just Updates for short) received from these links as coming from other Zones. When routers configure (that is, choose parent Landmarks) into the Landmark Hierarchy, they delay configuration with non-Zone routers until the Zone is completely configured—there is a Landmark in the Zone that has no peers (other Landmarks at the same level) inside the Zone. We call this Landmark the Zone-root. It is the Zone-root Landmark that subsequently configures with Landmarks from other Zones.

Because Zone routers base their Landmark addresses on Zone Landmarks, all traffic between Zone members will stay within the Zone. If Zone border routers do not let Updates that came from outside the Zone leave the Zone, then non-Zone routers will not see that Zone as providing a path to other non-Zone nodes.

Zones can be hierarchically layered and can overlap. Nodes in overlapping Zones, however, will have multiple addresses; one for each overlapping Zone it is in.

As stated previously, the hop-based Landmark Update provides the least common routing metric used for inter-Zone communications. Within a Zone, however, any additional routing metric(s) may be used in Routing Updates. These Routing Updates are simply not sent over Zone boundaries. Additional metrics (or metric combinations) require additional sets of Routing Updates in an additive fashion. This provides a significant amount of routing autonomy within a Zone.

ADDRESS BINDING IN LANDMARK ROUTING

Clearly one of the difficulties of Landmark Routing is that the Landmark Address of a router can change at any time, even though that router's point of attachment to the network does not change. This requires that all nodes (hosts and routers) in the network be identifiable by something other than their Landmark Address, and that it is possible to determine the current Landmark Address of a node from that identification (this identification is referred to as both an ID or a name).

The fundamental problem in any address binding scheme is locating the node which is holding the binding—that is, since the address of the desired destination is not known, then at least the address of a node that does know must be known. In the DARPA Domain Name system, the node (the name server) that knows the address of a given destination node, or at least where to search for it, is embedded in the name of that destination node. Any node that might generate a

binding query must know the addresses of the name-servers that it might need to query for bindings. This kind of binding technique will not work well in Landmark Routing because the address of the name server itself may change, thereby making it difficult for any node to keep a list of name server addresses.

In Landmark Routing, we use a different approach, called Assured Destination Binding (ADB), to find the address of the appropriate name server (or just server). Instead of putting the name of the server in the host name and then using a table lookup to find the address of the server, we derive the address of the server directly through an algorithmic manipulation on the host name—namely, a hash function. This hash function maps the name directly into the Landmark Address space (this is not quite literally true, but is conceptually accurate), thus unambiguously pointing to the address of a server. It does so in a uniform fashion, thus evenly distributing the mapping over the address space. The name itself requires no semantics (such as an embedded name server) other than the address semantics which is vicariously derived through the hash function.

The obvious problem with this is that the address found by the hash function can be any arbitrary Landmark Address. There is a high probability that the actual address will not exist anywhere in the network since the address space will be sparsely populated. To solve this problem, we rely on a simple resolution function which maps the hashed address into some real address. For instance, this resolution can be a simple series of increments to the hashed address until it matches a real address, wrapping around to the lowest number if necessary. As long as this resolution consistently maps to the same real address from anywhere in the network, the hash function followed by resolution is all that is needed to identify a server for any given named network entity.

For this resolution to be consistently mapped to the same server from all routers, resolution has to take place over a set of Landmarks which is identical for all routers that might be resolving the hashed ID. Therefore, the hash and resolution functions take place multiple times, once for each level of the hierarchy, starting at the global level. When a router wishes to either query or update an address, it first hashes the ID into the global Landmark Label (or just Label, for short) space, resolves the hash result to the next higher real global Landmark, and sends the update or query towards that Landmark. When the update or query reaches one of the offspring of that global Landmark, it executes the hash again, but this time into the Landmark Label space allocated for the children of the global Landmark, again resolves it to one of the children, and sends the query or update towards that Landmark. This continues until the query or update resolves to a single node. Clearly, updates are sent any time a node gets a new Address for itself, and queries are sent when a node needs the current Address for some destination.

If the Landmark Labels are clustered, the resolutions will not evenly distribute over the Landmarks. Instead, the routers with Labels at the bottom of the clusters will receive more updates and queries. To solve this problem, we hash the Labels into an Intermediate Hash Space (one for each level) to evenly distribute them, and then hash the IDs into this space instead of the Label space.

Since certain servers may have a greater capacity for name-serving than others, they can be

hashed into the Intermediate Hash Space more often than those servers with less capacity, thus causing more resolutions to fall on them.

We can gain both efficiency and robustness by sending updates to three or four servers rather than just one. This can be done by appending an octet to the ID, and incrementing this octet for each hash. This is more robust because there is a smaller chance of all servers going down at once. It is more efficient because requesting routers can check all potential servers, and send the query to the closest one.

Some destinations will be more popular than others, for instance a directory service. In order to prevent the servers for these destinations from being unfairly deluged with requests, popular destinations can send their updates to still more servers, resulting in fewer queries for each.

There are three situations that can cause the binding of a node to resolve to a different server even though that node's address did not change (i.e., the node does not immediately know that a new server must get its update). They are as follows:

1. A server obtains a new address, thus causing the hashed ID to no longer resolve to that server.

2. The addition of a Landmark somewhere in the hierarchy above the server causes the hashed ID that previously resolved to that server to resolve elsewhere. (This includes the case where a global Landmark advertises a new server capacity.)

3. A server crashes, thus losing the binding.

The first two cases can be handled on an event-driven basis. In other words, no action is necessary until the change occurs. In the first case, the server simply informs the router which originated the binding that it needs to re-update the binding. In the second case, a server can determine if a change to the Intermediate Hash Space affects any of the bindings it is holding. If it has, it again informs the router that it must re-update its binding. In the third case, there is obviously no way for the server to inform the node that it must re-establish its binding. Therefore, routers must periodically send out updates. Since they are sending to several servers, the time period can be large.

ADB can operate within a Zone so that routers in the Zone can guarantee that intra-Zone bindings will be serviced by Zone nodes (thus making queries and updates more local and limiting them to administratively trusted servers). Bindings must still be sent to non-Zone servers so that non-Zone routers can query successfully.

Finally, a Zone-root can send one binding outside of the Zone for all Zone members. Not only is this more efficient than sending out bindings for all Zone members, it also allows for privacy because the internals of the Zone are not being advertised. For this to work, however, requesting nodes must know a destination's Zone membership—that is, the name has additional semantics similar to those that already exist in Name-Domain names. Further, requests must be

two-part. First, the request goes to the node that says where the Zone is. Then a second request goes to the Zone itself, where it is resolved internally and then answered.

ADB is not a good mechanism for yellow-pages style binding or for binding on character strings whose syntax may not be precise. This is because the key to the hash function must be exactly specified—searches for matching strings or that sort of thing are not possible. Therefore, we assume that ADB will be confined primarily to finding Landmark Addresses given network layer identifiers (such as a DoD Internet Address). Getting the network layer identifier in the first place will still be within the purview of the more traditional, hierarchical name servers. ADB, however, can be used to help search for the name server by giving name servers well-known network layer identifiers.

ARCHITECTURAL AND ENVIRONMENTAL CONSIDERATIONS

Except for the global level, each level of Landmark Hierarchy address can be encoded in three bits. This is because each Landmark is constrained to have from 2 to 5 children (research shows that any more than 5 or 6 children per Landmark, and the performance of the Hierarchy begins to degrade). Twelve bits of address space for the global Landmarks is plenty. This gives around 4000 global Landmarks, or about 8000 table entries, which translates to over 1,000,000 routers in 4 octets. With the 20-octet ISO address space, and using 4 octets for the Landmark Address, we could have room for seven layers of hierarchy, a 4-octet ID to identify hosts, and two octets for the Zone ID.

If we fit Landmark Addresses into DoD Internet Addresses (32 bits), and we assume a class E address, we have 9 bits for the global Landmarks, and 6 levels of three bits each for rest, then we get around 350,000 routers. However, this leaves no room for the host identifier, and so an new option or encapsulation of one DoD IP packet in another would be required to hold the host identifier.

Section 7 also discusses several miscellaneous items related to Landmark Routing. A Landmark Routing implementation is described, giving the functions of neighbor configuration, routing, hierarchy maintenance, Zones, and binding, and their interactions. This discussion shows that there is considerable autonomy between the various functions. This is important when considering the potential effects that one function may have on another.

The problems of automatic configuration over large subnetworks (both broadcast and non-broadcast) are discussed. It is shown that configuration over networks such as the ARPANET can be efficient.

The architectural question of what should be implemented at the network layer, and what at the application layer, is discussed. It is determined that neighbor configuration must be at that network layer, but that everything else (routing, hierarchy maintenance, Zones, and binding) might better be implemented at the application layer.

The problems of interfacing Landmark routers with non-Landmark hosts is discussed. In particular, it should be possible to use existing hosts efficiently without modification.

We discusses how a Landmark Routing routing domain will interface with other, non-Landmark Routing ISO networks. The use of Zones is central to this problem.

Finally, we discuss the relationship between hosts and routers, and in particular, how a router may determine and convey the host ID, both in and out of data packets.

CONCLUSIONS

All the algorithms and technologies that make up Landmark Routing are feasible. Of course, simulation is required to support this claim, and to give performance estimates. The most difficult task in bringing Landmark Routing to Internet-wide implementation will be transitioning from the existing state of affairs to one in which Landmark Routing is the primary routing algorithm. This is true both because of problems stemming from design and implementation bugs in early implementations, and because of the problems associated with maintaining some degree of backwards compatibility with existing techniques.

1 INTRODUCTION

This paper is the second in a series of papers that document the research, development, specification, implementation, and deployment of a new routing technique called Landmark Routing. Landmark Routing is a distributed-adaptive hierarchical routing protocol for use in arbitrarily large networks and internets. Its primary features are that it is robust and durable in the face of rapid topological changes, that it is easy to administer, and that it provides full name-based addressing. These features arise from the fact that Landmark Routing dynamically establishes its own hierarchy and modifies it as the network undergoes changes. This is the first routing protocol with this capability.

In addition, Landmark Routing has features designed to facilitate its operation in the existing Department of Defense (DoD) Advanced Research Projects Agency (DARPA) and emerging International Organization for Standardization (ISO) internet environments. In particular, it embraces the concept of separately administered networks that require some level of autonomy and protection from each other.

1.1 Motivation

This work is supported by the Defense Communications Agency (DCA), and specifically by the Defense Communications System Data Systems organization, which manages the Defense Data Network (DDN). There are two similar problems which have motivated this work.

The DDN operates several long-haul packet switching networks for use by Department of Defense (DoD) subscribers. These networks are growing, and are scheduled to merge in the future. The size of these networks is becoming such that the efficiency of the existing non-hierarchical routing (Shortest Path First (SPF)) is questioned (Sparta, 1986), (Khanna, Seeger, 1986).

An even more severe problem is that of the growth of the Internet (The DDN plus connected networks, such as the NSFNET). This growth is greater than that of the DDN alone, and is now pushing the limits of the existing gateway routing protocols, for instance the Gateway-to-Gateway Protocol (GGP), and the Exterior Gateway Protocol (EGP) protocol. A further problem here is that, unlike SPF, the gateway routing protocols are not very robust.

Finally, Landmark Routing is attacking both the problems associated with separately administered networks, and those of address administration.

1.2 Scope of this Document

The first paper on Landmark Routing is "The Landmark Hierarchy: Description and Analysis" (Tsuchiya, 1987). This paper studies the Landmark Hierarchy alone—that is, without any regard to its use in a dynamic environment. The point of that paper is to determine the efficiency of the hierarchy in terms of the routing tables sizes and path lengths. The whole idea behind the use of any hierarchy in routing is to reduce the amount of routing information that must be spread around. The performance penalty paid for this reduction is increased path length. A hierarchy is overall beneficial if the reduction in overhead from decreased routing information outweighs the increase in overhead from longer paths. If this benefit doesn't exist, then there is no reason to have the hierarchy at all. The first paper shows that the Landmark Hierarchy provides this benefit. In particular, it shows that the routing table sizes and path lengths are on the order of those seen in area hierarchies.

Having convinced ourselves that this is the case, the next step is to do a medium to high-level design covering all aspects of Landmark Routing—dynamic management of the hierarchy, the routing algorithms, name-to-address binding, administrative boundaries, and implementation in existing networks. This paper documents that design. It first explores the feasibility of accomplishing all of the goals set for Landmark Routing by stating how each goal can be accomplished. Second, it documents the intended design of Landmark Routing so that a larger community of experts can become involved and provide critical feedback before the simulation and implementation stage of the work. Third, it provides a basis for future work.

This document leaves a lot of detailed issues for further study. None of the issues are the sort that may prevent Landmark Routing from being a success. Instead, these issues are things like what value should a timer or other parameter be set at, what is the optimal trade-off between this thing and that thing, and so on. In addition, this document does not do any kind of detailed performance analysis on its algorithms. The next stage of work, then, is to nail down these design and performance issues. This will be done primarily using simulation, and will be documented in the third paper.

The fourth paper will take the results of the third paper, and turn them into a protocol specification suitable for implementation. This implementation will be tested in increasingly hostile and complex environments (i.e., real world) as we move towards a public domain implementation of Landmark Routing.

1.3 Outline

This paper is divided into five major sections (the Introduction, Landmark Hierarchy description, and Conclusions make up the rest). We have attempted to make each section independent in that one does not need a complete and thorough understanding of any one section in order to understand another. One should be able to read the Executive Summary, Section 2, the Landmark Hierarchy description, and the Glossary of Definitions, and then be able to grasp any of the remaining sections individually. Later sections, however, do tend to build on earlier ones, so that if one wants to read the whole paper, one should read it in the order presented. Since, we have included the major results of the previous paper, this paper stands alone as a complete document.

Section 2 of this document explains the basic concepts behind Landmark Routing and the Landmark Hierarchy. Along with parts of Section 4, it contains the major results of the previous paper.

Section 3 discusses the algorithm(s) used to establish routes to a Landmark. Since any routing algorithm of the distance-vector variety (also known as Bellman-Ford, Old ARPANET, Tajibnapis, and others) will work in a Landmark Hierarchy, we start this section with a review of distance-vector routing and its problems in general, and of several known techniques in particular. Not being particularly happy with any existing technique, we develop a new technique for distance-vector routing. This technique, called Alternate-path Distance-vector Routing, improves on previous techniques in that there is no convergence time to discover new routes upon the degradation of an existing route. It does this by pre-establishing alternate routes before a primary route goes bad. It is not confined to use in a Landmark Hierarchy, and can be used in any routing architecture.

Section 4 discusses the algorithms used to dynamically maintain the Landmark Hierarchy. To justify the use of the Landmark Hierarchy over the area hierarchy, we present a brief review of pertinent work in the field of dynamic management of area hierarchies. We then present some of the analysis and major results of the previous paper (Tsuchiya, 1987) to establish a framework for the remainder of this section. The rest of Section 4 discusses the design trade-offs and the algorithms proposed for dynamic management of the Landmark Hierarchy.

Section 5 discusses the incorporation of administrative boundaries into the Landmark Hierarchy. This is necessary because, by its very nature, the Landmark Hierarchy has no hard boundaries of the sort seen in the area hierarchy. Real networks, on the other hand, are

characterized by hard boundaries that delineate one administration's networks from another's. Even within the scope of a single Landmark Hierarchy, different administrations require 1) control over where their packets travel with respect to other administrations' networks, and where other administrations' packets travel with respect to their own networks, 2) some amount of autonomy from other administrations' network operation, and 3) protection from failures in other administrations' networks. This section, then, describes how these boundaries are implemented (through what we call Administrative Zones), and how the Zones and the Landmark Hierarchy are statically and dynamically managed.

Section 6 discusses the problem, and our solution, of dynamically binding non-changing names to changing addresses. This problem is endemic to Landmark Routing in that addresses of nodes may change at any time because of changes in the Landmark Hierarchy (which in turn result from topological changes in the network itself). This characteristic of changing addresses poses special problems with regards to binding names to addresses. We have developed a general solution to this problem called Assured Destination Binding (Stine and Tsuchiya, 1987). In this section, we describe the specific application of Assured Destination Binding to the Landmark Routing environment. This solution accommodates mobile hosts as a matter of course.

Section 7 is a potpourri of architectural and implementation issues. It starts by discussing addressing in Landmark Routing, in particular the structure of Landmark Addresses using both the DoD IP address, and the ISO IP address (the Network Service Access Point (NSAP) Address). Next, it ties together the preceding four sections by describing how each function interacts in an implementation. Third, it discusses the problem of automatic configuration over large subnetworks (both broadcast and non-broadcast). Fourth, it discusses the architectural question of what should be implemented at the network layer, and what at the application layer? Next, it discusses the problems of interfacing Landmark routers (Intermediate Systems (ISs), or gateways) with End Systems (ESs, or hosts). In particular, it should be possible to efficiently use existing ESs without modification. Next, Section 7 discusses deployment issues. It determines that Landmark Routing can be implemented for ISO IP (ISO 8473), but not DoD IP. Finally, it discusses how a Landmark Routing routing domain can interface with other, non-Landmark Routing networks.

Section 8 gives conclusions from the previous sections, and gives general commentary. The conclusion is that all of the algorithms and technologies that make up Landmark Routing will work. Of course, simulation is required to support this claim, and to give performance estimates. The most difficult task in bringing Landmark Routing to Internet-wide implementation will be

transitioning from the existing state of affairs to one in which Landmark Routing is the primary routing algorithm. This is true both because of problems stemming from design and implementation bugs in early implementations, and because of the problems associated with maintaining some degree of backwards compatibility with existing techniques.

Appendix A is a glossary of the mathematical terms used in this paper. Appendix B is a glossary of definitions of (non-mathematical) terms. Although the terms are defined as they appear, we recommend that the reader look at this second glossary before reading the text.

2 THE LANDMARK HIERARCHY

We first describe the Landmark itself. Then, we describe a hierarchical structure built from Landmarks. Third, we describe how routers are addressed in a Landmark hierarchy. Finally, we show how routing may take place with the Landmark hierarchy. This description is available in the previous work (Tsuchiya, 1987), but is repeated here for completeness.

2.1 The Landmark

The description of a Landmark is simple. A Landmark is a router whose neighbor routers within a certain vicinity contain routing entries for that router. Determination of the vicinity is based on hops; that is, the distance between any two routers that share a link is measured as one.[1]

As an example, consider Router 1 in the network of Figure 1. Routers 2 through 6 have routing entries for Router 1 (as indicated by the arrowheads) and are therefore able to forward any packets addressed for Router 1 to Router 1. Routers 7 through 11 do not contain routing entries for Router 1. Therefore, Router 1 is a Landmark which can be "seen" by all routers within a distance of 2 hops. We refer to Router 1 as a Landmark of radius 2. This is distinguished from an area in the area hierarchy, where the routers in a given area *all have routing table entries for each other.*[2]

2.2 The Landmark Hierarchy

Next, let us consider a hierarchy built from Landmarks. The nomenclature LM_i refers to a Landmark of hierarchy level i, i=0 being the lowest level, and i=H being the highest level. Throughout this paper, the subscript i is reserved to mean a hierarchy level. The nomenclature $LM_i[id]$ refers to a specific LM_i with label id, called the Landmark Label, or just Label for short.[3]

Each $LM_i[id]$ has a corresponding radius $r_i[id]$. In the Landmark hierarchy, every router in a

---

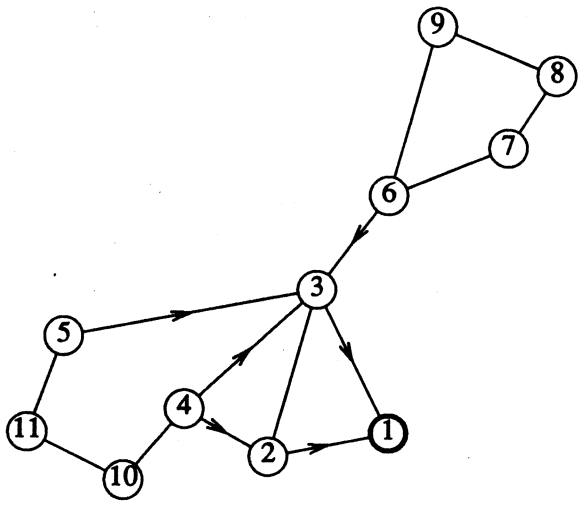
[1]We also require that all links be full duplex. The class of routing algorithm required for Landmark Routing (Distance-Vector) does not work with simplex links (see Section 3.2). We do, however, have techniques to account for link metrics other than hops, and for different metric values for each direction of a full duplex link. This requires that multiple routing algorithms be run in parallel.

[2]Appendix A contains a glossary of the mathematical terms used in this paper, Appendix B a glossary of other terms.

[3]The notation here varies slightly. In general, we use id generically to mean any single Landmark (as distinguished from meaning the average characteristics of a set of Landmarks). When it is necessary to distinguish between specific Landmarks in the same expression or sentence, we will use either id_x, id_y, or simply x, y. The latter form is usually in reference to a figure or specific example.

[4]Note that not all $r_i[id]$ are necessarily equal. In other words, $r_i[id_x]$ may or may not be equal to $r_i[id_y]$.

Figure 1
A Single Landmark



network is a Landmark $LM_0[id]$ of some small radius $r_0[id]$. Some subset of $LM_0[id]$'s are $LM_1[id]$'s with radius $r_1[id]$, and with $r_1[id]$ almost always greater than $r_0[id]$, so that there is at least one $LM_1[id]$ within $r_0[id]$ hops of each $LM_0[id]$. Likewise, a subset of the $LM_1[id]$'s are $LM_2[id]$'s, with $r_2[id]$ almost always greater than $r_1[id]$, so that there is at least one $LM_2[id]$ within $r_1[id]$ hops of each $LM_1[id]$.[4]

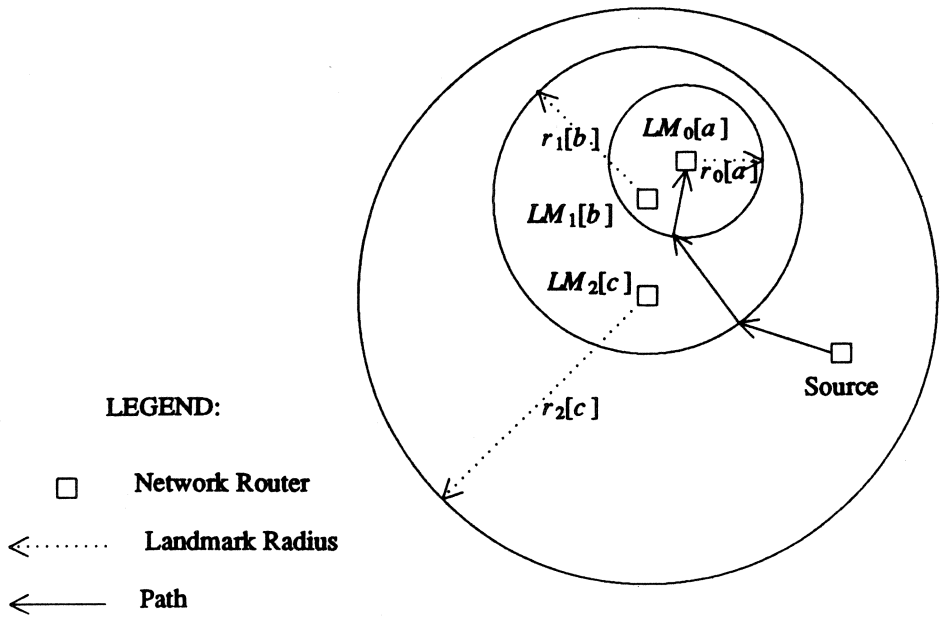
These iterations continue until a small set of routers are $LM_i^G[id]$'s each with an $r_i^G[id]$, with $r_i^G[id] \geq D$, D being the diameter of the network. Because the radius of these Landmarks is larger than the diameter of the network, all routers in the network can see these Landmarks. We call these global Landmarks, and give them the superscript G. The reason for this structure will become clear in section 2.5.[5]

[5]The network diameter is the distance between the two routers in the network furthest from each other.

Figure 2 illustrates the Landmark hierarchy by showing a portion of a network. This is a two-dimensional representation (meaning that only routers drawn physically close to each other would share a link). For simplicity, only four of the routers are shown, and no links are shown. The dotted arrows and circle indicate the radius of the Landmarks; that is, the vicinity within which routers contain routing entries for that Landmark. For instance, every router within the circle defined by $r_1[b]$ has an entry for, and can route to, $LM_1[b]$. Since a router at level i is also a Landmark at all levels $x < i$, it will have Landmark Labels for each level. Again for simplicity, the routers in Figure 2 are labeled only with the Landmark Labels which are pertinent to the examples herein.

In general, Landmark Labels only need to be locally unique, except at the highest level. The requirements for local uniqueness are discussed in Section 4.

Figure 2
Landmark Hierarchy



2.3 Routing Table

Each router in the network keeps a table of the next hop on the shortest path to each Landmark for which it has routing entries. Each router will therefore have entries for every $LM_0[id]$ within a radius of $r_0[id]$, every $LM_1[id]$ within a radius of $r_1[id]$, and so on.

Since every router is an LM_0, and since every router has entries for every $LM_0[id]$ within a radius of $r_0[id]$, every router has full knowledge of all the network routers within the immediate vicinity. Likewise, since a portion of all LM_0 are LM_1, every router will have knowledge of a portion of the network routers further away. Similarly, each router will have knowledge of even fewer routers further still, and so on. The end result is that all routers have full local information, and increasingly less information further away in all directions. This can be contrasted with the area hierarchy where a router on the border of an area may have full local information in the direction within the border, but virtually no local information in the direction across the border.

2.4 Addressing in a Landmark Hierarchy

In an area hierarchy, the address of a router is a reflection of the areas at each hierarchical level in which the router resides. The telephone number is a well-known example of this. In a Landmark hierarchy, the address of a router is a reflection of the Landmark(s) at each hierarchical level which the router is near. The Landmark Address (or just Address, for short), then, is a series of Landmark Labels: $LM_i^G[id_i].LM_{i-1}[id_{i-1}]. \ . \ . \ . \ .LM_0[id_0]$.

There are two constraints placed on Landmark Addresses. First, the Landmark represented by a given Landmark Label must be within the radius of the Landmark represented by the next lower Landmark Label. For instance, the router labeled $LM_0[a]$ in Figure 2 may have the Landmark Address $LM_2[c].LM_1[b].LM_0[a]$. In this case, we call $LM_2[c]$ a parent of $LM_1[b]$, and we call $LM_1[b]$ a child of $LM_2[c]$. In this paper, the terms parent and child will always refer to two Landmarks, the lower of which is using the higher as part of its address. The address of the router labeled $LM_0[a]$ could be $LM_2[c].LM_1[e].LM_0[a]$ if and only if there existed a Landmark $LM_1[e]$ (not shown) which was within the the radius of the router labeled $LM_0[a]$. The reason for this constraint will become clear in Section 2.5. Since more than one Landmark may be within the radius of a lower level Landmark, routers may have many unique addresses. Multiple addresses could be used to provide some traffic splitting.

Now, the set of routers that contain a routing table entry for a Landmark is not the same as the set of routers that use that Landmark for their Landmark Address. In general, the set of routers that

contain a routing table entry for a Landmark is larger than the set of routers that use that Landmark for their Landmark Address. The former set overlaps with analogous sets for other Landmarks (in other words, a router will typically have routing table entries for several Landmarks at a particular level), while the latter set usually does not overlap with other analogous sets (in other words, the Address tree is a strict tree).

2.5 Routing in a Landmark Hierarchy

Now we may consider how routing works in a Landmark Hierarchy. Assume we wish to find a path from the router labeled *Source* to the router labeled $LM_0[a]$ in Figure 2. The Landmark Addresses for the router labeled $LM_0[a]$ is $LM_2[c].LM_1[b].LM_0[a]$. The basic approach is the following: *Source* will look in its routing tables and find an entry for $LM_2[c]$ because *Source* is within the radius of $LM_2[c]$. *Source* will not, however, find entries for either $LM_1[b]$ or $LM_0[a]$, because *Source* is outside the radius of those Landmarks. *Source* will choose a path towards $LM_2[c]$. The next router will make the same decision as *Source*, and the next, until the path reaches a router which is within the radius of $LM_1[b]$. When this router looks in its routing tables, it will find an entry for $LM_1[b]$ as well as for $LM_2[c]$. Since $LM_1[b]$ is finer resolution, the router will choose a path towards $LM_1[b]$. This continues until a router on the path is within the radius of $LM_0[a]$, at which time a path will be chosen directly to $LM_0[a]$. This path is shown as the solid arrow in Figure 2.[6]

There are two important things to note about this path. First, it is, in general, not the shortest possible path. The shortest path would be represented in Figure 2 by a straight line directly from *Source* to $LM_0[a]$. This increase in path length is the penalty paid for the savings in network resources which the Landmark hierarchy provides. This will be analyzed in Section 4.

The other thing to note is that the path does not necessarily go through the Landmarks listed in a Landmark Address. This is more so if the Landmark vicinity for an LM_i goes beyond an LM_{i+1}. This is an important reliability consideration in that a Landmark may be heavily congested or down, and yet a usable path may be found using that Landmark (or, more literally, using previous updates received from that Landmark).

---

[6]In point of fact, *Source* might well see an $LM_1[b]$, because Landmark Labels are only locally unique with respect to their siblings. It would not, however, see an $LM_1[b].LM_2[c]$.

2.6 Landmark Hierarchy Example

To better illustrate the Landmark Hierarchy, consider Figure 3. This network has 3 hierarchical levels. All routers (small circles) are LM_0. Diamonds denote LM_1, and circles denote LM_2^G (again, the superscript G means that the Landmark is global). The rightmost address component is the $LM_0[id]$, and for this example is unique for each router in the network. The middle address component is the $LM_1[id]$ and indicates proximity to an LM_1, and the leftmost address component is an $LM_2^G[id]$, and indicates proximity to an LM_2^G. All $r_0[id] = 2$ hops, all $r_1[id] = 4$ hops, and all $r_2[id] = 8$ hops.

Figure 3
Landmark Routing Example

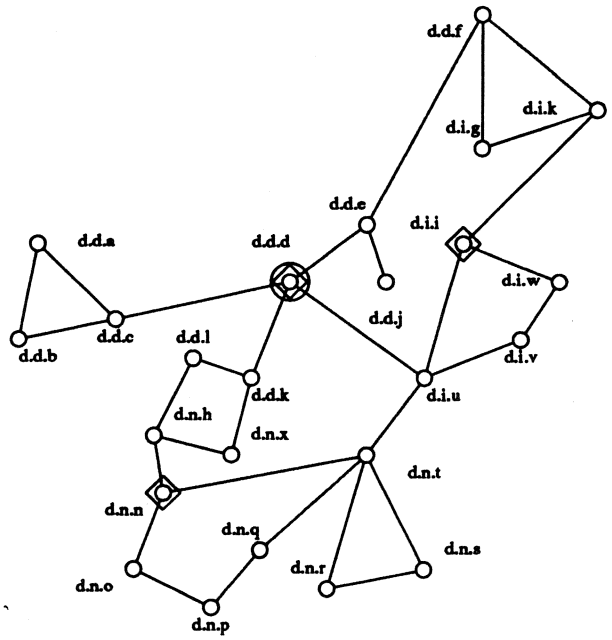


Table 1 shows the routing table for Router g in Figure 3. This length of this table has been optimized by including only one entry per router, even if that router is a Landmark at several different levels. Router g has less than one-fourth of the total network routers in its routing table.

