The Zone Routing Protocol (ZRP) for Ad Hoc Networks
<draft-ietf-manet-zone-zrp-02.txt>

Status of this Memo

This document is an Internet-Draft and is NOT offered in accordance with Section 10 of RFC2026, and the author does not provide the IETF with any rights other than to publish as an Internet-Draft.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Distribution of this memo is unlimited.

Abstract

This document describes the Zone Routing Protocol (ZRP), a hybrid routing protocol suitable for a wide variety of mobile ad-hoc networks, especially those with large network spans and diverse mobility patterns. Each node proactively maintains routes within a local region (referred to as the routing zone). Knowledge of the routing zone topology is leveraged by the ZRP to improve the efficiency of a reactive route query/reply mechanism. The proactive maintenance of routing zones also helps improve the quality of discovered routes, by making them more robust to changes in network topology. The ZRP can be configured for a particular network by proper selection of a single parameter, the routing zone radius.

The ZRP framework is designed to provide a balance between the contrasting proactive and reactive routing approaches. To underscore this general philosophy, both the proactive and reactive components of this ZRP implementation have been expanded. This draft provides specifications for both a (split-horizon) Distance Vector AND a Link State version of the proactive IntrAzone Routing Protocol (IARP). The reactive IntErzone Routing Protocol (IERP) provides a route caching option. When route caching is globally enabled, the IERP acts as a reactive Distance Vector protocol, distributing routing information among intermediate nodes.

On the other hand, when route caching is disabled, the routes are accumulated in the query packets and the protocol operates through source routing.
This ZRP specification also offers some additional enhancements. The reactive route query procedure now supports route querying for multiple destinations and multicast groups. Queries can be targeted for either ANY destination or ALL destinations (depending on the nature of the query).

By aggregating route queries, the amount of overhead traffic can be greatly reduced. In addition, the ZRP now supports QOS routing through the collection of various route quality metrics (in both the proactive and reactive routing components).

Contents

Status of this Memo ............................................. i
Abstract .................................................................. i

Applicability Statement ................................. v
 A. Networking Context .................................... v
 B. Protocol Characteristics and Mechanisms ........ v

1. Introduction ................................................. 1

2. Overview of the Zone Routing Protocol ............. 3
  2.1 The Notion of a Routing Zone and the
      Intrazone Routing Protocol (IARP) .......... 3
  2.2 The Interzone Routing Protocol (IERP) ......... 4
     2.2.1 Routing Zone Based Route Discovery ... 4
     2.2.2 Route Accumulation Procedure ......... 5
     2.2.3 Query Control Mechanisms .......... 6
     2.2.4 Route Maintenance ................... 7

3. The ZRP Architecture .................................... 8

4. Implementation Details .................................. 9
  4.1 Intrazone Routing Protocol (IARP) ............ 9
     4.1.1 Distance Vector Implementation .......... 9
        A. Packet Format ............................. 10
        B. Data Structures ....................... 11
        C. Interfaces .............................. 11
        D. State Machine ......................... 12
        E. Pseudocode Implementation ............ 14
     4.1.2 Link State Implementation ............ 16
        A. Packet Format .......................... 16
        B. Data Structures ....................... 18
        C. Interfaces .............................. 20
        D. State Machine ......................... 20
        E. Pseudocode Implementation ............ 21
Applicability Statement

A. Networking Context

The hybrid Zone Routing Protocol (ZRP) can adapt to a wide variety of network scenarios by adjusting the range of the nodes' proactively maintained routing zones. Large routing zones are preferred when demand for routes is high and/or the network consists of many slowly moving nodes. In the extreme case of a network with fixed topology, the ideal routing zone radius would be infinitely large. (Consider the purely proactive routing protocols used on the fixed Internet). On the other hand, smaller routing zones are appropriate in situations where route demand is low and/or the network consists of a small number of nodes that move fast relative to one another. In the "worst case", a routing zone radius of one hop is best, and the ZRP defaults to a traditional reactive flooding protocol.

When the ZRP is properly configured for a particular network scenario, it can perform at least as well as (and often better than) its purely proactive and reactive constituent protocols. In situations where the network behavior varies across different regions, the ZRP performance can be fine.
tuned by individual adjustment of each node’s routing zone.

The global reactive component of the ZRP uses the unicast/multicast based "bordercast" mechanism to propagate route queries throughout the network, rather than neighbor-broadcast based flooding found in traditional reactive protocols. Consequently, the ZRP provides the most benefit in networks where reliable neighbor broadcasting is either inefficient or all-together impossible. In particular, the ZRP is well suited for multi-channel, multi-technology routing fabrics and networks operating under high load.

B. Protocol Characteristics and Mechanisms

* Does the protocol provide support for unidirectional links? (if so, how?)
  Yes. The ZRP provides "local" support for unidirectional links. A unidirectional link can be used as long as the link source and link destination lie within each other’s routing zone.

* Does the protocol require the use of tunneling? (if so, how?)
  No.

* Does the protocol require using some form of source routing? (if so, how?)
  No. The ZRP’s global reactive route discovery mechanism may use either source routing or distributed distance vector based route accumulation.

* Does the protocol require the use of periodic messaging? (if so, how?)
  Yes. The ZRP proactively maintains local routing information (routing zones) based on periodic exchanges of neighbor discovery messages.

* Does the protocol require the use of reliable or sequenced packet delivery? (if so, how?)
  The ZRP only assumes that link-layer (neighbor) unicast is delivered reliably and in-sequence. Reliable and sequenced delivery of neighbor broadcasts and IP unicast is not required.

* Does the protocol provide support for routing through a multi-technology routing fabric? (if so, how?)
  Yes. It is assumed that each node’s network interface is assigned a unique IP address.
* Does the protocol provide support for multiple hosts per router? (if so, how?)

Yes. Multiple hosts may be associated with a router. These hosts can be identified by the neighbor discovery/maintenance process.

By default, it is assumed that a host’s association with a router is not permanent. As a result, the ZRP exchanges routing information for individual hosts in the same manner as routing information for routers.

In cases where a router is permanently associated with a network (subnetwork), the ZRP supports the exchange of network (subnetwork) prefixes in place of all aggregated host IP addresses.

* Does the protocol support the IP addressing architecture? (if so, how?)

Yes. Each node is assumed to have a unique IP address (or set of unique IP addresses in the case of multiple interfaces). The ZRP references all nodes/interfaces by their IP address.

This version of the ZRP also supports IP network addressing (network prefixes) for routers that provide access to a network of non-router hosts.

* Does the protocol require link or neighbor status sensing (if so, how?)

Yes. The ZRP proactively maintains local routing information (routing zones) based on detected changes in neighbor status.

* Does the protocol have dependence on a central entity? (if so, how?)

No. The ZRP is a fully distributed protocol.

* Does the protocol function reactively? (if so, how?)

Yes. The ZRP’s GLOBAL route discovery mechanism is reactive.

A route query is initiated, on demand, when a node requires routing information that is not immediately available in its routing table.

The route query propagates through the network, using a ZRP-specific packet delivery service called "bordercasting". Bordercasting leverages knowledge of local network topology to direct route queries away from the source, thereby reducing redundancy.

* Does the protocol function proactively? (if so, how?)

Yes. The ZRP proactively maintains local routing information (routing zones) for each node. The routing zone information is
leveraged, through the bordercasting mechanism, to support efficient global propagation of route queries.

* Does the protocol provide loop-free routing? (if so, how?)

Yes. Loop-freedom in the reactive route discovery process is ensured by labeling each route query and route reply with the IP address of its originating node AND a unique sequence number. Nodes that relay the route queries/route replies temporarily cache these identifiers in order to identify and terminate loops.

The routing protocols used for proactive routing zone maintenance are based on traditional Distance Vector or Link State routing protocols. The scope of these proactive route updates is limited to the extent of a node’s routing zone. The ZRP's Link State proactive protocol is inherently loop-free. The Distance Vector protocol may form temporary loops prior to converging. However, convergence occurs quickly due to the limited radius of the routing zones.

* Does the protocol provide for sleep period operation? (if so, how?)

No.

* Does the protocol provide some form of security? (if so, how?)

No. It is assumed that security issues are adequately addressed by the underlying protocols (IPsec, for example)

* Does the protocol provide support for utilizing multi-channel, link-layer technologies? (if so, how?)

Yes. It is assumed that each node’s network interface is assigned a unique IP address.

1. Introduction

One of the major challenges in designing a routing protocol for the ad hoc networks stems from the fact that, on one hand, to determine a packet route, a node needs to known at least the reachability information to its neighbors. On the other hand, in an ad hoc network, the network topology can change quite often. Furthermore, as the number of network nodes can be large, the potential number of destinations is also large, requiring large and frequent exchange of data (e.g., routes, routes updates, or routing tables) among the network nodes. Thus, the amount of update traffic can be quite high.
This is in contradiction with the fact that all updates in a wirelessly interconnected ad hoc network travel over the air and, thus, are costly in resources.

In general, the existing routing protocols can be classified either as proactive or as reactive. Proactive protocols attempt to continuously evaluate the routes within the network, so that when a packet needs to be forwarded, the route is already known and can be immediately used. The family of Distance-Vector protocols is an example of a proactive scheme. Reactive protocols, on the other hand, invoke a route determination procedure on demand only. Thus, when a route is needed, some sort of global search procedure is employed. The family of classical flooding algorithms belong to the reactive group. Some examples of reactive (also called on-demand) ad hoc network routing protocols are Dynamic Source Routing (DSR) [6], Ad-hoc On demand Distance Vector Routing (AODV) [11] and the Temporally Ordered Routing Algorithm (TORA) [9].

The advantage of the proactive schemes is that, once a route is needed, there is little delay until the route is determined. In reactive protocols, because route information may not be available at the time a datagram is received, the delay to determine a route can be quite significant. Furthermore, the global flood-search procedure of the reactive protocols requires significant control traffic. Because of this long delay and excessive control traffic, pure reactive routing protocols may not be applicable to real-time communication. However, pure proactive schemes are likewise not appropriate for the ad hoc networking environment, as they continuously use a large portion of the network capacity to keep the routing information current. Since nodes in an ad hoc networks move quite fast, and as the changes may be more frequent than the route requests, most of this routing information is never even used! This results in a further waste of the wireless network capacity. What is needed is a protocol that, on one hand, initiates the route-determination procedure on-demand, but at limited search cost. The protocol described in this draft, termed the "Zone Routing Protocol (ZRP)" ([1], [2]), is an example of a such a hybrid proactive/reactive scheme.