Table 1
Routing Table for Router g of Figure 3

Landmark	Level	Next Hop
$LM_2[d]$	2	f
$LM_1[i]$	1	k
$LM_0[e]$	0	f
$LM_0[k]$	0	k
$LM_0[f]$	0	f

Let's consider a routing example where Router *g (d.i.g)* is routing a message to Router *t (d.n.t)*. Router *g* examines Router *t*'s Landmark Address—*d.n.t*—and does not find entries for either $LM_0[t]$ or $LM_1[n]$ in its routing table. Router *g* does, however, have an entry for $LM_2[d]$, and therefore forwards the message towards $LM_2[d]$ via Router *f*. Router *f* also does not have entries for $LM_0[t]$ or $LM_1[n]$, and therefore forwards the message towards $LM_2[d]$ via Router *e*. Router *e* does have an entry for $LM_1[n]$ (but not $LM_0[t]$), and forwards the message towards $LM_1[n]$ via Router *d*. Router *d* does have an entry for $LM_0[t]$, as does Router *u*, and the message is delivered. The resulting path, *g—f—e—d—u—t*, is 5 hops, 1 hop longer than the shortest path, *g—k—i—u—t*.

3 ROUTING IN THE LANDMARK HIERARCHY

In this section, we consider how routing tables are established through the exchange of routing messages in the Landmark Hierarchy. We consider this apart from the problem of maintaining the Landmark Hierarchy itself. Put another way, we consider the problem of establishing routing tables assuming that the Landmark Hierarchy (the selection of Landmarks and their radii) is established and correct.

3.1 Overview of Routing Techniques

At a fundamental level, all routing comes down to one thing: a switch must decide on which outbound queue to place a message (or which circuit to establish) based on the destination address of the message (or call setup) or on some other information such as quality-of-service or logical channel number. This decision is made by querying a routing table. When we speak of routing techniques, however, we are not speaking of the act of querying the routing table and queueing a message; rather, we are speaking of how the routing table gets established in the first place.

Routing techniques can be partitioned into two broad classes, static and adaptive. (For a good overview of routing techniques, see papers by Gerla (Gerla, 1984) or Schwartz (Schwartz and Stern, 1980).) In static routing, routing tables are established at a certain time, before any data is being transmitted, and the routing table is not changed afterwards. Within this class, there are two sub classes, alternate and non-alternate. In alternate static routing, more than one choice of route is available. All routes may be used simultaneously, or secondary routes may go unused until primary routes are full or down. Alternate static routing can be very robust in environments where links stay up for long periods of time, and where traffic patterns are known in advance. Public circuit-switched telephone networks use static alternate routing.

In adaptive routing, the network environment (status of links, traffic levels, etc.) is monitored, and routing tables are dynamically adjusted to adapt to changing network conditions. We can partition adaptive routing techniques into two major sub classes, centralized and distributed. In centralized adaptive routing, switches monitor network changes and send them to a central machine, which calculates new routes and distributes them back to the switches. While this may sound straightforward, it is in fact difficult to do because 1) the central machine and the switches must have routes to each other in order for the routes to be distributed (chicken and egg), and 2) synchronizing the updates among all switches is not an easy task.

In distributed adaptive routing, switches monitor network status, inform each other of changes, and calculate routes themselves. There are two classes of distributed adaptive routing, distance-vector and link-state. These two descriptive terms were coined by Radia Perlman and are not yet in wide use in the literature. These are the terms which have been adopted in the recent American National Standards Institute (ANSI) work on routing and will be used here. In distance-vector routing, neighbor switches trade lists of their distances to every destination with each other, keeping the shortest distance for routing. Eventually, all routing tables will converge to shortest path. In link-state, switches distribute lists of the state of each of their links to all other switches in the network. Upon receiving these lists, each switch is able to build a consistent topology map of the network and then calculate routes based on the topology. Link-state routing is currently in use in the ARPANET and is often called Shortest Path First (SPF), New ARPANET, or Dijkstra's algorithm.

3.1.1 Comparison of Routing Techniques for Use in Landmark Routing

Of the routing techniques mentioned, centralized adaptive and link-state adaptive are not appropriate for Landmark Routing. This is because both of these techniques require real-time routing calculations based on the entire topology (links and routers) of the network. The benefit of the Landmark Hierarchy is that it allows each router to route without knowing the entire topology of the network, thus saving network resources. Using either of these techniques would defeat the purpose of the hierarchy.

One could use existing distance-vector routing techniques in the Landmark Hierarchy without significant modification. The only difference between how they would be used in the Landmark Hierarchy and how they would be used in non-hierarchical routing is that, in the Landmark Hierarchy, distance-vector routing updates are distributed only a certain distance (the radius of the Landmark) rather than network wide.

However, we have designed a distance-vector routing technique, called Alternate-path Distance-vector Routing (ADR), that improves upon any previous versions of distance-vector routing algorithms. The rest of this section discusses the various aspects of distance-vector routing—its history, its methods of operation, its problems, previous solutions, and our solution.

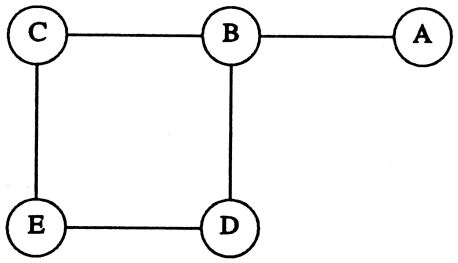
3.2 Evolution of Distance-Vector Routing

This section presents an evolution of distance-vector routing. For a thorough exposition of all relevant work in this field up to 1983, see Hagouel's thesis (Hagouel, 1983). (Distance-vector routing schemes are variously referred to in the literature as ARPANET, Old ARPANET, Bellman-Ford, Ford, Tajibnapis, and others.)

One of the earliest versions of the distance-vector routing algorithm was implemented in the ARPANET in the early 1970s. In this version, every router maintains a distance table and a routing table. The distance table contains the distance to every destination for every link the router has. The routing table contains the shortest distance to every destination and the link over which that distance is found. Every router in the network periodically broadcasts its routing table to its neighbors. Upon receiving a broadcast, a router stores in its distance table the distance advertised by the neighbor plus the distance from it to the neighbor for every destination. For routing purposes, a router puts in its routing table the smallest distance seen in the distance table. Then, when a router has a packet to route, it looks in the routing table to find the link over which the shortest path to the destination can be found.

To see how routing information is propagated this way, refer to Figure 5. We wish to see how routing entries to Router A are established. Assume initially that no routers except Router A have routing entries for Router A. Assume for the moment that all link costs are 1. Router A informs Router B of its identity (essentially, a routing entry for Router A with a distance of 0). Router B takes this and fills in its routing entry for Router A via Link B→A as distance 1. Later, Router B sends its routing table to Routers C, D, and A. Routers C and D then enter a distance of 2 in their routing tables for Router A via Links C→B and D→B respectively. Now consider what happens when Router C broadcasts its routing tables to Routers B and E. Router E will create an entry of distance 3 to Router A via Link E→C. Router B, however, will create and entry of distance 3 to Router A via Link B→C. After all of the routing tables have been exchanged several times and the algorithm converges, Router B will have routing entries for Router A over all its links. Those entries will indicate that Router A is reachable at distance 1 over Link B→A, at distance 3 over link B→C, and at distance 3 over link B→D. Since distance 1 is Router B's shortest distance to Router A, Router B will use Link B→A to route messages to Router A.

Figure 4
Distance-Vector Routing Example



3.2.1 The Count-to-Infinity Problem

The obvious problem is created when Router B finds that the value of Link B→A is increased, say to 20. Now Router B sees that it is distance 20 to Router A via Link B→A, but only distance 3 via Links B→C and B→D. Router B will therefore route messages to Router A via Router C or D. Of course, Routers C and D see Router B as the best path to Router A, and so will send the message right back to Router B, thus creating a loop.

It will take some amount of time for the exchange of routing tables to eliminate this loop. Notice that, during the next exchange of routing tables, Router B will inform Router C that its new shortest distance to Router A is 4. Router C will take this plus 1 as its new shortest distance, and later tell Router B. Router B will now have a distance of 6 to Router A via Router C. This counting up will continue until Router A sees a distance of 21 to Router A via Router C, and will finally correctly choose Link B→A as its Link to Router A. This counting up has been called the count-to-infinity (CT∞) problem.

A solution to this simple form of CT∞ (called the ping-pong loop or predecessor loop, because a message immediately loops back over the same link) was proposed as early as 1975 (Cegrell, 1975) (split-horizon technique), and several more solutions have been proposed since then. In our example, these solutions would either have Router C tell Router B that it is using Router B to get to Router A, or simply have Router C never repeat to Router B anything it heard from router B.

The CT∞ problem, however, still exists. When Link B→A increases to 20, Router B will tell Routers C and D that it is distance 20 from Router A. Routers C and D tell Router A that they are distance 4 from Router A (remembering the updates they got from Router E). Router B then uses either C or D for its path to Router A, and a loop is created around Routers B, C, E, and D. Router B then tells Routers C and D that it is distance 5 from Router A, and count-to-infinity ensues around Routers B, C, E, and D.

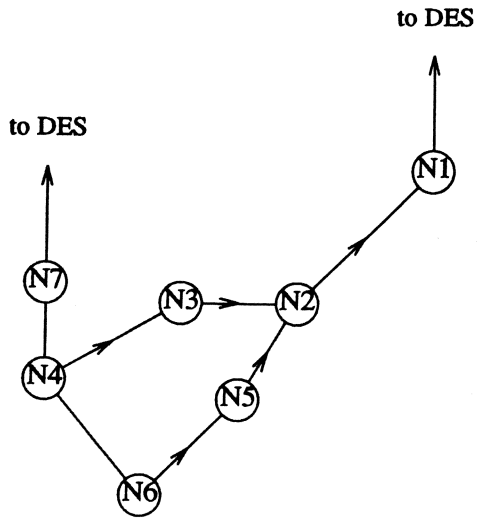
Solutions proposed by Jaffe and Moss (Jaffe and Moss, 1982), and by Hagouel (Hagouel, 1983) further attack the problem by not allowing Router C to tell Router B anything it hears about Router A as long as Router B is on Router C's path to Router A. In other words, the distance table is removed, and each router knows of only one possible path to the destination. Therefore, if the distance to a destination increases, a router will temporarily have no path to a destination, and must actively search out a possibly better path. These schemes trade not being able to use a second path when it exists for avoiding CT∞ when there is no second path. Although the two schemes differ in how they go about searching for a better path, both schemes base their searches on a fundamental characteristic of the CT∞ problem published by McQuillan, Richer, Rosen (McQuillan et al, 1978), which is paraphrased in the following paragraph.

A CT∞ may occur only if a routing update received by a router is based on an update which was previously sent by that router. In other words, a CT∞ will occur if a router is downtree from itself on the spanning tree defined by the routing table entries for a given destination. (Downtree is further from the destination router where the spanning tree is rooted. Uptree is closer to the destination router.) This condition will only occur if the router experiences an increase in its distance to a destination (bad news), but afterwards receives good news (any distances which are shorter than that in the bad news) that is based on information it had previously sent about that destination. This is possible because good news is retained in routers whereas bad news is forgotten. Good news, then, hangs around and eventually overtakes bad news, even if the good news is out of date.

3.2.1.1 The Jaffe-Moss Solution. The Jaffe-Moss scheme is to not let a router indirectly receive old good news from itself by not allowing that router to accept any good news about a destination until every router downtree from it has received the bad news. When a router receives bad news (or discovers it by a local link measurement), it passes that bad news to all other neighbors. If a router receives bad news from a non-downtree router, it acknowledges the router(s) from which it heard the bad news. When a router gets acknowledgments from every router to

which it sent bad news, it acknowledges its uptree routers, and afterwards accepts any good news it hears. To see this, refer to Figure 6.

Figure 5
Count-to-Infinity Examples



Assume that the arrows in Figure 6 denote the next hop from a router towards the destination DES. Assume further that N1 determines that its distance to DES has increased. N1 then sends this bad news to N2, and enters the freeze state. Similarly, N3 and N5 send the bad news to N4 and N6 respectively, and likewise enter the freeze state. When N4 sends the bad news to N7 and N6, both N6 and N7 acknowledge to N4 that they are not downtree from N4, thus allowing N4 to unfreeze. Likewise, N6 will receive an acknowledgement from N4. The wave of acknowledgements travel back downtree to N1, at which time all routers are unfrozen and may accept good news.

Since in the Jaffe-Moss scheme, good news is only sent when it is received (i.e., it is event driven), a router may receive bad news but not receive valid good news for a long time after. For instance, after all of the routers in Figure 6 are unfrozen, N7 may offer N4 a better path to DES than does N3. However, until N7 receives new news itself, it does not inform N4 of its distance to DES for an indeterminate period of time.

The obvious solution might seem to be to allow a router acknowledging bad news to include its current distance. This, however, will lead to CT∞. To see this, return to the situation where N1's distance to DES has increased and a wave of bad news is being sent downtree. Assume that the bad news reaches N6, and N6 forwards it to N4 before the bad news reaches N4 via N3. N4 will acknowledge N6 with its distance to DES via N3. N6 will spread the good news to N5, who will in turn spread it to N2, causing N2 to think that N5 is its next hop to DES. At this point, a CT∞ has occurred from N2 to N5 to N6 to N4 to N3 and back to N2.

Even if we do not allow distances to be reported with acknowledgments, as Jaffe-Moss specifies, CT∞ can still occur. Consider the case where N1's distance to DES decreases. N1 spreads this good news downtree to N2, and N2 passes it on to N3 and N5. Now assume that immediately after this, the link from N2 to N1 experiences an increase in distance. N2 naturally sends bad news to N3 and N5. Assume that the good news and the bad news travels to N4 via N6 before even the good news can travel to N4 via N3. Assume also that the good news that N4 heard doesn't affect its next hop to DES. Then, when N4 hears the bad news from N6, it will acknowledge N6. Assume that shortly thereafter N4 receives the good news from N3. Now N4 tells N6 of this good news, N6, tells N5, N5 tells N2, and again a CT∞ has occurred.

Of course, the scenario described would not happen often, if ever. In addition, Jaffe-Moss presents a variation of their algorithm as a worst-case performance optimization which does appear to prevent CT∞ in all cases. This variation requires that all routers remain frozen until the initiator of bad news receives all of its acknowledgments. The initiator unfreezes its downtree routers with a special unfreeze message. We will not comment further on this variation except to say that, while it may improve worst-case performance, it worsens average performance because of the extra unfreeze message required.

3.2.1.2 The Hagouel Solution. The Hagouel scheme does not prevent CT∞ although it does decrease the probability of such an event. In the Hagouel scheme, a router queries all of its neighbors except its uptree neighbor for a better route when its own path to a destination increases. If a router receiving such a query is downtree from the sender, it doesn't reply, thus avoiding a potential CT∞. If a router receiving such a query is not downtree from the sender and is closer to the destination router than the sender, then the router receiving the query responds. Unfortunately, the responding router has no way of knowing 1) whether it is not downtree on the tree from the router which originally started the wave of queries, or 2) whether it is downtree on the tree from the router which originally started the wave of queries, but has simply not yet received the bad news. If

it is case 1, then no CT∞ will occur. If it is case 2, then old good news is usurping the bad news and a CT∞ may form. This is similar to the example set forth in our discussion of the Jaffe-Moss scheme where a non-downtree router could acknowledge with its distance to the destination.

3.2.1.3 The Hold-down Solution. A third method of dealing with the CT∞ problem is for a router to simply wait a certain amount of time after hearing bad news about a destination before accepting any new news. This method is similar to the Jaffe-Moss and Hagouel schemes in that there is a time delay before new news may be accepted. It is different from those two schemes, however, in that there is no mechanism to actively flush out old news or search for new news. Instead, the router simply waits for a period of time, called the hold-down time, during which it can be sure that the old news has been completely flushed via the normal update process.

In routing schemes where the routing metric is hop count, the hold-down time is the diameter of the network multiplied by the maximum time it takes routing news to propagate from one router to the next. In routing schemes where the link routing metric is some variable quantity, the hold-down is the maximum value of the routing metric multiplied by the routing news propagation time—a much larger value. Therefore, hold-down is only practical in schemes that use hop-count as the routing metric (or as an ancillary metric for the purpose of reducing the hold-down time).

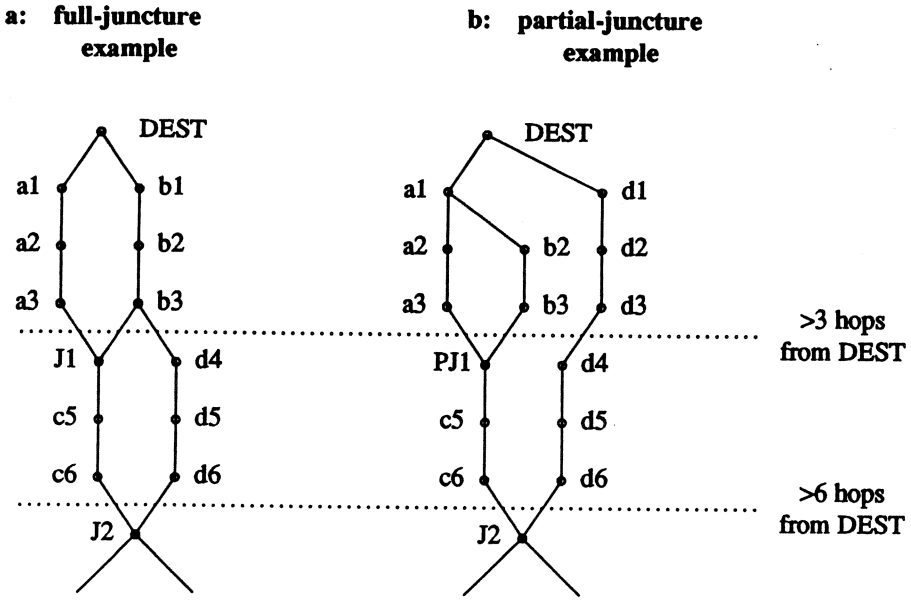
3.2.1.4 The Garcia-Luna Solution. More recently Garcia-Luna put forth yet another solution to the CT∞ problem (Garcia-Luna, 1987). His approach is similar to Hagouel's in that a search downtree for a non-downtree router takes place in the event of bad news. No routing over the increased links takes place until new news if found. Garcia-Luna improves on Hagouel in that he provides a method for forcing bad news sent downtree to stay in the system until new news is found. Without getting into details (the scheme is difficult to explain), a router must *prove* that it is a feasible next hop to a destination before it will be believed.

Unfortunately, there can be significant thrashing before convergence occurs. For instance, consider the network of Figure 7b. Assume that the link between Router *a1* and DEST crashes. Bad news would travel up the two branches from *a1*. One of them, say the one from *a3*, would arrive at *PJ1* before the other, and *PJ1* would believe it has found better news on the other (*b3*) uptree branch of *PJ1*. This causes false good news to travel back uptree towards *a1* via *a3*. Shortly afterwards, the bad news coming up from *b3* would reach *PJ1*. In this case, however, *PJ1* would not think that *a3* offered a better path because it will not have become feasible. (We do not show here exactly how this happens. The reader will have to read the literature). The bad news from *PJ1*

will then chase the incorrect good news down towards *a1* via *a3* and wipe out the good news when it reaches *a1*. In the meantime, the original bad news will have gone up to *J2* via *c5* and find new news there. This new news will travel back down to *a1* via *c6*, and only now will *a1* be able to successfully route to DEST.

This method does indeed prevent CT∞ from occurring, but only after several waves of messages were sent around. This thrashing, however, will not always occur. (We have no analysis to determine how often it would occur). In addition, the thrashing is never worse than the worst-case optimized solution of Jaffe-Moss and so is far superior to the Jaffe-Moss solution.

Figure 6
Juncture Router Examples



3.3 New Solution to the Count-to-Infinity Problem

We are not happy with any of these solutions to the CT∞ problem. Except for the simple hold-down, all of them trade both 1) delay in finding better paths and 2) traffic generated while searching for the better path for avoiding the CT∞ problem. Hold-down only trades off delay in finding better paths for avoiding CT∞.

In this section, we present a technique for dealing with the CT∞ problem called Alternate-path Distance-vector Routing (ADR). ADR requires roughly the same memory and link bandwidth as Garcia-Luna's algorithm (at least when one considers the thrashing problem, which Garcia-Luna does not address in his work). However, it provides *instant* rerouting in response to a metric change (increase or decrease). There is no "hold-down" or "freeze" time while a new path is being found because alternate paths are discovered before bad news occurs.[7]

In what follows, we do not give an exact algorithm for ADR. Instead, we provide a rough description of its operating. An exact algorithm and simulation results will be the topic of future work.

3.3.1 Alternate-path Distance-vector Routing

The Jaffe-Moss, Hagouel, and Garcia-Luna approaches to the CT∞ problem attempt to guarantee that no old news downtree from a router that has experienced a distance increase will be misconstrued as new good news. They do so by attempting to only accept news from routers not downtree from the distance increase. The trick, then, is in recognizing which routers are downtree from a given router and which are not.[8]

Consider two routing tree branches (or just branch for short) A and B emanating from router DEST. Assume that these two branches join at some router J such that Router J could just as easily route a packet to DEST via branch A or via branch B. For instance, if Router J is exactly x hops from DEST via either branch. We call Router J a juncture router. Note that a juncture can occur between two routers, for instance J_A and J_B, if both J_A and J_B share a link, but route to DEST only via their respective branches. In this case, we call both J_A and J_B juncture routers.

If there is a metric increase somewhere uptree from a juncture router (assuming that this is the first juncture router from DEST), then any new routing news will come from or through the juncture router because the juncture router is the first router that has access to another branch. Therefore, when a metric increase occurs, it is only necessary to find the *nearest* downtree juncture

---

[7]Notice that all of the examples herein consider all links to be non-broadcast point-to-point. We model multiple routers on a broadcast medium as a fully connected network of point-to-point links. In Section 7.3 we describe how, for the case where there are many such routers (more than 5 or so), the modeled topology is not fully connected—certain pairs of routers are not modeled as being connected.

[8]To be precise, a router b is considered uptree from another router a if b is on a's best path to a destination. If b is uptree from a, then a must be downtree from b. Otherwise, any two routers a and b are said to be horizontal from each other, even if a is closer to the destination (by some metric) than b.

router to get new routing information. Note that this search need not extend beyond the nearest downtree juncture router because a juncture router further down the tree will only produce a larger routing metric than the closer juncture router and would be ignored anyway. This is important not only because it means that searches can be limited in scope (thus saving both time and communications resources), but more importantly, it means that the juncture router lower down the tree does not need to concern itself with any metric increases which occur higher in the tree than the next higher juncture router (at least, not with respect to its role as a juncture router).

For example, consider Figure 7a. Router *J1* is a juncture router with equal distance paths to DEST. Router *J2* is a juncture router below *J1* with equal distance paths to DEST. If there is a link failure (or any metric increase) on a link between any two routers labeled *aX*, the search for another path need go only as low as Router *J1*. If there is a failure on a link or router downtree from *J1* or *b3*, the search for another path will go to *J2*. Note that *J2* does not care that one of its paths joins one of *J1*'s paths uptree from *J1*. A link or router failure at or uptree from Router *b3* will result in a search to *J1* for routing news, and doesn't concern *J2* at all. In other words, *J2* acts as a juncture router for only a subset of routers uptree from it—namely, those which do not have a closer downtree juncture router. (After the link failure, *J2* will no longer be a juncture router, but this is separate from the issue of routers above *b3* finding an alternate route.)

Consider Router *PJ1* in Figure 7b. It is not a full-juncture router because it does not attach to a separate branch emanating from DEST. In other words, it does not have two different paths back to DEST. It can, nevertheless, act as a juncture router for those routers downtree from *a1* and uptree from itself. We call router *PJ1* a partial-juncture router. For the sake of discussion, the unqualified term "juncture router" will be used to refer to both full-juncture routers and partial-juncture routers.

The question now is how can juncture routers determine that they are juncture routers? Assume that every update has two fields. (Actually, the packet with these fields is not a routing update, but what is called a Juncture Configuration (JC) message, which comes after or is tacked onto the same packet as the normal routing update.) The first field is for the purpose of configuring full-junctures, is called the FJ Tree Label, and contains a router ID. The second field is variable length, is for the purpose of configuring partial-junctures, is called the PJ Tree List, and contains a list of tuples each of which contain a router ID and the number of hops that router is from the destination. These two fields are filled in as follows:

1. The neighbor of the destination puts its ID in the FJ Tree Label.

2. Any router (including any router which modifies the FJ Tree Label field) with more than one downtree link adds to the PJ Tree List a tuple with its ID and number of hops from the destination.

3. A router knows that it is a full-juncture router because of the JC messages it has received on its uptree links; at least one of them has a different FJ Tree Label entry from the others. Any full-juncture router removes all entries in the PJ Tree List and writes its own ID into the FJ Tree Label field.

4. A router knows that it is a partial-juncture router because it has received JC messages over different uptree links, all of which have the same FJ Tree Label entries. Any partial-juncture router does the following:

    a. Finds the most recent entries in the same PJ Tree List slot that match for all of the JC messages received. (Imagine that the first entry placed in the PJ Tree List is in slot 0, the second in slot 1, and so on. Find the JC messages with the least entries, for instance, 3 entries. Then compare the entries in the highest slot, slot 2, for all JC messages. Some JC messages may have a different number of slots. All JC messages will have at least one entry. If all slot 2 entries are the same, then all previous entries for all JC messages must be the same. If two or more slot 2 entries are different, then compare slot 1 entries, and so forth.)

    b. If a match was found, record the hop count found in that field for later use in sending Alternate Path Priming (APP) messages uptree. To generate a JC message to send downtree, take one of the JC messages and remove all PJ Tree List entries from the matching slot and higher. If the matching message was in slot 0, add the router ID and number of hops to the destination in slot 0.

    c. If no matches were found, record the hop count found in slot 0 from each JC message (one per uptree link) for later use. Generate a new JC message to send downtree by taking one of the JC messages, removing all PJ Tree List entries, and adding the router ID and number of hops to the destination in slot 0.

As an example, lets consider the full-juncture case by referring to Figure 7a. The JC messages which originate from DEST and travel up the two paths will be identified as being

different branches by *J1* because they will have been given different FJ Tree Labels by the first Routers *a1* and *b1*. Therefore, *J1* will recognize itself as a full-juncture router. Since any full-juncture router downtree from *J1* doesn't care if its paths join a path uptree from *J1*, *J1* need only label Juncture Configuration messages that it passes downtree with its own ID. In Figure 7a, when *J2* receives Juncture Configuration messages from Routers *c2* and *d3*, they will be labeled differently, and *J2* will recognize that it is a full-juncture router.

The use of the PJ Tree List by partial-juncture routers is more complex. A juncture router is partial because there is only one unique path between it and the destination. In other words, all paths between it and the destination must pass through some individual router. A juncture router recognizes that it is partial because all of the entries in the FJ Tree Label field of the JC message are identical.

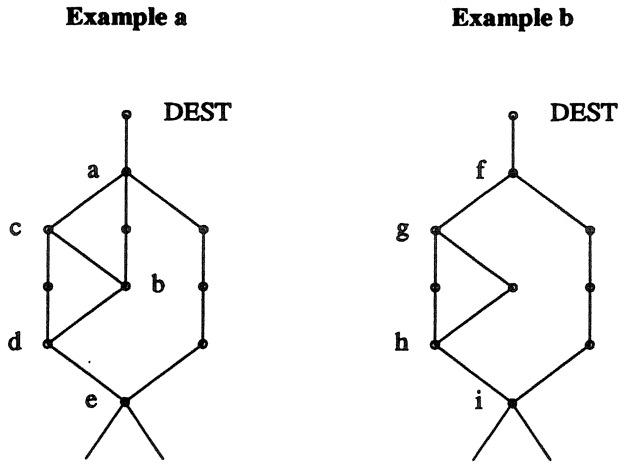
All JC messages received by a partial-juncture router must have one or more entries in the PJ Tree List for the following reason. If a juncture router is partial, then all messages it receives uptree must have passed through the same router (the one listed in the FJ Tree Label, either the last juncture router or the neighbor of the destination). If the router is a juncture router, however, then there must have been a fork in the tree somewhere between it and the router in the FJ Tree Label. The router at that fork will have put its ID in the PJ Tree List. Therefore, there must be at least one entry in the PJ Tree List of every JC message received by a partial-juncture router.

A partial-juncture router searches for the most recent (highest slot) matching entries in the PJ Tree Lists because the routers downtree from the router in this slot are the routers highest in the tree (closest to the destination) for which the partial-juncture router can be used to find alternate routing information. The reason for this is as follows. If all entries in a given slot of the PJ Tree Lists are the same, then all paths uptree from the partial-juncture router must pass through the router in that slot. If all entries for a higher slot of the PJ Tree Lists are also the same, then there must be one and only one path between the routers represented by the two matching sets of entries in the PJ Tree Lists. If there were multiple paths between the two routers, then the router lower down in the tree would be a partial-juncture router and would have removed the higher router from the PJ Tree List. However, a partial-juncture router may only provide a path to an alternate branch for those routers that are on two separate paths uptree from the partial-juncture router. Therefore, a partial-juncture router can be a juncture router for only those routers downtree from the highest matching entries in the PJ Tree List. If entries in a given slot of the PJ Tree Lists are different (these entries must be in

a higher slot from the matching entries), then the JC messages must have traveled different paths, and therefore the partial-juncture router can act as a juncture router for those routers.

A few examples will illustrate this. First, consider Figure 8b. Router *f* is a forking router (has more than one downtree neighbor with respect to destination DEST) and so fills in the first slot of the PJ Tree List with [(f,1)]. (The notation for the Tree List is *(router ID, hops from destination)* for each tuple and *[(tuple 1),(tuple 2),...]* for the entire list.) Router *g* receives this JC message and appends the PJ Tree List to read [(f,1),(g,2)]. Router *h* receives this message over both of its uptree links. Since the highest matching entry is (g,2), Router *h* knows that it can only provide the juncture function for those routers below Router *g*. (The hop count tells Router *h* how far uptree to send the APP message later on.) Router *h* deletes (g,2) from the message and sends [(f,1)] up to *i*. Router *i* receives [(f,1)] over both uptree links, stores this for later, and sends [(i,5)] downtree.

Figure 7
Partial-juncture Example



Now consider Figure 8a. Router *b* receives [(a,1),(c,2)] from its uptree link with *c*, and [(a,1])] from its other uptree link. Router *b* therefore knows that it is providing the juncture function to all routers uptree from it and downtree from Router *a*. Router *b* sends [(b,3)] to Router *d*. Router *d* receives [(a,1),(c,2)] over its other uptree link. Since there are no matches, it knows that it is providing the juncture function for all routers below *b* on its uptree link with *b* and providing the

juncture function for all routers below *a* on its other uptree link. Router *e* receives message [(d,4)] from *d* and [(a,1)] over its other uptree link, and makes the same decision as Router *i* of Figure 8b.

Once a router finds that it is a juncture router, it sends a message uptree called the Alternate Path Priming (APP) message. This message tells uptree routers 1) that they have an alternate path, 2) whether that path is through a full-juncture router or a partial-juncture router, and 3) how far the destination is via the alternate path. If the juncture router is a full-juncture router, the APP message simply travels uptree until it reaches another full-juncture router. If the juncture router is a partial-juncture router, the APP message has a field that states how far down the APP is to travel and which partial-juncture router it is from. The partial-juncture router knows what this distance is, based on the hop value received in the PJ Tree List of the JC message. For instance, the APP message sent by Router *h* in Figure 8b would state that it should travel to routers further than 2 hops from the destination. This message would therefore not travel to Router *g*. The APP message sent by Router *i* would state that it should travel to routers further than 1 hop from the destination.

3.3.1.1 Full Specification of Alternate-path Distance-vector Routing. Now that we understand the basic idea behind configuring juncture routers, we can fully specify ADR. We start this description assuming steady state—that is, all routing data bases are fully configured and no routing messages are active in the network. In steady state, each router has the following information:

1. The direction of all of its links (downtree, horizontal, or uptree) for each destination.
2. The metric value associated with each link.
3. Whether it is a full-juncture router, a partial-juncture router, or neither.
4. The contents of the last JC message received from each uptree and horizontal link.
5. The distance (in both hops and the link metric in use) to the destination via all of its uptree links.