The ZRP, on one hand, limits the scope of the proactive procedure only to the node’s local neighborhood. On the other hand, the search throughout the network, although global in nature, is done by efficiently querying selected nodes in the network, as opposed to querying all the network nodes.

A related issue is that of updates in the network topology. For a routing protocol to be efficient, changes in the network topology should have only a local effect. In other words, creation of a new link at one end of the network is an important local event but, most probably, not a significant piece of information at the other end of the network. Proactive protocols tend to distribute such topological changes widely in the network, incurring large costs. The ZRP limits propagation of such information to the neighborhood of the change only, thus limiting the cost of topological updates.
2. Overview of the Zone Routing Protocol

2.1 The Notion of a Routing Zone and the Intrazone Routing Protocol (IARP)

A routing zone is defined for each node X, and includes the nodes whose minimum distance in *hops* from X is at most some predefined number, which is referred to as the Zone Radius. An example of a routing zone (for node A) of radius 2 is shown below.

```
+-----------------------------------------+
|                       +---+             |
|            +---+   ---| F |-------|     |
| G |-----| D |--/  +---+                | E |   |  Zone of node A  |
|        +---+       |        +---+     +---+   |  of radius=2 |
|            | A |--/   +---+             |
|            +---+                        |
|            +---+   ---| B |-------|     |
|            | A |--/          +---+               |
|            |               +---+                 |
|            |               +---+                 |
|            +-----------------------------+
```

Note that in this example nodes B through E are within the routing zone of A. Node G is outside A’s routing zone. Also note that E can be reached by two paths from A, one with length 2 hops and one with length 3 hops. Since the minimum is less or equal than 2, E is within A’s routing zone.

Peripheral nodes are nodes whose minimum distance to the node in question is equal exactly to the zone radius. Thus, in the above figure, nodes D, F, and E are A’s peripheral nodes.

An important consequence of the routing zone construction is the ability of a node to deliver a packet to its peripheral nodes. This service, which we refer to as bordercasting, allows for more efficient network-wide searching than simple neighbor broadcasting. Bordercasting could be implemented either through a series of IP unicasts or an IP multicast (Distance Vector Multicast Routing Protocol [13]) to the peripheral nodes. (In cases where multicasting is supported, the multicasting approach is preferred to reduce the amount of traffic over the air.) However, as will be explained later, efficient ZRP operation requires that these unicast or multicast services be provided by the ZRP, with IP providing best-effort delivery to the specified ZRP next hops.

The ZRP supports the proactive maintenance of routing zones through its proactive Intrazone Routing Protocol (IARP). Through the IARP, each node learns the identity of and the (minimal) distance to all the nodes in its routing zone. The IARP may be derived from a wide range of proactive protocols, such as Distance Vector (e.g., [8], [10]) or Shortest Path First (e.g., OSPF [7]). Whatever the choice of IARP is, the base protocol needs to be modified to ensure that the scope of this operation is restricted to the radius of a node’s routing zone.
Because each node needs to proactively acquire route information only for the nodes within its zone, the total amount of IARP traffic does not depend on the size of the network, which may be quite large.

2.2 The Interzone Routing Protocol (IERP)

While the IARP maintains routes within a zone, the IERP* is responsible for finding routes between nodes located at distances larger than the zone radius. The IERP is distinguished from standard flood-search query/response protocols by exploiting the routing zone topology. A node is able to respond positively to any queries for its routing zone nodes. For large networks, relatively few destinations lie within any particular node’s routing zone. In this more likely case, the node can efficiently continue the propagation of the query, through the routing zone-based bordercast delivery mechanism.

2.2.1 Routing Zone Based Route Discovery

The IERP operates as follows: The source node first checks whether the destination is within its routing zone. (Again, this is possible as every node knows the content of its zone). If so, the path to the destination is known and no further route discovery processing is required. If, on the other hand, the destination is not within the source’s routing zone, the source bordercasts a route query to all of its peripheral nodes. Now, in turn, all the peripheral nodes execute the same algorithm: check whether the destination is within their zone. If so, a route reply is sent back to the source indicating the route to the destination. If not, the peripheral node forwards the query to its peripheral nodes, which, in turn, executes the same procedure. An example of this Route Discovery procedure is demonstrated in the figure below.

* Some functions of the IERP, including bordercasting, route accumulation, and query control (see later), are performed by a special component of the IERP called the Bordercast Resolution Protocol (BRP). Sections 3.0 and 4.0 describe, in detail, the relationship between the BRP and the IERP proper.
The node A has a datagram to send to node L. Assume a uniform routing zone radius of 2 hops. Since L is not in A’s routing zone (which includes B, C, D, E, F, G), A bordercasts a routing query to its peripheral nodes: D, F, E, and G. Each one of these peripheral nodes check whether L exists in their routing zones. Since L is not found in any routing zones of these nodes, the nodes bordercast the request to their peripheral nodes. In particular, G bordercasts to K, which realizes that L is in its routing zone and returns the requested route (L-K-G-A) to the query source, namely A.

### 2.2.2 Route Accumulation Procedure

The query propagation mechanism allows a route query to indirectly reach the desired destination (through some node Y, which discovers the destination in its routing zone.) To complete the route discovery process, Y needs to send a reply back to the query’s source, S, providing S with the desired route. To perform the route reply, sufficient information needs to be accumulated during the query phase so that Y is provided with a route back to S. Providing routes from discovering node Y to query source S, and from the query source S back to the query destination D (through Y), is the role of the Route Accumulation procedure.

In the basic Route Accumulation, a node appends its IP address to a received query packet. The sequence of IP addresses specifies a route from the query’s source to the current node. By reversing this sequence, a route may also be obtained back to the source. In this way, Y may send a reply back to S, through strict source routing.

Given sufficient storage space, a queried node may cache routing information accumulated in the query packet, allowing the information to be remove from the packet. This has the benefit of reducing the length of the query packet, thereby decreasing the query traffic and query response time. The IP addresses that remain in the packet can be used to form a loose source route back to the query’s source (If ALL nodes have cached the accumulated route information, then this effectively becomes next hop routing. If no nodes have cached accumulated route information, then this defaults to the basic case previously discussed). The same caching strategy can be applied to the reply packet on its way back to the source. In this case, a loose source route to the destination is formed.

### 2.2.3 Query Control Mechanisms

Bordercasting has the potential to support global querying schemes that are more efficient than flooding. To achieve this efficiency, the protocol should be able to detect and terminate a query thread when it appears in a previously queried region of the network (i.e. arrives at a node belonging to the routing zone of a previously queried node). This detection / termination capability is
significantly limited when bordercasting is implemented directly through IP unicast or IP multicast.

By implementing bordercasting within the ZRP, the nodes that relay the query to the peripheral node are able to detect the passing query. If the underlying IP delivery is (neighbor) broadcast or if IP is operating in promiscuous mode, then nodes that overhear a query transmission are also able to detect the query, further strengthening the Query Detection (QD) mechanism. Upon detecting a query, the identifying query parameters (i.e. query source, query ID) are recorded in a Detected Queries Table, to provide a basis for termination of future threads of that query.

A node can consider a query to be redundant if it has already detected that query, bordercasted by a different node. If bordercasting is implemented directly through IP unicast/ multicast, then a query thread could only be terminated after being received by the peripheral node (bordercast destination). This could result in wasted transmissions as a query penetrates into a previously queried region. Implementing bordercasting in the ZRP allows the protocol to provide an Early Termination (ET), as the redundant query enters the previously queried region.

2.2.4 Route Maintenance

In addition to initially discovering routes, the IERP may also assume responsibility for monitoring the integrity of these routes and repairing failed routes as appropriate.

Route failures can be detected either proactively or reactively. Proactive route failure detection is triggered by the IARP, in response to a node leaving the routing zone. Any IERP routes containing this node as the first hop can be considered invalid. Route failures may also be detected reactively, by IP, when the next hop in a datagram’s source route is determined to be unreachable (i.e. does not appear in the (Intrazone) Routing Table).

Upon detection of a route failure, a node may choose to notify the route’s source of the failure and / or attempt to repair the route. A route failure notification consists of a transmission back to the query source, indicating that the source route has failed, and possibly the hop at which it failed. This type of service is provided by protocols like ICMP. The node that detects the route failure may also try to repair the failed connection by discovering a route to bypass the failed connection. The repair discovery process is nearly identical to an initial route discovery. Route repairs should not be substantially longer than the original failed connection. Thus, the depth of a repair query can be limited, through the use of hop counter. This has the advantage of producing much less query traffic than an unrestricted initial route query. After a successful repair, the route’s source MAY be notified so that routes are properly selected for use. Alternatively, the repair could go unreported without compromising the connectivity between source and destination.
3.0 The ZRP Architecture

Legend:

A <---> B  exchange of packets between protocols A & B
A ===> B  information passed from protocol A to protocol B

Existing Protocols
-------------
IP  Internet Protocol
ICMP  Internet Control Message Protocol

ZRP Entities
------------
IARP  IntrAzone Routing Protocol
IERP  IntErzone Routing Protocol
BRP  Bordercast Resolution Protocol
     (component of IERP)

Additional Protocols
---------------------
NDM  Neighbor Discovery/Maintenance Protocol

Note, it is assumed that basic neighbor discovery operation is implemented by the MAC/link-layer protocols. Thus the NDM protocol remains unspecified here.

4. Implementation Details

4.1 The IntrAzone Routing Protocol (IARP)

The Intrazone Routing Protocol (IARP) is responsible for proactively maintaining routes within each node’s routing zone. This can be achieved
through a variety of different implementations. In fact, many
traditional proactive protocols can be modified to serve as an IARP by
limiting the range of route updates to the boundary of routing zones.

In this draft, we detail implementations for both a Distance-Vector
and a Link-State IARP. The convergence problems typically associated
with
the Distance-Vector approach are less of a liability in the IARP
implementation because of the finite hop count imposed by the routing
zone
radius. Additionally, techniques such as "hold-downs", "split horizons"
and "poison reverse" can be used to prevent instabilities. A stable
Distance Vector IARP implementation converges a rate comparable to
Link-State IARP, and generally with less control traffic and reduced
computational overhead. However, the Link-State IARP provides a complete
view of the routing zone topology, making it a favorable option for some
applications (including "on-line" routing zone re-sizing).

4.1.1 Distance Vector (with split horizon) IARP

In this version of the IARP, exchanged routing messages consist of the
route cost (including hop count) and the IP addresses of the route’s
destination and next TWO hops to the destination. The second hop
is included to identify routes where a node is it’s own second hop.
This particular looping condition result in the "counting to
infinity" problem. By deleting these routes, this problem can be
avoided, allowing the IARP to converge faster.