If the router is not a full-juncture router, then it also knows:

6. The distance (for the link metric in use only) to the destination via all downtree links with full-juncture routers.

7a. The distance (for the link metric in use only) to the destination via all downtree links with partial-juncture routers.

7b. The distance (in hops only) from the destination that the APP message from each partial-juncture router travels.

If the router is a partial-juncture router, then it also knows:

8. The distance (in hops only) from the destination that its APP message travels for each uptree link.

With this information, any router knows not only its primary path and distance to every destination, but its alternate path and the distance via that path. Therefore, when a router receives a routing update about a certain destination, it knows immediately whether its alternate path will become its primary path. There is no convergence time for finding a new primary path! Instead, the convergence time is spent finding a new alternate path; but this does not interrupt traffic flow.

Any router can see one of two possible distance changes over any of its links—the distance can increase (bad news) or the distance can decrease (good news). In the ADR algorithm, we do not distinguish between distance changes discovered through a link metric change or via the receipt of a routing update. Note that metric changes over a link are always with respect to the distance to the destination *via that link* (either primary or alternate). The metric change can of course change the direction (from downtree to uptree, for instance) of a link, and therefore, may change whether it is a primary or alternate path. A router's best path will be its primary path and will always be uptree. A router's second best path (if it has one) will be its alternate path and may be a horizontal or downtree path.

Recall that a Router *b* is considered uptree from another Router *a* if *b* is on *a*'s best path to a destination. If *b* is uptree from *a*, then *a* must be downtree from *b*. Otherwise, any two Routers *a* and *b* can determine their relationship through one of several means, as long as both routers agree on the link direction. For instance, consider Router *a* with neighbors *b* and *c*, where *b* is *a*'s primary path, and *c* is closer (by some metric) to the destination than *a*. If *c*'s distance to the destination is closer to *b*'s distance than it is to *a*'s distance, *a* could consider *c* to be uptree. Otherwise, *a* could consider *c* to be horizontal. If a router has two or more equal best paths, then one is considered primary, and the rest are alternate.[9]

[9]Notice that the fact that a path is labeled alternate does not mean that traffic can't be sent over that path to the destination. Path splitting can be accomplished over multiple uptrees (see Section 3.3.1.2).

Routers receive JC messages from uptree and horizontal neighbors and send JC messages to downtree and horizontal neighbors. Routers send APP messages uptree and receive APP messages from downtree. A router may not send a JC downtree or an APP uptree until it has received JCs from all of its uptree and horizontal neighbors. If the spanning tree direction of one or more links changes, a router will forget anything it heard over that link and either wait for a message or send messages, depending on the situation. For instance, if a link that was formally uptree or horizontal becomes downtree (we will discuss how this happens shortly), the router must forget any JC that it previously heard over that link and send a JC down that link based on JCs previously received over the uptree and horizontal links it still has. If a link that was previously downtree or horizontal becomes uptree, it must wait for a JC over that link. Once it gets one, it sends out new JCs and APPs based on the newly received JC and those previously received from unchanged uptree and horizontal links. In other words, messages previously received over links that have not changed direction remain valid and are used in later calculations.

Now we consider specifically how link direction changes occur, and generally how metric changes are handled. Consider the following:

1. Any router downtree from some router *a is* depending on *a*'s primary path to derive its primary path.
2. Any router horizontal from *a may be* depending on *a*'s primary path to derive its alternate path.
3. Finally, any router uptree from *a may be* depending on *a*'s alternate path to derive its alternate path.

Next, we consider how to respond to several types of metric changes:

1. The distance over *a*'s primary path to some destination gets smaller (by smaller, we mean small enough to invoke an update).

   1a. Assume first that *a* has only one uptree link. Then this change affects downtree routers and affects horizontal routers if they are using *a* as their alternate path. Since no link directions changed, *a* sends routing updates to its downtree and horizontal neighbors indicating the new, shorter distance to the destination. The downtree routing updates must indicate how many hops *a* is from the destination (the reason why is explained in 1c below).

1b. Now assume that *a* had other uptree links when it received the good news and for which either 1) *a* is a full-juncture router or 2) *a* is a partial-juncture router, but the good news is from a router that *a* is providing an alternate route (in other words, *a* knows that the good news will be received over only one of its uptree links). Then those uptree links may become horizontal links. (They may also become downtree links, but we discuss that in 1d.) Router *a* must send three pieces of information to its new horizontal neighbors: 1) *a*'s new distance from the destination, 2) *a*'s new relationship with those neighbors (horizontal instead of downtree), and 3) a JC message based on old JCs from *a*'s uptree link. These three pieces of information may all be contained in a single message. As soon as *a* receives JC messages from the new horizontal links, *a* must send new JC messages downtree, and a new APP message uptree. These downtree JC messages may also contain the good news, so that two downtree messages (a routing update followed by a JC) are not required.

1c. Now assume that *a* had other uptree links when it received the good news, and that *a* is a partial-juncture router, but the good news is from a router *above* those which *a* is providing an alternate route for (in other words, *a* knows that the good news will be received over more than one of its uptree links). In this case, *a* may set a timer and wait to hear news from the other uptree links before acting.

1d. Finally, assume that the good news caused one or more of *a*'s horizontal or uptree links to become downtree links. Router *a* must trash the JC messages received from those links, calculate a new JC message, and send that message downtree, along with the distance change. Note that *a* may hear APP messages from downtree in response to any JC messages it sent downtree.

2. The distance over *a*'s primary path to some destination gets larger.

2a. Assume first that *a* has only one primary path, and that this change does not cause any links to change directions. Then the action is the same as 1a above: *a* will send routing updates to its downtree and horizontal neighbors indicating the new, longer distance to the destination.

2b. Now assume that this distance increase causes another link (one that was previously either uptree, horizontal, or downtree) to become the primary, uptree link. Now, the router must take action similar to those in 1b-1d above, except that

any advertised distances will now be based on the distance to the destination over the new primary link.

3. The distance over *a*'s alternate path to some destination gets smaller.

   3a. If the change causes no changes in link direction, then all that is necessary is to send a new APP uptree on the primary paths to indicate that the alternate path is closer than before.

   3b. If the change causes the alternate path to become the primary path, then the router must take action similar to those in 2b above.

4. The distance over *a*'s alternate path to some destination gets longer. In any event, it is necessary is to send a new APP uptree on the primary paths to indicate that the alternate path is further than before. If the alternate path changed from horizontal to downtree, it will be necessary to send new JC messages on downtree and horizontal links.

5. The distance over a path which is neither the primary or the alternate changes. If the change resulted in no link direction changes, then no messages are necessary. If the change resulted in a new alternate (or primary) path, then changes similar to the ones described above are required.

Notice that ADR will always prevent CT∞ because any metric changes are followed by the appropriate JC and APP messages, thus giving routers information about where to find valid alternate routes. There will be periods of time during which a router may have no alternate route, but for any single failure, and assuming there is another path, a router will always have a primary route. For instance, assume that the link over a router's primary path fails. The alternate path (if there is one) will immediately become the primary path, and the router will have no alternate path. However, after the JC message goes out and the APP is returned, the router will have an alternate that it may use in case of another failure.

For double failures, it may be possible for a router to have no path to a destination. Over time, however, the routing updates and JC and APP messages will converge to correct routing. Still, there is no CT∞ in this situation.

3.3.1.2 Routing Packets Over Alternate Routes. Once routers have labeled all links as uptree, horizontal, or downtree with respect to a given destination, it is possible to use alternate routing for the purposes of 1) splitting traffic and 2) responding immediately to link failures. We have derived a simple set of routing rules for using alternate routes which does not involve either 1) labeling the packets as alternate route packets or 2) source routing the packets. The rules are as follows:

1. Any packet originated at a router (in other words the router is the first hop from a host) may be sent over a horizontal or uptree link and over a downtree link if there is a juncture router. (Obviously, if it is sent downtree, it will take a non-optimal path.)

2. Any packet received *from an uptree link* must be routed via 1) another uptree link if there is one, 2) a horizontal link if the router is a full-juncture router or is a partial-juncture router but knows of no higher juncture routers, or 3) on the downtree link to the nearest full-juncture router if there is one, or the furthest (from the destination) partial-juncture router if there is not.

3. Any packet received from a downtree link may be sent over a horizontal link or an uptree link.

4. Any packet received from a horizontal link must be sent over an uptree link.

With these rules, a packet may only go downtree by starting that way. A packet cannot be going uptree and then be changed to go downtree (unless a path change has caused an inconsistency in the definition of "downtree"). This prevents downtree-uptree loops. If a packet crosses a horizontal link, it must then go uptree. This prevents a loop of successive horizontal links.

These rules provide two functions. First, they allow a router to immediately reroute packets upon seeing a distance increase over its uptree link. For instance, when a non-juncture router a sees a distance increase over its uptree link, it will immediately choose a different neighbor b as its uptree link (if an alternate path is available via b). There will be a period of time before b receives an update from a during which b will consider a its uptree neighbor and a will consider b its uptree neighbor. As the update travels downtree from a, but before it reaches a juncture router, there will be successive pairs of routers that will consider each other their uptree neighbors. If we allow routers to receive packets from uptree links and pass them downtree to juncture routers, then a can pass a message to b, and b will not pass it right back to a. However, b must pass it to a full-

juncture router, because if a partial-juncture router passed it back down, it might reach the same router that routed it downtree in the first place.

The other function that these rules provide is path splitting. By sending packets over multiple links, a router can reduce the impact that a single traffic surge can have on network resources, thus avoiding congestion. The amount of path splitting is not fully predictable. For instance, there is some likelihood that traffic split at some router will join again at another. Consider Figure 7a. Here, some of the traffic split over *J2*'s uptree paths may join again at *b3*.

A possible good use for traffic splitting is avoiding local congestion when it occurs. It is not clear that traffic sensitive routing metrics are of much use for shunting short term traffic fluctuations. The problem of oscillation, for instance, requires that routing updates be filtered over many seconds at best (McQuillan, Richer, Rosen, 1978) (McQuillan, Richer, Rosen, 1980). If a queue suddenly becomes congested, several things must happen for routing updates themselves to successfully handle the congestion. First, the congestion must last long enough to give the routing updates time to propagate around the network and change routes. Second, the routes must change enough for the router(s) causing the congestion to be rerouted, but not so much that the rerouted traffic doesn't cause congestion in another place.

Normally, dropping packets is the best method of handling severe congestion. (One hopes there is a good congestion avoidance mechanism to avoid severe congestion in the first place. Unfortunately, this is not always the case.) However, if a router sees severe congestion on is primary path, but little congestion on a horizontal or downtree path, and provided that the downtree path doesn't take the packet too far out of its way, temporary immediate rerouting of traffic may be effective. This is a good topic for further study.

3.3.1.3 Overhead from Alternate-path Distance-vector Routing. In this section we briefly discuss the overhead due to ADR. With regards to memory, ADR requires a certain amount of information about each of its connected links for every destination. Router degrees of three or four are typical, so we are looking at memory consumption of on the order of three or four times the number of destinations (Landmarks) for this information. In addition to this information, a router may need to keep track of multiple partial-juncture routers. However, only configurations such as those in Figure 8b, where partial-junctures are nested, will cause multiple partial-juncture routers. We believe that this will happen only rarely and will not significantly impact memory usage.

With regards to link usage, in the worst case we will see two messages over the links of routers affected by a metric change. This will usually be all of the routers downtree from the change and some uptree. This is similar to the Jaffe-Moss and Garcia-Luna techniques. Remember, however, that these messages are traveling in search of an alternate path not a primary path as in Jaffe-Moss and Garcia-Luna. The JC messages will of course be longer than normal routing updates because they will contain the FJ Tree Label and the PJ Tree List. The PJ Tree List, however, will usually not be long because there will be a limited number of downtree forks in the spanning tree (much fewer than the diameter of the network). In *no* event will the PJ Tree List ever have more entries than D, the diameter of the network. In addition, many PJ Tree List entries will be removed as the JC travels downtree, because partial-juncture and full-juncture routers will remove PJ Tree List entries. Therefore, we do not believe the PJ Tree List will have a significant impact on link usage.

3.4 Routing Update Policies

There is considerable engineering leeway for implementing ADR (or any distance-vector routing algorithm). One can make routing as dynamic or as static as desired, depending on the routing update policies used.

There are two broad classifications of routing update policies. The first separates routing update policies into traffic-based or static. This determines a routing update's unit of measure, in other words, the routing metric.[10]

The second classification separates routing update policies into event-driven and timer-driven routing update policies. This determines when a routing update is sent. The first policy, the routing metric, has a strong impact on the second policy, the timing of the routing update. In other words, the decision of when to send a routing update is affected by how often the value of a metric is expected to change.

In the following sections, we briefly discuss the characteristics of the two classifications individually, followed by a more detailed discussion of both.

---

[10]Note that the metric used to maintain the Landmark Hierarchy—essentially to determine how far Landmarks are from each other—is hops. The reason for this is explained in Section 4.2.4. The routing metric we describe below is completely independent of the hops metric used to maintain the Landmark Hierarchy.

3.4.1 Routing Metrics: Traffic-Based vs. Static

A traffic-based routing metric is one whose value can change because of a change in traffic volume. A good example of this is a metric based on delay. More traffic causes longer queues, and therefore longer delay. A static routing metric, then, is one whose value is not effected by traffic volume. A good example of this is a metric which is based on the bandwidth of a link, such as is used in Burroughs Integrated Adaptive Routing System (Gruchevsky and Piscitello, 1987).

Obviously, a so-called static metric is not permanently static. The deletion or addition of a link will change the metric. However, once a link is established and before it is removed, the value of the metric will not change.

In addition, the dynamics of a traffic-based metric may vary greatly depending on the desired responsiveness and/or stability desired. It is appropriate to think of the possible types of routing metrics as being a continuous range from very dynamic to static. This will be discussed further.

Finally, hybrid metrics—metrics based on more than one measured characteristic—are possible. In particular, a metric based on both traffic volume and a static link value is useful, and is used in the ARPANET (McQuillan, Richer, Rosen, 1980). In this case, the composite metric can still be considered traffic-based, but the changes in metric value are biased by the link value, which has the effect of slowing the rapidity of metric value change, thus avoiding oscillations.

3.4.2 Event-driven vs. Timer-driven Routing Update Policies

In event-driven routing, updates are sent when a certain event occurs. Typical events are the changing of a local metric value, or the reception of a routing update from a neighbor. In timer driven routing, updates are sent periodically, whether there is something new to report or not. The advantages to timer-driven routing are simplicity and predictability. The simplicity comes from there being only one condition, an easily implementable timer, which triggers an update. The predictability comes from the number of updates in the network being consistent over time. The disadvantage of timer-driven routing is that there is a lag between an event occurring and it being reported in an update. Finally, there are certain things which can only be implemented in a timer-based fashion, like checking the aliveness of a link or a neighbor.

The advantage of event-driven routing is that it responds immediately to network conditions. The disadvantage is in its complexity and lack of predictability. Predictability is a problem for two reasons. First, if many events happen at once, a large number of updates can be generated causing

unexpected congestion in the network. Second, and perhaps more serious, is the fact that one event can trigger another, which can trigger more. If this sort of exploding chain of events is not carefully controlled, routing can self-destruct.

Again, a hybrid between these two is common. In the ARPANET, routing updates are largely event-driven, but with a lower bound on how many events can occur (thus improving stability at the expense of timeliness) and with an upper bound on how much time can pass before an update is sent.

3.4.3 Choice of Routing Update Policies

It is impossible to say much about the choice of routing update policies without knowing the user requirements and networking environment. We believe that an event-driven scheme, with a forced minimum time between events is a good policy in general. The choice of routing metric, however, is a whole other matter.

Extensive research has been done on the topic of routing metrics. One of the problems here is that sometimes an optimal end-to-end characteristic is not optimal for utilization of network resources. For instance, if delay is to be minimized, then often two or three (or more) network hops can be shorter than one, thus creating more network traffic. Furthermore, delay characteristics for a given link are different depending on the type and number of packets sent. For instance, a high-bandwidth satellite link exhibits large delay for small single packets, but very small overall delay for a large number of large packets.

Even further complicating delay-based metrics is the measurement of the delay itself. For one thing, the measurements should be filtered over a period of time in order to 1) dampen out transitory fluctuations and 2) prevent oscillation. Another method of preventing oscillation is to lessen the effect of delay on a link by giving more weight to transmission delay or propagation delay, as is done in the ARPANET.

The environment we envision for our initial implementation of Landmark Routing will be the DARPA Internet. This environment is extremely heterogeneous with regards to switch capacity, link capacity, traffic levels, and so on. Any sophisticated choice for a metric, especially a traffic-based metric, will be far less than optimal in one environment or another. We feel that the best we can do is to assign a variable static metric to each link. This is the approach used in two commercially available routing protocols recently proposed to the ANSI X3S3.3 standards committee, one from Digital Equipment Corporation (X3S3.3/87-150, 1987) and one from

UNISYS (X3S3.3/87-160R1, 1987). The UNISYS proposal provides a rule-of-thumb equation for calculating the link metric based on an approximation of the length of time it takes a maximum sized packet to cross the link. This metric will provide a reasonable traffic distribution over the network without requiring a centralized calculation of metric values. The metric can be artificially raised or lowered to either encourage or discourage traffic.

The idea behind the use of this sort of metric is that, as network experience is gained or as traffic distributions change, metrics can be adjusted to provide an acceptable traffic distribution. Of course, one can only go so far with this sort of thing before link capacities must be changed to accommodate traffic. This type of static strategy also assumes that some other mechanism for choking traffic is available. We have no intention of trying to solve the congestion control problem through the use of routing mechanisms.

3.5 Other Routing Functions

There has been considerable interest in enhanced routing protocols which provide multiservice routing and multipath routing (Perlman, 1981), (Gardner, 1985). Multiservice routing finds the best path for a given class-of-service, such as low delay or high bandwidth. Multi-path routing finds several simultaneous paths for the same source-destination pair, for the purposes of increasing end-to-end bandwidth and for balancing network load. Both of these functions are difficult to achieve in practice.

3.5.1 Multiservice Routing

In the case of Multiservice routing, there is the problem of service class explosion. For instance, assume we have two routing metrics, delays, and bandwidths. There are in fact at least four service classes (i.e., combinations of routing metrics) which can be derived from these two metrics: 1) low delay and bandwidth, 2) low delay and high bandwidth, 3) high delay and low bandwidth, and 4) high delay and high bandwidth. This can be made much worse if, as part of the service class, allowable ranges of delay and bandwidth are specified. When other metrics, such as dollar cost, security, and reliability are introduced, the potential number of service classes explodes as 2^m, where m is the number of metrics (Perlman, 1981).

This problem can be dealt with in a limited fashion using a link-state routing scheme. However, the problem in its general form is untenable using distance-vector routing. The reason for this is that, in link-state routing, the paths are calculated locally in each router. Therefore, given the m metric values for each link, every router can calculate all 2^m service classes locally. Since

only a few of the 2^m service classes are likely to be in use at any time, the number of calculations needed may be reasonable. In distance-vector, however, the paths are calculated in a distributed fashion. To calculate all service classes, at worst 2^m routing algorithms must be running at once, or at least a number equal to the number of needed service classes must be running at once (network-wide, not just locally).

In spite of all this, we believe that one can do reasonably well given a limited number of service classes. This is because the provision of services assumes that there are mechanisms in place which are able to provide those services. If one wants high bandwidth, one needs high bandwidth links. If the high bandwidth link has high delay, and there are no high bandwidth links with low delay, then there is no reason to have a routing service class that looks for both high bandwidth and low delay. If one wants to send real-time packetized voice, but there exists no subnetwork which can provide packetized voice, there is no reason to have a routing service class which is appropriate for packetized voice (i.e., a certain bandwidth, a fairly small delay but with little variance in delay over time, and possibly a fairly high error rate, depending on the encoding scheme).

The point is, one builds (or subscribes to) certain network infrastructures to provide specific end-user services. One therefore needs only as many routing services as one has different network infrastructures. In other words, the routing provides the user with the power of chosing a path over every type of network the user has made available to himself. Even a relatively sophisticated user is not liable to provide himself with more than a handful of significantly different services. Landmark Routing provides for these services by allowing several Routing Algorithms to run simultaneously, each optimizing on a different service profile.

3.5.2 Multipath Routing

Multipath routing can generally be described as a technique for making better use of a given set of network resources. It does so by dividing traffic among multiple paths in order to both more evenly load the network and provide more bandwidth to the user. Again, multipath routing is easier with link-state routing, because the whole path can be precalculated using the topology map, and then specified using source or partial source routing.

In Landmark Routing, we already have the technique of alternate paths for accomplishing some path splitting. In addition, we propose another technique which may be more probabilistic than the link-state method, but which we hope will be effective. The scheme is to allow routers to

obtain more than one Landmark Address. This can be done easily by insuring that every level i Landmark cover 2 level i+1 Landmarks (Tsuchiya, 1987). By doing this, each router will have 2^G possible different addresses, G being the number of hierarchy levels up to the global level. To see why, consider a local Landmark which has two parents, a_0 and b_0. Assume that each of these parents have the same two parents, a_1 and b_1; that these two also have the same parents, and so on. It is easy to see that 2^G different addresses can be constructed, although most of them may be quite similar. Consider again a local Landmark which has two parents, a_0 and b_0. Now consider that these two parents have completely different parents a_1, b_1, c_1, and d_1; and that all four of these have eight different parents, and so on. Again we easily see that there are 2^G different addresses, except that all of these are completely different. This latter case will never happen in an actual Landmark Hierarchy. The reader can prove to himself that all possible combinations of two parents will produce the same number of valid addresses.

If every Landmark at every level advertises two completely different sets of ancestors, one can guarantee that every router will end up with 2 completely different Landmark Addresses. If the source ES sends half of its packets using one address, and half using the other, some amount of path-splitting will occur. Additional path-splitting can be accomplished if each IS also split its traffic over paths with similar distances to any given destination. Future simulation will determine how much benefit come from these techniques.

3.5.2.1 Reliability Considerations Resulting From Multiple Addresses. In addition to the multipath benefits we hope to gain from two addresses, we also achieve additional reliability. First, if a Landmark becomes permanently or temporarily unreachable, it may be possible to use the other to deliver packets. Also, when a router gets a new address, the other address may still be good allowing continued communication while the new binding is taking place. Having two Landmark Addresses, however, roughly doubles the amount of binding that must be done.

4 DYNAMICALLY MANAGING THE LANDMARK HIERARCHY

This Section explores the issues associated with dynamically managing the Landmark Hierarchy. We define dynamic management of the Landmark Hierarchy as the process of automatically choosing Landmarks, Landmark Addresses, and radii in the face of changing topology. The main benefit of using the Landmark Hierarchy comes from the fact that it can be dynamically managed more easily than the area hierarchy. Without this benefit, the Landmark Hierarchy can hardly be considered superior to the area hierarchy (Tsuchiya, 1987).

First, we review the existing research on dynamic management of the area hierarchy. We then review the definition of the Landmark Hierarchy, define terminology, and summarize some useful results obtained from the study of the Landmark Hierarchy in a static state (Tsuchiya, 1987). Next, we establish our design goals for Landmark Hierarchy management. We then discuss the major design considerations associated with dynamic management of the Landmark Hierarchy. Finally, we present the design choices.

As with the rest of this document, we only present qualitative descriptions of hierarchy management. Later work will deal with quantitative issues such as the exact algorithm, performance, and parameter values.

4.1 Review of Dynamic Management of Area Hierarchy

Dynamic management of the area hierarchy can be divided into two categories: patching of area partitions and full reconfiguration of the area hierarchy. In the first category, which we will call partition patching, membership of routers to areas, areas to superareas, and so on, never change (that is, addresses do not change). The simplest way to patch a partition is to bridge the two partitioned segments by creating a higher-level logical link, such as a transport connection or partial source route, between the segments. This is inefficient during the lifetime of the partition. It is used in networks which have a relatively stable topology where partitions will be a rare event. It is not suitable in a network with continuously changing topology, such as a mobile packet-radio network. Perlman has studied the problem of partition patching to a satisfactory degree (Perlman, 1985), and partition patching is implemented in more than one commercial network.

The second category, which we will call area hierarchy management, has not been satisfactorily studied, possibly because of the difficulty of the problem. We could only find two papers that discuss area hierarchy management (Hagouel, 1983) (Shacham, 1985). Both of these

papers give an overview of area hierarchy management functions, but do not provide detailed protocols or significant analysis.

A significant portion of the Hagouel thesis is devoted to *centralized* algorithms which generate efficient area hierarchies. Even when run in centralized fashion, these algorithms are not trivial, and the quality of the algorithm can have a significant effect on the performance of the generated area hierarchy. A distributed version of the algorithm is more difficult. Hagouel only presented a brief appendix on the subject of distributed, dynamic management of the area hierarchy, this not being the focus of his research.[11]

Like Hagouel, only a small part of Shacham's paper is actually devoted to the problem of area hierarchy management. Whereas Hagouel treated the problem in a general fashion, Shacham was more interested in a packet-radio network. Hagouel's and Shacham's schemes have several differences, but the steps required for both can be summarized as follows:

1. **Choose a primary router.** In this step, a router is chosen from a group of routers to make clustering decisions. (In this document, we freely interchange the terms cluster and area.) In some cases, an election algorithm is required for this step. It was not clear that it could always unequivocally be determined which group of routers were running the election. For instance, there were situations where, after a set of routers were partitioned from their regular cluster, some of them could be running an election algorithm while others might be attempting to join other clusters. The Hagouel scheme used freeze states to address this (and other) problems.

2. **Form a cluster.** Once a primary router was chosen, it would then be in charge of forming a cluster. In the Shacham scheme, it was up to individual routers to attempt to join a cluster by requesting membership from the primary router. In the Hagouel scheme, the primary router actively searched out routers in a greedy fashion for cluster membership. In both cases, it was not clear that a router would always successfully join a cluster. In the Shacham scheme, a primary router could reject a router's request to join a cluster. The situation where a router could not get accepted into any cluster was not addressed. In the Hagouel scheme, both the situations where 1) a router was not

[11] When we speak of the "performance" of a hierarchy, we are speaking of the routing table sizes and the path lengths.

asked to join any cluster and 2) a router was asked to join multiple clusters were possible. In the second case, a mechanism for resolving contention was required.

3. **Repeat steps 1 and 2** for the next layer of the hierarchy.

We believe that the major problems with area hierarchy management are as follows:

1. **Potentially inefficient hierarchy structure.** The choice of clusters can affect the optimality of the hierarchy (see Figure 4.21 of the Hagouel paper).

2. **Amount of coordinated decision making.** There are instances in the maintenance of a area hierarchy where a group of routers must synchronize to make a decision, such as whether to form a new area or to break up and join other areas. Such synchronization is complex.

3. **Contention problems.** The same routers can be required by different clusters, requiring contention resolution procedures.

4. **Completeness problems.** Routers can be left out in the cold, so to speak, because they have no cluster membership.

As dynamic management of the Landmark Hierarchy is discussed, we will come back to these four points and compare. While maintenance of the Landmark Hierarchy is not necessarily completely free of these problems, it exhibits them to a far lesser extent.

4.2 Additional Aspects of the Landmark Hierarchy

In Section 2, we gave an overview of the Landmark Hierarchy. In this section, we present more parameters, present results from the previous work, and discuss the Landmark Hierarchy partition and the various approaches to dealing with it.

4.2.1 Landmark Hierarchy Parameters

In Section 2, we describe the parameter r_i (radius) as the average distance in hops within which routers have a routing entry for a Landmark LM_i at hierarchy level i. In other words, r_i describes the extent of the *vicinity* of a Landmark. When written $r_i[id]$, it refers to the radius of a particular Landmark $LM_i[id]$

The subscript i always refers to the hierarchical level. The subscript H refers to the highest level. The Landmark at this level is called the root Landmark. The lowest level is at $i = 0$, and

higher levels are denoted by increasing values of i. In other words, if a Landmark is at level i, then its child will be at level $i-1$.

The superscript G implies that a Landmark is global. In other words, its radius extends to all routers in the network. For example, LM_f^G is a global Landmark whose radius would be written r_f^G.[12]

The notation d generally refers to the distance between two routers. When written $d_i(id)$, it describes the distance between a router id and its closest level i Landmark LM_i. When written $d(x \rightarrow y)$, it refers specifically to the distance from router x to router y. Finally, when written as d_i, it refers to the average distance between every router and its closest LM_i

The parameter T_i describes the number of level i Landmarks in a network.

The function $v(x)$ describes the number of routers within x hops, $0 \leq x \leq D$, of some given router. This function is dependent on the number of routers in a network N, the average router degree C, and the diameter of the network D. This function starts at $v(0) = 1$, geometrically increases as x increases, then tapers off to a linear function of x as x increases further, and finally tapers off further and reaches N as x reaches D (Tsuchiya, 1987).

The parameter R_i describes the average number of routing table entries in a router for level i Landmarks. In other words, it describes the number of LM_i within r_i hops of a given router.

Finally, the parameter $\hat{P}_i$ refers to the average increase in path length for the Landmark Hierarchy over shortest path (the ratio of Landmark Hierarchy path lengths over shortest path lengths). A value of $\hat{P}_i = 1$ implies that the paths found in the Landmark Hierarchy are the shortest possible. The parameter $\hat{P}$ (without the subscript) refers to the average increase in path length overall.

4.2.2 Results of Previous Work

In the previous study of the Landmark Hierarchy (Tsuchiya, 1987), the routing table sizes R_i and path lengths $\hat{P}_i$ were studied in great detail. Those results are central to the reason why the Landmark Hierarchy is easier to manage than the area hierarchy. In particular, we find that the performance of the Landmark Hierarchy is not very sensitive to the number of and choice of

[12]This notation is required in addition to the subscript $i = H$ because, while there is one root Landmark that is always global, there are other Landmarks at levels $i < H$ that are also global (Section 4.3.2.1).

Landmarks. As a result, any election algorithm used in Landmark Routing has loose constraints and is therefore simpler. We repeat selected results of that study here.

There is a direct relationship between r_i, the average LM_i radius, and d_i, the average distance from all routers to their closest LM_i. Certainly, if LM_i have large r_i, higher level Landmarks can be far away from them, and hence d_i can be large. Conversely, if r_i is small, higher level Landmarks must be closer and d_i will be small. (It is of course possible to have large r_i and small d_i. It is not possible, however, to have small r_i and large d_i.

There is a direct relationship between the size of the routing tables R_i and the number of Landmarks T_i and the average Landmark radius r_i. Clearly, if there are either many LM_i in a network (large T_i) or if their r_i are large, then routers will have more entries in their routing tables.

On the average, every LM_i will have $v(r_i)$ routers within its radius, and each of those routers will have a routing entry for that LM_i. There will therefore be $T_i v(r_i)$ routing entries in the network, and on the average

$$R_i = \frac{T_i v(r_i)}{N} \qquad 1$$

routing entries in each router for each level i.

There is an inverse relationship between T_i, the number of LM_i in a network, and d_i. Consider that for each LM_i there are, on the average, $\frac{N}{T_i}$ routers closer to that LM_i than any other LM_i. Then there will exist some $\hat{d}_i$ for which

$$v(\hat{d}_i) = \frac{N}{T_i}. \qquad 2$$

Clearly, $\hat{d}_i$ and d_i are related. If the average distance to an LM_i (d_i) is small, then the average maximum distance from a router to an LM_i ($\hat{d}_i$) must also be small. In particular,

$$\hat{d}_i > d_i. \qquad 3$$

If we combine Equations 1 and 2, we get

$$R_i = \frac{v(r_i)}{v(\hat{d}_i)}. \qquad 4$$

Because of Equation 3, we see that if we calculate an R_i based on d_i rather than $\hat{d}_i$, the result will be larger that the actual expected R_i. Therefore,

$$R_i \leq \frac{v(r_i)}{v(\hat{d}_i)} . \qquad 5$$

Equation 4 indicates that routing table sizes are more dependent on the ratio of the radius of the Landmarks (r) and the density of Landmarks in the network (d) than on specific values of either r or d.