A node may receive new route information either from a received IARP
packet or from an interrupt generated by its Neighbor Discovery /
Maintenance (NDM) process*. In the special case when a host has
discovered a neighbor, the IARP is responsible for sending to the new
neighbor the shortest route to each host which is common to both of
their routing zones (i.e. each host with a hop count less than it’s
routing zone radius). The node then records the new route information
in its Intrazone Routing Table. If the shortest path to the host has
changed, the terminal broadcasts an IARP packet reflecting the
change.

* IARP relies on the services of a separate protocol (referred to
here as the Neighbor Discovery/Maintenance Protocol) to provide
current information about a host’s neighbors. At the least, this
information must include the IP addresses of all the neighbors.
The IARP can be readily configured to support supplemental
link quality metrics.

Haas, Pearlman                Expires December 1999                   [Page
9]
INTERNET DRAFT              The Zone Routing Protocol                June
1999

A. Packet Format

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destination Address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destination Subnet Mask (Optional)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Next Hop #1 Address</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Field Description:

* Destination Address (32 bits)
  IP address of the destination host

* Destination Subnet Mask (32 bits)
  IP subnet mask associated with the destination

* Next Hop # 1 Address (32 bits)
  IP address of the "next hop" host to the destination host

* Next Hop # 2 Address (32 bits)
  IP address of the Next Hop #1 ‘s "next hop" host to the destination host

* Node/Link Metrics (X * 32 bits)
  This section of the packet is used to report the quality of this link (or link source node).

  * Metric Type (8 bits)
    Type of metric being reported
    (based on pre-defined metric types)

  * Metric Value (16 bits)
    Value of node / link metric specified by Metric Type

  * Hop Count (8 bits)
    Length of the route to the destination host, measured in hops

Haas, Pearlman Expires December 1999
INTERNET DRAFT The Zone Routing Protocol June 1999

B. Data Structures

B.1 Intrazone Routing Table

<table>
<thead>
<tr>
<th>Dest Addr</th>
<th>Subnet Mask</th>
<th>Routes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>=&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>=&gt;</td>
</tr>
</tbody>
</table>
C. Interfaces

C.1 Higher Layer (N/A)
C.2 Lower Layer (IP)
   C.2.1 Send() (specified in IP standard)
   C.2.2 Deliver() (specified in IP standard)

C.3 NDM
   C.3.1 Neighbor_Lost(host,mask,metric)
       Used by the NDM to notify the IARP that direct contact
       has been lost with "host" (with subnet mask "mask").
       Any node/link quality measurements are reported in
       the "metric" list.
   C.3.2 Neighbor_Found(host,mask,metric)
       Used by the NDM to notify the IARP that direct contact
       has been confirmed with "host" (with subnet mask "mask").
       Any node/link quality measurements are reported in
       the "metric" list.

C.4 IERP
   C.4.1 Zone_Node_Lost(host)
       Used by IARP to notify the IERP that a node no longer
       exists within the routing zone.

D. State Machine

The IARP protocol consists of only one state (IDLE). Therefore,
no state transitions need to be specified. The IARP immediately
acts upon an event and then returns back to the IDLE state.

Notes:  1) X is used as a label for the host running this state
        machine.
        2) INF is a reserved field value corresponding to
           "infinity".

D.1

Event: An IARP packet is received containing route
        information to a destination D. The hop count
        associated with the received route is LESS THAN
        OR EQUAL TO the routing zone radius. The
        second next-hop is EQUAL to X.

Action: NONE
Event: An IARP packet is received containing route information to a destination D. The hop count associated with the received route is LESS THAN the routing zone radius. The second next-hop is NOT EQUAL to X.

Action: The received route is recorded in the Intrazone Routing Table. If the received route is shorter than the previous shortest route to D, then a new IARP packet containing route information to D through X is broadcasted.

D.3
Event: An IARP packet is received containing route information to a destination D. The hop count is EQUAL TO the routing zone radius. The second next-hop is NOT EQUAL to X.

Action: The received route is recorded in the Intrazone Routing Table.

D.4
Event: An IARP packet is received containing route information to a destination D. The hop count is equal to INF.

Action: The route to D is removed from the Intrazone Routing Table.
1) If the Intrazone Routing Table still contains a route to D and the length of the shortest route has increased due to the route removal, then the an IARP packet containing the shortest route to D through X is broadcasted.
2) If the Intrazone Routing Table contains no more routes to D, then an IARP packet containing a route to D through X with hop count of INF is broadcast. A "Host Lost" interrupt is generated to alert the IERP that D has moved beyond the routing zone.

D.5
Event: A "Neighbor Found" interrupt is received, indicating the discovery of a neighbor host N.

Action: For each destination in X’s Intrazone Routing Table, an IARP packet is sent to N containing the best route to that destination. An IARP packet is then broadcasted containing the 1 hop route to N through X.

D.6
Event: A "Neighbor Lost" interrupt is received, indicating that host N is no longer a neighbor of X.

Action: The one hop route to N is removed from the
Intrazone Routing Table.

1) If the Intrazone Routing Table still contains a route to N and the length of the shortest route has increased due to the route removal, then the an IARP packet containing the shortest route to N through X is broadcasted.

2) If the Intrazone Routing Table contains no more routes to N, then an IARP packet containing a route to D through X with hop count of INF is broadcast. A "Host Lost" interrupt is generated to alert the IERP that D has moved beyond the routing zone.

Haas, Pearlman                Expires December 1999                  [Page 13]
INTERNET DRAFT                The Zone Routing Protocol                June 1999

E. Pseudocode Implementation

We define two complimentary operations on packets: extract(packet) and load(packet)

extract (packet)
  extracts the fields of the IARP packet to the following variables: (dest, mask, next_hop_1, next_hop_2, route_metric, hop_count)

load (packet)
  loads the values of the aforementioned variables into the fields of the IARP packet.

E.1 Update Intrazone Routing Table

if (packet arrived)
  extract(packet)
else
  {
    {dest,mask,metric} <-- intrpt
    next_hop_1=dest
  if (type(intrpt) == "Neighbor Found")
  {
    for d = each node in Intrazone_Routing_Table
    {
      best_route = get_shortest_route(Intrazone_Routing_Table, d)
      mask = get_mask(Intrazone_Routing_Table, d)
      if (best_route->hop_count < ROUTING_ZONE_RADIUS)
      {
        packet<--{d,mask,MY_ID,best_route->next_hop, best_route->metric,best_route->hop_count+1}
        send(packet,dest)
      }
      hop_count=1
    } else
      hop_count=INF
  }
former_best_route = get_shortest_route(Intrazone_Routing_Table, dest)

if (hop_count < INF)
{
    if (next_hop_2 != MY_ID)
    {
        link_metric = get_metric(Intrazone_Routing_Table, next_hop_1)
        route_metric = add_metric(route_metric, link_metric)
        add(Intrazone_Routing_Table, {dest, mask, next_hop_1,
                                        route_metric, hop_count})
    }
}
else
remove(Intrazone_Routing_Table, {dest, next_hop_1})

best_route = get_shortest_route(Intrazone_Routing_Table, dest)
if (best_route != NULL)
{
    if (best_route->hop_count != former_best_route->hop_count
        (AND) best_route->hop_count < ROUTING_ZONE_RADIUS)
    {
        packet <-- {dest, mask, MY_ID, best_route->next_hop,
                    best_route->metric, best_route->hop_count+1}
        send(packet, BROADCAST)
    }
}
else
{
    force_intrpt("IERP", "Zone Node Lost", {dest})
    packet <-- {dest, mask, MY_ID, MY_ID, NULL, INF}
    send(packet, BROADCAST)
}

4.1.2 Link State IARP

In this version of the IARP, nodes compute intrazone routes based on the link state (neighbor connectivity) of each routing zone node. A node may receive link state updates either from an IARP link state packet or from an interrupt generated by its Neighbor Discovery / Maintenance (NDM) process*. Link states are maintained in a Link State Table. When all pending link state updates have been received (full link state updates may contain multiple links and span multiple packets), the routing table is recomputed, using a minimum spanning tree algorithm. The Link State Table is then updated to remove links that lie outside of the routing zone. All new link state updates for non-peripheral routing zone nodes are forwarded to all neighbors. In addition, if a new neighbor is discovered, the new neighbor is sent the FULL link states of all non-peripheral routing zone
nodes.

* IARP relies on the services of a separate protocol (referred to here as the Neighbor Discovery/Maintenance Protocol) to provide current information about a host’s neighbors. At the least, this information must include the IP addresses of all the neighbors. The IARP can be readily configured to support supplemental link quality metrics.

A. Packet Format

```
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Link Source Address                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Link Destination Address                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Packet Source Address                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Link State ID                          |
|                     Zone Radius                             |
|                     Flags                                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Link Destination Subnet Mask (Optional)                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Metric Type         |          Metric Value         |
|         Metric Type         |          Metric Value         |
|         Metric Type         |          Metric Value         |
|         Metric Type         |          Metric Value         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   RESERVED   |  Metric Type  |          Metric Value         |   |
|   RESERVED   |  Metric Type  |          Metric Value         |   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+   |
|   RESERVED   |  Metric Type  |          Metric Value         |   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ --+--
```

Field Description:

* Link Source Address  (32 bits)
  IP address of the reported link’s source node.

* Link Destination Address  (32 bits)
  IP address of the reported link’s destination node.

* Packet Source Address  (32 bits)
  IP address of the node that sent this packet.
  (Used to account for any outstanding link state information)

* Link State ID  (16 bits)
  Sequence number used to track the link state history of the Link Source node.

* Zone Radius  (8 bits)
  Routing zone radius of the link’s source node. Determines the extent that link state information propagates.
* Flags                         (8 bits)
  Flags(0)   Full Link Information
  Indicates whether this link state information was sent as:
  (0) a PARTIAL link state update
  OR
  (1) part of a FULL update of all the
       link source’s current links

  Flags(1)   Current Link State Update
  Indicates whether more link state information is pending
  for THIS link state update (link_source, state_id)
  (0) INCOMPLETE
  (1) COMPLETE

  Flags(2)   All Link State Updates
  Indicates whether more link state updates are pending
  (0) INCOMPLETE
  (1) COMPLETE

  Flags(3)   Link Destination Subnet Mask
  (0) INCLUDED
  (1) NOT INCLUDED

  Flags(4)   Link Status
  (0) DOWN
  (1) UP

  Flags(5..7)  RESERVED for future use.