This can be seen in the graphs of Figure 9. This shows the impact of the ratio $\frac{r}{d}$ on the average routing table size R and path length increase $\hat{P}$ for 36 different networks. (The networks ranged from 200 to 800 routers, router degrees ranged from 2.4 to 6 links per router, and diameters from 7 to 50 hops. Here d is the average distance from all routers to their closest Landmark.)

Figure 9a shows that the number of routing table entries increases as $r-d$ increases (from 2.6 to 6.4), as Equation 4 predicts. Figure 9b shows that the path lengths get shorter as $\frac{r}{d}$ increases. Figure 9c, however, shows that specific values of d varied significantly over the range of $\frac{r}{d}$. Therefore, as we would expect from Equation 4, the specific value of d (or r) doesn't determine routing table size (and path length) so much as the ratio $\frac{r}{d}$.

This is important for the dynamic management of the Landmark Hierarchy. The thing that affects d is the number of and placement of Landmarks in the Landmark Hierarchy. If d can vary without much impact on Hierarchy performance, it means that the election of Landmarks, which is the most complex part of Hierarchy management, can be very loose. What happens is Landmarks are elected in a fairly loose election algorithm, and then the radii are adjusted afterwards to "fine tune" the ratio $\frac{r}{d}$, thus making the hierarchy as efficient as possible. This is discussed further in Section 4.3.

Figures 10, 11, and 12 show the overall performance of the Landmark Hierarchy. Figures 10 and 11 show results for simulations run on networks ranging from 50 to 800 routers, and with small and large diameters. They show results for three different sets of hierarchy parameters, one which results in large routing tables and small path lengths (Simulation A), one which results in small

[13]Simulation A corresponds to experiment :1c,h1,o1, Simulation B corresponds to experiment :2,t2,c2, and Simulation C corresponds to experiment :1a,h1 (Tsuchiya, 1987).

Figure 8
Effect of r/d on Landmark Hierarchy Performance

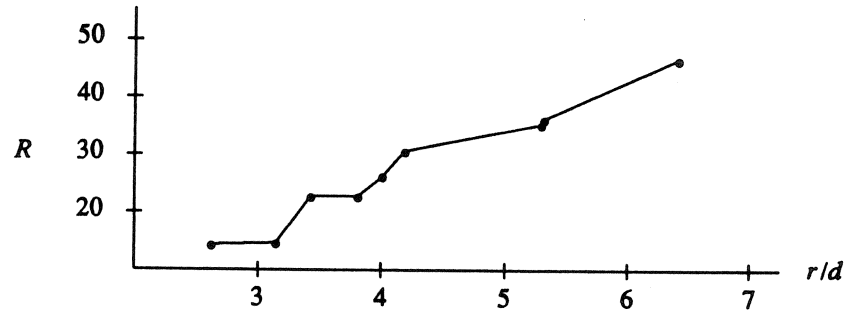


Figure a

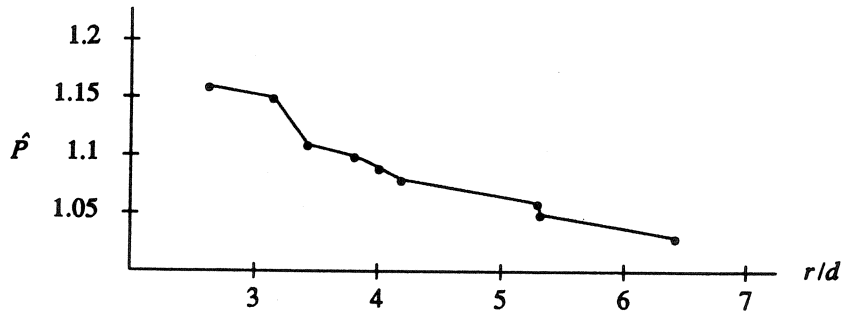


Figure b

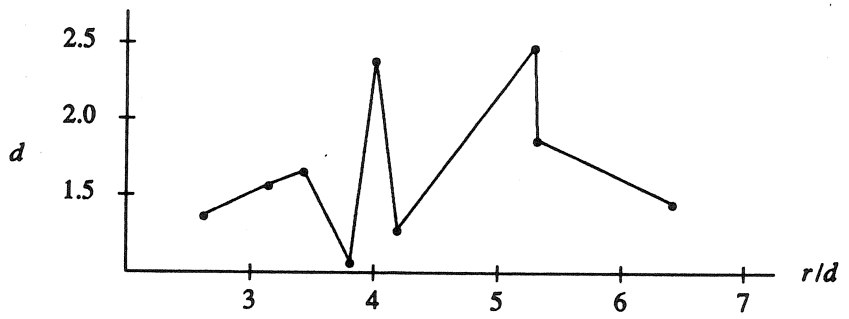
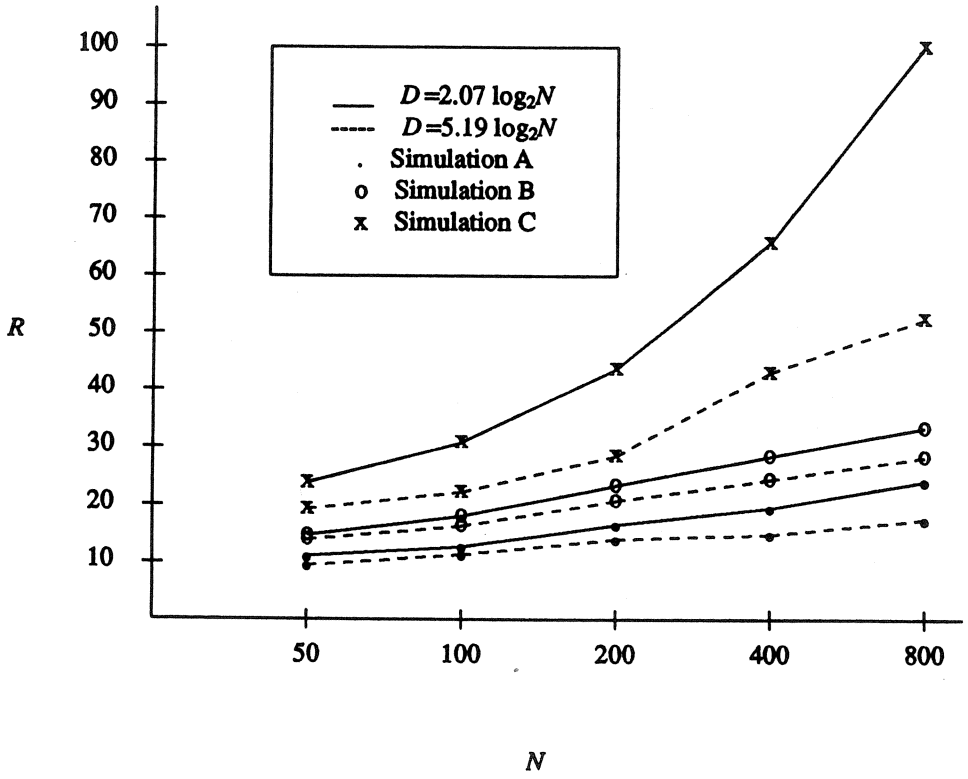


Figure c

routing tables and large path lengths (Simulation C), and one which falls in between (Simulation B). Figure 10 shows the routing table sizes R, and Figure 11 shows the path length increase $\hat{P}$.[13]

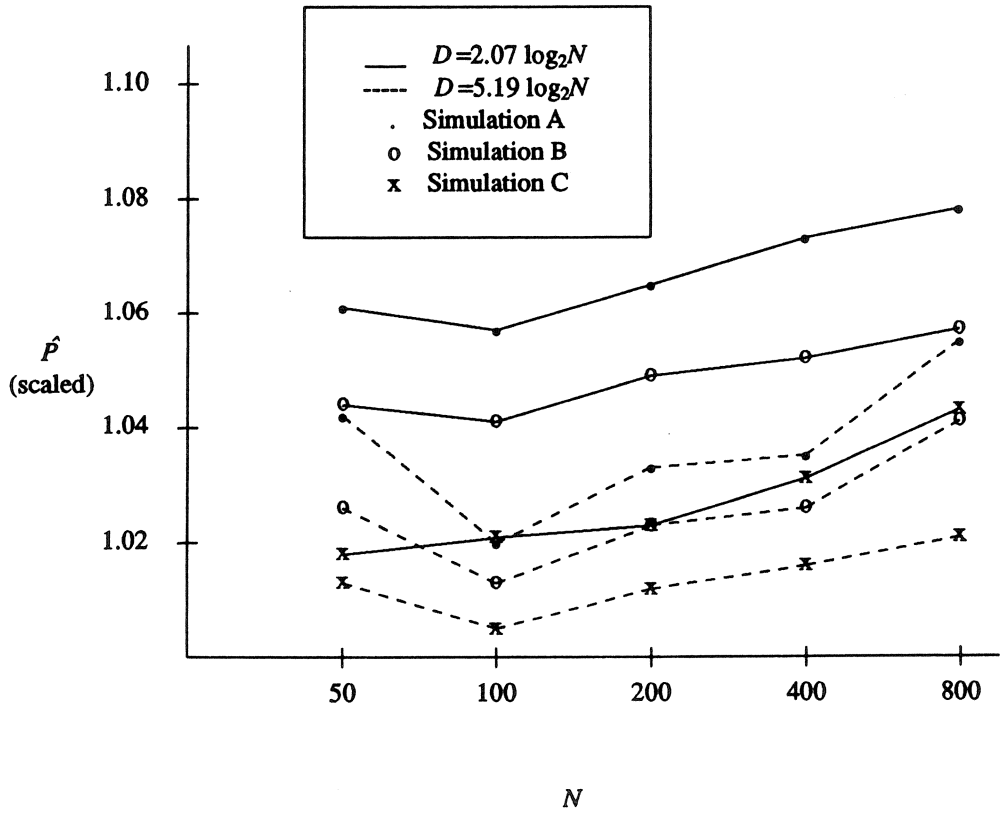
Figure 9
Routing Table Size for Realistic Networks



From Figure 10 and 11, we see first that both path lengths and routing table sizes are larger for networks with smaller diameters. Second, we see that larger routing table sizes results in smaller path length increases, and vice versa.

Figure 12 shows estimated routing table sizes for networks ranging from 100 to 1,600,000 routers. We see here that different values of R were achieved by adjusting the value of $\frac{r_i}{d_i}$. Note again that different diameters have an impact on routing table sizes. Here we see the impact of $v(x)$ in Equation 4. The different diameters change the function $v(x)$ (a small diameter makes $v(x)$ steeper), thus changing R.

Figure 10
Path Lengths for Realistic Networks and Scaled Traffic Matrix

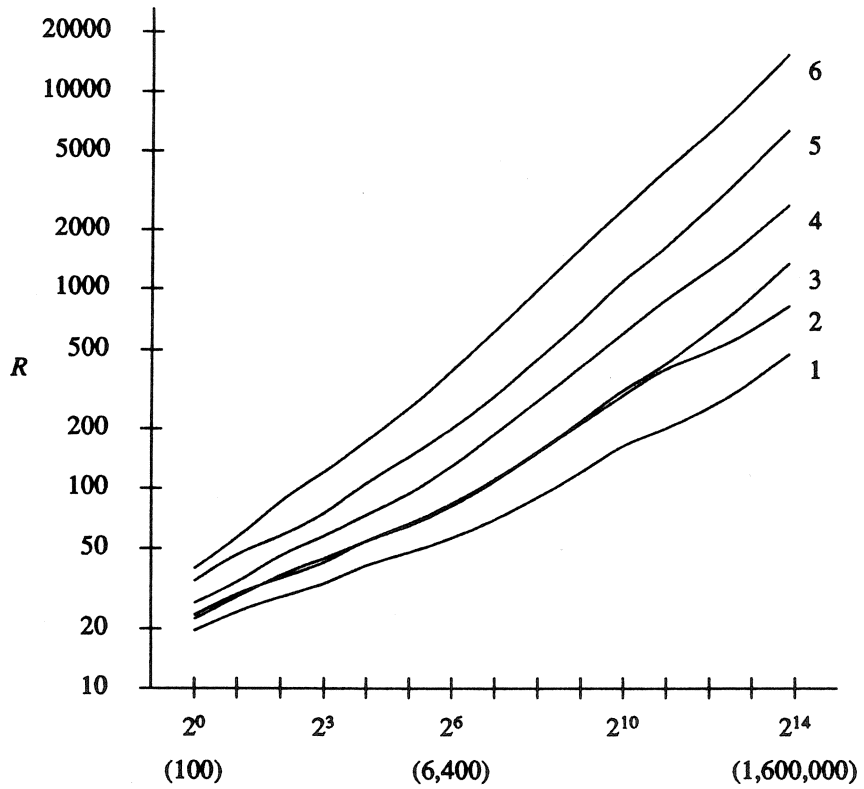


4.2.3 The Landmark Hierarchy Partition

For the Landmark Hierarchy to work, a parent Landmark must be within the radius of its children. It follows, then, that when a parent is not within the radius of one or more of its children, the hierarchy is broken—routing does not take place for some set on network routers. To see this, consider Figure 13—the same picture as Figure 2, but with the first level Landmark $LM_1[b]$ moved outside the radius of its child $LM_0[a]$. When the message from source destined for $LM_0[a]$ is routed to $LM_1[b]$, it goes no further because $LM_1[b]$ does not know how to route it to $LM_0[a]$. Often,

[14]If the source is on the "correct" side of the child with respect to the parent, then communications will succeed in spite of the partition.

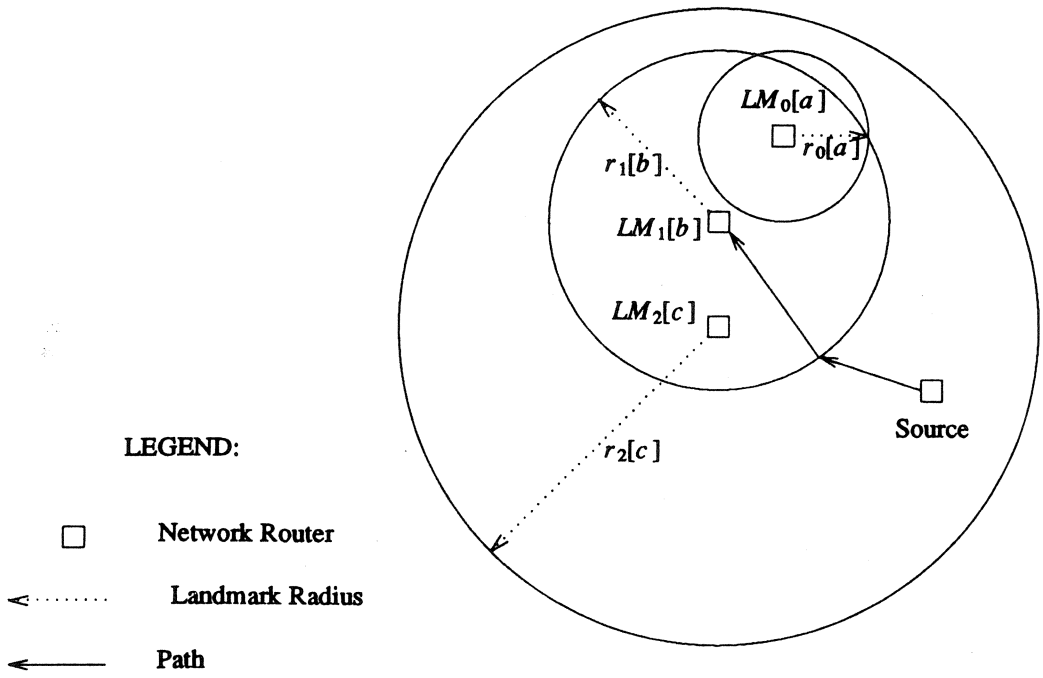
Figure 11
Estimated Performance for Networks Larger Than 800 Routers



1: $r_i/d_i = 3, D = 5.19 log_2 N, R = 4.4N^{.33}$
2: $r_i/d_i = 3, D = 2.07 log_2 N, R = 3.7N^{.38}$
3: $r_i/d_i = 4, D = 5.19 log_2 N, R = 3.4N^{.42}$
4: $r_i/d_i = 4, D = 2.07 log_2 N, R = 2.9N^{.48}$
5: $r_i/d_i = 7, D = 5.19 log_2 N, R = 2.9N^{.54}$
6: $r_i/d_i = 7, D = 2.07 log_2 N, R = 2.3N^{.62}$

messages addressed for the child (or one of its offspring) whose source is outside of the child's radius will route to the parent, but the parent will not be able to forward the message. We call this condition a partition. In particular, we say that the child Landmark has been partitioned from the parent Landmark.[14]

Figure 12
Landmark Hierarchy Partition



Note that a partition in the Landmark Hierarchy only affects messages whose source is outside of the radius of the partitioned child and whose destination is an offspring of the partitioned child. No other source-destination pairs are affected by the partition. This can be favorably compared to a partition in the area hierarchy where any source-destination pair, except that in which both routers are in the same partition, can potentially be effected by the partition. This is because two routers outside the partitioned area may be attempting to use that area as a transit area.

There are three ways of fixing a Landmark Hierarchy partition. (In this document, the word partition alone always refers to a Landmark Hierarchy partition.) Only one of the three ways actually involves the election of Landmarks.

The first and simplest way is for the child to increase its radius to again encompass the parent Landmark. This can only be done if the child can still see the parent—in other words, if the child is

within the radius of the parent. This will often be the case, such as is shown in Figure 13, because the radius of the parent will usually be substantially larger than that of the child. If the child cannot see its parent, then it has no way of knowing how much to increase its radius. In particular, it has no way of knowing if the parent simply no longer exists. If the parent is a global Landmark, then the child will always have the option of increasing its radius because it will always be able to see the parent.

At first glance, it may appear that this causes a potentially large amount of overhead to the network in the form of additional routing updates and routing table entries. However, this is not necessarily so. First of all, if a parent Landmark increases its distance from its child, it is very likely that some other routers have as well—namely, all of those routers downtree of the parent from the child. Therefore, there may be some routers who will actually see less routing overhead. Second, if the parent has increased its distance from its child, it may at the same time have moved closer to another child. This is particularly true of a mobile packet radio network, where the physical movement of packet radios accounts for the creation and destruction of links. In this case, while one child is increasing its radius, another is likely to be decreasing its radius.

The second way for a partitioned child to reestablish the hierarchy is to pick another parent. This will cause all of the offspring of the child to have new Landmark Addresses (or just Addresses for short). As discussed in Section 6.4.7, this reassignment of Addresses need not interrupt existing traffic or cause an excessive amount of additional traffic. Obviously, the child cannot pick another parent unless it is within the radius of another Landmark. It is possible, however, to arrange that any Landmark has two potential parents almost all of the time. Previous research shows that hierarchies constructed such that each child has two parents have acceptable routing table sizes ($O(\sqrt{N})$) (Tsuchiya, 1987). Even in this case, it is possible for the child to lose sight of both parents at the same time.

If a child picks a new parent which is at a higher level than its old parent, it has in essence become a higher level Landmark itself. This is third way to handle a partition: elect new Landmarks. This must be done when a child sees no potential parents, and so must elevate itself to the position of a higher Landmark. Creating new Landmarks has both the effects of 1) generating new Addresses and 2) generating new update traffic and routing table entries. For this reason, electing new Landmarks is the most obtrusive of the three approaches to handling partitions.

The creation or deletion of a Landmark does not necessarily happen because of a partition. More often it occurs as an adjustment of a non-partitioned hierarchy. This is discussed further in Section 4.3.2.

4.2.4 Use of Hop Count to Measure Landmark Radius

It is necessary for a child to know how far it is from its parent to set its Landmark radius appropriately. As discussed in Section 3, the means for a child to learn of its parent is via distance-vector style routing updates. From this information, however, a child is only able to learn how far its parent is from it, not how far it is from its parent. It is therefore necessary that all links have the same distance in both directions, at least as far as picking Landmarks and setting radii are concerned. Since it is also necessary that links be bidirectional, the distance from child to parent will be the same as the distance from parent to child.

It is therefore not possible to use a measurement which may be different in either direction on a link, such as delay. It may be possible to use a value proportional to the inverse of the bandwidth of the links as the distance measurement. It is not clear that doing this has any advantage, however, because bandwidth is both a function of available bandwidth (which is based on traffic) and of raw bandwidth. In addition, this will result in inaccuracies because the radius will have to be rounded up or down to an integer number of hops.

It seems natural, therefore, to use the simple measure of hop count to determine distance between two Landmarks. This does not mean that hop count must be used as the routing metric in the routing updates. Once a hierarchy is established, any routing metric may be used in the distance-vector routing updates (see Sections 5.6 and 7.2.3).

4.3 Design of Dynamic Landmark Hierarchy Management

In this section, we consider the design of the dynamic Landmark Hierarchy management scheme. We are interested in answering four questions:

1. When does a router become a Landmark?
2. When does a router cease being a Landmark?
3. How does a Landmark determine its radius?
4. Which parent Landmark does a child Landmark pick?

These questions must be answered in the context of several different situations, such as:

1. When the hierarchy is partitioned.
2. When the hierarchy is not partitioned.
3. When two networks are joined.
4. When a network is initially brought up.
5. When a router joins a network.

Situation 1 requires a fast response because communications is not taking place while the partition exists. Situation 2 should involve slower change with the goal of avoiding partitions. From the research which has been done, it does not appear to be terribly beneficial to reassign Landmarks in a non-partitioned hierarchy with the goal of making routing more efficient. This is best done by simply adjusting the radii, a much simpler and less obtrusive task. Situation 3 promises to be the most chaotic scenario, because there may be Landmark Label (or just Label for short) collisions that will need to be resolved. Situation 4 is difficult to imagine, because one normally thinks of networks as a series of incremental growths to the network, not as a large group of routers suddenly powering up at once. Even so, it is possible that whole networks can be employed quickly, such as mobile-packet radio networks. We hope that the same mechanisms used to deal with Situations 1 through 3 can handle Situation 4 as well. Situation 5 should be straightforward.[15]

Finally, we are able to articulate several, sometimes antithetical, overall goals for our Landmark Hierarchy management scheme.

1. It should generate as little overhead traffic as possible.
2. It should only generate traffic when necessary. In other words, it should avoid generating traffic when it is not necessary to adjust the hierarchy.
3. It should respond quickly to partitions.
4. It should dampen and eliminate oscillations, especially those resulting from adjustments to a non-partitioned hierarchy.

[15] As discussed in Section 2.4, the Landmark Label is one component of the Landmark Address. In particular, it is the component contributed by a single Landmark.

In discussing of dynamic hierarchy management, we first describe how a network can configure the hierarchy from scratch. In other words, we assume that an entire network is powered up at once. Although we do not expect this to happen often (if ever), the same techniques will also apply to patching the more pathological hierarchy partitions and to configuring a router which has just entered the network. This discussion will also serve to show what the normal non-partitioned structure of the hierarchy should be, thereby leading into a discussion of managing the non-partitioned hierarchy. Finally, we then describe how to deal with several classes of hierarchy partitions.

4.3.1 Configuring a Landmark Hierarchy from Scratch

There are only two types of messages required for configuring a Landmark Hierarchy. One message says "I am an LM_i, but I have no parent" (unsatisfied). The other says "I am an LM_i, and I have a parent" (satisfied). In addition, there are two sets of static parameters which each Landmark uses to determine whether it may need to become a higher Landmark or not. The two parameters are d_i^{max}, the maximum distance which any LM_i can be from an LM_{i+1}; and $r_i^{initial}$, the initial radius of an LM_i. The initial radius is generally reduced after the hierarchy is established to optimize routing table sizes. The first paper on Landmark Routing determines the limits placed on $r_i^{initial}$ and d_i^{max} (Tsuchiya, 1987).

When a router becomes an LM_i, it determines whether or not it is within d_i^{max} hops of an LM_{i+1} that does not have a full quota of children. If it is, then it is said to be "satisfied", and the closest LM_{i+1} without a full quota of children will become its parent. If it is not, then either it or some other LM_i within d_i^{max} hops of it that is not satisfied (that is, has no parent) must become an LM_{i+1}. An election among the non-satisfied LM_i is held to determine which will become the LM_{i+1}. This can be a simple so-called bully election (Garcia-Molina, 1982), where the LM_i with the highest Landmark Priority Number is the winner. The Landmark Priority Number can be a globally unique identifier, a randomly chosen number, or a pre-determined priority number. The important point to remember is that since there is a range of several factors in the number of Landmarks that can be elected, this algorithm can be very loose in terms of timing and in terms of the specific Landmarks chosen. This is one of the special advantages of the Landmark Hierarchy.[16]

---

[16]There is a network parameter which describes the maximum number of children a Landmark may have. The value of this parameter should be around 5 (see Section 4.3.2.2.4).

Note that each LM_i may have a different notion of which LM_i's are participating in the election. For instance, consider four non-satisfied Landmarks, $LM_i[a]$, $LM_i[b]$, $LM_i[c]$, and $LM_i[d]$. Assume that $LM_i[a]$, $LM_i[b]$, and $LM_i[c]$ are within d_i^{max} hops of each other, and that $LM_i[b]$, $LM_i[c]$, and $LM_i[d]$ are within d_i^{max} hops of each other, but that $LM_i[a]$ and $LM_i[d]$ are greater than d_i^{max} hops from each other. In this case, $LM_i[a]$ will see itself running an election with $LM_i[b]$ and $LM_i[c]$, but $LM_i[b]$ will see itself running an election with $LM_i[a]$, $LM_i[c]$, and $LM_i[d]$.

This, however, is not a problem. Assume that $LM_i[d]$ has the highest Landmark Priority Number. Then $LM_i[d]$ will win its election and become an LM_{i+1}, $LM_i[b]$ and $LM_i[c]$ will become satisfied, and drop out of the election with $LM_i[a]$. Then $LM_i[a]$ will win its election (by default) and become an LM_{i+1}.

Now we can describe the configuration process from power-up. We assume that every router can establish communications with its immediate neighbors—that is, those routers with which it shares a link. For the sake of discussion, we also assume that the values of d_i^{max} and $r_i^{initial}$ increase in powers of 2 as i increases, and that $d_0^{max} = 1$, and $r_0^{initial} = 2$.[17]

Initially, all routers become LM_0's, and send out Landmark Updates LU(0,2,U), which state that the LM_0 has no parents. (The nomenclature for a Landmark Update is LU(*level,radius,satisfied/unsatisfied*). LU(0,2,U) indicates that the update is from a level 0 Landmark, should travel a distance of 2 hops (because $r_0^{initial} = 2$), and is unsatisfied.) The LU's also contain the Landmark Priority Number. Each router runs an election with the unsatisfied LM_0's within $d_0^{max} = 1$ hops. The winners of these elections become LM_1's, and send out Unsatisfied Landmark Updates LU(1,4,U). The losers send out Satisfied Landmark Updates LU(0,2,S), thus pulling themselves out of any remaining elections. In addition, the losers will obtain a level 0 Landmark Label. This is discussed further in Section 4.3.2.2.1. They will also adjust their radii based on the distance to their parents and their children, if any.

Next, the LM_1's run elections with other LM_1's within d_2^{max} hops of each other. These elections take place only after an appropriate delay, or after each LM_1 has heard from a certain number of other LM_1. This prevents an LM_i from prematurely electing itself a winner before other LM_{i-1} have had a chance to become LM_i. These elections will result in a set of LM_2's, which will in

[17] Analysis in (Tsuchiya, 1987) sets limits on these parameters. Further simulation is required to determine the optimal values.

turn run elections with each other. This will continue until, at some high level, there will be only one Landmark. When this happens, the hierarchy is complete.

The resulting structure is H levels of hierarchy, with a single routing as Landmark at the root level and all routers as Landmarks at the lowest level. The parent-child relationships form a single tree. The Addresses of the routers reflect this tree structure. Each parent will have some number of children ranging from 1 (itself) to several. In previous research (Tsuchiya, 1987) using the parameters of d_i^{max} and $r_i^{initial}$ given above, it was found that each parent had on the average between 2 and 3 children. The variance on the number of children was not measured. However, since the algorithm for configuring the hierarchy resulted in a fairly uniform distribution of Landmarks, it is unlikely that the variance is very large.

Once the network is configured as described above, it is non-partitioned. However, it is not necessarily in an "optimal" configuration. We do not mean optimal in terms of efficiency measures such as routing table sizes or path lengths. Previous research (Tsuchiya, 1987) shows that there is considerable flexibility in the assignment of Landmarks with regards to these efficiency measures. It is the radii which most strongly affect efficiency. What we mean by optimal is that the configuration is such that a partition, especially one affecting a large number of routers, is the least likely. In addition, changes in the network topology, while they may not always cause a partition, may make the network more likely to experience a partition by moving Landmarks further from their parents. Therefore, the hierarchy should adjust itself to a more optimal configuration while in a non-partitioned state. A non-partitioned Hierarchy adjustment is less obtrusive than a partitioned Hierarchy adjustment because it is controlled and can take place smoothly over time. In the following sections, we describe this adjustment process.

4.3.2 Managing the Non-Partitioned Hierarchy

In this section, we discuss two aspects of managing the non-partitioned hierarchy. First, we discuss the issue of creating multiple global Landmarks. Then we discuss the issues associated with adjusting Landmarks in general.

4.3.2.1 Multiple Global Landmarks. One of the problems with the non-partitioned hierarchy described above is that there is a single root Landmark at the top. This means that all Addresses depend on that Landmark. If that Landmark dies, or somehow changes its Label, every router in the network will be affected. To reduce the impact of this problem, we make some number of Landmarks T^G global by simply increasing their radii to infinity. We do this to the T_G

Landmarks that are highest in the Hierarchy. The result of this is that there will be some level i above which all Landmarks will be global, and at which some Landmarks will be global. The level at which this occurs is the level i such that $T_i \geq T^G$ and $T_{i+1} < T^G$.

The levels of hierarchy above this level will still exist and will still be reflected in the Landmarks Updates sent. It is necessary to keep these levels of hierarchy to deal with the problem of merging networks (see Section 4.3.4). It will not, however, be necessary to encode these levels in the Landmark Addresses.

The resulting Address structure is multiple Address trees, each with its root at the global Landmark. Above these multiple roots, however, is the rest of a single Landmark Hierarchy that is not reflected in the Addresses. The single monolithic Landmark Hierarchy provides stability (especially for merging networks). The multiple Address trees lessen the independence of routers on any given Address tree.

Two problems remain: 1) how to determine the number of global Landmarks T^G and 2) how to determine which Landmarks at which level become global. These questions are answered in turn in the following two sections.

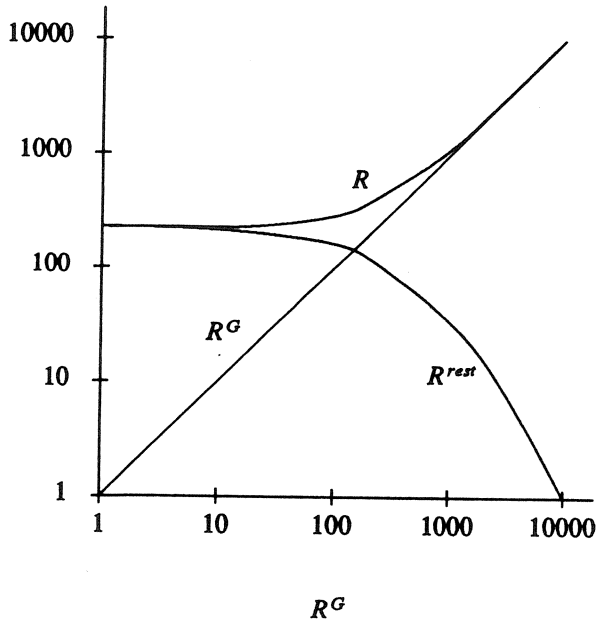
4.3.2.1.1 Determining the Number of Global Landmarks. There should be as many global Landmarks as possible within some set of performance constraints, so that the impact of each global Landmark on its offspring can be as small as possible. Of course, the number of global Landmarks affects the total number of routing table entries R in a router. Let us therefore first consider the impact of a large number of global Landmarks on the routing table sizes.