Haas, Pearlman                  Expires December 1999                  [Page 17]
INTERNET DRAFT              The Zone Routing Protocol                June 1999

* Node/Link Metrics             (X * 32 bits)
  This section of the packet is used to report the quality of
  this link (or link source node).

  * Metric Type           (8 bits)
    Type of metric being reported
    (based on pre-defined metric types)

  * Metric Value          (16 bits)
    Value of node / link metric specified by Metric Type

B.  Data Structures

B.1 Intrazone Routing Table

+---------+--------+-----------------------------------------+
|  Dest   | Subnet |                Routes                   |
|  Addr   |  Mask  |-----------+-----------+-----+-----------+
|         |        |     0     |     1     | ==> |    M-1    |
+---------+--------+-----------+-----------+-----+-----------+
|         |        |           |           | ==> |           |
|---------+--------+-----------|-----------|-----|-----------|
|         |        |           |           | ==> |           |
|---------+--------+-----------|-----------|-----|-----------|
|         |        |    | |    |           | ==> |           |
+---------+--------+----| |----+-----------+-----+-----------+
B.2 Link State Table

<table>
<thead>
<tr>
<th>Link Source</th>
<th>Zone Radius</th>
<th>Link State ID</th>
<th>Link State Information</th>
</tr>
</thead>
</table>
+-------------+-------------+---------------+------------------------+
|             |             |              |                        +--|
|             |             |              |                        |   |   |...|   |   | | |
|             |             |              |                        |   |   |   |   |   |
** The first link state entry for each link source contains the complete link state information corresponding to the Link State ID. Subsequent entries contain only changes of a single link state.

A FULL link state entry of link state #k and a PARTIAL entry of link state #(k+1) can be merged into a FULL link state entry of link state #(k+1)

**

B.3 Pending Link State List
(list of neighbors in the process of sending link state updates)

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(a) Link Destination Address
(b) Link Destination Subnet Mask
(c) Link Metrics
(d) Forward Flag (indicates if link state information has been forwarded)
B.4 New Neighbors List  
(list of new neighbors to receive a copy Link State Table, 
upon completion of updates) 

+----------+      +----------+        +----------+  
|          | ---> |          |  . . . |          |  (node addresses)  
|          |      |          |        |          |  
|          |      |          |        |          |  
|          |      |          |        |          | 

B.5 Former Routing Zone Nodes  
(list of routing zone nodes, prior to routing table updates) 

+----------+      +----------+        +----------+  
|          | ---> |          |  . . . |          |  (node addresses)  
|          |      |          |        |          |  
|          |      |          |        |          |  
|          |      |          |        |          | 

C. Interfaces  

C.1 Higher Layer (N/A)  
C.2 Lower Layer (IP)  
C.1.1 Send() (specified in IP standard)  
C.1.2 Deliver() (specified in IP standard)  
C.3 NDM  
C.3.1 Neighbor_Lost(host,mask,metric) 
Used by the NDM to notify the IARP that direct contact 
has been lost with "host" (with subnet mask "mask"). 
Any node/link quality measurements are reported in 
the "metric" list.  
C.3.2 Neighbor_Found(host,mask,metric) 
Used by the NDM to notify the IARP that direct contact 
has been confirmed with "host" (with subnet mask "mask"). 
Any node/link quality measurements are reported in 
the "metric" list.  
C.4 IERP  
C.4.1 Zone_Node_Lost(host) 
Used by IARP to notify the IERP that a node no longer 
exists within the routing zone.  

D. State Machine  

The IARP protocol consists of only one state (IDLE). Therefore, 
no state transitions need to be specified. The IARP immediately 
acts upon an event and then returns back to the IDLE state.  

Notes: 1) X is used as a label for the host running this state 
        machine.  
        2) INF is a reserved field value corresponding to 
           "infinity".  

D.1  
Event: An IARP link state packet is received.  
Action: The link state update is recorded in the Link State 
        Table.  
If more link state updates are pending, then the IARP
returns to an idle state. Otherwise, X rebuilds its routing table. Links that lie outside of the routing zone are removed from the Link State Table. X sends its neighbors all previously unforwarded link state updates from its NON-peripheral nodes. Finally, all neighbors in the New Neighbor List are sent the link states of X’s NON-peripheral nodes, and the New Neighbor List is cleared.

### D.2

**Event:** A "Neighbor Found" interrupt is received, indicating the discovery of a neighbor N.

**Action:** X’s new link to N is recorded in the Link State Table and N is added to the New Neighbors List. If more link state updates are pending, then the IARP returns to an idle state. Otherwise, X rebuilds its routing table. Links that lie outside of the routing zone are removed from the Link State Table. X sends its neighbors all previously unforwarded link state updates from its NON-peripheral nodes. Finally, all neighbors in the New Neighbor List are sent the link states of X’s NON-peripheral nodes, and the New Neighbor List is cleared.

### D.3

**Event:** A "Neighbor Lost" interrupt is received, indicating that node N is no longer a neighbor of X.

**Action:** The lost link to neighbor N is removed from the Link State Table and N is removed from the New Neighbor List. If more link state updates are pending, then the IARP returns to an idle state. Otherwise, X rebuilds its routing table. Links that lie outside of the routing zone are removed from the Link State Table. X sends its neighbors all previously unforwarded link state updates from its NON-peripheral nodes. Finally, all neighbors in the New Neighbor List are sent the link states of X’s NON-peripheral nodes, and the New Neighbor List is cleared.

### E. Pseudocode Implementation

We define two complimentary operations on packets: 

- `extract(packet)`
- `load(packet)`
extract (packet)
extracts the fields of the IARP packet to the following
variables: (link_source, link_dest, pk_source, state_id,
radius, flags, mask, link_metric)
* full_link_state -> flag(0)
* current_update -> flag(1)
* all_updates    -> flag(2)
* mask           -> flag(3)
* link_status    -> flag(4)

load (packet)
loads the values of the aforementioned variables into
the fields of the IARP packet.

E.1 Update Intrazone Routing Table

if (packet arrived)
{
    extract(packet)
    my_link_changed = FALSE
}
else
{
    {link_dest,mask,link_metric} <-- intrpt
    link_source = MY_ID
    pk_source  = MY_ID
    state_id   = MY_LINK_STATE_ID
    radius     = MY_ROUTING_ZONE_RADIUS

    full_link_state = 0
    current_update = COMPLETE
    all_updates    = COMPLETE

    if (type(intrpt) == "Neighbor Found")
        link_status = UP
    else
        link_status = DOWN

    my_link_changed = TRUE
}

add(Pending_Link_State_List, link_from_id)
status = add(Link_State_Table,link_source, link_dest, mask, radius,
link_metric, state_id, flags)
if (status == NEW_LINK_INFO)
{
    cum_status = UPDATE_IN_PROGRESS
    if (my_link_changed)
    {
        if (link_status == UP)
            add(New_Neighbor_List, link_dest)
        else
            remove(New_Neighbor_List, link_dest)
        MY_LINK_STATE_ID++
    }
}
if(all_updates == COMPLETE)
    remove(Pending_Link_State_List, link_from_id)

if(empty(Pending_Link_State_List) (AND)
    cum_status == UPDATE_IN_PROGRESS)
{
    for(each node (BELONGING TO) Intrazone_Routing_Table)
        add(Former_Routing_Zone_Nodes, node)

    rebuild = TRUE;
    while (rebuild)
    {
        Intrazone_Routing_Table
        =
        construct_spanning_tree(Link_State_Table);
        rebuild = update(Link_State_Table);
    }

    broadcast_link_state_updates(Link_State_Table);

    for (each node (BELONGING TO) New_Neighbor_List)
        send_link_state_table(Link_State_Table, node);

    clear(New_Neighbor_List)

    cum_status = UPDATE_COMPLETE;

    for (each node (BELONGING TO) Former_Routing_Zone_Nodes)
    {
        if(node !(BELONGS TO) Intrazone_Routing_Table)
            force_intrpt("IERP","Zone Node Lost",{node})
    }

    clear(Former_Routing_Zone_Nodes)
}

4.2 IntErzone Routing Protocol (IERP)

The Interzone Routing Protocol (IERP) is responsible for discovering
and maintaining routes to hosts which are beyond a node’s routing
zone. The IERP acquires routing information reactively, exploiting
the knowledge of the routing zone topology through the bordercasting
delivery service.

This version of the IERP extends the basic query / reply mechanism
described in Section 3. In the basic protocol, queries are
terminated before reaching the query destination. This provides
a faster response to the route query than if the destination, D,
were to respond directly. However, because D never receives the query, it does not acquire a route back to the source, S. If the D needs to send data back to S (which is often the case, given the bi-directional nature of many applications), a separate route query from D to S would have to be initiated. This substantial overhead is avoided by having the query forwarded to D, by the node Y that discovers D in its routing zone. We refer to this as a QUERY_EXTENSION.

Both the source and destination can acquire routes to each other through the BRP route accumulation mechanisms (to be discussed later). Distributing route information in the caches of the route’s nodes can significantly reduce the size of the IERP packets and provide for a faster query response. On the other hand, QoS requirements for a particular application may favor a source specified route (rather than a distributed next-hop approach). Source routing requires that the discovered route information be accumulated in the IERP packets, rather than cached. This IERP implementation satisfies the demands for rapid query response and source routing support by two stages of route reporting. In the first two stages, route information is cached by the route’s nodes (when possible). Next-hop route are quickly returned to the source in ROUTE_REPLY packets and forwarded to the destination in QUERY_EXTENSION packets. At this point, bi-directional connectivity is established. In the third stage, the source and destination can provide each other with complete source routes, by sending each other ROUTE_ACCUMULATION packets. As these packets traverse the route, the cached route information is accumulated in the packets, thereby constructing the desired source route.

Variations on this IERP implementation can be realized by combining or eliminating some of these stages. For example, if source routing is not desired, the ROUTE_ACCUMULATION stage can be eliminated. Also, if all of the route’s nodes elect not to cache the routing information (perhaps due to storage limitations), then the ROUTE_REPLY and QUERY_EXTENSION packets essentially serve the role of ROUTE_ACCUMULATION packets, obviating the need for a separate ROUTE_ACCUMULATION stage.

Because each node proactively maintains the local topology of its routing zone, it is not necessary for a source route to specify every hop along the route (i.e. strict source routing). A properly chosen subset of the complete source route can be used to specify the route to the destination, and where desirable, the reverse route back to the source. The IERP supports such an optimization through a final ROUTE_OPTIMIZATION stage. The details of the route optimization are described in the BRP specification. Upon completion of the ROUTE_OPTIMIZATION stage, sufficient routing zone membership is acquired for each node in the route so that the source route may be reduced (by translating this route reduction into an appropriate set covering problem, and employing a suitable heuristic).

+-----+                    +-----+       +-----+
|  S  |      . . . .       |  Y  | . . . |  D  |
Sequence of the IERP Route Discovery Stages

Haas, Pearlman                      Expires December 1999                  [Page 25]
INTERNET DRAFT                   The Zone Routing Protocol                June
1999

A.  Packet Format

```
+-----+               +-----+                 +-----+
| 1   | 2            | 3    |
| 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 |
+-----------------------------------------------
| Type | TTL | Hop Count | Flags |
+-----------------------------------------------
| Current | ROF Ptr | Num Dests = D | Num Nodes = N |
|-----------------------------------------------
| Query ID | Reply ID |
+-----------------------------------------------
| Query/Route Source Address |
+-----------------------------------------------
| Replying Node Address |
+-----------------------------------------------
<table>
<thead>
<tr>
<th>Bad Link Source Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query/Route Destination (1) Address</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>Query/Route Destination (2) Address</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>Query/Route Destination (D) Address</td>
</tr>
</tbody>
</table>
+-----------------------------------------------|
| Next IERP Address |
+-----------------------------------------------
| Next BRP Address |
| Prev IERP Address |
| Intermediate Node (1) Address |
```

(1) search for route
(2) establish connectivity
(3) provide source route (OPTIONAL)
(4) optimize source route (OPTIONAL)
Field Description:

* Type (8 bits)
  Identifies the type of IERP packet. The current version of IERP contains five packet types:

  **ROUTE_QUERY:**
  request for an Interzone source route to the destination specified by the Destination IP Address

  **QUERY_EXTENSION:**
  extension of a QUERY packet sent from the node that discovers the Destination to the Destination itself.
ROUTE_REPLY:
response to a ROUTE_QUERY packet, sent from the node that discovers the Destination back to the Source. 
At a minimum, the ROUTE_REPLY contains next-hop route information to the Destination. 
(In general, returns the source route up to the first node which has cached routing information. If no nodes have cached routing information, then the complete source route is returned.)

ROUTE_ACCUMULATION (optional):
sent by the Source to the Destination, in response to a ROUTE_REPLY, and sent by the Destination to the Source, in response to a QUERY_EXTENSION. Routing information cached at the route’s nodes is accumulated in this packet, providing a complete source route at the destination of this packet.

ROUTE_OPTIMIZATION (optional):
sent by the Source to the Destination, and by the Destination to the Source, in response to a ROUTE_ACCUMULATION packet. Route Optimization Flags (ROF) are appended to this packet, reflecting the mutual routing zone membership of each node in the source route.

* TTL (Time to Live) (8 bits)
Number of hops that a route query may continue to propagate. This field allows a querying node to configure the depth of a route search, in order to control the amount of IERP traffic generated.

* Hop Count (8 bits)
Hop count from source to current node (ROUTE_QUERY, QUERY_EXTENSION) or hop count from source to route destination (ROUTE_REPLY, ROUTE_ACCUMULATION, ROUTE_OPTIMIZATION).

* Flags (8 bits)
Flags(0) ANY destination (0) / ALL destination (1)
In the case of multiple destinations, specifies whether the ROUTE_QUERY should return routes for ANY specified destination, or ALL specified destinations.
In the case of a single MULTICAST IP address, specifies whether the ROUTE_QUERY should return routes to ANY member of the multicast group, or ALL members of the multicast group.
Flags(1)       Passed Bad Link Source
    In the case of a "route repair", indicates whether the
    ROUTE_REPLY has passed the Bad Link Source node.

Flags(2..7)     RESERVED for future use

* Current (8 bits)
    INDEX of the route (see below) corresponding to the route
    node that has most recently received this packet. (the first
    node in the route has an index of 1).

* ROF Pointer (8 bits)
    Pointer to the starting location of the Route Optimization
    Flags (ROUTE_OPTIMIZATION phase). The ROF Pointer is
    measured in units of 32 bit words from the front of the packet.

* Number of Destinations = D (8 bits)
    Number of destinations included in the route query/reply
    packet. This allows multiple route discoveries to be consolidated
    into a common route query process. The multiple query
    destinations feature is particularly useful for routing to
    a multicast group with known members, or for locating
    downstream nodes during the route repair phase.

Haas, Pearlman                Expires December 1999                  [Page 28]
INTERNET DRAFT              The Zone Routing Protocol                June 1999

* Number of Nodes = N (8 bits)
    Number of nodes IP addresses appearing in the route
    specification.

* Query ID (16 bits)
    Sequence number which, along with the Query Source Address
    (see below) uniquely identifies any ROUTE_QUERY in the
    network.

* Reply ID (16 bits)
    Sequence number which, along with the Replying Node Address
    (see below) uniquely identifies any ROUTE_REPLY in the
    network.

* Query/Route Source Address (32 bits)
    IP address of the node that initiates the ROUTE_QUERY.
    In subsequent stages, this corresponds to the IP address of
    the discovered route’s source node.

* Replying Node Address (32 bits)
    IP address of the node that initiates the ROUTE_REPLY. Note
    that this is usually NOT the destination node.
- Bad Link Source Address (32 bits)
  Used during route repairs. Contains the IP Address corresponding to the source of routes failed link.

- Query/Route Destination Addresses (D * 32 bits)
  List of IP addresses to be located during the ROUTE_QUERY phase.
  (Either ANY or ALL addresses, depending on the setting of Flags(0))
  In subsequent stages, this field contains the IP address of the discovered route’s (single) destination node.

- Next IERP Address (32 bits)
  IP address of the next IERP recipient

- Next BRP Address (32 bits)
  IP address of the next BRP recipient.
  (i.e. next hop to the next IERP recipient)

- Prev IERP Address (32 bits)
  IP address of the previous IERP recipient

- Route (N * 32 bits)
  Variable length field that contains the recorded IP addresses of nodes along the path traveled by this ROUTE_QUERY packet from the Query Source.
  In subsequent stages (after a route to a Query Destination has been discovered), this set of IP addresses provides a partial specification of the route between the Route Source and Route Destination.

* Node/Segment Metrics (X * 32 bits)
  This optional section of the packet can be used to record a variety of node and segment quality measurements.
  (In this context, a segment refers to the communication path between consecutive nodes in the packet’s accumulated route.
  In the case of neighboring nodes, a segment is equivalent to a (one-hop) link).

  * Node (8 bits)
    Index of the route corresponding to a particular node.

  * Metric Type (7 bits)
    Type of metric being recorded
    (based on pre-defined metric types)

  * Direction Flag (1 bit)
    For segment metrics, specifies either the upstream segment (toward Query/Route Source) or the downstream
**segment (toward Query/Route Dest).**

\[
\begin{align*}
\text{upstream} &= 0 \\
\text{downstream} &= 1
\end{align*}
\]

* Metric Value (16 bits)

Value of node / segment metric specified by Metric Type

* ROF (Route Optimization Flags) \(((N+2) \times 32 \cdot \text{ceil}((N+2)/32))\) bits

The k-th bit of the n-th ROF subfield indicates whether Node k is located within Node n’s routing zone. This routing zone membership information, collected during the optional ROUTE_OPTIMIZATION stage, may be used to determine the shortest possible specification for the accumulated source route.

---

**Haas, Pearlman**

Expires December 1999

[Page 30]

INTERNET DRAFT

The Zone Routing Protocol

1999

B. Data Structures

B.1 Intrazone Routing Table (See IARP Description)

B.2 Interzone Routing Table

```
+---------+-----------------------------------------+ 
|         |                 Routes                  | 
| Dest    +-----------+-----------+-----+-----------+ 
<p>|         |     0     |     1     | ==&gt; |    M-1    |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>==&gt;</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>==&gt;</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----</td>
<td>-----------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>----</td>
<td></td>
<td>----+-----------+-----+-----------</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>/</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>/</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>/</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>/</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>/</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>/</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
|         | +---\  +------|---|--+--|+   +--+
|         | +-----/      |   |  |  |...|  | --+   (node 1) 
+---------+----| |----+-----------+-----+-----------+
```

(a) Node ID
(b) Required Node Flag (for Route Optimization)
(c) Node/Link Metric
C. Interfaces

C.1 Higher Layer (N/A)
C.2 Lower Layer (BRP)
  C.2.1 Send() (same interface as IP send())
      Used by the IERP to request transmission of an
      IERP packet.
  C.2.2 Deliver() (same interface as IP deliver())
      Used by the BRP to alert the IERP of the arrival of a
      data packet.

C.3 IARP
  C.3.1 Zone_Node_Lost(host)
      Used by the IARP to notify the IERP that a node has left
      the routing zone

C.4 ICMP
  C.4.1 Host_Unreachable() (specified in ICMP standard)
  C.4.2 Source_Route_Failed() (specified in ICMP standard)

Haas, Pearlman                Expires December 1999                  [Page
31]
INTERNET DRAFT              The Zone Routing Protocol       June
1999

D. State Machine

The IERP protocol consists of only one state (IDLE). Therefore,
no state transitions need to be specified. The IERP immediately
acts upon an event and then returns back to the IDLE state.

Note: 1) X is used as a label for the host running this state
machine.

D.1
  Event: A "Zone_Node_Lost" interrupt is received,
      indicating that the node H has moved beyond the
      routing zone.
  Action: Remove every route from the Interzone Routing Table
      which contains H. If any routes containing H are
      found, then a route repair (limited depth route
      discovery) to H is initiated.

D.2
  Event: A "Host_Unreachable" interrupt is received from the
      ICMP, indicating an unknown destination host D.
  Action: A full depth route discovery to D is initiated.
      X’s query_id is incremented and assigned to a new
      ROUTE_QUERY packet. The route is initialized with the
      IP addresses of the route’s source and destination
      IP addresses (X and D). Finally, the ROUTE_QUERY
      packet
      is bordercasted.

D.3
  Event: A "Source_Route_Failed" or "Proactive_Repair"
      interrupt is received, indicating that the next
      hop, H, specified in a source route does not
      appear within the routing zone.
A limited depth route discovery to H is initiated. The query_id is incremented and assigned to a new ROUTE_QUERY packet. The route is initialized with the valid portion of the failed route, and the bad link address field is set with X’s IP address, to indicate the location of the route failure. Finally, the ROUTE_QUERY packet is bordercasted.

D.4

Event: A ROUTE_QUERY packet is received with a destination that appears within X’s routing zone.

Action: X copies the ROUTE_QUERY packet’s contents to a ROUTE_REPLY, labelling it with its IP address and incremented reply_id. A QUERY_EXTENSION is sent to the query destination.

D.5

Event: A ROUTE_QUERY packet is received with a destination that DOES NOT appear within X’s routing zone.

Action: The ROUTE_QUERY packet is bordercasted.

D.6

Event: A QUERY_EXTENSION packet is received.

Action: The packet contents are moved to a ROUTE_ACCUMULATION packet. The ROUTE_ACCUMULATION packet is sent to the query source.

D.7

Event: A ROUTE_REPLY packet is received.

Action: The packet contents are moved to a ROUTE_ACCUMULATION packet. The ROUTE_ACCUMULATION packet is sent to the query destination.

D.8

Event: A ROUTE_ACCUMULATION packet is received. X is not the final destination of this packet (i.e. X != IERP_next). This only occurs when the accumulated route contains a repair.

Action: The bad link is replaced by the path repair in the Interzone Routing Table.

D.9

Event: A ROUTE_ACCUMULATION packet is received. X is either the route source or (route destination).

Action: The (reversed) accumulated route is added to the Interzone Routing Table or replaces a failed route if the packet specifies a bad link. In addition, if X is the ROUTE_ACCUMULATION destination, the packet contents may be moved to a ROUTE_OPTIMIZATION packet.
packet, which is then sent to the query destination (query source).

D.10
Event: A ROUTE_OPTIMIZATION packet is received.
Action: The routing zone membership information that is collected in the ROUTE_OPTIMIZATION packet is processed. The resulting optimized route(s) are added to the Interzone Routing Table.

E. Pseudocode Implementation

We define two complimentary operations on packets: extract(packet) and load(packet)

extract (packet)
extracts the fields of the IERP packet to the following variables: {type, TTL, hop_count, flags, current_hop_ptr, query_id, reply_id, source, reply_node, bad_link_source, dests, next_IERP, next_BRP, prev_IERP, route, metric, ROF}

load (packet)
loads the values of the aforementioned variables into the fields of the IERP packet.
load(packet) automatically computes the values of:
num_dests = |dests|
num_nodes = |routes|

E.1 Routing Zone Node Lost

{lost_host} <-- intrpt
repair_link = FALSE
for host = each host in Interzone Routing Table
{
  for (route = each Interzone route to host)
  {
    if (lost_host (EXISTS IN) route)
    {
      if (PROACTIVE_REPAIRS_ENABLED)
      {
        removal_timer = ROUTE_QUERY_TIMEOUT
        repair_link = TRUE
      }
      else
      {
        removal_timer = 0
        schedule(
          remove(Interzone_Routing_Table[host]->route(m)),
          removal_timer)
      }
    }
  }
}
if(repair_link)
{
dests = lost_host
force_intrpt("IERP","Proactive_Repair",{MY_ID,dests,ALL})

E.2 Initiate Route Discovery

{source,dests,flag} <-- intrpt
if (type(intrpt) == "Proactive_Repair" (OR)
    type(intrpt) == "Source_Route_Failed")
{
    TTL = MAX_REPAIR_HOPS
    bad_link_source = MY_ID
}
else
{
    TTL = MAX_FULL_QUERY_HOPS
    bad_link_source = NULL
}
query_id = MY_QUERY_ID++
load (packet)
send (packet, BORDERCAST)

E.3 Receive IERP Packet

extract(packet)
switch(type)
{
    case: ROUTE_QUERY
        if (dest (EXISTS IN) Intrazone_Routing_Table)
        {
            type = ROUTE_REPLY
            reply_id = MY_REPLY_ID++
            if(bad_link_source)
                IERP_next = bad_link_source
            else
                IERP_next = source
            load(packet)
            send(packet,IERP_next)
            type = QUERY_EXTENSION
            IERP_next = dest
            load(packet)
            send(packet,IERP_next)
        }
        else
            send(packet,BORDERCAST)
    break
    case: QUERY_EXTENSION
        type = ROUTE_ACCUMULATION
        IERP_next = source
        load(packet)
        send(packet,IERP_next)
    break
    case: ROUTE_REPLY
type = ROUTE_ACCUMULATION
IERP_next = dest
load(packet)
send(packet, IERP_next)
break

case: ROUTE_ACCUMULATION
       if (bad_link_source)
           {
             if(bad_link_source != source)
                repair_src_ptr = get_index(route, bad_link_source)
             else
                repair_src_ptr = 0

             bad_link    = {bad_link_source,dest}
             path_repair = {bad_link_source,
                            route(repair_src_ptr+1:|route|),
                            dest}
             replace_link(Interzone_Routing_Table,IERP,bad_link,
                          path_repair)
           }

           else
           {
             if (IERP_next == source)
                add(Interzone_Routing_Table, IERP, dest,
                                next_hops,metric)
             if (IERP_next == dest)
                add(Interzone_Routing_Table, IERP, source,
                                reverse(prev_hops),metric)
           }

           if (MY_ID == IERP_next)
           {
             if (MY_ID == source)
                IERP_next = dest
             if (MY_ID == dest)
                IERP_next = source

             type = ROUTE_OPTIMIZATION
             load (packet)
send (packet, IERP_next)
           }
break

case: ROUTE_OPTIMIZATION
       if (MY_ID == source)
           add(Interzone_Routing_Table,IERP,dest,route,NO_METRIC,ROF)
           if (MY_ID == dest)
              add(Interzone_Routing_Table, IERP, source,
                               reverse(route), NO_METRIC, ROF)
break
4.3 Bordercast Resolution Protocol (BRP)

The BRP is a sublayer of the IERP that performs the operations of bordercasting, query control, route accumulation and routing zone labelling, which form the basis for the route discovery procedure.

In this BRP implementation, bordercasting is implemented as a series of unicasted messages to the peripheral nodes. The BRP is able to identify the peripheral nodes based on the information contained in the Intrazone Routing Table. Rather than unicasting to the peripheral node directly through IP, the bordercasted packets are relayed to the peripheral nodes by the BRP layer. IP is used only to send these messages one hop toward the peripheral nodes. This allows the BRP to detect all ROUTE_QUERY packets that are received by that node.

To efficiently terminate the query, the BRP needs to record sufficient information from each detected query. The query’s source and ID, which serve to uniquely identify a query, are added to the Detected Queries Table. In addition, the IP address of the last node to bordercast the query is also recorded in the Detected Queries table. Any threads of this query that are later received from a different bordercasting node are discarded. Each query also contains a hop counter that is decremented at each node. When the counter expires, the packet is discarded.

IERP packets (excluding ROUTE_OPTIMIZATION packets) that are not terminated next undergo a route accumulation procedure. As described earlier, route accumulation is used to construct routes, by recording the IP addresses of a route’s nodes in the IERP packet or local cache. The Detected Queries Table, extended by two columns, serves as a convenient route accumulation cache.

The node begins the route accumulation procedure by adding its IP address to the IERP route. This is followed by the IP addresses of any cached nodes leading to the packet’s destination (the “next hops”). This is sufficient processing for ROUTE_ACCUMULATION packets, where complete source routes are constructed. On the other hand, ROUTE_QUERY, QUERY_EXTENSION and ROUTE_REPLY packets should carry as little of the route as possible. Therefore, if cache space is available, the route accumulation process records the IP addresses of the "previous hops" from the packet’s source, and removes them from the IERP packet.

The final role of the BRP is to contribute to the ROUTE_OPTIMIZATION process by indicating the mutual routing zone membership of the nodes in the source route. This is done by appending a special flag field to the ROUTE_OPTIMIZATION packet. The status of the k-th bit in the flag field indicates whether the k-th hop in the source route is a member of the node’s routing zone. This membership information is eventually processed at the source to determine the smallest set of routing zones that cover the route (and therefore the smallest set of nodes needed to specify this route through loose source routing).
A. Packet Format

See IERP packet format in Section 4.2

B. Data Structures

B.1 Intrazone Routing Table    (see IARP description)
B.2 Interzone Routing Table    (see IERP description)
B.3 Detected Queries Table

+--------------------+--------+
|  Query  |  Query  |  Prev  |
|  Source  |   ID    |  IERP  |
+--------------------+--------+

B.4 Detected Replies Table

+--------------------+
|  Reply  |  Reply  |
|  Node    |   ID    |
+--------------------+

C. Interfaces

C.1 Higher Layer (i.e. IERP)
   C.2.1 Send()       (same interface as IP Send() primitive)
        Used by higher layer protocol to request transmission of
        a data packet
   C.2.2 Deliver()    (same interface as IP Deliver() primitive)
        Used by the BRP to alert the higher layer protocol of
        the arrival of a data packet
C.2 Lower Layer (IP)
   C.2.1 Send()       (specified in IP standard)
   C.2.2 Deliver()    (specified in IP standard)
The BRP protocol consists of only one state (IDLE). Therefore, no state transitions need to be specified. The BRP immediately acts upon an event and then returns back to the IDLE state.

Notes: 1) X is used as a label for the host running this state machine.
2) NULL is a reserved field value, corresponding to an invalid IP address.

D.1 Event: A ROUTE_QUERY packet is received from the IERP.
Action: If X is the query’s source, then the identifying querying information is recorded in the Detected Queries Table. X adds its IP address to the packet’s route. A copy of the packet is sent to the IERP layer of each peripheral node, by way of a BRP transmission to the next hop to that peripheral node.

D.2 Event: A ROUTE_QUERY packet is received from the IP. The hop counter has expired or the query was already detected from another bordercasting node.
Action: The packet is discarded.

D.3 Event: A ROUTE_QUERY packet is received from IP. The hop count has not expired and the query has not been previously detected (or was detected from the same bordercasting node). X is not the intended BRP recipient.
Action: Identifying query information is recorded in the Detected Queries Table. The packet is then discarded.

D.4 Event: A ROUTE_QUERY packet is received from the IP. The hop count has not expired and the query has not been previously detected (or was previously detected from the same bordercasting node). X is the intended BRP recipient, but is not the intended IERP recipient and the query destination does not lie within X’s routing zone.
Action: The hop counter is decremented. Identifying query information is recorded in the Detected Queries Table and accumulated route information is recorded in the Interzone Routing Table. The recorded route information is removed from the packet and X adds its IP address to the route. The packet is then sent to the BRP of the next hop to the intended IERP recipient.
D.5
Event: A ROUTE_QUERY packet is received from the IP. The hop count has not expired and the query has not been previously detected (or was previously detected from the same bordercasting node). X is the intended BRP recipient, and either X is the intended IERP recipient, OR the query destination lies in X’s routing zone.

Action: The hop counter is decremented. Identifying query information is recorded in the Detected Queries Table and accumulated route information is recorded in the Interzone Routing Table. The recorded route information is removed from the packet and X adds its IP address to the route. The packet is then delivered up to the IERP.

D.6
Event: A QUERY_EXTENSION is received from the IERP.

Action: The packet is sent to the BRP layer of the next hop to the specified IERP recipient (in this case, the query destination).

D.7
Event: A QUERY_EXTENSION is received from the IP. X is not the intended BRP recipient.

Action: Identifying query information is recorded in the Detected Queries Table and accumulated route information is recorded in the Interzone Routing Table. The packet is then discarded.

D.8
Event: A QUERY_EXTENSION packet is received from the IP. X is the intended BRP recipient, but is not the intended IERP recipient.

Action: Identifying query information is recorded in the Detected Queries Table and accumulated route information is recorded in the Interzone Routing Table. The recorded route information is removed from the packet and X adds its IP address to the route. The packet is then sent to the BRP of the next hop to the intended IERP recipient.
IERP recipient.

Action: Identifying query information is recorded in the Detected Queries Table and accumulated route information is recorded in the Interzone Routing Table. The recorded route information is removed from the packet and X adds its IP address to the route. The packet is then delivered up to the IERP.

D.10
Event: A ROUTE_REPLY packet is received from the IERP.
Action: The IP addresses of X and the previous hops back to the query source (cached in the Detected Queries Table) are added to the route. The packet is sent back to the IERP layer of the query source, by way of a BRP layer transmission to the first hop back to the query source.

D.11
Event: A ROUTE_REPLY packet is received from the IP. X is not the intended BRP recipient or the ROUTE_REPLY was previously detected.
Action: The packet is discarded.

D.12
Event: A ROUTE_REPLY packet is received from the IP. X is the intended BRP recipient, but not the intended IERP recipient. The ROUTE_REPLY was not previously detected.
Action: Identifying query information is recorded in the Detected Replies Table and accumulated route information is recorded in the Interzone Routing Table. The IP addresses of X and the previous hops back to the query source (previously recorded in the Interzone Routing Table) are added to the route. The packet is then sent to the BRP layer of the previous hop back to the query source.

D.13
Event: A ROUTE_REPLY packet is received from the IP. X is the intended BRP recipient and the intended IERP recipient. The ROUTE_REPLY was not previously detected.
Action: Identifying query information is recorded in the Detected Replies Table and accumulated route information is recorded in the Interzone Routing Table. The recorded route information is removed from the packet. The packet is then delivered up to the IERP.
Event: A ROUTE_ACCUMULATION packet is received from the IERP.

Action: The packet is sent to the IERP layer of the specified IERP recipient by way of a BRP transmission to the next hop toward the IERP recipient.

D.15

Event: A ROUTE_ACCUMULATION packet is received from the IP. X is not the intended BRP recipient.

Action: The packet is discarded.

D.16

Event: A ROUTE_ACCUMULATION packet is received from the IP. X is the intended BRP recipient, but not the intended IERP recipient.

Action: The IP addresses of X and the next hops to the intended IERP recipient (previously recorded in the Detected Replies Table) are added to the route. If this packet contains a route repair and the repair has already been accumulated, then a copy of the packet is delivered to the IERP. The packet is then sent to the BRP layer of the next hop toward the IERP recipient.

D.17

Event: A ROUTE_ACCUMULATION packet is received from the IP. X is the intended BRP recipient and the intended IERP recipient.

Action: The packet is delivered up to the IERP.

D.18

Event: A ROUTE_OPTIMIZATION packet is received from the IERP.

Action: X indicates (in its ROF field) those route nodes that belong to its routing zone. The packet is then sent to the IERP layer of the specified IERP recipient, by way of a BRP transmission to the next hop toward the IERP recipient.

D.19

Event: A ROUTE_OPTIMIZATION packet is received from the IP. X is not the intended BRP recipient.

Action: The packet is discarded.

D.20

Event: A ROUTE_OPTIMIZATION packet is received from the IP. X is the intended BRP recipient, but not the intended IERP recipient.

Action: X indicates (in its ROF field) those route nodes that belong to its routing zone.
The packet is then sent to the IERP layer of the specified next hop toward the IERP recipient.

D.21

Event: A ROUTE_OPTIMIZATION packet is received from the IP. X is the intended BRP recipient and the intended IERP recipient.

Action: X indicates (in its ROF field) those route nodes that belong to its routing zone. The packet is then delivered up to the IERP.

E. Pseudocode Implementation

We define two complimentary operations on packets: extract(packet) and load(packet)

```
extract (packet)
extracts the fields of the IERP packet to the following variables: {type, TTL, hop_count, flags, current_hop_ptr, query_id, reply_id, source, reply_node, bad_link_source, dests, next_IERP, next_BRP, prev_IERP, route, metric, ROF}

load (packet)
loads the values of the aforementioned variables into the fields of the IERP packet.
load(packet) automatically computes the values of:
num_dests = |dests|
num_nodes = |routes|
```

E.1 Receive Packet from IERP

extract(packet)

```
switch(type)
{
    case: ROUTE_QUERY
        IERP_prev = MY_ID
        if ((bad_link_source == MY_ID (OR) source == MY_ID)
            { 
                if(source != MY_ID)
                    { 
                        route = reverse(Interzone_Routing_Table(source))
                        route = {route, MY_ID}
                    }
                else
                    route = NULL
            
        current_hop_ptr = |route|
        if(bad_link_source)
            add(Detected_Queries,(bad_link_source,query_id, IERP_prev))
        else
```
add(Detected_Queries, {source, query_id, IERP_prev})

for (IERP_next = each host in Intrazone_Routing_Table)
{
    if (IERP_next is a peripheral node)
    {
        BRP_next = get_shortest_route(Intrazone_Routing_Table, IERP_next)->next_hop
        metric = get_metric(Intrazone_Routing_Table, BRP_next)
        load(packet)
        send(packet, BRP_next)
    }
}
break

case: QUERY_EXTENSION
    BRP_next = get_shortest_route(Intrazone_Routing_Table, IERP_next)->next_hop
    load(packet)
    send(packet, BRP_next)
break

case: ROUTE_REPLY
    prev_hops = route(1: current_hop_ptr-1)
    next_hops = route(current_hop_ptr+1 : |route|)
    if (prev_hops(1) == MY_ID)
    {
        prev_hops = reverse(Interzone_Routing_Table[IERP_next])
        if (prev_hops(1) == IERP_next (OR) prev_hops == NULL)
        {
            prev_hops = NULL
            BRP_next = IERP_next
        }
        else
        {
            BRP_next = prev_hops(|prev_hops|)
        }
    }
    else
    {
        BRP_next = prev_hops(|prev_hops|)
    }
    if (!is_neighbor(Intrazone_Routing_Table, BRP_next))
    {
        prev_hops = prev_hops(|prev_hops|)
        BRP_next = prev_hops(|prev_hops|)
    }
    metric = get_metric(Intrazone_Routing_Table, BRP_next)
    current_hop_ptr = |prev_hops|+1
    route = {prev_hops, MY_ID, next_hops}
    load(packet)
    send(packet, BRP_next)
break
case: ROUTE_ACCUMULATION
if(IERP_next == source)
{
    BRP_next = route(|route|)
    current_hop_ptr = |route|+1
}
if(IERP_next == dest)
{
    BRP_next = route(1)
    current_hop_ptr = 0
}
load(packet)
send(packet,BRP_next)
break

Haas, Pearlman                Expires December 1999                  [Page
INTERNET DRAFT              The Zone Routing Protocol         June 1999

case: ROUTE_OPTIMIZATION
ROF = NULL
for (node = {source,route,dest})
{
    if ((EXIST) IntrazonRouting_Table[node])
        ROF = (ROF,1)
    else
        ROF = (ROF,0)
}
if(IERP_next == dest)
{
    BRP_next = route(1)
    current_hop_ptr = 0
}
if(IERP_next == source)
{
    BRP_next = route(|route|)
    current_hop_ptr = |route|+1
}
load(packet)
send(packet,BRP_next)
break

E.2 Receive Packet from IP

extract(packet)

switch(type)
{
    case ROUTE_QUERY:
    if (TTL > 0 (AND)
        (!EXISTS(Detected_Queries[source,query_id] (OR)
            Detected_Queries[source,query_id]->from == IERP_prev))
    {
        TTL--
hop_count++
        prev_hops = route(1 : current_hop_ptr)
        next_hops = route(current_hop_ptr+1 : |route|)
        if(bad_link_source)
add(Detected_Queries, {bad_link_source, query_id, IERP_prev})
status =
add(Interzone_Routing_Table, BRP, bad_link_source,
    prev_hops, metric)
} else
{
    add(Detected_Queries, {source, query_id, IERP_prev})
    status = add(Interzone_Routing_Table, BRP, source,
        prev_hops, metric)
}

if (status == RECORDED_ROUTE)
{
    prev_hops = NULL
    metric = compress_metric(metric)
} route = {prev_hops, MY_ID, next_hops}
current_hop_ptr = |prev_hops|+1

if(BRP_next == MY_ID)
{
    if(IERP_next == MY_ID)
    {
        load(packet)
        deliver(packet, IERP)
    } else
    {
        d = dests (BELONGING TO) Intrazone_Routing_Table
        if(|d| > 0)
        {
            load(packet)
            deliver(packet, IERP)
        }
        if(||d| == 0) (OR)
          (|d| < |dests| (AND) flag(0) == ALL))
        {
            BRP_next=get_shortest_route(
                Intrazone_Routing_Table,
                IERP_next)->next_hop
            metric = {metric, get_metric(
                Intrazone_Routing_Table,
                BRP_next)}
            load(packet)
            send(packet, BRP_next)
        }
    }
} else
discard(packet)
else
{
    if(bad_link_source)
    {
        add(Detected_Queries,{bad_link_source,query_id,IERP_prev})
    }
    else
    {
        add(Detected_Queries,{source,query_id,IERP_prev})
    }

discard(packet)
}
break

case QUERY_EXTENSION:
    if (!EXISTS(Detected_Replies[reply_node,reply_id])
    {
        hop_count++
        prev_hops = route(1: current_hop_ptr)
        next_hops = route(current_hop_ptr+1 : |route|)

        if(bad_link_source)
        {
            add(Detected_Queries,{bad_link_source,query_id,IERP_prev})
            status = add(Interzone_Routing_Table,BRP,bad_link_source,
                prev_hops,metric)
        }
        else
        {
            add(Detected_Queries,{source,query_id,IERP_prev})
            status = add(Interzone_Routing_Table,BRP,source,
                prev_hops,metric)
        }

        if (status == RECORDED_ROUTE)
        {
            prev_hops = NULL
            metric = compress_metric(metric)
        }

        route = {prev_hops, MY_ID, next_hops}
        current_hop_ptr = |prev_hops|+1

        if(BRP_next == MY_ID)
        {
            if(IERP_next == MY_ID)
            {
                load(packet)
                deliver(packet,IERP)
            }
            else
            {

                BRP_next=get_shortest_route(Intrazone_Routing_Table,
metric = IERP_next)->next_hop
(metric, get_metric(Intrazone_Routing_Table, BRP_next))

load(packet)
send(packet, BRP_next)
}

else
discard(packet)
}
else
discard(packet)
break

case ROUTE_REPLY:
if(MY_ID == BRP_next (AND)
!EXISTS(Detected_Queries[source,query_id]))
{
    prev_hops = route(1: current_hop_ptr-1)
    next_hops = route(current_hop_ptr : |route|)
    add(Detected_Replies, {reply_node,reply_id})
    status=add(Interzone_Routing_Table,BRP,dest,
    next_hops,metric)
    if (status == RECORDED_ROUTE)
    {
        next_hops = NULL
        metric = compress_metric(metric)
    }

    if (prev_hops(1) == MY_ID)
    {
        prev_hops=reverse(Interzone_Routing_Table[IERP_next])
        if(prev_hops(|prev_hops|) == IERP_next (OR)
            prev_hops == NULL)
        {
            prev_hops = NULL
            BRP_next = IERP_next
        }
        else
            BRP_next = prev_hops(|prev_hops|)
    }
    else
        BRP_next = prev_hops(|prev_hops|)

    if (!is_neighbor(Intrazone_Routing_Table, BRP_next))
    {
        prev_hops={prev_hops,get_route(Intrazone_Routing_Table,
        BRP_next))
        BRP_next = prev_hops(|prev_hops|)
    }

    if(MY_ID == IERP_next)
current_hop_ptr = 0
load(packet)
deliver(packet, IERP)
}
else
{
    metric = {metric, get_metric(Intrazone_Routing_Table, BRP_next)}
    route = {prev_hops, MY_ID, next_hops}
    current_hop_ptr = |prev_hops|+1
    load(packet)
send(packet, BRP_next)
}
else
discard(packet)
break

case ROUTE_ACCUMULATION:
    if(MY_ID == BRP_next)
        {
            if(IERP_next == source)
            {
                prev_hops = route(1: current_hop_ptr-1)
                next_hops = route(current_hop_ptr : |route|)
                if (prev_hops(1) == MY_ID)
                {
                    prev_hops=reverse(Interzone_Routing_Table[IERP_next])
                    if(prev_hops(1) == IERP_next (OR) prev_hops == NULL)
                    {
                        prev_hops = NULL
                        BRP_next = IERP_next
                    }
            }
            else
            {
                BRP_next = prev_hops(|prev_hops|)
            }
        }
        else
        {
            BRP_next = prev_hops(|prev_hops|)
        }

        if(MY_ID == bad_link_source)
        {
            flags(1) = 1
        }
    
    if(IERP_next == dest)
    {
        prev_hops = route(1: current_hop_ptr)
        next_hops = route(current_hop_ptr+1 : |route|)
        if (next_hops(|next_hops|) == MY_ID)
        {
            next_hops=Interzone_Routing_Table[IERP_next])
if(next_hops(\mid next_hops\mid) == IERP_next (OR)
    next_hops == NULL)
{
    next_hops = NULL
    BRP_next = IERP_next
} else
    BRP_next = next_hops(1)
else
    BRP_next = next_hops(1)

else
    BRP_next = next_hops(1)

if(MY_ID == IERP_next)
{
    load(packet)
    deliver(packet, IERP)
} else
{
    if(flags(1) == 1)
    {
        load(packet)
        deliver(packet, IERP)
    }
    metric = {metric, get_metric(Intrazone_Routing_Table, BRP_next)}
    route = {prev_hops, MY_ID, next_hops}
    current_hop_ptr = \mid prev_hops\mid+1
    load(packet)
    send (packet,BRP_next)
} else
    discard(packet)
break

case ROUTE_OPTIMIZATION:
    if(MY_ID == BRP_next)
    {
        f = NULL
        for (node = {source,route,dest})
        {
            if ((EXISTS) Intrazone_Routing_Table[node])
                f = {f,1}
            else
                f = {f,0}
        }
        if (IERP_next == source)
        {
            current_hop_ptr--
            prev_hops = route(1: current_hop_ptr-1)
            next_hops = route(current_hop_ptr+1 : \mid route\mid)
            BRP_next = prev_hops(\mid prev_hops\mid)
ROF = {f,ROF}
}
if (IERP_next == dest)
{
current_hop_ptr++
prev_hops = route(1: current_hop_ptr-1)
next_hops = route(current_hop_ptr+1 : |route|)
BRP_next = next_hops(1)
ROF = {ROF,f}
}

Haas, Pearlman                Expires December 1999                  [Page
51]
INTERNET DRAFT              The Zone Routing Protocol                June
1999

if(MY_ID == IERP_next)
{
    load(packet)
    deliver(packet, IERP)
}
else
{
    load(packet)
    send (packet,BRP_next)
}
else
discard(packet)
break

Haas, Pearlman                Expires December 1999                  [Page
52]
INTERNET DRAFT              The Zone Routing Protocol                June
1999

5. Other Considerations

5.1 Sizing the Zone Radius

The performance of the Zone Routing Protocol is determined by the
routing zone radius. In general, dense networks consisting of a few
fast moving nodes favor smaller routing zones. On the other hand, a
sparse network of many slowly moving nodes operates more efficiently
with a larger zone radius.

The simplest approach to configuring the routing zone radius is to
make the assignment once, prior to deploying the network. This can
be performed by the network administration, if one exists, or by
the manufacturer, as a default value. This may provide acceptable
performance, especially in situations where network characteristics
do not vary greatly over space and time. Alternatively, the ZRP can
adapt to changes in network behavior, through dynamic configuration
of the zone radius [3]. In [4], it was shown that a node can accurately
estimate its optimal zone radius, on-line, based on local measurements of
ZRP traffic. The re-sizing of the routing zone can be carried out by a
protocol that conveys the change in zone radius to the members of the
routing zone. The details of such an update protocol will be provided in
a future version of this draft.
5.2 Hierarchical Routing and the ZRP

In a hierarchical network architecture, network nodes are organized into a smaller number of (often disjoint) clusters. This routing hierarchy is maintained by two component routing protocols. An intra-cluster protocol provides routes between nodes of the same cluster, while an inter-cluster protocol operates globally to provide routes between clusters.

The ZRP, with its routing zones and intrazone / interzone routing protocol (IARP/IERP) architecture may give the appearance of being a hierarchical routing protocol. In actuality, the ZRP is a flat routing protocol. Each node maintains its own routing zone, which heavily overlaps with the routing zones of neighboring nodes. Because there is a one-to-one correspondence between nodes and routing zones, the routing zones, unlike hierarchical clusters, do not serve to hide the details of local network topology. As a result, the network-wide interzone routing protocol (IERP) actually determines routes between individual nodes, rather than just between higher level network entities.

For small to moderately sized networks, flat routing protocols, like the ZRP, are preferable to hierarchical routing protocols. In order for a node to be located, its address needs to reflect the node’s location within the network hierarchy (i.e. \{cluster id,node id\}). Because of node mobility, a node’s cluster membership (and thus address) is subject to change. This complicates mobility management, for the benefit of more efficient routing. In many hierarchical routing protocols, each cluster designates a single clusterhead node to relay inter-cluster traffic. These clusterhead nodes become traffic "hot-spots", potentially resulting in network congestion and single point of failure. Furthermore, restricting cluster access through clusterhead nodes can lead to sub-optimal routes, as potential neighbors in different clusters are prohibited from communicating directly. Some hierarchical routing protocols, such as the Zone-Based Hierarchical Link-State (ZHLS) \[^5\]\\(^{5}\), avoid these problems by distributing routing information to all cluster nodes, rather than maintaining a single clusterhead.

In spite of these disadvantages, hierarchical routing protocols are often better suited for very large networks than flat routing protocols. Because hierarchical routing protocols provide global routes to network clusters, rather than individual nodes, routing table storage is more scalable. Similarly, the amount of route update messages is also more scalable. To some extent, the reduction in routing traffic is offset by extra mobility management overhead (i.e. identifying which cluster a node belongs to). However, it is quite common that the environment or existing organizational structure causes nodes to naturally cluster together. In these cases, there may be high degree of intra-cluster mobility, inter-cluster mobility is less common.
A hierarchical routing protocol can be viewed as a set of flat routing protocols, each operating at different levels of granularity. In a two-tier routing protocol, the inter-cluster components is essentially a flat routing protocol that computes routes between clusters. Likewise, the intra-cluster component is a flat routing protocol, that computes routes between nodes in each cluster. For example, the Near Term Digital Radio (NTDR) System [12] and ZHLS both employ proactive link state protocols to determine inter and intracluster connectivity.

In place of traditional proactive or reactive protocols, we recommend that the intra-cluster and inter-cluster routing protocol components be implemented based on the hybrid proactive/reactive ZRP. As described throughout this draft, the ZRP is designed to provide an optimal balance between purely proactive and reactive routing. This applies equally well to routing between nodes at the intra-cluster level and between clusters at the inter-cluster level. Additionally, the hybrid ZRP methodology can be applied to the related mobility management protocols, which determine complete node addresses based on a node’s location in the network hierarchy.

6.0 References


INTERNET DRAFT The Zone Routing Protocol June 1999

Authors’ Information

Zygmunt J. Haas  
Wireless Networks Laboratory  
323 Frank Rhodes Hall  
School of Electrical Engineering  
Cornell University  
Ithaca, NY 14853  
United States of America  
tel: (607) 255-3454, fax: (607) 255-9072  
Email: haas@ee.cornell.edu

Marc R. Pearlman  
389 Frank Rhodes Hall  
School of Electrical Engineering  
Cornell University  
Ithaca, NY 14853  
United States of America  
tel: (607) 255-0236, fax: (607) 255-9072  
Email: pearlman@ee.cornell.edu

The MANET Working Group contacted through its chairs:

M. Scott Corson  
Institute for Systems Research  
University of Maryland  
College Park, MD 20742  
(301) 405-6630  
corson@isr.umd.edu

Joseph Macker  
Information Technology Division  
Naval Research Laboratory  
Washington, DC 20375  
(202) 767-2001  
macker@itd.nrl.navy.mil