Divide the total number of routing table entries R into two groups: those which come from the global Landmarks (and above) R^G and all others R^{rest}. If there is only one global Landmark, then the large majority of R comes from the R^{rest} contribution. If all Landmarks are global, then all of R comes from the R^G contribution. As R^G grows, therefore, R^{rest} shrinks.

Figure 14 shows R, R^G, and R^{rest} for a network as R^G is increased from 1 to N. In this example, $N = 10000$ (roughly the number of routers we might expect to see in a large internet), $D = 30$, $C = 4$, $r_0 = 1.5$, and $\frac{r_i}{d_i} = 3.5$. The technique used to calculate the routing table sizes is derived in (Tsuchiya, 1987), and will not be repeated here.

We see from Figure 14 that R^G can increase fairly dramatically before R begins to increase significantly. This is because the decrease in R^{rest} offsets the increase in R^G until R^G becomes

Figure 13
Effect of the Number of Global Landmarks on Routing Table Sizes



fairly large. In fact, while R^G increased from 1 to 124, R increased from 231 to 286, an increase of only 24%. At the same time, however, the impact of a single global Label change was reduced from 10000 routers to only 81 routers on the average (of course, this is only counting routers, not hosts). Also note that R^G accounts for 43% of the total routing table entries.

It seems appropriate that the number of global Landmarks should be a function of the fraction of the routing table contributed by the global Landmarks. Consider, for instance, the rule of thumb for routing table sizes (Tsuchiya, 1987): $R = 3\sqrt{N}$. Even if we ignore the reduction in routing table size due to decreasing R^{rest}, if we make $R^G = \sqrt{N}$, we have only increased the routing table size by 33%, but we have reduced the impact of a global Label change to $\sqrt{N}$ routers. In this case, the fraction of the total routing table contributed by the global Landmarks is 25%. We believe an appropriate figure for the percentage of R consumed by R^G is anywhere from 25% to 50%.

The idea, then, is that each router be able to calculate the number of global Landmarks that must be added or deleted $T^{G\Delta}$ based on the number of global Landmarks T^G ($T^G = R^G$), the global network average number of routing table entries R, and some acceptable range of percentages $p_{\min} \leq \frac{R^G}{R} \leq p_{\max}$.

It is possible to determine the network-wide average for R in a straightforward and efficient manner. Each LM_0 includes in its Landmark Updates its own R. Then each LM_1 calculates an average $R^{children}$ by adding all of its children's values and dividing by the number of children. The LM_1 then puts a tuple containing this average and the total number of children in its updates. The LM_2 then averages the values from its children, and so on up the hierarchy. The global Landmarks then include this tuple in their Landmark Updates. The network-wide average is then calculated by all routers, resulting in a network-wide uniform value of R. These values need be sent out only when they have changed a significant amount, say 10% from the previous advertised value.

Based on this R and the number of global Landmarks $T^G = R^G$, we calculate $p^{real} = \frac{R^G}{R}$, and $R^{rest} = R - R^G$. If $p^{real} > p_{\max}$ or $p^{real} < p_{\min}$, then there are too many or too few global Landmarks respectively. If this is the case, then the number of global Landmark that need to be added or deleted is calculated by every node as:

$$T^{G\Delta} = R^G - \frac{p_{mid}\, R^{rest}}{1 - p_{mid}} \qquad 6$$

where $p_{mid} = \frac{1}{2}(p_{\min} + p_{\max})$.

The value $T^{G\Delta}$ is the surplus of global Landmarks. If it is positive, then $T^{G\Delta}$ global Landmarks must be removed. If it is negative, then $|T^{G\Delta}|$ global Landmarks must be added.

The reason for allowing an acceptable range of percentage is that if there were only one percentage (p_{mid}), Equation 6 would overshoot or undershoot that percentage, thus requiring subsequent calculations and resulting in a damped oscillation. Typical values are $p_{\min} = .3$, $p_{mid} = .4$, and $p_{\max} = .5$.

Equation 6 is recalculated whenever there is a change in either R or R^G. R is only advertised when there is a significant change in its value. For instance, each Landmark only advertises its R when it has seen a 10% or greater change in the values of R for its offspring. This way, any given global Landmark is unlikely to be advertising a new R very often. When it does, however, all

routers will recalculate $T^{G\Delta}$ at roughly the same time, so that the function of adding or deleting global Landmarks is synchronized.

4.3.2.1.2 Adding and Removing Global Landmarks. Given that it is known how many more or less global Landmarks are needed at a given point in time, it is a fairly simple matter to determine which Landmarks should either become global, or stop being global. We do this by defining an ordering of Landmarks from those in the highest hierarchy level to those in the lowest. In other words, no level i Landmarks will become global until all level i+1 Landmarks are global.

Within a hierarchy level the ordering is as follows (assume that all level i+1 Landmarks are global, that there are n level i+1 Landmarks, and that we wish to describe the ordering of the level i Landmarks). The 1^{st} one is the child with the (numerically) highest Landmark Label of the children of the level i+1 Landmark with the highest Label of the level i+1 Landmarks. The 2^{nd} one is the child with the highest Label of the children of the level i+1 Landmark with the 2^{nd} highest Label of the level i+1 Landmarks. The n^{th} one is the child with the highest Label of the children of the level i+1 Landmark with the n^{th} highest (in this case, the lowest) Label of the level i+1 Landmarks. The n+1^{th} one is the child with the 2^{nd} highest Label of the children of the level i+1 Landmark with the highest Label of the level i+1 Landmarks, and so on.

For instance, assume there are three level i+1 Landmarks with Labels 7, 8, and 9, which are all global. Assume also that each of them have three children (not including themselves) with Labels 1, 2, and 3. (We say not including themselves because every level i+1 Landmark has a level i child which is in the same router as itself. Since the level i+1 Landmark is already global, the child is in essence also already global, and so is not compared with the other children in the ordering.) The order in which level i Landmarks become global is 9.3, 8.3, 7.3, 9.2, 8.2, 7.2, 9.1, 8.1, and 7.1. They stop being global in the reverse order.

Any Landmark can very easily determine where it is in the ordering. All Landmarks know of all global Landmarks, and all Landmarks know of their own siblings. Therefore, a Landmark, say level i, knows how it is ordered among its siblings, knows the ordering of the level i+1 Landmarks, and knows how many of the children of the level i+1 global Landmarks have become global Landmarks.

4.3.2.2 Adjusting Landmarks is a Non-partitioned Hierarchy. The idea behind adjusting Landmarks in a non-partitioned hierarchy is to create a uniform distribution of Landmarks, both in terms of the placement of level i Landmarks with respect to each other and in terms of the number of children Landmarks have. This uniform distribution minimizes the possibility of a hierarchy partition by keeping Landmarks close to their parents. The goal is for Landmark adjustments to be as simple as possible. We define three types of adjustments:

1. An LM_i adopts a new LM_{i+1} parent.
2. An LM_{i+1} demotes itself to an LM_i.
3. An LM_i is promoted to an LM_{i+1} via the election process.

Since each of these adjustments is made when the hierarchy is intact and routing is successfully taking place, the adjustments can be made in a leisurely and controlled fashion. When a Landmark changes roles, getting a new Landmark Label for itself and its offspring in the process, it will typically keep its old role for a period of time. This allows the address binding function to take place gradually over that period of time, thus avoiding a surge of address binding traffic. It also allows subsequent adjustments that result from a previous adjustment to occur over a period of time.

We discuss each of these adjustment types in turn.

4.3.2.2.1 Adopting a New Parent. When an LM_i finds itself closer to some LM_{i+1} than it is to its own parent, it should adopt the new LM_{i+1}. This reduces the routing table sizes because the LM_i can reduce its radius. More important, however, is that it reduces the potential for the LM_i to be partitioned from its parent. This is for two reasons. First, since the LM_i would be closer to its parent, there are a smaller number of intermediate routers and links which could crash, thus potentially causing the LM_i to be partitioned. Second, since the LM_i is closer, its parent's radius would usually extend further past the LM_i, thus making it less likely that the LM_i could lose track of its parent in the face of topology changes.

In adopting a new parent, a Landmark must be careful of several things. First, it must avoid oscillating between the old parent and the new parent. This could happen if a router or link was periodically crashing and coming up again, causing the distance to an LM_{i+1} to change between two values such that first it was farther than the distance to another LM_{i+1}, and then it was closer. One way we can reduce the possibility of this kind of oscillation is with hysteresis. That is, we don't

allow an LM_i to switch to a new parent unless the new parent is closer than the old parent by some number of hops. This, however, doesn't prevent all oscillations. It also prevents the LM_i from adopting a better parent in many cases.

Another way to reduce the possibility of oscillation is to wait a period of time before adopting a new parent. If this time period is longer than that of the oscillation period, then the new parent will not be chosen, and oscillation will not occur. An intuitively appropriate solution to this problem seems to be to use a small hysteresis value (about one hop), and a moderate time delay (several minutes at least for land-based internets).

Second, we wish to avoid the situation where two or more LM_i try to adopt the same LM_{i+1} at the same time, thus causing the LM_{i+1} to have too many children, which in turn results in one or more of the LM_i having to adopt yet another parent (or run an election). This can be avoided with a simple three-way handshake. When an LM_i sees that an LM_{i+1} has room for more children, and wishes to adopt it, it communicates this to the LM_{i+1}. If the LM_{i+1} has no outstanding adoptions in progress, it reserves a space for the LM_i, and tells the LM_i that it may be adopted. At this time, the LM_{i+1} also chooses a Label for the new LM_i from open slots in its Label space. Since the LM_{i+1} must advertise its new child to all of its other children (partly for the purpose of address binding, and partly for the purpose of choosing Label values), the response can be in the form of a Landmark Update. If the LM_{i+1} has another LM_i trying to adopt it which would exhaust its quota of children, then the LM_{i+1} can disallow the new adoption. In this case, the LM_i must either find another LM_{i+1} to adopt, or run an election (even if the election is only with itself).

Once an LM_i has been accepted by its new parent, it needs to send out a Landmark Update indicating its new Label. This allows its offspring to start the process of rebinding their Addresses. It also allows its new parent to see that the adoption has taken place (the third part of the three-way handshake). The LM_i should keep the old parent for a period of time while it and its offspring are rebinding their Addresses. In other words, they will have two valid Addresses for a period of time.

4.3.2.2.2 Demotion of a Landmark. There are two situations where an LM_i will demote itself to an LM_{i-1} or lower. First, the LM_i will demote itself if it has no children (other than itself, of course). It can lose its children either because its children adopted new parents, or because the children crashed or were separated from it by topology changes. Before an LM_i demotes itself, it must adopt a new LM_i parent. If it cannot, then it will not demote itself.

The other reason that an LM_i will demote itself is because it is too close to another LM_i. This can happen because of the addition of links or routers. We need another parameter, d_i^{min}, which gives the minimum distance two LM_i's can be from each other. In this case, the LM_i will demote itself as many levels as necessary to satisfy d_i^{min} for all levels of i. Clearly, only one LM_i should demote itself—the one with the smallest Landmark Priority Number for instance.

This demotion causes more network perturbation than the no-children demotion, because it affects more offspring. The LM_i demoting itself can expect to have one or more children. These children must find new parents, either through adoption or election. Again, the demoting LM_i will keep its old status for a period of time to smooth the transition.

4.3.2.2.3 Promotion Through Election. The only way that an LM_i can become an LM_{i+1} is through election. Elections occur any time a Landmark is not satisfied—it has no valid parent within d_i^{max} hops. This can happen when the network is non-partitioned, because an LM_i can temporarily have an LM_{i+1} for a parent which is more than d_i^{max} hops away (for instance, if a recent topology change caused the parent to become further away). It can also happen when a parent is going through the process of demoting itself. In other words, a partition is immanent. The election takes place exactly as described in Section 4.3.1, and needs no further discussion here.

4.3.2.2.4 On the Number of Children per Landmark. We are interested in determining an appropriate value for the maximum number of children per Landmark. We ran an experiment where we simulated networks with from 2 to 10 children per Landmark on the average. These networks were generated the same way as the random experiments documented in previous work (Tsuchiya, 1987). The results are shown in Table 2.

Here we see that, as the average number of children is raised from 2 to 6, the routing table size R and the number of hierarchy levels H shrink, while the increase in path length over shortest path $\hat{P}$ grows. This means that the efficiency of the hierarchy does not necessarily get worse as the average number of children increases to 6. Higher than 6, however, we see that both the routing table sizes *and* the path lengths increase with a higher average number of children. In addition, the number of hierarchy levels begins to level out.

We conclude, then, that we should limit the number of children per Landmark to around 6. This allows us to require only 3 bits of Label space per hierarchy level (except for the global level) for the Address (see Section 4.3.2.1).

Table 2
Number of Children per Landmark

	H	R	$\hat{P}$
2 Children	9	22.98	1.10
3 Children	6	26.40	1.09
4 Children	4.73	29.55	1.07
6 Children	3.73	36.47	1.05
8 Children	3.5	40.15	1.06
10 Children	3.5	43.2	1.06

Note once again the relatively insensitivity of the Hierarchy performance to the number of Landmarks. Here we varied the number of Landmarks over a factor of 4 without necessarily impacting performance. As already stated, this eases the constraints, such as timing, placed on the election algorithm.

4.3.3 Managing the Partitioned Hierarchy

Now that we have discussed how to manage the non-partitioned hierarchy, we can discuss how to manage the partitioned hierarchy. Two points need to be made. First, the purpose of the non-partitioned hierarchy management is to maneuver the hierarchy into a position where partitions are the least likely—essentially, to optimize the hierarchy. Second, having non-partitioned hierarchy management means that all we need to do in partitioned hierarchy management is to get the hierarchy to a state where it is not partitioned. Since communications are not taking place for a set of routers while the hierarchy is partitioned, partition repair should be rapid. Once the partition is repaired, adjustment of the hierarchy can occur at a more leisurely, more stable pace.

As stated in Section 4.2.3, there are three types of partitions:

1. The child can still see its parent, but the parent can no longer see the child.

2. The child can no longer see its parent, but can see another Landmark with room for more children at the same level as the parent.

3. The child can see no valid Landmarks at the level of its parent.

The first two cases are easy to handle. In the first case, the child simply increases its radius to encompass the parent. In this case, no Address change has taken place. Depending on the situation, the non-partitioned hierarchy management techniques may cause a subsequent adjustment of Landmarks.

In the second case, the child adopts a new parent, and adjusts its radius accordingly. This is similar to the non-partitioned adoption, except that in this case, there may be no overlap during which the child has two Addresses. Therefore, address binding will have to be expedited.

We believe that these first two cases will constitute the nearly all of partitions. In the simulations documented (Tsuchiya, 1987), there were very few cases where a router had only 1 Landmark at some hierarchy level in its routing tables, and therefore would not have other parents to choose from in case of a partition. Furthermore, most of these cases were for the networks where the radii were as small as possible, which is not a normal or recommended mode of operation.

In the third situation, elections must take place to reestablish the hierarchy above the Landmark which sees no parents. The elections start at the level at which the Landmark can see some peers (it is possible that the LM_i cannot see any LM_{i-1}, LM_{i-2}, and so on). The election continues until all Landmarks are satisfied—that is, have valid parents.

4.3.4 Managing Merging Networks

Probably the most difficult hierarchy management problem is dealing with two or more networks or network segments which merge. The main problem here is that there will most likely be some number of global Landmarks which have the same Label. Another problem is that there will be a surplus of global Landmarks.

We categorize merging networks into two types: those where one network is significantly smaller than the other and those where the networks are roughly the same size. The crucial point is

[18]For the sake of this algorithm, if two networks have global Landmarks at the same level i, then we consider them *by definition* to be the same size.

that if one network is smaller, it will have fewer hierarchical levels, and its global Landmarks will be at a lower level than the larger network's global Landmarks. If the networks are roughly the same size, then their global Landmarks will be at the same level.[18]

When a large and a small network merge, the global Landmarks from the small one below the level of those from the large one will be superfluous, and will need to go away. We wish to do this with a minimum of perturbation, especially to the large network portion. When the two networks are roughly the same size, then there will probably still be too many global Landmarks, but global Landmarks from both sides will need to disappear. In both cases, there will be some number of Label collisions (where global Landmarks from each network portion have the same Label).

We need two simple rules to deal with merging networks. First, a router should never forward the LU for a global Landmark if the level of the global Landmark is below that which the router expects to see. From the algorithm that determines which Landmark is next scheduled to either become global or stop being global, every router knows the level at which the next global Landmark will be chosen. If no router forwards LUs for global Landmarks below what it considers the proper level, then the global LUs from the small portion will not be seen in the large portion, thus saving the large portion from any additional traffic and computation.

The other rule is that, when a router sees a global LU with the same Label as a current one, but from a different router (which it can distinguish because of the router ID), it passes the LU on and processes it normally, except that it does not add the global Landmark to its routing table or the Intermediate Hash Space table used for binding IDs to Addresses. This way, global Landmarks from one merging network portion will not be confused with global Landmarks from the other network portion until the colliding Labels are resolved. Since the LUs are passed on, however, the colliding Landmarks will hear about each other and resolve the Labels. When the Labels are resolved, the new global Landmark can be added to the routing table.[19]

Except for these two rules, the existing techniques for managing the hierarchy will suffice to merge two networks. In particular, when two networks merge, the root Landmark from the smaller portion will suddenly see peers and possibly higher level Landmarks, and will know that it must continue the election process. It can then either adopt a parent, or run an election, depending on

[19]The Intermediate Hash Space is required for the Assured Destination Binding function. Its specific use is not of concern here—we are only concerned that new global Landmarks can be distinguished and separated from current ones.

whether it is satisfied. If two equal size networks are joined, then the two root Landmarks can run an election between themselves.

4.3.4.1 Picking and Resolving Global Addresses. We are interested in choosing a method of picking global Landmark Labels so that when two networks merge, we minimize the number of Label collisions. We believe the only way to do this is to have global Landmarks pseudo-randomly pick Labels from the global Label space.

This complicates the selection of Labels for any given global Landmark. Clearly, the simplest global Label assignment scheme is for global Landmarks to pick Labels in order of their assignment. For instance, the first Landmark to become global (the root), would pick the value 1, the next the value 2, and so on. Since global Landmarks are chosen in a deterministic order, this simple method would result in no collisions. (There would, however, have to be some garbage collection mechanism for global Landmark that crash and leave a Label open.)

The problem with this is that, when two networks that have assigned Labels the same way merge, every Label (or nearly every Label) would collide. By picking global Labels pseudo-randomly, we can reduce the severity of this problem. We recommend a hash function with the Landmark's ID as the hash key for the pseudo-random function.

Since Labels are being pseudo-randomly picked, there is the danger of Label collisions in the global Landmark assignment process—for instance, if two Landmarks become global at the same time and happen to pick the same Label. We can avoid this problem by having each Landmark wait for the Landmark ahead of them in the selection process to become global, and therefore know which Labels are left to pick from. Of course, the Landmark will need to have a time-out set in case the other global Landmark never materializes. In the rare event that a collision does occur, it can be resolved in the same way as collisions that result from merging networks.

5 ADMINISTRATIVE BOUNDARIES AND AUTONOMY IN LANDMARK ROUTING

A disadvantage of the Landmark Hierarchy is that it does not recognize boundaries—boundaries between different network types, between differently administered networks, and between groups of routers which communicate extensively. It is therefore necessary to add boundaries to the Landmark Hierarchy to accommodate certain requirements.[20]

We classify those requirements into three categories:

1. A group of routers should be able to manipulate paths so that all traffic between two group members never transits routers outside of that group. A variation on this is that a group of routers should be able to still route traffic between group members in the face of routing failures in routers outside that group.

2. A group of routers should be able to manipulate paths so that all traffic between two non-group routers will never transit routers within the group. A further refinement on this is that a group of routers should be able to select which non-group routers can transit group routers.[21]

3. A group of routers should be able to operate their routing protocols (metrics used, frequency of updates, types of service) differently than another group of routers.

More succinctly put, routers should be able to choose certain paths, prevent certain paths, prioritize routing information. In addition, groups of routers should be able to act with some limited autonomy from other groups. Furthermore, any router should be able to belong to several groups whose relationships may be nested or overlapping.

Let us illustrate this. Consider the structure of the NSFNET, with campus networks and regional networks. The campus networks should be able to restrict traffic between campus locations to stay within the campus net. Further, they should be able to restrict traffic from other

---

[20]Of course it is this lack of boundaries that makes the Landmark Hierarchy easier to dynamically manage than the area hierarchy, so it is only a disadvantage in a certain perspective.

[21]Note that the methods introduced in this section do not provide access control or authentication. These functions must be added by the administration if they are required. What is provided here is a way for routers to avoid sending packets down paths which are only going to fail because of access control restrictions.

[22]The existence of overlapping groups does not mean that there necessarily is a set of equipment that is somehow jointly owned and maintained. It only means that there is a desire to control paths for certain groups that happen to overlap with other path control requirements.

campuses from transiting their nets. This type of restriction can be nested in the regional network, for instance with intra-regional traffic restricted to regional facilities only. Now assume that two campuses in different regions wish to establish a link for certain experiments between the two campuses, and that they wish to limit traffic on that link to only those two campuses. Here we have a situation where the two campuses are involved in multiple, overlapping groups. They each belong to a campus, to a regional group, and to a third group which includes only the two campuses.[22]

5.1 Administrative Zones

To handle these situations, we have devised the Administrative Zone. The first requirement of an Administrative Zone (or just Zone, for short) is that all members be connected—that is, there exists a path from any Zone member to any other Zone member which crosses only Zone members. This is done through appropriate configuration of routers and links. (This requirement is similar to that of an area in an area hierarchy.)

The second requirement of a Zone is that Zone members choose Landmarks and Addresses so that any intra-zone routing uses only Landmarks within the Zone. In other words, when a Zone router examines an Address for routing to a destination within the Zone, at least one of the Landmarks in its Address must both 1) be a Zone member and 2) match one of the Landmarks in the destination's Address. If this requirement is satisfied, then all paths between Zone members will stay within the Zone.

In order to accomplish this, Zone members must be able to distinguish between Landmarks within the Zone and those outside of the Zone. The best way to do this is to have routers which border other Zones tag LUs from that Zone as having come from another Zone. Since LUs from other Zones are tagged based on which link they came in on and not based on information provided by a router in the other Zone, routers outside the Zone cannot lie about whether they are Zone members.

Since this tagging is simple to do (see Section 5.2.1), all that is necessary to satisfy the second requirement is to have Zone members apply the Landmark satisfaction criteria used to build the hierarchy to Zone members only. In other words, when a Landmark is deciding whether or not it is close enough to a higher level Landmark, it should only consider Zone members. Further, when it runs an election, it should do so with other Zone members only.

The result is that elections will occur only among Zone members until the hierarchy is built to the point where there is only one level *i* Landmark in the Zone. We call this Landmark the Zone-root Landmark. Since the Zone-root Landmark will have no other Zone members with which to run elections, it must participate in elections with Landmarks outside of the Zone. All Zone members will have the Zone-root Landmark and its offspring in their Addresses, and all intra-Zone routing decisions will be made on these Landmarks only. The Zone is essentially a mini-Landmark Hierarchy within a larger Landmark Hierarchy.

In the following sections we develop the idea of Zones. In particular, we discuss nesting of Zones and overlapping Zones, the effect of Zones on routing tables and paths, controlling traffic flow through Zones, and how to deal with Zone partitions. Finally, we discuss autonomy between Zones.

Through the use of Zones, and particularly by hiding information within a Zone, the structure of an area hierarchy can very closely be approximated. The difference here, however, is that the auto-configuration and partition repair functions are still based on the Landmark Hierarchy that exists as a kind of inner skeleton holding up the Zone hierarchy. This will become clear in the following sections. Through the use of the Landmark Hierarchy and Zones, one has full control over how explicit one's administrative boundaries are. With the area hierarchy, these boundaries are forced upon you whether you want them or not.

5.2 Nested and Overlapping Zones

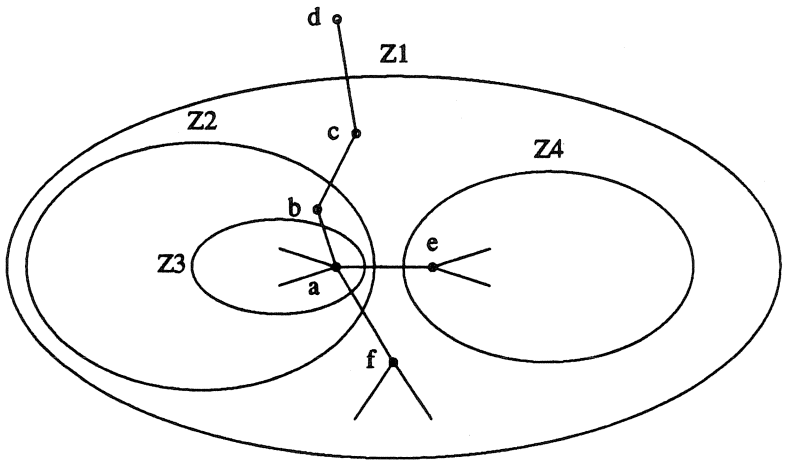
As stated in the previous example, it may be necessary to nest or overlap Zones. Both requirements are possible, but overlapping Zones requires more explicit Zone membership information than the simple tagging of LUs as in-Zone or out-Zone by the border Landmarks. As a result, authentication may be required (depending on the level of trust).

5.2.1 Nested Zones

Figure 15 shows a portion of a network with three levels of nested Zones. Note the various possible relationships between routers in different Zones. Router *a* considers Router *b* to be in a different Zone, but Router *b* considers Router *a* to be in the same Zone. In other words, Router *b* sees Router *a* as a *Z2* router, and does not care that Router *a* is in a sub-Zone within *Z2*. Likewise, Router *a* sees Router *c* as being two Zones away, but Router *c* sees Router *a* as being in the same Zone as itself. Both Routers *a* and *e* view themselves as being in different Zones, but Router *e* sees Router *a* as being one Zone away, whereas Router *a* sees Router *e* as being two Zones away. The

point here is that relationships between routers in different Zones are not necessarily symmetric. The significance of this is discussed below.

Figure 14
Nested Zones Example



We see from Figure 15 that Router *a* will configure first with other routers in *Z3*. Once *Z3* is configured internally, the *Z3* Zone-root Landmark can configure with routers in *Z2*. The *Z2* Zone-root will then configure with *Z1* Landmarks, and so on.

For this to happen, Router *a* must be able to recognize whether LUs are from *Z3* routers, from *Z2* routers (excluding *Z3* routers), from *Z1* routers, and so on. One way to accomplish this is to have a field in the LU which is incremented every time the LU enters a Zone and decremented each time it leaves a Zone. This parameter, which we call the Zone-distance parameter, cannot be decremented below zero. For example, when *a* receives an LU from *b*, it increments the Zone-distance by one (notice that *b* does not decrement the Zone-distance in this transaction). Likewise, when *b* receives an LU from *c*, it also increments the Zone-distance by one. Therefore, when *a* receives an LU originated by *c*, *a* will know that the LU came from two Zones away. When *a* sends an LU to *f*, it will decrement the Zone-distance by two, but *f* will not increment it. When *a* sends an LU to *e*, *a* will decrement the Zone-distance by two (but not below zero), and *e* will

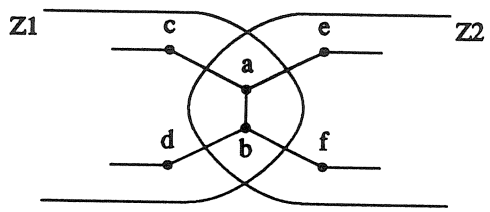
increment it by one. Therefore, *e* will see an LU originated by *a* as being one Zone away because *a* is outside of *Z4*.

Note that for nested Zones, there are no semantics required in LUs to indicate Zone membership—the Zone-distance parameter is adequate for determining Zone relationships. Note also that each router requires only one Address, because its Zone membership is a strict tree. Each Landmark is in one Zone only for each level of Zone. Unfortunately, this is not the case with overlapping Zones, which we discuss next.

5.2.2 Overlapping Zones

In Figure 16, we show two overlapping Zones, with Routers *a* and *b* in both Zones. Clearly, *a* (and *b*, but we will focus on *a*) must participate in the configuration of the hierarchy in both Zones *Z1* and *Z2*. Since any Zone member must assume an Address with Landmark ancestors up to and including the Zone-root, *a* must have two Addresses: one from *Z1* and one from *Z2*. Furthermore, if a *Z1* router sends a message to *a*, it must use *a*'s *Z1* Address to guarantee that the message will stay in *Z1*. The question, therefore, is how can *a* configure in both *Z1* and *Z2* without confusing messages between Zones?

Figure 15
Overlapping Zones Example



If *a* receives an LU from either *c* or *e*, it knows that the LU came from *Z1* or *Z2* respectively, and therefore knows how to configure into *Z1* and *Z2*. The problem arises when *a* receives an LU from *b*. From the Zone-distance parameter alone, Router *a* has no way of knowing whether the LU originated from *Z1* or *Z2*.

For instance, consider an LU originating from Router *d*. When *b* receives the LU, it must treat the LU two ways: as a member of *Z1* and as a member of *Z2*. As a member of *Z1*, it will not increment the Zone-distance parameter. As a member of *Z2*, it will increment the Zone-distance parameter, and treat the LU as it would any LU from a different Zone. Likewise, if *b* receives an LU from *f*, it will also treat it two different ways. When *b* passes an LU on to *a*, *a* has know way of knowing whether the LU came from *Z1* or *Z2*, because the Zone-distance parameter may be incremented or not incremented either way. The Zone-distance parameter, therefore, is not sufficient information for *a* to know how to process an LU received from *b*. Instead, *b* must explicitly label the LU as having come from *Z1* or *Z2*—in other words, *the Zones must have labels*. We call this label the Zone ID, or ZID.

Clearly, labeling of Zones requires coordination among several routers. For instance, *c* and *d* must have the same label for *Z1*. Should labeling of Zones be automatic, or should labels be pre-configured. Since each Zone has a Zone-root, it would be possible to use a label derived from the Zone-root Landmark, such as the Zone-root's Address or ID. This could not happen until the hierarchy in the Zone was configured. The routers in overlapping Zones, however, cannot configure until the Zones are labeled. Therefore, the routers in overlapping Zones would have to postpone joining the hierarchy until the hierarchy in the non-overlapping portion of the Zone was completely configured.

To statically preconfigure the Zone IDs, we only need to preconfigure the border routers. Non-border routers in non-overlapping portions of the Zone do not need to know their Zone IDs because they can differentiate Zones using the Zone-distance parameter. Non-border routers in overlapping portions of the Zone do not need to know their Zone IDs because they will eventually receive LUs with Zone IDs. These routers can infer from the various LUs that they are in overlapping Zones. Given that the border routers need to be pre-configured as border routers whether or not they are in overlapping Zones and whether or not the Zone IDs are dynamically assigned or statically assigned, the burden of adding a Zone ID is not excessive. Dynamic assignment of Zone IDs does not appear to be terribly beneficial.

One might argue that, with the preconfiguring that has to be done to accommodate Zones, there doesn't appear to be much auto-configuration in the Landmark Hierarchy. Note, however, that Landmark Addresses do not need to carry Zone information—that information is carried in the LUs and bound to Addresses as the Addresses are assigned. As a result, no reconfiguration of the

Zone ID is necessary in the event of topological changes (for instance, those which cause the partition of a Zone) as is the case with the area hierarchy (see Section 4.1).

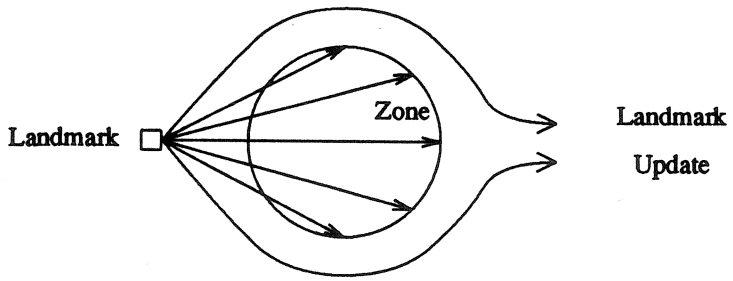
5.3 Controlling Third-Party Traffic

One of the requirements of Zones is to allow a Zone to not pass third-party traffic—that is, to not forward any traffic which does not originate or terminate in the Zone. We can partition this requirement into two classes: Zones which allow no third-party traffic and Zones which allow third-party traffic for a certain set of sources and destinations, but not others.

The first class is easy. If a Zone never lets LUs which originated outside of the Zone leave the Zone, then routers outside the Zone will never see that a Landmark is reachable through the Zone.

Figure 17 illustrates this. The arrows indicate the path of the LU from the Landmark. Note that the LU enters the Zone straight from the Landmark, but ends without leaving the Zone. The LU which does not enter the Zone "wraps around" the Zone to those routers which would have otherwise seen the LU straight from the Landmark. The path of traffic going to the Landmark is the reverse of that of the LU. Therefore, the traffic from the routers to the right of the Zone will go around the Zone on its way towards the Landmark.

Figure 16
Preventing Third-Party Traffic



Note that the Zone border routers should implement access control to enforce prevention of third-party traffic. Halting LUs only prevents the spread of information about a path through a Zone, but does not prevent a router from attempting to pass traffic through the Zone.

5.3.1 Selectively Controlling Third-Party Traffic

A Zone may wish to transit certain third-party traffic, but not others. In some cases, this requirement may be satisfied using Zones. For instance, if all government networks were required to transit each other's traffic, but not commercial, private, or foreign traffic, the government networks (each of which is a Zone) could form a super Zone around the individual Zones. This super Zone would not transit any third-party (non-government) traffic. This solution requires that all government Zones can communicate without going through any non-government Zones.

Barring the use of Zones, selective third-party traffic is a difficult requirement to satisfy efficiently (using any routing scheme, not just Landmark Routing). In fact, we have chosen not to include it as a feature of Landmark Routing because of its difficulty. None-the-less, we discuss the issues of selective third-party traffic below.

The general solution to handling third-party traffic is to label LUs as to which kinds of traffic the Zone will transit when the LU leaves the Zone. Many criteria can determine the kinds of traffic. Destination address (or Zone, which can be reflected by address) is the most obvious, but source address/destination address pair may be used to distinguish traffic, as well as some type-of-service such as security level. This labeling may be inclusive (lists only that traffic which will be forwarded) or exclusive (lists only that traffic which will not be forwarded). Inclusive labeling is used when the majority of traffic types will not be forwarded, and exclusive labeling is used when the majority of traffic types will be forwarded.

Usually, a Zone will want to determine which traffic it will or will not forward based on Zone membership. In other words, a Zone may not want to forward any traffic originating from Zone x. For this to happen, Zone x clearly must be labeled, because a router or routers outside of the Zone wish to identify it. Routers in the Zone must therefore know which addresses belong to a particular Zone at any given time, so that they can appropriately label the LUs. This complicates the business of selective transit considerably.

First, it means that address binding requests must be made about the Zone in question. If these binding requests are made by the border routers, there will potentially be a significant number of such requests. If the binding requests are made by the Zone-root router, there will less requests, but the Zone-root must 1) have been preconfigured to know which Zones are going to be selectively treated (any router can potentially be the Zone-root, so all Zone routers would need to be preconfigured) and 2) must have a way to get the bindings to the border routers, since they are the

ones which will label the outgoing LUs. These bindings and labeling can be done, but add considerable complexity and overhead to the routing process.

A router which receives a labeled LU must pass it on, as well as pass on additional LUs that are labeled differently. For instance, assume a router receives an LU for $LM_i[a]$ with a label stating that no traffic for a certain child of $LM_i[a]$, say $LM_{i-1}[b]$, will be forwarded. That router must pass on that LU, plus any subsequent LU for $LM_i[a]$ that does allow transit traffic for $LM_{i-1}[b]$.

This is inefficient for two reasons. First, two LUs are being forwarded for the same Landmark. In general, every set of addresses which is excluded in one LU will require an additional LU that includes it. Of course, two addresses that are excluded in one LU can both be included in the same LU. Second, information is being sent (and stored) about Landmarks that otherwise would not need to be sent. For instance, in the above example, the information about $LM_{i-1}[b]$ would not typically travel as far as the information about its parent, $LM_i[a]$. However, information about $LM_{i-1}[b]$ has been added to the $LM_i[a]$ update, resulting in some loss of the efficiency gained by using the Landmark Hierarchy in the first place.

Given the additional complexity and overhead associated with selective third-party transiting using LU labeling, it does not appear beneficial to include this capability in Landmark Routing. The use of Zones will need to suffice.

5.4 Restricted Flow of Landmark Updates Across Zone Borders

There are two reasons for not passing LUs across Zone borders. The first reason is efficiency. If routers near the Zone border can only configure with routers within the Zone, then there may end up being a higher concentration of Landmarks at the border. This is because an LM_i may exist right over the border which would otherwise be a valid parent for an LM_{i-1}, but because it is across the border, the LM_{i-1} must promote itself to an LM_i to satisfy its configuration requirements. Routers in a Zone are therefore listening to and processing LUs from the other Zone which they cannot use as parents. Since we assume there is less traffic across Zones than within a Zone, the inter-Zone LUs are not as useful as intra-Zone LUs for providing routing information. In the extreme, if a Landmark in another Zone is malfunctioning by sending out extraneous LUs or incorrect LUs, it is desirable to ignore that Landmark. A Zone may therefore not allow LUs to enter its Zone until it is necessary.

The second reason is privacy or security. A Zone may not want knowledge about the size or

topology of itself to leak outside of the Zone via LUs. The border routers may therefore not allow LUs to leave its boundaries until it is necessary.

Let us first consider preventing external LUs from entering a Zone. Assume that *Zp* is the Zone that wishes to prevent as many LUs as possible from entering its Zone and that *Zn* is the neighboring Zone. Now assume that *Zp* is smaller than *Zn*, so that the proper configuration would have the *Zp* Zone-root Landmark $LM_i[Zp_{root}]$ joining the *Zn* hierarchy. Clearly, the $LM_i[Zp_{root}]$ needs to see the LM_{i+1} from *Zn* so that it can get a parent. Therefore, the *Zp* border routers should ignore any LUs from *Zn* which are at level *i* or below, but must pass LUs at level *i*+1 and above.

Now assume that *Zp* is larger than *Zn*, so that the *Zn* Zone-root $LM_i[Zn_{root}]$ will configure to one of the *Zp* LM_{i+1}. In this case, the *Zp* border routers must allow the LU from $LM_i[Zn_{root}]$ to enter Zone *Zp* so that routers in *Zp* can route messages to routers in *Zn*. The rule that border routers must follow, then, is as follows:

> If LUs from the neighboring Zone are at a higher level than the Zone-root, let them enter. If LUs from the neighboring Zone are at a lower level than the Zone-root, then only let LUs from the neighboring Zone-root enter.

Clearly, LUs must be labeled as to whether they are Zone-root LUs or not.

The same logic used above applies to the situation where a Zone wishes to prevent LUs from leaving the Zone to the extent possible. In this case, the rules are as follows:

> If any LUs are at a higher level than those of the neighboring level *i* Zone-root, then at least one LU of level *i*+1 which has sufficient radius to reach the neighboring Zone-root, or at most all LUs at level *i*+1 or greater, must be sent to the neighboring Zone. If the LUs from the neighboring Zone are higher that the Zone-root, then the Zone-root LU must be allowed to pass to the neighboring Zone.

If two neighboring Zones are preventing LUs from leaving the Zone, they must at least send their Zone-root LUs across so that they can configure to each other. After the smaller Zone has sent its Zone-root LU, the larger Zone will know what level of LU to send back. Since both of the

above sets of rules require that at least the smaller Zone's LU must pass through the Zone boundary, this is not a problem.

5.5 Zone Partitions

Having introduced Zones to the Landmark Hierarchy, we have also introduced the possibility of Zone partitions. The Zone partition is not the same as an area partition in the area hierarchy, and is easy to handle. The type of partition we are talking about here is where the Zone members are not longer directly connected, but can reach each other through non-Zone routers. In an area partition, routers in different partitions do not know how to route to each other until some partition handling mechanism is executed. In the Landmark Hierarchy, a Zone partition does not cause the same situation because the Landmark Hierarchy adjusts to whatever caused the partition, and routing can still take place. When a Zone partitions, it is basically treated as two or more Zones (depending on the number of partitions).

Let us consider several cases. First, assume that the Zones have not been labeled. In this case, the only way two routers can know that they are in the same Zone is because they have received LUs from each other or from a common ancestor that has not crossed a Zone border. If the Zone partitions, then routers in different part of the Zone have no way of knowing that they were formerly in the same Zone. They would then treat each other as they would any router in a different Zone. However, this is what they would normally want to do, since they are now going through non-Zone routers to communicate with each other.

Second, consider the case where Zones are labeled, and LUs reflect Zone membership. If, after the partition, the Zone partitions are close enough to each other that they receive each other's LUs, then they may detect that they have become partitioned because although the Zone IDs are the same, the Zone-root routers are different. In this case, they can at least decide if they are willing to send traffic to each other or not, based on their policy of routing through non-Zone routers. If they do not receive each others LUs, then the situation is the same as if the Zones were not labeled.

5.6 Routing Autonomy

We have already established that one can give routers in Zones a certain amount of autonomy from routers in other Zones by confining intra-Zone traffic to the Zone and by preventing the spread of certain LUs. We want to explore the amount of autonomy which can be achieved with regards to the routing protocol used within a Zone.

The reasons for wanting routing autonomy between Zones are apparent. The style of routing says a great deal about what kind of overhead will be levied on network resources and largely decides what kind of service is offered to users. At the low-capability end of the scale are single static metrics based on either hops or some link capacity. In this case, new routing updates are required only when a link or router goes up or down. At the high-capability end are such features as delay-based metrics (which require many periodic routing updates), multiple metrics, and path splitting.

5.6.1 Two Approaches to Routing Autonomy

Look at Figure 15. Assume that *Z3* wishes to use a different style of routing than *Z2* (for instance, *Z3* is doing delay-based routing, while *Z2* is using static link metrics). There are two ways that this can be accomplished.

In the first way, *Z3* routers could participate in both the *Z2* and the *Z3* routing styles, while *Z2* routers only participate in *Z2*-style routing. Routing Updates (RUs) from *Z3* routers would need to be generated for both the *Z3*-style and the *Z2*-style routing if the *Z2*-style RUs are allowed to leave *Z3*. (Note that there would be far fewer *Z2*-style RUs than *Z3*-style RUs, since the *Z3*-style metric is static.) If the *Z3*-generated *Z2*-style RUs are not allowed to leave *Z3*, then only the *Z3* Zone-root RUs need to be carried in *Z2*-style RUs. Note also that when *Z3* routers receive RUs from *Z2* routers, they must pass them inward in the style of *Z2* RUs. The advantage of this method is simplicity: the various routing styles are kept separate and run in parallel. The disadvantage is that it incurs some extra overhead, especially for those Zones which are multiply nested.

The second way is that *Z3* routers could participate in just *Z3*-style routing, and make a metric translation between *Z2*-style routing and *Z3*-style routing when RUs cross the border. This, however, requires a coupling between different routing metrics and styles on a pairwise basis. For instance, in Figure 15, *Z2* routers would need to understand *Z4*'s routing style in order to make the appropriate metric change. These translated RUs can lose meaning and become complex as the RUs pass through several Zones. Because of this interdependency, we reject this method of providing routing autonomy between Zones.

5.6.2 Using Landmark Updates as the Common Routing Style

Referring to Figure 15 again, we see that *Z3* routers must also participate in the *Z2* routing technique. Similarly, *Z2*, *Z3*, and *Z4* must all share the routing technique of *Z1*. Ultimately, for any Landmark Hierarchy, there must be some routing style that all routers in all Zones recognize.

(This is not true for Zones which are stub Zones, do not run Landmark Routing, do not forward third-party traffic, and are front-ended by border routers which speak Landmark Routing on behalf of the Zone.) Note, however, that LUs can also be used for routing. They provide a static metric based on hops only. Since all routers in a Landmark Hierarchy must participate in the hierarchy maintenance, this is an appropriate default routing style to use.

Any routing techniques used in addition to the LUs must be operated in a completely decoupled fashion. In other words, RUs should not be placed on top of LUs, and RUs should not be used to give hierarchy maintenance information even though they may be capable of doing so (such as indicating that a Landmark is no longer reachable). This decoupling allows different routing techniques to be plugged into and taken out of Zones without regard to the hierarchy maintenance processes.

Note that the Landmark Hierarchy maintenance information does feed into the routing techniques in that the Landmarks and their radii determine which routers RUs are sent about and how far those RUs travel. Basically, RUs travel exactly as far as LUs. When a router decides not to pass on an LU (either because the radius has expired, or because of a Zone boundary), the router indicates in a table that RUs for that Landmark are not to be passed over the appropriate links. When an RU is received, therefore, the router checks that table before passing the RU on. By maintaining this table, it is not necessary to carry any Zone information in the RUs themselves.

6 ADDRESS BINDING IN LANDMARK ROUTING

Clearly one of the difficulties of Landmark Routing is that Landmark Addresses can change at any time, even though a node's point of attachment to the network does not change. This requires that all nodes in the network be identified by something other than their Landmark Address and that it is possible to determine the current Address of a node from that identification (referred to as an ID or as a name).

In a previous document, we outlined a technique, called Assured Destination Binding (ADB), for determining the address of a node given its ID (Stine and Tsuchiya, 1987). That document presented a general treatment of ADB—it did not presuppose the environment in which ADB might function. In this section, we specify the operation of ADB in the Landmark Routing environment.

6.1 What Is It We Are Binding?

We are binding two objects, an address and an ID. In particular, we are assuming that 1) the ID is a known and stable object, 2) it is the input to the binding function, and 3) the address is the result of the binding function. The address we are referring to here is clearly the Landmark Address—the changing object that says where in the network a node is. The Landmark Address goes into the address field of the header of the network layer packet, and is used by routers to make routing decisions. The Landmark Address may be the only component of the address field, or may be one of several components such as an ID or a Zone ID.

The exact form of the ID which is input into the binding function is less clear and will depend on the environment in which Landmark Routing is being used. We can state, however, that the ID should be a *network layer ID*. That is, it should uniquely identify a network layer entity.

For instance, in the DoD Internet, a network layer entity is identified by the Internet Address. It specifically identifies a host's interface to the Internet, but more generally identifies the host itself. In OSI, at least for our purposes, a network layer entity is identified by the Network Service Access Point (NSAP). Although OSI abstractly defines what the NSAP is, I believe that practically speaking, it also generally identifies a host (for instance, NSAP Addresses (NSAPA) minus the NSAP Selector Field are typically defined on a per host basis).

We do not want to constrain our discussion of ADB to any particular naming or addressing convention. For the rest of this Section, unless otherwise stated, we will assume that the input to the ADB function is a string of bits of arbitrary length and that the output is a Landmark Address.

6.1.1 Architectural Considerations About Hosts

In the architecture put forth in this paper, network-layer routing and addressing functions are hidden from the hosts by the routers. The hosts see a "flat" address space: they address each other through the use of IDs, not Landmark Addresses. This model has routers executing the binding function on behalf of, and transparently to, the hosts.

This is an outgrowth of the principle that hosts should not be concerned about routing. For instance, hosts normally do not have to determine the path that a packet should take and generally do not care about the topology of the network. In Landmark Routing, we go one step further and say that hosts should not care even about their "addresses"—that is, their location in the network with respect to the rest of the topology. In Landmark Routing, routers support the binding function just as they support the routing function.

This being said, it may be desirable for certain hosts to store their own bindings. First, hosts often have more memory than routers that can be dedicated to storing bindings. Second, it is more fair. If a host happens to communicate with a large community of other hosts, why should the router need to keep track of that community?

When a host is sophisticated enough to keep its own bindings, then we model the host as an extension of the router. The host and router run a protocol whereby the host tells the router what its capacity for storing bindings is, so that the router can act accordingly (as explained in Section 6.2). The router must tell the host when it has a new address.

In the balance of this Section, for the sake of convenience, we will assume that the router is doing all of the binding function. If indeed hosts are participating, we will simply consider them as extensions to the routers.

6.2 Design of Assured Destination Binding in Landmark Routing

These sections first describe ADB as it is used in Landmark Routing and then further develop various aspects of ADB.

6.2.1 Assured Destination Binding in Landmark Routing: Basic Concepts

The fundamental problem in any address binding scheme is locating the node is holding the binding—that is, since the address of the desired destination is not known, then at least the address of a node that does know must be known. In the DARPA Domain Name system, the node (the name server) that knows the address of a given destination, or at least where to search for it, is

embedded in the name of that destination. So, when a destination address is desired, the source, which has some way of knowing the name of the destination (for instance, the user types it into his terminal), looks into that name, finds the name of a name server which can help find the address, looks into a table to find the address of the name server, and then sends a query to the name server. The name server, if it doesn't have the binding itself, looks at the destination's name and finds the name of yet another server (closer to the destination), which may have the binding. This search repeats until the binding is found.

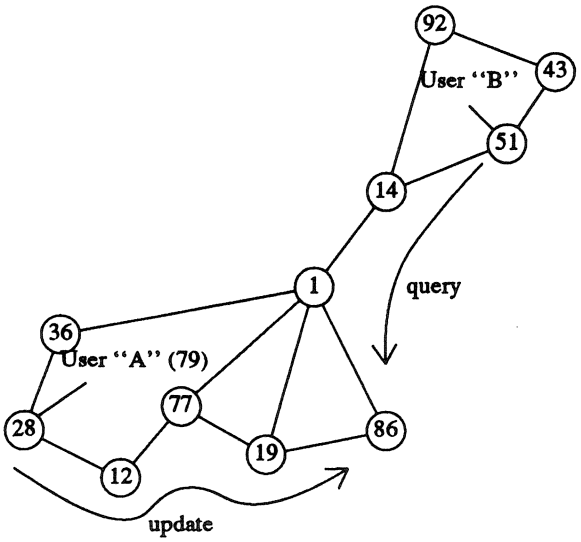
This kind of binding technique will not work well in Landmark Routing because the address of the name server itself may change, making it difficult for any router to keep a list of name server addresses. This name server technique is clumsy for several other reasons. First, it requires a certain amount of semantics in the name to identify the appropriate name servers. This creates an administrative burden, and results in a name change every time a destination associates itself with a new name server. Second, it often requires a series of queries to find the address binding, first to a high-level server, then to a lower-level server, and so on. This is inefficient and results in delays. Third, a router can become partitioned from other routers simply because its name server (or servers, if there is redundancy) has gone down.

In ADB, we use a different approach to find the address of the appropriate name server. Instead of putting the name of the server in the host name, and then using a table lookup to find the address of the server, we derive the address of the server directly through an algorithmic manipulation on the host name—namely, a hash function. This hash function maps the name directly into the Landmark Address space (this is not quite literally true, but is conceptually accurate), thus unambiguously pointing to the address of a server. The name itself requires no semantics (such as an embedded name server) other than the address semantics which is vicariously derived through the hash function.

The obvious problem with this is that the address found by the hash function can be any arbitrary Landmark Address. It is highly probable that the actual address will not exist anywhere in the network, since the address space will be sparsely populated. To solve this problem, we rely on a simple resolution function which maps the hashed address into some real address. For instance, this resolution can be a simple series of increments to the hashed address until it matches a real address, wrapping around to the lowest number when necessary. As long as this resolution consistently maps to the same real address from anywhere in the network, the hash function followed by resolution is all that is needed to identify a server for any given named network entity.

To illustrate this, see Figure 18. Here we see a network with 11 routers whose addresses are taken out of the address space 0-99. (Note that this is a flat routing architecture. We consider how to do this in a hierarchy in Section 6.4.1.) We also see two users, *A* and *B*, and assume that User *B* wants to learn the address of User *A*. Assume also that the ID of User *A* is "*A*".

Figure 17
Assured Destination Binding: Hashing and Resolution



Hash ("A") → "79"

Resolve ("79") → "86"

First *A* must establish who its server is. To do this, *A* executes a hash function on its name "*A*", resulting in this example in the value *79*. User *A* then hands a binding update addressed to Router *79* to the network for resolution and delivery. Router *28* receives this message, looks in its routing table and does not find a Router *79*. Therefore, Router *28* resolves *79* to a real address by incrementing it until it matches an entry in its routing table—namely Router *86*. Router *86* then receives the binding update and stores the binding. Later, when User *B* needs the address of User

A, it also hashes "*A*", and this time Router 51 resolves the resulting address to Router *86*, the query is delivered, and User *B* receives the binding.

The reader should note that, in its most general sense, ADB allows the server for any given node to be anywhere in the network. This contradicts to the widely accepted idea that a server should be as close to the node it is serving as possible. One reason for this idea is that, since nodes that wish to communicate are normally close to each other, they should not have to go far to learn each other's addresses.

We agree with this notion, but point out that the closer a server is to the node it is serving, the more difficult the search for that server becomes. The name-domain hierarchy and naming convention exemplifies this difficulty. In Landmark Routing, we allow servers to be nearby through the use of Zones. This allows updates and queries between Zone members to stay within the Zone, thus providing efficiency and self-sufficiency. However, it does complicate the process of querying for nodes outside of ones Zone. This is discussed in Section 6.5. We mention it here to assure the reader that it is possible, since most of the following discussion is outside the context of Zones.

6.2.2 Assured Destination Binding in Landmark Routing: Development

Clearly, a bit of engineering is required to bring the simple idea of ADB described above to practice. In the following sections, we raise several problems and present solutions.

6.2.2.1 Resolution in a Landmark Hierarchy. In the example of Figure 18, resolution takes place in a flat routing space—that is, every router knows of every other. In the Landmark Hierarchy, this is not the case. Each router has its own partial view of the world. Clearly, if each router tried to completely resolve a hashed Address based on its view of the world, different routers would resolve hashed Addresses to different real Addresses, and the binding would not take place. Therefore, the resolution needs to occur one hierarchy level at a time, starting at the global level.

Notice that every router in the Landmark Hierarchy sees the same set of global Landmarks. Therefore, if every router initially resolves the hashed Addresses only to the collection of global Landmarks, the resolution will always be the same. Now, assume that the offspring of each global Landmark (those routers which contain that global Landmark in their Addresses) knows about all the children of that global Landmark. Any offspring of the global Landmark can therefore resolve the hashed Address to one of the children of the global Landmark. Likewise, each offspring of that

child can further resolve the hashed address to one of its children, and so on until the hashed address is resolved to a single router.

It is a simple matter to let the offspring of a Landmark know of that Landmark's children. The Landmark simply lists its children in its own LUs. The offspring of that Landmark then store that list. (Note that the offspring of a Landmark do not necessarily receive LUs from all of that Landmark's children.)

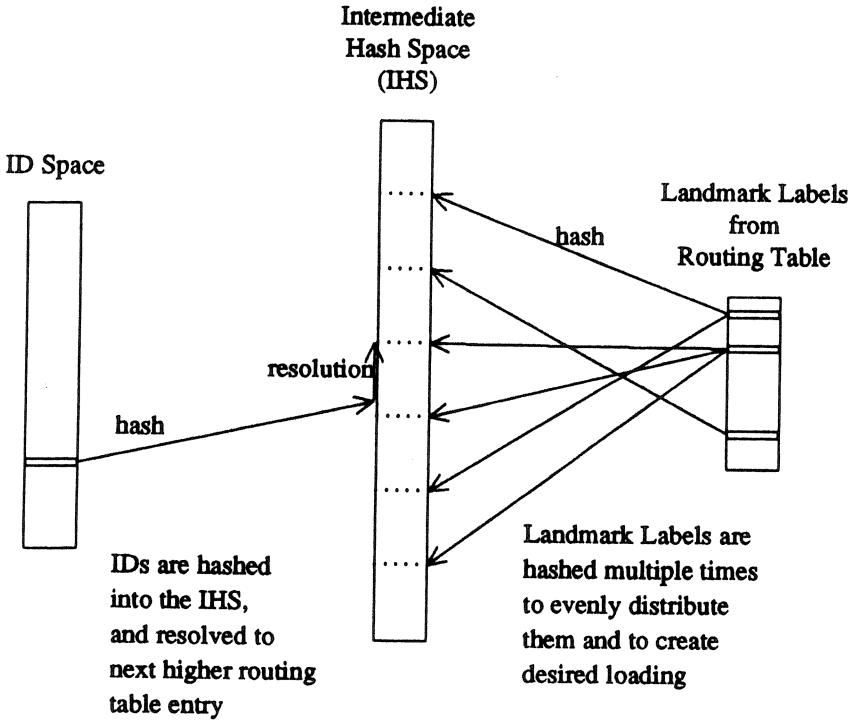
We can now state the method of hashed address resolution in the Landmark Hierarchy. Assume that the binding message holds the ID (or name) that will get hashed. Assume also that each Landmark Label has a value which means "needs resolution" or simply "null". The first router to receive the binding packet hashes the ID, resolves the global Landmark Label, fills it in with the proper value, and sends it towards the chosen global Landmark. As soon as one of the offspring of that global Landmark receives the packet, it hashes the ID again, resolves the next lower Landmark Label, fills it in, and sends it on. Note that the global Landmark itself did not necessarily need to do this hash and resolution. This continues until the full Landmark Address is resolved. When the final Landmark receives the binding message (either update or query), it responds appropriately.

6.2.2.2 Even Distribution of Bindings. There are two reasons for hashing the ID to obtain an Address (or Label). The first is that the hash gets the ID from its space or syntax into that of the Label. The far more important reason, however, is to distribute the hashed IDs evenly over the full Label space. Even this, however, is not fully adequate to evenly distribute the bindings among the routers in the network.

First, the Landmarks themselves may not be evenly distributed across their respective spaces. What if global Landmarks end up with their global Landmark values being clumped rather than evenly distributed? For instance, let us assume that the values for global Landmarks range from 1 to 255, but that for some reason all the global Landmarks have ended up with values greater than 150. Then all the IDs that hash to values between 1 and 150 will resolve to the same global Landmark—namely, $LM^G[150]$. The offspring of $LM^G[150]$ will receive a disproportionate number of binding updates and requests.

The full solution is to hash the routing table entries for each level into yet another, larger space, in order to evenly distribute them. We call this the Intermediate Hash Space (see Figure 19). One reason that this space should be larger than the original Label space is that since the Label

Figure 18
Intermediate Hash Space



space is so small in the first place, (just 5 or 6 values in the case of the hierarchy levels below the global level), it is impossible to get much of an even distribution in that small space. For instance, if there was a space of 6 slots for some level (1-6), and slots 1-5 are taken up with Landmarks (which is as even a spacing one can get with only 5 values into 6 slots), then the Landmark with value 1 will get twice as many resolutions as the others. By hashing multiple times into a larger space, we can get a more even distribution.

Another reason this space should be larger is to avoid collisions when the addresses are hashed into the Intermediate Address Space. When the hashed addresses collide, the smaller (or larger) of the colliding addresses must be thrown out. Naturally, we would like to minimize the possibility of throwing out hashed addresses. Note that we cannot just keep the colliding address by incrementing its location in the Intermediate Hash Space until it finds an empty location. This is

because the order in which routing table entries were received would then determine where hashed addresses ended up, thus causing an inconsistency between routers.

Note that the simple adoption of a large Intermediate Hash Space does not alone ensure an even distribution of addresses. If there are only two or three Landmarks at a given level, and each is hashed into the same large space, it is probable that two of them will end up disproportionately close together. The solution, then, is to hash each of the addresses several times, thus increasing the evenness of the distribution. The use of modified hash keys (for instance, a byte appended to the address itself, whose value was incremented for each hash) would cause different hash results for each hash.

Note that the IDs must be hashed into the Intermediate Hash Space rather than directly into the Label space. This hash is resolved to the first real value in the Intermediate Hash Space, which, in turn, points back to a real Label value.

6.2.2.2.1 Why Not Deterministically Distribute Addresses Into the Intermediate Hash Space. The reader may be wondering why we do not just deterministically distribute the addresses evenly into the Intermediate Hash Space rather than depend on the pseudo-randomness of the hash function. After all, at any given time, all of the addresses to be distributed are known, making it easy to evenly distribute them across the Intermediate Hash Space. The reason we do not do this is that if the addresses are deterministically distributed, then the addition or deletion of a single address will cause all of the locations in the Intermediate Hash Space to shift in order to reestablish an even distribution. This shift will cause a large percentage (on the order of 50%) of the bindings to be in new locations. In other words, the hash of an ID into the Intermediate Hash Space will probably no longer resolve to the same address, and until another update binding is sent, binding cannot take place.

If, on the other hand, the addresses are hashed into the Intermediate Hash Space, the addition or deletion of an address only changes the binding location for those IDs which hashed directly below the added or deleted address.

6.2.2.3 Controlled Distribution of Bindings. For several reasons, a simple even distribution of bindings over all routers is not necessarily desirable. It may be desirable that certain routers not hold any bindings, or that some hold more than others, depending on their processing and communications capacity. This is especially true if hosts are holding the bindings in addition to the routers themselves.

It is a simple matter to accommodate differing binding server capacities. First, it is necessary to maintain information about a given router's capacity to be a binding server in the LUs. This information is then used to determine how many times a particular Label is hashed into the Intermediate Hash Space. If the router at a given Address (or the hosts attached to that router) have a high capacity for holding bindings, then that location is hashed more times into the Intermediate Hash Space. The result is that IDs hashed into that space will have a higher probability of resolving to the high capacity router.

For the hashes at the higher levels of the hierarchy, it will be necessary to accumulate the total amount of binding server capacity which exists under a Landmark. This is done in the same way as collecting routing table information (see Section 4.3.2.1.1). Level 0 Landmarks include in their LUs their capacity for holding bindings. Level 1 Landmarks add the capacities for their children, and put the sum in their LUs. This continues up the hierarchy to the global Landmarks, which include the total binding capacity for their offspring in their LUs.

As this information percolates up the hierarchy, it must be aggregated in increasingly gross units of measure. This is because we don't want a small change in router membership below a global Landmark to result in a change in the binding capacity that the global Landmark advertises. If it did, the global Landmark would need to send out a new LU every time a single router was created or destroyed, so that all routers could adjust their Intermediate Hash Space entries. If the unit of measure up the hierarchy is gross, then only significant changes in router membership would cause a new binding capacity message. A resolution of 4 or 5 is probably more than adequate at any level of the hierarchy.

6.2.2.4 The Problem of Popular Destinations. Another problem which creates an uneven distribution of binding workload at the servers is that of popular destinations. If a node is one which communicates with a large community of other hosts, like a name server, then there will be a proportionately larger number of requests for the Address binding for that node. The server for the popular destination will be deluged with queries, resulting in unfair distribution of binding workload and a network bottleneck.

The solution to this problem is for the popular destinations to distribute their binding updates to more than one binding server, again by hashing their IDs. When a router needs to query for the Address binding, it either randomly picks one of the several hashes and sends the query there, or it calculates some or all of the hashes and picks the closest one.

The problem here is that the querying router needs some way to know how many places the destination has put its current Address binding. If the destination has put its binding in 10 places and the querying router thinks it has been put in 20 places, it is likely to query servers which never got the update. On the other hand, if the querying router thinks the binding has been put in fewer places than it really has, there will still be an unfair distribution of binding workload.

The nature of the binding problem is that the initiator of communications has a limited amount of knowledge about the destination—in our case, only the ID of the destination. Therefore, the only way for the initiator to know how many places the destination has put its binding is for that information to be included in or with the ID of the destination. This has the unhappy consequence of adding a constraint to the makeup or distribution of the ID. Also, the number of binding holders for a given destination may change over time, as that destination's popularity changes.

However, a small resolution granularity (3 or 4 at most) should be enough to convey this information. In this case, it will often not be very difficult to guess which popularity level any given destination will be. One can always fudge one's guess on the more-popular side and refine the search if the binding request turns up nothing. With a granularity of only 3 or 4, this search would not require very many requests.

6.2.2.5 Efficiency and Robustness With a Single Stone. Clearly there is a robustness problem in keeping a binding at only a single Address server. If the server goes down, there will be a period of time after the server goes down and before a new server is updated during which the Address binding cannot be found. Therefore, the binding should be kept at several servers simultaneously, even for the least popular destinations. This increases the probability that there is at least one place that he binding can be found.

Fortunately, this also usually makes binding more efficient. The reason for this is that there will usually be more queries than updates. If there are several servers spread around the network more-or-less at random, then the querying router can always pick the closest one. This saves on link resources (because the query travels over fewer links to reach the server) at the expense of processing resources at the source router (to run multiple hashes on the ID in order to find the closest server). If processing resources are scarce compared to link resources, then the querying router needs to run only one hash.

The exact optimal number of servers for each binding will have to be determined case by case. However, 3 or 4 seems to be appropriate. For instance, assume that the number of routers x

hops away from any given router is uniform—in other words, if there are y routers one hop away from every router, then there are also y routers two hops away, y routers three hops away, and so on. (This does not really happen, but will suffice for this example.) Then, if an Address binding is stored at only one location, binding queries will on the average traverse half the diameter ($0.5D$) of the network to reach the server. If there are two servers, then binding queries will traverse on the average $0.33D$ links. If there are three servers, $0.25D$ links; if there are four, $0.2D$ links; and if there are five, $0.17D$ links. Clearly after four servers, we are reaching a point of diminishing returns.

A similar example can show that 3 or 4 servers also is an appropriate number for achieving good robustness.

6.2.2.6 Handling Server Changes. This section tells how to handle changes that cause a binding for a particular ID to resolve to a new server. There are three cases:

1. A server obtains a new Address, thus causing the hashed ID to no longer resolve to that server.
2. The addition of a Landmark somewhere in the hierarchy above the server causes the hashed ID that previously resolved to that server to resolve elsewhere. (This includes the case where a global Landmark advertises a new server capacity.)
3. A server crashes, thus losing the binding.

The first two cases can be handled on an event-driven basis. In other words, the change can be detected when it occurs. In the first case, the server simply informs the router which originated the binding that it needs to re-update the binding. Note that is possible for the server to re-update the binding it was holding on behalf of the destination router. To do this, the server needs to know which hash (the first hash, second hash, etc.) resolved to it. Since the server knows the ID of the destination router, it could recreate the binding message and send it to the new server. The problem here is that authentication techniques could not be used to verify the originator of the binding update.

In the second case, a server can tell if a change to the Intermediate Hash Space affects any of the bindings it is holding. We first consider the addition of an addition to the Intermediate Hash Space. For instance, assume that the level 0 Label of Server S hashes into the Intermediate Hash Space at location 100. Assume also that the next lower entry in the Intermediate Hash Space is at location 90. All IDs which hash to locations 91 through 100 will therefore be stored at Server S.

If, for instance, a new Landmark results in an entry at location 96, then all of the IDs which hashed to locations 91 through 96 will now resolve to that new Landmark, and will no longer be stored at Server *S*.

Therefore, whenever creating a new Landmark causes a change in the Intermediate Hash Space at a location between that of the server and the entry lower than the server, the server must determine whether any of the bindings it is holding are affected by the new entry. It can do this either by rehashing the IDs of its entries or by storing the hash values of the entries when it receives them (this is a processor vs. memory tradeoff). If the new Landmark did result in a change of servers, the old server again informs the router originating the binding of the new change so the originator can send a new update.

Note that a deletion from the Intermediate Hash Space (provided that it did not result from a change in the server's Address) will not affect any of the bindings currently held by that server. However, it will cause more bindings to be directed towards that server. A deletion from the Intermediate Hash Space means either that some set of routers received new Addresses, or that some router crashed. In the first case, the routers are going to send messages to their clients indicating that a new binding update is required (as discussed under case one). In the second case, periodic updates are required as discussed below. Therefore, a router which sees a deletion from the Intermediate Hash Space does not need to do anything except field new updates and queries.

Also the second case (where a server crashes) the server obviously cannot inform the binding originator that it is no longer holding its binding. There are two approaches to this problem. In the first approach, the binding originator periodically sends the update to its servers. If the binding originator has several servers simultaneously, then the probability that they have all crashed should be small, thus allowing the update period to be large.

The other approach is for the server to tell one or more of its neighbors which nodes it is holding bindings for. Since routers must ping their neighbors as a matter of course (to send LUs and RUs), this information could be biggybacked onto the existing pings. Each router must obviously hold additional information, but this approach saves link bandwidth compared with the periodic update approach. Unfortunately, it is possible for both the server and its alternate to crash at the same time, thus bringing us back to the need for periodic updates. Ultimately, periodic updates are needed to ensure that the condition where no servers are holding the binding for a long period of time is avoided. We therefore reject the second approach mainly because it adds

complexity and memory requirements to the system without eliminating the need for periodic updates.

6.2.2.7 Smooth Address Transitions. Sometimes a router will get a new Address without being able to keep the old one for a while. We want to avoid glitches in existing communications when this occurs. There are two solutions. One is for the router to send update messages to any other routers from which it has recently received data packets. Any subsequent packets sent by the source routers will have the new, correct Address.

There are, however, several major problems with this technique. First, a significant number of updates will go out immediately after an Address change. Since many routers in the same vicinity will often all get new Addresses at once (because a higher-level Landmark gets a new Address), this amounts to a significant traffic surge. Second, if the router sends out too many updates, it is inefficient; but if it does not send enough, some nodes that are still communicating will not be updated. Third, any packets sent after the Address change but before the update is received will go to the wrong router and be dropped.

A better solution is for a router with a new Address to send out an update to its previous Address. This will resolve to some router somewhere (one close by if the Address change was not at the global level), and that router will retain the binding information. Then, when packets are sent into the network by sources that have not learned of the new Address, they will be resolved to the same router that now has the new binding. This router can inform the source of the new Address.

This function requires that routers be prepared to do resolution as part of their network-layer, packet-forwarding function. We point this out because most of the binding functions will be implemented in the application layer (see Section 7.4). However, we feel that this function is important enough to do in the network layer. This technique is more efficient than the first because only one binding update is sent originally, with additional binding updates sent on an event-driven, as-needed-basis—that is, to those nodes that are actively trying to communicate.

6.2.3 Assured Destination Binding in Administrative Zones

Naturally, if a community of routers has established an Administrative Zone for itself, it would like to control bindings so that binding queries made by Zone members can be resolved by other Zone members. This is accomplished simply by limiting the resolution function to only those Landmarks within the Zone. Therefore, when a router sends out binding updates, it must send out updates for each Zone for which the router requires intra-Zone binding. If a querying router does

not know already (either from some semantics in the name or ID, or by some additional information) that the destination router is in its Zone, but still prefers intra-Zone binding over inter-Zone binding; it must send out a series of binding queries (one for each Zone it is in) each time expanding the scope of the query over higher and higher-level Zones. If a router is in several overlapping Zones, it will have several different Addresses, and can hash into the appropriate Addresses depending on which Zone the update is constrained to.

For the sake of privacy or efficiency, a Zone may not care to spread bindings about its individual nodes outside of the Zone. In this case, it may be possible for the Zone-root to send out one binding for the whole Zone. We call this Binding By Zone (BBZ). However, requesting routers will need to know the identifier for the Zone itself, and need to know that the destination they are interested in resides in that Zone. This, of course, places semantics in names. Now, instead of being just Paul_Tsuchiya (for instance), I must be Paul_Tsuchiya#MITRE (where MITRE represents a Zone, not a router holding the binding). This is much like the current way of doing binding: semantics in the name tell where to search for the Address.

In addition, the Zone ID within a data packet (if it is there) could identify the Zone for binding. Depending on the type of ID used (DoD address or ISO address), this could be a network number, or it could be the part of the ISO address that is assigned administratively (AFI, IDI, Organization Identifier, and Zone ID, for instance). The main point is that there will need to be some kind of well-understood convention for knowing which part of the ID to use to identify a Zone so that the hashed ID can be formatted correctly.

The response to the binding request can be one of two things, depending on how we want to operate. First, it can simply be the Address of the Zone-root (possibly with the levels of the Landmark Address that only have significance inside the Zone masked out). In this case, the requesting router would only need to maintain one binding for all nodes in the Zone. When it sends a regular data packet to the Zone, the packet would first reach a Zone border router. This router would then need to make another binding request, this time within the Zone, to find the Landmark Address of the destination. The border router would fill in the rest of the Landmark Address in the data packet, and forward it on. The border router would also cache the binding to use for subsequent data packets.

Second, the requester of the binding may want the full Landmark Address of the destination. In so (after receiving the Zone-root Address), the requester could generate another binding request,

but this time to the Zone itself. The border router would then receive a binding request (not a data packet) and (after making authentication checks if necessary) make a further binding request inside the Zone. The border router would then return the result of this request to the original requester. We believe that this second method is preferable becuase it avoids doing binding requests when data packets arrive. We would like to avoid these three requests because 1) they make the whole forwarding process less efficient and 2) they require an interaction between the network-layer software (which is where the forwarding software is) and the Application software (which is where the binding software is), which weakens the layering.

Note that BBZ can be recursed hierarchically. For instance, an organization (denoted by some Organization ID) may wish to send out one binding for the entire organization. Within the organization, each Zone could send out one binding, etc.

As before, we still have the constraint that all Zone members must be able to communicate without leaving the Zone. If a Zone partitions, then each partitioned segment will be sending out the same binding update, because each segment will think that it is the whole Zone (assuming that the Zone labels don't change because of the partition). In this case, several different bindings will converge on the same servers, therefore allowing a querying router to detect that the partition has occurred. The querying router must then send out multiple binding requests, one for each Zone partition, in order to find the destination. Zone partitions, therefore, are not difficult to deal with operationally. They simply cause inefficiencies.

6.2.3.1 Assured Destination Binding and Higher Level Name Servers. We do not believe that ADB, even with the BBZ modification, can or should take the place of higher-level, directory-style name servers such as the Name Domain System. One reason for this is because yellow-pages types of queries are not efficient using ADB. Another is because queries where only part of the name is known, or where the syntax of the name is incorrect (such as the spelling) cannot be done using ADB.

Both of these are hard because the syntax of the name or ID must be exactly specified for ADB to work. This is straightforward if one uses network layer identifiers (such as a DoD Internet address) as the name, but harder if one uses a character string name, and harder still if one uses a set of attributes (such as printer with font xyz).

ADB is most efficient when the lowest possible level (network layer) identifier is used as the

input to the ADB process. This assumes, then, that some higher-level name service was used to determine the appropriate network layer identifier.

The other problem with trying to identify higher-layer objects (mailboxes, individuals names, processes, files, and so forth) using pure ADB is that there is a multitude of higher-layer objects that must be updated for the method of ADB to work. Assuming that any given computer may have several hundreds of objects that could potentially be identified and searched for, it is unrealistic that binding updates for each of these objects be sent into the network. (Over a single LAN for instance, it may be feasible to bind objects in a distributed operating system, if the bindings are confined to the local system.)

Again, the most reasonable approach is to use ADB only to bind to network layer identifiers, and use a higher-layer name-service for the more general-purpose binding.

7 ARCHITECTURAL AND ENVIRONMENTAL CONSIDERATIONS

This section collects issues and ideas that do not easily fit into any of the previous sections. First, we discuss addressing considerations. We then tie all of the previous sections together by describing the framework of an implementation of Landmark Routing. Third, we consider the problem of auto-configuring over a large non-broadcast subnetwork like the ARPANET, and offer a solution. Fourth, we consider the appropriate layer (network or application) for the various parts of Landmark Routing. We then consider the role of hosts and routers in Landmark Routing. We discuss how Landmark Routing fits into the global ISO routing architecture. Finally, we discuss deployment and transition issues.

7.1 Addressing

Sections 2 and 4 state what a Landmark Address is and how it comes to be. This section we discusses what we call the Network Layer Address, of which the Landmark Address is one part.

The purpose of the Network Layer Address is to identify the network layer entity of a network. (When we are being less precise, we will use the vernacular term *node* instead of *network layer entity*.) This is true of both the DoD Internet Address, and the OSI Network Service Access Point (NSAP) Address (NSAPA). Usually, there is embedded in the Network Layer Address some information which hints at *where* in the network the network layer entity is. This is the role of the Landmark Address.

The Network Layer Address is in the header of the network layer packet, and routers use the Network Layer Address to make routing decisions as that packet travels through the network. By network layer, we mean that layer where the routing function takes place. The sponsor of this paper is primarily interested in the internetwork layer of the DoD internet (the Internet Protocol, MIL-STD-1777), or of OSI (Open Systems Interconnection) (ISO 8473, "Protocol for Providing the Connectionless-mode Network Service"). However, other environments where the routing function is executed on headers other than these two are certainly valid. These include packet radio networks, special purpose (non-standard, vendor-specific) networks such as the ARPANET, X.25 networks, or even circuit switched networks (where there may not be a header per se, but some kind of out-of-band signalling instead).

7.1.1 Components of the Network Layer Address

The primary component of the Network Layer Address, as far as routers are concerned, is the Landmark Address. (The existence of Landmark Addresses is normally hidden from hosts, and so for them, the ID is the primary component of the Network Layer Address.) The Landmark Address is a series of Landmarks, one at each level of the hierarchy, which denotes the location of the router in the network. This component is necessary for routing in a Landmark Hierarchy. It also fulfills the role fo the Network Layer Address by uniquely identifying a network layer entity (at least within the Landmark Hierarchy)

There are two other components of interest that may exist in the Network Layer Address field of the packet's network layer header. One is the ID, which uniquely identifies the network layer entity and does not change even though the Landmark Address may change. The other is the Zone ID, which identifies which Administrative Zone the network layer entity is in. The semantics of these two components may be at least partially conveyed in the same part of the Network Layer Address; that is, the same bits may contribute to both the ID and the Zone ID. The existence of these two components will depend in large part on the size of the Network Layer Address. (In OSI, there will be strictly speaking be a fourth component, the NSAP Selector. This information is conveyed in the Protocol field of DoD IP. We will not consider that part of the address further.)

7.1.2 The Landmark Address

As stated above, the Landmark Address is nothing more than a series of Landmarks, one at each level up to the global level, which are the ancestors of a router. One of the goals in the design of the structure of the Landmark Address is that it be as compact as possible. In section 4.3.2.2.4 it is shown that an appropriate number of children for any Landmark should vary from 2 to 5. Therefore, 3 bits is adequate to identify each child of any Landmark, with room left over for 3 additional values. Possible uses for the additional values are "don't care" (for instance, when a Landmark Address has not yet been fully resolved during the binding process), "broadcast", and "escape value" (meaning that the rest of the Landmark Address should be interpreted differently).

At the global level, all Landmarks must have a different global Label. It is not known in advance how may global Landmarks there may be for any given network. Figure 14 indicates that for a 10000 router network, roughly 200 global Landmarks is appropriate. A global Landmark address space of 12 bits allows for 4096 global Landmarks. Assuming that half of the routing table consists of global Landmarks, then 4096 global Landmark implies roughly 8000 routing table

entries If we ignore the global Landmarks, then using $R = 4000$ in Figure 12 implies over one-and-a-half million switching routers. At the expense of making predictions about the size of future networks (since most previous guesses seem to have fallen short), it seems that single routing structures of one-and-a-half million switching routers should last us awhile.

Let us consider the address space available in the DoD Internet Address—32 bits. Assume also that a Class E address is assigned to Landmark Addresses. This allows 27 bits for the Landmark Address. Because of the cramped address space, let us assume that only 9 of these are used for the global level, and the other 18 provide 6 levels of 3 bits each for the hierarchy levels below the global level, for a total of 7 hierarchy levels. Assuming that each Landmark has on the average 3 children, this gives 729 routers per global Landmark. If we have 500 global Landmarks, then we get a total of roughly 350,000 routers. If we assume an average of 4 children, then each global Landmark can have 4096 offspring, for a total of roughly 2,000,000 routers.

However, a problem remains here in that this address is only sufficient to identify a router. It is still necessary to identify the hosts which are directly reachable via that router. One solution is encapsulation of the hosts normal IP address in an IP packet containing the Landmark Address. Another solution is to create a new IP option, that would hold the Landmark Address. This latter solution is preferable to encapsulation because it is more efficient, and because the Landmark Address could be longer—say 4 bytes.

A third solution (not recommended) is to encode the identity of the host (with respect to a router) in the 32-bit address. Such an encoding might provide 9 bits for the global Landmarks, 6 bits for the host identifier, and four intermediate levels of 3 bits each. Again assuming an average of 3 children per Landmark, we have 81 routers per global Landmark, and 40,000 routers given 500 global Landmarks. Assuming 4 children per Landmark, we get 120,000 routers. Each Landmark could accommodate a maximum of 60 or so hosts. If a particular router had more than 60 hosts attached to it (i.e., through a LAN), it would have to spawn level 0 Landmarks. 40,000 to 120,000 routers seems to be cutting it pretty close in terms of handling Internet growth even over the short term, say 5 years. Further, this solution requires additional complexity in the configuration between hosts and routers, because the hosts would need to assign host numbers. Therefore, this third solution is not recommended.

7.1.3 Other Network Layer Address Components

If we are using the larger ISO network layer address space (the 20 octet NSAPA), then there is more room for the Landmark Address. There is also room to embed both the ID and the Zone ID in the NSAPA. For instance, consider Table 3 (taken from ISO 8348/AD2, "Network Layer Addressing"). The Domain Specific Part (DSP) of the NSAPA is the part which left to be defined by the user. The Initial Domain Identifier (IDI) is specified by ISO, and it determines which of several existing address spaces the IDI comes from. For instance, if the IDI is X.121, then it will contain a valid X.121 address. Since the various IDI's are different sizes, different sizes of DSP are left over for the user to define.[23]

Table 3
NSAPA Domain Specific Part Field Sizes

IDI Format	Binary Octets in DSP
X.121	9
ISO DCC	14
F.69	12
E.163	10
E.164	9
ISO 6523-ICD	13
Local	15

[23]Note that our term router corresponds to the ISO Intermediate System (IS), and host corresponds to the ISO End System (ES).

Assume that we want to put both an ID and a Zone ID in the address. If the IDI formats of X.121, F.69, E.163, or E.164 are used, then the IDI, along with the Authority and Format Identifier (AFI), come close to uniquely identifying the NSAPA globally, and only a small amount of additional information may be necessary in the DSP for this purpose. If ISO Data Country Code (DCC) or ISO 6523-International Code Designator (ICD) are used, then more information in the DSP must be used to uniquely identify the NSAP. With the ISO 6523-ICD (just ICD for short), there are 13 octets in the DSP. With the DCC, there are 14. In the Government Open Systems Interconnection Profile (GOSIP), 2 of the 13 octets are dedicated to the Organization Identifier (OI), leaving 11 octets (GOSIP, 1987). The American National Standards Institute (ANSI) will be assigning 3 octets as the OI, leaving 11 of the 14 octets. Of these 11 octets, we assume 1 is used for the NSAP Selector, and 4 are used for the Landmark Address, leaving 6 octets for both the ID and the Zone ID. If we assume 2 octets for the Zone ID, and 4 for the ID, then the ID can come out of the DoD Internet address space. Note that the 2 octets of Zone ID need not be themselves globally unique. They can be combined with the OI and the IDP or just the IDP to create a globally unique Zone ID. If we want to take the ID out of the six octet IEEE 802.3 address space (previously the Ethernet address space), then we have no room left for the Zone ID. If we use the X.121 or E.164 formats, and subtract 7 octets for NSAP Selector, Landmark Address, and Zone ID, we still have 2 octets left to complete the ID, which should be enough.

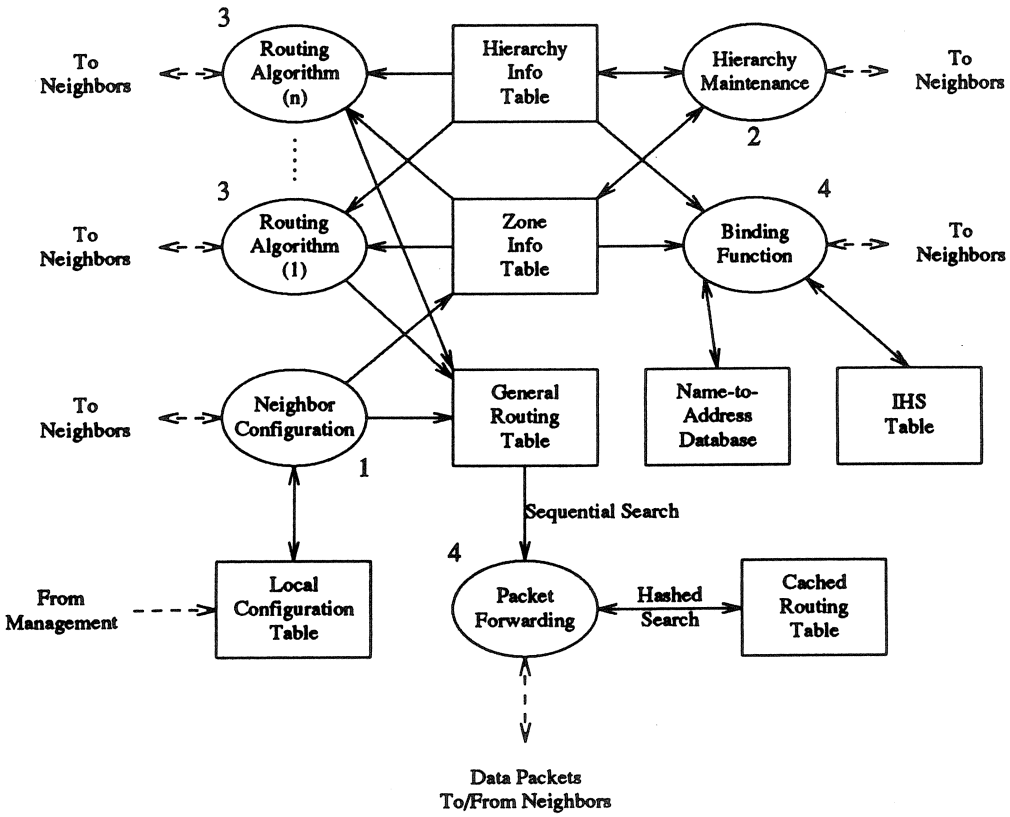
Note that the four octets of Landmark Address results in 7 levels of hierarchy. This includes 14 bits of global Landmark, and 6 lower layers of hierarchy, at 3 bits per layer.

7.2 Structure of Landmark Routing Implementation

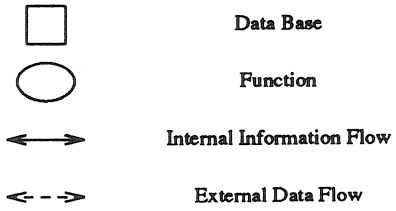
This section ties together the various components of Landmark Routing. Figure 19 is a functional diagram of a Landmark Routing implementation. This diagram shows the relationships between the various functions of Landmark Routing. In Figure 19, dashed lines denote communications with outside entities (other nodes, or management), solid lines show internal information flow, ellipses denote processes, and boxes denote data bases.

The first thing to note about Figure 19 is that the processes are not shown directly communicating with each other. All their interactions are via the various data bases. This is to show that the processes are modular in that they can be changed individually without affecting other processes. For instance, one should be able to change routing algorithms provided that the routing algorithm can act (forward or not forward routing updates) according to the contents of the

Figure 19
Landmark Routing Implementation: Functional Diagram



LEGEND:



Hierarchy Information Table and the Zone Information Table, and can provide the kind of information needed by the General Routing Table (basically, Distance-Vector style routing information).

Note also that there are numbers by the processes. These numbers denote the order in which the processes can function at start-up. The flow of information is from the Neighbor Configuration function and the Local Configuration Table, to the Hierarchy Maintenance function, then to the Routing Algorithms and the Binding function via the Hierarchy Information Table and the Zone Information Table, then finally to the Packet Forwarding function via the General Routing Table. We will discuss each process in this order.

7.2.1 Neighbor Configuration

Neighbor configuration is the process of discovering ones neighbors and verifying their identity if necessary. This function is discussed in Section 7.3. This function is not necessarily trivial. Over a single link, it is not too difficult. It is more difficult over a LAN, and more difficult still over a WAN.

Neighbor configuration takes place based entirely on information in the Local Configuration Table. This table contains all of the information which must be locally loaded into the machine upon power-up, either via the console or through a management function. In particular, it contains local address information such as Zone membership and ID. It contains all of the necessary information about neighbors, such as what their IDs should be, authentication information like public keys or passwords, link status information like a metric value, and any link layer or sub-network layer addressing needed to establish neighbor connectivity.

After neighbor configuration takes place, the Neighbor Configuration function can put routing table entries into the General Routing Table for each neighbor it has discovered. This allows packets destined for the neighbors to be routed via the normal network layer packet forwarding function. The Neighbor Configuration function also puts information into the Zone Information Table about any Zone borders that may exist across the links. This information is used by the Hierarchy Maintenance function and the Routing Algorithms as described in Section 5.

7.2.2 Hierarchy Maintenance

Once the neighbors have been configured, the Hierarchy Maintenance function can talk to the neighbors and begin to configure the Landmark Hierarchy. (Notice that we do not show in Figure

19 how the Hierarchy Maintenance function determines the identity of the neighbors. For the sake of simplicity, Figure 19 only shows the major information flows in the implementation.) Until the Hierarchy is configured, and in particular, until the local router has established its own Landmark Address, it is not really able to participate in the Routing Algorithms. The Hierarchy Maintenance function feeds information into the Zone Information Table and the Hierarchy Information Table. This information is used primarily by 1) the Routing Algorithms, to determine when to pass on Routing Updates (RUs), and how far to send its own RUs, and 2) by the Binding function to establish the Intermediate Hash Space and to process queries and updates.

7.2.3 Routing Algorithms

Once the hierarchy is configured, the Routing Algorithm functions are able to start. There may be several reasons for having more than one Routing Algorithm. First, if the router is in several Zones, it may require different Routing Algorithms for the different Zones. Second, even within a single Zone, there may be different types of service. Each type-of-service may have a separate Routing Algorithm. Also note that there may be no Routing Algorithms—the routing information learned from the Hierarchy Maintenance function may suffice for hop-based routing.

The Routing Algorithm functions feed directly into the General Routing Table. This is the table used by the Packet Forwarding function for forwarding network layer packets such as IP packets.

7.2.4 Binding Function

Like the Routing Algorithm functions, the Binding function gets its information from the Hierarchy Information Table and the Zone Information Table. Therefore, it also may not start running until the hierarchy is configured. Since however, the Binding function depends on the Packet Forwarding function to send its binding messages, the Binding function may not really fully run until the Routing Algorithms have run. Once the router has established its Landmark Address, it will send name-to-address bindings for its hosts to their designated servers, using information in the Hierarchy Information Table and Zone Information Table to begin the resolution process. It will also receive updates and queries from other routers. There are several options for architecting the Binding function in-so-far-as how hosts and routers participate in the Binding function. This is discussed in Section 7.7.

7.2.5 Packet Forwarding

The ultimate goal of all of the Landmark Routing Functions is to build the General Routing Table so that the Packet Forwarding function can do its job. The only thing the Packet Forwarding function needs to understand is how to read the General Routing Table, and how to parse the network layer headers of data packets. Also, the Packet Forwarding function should be as efficient as possible, since it must operate in real time on a packet arrival basis. The purpose of the Cached Routing Table is to indicate that in most environments, the Packet Forwarding will be able to keep a cache of recently routed packets to increase efficiency. This is necessary because a hierarchically routing table cannot be accessed via a hash. The Cached Routing Table can be accessed via a hash function on the Landmark Address or full Network Layer Address.

7.3 Neighbor Configuration Over Large Subnetworks

There is a problem in Landmark Routing with its operation over large subnetworks (large meaning lots of attached routers). We are using the word subnetwork in the ISO sense—that is, a network connecting two or more routers but which does not recognize Internetwork packets. For instance, the ARPANET does not process IP packets. Instead, routers and hosts must envelop their IP packets with X.25 or 1822 headers to cross the ARPANET.

The problem is that, since there are many routers connected to a large subnetwork, any one of them can find itself with many neighbors (several hundred, if not more). In the previous research, it was shown that the Landmark Hierarchy exhibits larger routing tables in networks with small diameters (Tsuchiya, 1987). Since large router degrees lead to smaller diameters, the Landmark Hierarchy can be expected to perform poorly if the configuration across large subnetworks is fully connected. Therefore, we must configure routers across large subnetworks so that they are not fully connected.

The technique for doing this is straightforward and efficient. First, we assume that there is some way for every router on the subnetwork to discover the existence of and subnetwork address of every other router on the subnetwork (see Section 7.3.1). (This might seem to be leading to large routing tables in and of itself. However, the large table sizes resulting from full subnetwork

[24]This is the same technique used by the author to automatically generate quasi-random networks on a computer for simulation purposes. The technique, based on the loop-span network model, is discussed in Appendix C of (Tsuchiya, 1987).

configuration are more due to Landmark Updates from the Landmarks behind the connected routers, not so much from the connected routers themselves.) Given this list, each router then configures itself—that is, establishes a virtual link—with the router with the numerically higher subnetwork address, the router with the numerically lower subnetwork address, and a third randomly chosen router. The numerically highest router would configure with the lowest. This results in a virtual network across the subnetwork which is connected (that is, every router can reach every router after a finite number of hops), and which has a diameter of workable (not too large) size.[24]

Note that if every router picked all of its neighbors randomly, one could not guarantee that the resulting virtual network was connected. There would likely be islands of disconnected routers.

Once the virtual network is configured, a Landmark Hierarchy is generated in the usual way. When a router needs to generate or pass on a LU, it sends it directly to the neighbors it has configured. Clearly, the same LU could cross the subnetwork several times, as it worked its way from router to router across the virtual links. Since LUs occur relatively infrequently, compared to regular data traffic, this is acceptable.

It is not acceptable, however, for data traffic to follow a path of virtual links several times across the subnetwork. A single data packet should almost always cross the subnetwork only once. We can use Redirects to prevent data packets from multiply traversing the subnetwork. When a router receives a data packet from the subnetwork which it subsequently routes right back onto the subnetwork, it sends a Redirect to the router that sent it the data packet informing it of the better router. The router which received the Redirect stores this in a cache. When it receives another packet for the same destination (which it almost always will, since most packets result from a transport connection) it sends it directly to the better choice. This method of circumventing the virtual network is only efficient if routers are passing traffic for a reasonably small set of destinations at any given time, and if most instances of communication between source-destination pairs involve a significant number of packets.

As an example, assume a subnetwork with 200 routers, and a virtual configuration as described above. The diameter of such a virtual network would be around 8 (based on a network with router degree of 4, and with the smallest diameter, (Tsuchiya, 1987)). The average distance between any two routers on the virtual network would be 4. Therefore, it would take on the average 3 Redirects before a router would know the optimal router on the subnetwork to forward

packets to. (With 400 routers, the average would be 3.5 Redirects, and with 800 routers, the average would be 4.5 Redirects.)

Using this scheme without a particular modification can result in routing loops of the following type. Assume that router *A* has been Redirected to router *B* for a particular destination. Assume further that router *B*'s path to the destination is no longer available, and so *B* uses an alternate path across the subnetwork. If this alternate path sooner or later includes router *A*, a loop will have formed. To prevent this, we impose the rule that a router can never forward packets received from the subnetwork to, or send Redirects for a router which is not a virtual neighbor. This way, when the packet which *B* forwarded worked its way back to *A*, *A* would not send it on to *B*, thus breaking the loop.

Since routers establish one virtual link with another router chosen at random, it is possible for many routers to try to establish virtual links with the same router. To prevent this, we suggest that each router not allow more than some number of virtual links to be established, say 6 or 7. If possible, each router can include in the discovery process how many virtual links it can still accept, so that other routers can avoid choosing it if it is getting too full.

7.3.1 Router Discovery Over Large Subnetworks

Before routers can configure a virtual network on a subnetwork, they must first discover which routers are on the subnetwork. If the subnetwork is a broadcast network, like an IEEE 802.3 LAN, then discovery is easy. Each router simply periodically multicast its presence on the subnetwork to the address "all routers" (the ISO ES-IS routing protocol, ISO 9542, already does this). The periodicity need not be rapid because the established virtual neighbors of a router which has gone down can multicast that fact—a short time-out is not necessary. When a router comes up, other routers will establish virtual links with it. When this occurs, the other routers can dump their list of discovered routers as part of the link establishment process.

If the subnetwork is not a broadcast network, as the ARPANET is not, then some broadcast-server mechanism is required. For instance, a few routers could be designated as broadcast servers. These routers would be well-known by their subnetwork addresses. (They could not be well known by their Landmark Addresses since the Landmark Addresses change.) When a router comes up, it registers with one or more of the broadcast servers. The broadcast server responds by dumping its list of routers to the new router. Further, the broadcast server must update all of its registered routers with the new router, as well as the other broadcast servers, which will in turn update their

registered routers. Routers must periodically re-register themselves with their broadcast servers, but as with the broadcast subnetwork, the period can be long.

Note that this method is similar to the core gateway technique of passing Exterior Gateway Protocol (EGP) updates around the Defense Data Network (DDN). In this case, the core gateways are acting as broadcast servers.

7.4 Application Layer or Network Layer?

One of the issues we face is whether to position Landmark Routing at the Application Layer or at the Network Layer. Referring again to Figure 20, we see five functions. Packet Forwarding and Neighbor Configuration need to be at the network layer. Packet Forwarding occurs on network layer packets, and so is clearly in the network layer. Neighbor Configuration occurs before the network layer is configured, and therefore cannot depend on the network layer for communications services. The other three functions, however, do not necessarily need to be in the network layer, in spite of the fact that they all manipulate network layer entities.

The advantages to implementing functions in the application layer are 1) portability of implementation, and 2) better underlying communications functions, such as in-sequence packet delivery. The disadvantage is that the implementation is further away from the characteristics of the underlying medium, and may not take advantage of certain features, such as the availability of broadcast, as well as an implementation in the network layer might.

Both the Binding function and the Hierarchy Maintenance function seem to be appropriate for application layer implementation. Both of them need in-sequence delivery for proper functionality, and neither of them have tight timing constraints. The Binding function is particularly appropriate because of its possible relationship with directory services or other naming services. For instance, a host may request a binding from its router as an application level request using a protocol different from the one used by the router for the binding itself. However, as discussed in Section 6.4.7, some resolution must be done by the network layer forwarding function.

Since it is not known in advance which routing algorithms will be in force with any particular implementation of Landmark Routing, it is not possible to say whether they will be at the application layer or at the network layer. Some routing algorithms, especially delay-based ones, may have tight timing constraints which make a network layer implementation more appropriate. Much of this depends on the implementation itself, and on the machine it is implemented on.

7.5 Fitting Into Global ISO Routing Architecture

The global ISO network layer routing architecture consists of routing domains which are separately administered. Each of these routing domains, which are called Administrative Domains in the ISO Routing Framework, handles its own internal routing separately. Therefore, each Administrative Domain interprets addresses differently in terms of the location information content of the addresses.

It is assumed that Landmark Routing will exist within these Administrative Domains. In other words, the global ISO network will not be one huge Landmark Hierarchy. Instead, there will be multiple Landmark Hierarchies, and other types of routing hierarchies, within a global area hierarchy-type structure. Therefore, routers in Landmark Hierarchies need to have some notion of the world outside of the hierarchy.

The concept of Zones applies to the boundaries which define an Administrative Domain. In other words, when a complete Landmark Hierarchy is an Administrative Domain within the global ISO network, then the Zone defined by that Landmark Hierarchy constitutes the Administrative Domain.

A router on the border of an Administrative Domain must pass routing information about the outside world inward, and may pass routing information about the Administrative Domain outward (via a routing protocol or a directory service). In order for a border router to pass routing information outward (or for any outside router to identify the Administrative Domain), there must be a range or ranges of addresses which identify all members of the Administrative Domain. Because of the administrative method of assigning NSAP addresses in OSI, this should normally be possible. For instance, usually a corporation will obtain a block of NSAP addresses for use on their corporate network. This block can be used to identify all routers within that network. Nodes outside of the Administrative Domain can interpret the range of addresses as they like; that is their business.

When a border router passes information about outside Administrative Domains inward, it must also identify the nodes in those Administrative Domains via a range or ranges of addresses. There are two ways to pass this information inwards. The first, which we call direct routing, is for the border router to generate a global Landmark Update containing the outside Administrative Domains as pseudo global Landmarks. They are not really global Landmarks in the sense that they do not belong in the Landmark Hierarchy. They are like global Landmarks in the sense that

distance-vector routing information is passed about them in the same fashion that it is passed about global Landmarks—that is, all Landmarks can route to them. While this way will work without any new protocol mechanisms, it is generally inefficient. This is because there will be a large number of outside Administrative Domains, many of which will never or rarely be communicated with. However, for outside Administrative Domains with which there is extensive communications, this method is a good one.

The other method, which we call indirect routing, is for the border router to advertise that outside Administrative Domains are reachable through it. To get to an outside Administrative Domain, a router need only know how to get to the border router, not how to get to the Administrative Domain itself. The difference between these two methods is a little subtle. In direct routing, each router has routing information *only* (about the outside Administrative Domains). In indirect routing, each router has routing information (about the border routers) *and* binding information (about the association between the border router and the Administrative Domain). The advantage of indirect routing is that the routing information already exists from the normal Landmark Routing, and the binding information is easier to distribute than the routing information. This is because routing information requires a routing protocol for its distribution, while binding information requires a directory service, which is simpler.

In indirect routing, when a router needs to send a message to an outside Administrative Domain, it first must discover which border router can reach the Administrative Domain, and which is closest if there are several. To do this, we use Assured Destination Binding (ADB), but with the address of the Administrative Domain itself as the hash key, rather than an ID. This will be a little inefficient because any address can be masked several different ways, resulting in several different hash keys. However, we can predetermine mask values for most types of NSAP addresses. For instance, if the NSAP address came out of the part of the Data Country Code (DCC) assigned to the American National Standards Institute (ANSI), the mask would most likely extend into the first three octets of the Domain Specific Part (DSP), because this is where ANSI puts the Organization Identifier. Likewise, for the part of the International Code Designator (ICD) which denotes the U.S. Government, the first two octets contains the Organization Identifier. If nothing else, the Initial Domain Identifier (IDI) of the NSAP address could be the masked portion of the hash key. As long as the border routers (the updating routers) and the rest of the routers (the querying routers) agree on the mask values, the binding can be made.

If all routers have implemented partial source routing, then the originating router can route the message by putting the border router in the ISO IP header using partial source routing. Otherwise, all routers on the path between the originating router and the border router must have the binding. To avoid having each of these intermediate routers initiate a binding request when it first gets the packet destined for the outside Administrative Domain, the originating router can send a message along the path to the border which contains the binding information. Each router on the path will receive this message, record the binding information in it, and pass it on to the next router. If the path should change later when packets are being sent to the outside Administrative Domain, a new router on the path will need to execute the binding. This will cause a momentary glitch in the delivery of packets, but otherwise poses no problem. Clearly, partial source routing is the preferred method.

7.6 Concerning Hosts, Routers, and Addresses: or, Where's the State?

In Section 6.1.1 we discuss how routers act as binding agents on behalf of, and transparently to, hosts. Further, in Section 7.1, we discuss how NSAPAs may contain both unique ID and Landmark Address parts, and how DoD IP addresses may be conveyed in two 32 bit fields, one for the unique ID and one (an option or an encapsulation) for the Landmark Address. These discussions raise the questions: What do hosts know of Landmark Addresses, and how do routers handle Landmark Addresses?

In general, we prefer hosts to have no notion of the Landmark Address. A host should have one label (the host ID), and that label should not change even if the host is unplugged from one network and into another. For this to happen, routers must maintain information about the host ID to Landmark Address binding.

In the case of NSAPA addresses, one way to make the Landmark Address transparent to the host is to assign an NSAPA that has four octets of zeros. These zeros amount to a place holder for the Landmark Address. When the source router receives a packet from the host, it fills in the Landmark Addresses for the source host and the destination host. If it does not have the destination host Landmark Address in its cache, it must execute the ADB function. The destination router must remove (write to zero) the Landmark Addresses before it delivers the packet to the destination host.

ISO 8473 (ISO's IP) does not allow modification of NSAPAs by routers. Therefore, legal

operation of ISO 8473 using Landmark Routing as described above would require changes to ISO 8473.

In the case of DoD IP addresses, we mentioned both encapsulation and a new option of a means of conveying the Landmark Address. The new option is the preferable method. In this case, the option would contain the source and destination Landmark Addresses (plus Zone ID and maybe other information such as Autonomous System number), and the regular source and destination address fields would contain the host address pretty much as we now know them. Since routers (gateways) are supposed to be able to ignore unknown options, this should not create a problem with regards to interoperating with existing gateways. The source router would add the option, and the destination router would remove it.

In both of the above scenarios, intermediate routers would first check for the NSAPA or destination address in their caches. If the cache lookup failed, then the Landmark Address (either from the NSAPA or the options part), would be extracted for the routing lookup, and this result would be cached. This provides efficient operation. If the Landmark Address of the destination changed, or the cache entry timed out, then the cache lookup on subsequent packets would fail, and the Landmark Address would again be extracted.

An interesting modification to this mode of operation is as follows. Assume that instead of putting the Landmark Address, Zone ID, and so on, in an options part of the address, do not put it in the address at all. Then, when a source router looks up the Landmark Address for its host via ADB, instead of putting the Landmark Address in the data packet itself, it sends a priming message along the path that the data packet would follow, providing the binding information it just discovered. Each intermediate routers along this path would cache this information for routing the data packets. An intermediate router might not have an entry in its cache if 1) the path itself changed, so that a new router began receiving data packets, 2) the router crashed and came up again, and 3) the router's cache space overflowed, and so it deleted older entries.

In the first two cases, it should be possible for the router's adjacency to re-prime its binding. The adjacency, having determined a new path, would send the binding along the new path before forwarding a packet that way. In the third case, a serious problem exists because, if the router's cache space overflowed, that means that it may be handling more transport connections than it has cache space for. If this is the case, then the router will be deleting new cache entries every time it receives a data packet for an entry it recently deleted. If it has to request the binding every time this

happens, either from the source router or through the normal ADB process, then it will be generating an unacceptable amount of traffic, and producing unacceptable delays for regular data packets.

Of course, this overflow state should be avoided by giving routers adequate memory. Since it is not possible to prevent such a state in all cases, however, there should be some mechanism for dealing with it. Two mechanisms come to mind, both of which have the overflowed router tell the source router that it is in overflow state. First, and preferably, the source router can simply put the Landmark Address in the options field for the time being. In this case, the intermediate router only needs to look into the options field for those addresses that it cannot put in its cache. Second, the source router can prevent data packets listed by the overflowing router from entering the network.

This mode of operation whereby the address is not in the data packet itself is architecturally similar to connection oriented modes of switching, whereby an address is encoded into a logical channel number. The difference here is that the "call setup" (address priming) is softer, and, if the option whereby the Landmark Address is put into the options field upon overflow condition, there is no means to "refuse a connection". This style of networking has been suggested before, and a study of it is outside the scope of this paper.

8 CONCLUSIONS

One purpse of this paper is to present a global view of all aspects of Landmark Routing at a level of detail that on one hand argues convincingly that the algorithms will work, but on the other hand does not provide much of hard proof or detailed performance estimates. Thorough research is needed.

The other point behind this paper is to disseminate ideas to as wide an audience as possible to get feedback and start an open discussion.

Our main conclusion is that the various functions presented in this paper seem feasible. None appears more complex than some existing routing algorithms (for instance, SPF in the ARPANET). Together, the individual functions make a fairly complex system, but each function is relatively decoupled from the rest in that 1) each is neatly compartmentalized, and 2) their interfaces to each other are well-defined. We feel optimistic that Landmark Routing is technically a very real possibility.

Nevertheless, this work is rather weak in a few important areas. One such area is that of failure modes. We have not explored possible complex interactions between the various functions, especially those that may result in unstable oscillatory behavior. In other words, we have not yet analyzed the Landmark Routing system as a whole. We have not yet discovered any such interactions, but a thorough search for these sorts of failure modes needs to be conducted.

Another possible problem with Landmark Routing is its debugability. Landmark Routing is not a deterministic system—that is, one cannot look at the current state of the network (connectivity, traffic flows, administrative boundaries, and so on) and determine what the state of Landmark Routing should be (in particular, which nodes should be Landmarks). The selection of Landmarks is based on past history, for instance, on which nodes came up first. This complicates the problem of debugging failures in the network.

In addition, the fact that no node has complete information about the state of the network makes debugging more difficult. (This problem is not isolated to Landmark Routing. Any hierarchical routing system has this quality, although link-state routing algorithms exhibit it to a far lesser degree). This paper does not consider the problem of debugging and has not considered what functions can be added to Landmark Routing to aid in debugging. These too are for future study.

Finally, if anything is destined to prevent Landmark Routing, it would be the problems associated with operating Landmark Routing in the existing internet. First, the technical complexities of interfacing Landmark Routing with existing routing protocols, and interfacing an ISO IP implementation with DoD IP implementations are certainly on the order of magnitude of Landmark Routing itself.

Second, Landmark Routing provides the most benefit when everybody uses it (as is true with any standard, but even more so). However, there will be an extended period during which only pockets of users will use Landmark Routing. At any point in time, for any given administration, it may be easier to implement a simpler routing algorithm. Over the long run, though, I believe that the wide-spread use of Landmark Routing will provide benefits that way outweigh the short-term pain of fragmented implementations.

8.1 Future Work

The next stage in this work is to fully specify and simulate Landmark Routing. This work is scheduled for 1988. The simulations will be written on a commercial network simulation tool called OPNET (c). OPNET is an event-driven simulator that allows the programmer to write the algorithms to be tested in C as though it were an actual implementation. OPNET provides tools to architect nodes, networks, and simulation sequences. It also provides a rich set of analysis capabilities. Using OPNET, we hope to simulate networks of at least 200 nodes.

The result of the simulation phase will be 1) a working implementation of Landmark Routing, and 2) performance measurements. The next stage of this work is to test this implementation in a real network. The implementation in the simulation package will be ported to a Unix system. Our intent is to write all of the software in user space, use Transmission Control Protocol (TCP) as a link protocol, and build a virtual network consisting of Unix hosts connected via TCP links. The "network layer" will be implemented above TCP. This allows one to test a large network without getting into existing IP implementations or routing algorithms. It also gives one significant flexibility with regards to the network topology, as TCP connections can be brought up and down as desired.

The test phase will debug the implementation further, and will give us experience in running and debugging Landmark Routing over a large network. The final phase after this will be a real implementation. It is not yet clear in which network this implementation will be.

APPENDIX A

Glossary of Mathematical Expressions

N	The total number of nodes in a network.
D	The diameter of a network (the maximum distance between any two nodes).
C	The average node degree of a network, where the node degree of a single node is the number of nodes it is connected to.
L	The number of links in a network. (When written as a subscript it refers to the lowest level of the Landmark Hierarchy, known as the *local* level.)
H	The total number of hierarchy levels.
i	The Landmark Hierarchy level.
LM_i	A Landmark of level i.
^{G}	The superscript ^{G} indicates global level. For instance, LM_i^G is a global Landmark.
$LM_i[id]$	An LM_i with ID = id.
$r_i[id]$	The radius of a Landmark Vicinity for a particular $LM_i[id]$.
r_i	The average radius of a group of LM_i. When this appears in the text, the set of $r_i[id]$ which are being averaged is understood by context. If a single network or simulation is the context, r_i is averaged over all Landmarks $LM_i[id]$ in the network. When an experiment (a group of simulations) is the context, r_i is averaged over all Landmarks for all simulations in the experiment.
$d_i(id_x \rightarrow id_y)$	The distance from a particular node with ID id_x to either, 1) $LM_i[id_y]$ if id_y is specified, or to the closest LM_i if it is not.
d_i	The average distance from each node to its closest LM_i.
$d_{i\ max}$	The maximum distance a node may be from an LM_i.
$v(x)$	The number of nodes within x hops of a node.
T_i	The total number of LM_i in a network.
R	The average total number of entries in node's routing tables ($R[id]$ for a specific node). (When written as a subscript, it refers to the root level of the Landmark Hierarchy.)

R_i The average total number of LM_i in a node's routing table ($R = \sum_{i=0}^{H} R_i$).

$\hat{P}_i$ The average increase in path length over shortest path for all P_i^{lm} ($\hat{P}_i = \frac{\sum_{node\ pairs} \frac{P_i^{lm}(node\ pair)}{P^{sh}(node\ pair)}}{node\ pairs}$).

$\hat{P}$ The average $\hat{P}_i$ for all levels i.

APPENDIX B

Glossary of Definitions

Address: When used with an upper-case "A", it means the same an *Landmark Address.*

Administrative Zone: That group of Landmarks that have been grouped together administratively for the purposes of controlling certain routing functions with regards to that group.

Ancestors: This term is used only with respect to the address tree formed by the Landmark Addresses of nodes. The *ancestors* of a level i Landmark x are all of those level i+1 and higher Landmarks upon which Landmark x has based its Landmark Address. In other words, Landmark x's parent, its parent, and so on.

Binding Server: The Node s that holds the ID-to-Landmark Address binding for some other Node x. In particular, Node s is said to be Node x's Binding Server.

Child: This term is used only with respect to the address tree formed by the Landmark Addresses of nodes. Given that a level i Landmark has based its Landmark Address on the Landmark Label of a particular level i+1 Landmark, the level i Landmark is the *child* of the level i+1 Landmark.

Downtree: This term is used only with respect to the routing spanning-tree that emanates from a destination node, and is determined by the routing table entries for that destination node. Downtree means in the direction of the spanning-tree leaves (or, identically, away from the destination node).

Fork (or forking router): A router that has two or more downtree links with respect to a given destination.

Global Landmark: Any Landmark whose Landmark Vicinity includes all routers in the network—whose radius is as large as or larger than the network diameter.

Horizontal: This term is used only with respect to the routing spanning-tree that emanates from a destination node, and is determined by the routing table entries for that destination node. Horizontal means neither towards nor away from the root of the spanning tree.

Host: The physical computer that is a source or sink for traffic. Hosts do not participate in Landmark Routing per se. Instead, they associate themselves with routers through whatever means are available, such as ISO 9542, the ES-IS routing protocol. Ideally, a host should have no notion of Landmark Addresses. An ISO End System is a host.

ID: The permanent label that distinguishes all routers and hosts from all other routers and

hosts. The Assured Destination Binding function is used to map IDs to Landmark Addresses.

Juncture (router): A router than has more than one uptree or horizontal link towards a destination. A full-juncture router is one where at least two of its paths to the destination share no uptree routers. A partial-juncture router is one where all of its paths to the destination share at least one router.

Landmark: That logical entity in a router that participates in Landmark Routing, and for which neighboring routers maintain a routing table entry. This is to be distinguished from the term *router*, that is the physical computer which acts as a Landmark, performs the forwarding function, may act as a server, is connected to other routers, and so on.

Landmark Address: The label that determines a router's position in the Landmark Hierarchy. The *Landmark Address* consists of a series of Landmark Labels, one for each level of the Landmark Hierarchy.

Landmark Label: The individual components of a Landmark Address that identify each Landmark at each level i. Except for Global Landmarks, Landmark Labels are not globally unique. Rather, level i Landmark Labels are unique among the children of a level $i+1$ Landmark. Landmark Labels at the global level must be unique among themselves.

Landmark Vicinity: That group of routers that have a routing table entry for a particular Landmark. The Landmark Vicinity is determined by the radius of a Landmark.

Network Layer Address: The address field of a data packet. This field must at least contain the ID of the destination, and will normally also carry the Landmark Address. It also may carry the Zone ID.

Node: Used to refer to both routers and hosts.

Offspring: This term is used only with respect to the address tree formed by the Landmark Addresses of nodes. The *offspring* of a level i Landmark x are all of those level $i-1$ and lower Landmarks that base their Landmark Address on that of Landmark x. In other words, Landmark x's children, their children, and so on.

Parent: This term is used only with respect to the address tree formed by the Landmark Addresses of nodes. Given that a level i Landmark has based its Landmark Address on the Landmark Label of a particular level $i+1$ Landmark, the level $i+1$ Landmark is the *parent* of the level i Landmark.

Peer: All Landmarks at the same level are *peers*. This is distinguished from *siblings*, which are *peers* that have a common parent.

Root Landmark: The highest Landmark in a Landmark Hierarchy.

Router: The physical computer that forwards traffic to and from other routers and hosts. This is distinguished from a host, which does not forward traffic, but is only a source or sink for traffic. Thus, gateways, Intermediate Systems, Packet Switched Nodes, packet radios, and so on, are all referred to as *routers*.

Server: When used with an upper-case "S", it means the same an *Binding Server*.

Sibling: This term is used only with respect to the address tree formed by the Landmark Addresses of nodes. Level i Landmarks that have the same level $i+1$ parent are *siblings*.

Uptree: This term is used only with respect to the routing spanning-tree that emanates from a destination node, and is determined by the routing table entries for that destination node. Uptree means in the direction of the spanning-tree root (or, identically, towards the destination node).

Zone: When used with an upper-case "Z", it means the same as "Administrative Zone".

Zone ID: The ID of a Zone. This ID must be unique among all Zones. In the case of hierarchically nested Zones, the Zone ID (or ZID) may, for the sake of efficient and convenient encoding, also be hierarchically structured.

Zone Root: The highest-level Landmark within a particular Zone.

LIST OF REFERENCES

American National Standards Institute Accredited Standards Committee X3S3.3, (July 1987[a]), *Intermediate System to Intermediate System Intra-Domain Routing Exchange Protocol*, X3S3.3/87-160R1.

American National Standards Institute Accredited Standards Committee X3S3.3, (July 1987[b]), *Intermediate System to Intermediate System Intra-Domain Routing Exchange Protocol*, X3S3.3/87-150.

Cegrell, T. (June 1975), *A Routing Procedure for the Tidas Message-Switching Network*, IEEE Trans. on Communications, Vol. COM-23, No. 6, pp. 575-585.

Garcia-Luna-Aceves, J. J. (1987), *A New Minimum-Hop Routing Algorithm*, Proceedings for the IEEE Infocom '87, pp. 170-180.

Garcia-Molina, H. (January 1982), *Elections in a Distributed Computing System*, IEEE Transactions on Computers, Vo. C-31, No. 1, pp. 48-49.

Gardner, Dr. M. L. (1985), *Type of Service Routing Requirements*, Report No. 6096, Cambridge, MA: BBN Communications Corporation.

Gerla, M. (November, 1984), "Controlling Routes, Traffic Rates, and Buffer Allocation in Packet Networks", *IEEE Communications Magazine*, Vol. 22, No.11, pp. 11-23.

Gruchevsky, S. and D. Piscitello (January/April 1987), "The Burroughs Integrated Adaptive Routing System (BIAS)," ACM Computer Communication Review, Volume 17, Nos. 1 and 2.

Hagouel, J. (1983), "Issues in Routing for Large and Dynamic Networks," PhD. Thesis, Columbia University.

International Organization for Standardization (ISO) 8473, *Protocol for the Provision of the Connectionless-mode Network Service.*

International Organization for Standardization (ISO) 9542, *End System to Intermediate System Routing Exchange Protocol for Use in Conjunction with the Protocol for the Provision of the Connectionless-mode Network Service (ISO 8473).*

International Organization for Standardization (ISO) Routing Framework.

Jaffe, J. M. and F. H. Moss (July 1982), "A Responsive Distributed Routing Algorithm for Computer Networks," IEEE Trans. on Communications, COM-30, No. 7, pp. 1758-1762.

Khanna, A. and J. Seeger (January 1986), "Large Network Routing Study Design Document," Report No. 6119, Cambridge, MA: BBN Communications Corporation.

McQuillan, J. M., I. Richer, and E. C. Rosen, (April 1978) "ARPANET Routing Algorithm Improvements First Semiannual Technical Report," Bolt Beranek and Newman Inc., Report No. 3803.

McQuillan, J. M., I. Richer, and E. C. Rosen, (May 1980) "The New Routing Algorithm for the ARPANET," IEEE Trans. on Communications, Vol. COM-28, No. 5, pp. 711-719.

National Bureau of Standards (April 1987), *Government Open Systems Interconnection Profile*

(GOSIP), Draft FIPS PUB, Gaithersburg, MD: U.S. Department of Commerce, NBS, U.S. Government OSI User's Committee.

Perlman, R. (October, 1981), "Incorporation of Service Classes into a Network Architecture," Proceedings of Seventh Data Communications Symposium, SIGCOMM, Vol. 11, No. 4, 204-210.

Perlman, R. (1985), "Hierarchical Networks and the Subnetwork Partition Problem," Computer Networks and ISDN Systems 9, North-Holland, pp. 297-303.

Perlman, R. (1983), "Fault Tolerant Broadcast of Routing Information," Computer Networks, Vol. 7, pp. 395-405.

Schwartz M. and T. Stern (April 1980) "Routing Techniques Used in Computer Communication Networks," IEEE Transactions on Communications, Vol. COM-28, No. 4.

Shacham, N. (November 1985), "Hierarchical Routing in Large, Dynamic Ground Radio Networks," Menlo Park, CA: SRI International.

Shoch, J., D. Cohen, and E. A. Taft (July 1981), "Mutual Encapsulation of Internetwork Protocols," Computer Networks, Vol. 5, No. 4, 287-301.

Sparta Incorporated, (April 1986), *Design and Analysis for Area Routing in Large Networks,*, McLean, VA: Sparta Incorporated.

Stine, Robert H. Jr. and Paul F. Tsuchiya (March 1987), *Assured Destination Binding: A Technique for Dynamic Address Binding*, MTR-87W00050, McLean, VA: The MITRE Corporation.

Tsuchiya, P. F. (June 1987), *The Landmark Hierarchy: Description and Analysis*, MTR W87-87W00152, McLean, VA: The MITRE Corporation.

PSN	Packet Switched Node
RU	Routing Update (message)
SPF	Shortest Path First
TCP	Transmission Control Protocol
WAN	Wide Area Network
ZID	Zone ID

GLOSSARY

ADB	Assured Destination Binding
ADR	Alternate-Path Distance-Vector Routing
ANSI	American National Standards Institute
APP	Alternate Path Priming (message)
BBZ	Binding By Zone
CT∞	Counting-to-Infinity
DARPA	Defense Advanced Research Projects Agency
DCA	Defense Communications Agency
DCC	Data Country Code
DDN	Defense Data Network
DoD	Department of Defense
DoD IP	MIL-STD-1777
DSP	Domain Specific Part
EGP	Exterior Gateway Protocol
FJ	Full Juncture (router)
GGP	Gateway-to-Gateway Protocol
GOSIP	Government Open Systems Interconnection Profile
ICD	International Code Designator
ID	Identifier
IDI	Initial Domain Identifier
IP	Internet Protocol
ISO	International Organization for Standardization
ISO IP	ISO-8473
JC	Juncture Configuration (message)
LAN	Local Area Network
LPAT	Loop Prevention Algorithm T
LU	Landmark Update (message)
NSAP	Network Service Access Point
NSAPA	Network Service Access Point Address
OI	Organization Identifier
OSI	Open Systems Interconnection
PJ	Partial Juncture (router)

DISTRIBUTION LIST

MITRE Washington

A-10 B. Horowitz
G. MacDonald
C. Zraket

D-14 J. Quilty

W-30 R. Binder
R. Britton
G. Lipsey
R. Shirey
L. Wentz
D. Woodall

W-31 J. Ackermann
S. Bhanji
H. Duffield
D. Gomberg
R. Hansen
W. Lazear
S. McLeod
M. Meltzer
A. Messeh
R. Miller
G. Reichlen
M. Stella
P. Tsuchiya (100)
A. Whitaker
R. Wilmer
D. Wood (2)
Associate Technical Staff
Technical Staff

W-33 J. Rubin
A. Schoka

W-34 E. Schmidt

W-35 C. Bowen
D. Jurenko
W. Kinzinger

W-37 R. Haller
R. Pesci
D. Zugby

W-75 W. Blankertz

MITRE Washington Library

Record Resources (2)

MITRE Bedford

D-110 G. Koehr
I. Richer

D-111 J. Woodward

D-114 H. Bayard

D-118 S. Ames

MITRE Bedford Library

External

DCA/DCSO

COL T. Herrick, Code B600
Mr. E. Schonborn, Code B601
LtCol G. Mundy, Code B602
Mr. L. Tabacchi, Code B602
Mr. W. Grindle, Code B610
LTC A. Maughan, Code B620
LtCol C. Holland, Code B630
LTC D. Schreiner, Code B640
Mr. J. Milton, Code B650

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) MTR-87W00174	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION The MITRE Corporation 6b. OFFICE SYMBOL (*If applicable*) W31	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (*City, State, and ZIP Code*) 7525 Colshire Drive McLean, Virginia 22102	7b. ADDRESS (*City, State, and ZIP Code*)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION DCA 8b. OFFICE SYMBOL (*If applicable*) B620	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (*City, State, and ZIP Code*) Deputy Director Defense Communications Agency 8th and South Courthouse Rd. Arlington, VA 22204-2199	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. 359Y / TASK NO. 5000.626 / WORK UNIT ACCESSION NO.
11. TITLE (*Include Security Classification*) Landmark Routing: Architecture, Algorithms, and Issues	
12. PERSONAL AUTHOR(S) Paul F. Tsuchiya	
13a. TYPE OF REPORT Final 13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (*Year, Month, Day*) 1988-05-30 15. PAGE COUNT 130
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES			18. SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (*Continue on reverse if necessary and identify by block number*)

This paper is the second in a series of papers that document the research, development, specification, implementation, and deployment of a new routing technique called Landmark Routing. Landmark Routing is a distributed and adaptive hierarchical routing protocol for use in networks and internets of any size. Its primary features are that it is robust and durable in the face of rapid topological changes, that it is easy to administer, and that it provides full name-based a-dressing. The reason for these advantages is that Landmark Routing dynamically establishes its own hierarchy and modifies it as the network undergoes changes. This paper gives a medium to high-level design of all of the components of Landmark Routing--the routing algorithms, the hierarchy maintenance algorithms, the administrative zones, and the name-to-address binding algorithms. This paper also discusses ancillary issues such as transition from existing routing schemes and implementation design decisions. The paper concludes that Landmark Routing is a workable solution to a host of large network routing problems.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION
22a. NAME OF RESPONSIBLE INDIVIDUAL LTC Allan M. Maughan, Code B620	22b. TELEPHONE (*Include Area Code*) 703-285-5101 22c. OFFICE SYMBOL Code B620

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED