Nimrod Working Group
Internet Draft
March 1996
draft-ietf-nimrod-fun-pro-spec-00.ps

Ram Ramanathan
Martha Steenstrup
BBN Systems and Technologies
Expires 30 August 1996

Nimrod Functionality and Protocol Specifications, Version 1

Status of this Memo

This document is 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. Internet Drafts may be updated, replaced, or obsoleted by other documents at any time. It is not appropriate to use Internet Drafts as reference material or to cite them other than as a "working draft" or "work in progress".

Please check the 1id-abstracts.txt listing contained in the "internet-drafts" directories on ftp.isi.edu (U.S. West Coast), ds.internic.net (U.S. East Coast), munnari.oz.au (Pacific Rim), nic.nordu.net (Europe), or ftp.is.co.za (Africa) to learn the current status of any Internet Draft.

Distribution of this Internet Draft is unlimited. Please send all comments to nimrod-wg@bbn.com.

Abstract

Nimrod is a scalable routing architecture designed to support a dynamic internetwork of arbitrary size, to provide service-specific routing in the presence of multiple constraints, and to admit incremental deployment throughout an internetwork. The key features of Nimrod include representation of internetwork connectivity and services in the form of maps at multiple levels of abstraction; source- and destination-controlled route generation and selection based on maps and traffic service requirements; and source- and destination-controlled message forwarding according to the routes selected. This document contains a description of Nimrod functionality and a specification of the protocols constituting Nimrod. In particular, the operations pertinent to the map, locator, adjacency, route, and forwarding databases are described, and the Reliable Transaction, Update, Query-Response, Path Management, and Discovery protocols are specified.

Acknowledgments

We thank Tom Calderwood, Winston Edmond, Charlie Lynn, Trevor Mendez, Betty O'Neil, Mike Patton, Ram Ramanathan, and Tim Shepard for producing an experimental Nimrod software system that has enabled us to test the practicality of Nimrod. We are especially grateful to Charlie Lynn, chief architect of the Nimrod software, for his flexible system design, his careful and critical analysis of the Nimrod protcols, and his detailed packet formats (depicted in this document).

Contents

1 Scope and Overview

This document contains a description of Nimrod functionality,and a specification of the protocols constituting Nimrod. While it has been our intention that the document be self-contained, it would help the reader to be familiar with the Nimrod architecture and functionality as described in [1] and [2]. Nimrod does not specify support for mobility or multicast, but does specify requirements for solutions to mobility and multicast within the Nimrod context. A discussion of these issues can be found in [3] and [4].

The document has been organized so that readers may inform themselves at various levels of detail. Specifically, readers wishing to know only what Nimrod's functionality is may confine themselves to sections 2 and 3. For readers wishing to understand and evaluate the protocols comprising Nimrod, we additionally recommend sections 4, 5, 6, 7, and 8. Finally, for Nimrod implementors, sections 9 and 11 give additional details.

2 Introduction

Nimrod is a scalable routing architecture designed to support a dynamic internetwork of arbitrary size, to provide service-specific routing in the presence of multiple constraints, and to admit incremental deployment throughout an internetwork. The key features of Nimrod include representation of internetwork connectivity and services in the form of maps at multiple levels of abstraction; source- and destination-controlled route generation and selection based on maps and traffic service requirements; and source- and destination-controlled message forwarding according to the routes selected.

At the most general level, one may view any routing system as a set of basic functions which are producers and consumers of certain databases of routing information. These routing functions and their associated routing information include:

1. Assembling, distributing, and collecting information necessary for route generation and selection. This information includes internetwork connectivity and services, traffic service requirements, and locations of traffic sources and destinations.

2. Generating and selecting routes, based on the collected information.

3. Establishing in routers information necessary for forwarding messages, based on the selected routes.

4. Forwarding messages along these routes.

Routing systems may, however, differ in the details of the mechanisms that provide a particular routing function. As Nimrod has been designed for routing in large, heterogeneous, and dynamic internetworks, its basic routing functions include additional mechanisms for reducing the quantity of routing information that must be distributed, processed, and stored throughout an internetwork; for discovering and accommodating changes in routing information caused by physical changes in an internetwork; and for protecting the integrity of routing information.

2.1 Overview of the Nimrod Architecture

Before Nimrod routing can be applied within an internetwork, the internetwork must be represented in terms of the two basic Nimrod entities: *nodes* and *endpoints*. The internetwork's physical assets, such as routers, point-to-point links, and multiaccess networks, must be captured in Nimrod *maps* comprising interconnected nodes. Traffic sources and destinations must be cast as Nimrod endpoints. Nimrod entities possess attributes (e.g., location with respect to the maps, interconnectivity with other entities, and service information) which are important for routing.

2.1.1 Clustering and Abstraction

Ideally, Nimrod maps should be constructed so as to satisfy the following two primary, and potentially conflicting, goals:

1. Minimize the amount of routing information maintained throughout an internetwork.
2. Maintain routing information sufficient to generate routes that meet traffic service requirements.

To satisfy these goals, Nimrod employs two complementary map construction procedures, namely *clustering* of internetwork physical assets into nodes and *abstraction* of attributes of the component physical assets resulting in node attributes.

The objective of clustering is to reduce the number of entities visible to Nimrod routing at any given level of the hierarchy. Nodes are usually formed by clustering physical assets possessing similar attributes. These attribute similarities might be in terms of, for example, qualities of service, restrictions on access to services, or ownership of these assets. Such clustering results in a reduction in the amount of information necessary to characterize these physical assets, without a reduction in information detail. However, an internetwork's physical assets may be diverse enough so that clustering according to attribute similarity produces no significant reduction in the number of entities visible to Nimrod routing. In this case, alternative clustering criteria (e.g., geographical locality of physical assets) may be employed.

Clustering may be applied repeatedly, such that physical assets are first clustered into nodes, and then nodes are themselves clustered into larger nodes, and so on. Iterative clustering further reduces the number of entities visible to Nimrod routing at a given level of the hierarchy, and results in a hierarchical organization of nodes with a single top-level *universal* node containing all other entities. In the clustering hierarchy, the clustering criteria applied at different levels may not necessarily be the same.

The objective of abstraction is to reduce the amount of information required to characterize an entity visible to Nimrod routing. Nodes whose component physical assets possess different attributes rely on information abstraction in order to reduce the number of attributes used to characterize them. Abstraction procedures include, for example, eliminating attributes possessed by only a small percentage of the component physical assets or expressing attributes in terms of ranges of values exhibited by these physical assets. Multiple abstraction procedures may be applied to produce the attributes of a given node (e.g., first eliminating attributes possessed by only a few physical assets and then taking the average values of the remaining attributes for the physical assets in the node).

Nimrod does not mandate the choice of clustering and abstraction procedures to invoke in an internetwork. Rather, this choice is a local one under the control of the managers of the portions of the internetwork to be represented as Nimrod nodes, and hence network managers may develop procedures that best suit their needs. We note that the specific clustering and abstraction procedures employed in an internetwork may have a significant effect on the quality of routes generated and on the cost of routing information maintenance. Hence, network managers should exercise care in selecting and using these procedures and may wish to experiment with several different ones during the evolution of their nodes. Although clustering and abstraction procedures may be fully automated, we recommend allowing manual intervention in order to enable network managers to make cost-benefit tradeoffs appropriate for their particular networks.

2.2 Nimrod Entities

All of the Nimrod routing functions are performed with respect to an internetwork's representation in terms of the basic Nimrod routing entities, namely nodes and endpoints. Each Nimrod entity has a set of attributes, each of which may be established through one or more of the following methods:

1. Manual configuration.
2. Automatic acquisition during initialization.
3. Active measurement.
4. Abstraction of attributes of component nodes.

A node is a set of contiguous internetwork physical assets. It may be formed by clustering physical assets directly or by clustering existing nodes. If the given node itself comprises component nodes, the routing system employed to route traffic within or across the node is Nimrod routing. Otherwise, this routing system may use any other routing protocol(s). A node's attributes include its:

1. Node identifier (NID). An *NID* is a location-independent referent for a node. It is a globally unique, relatively short, fixed-length bit string used by Nimrod-capable devices to communicate node identity (primarily used before a node acquires its locator).
2. Locator. A node's *locator* is a globally unique label describing the node's position in the clustering hierarchy. It consists of the global locator of the node's enclosing node in the clustering hierarchy, concatenated with a local bit-string, called an *element*, unique among all component nodes and endpoints of the enclosing node.
3. A pool of locator elements that may be assigned to its component nodes and endpoints.
4. Constraints on forming *associations* with endpoints. An association is a relationship formed between a node and an endpoint, such that the endpoint acquires a locator from the node. A node may be associated with multiple endpoints, and an endpoint may be associated with multiple nodes.
5. Constraints on forming *adjacencies* with other nodes. An adjacency is a neighbor relationship formed between two nodes that have a direct communication capability. The neighbor

relationship need not be symmetric. For example, nodes X and Y may agree to a relationship in which Y is adjacent to X, but X is not adjacent to Y.

6. Maps consisting of the node's current adjacencies and service offerings.

7. Credentials of the node's manager, used in forming node adjacencies and endpoint associations.

An endpoint is a traffic source, destination, or both that is visible to other Nimrod entities through association with one or more Nimrod nodes. Examples of endpoints include hosts and routers or even processes within hosts and routers. An endpoint's attributes include its:

1. Endpoint identifier (EID). An *EID* is a location-independent referent for an endpoint. It is a globally unique, relatively short, fixed-length bit string used by Nimrod-capable devices to communicate endpoint identity.

2. Names. A endpoint *name* is a globally unique, variable-length, structured, ASCII string used primarily by humans to refer to the endpoints. Nimrod uses Domain Name System (DNS) names for this purpose.

3. Constraints on forming associations with nodes.

4. Locators. An endpoint's locators are obtained from the nodes with which the endpoint is associated. Therefore, as an endpoint may be associated with more than one node, it may obtain more than one locator.

5. Traffic service requirements from the perspectives of the endpoint as source and as destination.

6. Credentials of the endpoint and its manager, used respectively in authentication of routing information and in forming node associations.

2.3 Nimrod Routing Functions and Databases

At the core of Nimrod lies a set of distributed databases containing routing information that is constructed, accessed, and acted upon by the routing functions. These databases and their relationships to the routing functions are as follows:

1. *Node attributes.* Each node has a set of attributes used in forming node adjacencies and endpoint associations, in constructing maps, and in assigning locators to component nodes and endpoints.

2. *Endpoint attributes.* Each endpoint has a set of attributes used in forming node associations and in selecting routes.

3. *Endpoint/locator associations.* Nimrod endpoint locators are used in generating routes between and in forwarding messages toward those endpoints. Endpoint/locator associations are stored and accessed through the DNS.

4. *Maps.* Each Nimrod node has a set of maps describing its traffic service offerings and adjacencies to other nodes, collectively called *connectivity specifications* and used in generating routes.

5. *Routes.* Routes are generated in response to requests on behalf of traffic sessions between endpoints. Route generation works within the constraints of the service requirements specified by the endpoints and the services and adjacencies advertised in maps. Each route is expressed in terms of nodes and their corresponding connectivity specifications and is used to construct forwarding information to be installed in routers.

6. *Forwarding information.* Nimrod traffic forwarding is path-oriented, where a *path* is defined by the forwarding state stored in routers according to the route selected and is under the control of the source and destination endpoints.

Databases relevant to but not maintained by Nimrod are the pool of NIDs that may be assigned to nodes and the pools of EIDs and names that may be assigned to endpoints. EIDs and NIDs may be drawn from the same number space.

Each of the Nimrod routing functions uses portions of the contents of one or more of the Nimrod databases. In a dynamic internetwork, the procedures for updating and retrieving the contents of Nimrod databases will be performed frequently. Therefore, each Nimrod database is organized to optimize the performance of these procedures along the dimensions of delay, internetwork resource consumption, fault tolerance, and load balancing.

Nimrod databases are maintained by a combination of routers, hosts, and special-purpose physical devices. The use of special-purpose devices means that routers and hosts do not have to assume all of the processing and memory load related to routing. For example, as route generation is a computationally intensive procedure, some network managers may elect to use dedicated devices, distinct from routers, whose sole purpose is to generate Nimrod routes.

For most Nimrod databases, we suggest distributing database contents over several physical devices throughout an internetwork. In a large internetwork, one may in fact have no other choice; the memory required for a single Nimrod database may exceed the storage capacity of any one device. Also, we suggest distributing database contents with partial redundancy, such that each database entry is stored in more than one device. Distributed organization of Nimrod database contents helps to reduce the database maintenance and query-response costs borne by any one physical device. Partial redundancy helps to increase the availability of database contents and to reduce the costs of the average database query-response; it may also increase the cost of the average database update, however.

2.3.1 Nimrod Database Consistency

Each Nimrod database contains routing information crucial for successful communication between endpoints. Inconsistencies between the actual state of the internetwork and the state as reflected by Nimrod database contents may result in impaired communication between a pair of endpoints and, in the worst case, may completely disrupt communication among all endpoints. Thus, minimizing the number as well as the consequences of such inconsistencies is a primary objective of the Nimrod database maintenance procedures.

Inconsistencies between database contents and actual internetwork state may result from delays incurred in updating database contents following internetwork state changes. Many of the Nimrod databases are volatile and hence require mechanisms for keeping the contents current, in order to prevent propagation and use of stale routing information. Database maintenance includes rapid and reliable updating with new information as well as removal of old information. We recommend that each Nimrod database be maintained as a cache, such that each entry has a finite lifetime and may be removed from the database when it expires. Cache entry lifetimes will depend upon the expected duration of the usefulness of the cached information.

Inconsistencies between database contents and internetwork state may also result from errors introduced directly into database contents. Errors in Nimrod database contents may be injected inadvertently, through faults in the transmission media or in physical device memory, through misconfiguration, or through incorrect implementation of the database maintenance procedures. Errors may also be injected intentionally by malicious parties, through distribution of fictitious database updates and responses to queries (by capturing and corrupting existing database messages or by generating new messages) or through modification of database maintenance procedures.

Updating and retrieval of Nimrod database contents involve frequent communication of routing information over an internetwork and hence expose this routing information to numerous potential opportunities for error introduction. Therefore, the protocols that carry out these procedures attempt to protect the routing information from introduced errors and malicious parties. In particular, the protocols for communicating information to or from a Nimrod database permit the intended recipient of that information to:

1. Authenticate the information.
2. Detect corruption of the information.
3. Determine whether the information received is newer than any related information the recipient already possesses.
4. Indicate to the sender the receipt of acceptable or unacceptable information.

These protocols also permit the sender to retransmit to the intended recipient any information that it perceives the recipient has failed to receive successfully.

We note that while Nimrod requires consistency between database contents and internetwork state, it does not require different physical devices to maintain identical views of internetwork state (e.g., two different routers might maintain different maps for the same node, both of which are consistent with the physical assets of that node). Furthermore, Nimrod does not require consistency in route selection across different physical devices (e.g., two different routers might select routes to the same destination such that each router is included in the other's route). The underlying path-oriented nature of message forwarding in Nimrod enables loop-free forwarding in the presence of such inconsistencies in route selection among routers.

2.4 Nimrod Agents

Within an internetwork, each Nimrod database is stored in a set of physical devices. Each physical device containing a portion of a Nimrod database executes a set of functionality, called a Nimrod *agent*, for manipulating the database contents. A single device may contain portions of more than one Nimrod database and hence may contain more than one Nimrod agent. Each Nimrod agent is a Nimrod endpoint.

For each Nimrod database, certain agents are responsible for maintaining specific portions. Such an agent is designated as an *authoritative source* for that portion of the database. A specific portion of a database may have multiple authoritative sources. Each agent is an authoritative source for some portion of a database but may also obtain and cache information learned from authoritative sources for other portions of that same database. In addition to receiving unsolicited database updates, a Nimrod agent may also refresh its database by querying other agents of the same type for their database contents.

Nimrod agents and databases are organized according to the clustering hierarchy, such that each node has a set of agents that act on its behalf to answer or forward database queries. The Nimrod agents and their corresponding functions are as follows:

1. *Node representatives.* Each Nimrod node must have one or more representatives which maintain the database of the node's attributes and act on its behalf. A node representative is responsible for forming adjacencies with other nodes; forming associations with endpoints; assigning locator elements to component nodes and endpoints; receiving maps from component nodes; and constructing its node's maps. All node representatives for a given node must construct the same map for a node, i.e., must use the same algorithms for map construction. Node representatives are authoritative sources for the maps of the nodes they represent.

2. *Endpoint representives.* Each Nimrod endpoint must have one or more representatives which maintain the database of the endpoint's attributes and act on its behalf. An endpoint representative is responsible for forming associations with nodes; discovering, through the DNS, locators of the endpoints with which its endpoints wish to communicate; requesting routes to those endpoints by querying route agents, and ensuring that the routes satisfy its endpoints' service requirements; initiating path setup along the selected routes; and forwarding data along the established paths. Endpoint representatives are authoritative sources for the locators and service requirements of the endpoints they represent.

3. *Route agents.* Each node may have one or more route agents responsible for collecting maps from nodes throughout the internetwork and for generating and dispensing routes based on endpoint service requirements and node connectivity specificiations advertised in maps. Route agents are authoritative sources for the routes they generate.

4. *Forwarding agents.* Each node must have one or more forwarding agents responsible for initiating neighbor relationships with forwarding agents in other nodes; requesting routes; installing forwarding information in routers; forwarding messages along established paths; and controlling traffic flow into and out of the node according to the node's access restrictions. While each of these functions could be performed by a different type of agent, we have elected to concentrate them in the forwarding agents, in order to minimize the number of different agent types performing the Nimrod routing functions. Forwarding agents are authoritative sources for the portion of the paths that traverse them.

Agents acting on behalf of a node need not reside within that node. Nevertheless, we recommend locating all Nimrod agents (and their databases) close to the entities on whose behalf they act. Such location minimizes delay and internetwork resource consumption when updating the databases corresponding to those entities and in responding to queries from other agents in the vicinity of those entities. A Nimrod agent residing external to the node on whose behalf it acts must be configured with location information for that node, and in some cases for ancestral and descendant nodes as well, in order communicate with other agents that act on behalf of the same node. Also, the forwarding agents within a node must be configured with the location of agents external to but acting on behalf of that node. We recommend placing all agents within the node on whose behalf they act, and henceforth we describe agent behavior from this perspective.

3 Nimrod Operation : An Overview

This section describes key operating procedures in Nimrod from the viewpoint of how the various databases are managed. The description is organized based on the pertinent database. Specifically, maps (construction, dissemination), routes (generation, acquisition), locators (acquisition, notification, release), adjacencies (formation, release), paths (setup, teardown, forwarding), and discovery (of neighbors, agents) are addressed. This section only provides a brief summary of the operations. For a detailed exposition, the reader is referred to [2]. In the postscript version of this document, the reader may refer to Figures 1-4 (given in Appendix 1), for a quick overview of some of these operations. Throughout this section, the text contains references in parentheses to labels in these figures.

3.1 Maps

Each Nimrod node has a set of maps describing its traffic service offerings and its adjacencies. Maps are maintained and updated by node representatives. A node representative maintains two kinds of maps for its node: a *basic map* that depicts the child nodes, the adjacencies between child nodes and the adjacencies between child and external nodes, plus the services provided by each of these nodes; an *abstract map* that depicts the node, its adjacencies to other nodes and the services provided by the node between any pair of such node adjacencies. A node representative may construct its abstract map using information obtained from abstraction of basic map, configuration, or measurement of service qualities across the node. Abstraction mechanisms are *not* a part of the Nimrod specifications. Rather, each node may choose to implement its own abstraction algorithm (uniform throughout a given node). Maps are updated using *map updates*, and obtained using *map queries* as described below.

3.1.1 Map Update

Map updates are distributed using the Update protocol described in section 5. Maps are automatically updated in response to topological changes using constrained hierarchical flooding, according to the following procedure. The update originates from a representative (Nr in Figure 1) of the node whose abstract map has changed. This node representative sends (arrow 1 in Figure 1) the new abstract map to each boundary forwarding agent (F) which is a neighbor of a boundary forwarding agent of its parent. Note that sending to *each* forwarding agent is necessary in order to handle the case of a partitioned node. F in turn forwards (arrow 2) the update to each F_p to which it is adjacent. That F_p sends (3.x) the map to all of the node representatives (Nr_p) and all of the route agents (R_p) in the node.

The change in the abstract map of a node causes a change in the basic map of the parent node. This in turn may or may not cause a change in the abstract map of the parent node. If it does, then a designated node representative originates another update to the next higher level by sending (4) the new abstract map to a boundary forwarding agent. The procedure described above now applies again at this higher level. In the worst case, a change in a node's abstract map may force changes in all of its ancestral nodes' abstract maps. We expect such changes to be rare, especially in nodes whose descendants are multiply connected.

3.1.2 Map Request and Release

Map requestsm, responses, and releases are transmitted using the Query-Response protocol described in section 6. A map request is sent by a route agent, acting on behalf of an endpoint wishing to obtain or subscribe to the map of a node for route computation. A map release unsubscribes to the map of the node for which a subscribe request was sent. Our description below is in terms of map request; map release is very similar. The kinds of maps that can be requested are as follows:

1. Abstract Maps. Two kinds of abstract maps can be requested - *abbreviated* or *full*. An abstract map is full if it contains service information (i.e., connectivity specifications) and abbreviated if it does not.
2. Basic Maps. Two kinds of basic maps can be requested - *complete* or *partial*. A basic map is complete if it contains abstract maps of all component nodes and is partial if it only contains maps of a proper subset of the component nodes. The abstract map of the component nodes contained in a basic map may all be either full or abbreviated.

A route agent sends the map request towards the targeted node. A flag in the request indicates whether or not a subscription is requested, i.e., updates are to be automatically sent. When the map query reaches a boundary forwarding agent for the targeted node, this forwarding agent relays the query to a node representative for that node. The node representative responds to the route agent with the largest subset of the requested map that is consistent with the map distribution restrictions. If it does not have the maps to fulfill the query, or if its restrictions do not permit it to respond, it still sends a reply to the requestor containing a reason for failure. A node representative is not required to support the map subscription service.

3.2 Routes

Route agents use the maps obtained, either through automatic updates or in response to explicit map requests, in order to do *route generation*. Endpoint representatives and forwarding agents obtain these routes from the route agent using *route requests*. A route specification is expressed in terms of nodes and the connectivity specifications through those nodes, and is used to specify forwarding information to be installed in routers. It also lists the services provided by the route.

3.2.1 Route Generation

The input to route generation includes maps of the topology, a set of session service requirements, and the source and the destination node locators. Its output includes a (set of) route(s), or an indication that no route can be found. Each route contains a sequence of node locators and connectivity specification labels for nodes that have to be traversed in order to meet the service requirements.

We note that the topology used for route generation typically represents the network in varying levels of detail for different regions. Thus, the route constructed by the route generation algorithm will typically not contain the complete list of all routers through which a datagram should pass. The details are filled in at the node where they are required in a recursive fashion when setting up a path or forwarding datagrams (see section 7 for details). Route generation algorithms are *not* specified by Nimrod. Rather, each route agent may choose to implement its own route generation algorithm, even within a single node.

3.2.2 Route Request and Release

Route requests, response, and releases are transmitted using the Query-Response protocol described in section 6. An endpoint representative or a forwarding agent that wishes to obtain a route to a destination may request a route from a route agent. The route request contains the source endpoint's EID and locators, the destination endpoint's EID and locators, and the source and/or destination traffic service requirements. Note that if there are no strict traffic service requirements, in terms of quality, monetary cost, or access restrictions, a route may not need to be acquired (the source and destination endpoint locators may suffice for the route). Note also that the route agent to which a route request is sent need not be in the same node as the requestor.

A route request may also be a subscription to a route, i.e., updated routes are automatically sent to the subscriber. A route release is used to unsubscribe to a specific route. Route agents are not required to support the route subscription service in this version of Nimrod.

In response to a route request, a route agent first searches its route database for a set of feasible routes and if unsuccessful, invokes the route generation algorithm. The route agent may, in the process of attempting to generate feasible routes, obtain more maps of nodes using the map query procedure described in section 3.1.2. A route agent responds to the requestor with either a set of feasible routes or an indication that no feasible route could be found.

3.3 Locators

Nimrod nodes and endpoints require locators for routing. Each node has exactly one locator and each endpoint is associated with at least one locator. Locators are assigned during initialization following activation, reconfiguration, or movement. The representatives of a node are responsible for acquiring the node locator, and the endpoint representative is responsible for acquiring the locators for each of its endpoints.

3.3.1 Acquiring and Releasing Node Locators

Node locator requests, responses, and releases are transmitted using the Query-Response protocol described in section 6. Node locator acquisition involves two phases. First, the designated representative of a node *acquires* a locator from a representative of its parent node. Next, it *notifies* all of the representatives within its node and all descendant nodes of the existence of the new locator. The first phase is illustrated in Appendix 1, Figure 2 and the second phase in Figure 3.

The designated node representative (Nr in Figure 3) wishing to acquire a locator for a node Z first determines the node P from which its locator should be acquired. This node representative (Nr) then sends (1) a locator request message to a boundary forwarding agent (F) which forwards (2) the locator request message to the neighboring boundary forwarding agent in the parent node, which in turn relays (3) the message to a node representative for P (Nr_p). The request contains information that enables the recipient node representative (Nr_p) to evaluate the request and decide whether it can or wants to honor the request.

The node representative responds (4) either with a new node locator (if it decides to honor the request), unique among the locators of P's component nodes, or a denial response otherwise. Upon receiving a positive response, the originating node representative (Nr) decides whether to accept the locator.

If it (Nr) decides to accept the locator, the node representative starts the notification phase (refer Fig 3). Locator notifications are distributed using the Update protocol described in section 5. The node representative notifies all agents in its node (arrows 5.x), and all agents in Z's descendant nodes, of the new locator. The latter is done by forwarding the message to each forwarding agent in Z which is a neighbor of a forwarding agent for one of Z's component nodes. These forwarding agents (F_c) in the component nodes distribute the message to all agents in the component nodes including boundary forwarding agents (F_{c_c}) to their children and so on until the locator trickles down to all of the nodes that have Z as an ancestor.

A locator release may be sent by a node representative of a node wishing to unsubscribe to a locator. This could happen, for instance, if an organization changes its service provider, or due to mobility of a network. The node representative includes the old locator to be released, and the locator and EID of the node representative that issued the locator, in its locator release message to its parent.

3.3.2 Acquiring and Releasing Endpoint Locators

Endpoint locator requests, responses, and releases are transmitted using the Query-Response protocol described in section 6. An endoint representative attempts to acquire a set of locators for each of its endpoints. The endpoint representative, say Er, selects a set of target nodes and for each selected node Z, sends a locator request message identifying Z (label 5 in Figure 2) to a node representative for Z. As in the case of node locators, if this node representative decides to honor the request, it sends a response (6) containing the locator.

3.4 Adjacencies

An adjacency is a neighbor relationship formed between two nodes that are physically joined. The neighbor relationship need not be symmetric, i.e., node A may be adjacent to node B but not vice versa. Adjacencies of a node Z to an external node Y may be formed by clustering together adjacencies of component nodes of Z to Y. At the lowest level, adjacencies are the physical connections themselves.

3.4.1 Acquiring, Activating, and Releasing Adjacencies

Adjacency requests, responses, releases, and activations are transmitted using the Query-Response protocol described in section 6. The distribution of A single designated node representative is responsible for forming adjacencies between its node and neighboring nodes. When forming adjacencies by clustering existing adjacencies (or physical connections), the node representative obtains candidate external adjacencies from the node's basic map and groups these adjacencies according to which of their destination nodes are components of the same enclosing node. This information defines the target node for the adjacency formation requests.

For each candidate adjacency, the node representative initiates an adjacency formation procedure (depicted in Appendix 1 - Figure 4). The node representative (Nr) begins by sending an adjacency request (1) to a node representative for the specified node (Nr_s). Using information present in this request, the recipient node representative (Nr_s) determines whether or not to honor the request, and replies (2) to the requesting node representative (Nr) about its decision. If the response is positive, then the node representative decides whether or not to accept the adjacency. If it decides to accept, then it updates (3.x) all of the node representatives of the newly formed adjacency. The node representative at the other end of the adjacency (Nr_s) also updates (4.x) all node representatives within its node. The adjacency updates to node representatives in the nodes forming the adjacency are distributed using the Update protocol described in section 5. In addition, the adjacency is "activated" by having the two node representatives inform the respective boundary forwarding agents constituting the adjacency that Nimrod data traffic may now be passed.

If a node representative receives a negative reply to an adjacency request message, the message may contain information that indicates that the adjacency is not appropriate. An adjacency is terminated by sending an adjacency release request to the node representative which granted the adjacency. Management decisions and lack of data for a specified period of time may be other reasons for terminating an adjacency.

3.5 Paths

Nimrod supports two distinct data message forwarding modes: *flow* and *datagram*. For each mode, a forwarding agent's "next-hop" forwarding decision is dictated by the information stored in its forwarding database and by information contained within the message to be forwarded.

Flow mode requires the establishment of session-specific forwarding state in certain forwarding agents along the routes selected for a traffic session. With flow mode, each session is assigned one or more *paths*, derived from the selected routes. A path corresponds to forwarding state stored in forwarding agents along a route, and each path has a label which is unique within all of these forwarding agents (but not necessarily globally unique). Distinct traffic sessions may use the same path, and distinct paths may use the same route. The minimum forwarding state required for flow-mode forwarding includes linkages between the path label and the path's previous- and next-hop forwarding agents (and service guarantees for traffic control, if any). In flow mode, data messages carry the path label(s) that guides the message forwarding decisions at forwarding agents along the path.

Datagram mode does not require the establishment of any session-specific forwarding state. In datagram mode, data messages carry a description of the selected route, which guides the message forwarding decisions at forwarding agents along the route. Each forwarding agent at the beginning of a route *segment* (the portion of a route between two successive nodes listed in a route specification) makes an independent forwarding decision for that segment, and hence the session source and destination relinquish some control over message forwarding. However, datagram mode provides robust forwarding, in the sense that the intermediate forwarding agents can base their message forwarding decisions on the current state of their portion of the internetwork.

Both of the Nimrod forwarding modes rely on the existence of underlying paths to fill in route segments. A path may connect a source endpoint to one or more destination endpoints. Forwarding agents execute path management procedures to install path state in and remove path state from their forwarding databases. With these procedures, Nimrod provides support for management and use of unicast and multicast paths. We note, however, that multicast group management and multicast route construction are not part of this initial version of Nimrod. These and other multicast issues are treated in detail in [4]. For simplicity of discussion, we focus on unicast paths in the remainder of this section.

3.5.1 Path Setup

Paths may be set up from source to destination or from destination to source. Each path has an *initiator* and a *target*. We expect that most paths will be set up from the source endpoint to the destination endpoint. Hence, the initiator usually begins the path setup procedure on behalf of the source endpoint, and the target usually accepts or rejects a path on behalf of the destination endpoint.

Nimrod paths are inherently multilevel as follows. We begin with a single path, p^0, derived from the selected route between the source and destination endpoints for the traffic session. (The superscript indexes the level of the path, where the top level is 0.) This path comprises multiple contiguous paths, ${p^1}_1, \ldots, {p^1}_n$, one for each of the n segments of the route on which p^0 is based. (The subscript indexes the path for the corresponding route segment of the higher level path.) Each ${p^1}_j$ itself comprises multiple contiguous paths corresponding to each of its segments, and so on. In general, for each ${p^i}_j$ composing ${p^{i-1}}_k = p^i$, the initiator and target of ${p^i}_j$ maintain linkages to the path ${p^{i-1}}_k(p^i)$, which helps to guide forwarding along the successive segments of ${p^{i-1}}_k(p^i)$.

Forwarding agents and endpoint representatives try to form paths by piecing together existing paths rather than by setting up new paths. This method provides the lowest-cost message forwarding in terms of the amount of route generation and forwarding state installation required. In a busy internetwork, there are likely to be many existing paths, and hence we expect this mechanism to be much less expensive than individually setting up and maintaining paths for each traffic session.

We now describe how a new traffic session uses paths at multiple levels, distinguishing the actions in flow mode and datagram mode where appropriate. Whenever the endpoint representative receives a data transport request, it always checks whether there already exists a satisfactory path for the session. This is true whether the new session desires flow or datagram mode forwarding. If a satisfactory path exists, the endpoint representative links the session to the path and forwards session traffic along that path. If no such path exists, however, the endpoint representative attempts to obtain a feasible route for the session. Note that route generation might not be required and that a feasible route might include only the source and destination locators.

After obtaining a feasible route, the endpoint representative proceeds to determine where to install the necessary forwarding state.

Flow mode: The endpoint representative becomes the initiator of a new path, p^0, and generates a path setup message.

If the route contains more than the source and destination locators, the endpoint representative then checks whether there already exists a satisfactory path for the session from itself to the next node in the route specification. Provided such a path, ${p^1}_1$, exists, the endpoint representative proceeds as follows:

Flow mode: The initiator of ${p^1}_1$ (which is also the initiator of p^0) links p^0 and ${p^1}_1$ in its forwarding database and sends p^0's setup message to the target of ${p^1}_1$. Upon reception of the setup message, the target of ${p^1}_1$ also links p^0 and ${p^1}_1$ in its forwarding database.

Datagram mode: The initiator of ${p^1}_1$ sends the data message to the target of ${p^1}_1$.

If no satisfactory path, ${p^1}_1$, yet exists between the first two nodes in the route specification, the endpoint representative attempts to form such a path by piecing together existing paths. The endpoint representative attempts to find an existing path whose destination locator is the longest match on the next node's locator and is within the *context* of the two nodes (i.e., the lowest node in the hierarchy that contains both nodes). If such a path, ${p^2}_1$, exists, the endpoint representative proceeds as it did with ${p^1}_1$.

If the endpoint representative fails to find a satisfactory path to any of the second node's ancestral nodes contained within the context, then there are no "direct" paths to the second node. The endpoint representative then seeks a path up to the its node's enclosing node, as there are likely to be more existing paths between higher-level entities. To this end, the endpoint representative checks whether there already exists a satisfactory path whose target is an exit point of the its node's enclosing node. If such a path, ${p^2}_1$, exists, the endpoint representative proceeds as it did with ${p^1}_1$. Otherwise, the endpoint representative attempts to obtain a feasible route and set up a path, ${p^2}_1$. If there are no short-cut paths from the its node's enclosing node to any of the second node's ancestral nodes in the context, the above procedure may need to be repeated for successively higher-level ancestors of the endpoint representative's node, up to but not including the context. If no short-cut paths exist at any of these levels, a route must be generated and a path set up, from the node below the context and containing the first node to the second node.

This iterative path formation procedure is performed by the target of each path thus selected, which then becomes the initiator for the path for the next segment, and so. Note that in the above description, the words "endpoint representative" should be replaced by "forwarding agent" when referring to the actions taken by intermediate agents along a path. The procedure terminates after attaining the last node in the route and the target endpoint representative in that node, possibly linking together paths at many different levels.

In the presence of multilevel paths, each data message carries a nested sequence of path labels, in order to enable all forwarding agents involved in the paths to forward the message correctly. Intermediate forwarding agents update the path labels in the message, according to the linkages between paths stored in their forwarding databases. Upon receipt of a data message, the target of path ${p^j}_i$ finds it is linked to path $p_k^{j-1} = p^j$ which in turn is linked to path ${p^j}_{i+1}$ and hence is the initiator of ${p^j}_{i+1}$. This forwarding agent strips off the label for ${p^j}_i$ and replaces it with the label for ${p^j}_{i+1}$, before forwarding the message along that path.

3.5.2 Path Acceptance

Each setup message in flow mode and each data message in datagram mode contains the route specification and additional service requirements, such as resource reservation requests. A boundary forwarding agent or endpoint representative receiving a setup or datagram message determines message acceptability. Acceptability is in part based on the perceived consistency between the route specification and service requirements contained in the message and the service attributes of each node traversed.

When a forwarding agent refuses a setup message, it informs the other forwarding agents on the path between and including itself and the initiator. At the target, once a setup message passes the service attribute consistency check, it must also pass an endpoint-specific consistency check. In particular, the target determines the perceived consistency between the route specification and service requirements contained in the message and the service requirements of the target endpoint. Each target that accepts a setup message informs the initiator. If there is an inconsistency with the target endpoint's service requirements, the target takes one of two actions, depending upon whether the target is the path's destination or source:

1. If the target is at the destination endpoint, it returns to the initiator a message containing its endpoints' destination service requirements. The initiator is then responsible for obtaining a route and setting up a path that is consistent with both the source and destination service requirements.
2. If the target is at the source endpoint, it returns to the initiator a message indicating that it will generate its own route. The target is then responsible for obtaining a route and setting up a path that is consistent with the source service requirements and the destination service requirements contained in the setup message.

Any forwarding agent or endpoint representative may tear down a path by removing the corresponding forwarding state from its forwarding database. Reasons for path teardown include:

- Detection of a connectivity failure along a path.
- A change in node service attributes or traffic service requirements such that the route on which the path is based is no longer feasible.
- Path expiration if a path exceeds a maximum prescribed lifetime.
- Path preemption in favor of another path.

3.6 Control Message Integrity and Authentication

Nimrod control messages (all messages except data messages are considered to be control messages) include several pieces of information which permit recipient agents to determine whether the message has been corrupted in some way. In addition to information on type and length of various sections of the message, each Nimrod control message contains its generation timestamp, expressed in seconds elapsed since 0 hours on 1 January 1900 (same format as the NTP timestamp [6]), as well as "authentication" information that simultaneously acts as a checksum and as source authentication.

3.6.1 Timestamps

Timestamps establish message recency and hence help recipients detect message replays. In order to detect whether a Nimrod control message is timely, the recipient agent compares its local time with the timestamp contained in the control message. If the timestamp is less recent than the local time by no more than δ seconds or more recent than the local time by no more than ϵ seconds, the message is considered to be timely. Otherwise, the message is considered to be out-of-date. Nimrod agents do not require fine-grained time synchronization in order to make their message recency determinations. Time synchonization on the order of minutes is all that is required. In fact, periodic manual adjusting of local clocks should be sufficient to maintain the necessary synchronization among agents.

3.6.2 Authentication

This initial version of Nimrod does not contain any specification of security measures for Nimrod but rather place holders for such security measures to be introduced in a future version. Nevertheless, we do make recommendations for what these security measures might be.

Most Nimrod control messages are generated by a single agent but distributed to many different agents, and most parts of these messages remain constant as the messages are passed among agents. To prevent communication problems caused by errors introduced into these messages which carrying routing-related information, each recipient agent should be able to determine with high confidence whether the message has indeed been generated by the stated source and whether the constant portions of the message have been modified since being generated by that source.

We recommend that each Nimrod control message carry a public-key-based digital signature covering a reduced form of the constant portions of the message (e.g., apply the MD5 hashing function followed by the RSA signing function to the constant portions of the message). The authentication information may also include the public key to be used to verify the signature together with its certificate. While the RSA signing procedure is computationally intensive, signature verification is not. As long as control message generation at a particular agent is infrequent, that agent should be able to handle the load imposed by signing. Discovery messages are the only control messages generated frequently (i.e., inter-message period on the order of tens of seconds); an alternative mechanism may be required to protect these messages.

The authentication information field in Nimrod control messages is represented as type, length, and value and hence is able to accommodate any integrity and security information that may be desired or required in the future.

4 Reliable Transaction Protocol

Many Nimrod control messages *reliable* delivery. Rather than have each agent duplicate this reliability functionality, Nimrod includes a reliable transaction service, which provides its clients the ability to reliably communicate arbitrary size blocks of information between a client and a peer and to receive an arbitrary size reply in approximately one round-trip time.

The reliable transaction service is built on Transaction-TCP (T/TCP) [10], [9]. T/TCP is a backwards-compatible extension of TCP, which is opti,mized for request/response interactions. In particular, T/TCP may bypass the normal three-way handshake required at TCP connection setup time. This bypass is accomplished by adding a "connection count" option in the TCP header, and by maintaining per-host connection history at both client and server. This information allows the server to correctly distinguish a new connection open (SYN, no ACK) from a duplicate or out-of-order open, without shaking hands with the client. Using T/TCP, the client can obtain a response to a request message in one round-trip-time to the server and back (plus the server's processing time). T/TCP uses the normal three-way close handshake; it does not impact transaction latency.

4.1 Services Interface

The reliable transaction service permits one or more transactions to be invoked at a peer. The service interface is:

- **Flags**, misellaneous flags.
- **Source locator** and **EID** of the transaction.
- **Destination** locator and EID of the transaction.
- **Keying info**, for authentication purposes.
- **Service requirements** for the transactions.
- **Transaction**, beginning with a protocol header (e.g., Query-Response or Update).

Each transaction uses a separate TCP connection. We note that this may cause excessive overhead if the client(s) invoke many transactions within a short period of time, and is an issue to be examined more carefully in future versions of Nimrod. The beginning of each transaction is the transaction header, containing the following fields. The packet formats are illustrated and specified values given in section 9.4.

- **Length**(32 bits) of the transaction, including this field.
- **Version**(2 bits) of Nimrod update and query-response protocols.
- **Protocol**(2 bits) identifier. Whether Update, Query, Response, or Discovery message.
- **Operation**(4 bits). Particular operation within the protocol.
- **Phase**(8 bits). The Update protocol uses several phases for certain operations. This denotes the current phase.
- **Transaction ID**(16 bits). To identify the transaction.

- **Timestamp**(32 bits). Seconds since 1/1/1900, 00:00.

The user may abort an initiated transaction at any time. Note that race conditions are possible as the aborted opertion may have actually been performed by the peer. There is no rollback facility provided by the reliable transaction service.

5 The Update Protocol

The Update Protocol is used to update database contents (e.g., the map database). The peers in the Update Protocol are the Nimrod agents, currently including the forwarding agents, node representatives, route agents, and endpoint representatives. These agents participate in the distribution of the updated information in the required portion of the network. The implicit flooding constituting the protocol is carefully constrained by involving only a few agents per update.

5.1 Service Interface

The Update Protocol offers a distributed database update service in a manner that renders the exact locations of the database transparent to the user. The portion of the distributed database updated is dependent on the particular database and the operation as indicated by the user. The service interface includes the following:

- **Source**. EID and optionally locator of the agent initating the update.
- **Destination**. EID and optionally locator of a specific agent (e.g., endpoint representative whose locator has changed). May be left unspecified.
- **Operation**. Indicates what kind of update (i.e., for what database) it is. Current values are shown in section 9.
- **Keying info** for authentication purposes.
- **Service requirements** if any. Will be "best effort" if unspecified.
- **Patience**. A time interval within which the user wishes to hear about the success of the update.

The Update Protocol provides hop-by-hop reliablity, but no effort is made to ensure end-to-end reliablity. An agent that has initiated an update cannot be certain that the message has been delivered to all intended agents, or that all intended database portions have been updated. We believe that the resource and complexity overhead demanded by an end-to-end reliablity mechanism is not justified by the importance for database updates (note that Nimrod does not require absolute database consistency (see section 2.3.1)). The user may abort a previously initiated update, for instance, because an update is superseded by more recent information. The protocol will discard the update if it has not already been sent out. However, no rollback facility is provided.

5.2 Protocol Operation

The Update Protocol consists of an *Update Message* that is generated by the agent wishing to make an update to a particular database. (The Update Message consists of a variable length database specific portion, described in section 5.3, prepended by a common update protocol header, described in section 5.2.1 below, prepended by the transaction header, described in section 4.) The database may be held redundantly or cooperatively by multiple agents in a node, and an update may involve several nodes in the hierarchy. Thus, the update protocol involves several cooperating communicating agents.

We classify the participating (peer) agents into two for ease of description: the *originating agent* and the *transit agents*. The originating agent forwards the message to one or more agents which further forward it to other agents and so on, until all the necessary database locations have been updated. Once an originating or transit agent has successfully forwarded a message, it does not retain any state corresponding to the message. The originating and transit agent operations are described in more detail later in this section.

The Update Message uses the reliable transaction service (see section 4). Since no effort is made to provide end-to-end reliability, no acknowledgements (positive or negative) are part of the Update Protocol. Exceptions are handled by making a log entry into a file.

The actions performed by an agent upon receipt of an Update Message is a function of the receiving agent type and of the user supplied *Phase* of the message, which are contained in the transaction header. Examples of operation types are map update, locator update, etc. The actions include the decision of whether to cache the message or not, and whether to forward the message further, and if so to whom. We specify such actions corresponding to each agent type (columns) and operation type (rows) pair using an *Update Message Action Table* (UMAT) shown in section 5.2.4.

We note that the update protocol is an application level protocol between a set of peer agents indicated in the destination field of the Nimrod header (or the IP header). In transit between these peer agents, the Update Message map may be forwarded through other intermediate agents, which are not peers in the protocol. For instance, an update message from agent A_1 to agent A_2 may go through (forwarding) agents a_1, a_2, ..., a_k before reaching A_2. However, such an agent a_i is not a peer, and does not act upon the message using the UMAT.

5.2.1 Update Header

Each item in the common update header is explained below. The packet format of the header is illustrated in section 9.5.

- **Originating agent type**(8 bits). The type of agent originating the update. Current agent types include Forwarding Agent, Endpoint Representative, Node Representative, and Route Agent.
- **Destination agents type**(8 bits). The type(s) of agent(s) for whom the update is intended. For multiple agents, the field contains a bitwise-OR of respective agent types.
- **Flags**(16 bits). Miscellaneous operation dependent flags.
- **Database Type**(8 bits). The type of the database that is being updated. At most one kind of database can be updated with an update message.
- **Database timestamp**(24 bits). Denotes the origination time of the message with respect to the originating agent EID. That is, the timestamp and the EID together identify the packet uniquely, modulo wraparounds. The timestamp is the lower 24 bits of the the current time in seconds, beginning 0:00 1 January 1900.
- **Destination NID** (8 bytes). The update is restricted to this node. Refer to section 11.5 for NID format.
- **Originating EID** (8 bytes). EID of the agent that issued the update. Refer to section 11.3 for EID format.

- **Originating Locator**. Locator of the agent that issued the update. Refer to section 11.2 for locator format.
- **Authenticator**. Authentication field. Contains authentication information for the agent originating the update.

5.2.2 Originating Agent Operations

An originating agent issuing the update constructs Update Message database-specific information (update) and fills in the transaction and common update protocol headers, including the timestamp that is incremented for every update originating from the agent. Note that the user-specified Operation is placed in the "operation" field of the transaction header. The UMAT is then consulted to obtain the actions, which typically involve sending the Update Message to one or more next-hop agents. This could be in terms of specific agents, all agents of a given type, or any agent of a given type.

Using the destination agent's EID, locator, etc., the Update Message is enclosed within the appropriate headers (see section 9) and sent. Note that the protocol calls for one-to-one individual transmissions (no multicast) to the next-hop peer agents. If it is required that the message be sent to any one agent of a given type, each agent of that type is tried until successful. An *update failure* occurs if a specified agent is unreachable or if (in case of "any agent of given type") no agent of a given type is reachable. Update failures should be logged for possible network management action.

5.2.3 Transit Agent Operations

A transit agent receives an Update Message as "TCP data". It then performs checks on the message to determine whether the message is a valid one. This may include checking the timestamp in the update header to ensure that the Update Message is not a duplicate, and verifying the authenticator in the update header. If any of the checks fail, the error should be logged for possible network management action.

If the checks pass, then the UMAT is consulted for actions using the Phase field in the transaction header. This may involve caching the information (i.e., updating the relevant database using the message contents) and/or sending the message with a changed Phase, to one or more next-hop agents. This could be in terms of specific agents, all agents of a given type, or any agent of a given type.

Using the destination agent's EID, locator, etc., the Update Message is sent as TCP data. Note that the protocol calls for one-to-one individual transmissions (no multicast) to the next-hop peer agents. If it is required that the message be sent to any one agent of a given type, each agent of that type is tried until successful. An *update failure* occurs if a specified agent is unreachable or if (in case of "any agent of given type") no agent of a given type is reachable. Update failures should be logged for possible network management action.

5.2.4 The Update Message Action Table (UMAT)

The UMAT represents Update Message forwarding instructions based on agent and phase, and depends on what functionality is mapped into the protocol and how the mapping is done. The Update Protocol is used for map updates (section 3.1), locator updates(section 3.3), and adjacency

	F	N	R	E
CLIENT-MAP-UPD		send(1, ,F-P)[1]		
phase-map-forw-par (1)	send(2,P,F-C)			
phase-map-distrib (2)	send(3, ,{R*,N*})			
phase-map-notify (3)		cache	cache	
CLIENT-LOC-UPD		send(4, ,*)[2]		
phase-loc-notify (4)	cache send(5,C,F-P)	cache	cache	cache
phase-loc-child (5)	send(4,*)			
CLIENT-ADJ-UPD		send(6, ,N*)		
phase-adj-notify (6)		cache		

Table 1: Update Message Action Table for each agent type (columns) upon receipt of message with each Operation Type (rows).

.

updates (section 3.4). Our use of the UMAT is mainly to provide a succinct and flexible protocol specification. While it is clearly not necessary that an implementation use an UMAT-equivalent, it is strongly recommended from experience since it provides flexibility by making it easy to change the functionality and the mapping - one simply needs to add additional message types and/or alter the entries in the table. The Update Protocol is typically used by Nimrod agents or other "users" in order to initiate updates. We use the term *client* to denote such users.

For each of the four agent types (Forwarding Agent, Endpoint Representative, Node Representative, and Route Agent), we give below the actions upon receipt of an Update Message of each phase. The phases form the rows, and the value of each phase is indicated within paranthesis. Some operations are client requests, and these are denoted in upper case. Note that to a given agent, only the column corresponding to its agent type is of interest, and thus every agent may be thought of as implementing a column of the UMAT.

The actions primarily involve the functions described, along with their parameter legends, below. We assume the existence of forwarding functionality required to realize these functions.

1. *send(Phase, [Node] , [AgentType][*]).* This sends an Update Message with phase field denoted by Phase to an agent of AgentType in Node. The AgentType is one of N, F, R, E, F-P (boundary to/from parent), or F-C (boundary to/from child). A suffix '*' denotes all agents of the type. If AgentType is omitted, it means all agents in the specified node. For the Node field, P, C and S denote a parent, child, and sibling nodes respectively. If it is absent, it means the current node.

2. *cache.* Update the relevant data structures containing the database.

In the postscript version, the reader may refer to Figures 1 through 4 in Appendix 1 for assistance in understanding the protocol.

5.3 Database Specific Updates

As mentioned earlier, the Update messages contain a database specific information, depending on the operation being performed. In this section, we describe the database specific contents and their semantics for each operation. This information is referred to as "additional information" in the following. The packet formats for the additional fields are illustrated in section 9.6.

5.3.1 Adjacency Updates

After an adjacency has been formed, the node representatives of the nodes constituting the adjacency have to be informed, so that they may modify their maps accordingly. Note that there are two adjacency updates sent for each uni-directional adjacency: one from the node representative that sent the Adjacency Request query and one from the node representative that sent the Adjacency Request response. The additional information in the adjacency updates is:

- **Flags**, indicating whether the adjacency is to a parent, child, or sibling.
- **Neighbor** node NID and locator.
- **Locator** of boundary forwarding agent that implements the adjacency.

5.3.2 Locator Updates

A node representative that changes a locator acquired by an endpoint representative must notify that endpoint representative if the locator changes or becomes unusable, e.g., the association between the node and endpoint is being terminated. The additional information contained in such an update is:

- **Flags**, indicating nature of change (e.g., depreciate use of old locator, terminate use of old locator).
- **Credentials** of the representative originating update.
- **Old locator** that is being changed/terminated.
- **EID** and **locator** (optional) of the supplier of the old locator, if different from the originator. Either both EID and locator are present or both are absent.
- **New locator** (optional) or reassigned locator.
- **Expiration** (optional, present only if new locator is present) time for the new locator.

The representative of a node that acquires a new locator must update all of its children so that they can change their locator. Also, a node representative that changes a locator acquired by a component node must notify that component node if the locator changes or becomes unusable, e.g., the parent-child relationship is being terminated. The additional information contained in such an update is:

- **Flags**, indicating nature of change (e.g., depreciate use of old locator, terminate use of old locator).
- **Credentials** of the representative originating update.
- **Old locator** that is being changed/terminated.

- **EID** and **locator** (optional) of the supplier of the old locator, if different from the originator. Either both EID and locator are present or both are absent.
- **New locator** (optional) or reassigned locator.
- **Expiration** (optional, present only if new locator is present) time for the new locator.

5.3.3 Map Updates

Whenever a node's topology or offered services change, it must generate a new set of maps. The new maps must be propagated to the node representatives and route agents in the node's parent. The maps are also sent to any agents that have explicitly requested to be notified of updates, if an implementation supports the subscription functionality.

- **Flags**. Qualifies the map (e.g., one or more component nodes not in map, map is of a partitioned node).
- **Sequence number** (24 bits) of the transaction that requested explicit (automatic) updates.
- **Map**. Abstract map of node.
- **Maps** (optional) to specific agents as requested.

6 The Query-Response Protocol

The Query-Response (Q-R) Protocol is used to obtain database information(e.g., portions of the map database) in an efficient manner. The Q-R Protocol consists of two messages: the Query Message and the Response Message. (The Query and Response Messages consist of a variable length database specific portion, described in section 6.4, prepended by a common Query/Response Protocol header, described in section 6.3 below, prepended by the transaction header, described in section 4.) The Query Message is generated by the agent wishing to make a query, contains the nature of the information required, and is sent directly to a destination agent that the originating agent believes is in possession of the information. The destination agent obtains the requisite information and sends the Response Message back to the originating agent. Note that the destination agent may obtain the information from its own database, or may in turn send a Query to another agent in order to obtain this information.

6.1 Service Interface

The Q-R protocol offers a reliable query-response service in one round trip time. It uses the reliable transaction service. In fact, excepting headers and minor interface differences, the Q-R protocol adds very little to the service provided by the transaction service.

- **Originator**. EID (and optionally locator) of the agent initiating the query.
- **Destination**. EID (and optionally locator) of the destination agent being queried.
- **Operation**. Indicates what kind of query (i.e., for what database) it is. Current values are shown in section 9.
- **Keying info** for authentication purposes.
- **Service requirements** if any. Will be "best effort" if unspecified.
- **Patience**. A time interval that the user wishes to wait for the response to the query. If there is no response within this time, the user expects to be informed.

6.2 Protocol Operation

Unlike the update protocol, the Q-R Protocol involves only two agents - the originator and destination. The Query Message header (given below) contains the EID and locator of the querying agent. These fields are used by the destination agent for the destination of the response. The destination agent verifies the authentication information to ensure that the query can indeed be honored. Should the authentication check fail or if the destination agent is unable to supply the required information, it still sends a response back with the appropriate error code.

If the originator does not get a response within a certain time t, it is informed of a query failure. The value of t is given to the protocol by the application (e.g., map request). The application also has the option of requesting an abort of the query - in this case, the state is reset and the response is ignored. As in the case of the Update Protocol, exceptions are handled by making a log entry into a file for possible network management action.

6.3 Query/Response Header

Each item in the common Query/Response header is given below. The packet format of the header is illustrated in section 9.5.

- **Originating agent type** (8 bits). The type of agent originating the query. Agent types include Forwarding Agent, Endpoint Representative, Node Representative, and Route Agent.
- **Destination agent type** (8 bits). The type of agent for whom the query is intended.
- **Flags** (16 bits). Miscellaneous operation dependent flags.
- **Database Type** (8 bits). The type of the database to which the information being queried pertains. At most one kind of database can be queried with a query message. Database types are defined in section 9.5.
- **Count** (8 bits). Operation dependent.
- **Opcode** (16 bits) In a query, operation specific or database specific information. In a response, zero if query is being responded to successfully, otherwise an error code indicating the reason for failure.
- **Originating EID**. In a query, the EID of the agent that issued the query. Used to obtain the destination for the response. In a response, the EID of the agent issuing the response.
- **Originating Locator**. In a query, the locator of the agent that issued the query. Used to obtain the destination for the response. In a response, the locator of the agent issuing the response.
- **Authenticator**. Authentication field contains the information used to authenticate the query (in Query) or the response (in Reply).

6.4 Database Specific Request/Release

As mentioned earlier, the Query and Response messages contain a database specific information, depending on the operation being performed. In this section, we describe the database specific contents and their semantics for each operation. This information is referred to as "additional information" in the following. The packet formats for all of the packets in this section are illustrated in section 9.7.

6.4.1 Adjacency Formation

The designated representative of a node forms adjacencies with a neighboring node by sending an adjacency request message to one of the neighboring node's representatives. Any resulting adjacency is one-way, from the node requesting the adjacency to that which granted it. The additional information in an adjacency request Query Message is:

- **Locator** of the node initiating the adjacency.
- **NID** of the node initiating the adjacency.

- **Circuit ID**. Physical circuit identifier of the link to form the adjacency, as known in the originating node.
- **Neighbor NID** of the intended neighbor node in the adjacency.

The additional information in an adjacency request Response Message is:

- **Credentials** of the granting node.
- **Locator** of the granting node.
- **NID** of the granting node.
- **Circuit ID**. Physical circuit identifier of the link to form the adjacency, as known in the granting node.

6.4.2 Adjacency Release

If a node representative receives a positive reply to an adjacency request message, the message may contain information that indicates that the adjacency is not appropriate. An adjacency is terminated by sending an adjacency release request to the node representative which granted the adjacency. Node mobility and management policy changes may also induce a node to release adjacencies. The additional information in an adjacency release Query Message is:

- **Opcode** giving a reason for terminating the adjacency (e.g., policy, lack of traffic, movement, etc.).
- **Locator** of the requesting node.
- **NID** of the requesting node.
- **Circuit ID** of the link forming the adjacency, as known in the requesting node.
- **Neighbor NID** of the granting neighbor node.
- **Circuit ID** of the link forming the adjacency, as known in the granting neighbor.

The adjacency release response contains indication of success or failure (reason code if latter). It does not contain any other message-specific information.

6.4.3 Adjacency Activation

When a node representative has formed an adjacency with a neighbor node, the boundary forwarding agent connected to the neighbor must be informed that Nimrod data traffic may be passed to the neighbor. Note that there are two instances of the Adjacency Activation associated with each uni-directional adjacency, one from the node representative that sent the Adjacency Request query to its boundary forwarding agent indicating outgoing connectivity, and one from the node representative that sent the Adjacency Request reply to its boundary forwarding agent indicating incoming connectivity. The additional information in an Adjacency activation request is:

- **Flags**, indicating whether the adjacency to be activated is to a parent, child, or sibling.

- **Locator** of requesting node.
- **NID** of requesting node.
- **Circuit ID** of the link forming the adjacency as known in the requesting node.
- **Neighbor NID** of granting node.
- **Circuit ID** of the link forming the adjacency as known in the granting node.

6.4.4 Locator Acquisition

There are two forms of locator acquisition: one for a component node requesting its locator from a node representative of the parent node, and one for endpoint representatives requesting a locator for an endpoint from a node representative of the node. The two forms are distinguished by the originating agent type field in the common Query/Response header. The additional information in a locator acquisition Query Message, from a component node is:

- **Old locator (optional)**, previously assigned locator, requestor wants it to be reassigned.
- **EID** and **locator** (optional) of the node representative that previously assigned the locator. Either both EID and locator are present or both are absent.
- **Parent NID** of the node to provide the locator.
- **Child NID** of the node requesting the locator.

The additional information in a locator acquisition Query Message, from an endpoint representative is:

- **Old locator** (optional). Previously assigned locator; requestor wants it to be reassigned.
- **EID** and **locator** (optional), of the node representative that previously assigned the locator. Either both EID and locator are present or both are absent.
- **Provider NID** of the node to provide the locator.
- **Name** of the endpoint requesting a locator.
- **EID** of the endpoint requesting a locator.

The locator acquisition Response Message, for a request from a component node or an endpoint contains the following additional information:

- **Flags** indicating nature of error if any, 0 if okay (e.g., might indicate that another agent may be able to provide it (see below)).
- **New/Reassigned Locator** (optional, present when successful) that the requestor and its descendants should use as prefix.
- **Expiration** time of the locator supplied.
- **EID** (optional) of node representative that might be able to (re)assign the locator.
- **Locator** (optional) of node representative that might (re)assign the locator.
- **NID** (optional) of node containing representative that might (re)assign locator.

6.4.5 Locator Release

The locator release operation is used by a component node or endpoint representative that wishes to release a locator, typically due to mobility/relocation of a network or endpoint. The additional information in a locator release Query Message is:

- **Opcode** indicating reason for release.
- **Flags** indicates if a different agent should be contacted (see below).
- **Old locator** to be released.
- **Issuing NR EID**, that issued the locator that is being released.
- **Issuing NR locator**, that issued the locator that is being released.
- **Different NR EID** (optional, only if flag is set) that might release the locator.
- **Different NR locator** (optional, only if flag is set) that might release the locator.

6.4.6 Map Acquisition

Route agents request maps from node representatives. Since the requesting route agent is both acting on behalf of an endpoint, either through the endpoint representative or a forwarding agent, and possibly on behalf of other route agents that delegated the original route request, there are usually two or more sets of credentials associated with a map request. These credentials are used by the node representative's filter module, which is tasked with enforcing any distribution restrictions on the maps it dispenses. The additional information in a map Query Message is:

- **Flags** qualifying the maps requested. Values allow to indicate: authoritative response required, abbreviated map required, complete map required, basic map required, need automatic map updates.
- **Locator** of the node whose map is desired.
- **Child elements** (optional, if flag does not indicate complete). The locators of children whose map is desired.
- **Credentials** of requesting node.

The map request Response Message contains the following additional information:

- **Flags** qualifiying map supplied. Values allow to indicate: partitioned node, map is of partitioned node, access denied/not available, automatic updates not supported
- **Requested Map**.
- **Child Maps** (optional). Maps of children nodes as requested (partial or complete).
- **Agents** (e.g., in other partitions) that may be able to answer query.

Note that the reply for a basic map may contain several maps, one for the requested node (map) and an abstract map for each of the child component nodes (child maps). The signature of the basic map covers the basic map and each of the maps of the child component nodes.

6.4.7 Map Termination Request

An agent that has explicitly requested to be notified of map updates may choose to terminate that subscription request. The map termination Query Message contains the following additional information:

- **Transaction number** (16 bits) of the map request that requested automatic updates.

The map termination Response Message contains the following additional information:

- **Flags**. Allow to indicate: Automatic updates not supported, try alternate agent.
- **Opcode**. Zero if ok, error code (e.g., no record of automatic update request) otherwise.
- **Agent** (optional) that might have the automatic update request.

6.4.8 Path Information Request

Route agents that are generating multicast routes may require access to existing path information for a multicast group so that more optimal routes may be generated. The path information is distributed among the forwarding agents supporting the multicast group. Route agents use path database queries to obtain the necessary information. The additional information in a path database Query Message is:

- **Flags**. Which info to return.
- **Path Labels** about which info is desired.

The additional information in path database Response Message is:

- **Opcode** indicating error code or zero.
- **Path entries**. List of path entries requested.

6.4.9 Route Generation Request/Reply

Route agents generate routes on behalf of an endpoint in response to requests received from endpoint representatives and forwarding agents. Requests from forwarding agents will contain the credentials of the forwarding agent in addition to those provided by the endpoint representative. These credentials, along with those of the route agent (and any other route agents that delegated the request) are passed to any node representatives when requesting the node's map(s). The additional information in a Route Generation Query Message is:

- **Misc. Flags** qualifying the nature of route required.
- **Count**, number of feasible routes required.
- **Sources**. Source locators for the route.
- **Destinations**. Destination locators for the route.

- **Services** required by the **initiating** endpoint.
- **Services** required by the **target** endpoint.

The route generation Response Message contains the following additional information:

- **Count**. Number of feasible routes returned.
- **Routes** (optional, only if Count is non-zero). A list of routes from source(s) to destination(s) meeting the requirements.

7 Path Management Protocol

Nimrod endpoint representatives and forwarding agents are responsible for establishing, in hosts and routers, state information necessary for forwarding Nimrod data messages. These agents are also responsible for forwarding Nimrod data messages according to this state information and the forwarding directives carried along in the messages. For a particular traffic session, the forwarding state information installed and maintained by endpoint representatives and forwarding agents is derived from the routes selected for the session. The forwarding state corresponding to a particular session and route is called a *path*. Multiple traffic sessions may use the same path, and multiple paths may be established based on the same route. A path may connect one or more source endpoints and one or more destination endpoints. In this initial version of Nimrod, each multicast path is either a source tree or a sink tree. (We note that this version of Nimrod includes procedures for installing and removing forwarding state for multicast trees and for forwarding session traffic along such trees. It, however, does not include procedures for multicast route construction and group management. For more information on these and other multicast issues as they relate to Nimrod, see [4].)

Endpoint representatives and forwarding agents use the *path management protocol* to install and remove state information from their forwarding databases. Each endpoint representative and forwarding agent maintains state information for those paths that originate, terminate, or pass through it. Paths may be set up from source to destination or from destination to source. Each path has an *initiator* and a *target*. We expect that, in most cases, paths will be set up from the representative of the source endpoint to the representative of the destination endpoint. Hence, the initiator usually begins the path setup procedure on behalf of the source endpoint, and the target usually accepts or rejects a path on behalf of the destination endpoint. Forwarding state is established such that path management messages may be forwarded in both directions along the path; data messages always flow from source to destination.

Paths are identified by path labels, which are unique along the path but not necessarily globally unique throughout the internetwork. Multiple non-intersecting paths may carry the same path label. By eliminating the requirement for global uniqueness of path labels, we can allow paths labels to be relatively short (24 bits), thus reducing the cost of carrying them in data messages and the cost of accessing information in forwarding databases indexed by them. The labels for each direction of a path are distinguished by a bit that indicates whether the message is flowing from from source to destination or from destination to source.

Endpoint representatives and forwarding agents try to form a path for a new session by piecing together existing paths, rather than by setting up entirely new paths, provided the existing paths meet the new session's service requirements and permit its traffic to flow over them. This method of path construction incurs the least cost, in terms of the amount of route generation and forwarding state installation required per session. In a busy internetwork, there are likely to be many existing paths. Moreover, we expect that most traffic sessions will not require specific service guarantees and most networks will not refuse to carry this "best effort" traffic. Therefore, we expect this method of path constructruction to be much less expensive than individually setting up and maintaining distinct paths for every traffic session.

Most Nimrod paths are likely to be composed of paths which in turn are composed of lower-level paths, and so on. A Nimrod flow-mode data message (refer to section 3.5 for a description of flow mode and datagram mode message transmission) travelling over one of these multilevel, multi-segment paths must carry the path labels of all component paths it is currently traversing.

These path labels are stacked in the data message and manipulated, i.e., pushed and popped, by the agents handling the message. Note, however, that an endpoint representative or forwarding agent uses only one of these path labels at a time in making a forwarding decision for the message.

7.1 Protocol Messages

The path management protocol uses five types of messages: *setup*, *accept*, *teardown*, *status*, and *ack*. Message contents are described below, but explicit formats are depicted in section 9. Each path management message is covered by the same basic types of integrity and authentication checks as other Nimrod control messages, including checks on message length, timestamp validity, content corruption, and source authentication. (Refer to section 3 for more information on Nimrod control message integrity and authentication.)

All path management messages travel along the path to which they refer. Endpoint representatives and forwarding agents respond to the receipt of a path management message in different ways, depending upon the type of agent and the type of message. Path management protocol messages may be used not only to set up and teardown a path, but also to collect and report performance monitoring information for a path (e.g., path delay and throughput). Path monitoring operates in two modes: *collection* and *report*. Hence, monitored information for a path may appear in a message as information being collected or reported.

7.1.1 Setup

A setup message is generated by the initiator and travels along the path toward the target. It is used to establish forwarding state in endpoint representatives and forwarding agents. In this initial version of Nimrod, all endpoint representatives and forwarding agents retain the setup message for each active path that traverses them. This copy is used for detection of duplicate setup messages and for selecting alternate lower-level paths if the one initially chosen fails. Each setup message contains:

1. The label of the path to be established.
2. The time at which the setup message was originally generated.
3. Path indications:
   (a) Source-initiated or destination-initiated.
   (b) Unicast or multicast.
   (c) Available for shared use by multiple sessions or dedicated to a single session.
4. Requested services for the session, indicated as either requirements or preferences and expressed as type, length, and value, from the perspective of the initiator. These may include but are not limited to:
   - delay;
   - variation in delay;
   - throughput;
   - variation in throughput;

- bit error rate;
- packet error/loss rate;
- monetary cost (per byte, per packet, or per unit time);
- whether packet order must be preserved.

   They may also include information about the session itself, such as its expected lifetime, the type of organization to which the originator belongs (e.g., academic, government, commercial), and the characterization of session traffic (e.g., in terms of average and peak rates and burst durations). As the work of the Integrated Services working group progresses, we plan to integrate these service specifications into future versions of Nimrod.

5. The route, in terms of the locators of the nodes through which it must pass and, for each of these nodes, the label of the relevant connectivity specification to invoke across the node.

6. The EIDs of the source and destination endpoints and their representatives.

   (a) Unicast path setup. The EID information for the initiator is included to enable the target to establish a path in the reverse direction, if desired. It is also useful for network management. The type of the target agent determines what EID information is included for the target in the setup message.

      i. The target is the representative of one of the session's endpoints. Hence, EID information about this specific target must be included in the setup message, in order for the path to connect to the intended endpoint.
      ii. The target is any boundary forwarding agent for the last node on the route. Hence, there is no specific target and thus EID information related to the target may be left unspecified in the setup message. Such a situation is likely to arise when constructing a higher-level path intended to carry traffic for multiple sessions.

   (b) Multicast path setup. Although source and destination EID information is not strictly necessary in this case, it is included for network management reasons. Note that the multicast path is either a source tree or a sink tree. Hence, EID information is included for either the source or the destination but not for both.

7. Collection-mode monitored information expressed as type, length, and value. This information can be used to determine the actual services available over the path. The initiator determines whether to collect this information during path setup.

8. Integrity and authentication information covering all but collection-mode monitored information.

7.1.2 Accept

An accept message is generated by the target and travels along the path toward the initiator. It is used to indicate to the initiator that the path has been successfully established. Each accept message contains:

1. The label of the accepted path.

2. The EID of the agent that generated the accept message.

3. The time at which the accept message was generated.

4. Report-mode monitored information, expressed as type, length, and value. If the setup message collected monitored information, the accept message reports this information as the services available over the path.

5. Integrity and authentication information covering all of the above.

Provided the initiator is the source, it may send data over the path before it receives an accept message from the target. Circumstances under which an initiator may wish to wait for an accept message before sending data on a path include the following example. If the source pays for all data messages sent, whether or not they are successfully received at the destination, it may want to wait to make sure that the path is successfully established before sending data to the destination. In this case, the source should be prepared to buffer data received from the host until the accept message is received. An initiating source determines whether to wait for an accept message before transmitting data, based upon the session's requested services.

7.1.3 Teardown

A teardown message is generated by any endpoint representative or forwarding agent on the path. It usually travels outwards, in both directions, towards the initiator and the target. In some cases (e.g., incomplete path setup or teardown generation by initiator or target), the teardown message may travel in only one direction. Teardown messages are used to remove forwarding state in endpoint representatives and forwarding agents. Each teardown message contains:

1. The label of the path to be torn down.

2. The EID of the agent that generated the teardown message.

3. The time at which the teardown message was generated.

4. The reason for the teardown, expressed as type, length, and value. A teardown message may be generated in response to any of the following events:

   Type 1: A path timeout. Each path has a specified maximum lifetime, in order to ensure that forwarding state is eventually removed, no matter how a path might fail. The agent detecting the path timeout generates two teardown messages, sending one toward the initiator and one toward the target.

   Type 2: Setup message is out-of-date. A setup message is out-of-date if the absolute value of the difference between its timestamp and the local time kept by the recipient agent varies by more than a specified maximum value. The agent receiving the out-of-date setup message generates a teardown message and sends it toward the initiator.

   Type 3: Route specification carried in the setup message is inconsistent with the node.

      Subtype 1: Recipient agent's node does not appear in the route specification.

      Subtype 2: Connectivity specification carried in the setup message is not a valid connectivity specification for the recipient's node.

   These situations might indicate that the setup message has been corrupted in an undetected way, that a route agent used an out-of-date or corrupted map for the node when constructing the route, or that a setup message was misdelivered. In any case, the agent unable to recognize the node or connectivity specification generates a teardown message and sends it toward the initiator.

Type 4: Conflict between the path and either the services provided by the recipient's node or the session service requirements from the target's perspective.

Subtype 1: During path setup, an agent detects a conflict between the path and the services provided by its node (as reflected in the node's connectivity specifications) such that the node refuses to carry the session traffic or cannot meet the session service requirements. The agent generates a teardown message and sends it toward the initiator.

Subtype 2: During path setup, the target detects a conflict between the path and the session service requirements from its perspective such that the path fails to meet these requirements. The target generates a teardown message containing its service requirements and sends the message toward the initiator. In this case, the initiator will attempt to find a path that is consistent with the target's service requirements as well as its own.

Subtype 3: After path establishment, a forwarding agent detects a change in its node's services (as reflected in the node's connectivity specifications), which conflicts with the path such that either the node refuses to carry the session traffic or cannot meet the session service requirements. The agent generates a teardown message and sends copies toward the initiator and target.

Subtype 4: After path establishment, the initiator or target detects a change in session service requirements which conflicts with the path such that the path fails to meet the new requirements. The initiator (or target) generates a teardown message and sends it toward the target (or initiator).

Type 5: Preemption in favor of another path. The preempting agent generates a teardown message and sends copies toward the initiator and target. Endpoint representatives and forwarding agents are free to implement their own preemption criteria, but these are not part of this initial version of Nimrod. In this version, all paths are established on a first-come, first-served basis, with no preemption.

Type 6: Unresolvable path label collision during path setup. (Refer to the discussion of status and ack messages below and to section 7.2.3 below for information on collision resolution.) The agent unable to resolve the path label collision generates a teardown message and sends it toward the initiator.

Type 7: Insufficient resources to any next-hop agent during path setup. The agent detecting a lack of resources to reach a specific next-hop agent attempts to find a suitable alternate next-hop agent. If it fails to find an alternate agent, the agent generates a teardown message and sends it to the previous-hop agent on the path.

Subtype 1: Insufficient space in the forwarding database. The agent has no room for another entry in its forwarding database.

Subtype 2: Insufficient resources for the path (e.g., unable to reserve required capacity for the session).

Type 8: Loss of connectivity in the path. An agent may detect a loss in path connectivity through neighbor discovery (see section 8.1), agent discovery (see section 8.2), or through the loss of a lower-level path forming part of the given path.

Subtype 1: An agent detecting a downstream loss to a specific next-hop agent attempts to repair the path by sending the original setup message to an alternate next-hop agent. If it fails to find an alternate agent, the agent generates a teardown message and sends it to the previous-hop agent on the path.

Subtype 2: An agent detecting an upstream loss sets a repair timer. If the timer fires before the agent detects the path is repaired, the agent generates a teardown message and sends it to the next-hop agent on the path. An agent detects a repaired path through receipt of a copy of the original setup message (described in more detail in section 7.2).

Type 9: Initiator exceeds specified maximum number of setup attempts. The initiator generates a teardown message and sends it toward the target, in order to clear any partially established forwarding state for the path.

5. Report-mode monitored information, expressed as type, length, and value. If the setup message collected monitored information, and the teardown is generated in response to the setup message, then the teardown message reports this information as the services available over the path thus far.

6. Integrity and authentication information covering all of the above.

7.1.4 Status

Status messages may be generated by any agent along a path, in order to report path information or modify path characteristics. Each status message contains:

1. The label of the path to which the status pertains.

2. The EID of the agent that generated the status message.

3. The time at which the status message was generated.

4. The reason for the status message, expressed as type, length, and value. A status message may be generated for any of the following reasons:

   Type 1: Path monitoring. The initiator (or target) generates a status message containing collection-mode monitored information and sends it toward the target (or initiator). The ultimate recipient (which is usually the target or initiator but which may be an intermediate forwarding agent along the path if the path has failed in some way) responds by generating a status message containing report-mode monitored information and sends it toward the initiator (or target).

   Type 2: Path lifetime extension. The initiator generates a status message containing the amount of time by which to extend the path's life, hence preventing a path teardown prior to session cessation.

   Type 3: Replacement path label. When a path label contained in a setup message collides with a path label already in use at a forwarding agent or endpoint representative, that agent usually generates an ack message containing a replacement label for the new path that the previous-hop agent should use when sending messages along the new path toward this agent. In some cases, the agent may instead generate status messages containing replacement labels to use for an existing path, one for the previous-hop agent and one for the next-hop agent on the path. For example, if the existing path has not been used in a long time, the agent may choose to alter the forwarding information for that path rather than for the new path, in order to speed processing of data flowing along the new path.

Type 4: Unrecognized requested session service contained in a setup message. This might indicate that the setup message has been corrupted in an undetected way or that the initiator is requesting new service requirements not yet known throughout the internetwork. In any case, the agent unable to recognize the session service request generates a status message, sends it toward the initiator, and ignores the unknown service when making next-hop decisions.

Type 5: Failure to transmit a setup message successfully over a path hop. An agent detects this failure through an indication of unsuccessful transmission provided by the mechanism for hop-by-hop reliability (refer to the ack message discussion below for more information). It then generates a status message and sends it toward the initiator. In response to this type of status message, the initiator may either resend the original setup message or teardown the established portion of the path.

Type 6: Unrecognized path label carried in a data message. This might indicate that the data message has been corrupted in an undetected way or that the agent receiving the message has failed recently and has lost state concerning the paths previously established through it. In any case, the agent unable to recognize the path label generates a status message and sends it to the agent from which it received the data message containing that path label.

5. Integrity and authentication information covering all but collection-mode monitored information.

7.1.5 Ack

The path management protocol provides reliable transmission of setup, accept, teardown, and status messages between successive sending and receiving agents along a path. After transmitting one of these messages along a path, the sending agent expects to receive, within a specified period of time, an ack message acknowledging successful receipt of the message at the next agent. If no ack message is forthcoming, the sending agent retransmits the setup, accept, teardown, or status message up to a specified maximum number of times. Furthermore, if the sending agent fails to receive an ack message after the specified number of retransmissions, it logs the event for possible network management action. Ack messages are generated by each agent along a path in response to the receipt of a setup, accept, teardown, or status message. Each ack message contains:

1. The label of the path to which the ack pertains.

2. The EID of the agent that generated the ack message.

3. The time at which the ack message was generated.

4. In the case of a path label collision, the replacement path label to use when sending messages along that path toward this agent.

5. Integrity and authentication information covering all of the above.

Given that the setup portion of the path management protocol is already reliable end to end (i.e., the initiator expects to receive either an accept or teardown message in response to its setup message and retransmits the setup message if no response if forthcoming within a specifed time period), one might consider hop-by-hop reliability overkill. Note that in lossy networks, the additional hop-by-hop reliability increases the reliability and responsiveness of the path management protocol and reduces the number of end-to-end path setup message retransmissions required for successful path establishment.

In this version of Nimrod, the path management protocol, unlike the other Nimrod protocols, uses its own reliability mechanism rather than T/TCP. A simpler and more appealing design is one that employs a single reliable transaction protocol for all Nimrod control messages, either T/TCP or perhaps a Nimrod-specific transaction protocol. The reason for treating path management messages differently from other Nimrod control messages is performance. In the prototype implementation of Nimrod, much of the path management functionality has been placed in the packet handling "fast path" while T/TCP is not. We expect that other implementations would likely be structured similarly. To improve message handling efficiency, the path management protocol uses its own separate mechanism for reliability of message transmissions. Thus, for this initial version of Nimrod, practical considerations have prevailed in the area of packet handling.

7.2 Protocol Finite-State Machines

The path setup procedure for this initial version of Nimrod operates as follows. An endpoint representative initiates path setup either under control of network management or when it receives, from an endpoint it represents, a data message that requires Nimrod routing and for which no existing path is suitable. After determining the location of the message's destination (by consulting its local locator cache or by querying the DNS) and obtaining a feasible route to that destination (by consulting its local route cache or by querying a route agent), the endpoint representative generates a setup message and sends it toward the target. The endpoint representative expects to receive an accept message from the target, indicating successful path establishment, within a specified time interval. If the endpoint representative fails to receive an accept message for the path within the allotted time interval, it retransmits the setup message, provided it has not yet exhausted its permitted number of setup attempts. The endpoint representative may make a specified maximum number of path setup attempts, in an effort to establish the path successfully.

Unsuccessful path establishment manifests itself as either failure to receive an accept message after the maximum number of setup attempts or receipt of a teardown message instead of an accept message for the path. In either case, the initiator removes the path's state from its forwarding database and the corresponding route from its route cache, so that it does not use that route again immediately. It also logs the event for possible network management action. Moreover, when the initiator exhausts its maximum number of setup attempts and does not receive a teardown message, it generates a teardown message of type 9 (exceeded maximum number of setup attempts) and sends it toward the target, in order to remove any partially established forwarding state for the path.

The agents in which a path's state is stored include boundary forwarding agents for nodes listed in the path's route specification, as well as endpoint representatives. When a boundary forwarding agent at the entrance to a node X receives a setup message for a path p, it performs a set of consistency and resource availability checks, in order to determine whether to install state for p in its forwarding database. Provided it does not reject p, the boundary forwarding agent installs state relating to the previous hop on p. For example, if the setup message for p arrives on a lower-level path q from agent G, the boundary forwarding agent installs state information in its forwarding database, indicating that the previous hop for p is via q and ends at agent G. It then looks up the next node Y in the route specification carried in the setup message.

If Y is adjacent to X, the boundary forwarding agent attempts to reach any boundary forwarding agent in X participating in the adjacency to Y.

Case 1: X supports Nimrod routing internally. The boundary forwarding agent attempts to obtain a path to the boundary of X adjacent to Y, which may also involve obtaining a route. If there is an existing path r to the boundary of X adjacent to Y that is consistent with the session's requested services, the boundary forwarding agent sends the setup message for p over r. If there is no such path r, but a route consistent with the session's requested services does exist, the boundary forwarding agent initiates establishment of a path r for that route. It then installs information in its forwarding database, indicating that the next hop for p is via r, and then sends the setup message for p over r. Upon receipt of the accept message for r, the initiating boundary forwarding agent learns the identity of the next-hop agent H for p and then installs this information in its forwarding database entry for p.

Case 2: X does not support Nimrod routing internally. The boundary forwarding agent picks a boundary forwarding agent H participating in the adjacency with Y (the existence of such boundary forwarding agents is learned through agent discovery, described in section 8.2). It then installs information in its forwarding database, indicating that the next-hop agent for p is H and sends the setup message to H.

In either case, H then proceeds to install state for p in its forwarding database and to send the setup message on toward a boundary forwarding agent for Y.

If Y is not adjacent to X, then the boundary forwarding agent attempts to obtain a path to Y, which may also involve obtaining a route to Y. If there is an existing path r to Y that is consistent with the session's requested services, the boundary forwarding agent sends the setup message for p over r. If there is no such path r, but a route consistent with the session's requested services does exist to Y, the boundary forwarding agent initiates establishment of a path r to Y. It then installs information in its forwarding database, indicating that the next hop for p is via r, and then sends the setup message for p over r. Upon receipt of the accept message for r, the initiating boundary forwarding agent learns the identity of the next-hop agent H for p and then installs this information in its forwarding database entry for p.

There are two finite-state machines for the path management protocol, one applicable to the initiator and one applicable to the target or any intermediate forwarding agent in the path. The difference in state machines arises because the accept message contents only affect the behavior of the initiator. All state transitions caused by receipt of messages are described below. Except in one case, receipt of unexpected messages, such as an accept following a teardown, does not cause state changes, further propagation of these messages, or issuing of new messages. The exceptional case is the receipt of a Nimrod data message with an unrecognized path label. This event causes no state transitions nor propagation of this message, but it does cause the recipient agent to generate a status message of type 6 and send it to the agent from which it received the unexpected data message. All anomalous and error events in the path management protocol are logged for possible network management action.

7.2.1 Initiator

The state machine for the initiator has four states: *idle*, *check*, *ready*, and *done*. State transitions are described below:

- idle $\rightarrow$ check: This transition occurs when the initiator begins to set up a path. The initiator then performs the consistency and resource availability checks for the path (described below).
- check $\rightarrow$ idle: This transition occurs if the initiator cannot successfully complete the consistency and resource availability checks for the path because a check failed.
- check $\rightarrow$ ready: This transition occurs after the initiator has successfully completed all of the consistency and resource availability checks for the path. The initiator then installs state for the path, including next-hop information, in its forwarding database, sends the setup message to the next hop, starts the accept timer, and initializes the setup transmission count. At this point, the initiator may begin to send data traffic over the path or may withhold data traffic until receipt of an accept message, depending upon the session service requirements.
- ready $\rightarrow$ done: This transition occurs when the initiator receives an accept message from the target. The initiator then stops the accept timer for the path. If the initiator had been withholding data traffic until this point, it may now send this traffic over the path.
- ready $\rightarrow$ check: This transition occurs when:
  - The initiator receives from the next-hop agent a teardown message of type 7.1 (insufficient space in forwarding database), 7.2 (insufficient path resources), or 8 (loss of path connectivity). It then attempts to repair the path by sending a copy of the original setup message to an alternate next-hop agent.
  - The initiator receives from the next-hop agent a status message of type 6 (unrecognized path label in data message) containing a path label matching one of its own. It then seeks to re-establish the path by resending the original setup message to the next-hop agent.
- ready $\rightarrow$ idle, done $\rightarrow$ idle: This transition occurs when:
  - The initiator generates a teardown message for the path because the path lifetime expires, the node's offered services or the session service requirements have changed in a way that is inconsistent with the path, the path is preempted, or the accept timer fires before the path is successfully established. It then generates a teardown message and sends the message toward the target.
  - The initiator is unsuccessful at finding an alternate next-hop agent, following receipt of a teardown message of type 7.1 (insufficient space in forwarding database), 7.2 (insufficient path resources), or 8 (loss of path connectivity).
  - The initiator receives a teardown message of type 1, 2, 3, 4, 5, or 6.

  The initiator then removes the state information for the path from its forwarding database.

7.2.2 Intermediate Forwarding Agents and Target

The state machine for the intermediate forwarding agents and the target has three states: *idle*, *check*, and *ready*. There is no *done* state for these agents, because the accept message is only meaningful to the initiator. State transitions are described below:

- idle → check: This transition occurs when the agent receives a setup message for a new path. The agent then performs the consistency and resource availability checks for the path (described later on).
- check → idle: This transition occurs when:
  - The agent cannot complete the consistency and resource availability checks for the path, because a check failed.
  - The repair timer fires before completing the checks.
  - The agent receives from the previous-hop agent, a teardown message of type 4.3 (change in node offered services), 4.4 (change in session service requirements), 8 (loss of path connectivity), or 9 (exceeded maximum number of setup attempts), before completing the checks.

  The agent then removes any partial state it has installed for the path.
- check → ready: This transition occurs after the agent has successfully completed all of the consistency and resource availability checks for the path. If the repair timer is ticking, the agent stops the timer. The agent then installs state for the path, including previous hop and next hop and sends the setup message to the next hop. From the agent's perspective, the path may be used to carry data traffic.
- ready → check: This transition occurs when:
  - The agent receives, from the next-hop agent, a teardown message of type 7.1 (insufficient space in forwarding database), 7.2 (insufficient path resources), or 8 (loss of path connectivity). It then attempts to repair the path by sending a copy of the original setup message to an alternate next hop.
  - The agent receives, from the next-hop agent, a status message of type 6 (unrecognized path label in data message) containing a path label matching one of its own. It then attempts to re-establish path state by resending a copy of the original setup message to the next hop.
  - The agent receives, from the previous-hop agent, a setup message for the path. This may occur when an upstream agent attempts path repair.
- ready → idle: This transition occurs when:
  - The agent generates a teardown message for the path, because the path lifetime expires, the node's services or the session service requirements have change in a way that is inconsistent with the path, the path is preempted, or the repair timer fires before the path is repaired.
  - The agent receives from the previous-hop or next-hop agent a teardown message of type 1, 2, 3, 4, 5, 6, or 9 or receives from the previous-hop agent a teardown message of type 8 (loss of path connectivity).

The agent then removes the state information for the path from its forwarding database and sends the teardown message in the path direction indicated in the message.

When in the check or ready states, if the agent detects a loss in connectivity with the previous-hop agent on the path, it sets the repair timer and waits for attempted path repair. The agent generates a teardown message of type 8 (loss of path connectivity), if the repair timer fires before the previous-hop state information is repaired, and sends the message toward the target.

7.2.3 Check State Actions

When in the check state, an endpoint representative or forwarding agent must perform a series of tests to determine whether to install forwarding information for the path. These tests are partitioned into two sets, those related to consistency between the path and the services provided over the next hop and those related to resource availability over the next hop. Note that Nimrod datagram-mode data messages carrying requested session services and route specifications are subject to the same consistency checks as setup messages, even though no session-specific forwarding state is installed for these messages. Consistency checks performed by the intermediate forwarding agents and the target include the following:

- The setup message is timely. Detection of out-of-date setup messages can help prevent denial of service attacks caused by replays. If the setup message is out of date, the agent sends a teardown message of reason type 2 (out-of-date setup message), toward the initiator.
- The path label carried in the setup message is not already in use. If the path label is already in use, one of the following holds:

  Case 1: The timestamp and route specification carried in the setup message are identical to the timestamp and route specification for an already-established path.

  - If the previous-hop agent for the setup message (previous-hop information appears in messages as the source EID carried in the EID option, described in section 9), is the same as the previous-hop agent for the existing path, the agent assumes that the setup message is a duplicate submitted by the initiator subsequent to the firing of the accept timer for the path. The agent then forwards the setup message along the path toward the target to enable any downstream agents to establish state for the path, in case they failed to receive the original copy of the setup message.
  - If the previous-hop agent for the setup message is different from the previous-hop agent for the existing path, the agent assumes that the setup message is a duplicate submitted by an upstream agent attempting path repair. The agent updates its previous-hop information for the path in its forwarding database and does not forward the setup message.

  Case 2: The timestamp or route specification carried in the setup message are different from those of any already-established path.

  - If the setup message is for a unicast path, the agent assumes that the setup message is for a new path. The agent then attempts to obtain an alternate label either for the new path or for the existing path with which the new path label collides.

    Subcase 1: The agent attempts to select a non-colliding path label for the new path if the most recent use of the existing path occurred within a specified interval relative to the current time. If the agent's supply of usused path labels is exhausted, the agent generates a teardown message of type 6 (unresolvable path label collision) and sends it toward the initiator of the new path. Otherwise, the agent installs the appropriate forwarding state and selects a next-hop agent for the new path. It then includes in its ack message for the previous-hop agent and in its setup message for the next-hop agent, the replacement path labels to use when sending messages over the path and through this agent. Finally, it sends the ack message to the previous-hop agent and the setup message on toward the target.

Case 2: The agent attempts to select a non-colliding path label for the existing path if the most recent use of the existing path occurred outside a specified interval relative to the current time. In this case, the agent installs the appropriate forwarding state, selects a next-hop agent, and sends the setup message on toward the target. The agent also changes the forwarding state for the existing path. If the agent's supply of usused path labels is exhausted, the agent generates a teardown message of type 6 (unresolvable path label collision) and sends it toward the initiator of the existing path. Otherwise, the agent substitutes the replacement path label in the forwarding state for the new path. It then generates two status messages of type 3, sending one to the previous-hop agent and one to the next-hop agent, on the existing path. These status messages contain the replacement path labels to use when sending messages over the existing path and through this agent.

– If the setup message is for a multicast path, the agent assumes that the setup message is for a new branch of a multicast tree, other branches of which may already exist. The agent then installs new forwarding state for the indicated branch of the tree and sends the setup message toward the target.

Consistency checks performed only by the intermediate forwarding agents include the following:

- The route specification carried in the setup message is consistent with the node. Specifically, the recipient agent's node appears in the route specification and the connectivity specification contained in the route specification is valid for this node. If either of these conditions is not satisfied, the agent generates a teardown message of type 3.1 (node not present in route specification) or 3.2 (invalid connectivity specification) and sends the message toward the initiator.

- The path is consistent with the node's offered services. Specifically, the node permits the session traffic to use its resources and provides the required services. If these conditions are not satisfied, the agent generates a teardown message of type 4.1 (service conflict during setup) and sends the message toward the initiator.

Consistency checks performed only by the target include the following:

- The path is consistent with the session service requirements for the target. If this condition is satisfied, the target generates an accept message and sends it toward the initiator. Otherwise, the target generates a teardown message of type 4.2 (target service conflict) and sends the message toward the initiator. The teardown message contains the service requirements from the target's perspective. Upon receipt of such a teardown message, the initiator is responsible for attempting to obtain a feasible route that accounts for the session service requirements from the target's perspective.

Resource availability checks performed by all agents include the following:

- The forwarding database can accommodate state for the new path. If there is insufficient room in the forwarding database, the agent sends a teardown message of type 7.1 (insufficient space in forwarding database) to the previous-hop agent.

Following successful completion of this check, the agent installs the previous-hop information in its forwarding database so that path management messages can flow in both directions along the path. Resource availability checks performed by the initiator and intermediate forwarding agents include the following:

- There exists a way to reach the next node in the route specification, which provides the required services for the session and permits access to session traffic. This check is extensive and may require additional route generation and path setup for lower-level paths. To illustrate, let the current path be p.
  - The selected next-hop agent is directly connected to the agent, and thus there is no need to establish a lower-level path. In this case, the agent simply installs the next-hop information in its forwarding database and sends the setup message for p to the next-hop agent.
  - There is an already-established feasible lower-level path q with the required resources to a forwarding agent in the next-hop node. In this case, the agent links p with q in its next-hop entry for p in its forwarding database and sends the setup message for p to the next-hop agent, using q.
  - There is no established feasible lower-level path to the next-hop node. In this case, the agent attempts to obtain a route for, set up, and obtain the necessary resources for a path q. The agent may either piggyback the setup information for q on the setup message for p or send a separate setup message for q, depending upon the session service requirements. Piggybacking reduces the time to set up a path at the expense of flexibility in collecting monitored information (monitored information can only be collected for one path at a time). If session traffic is allowed to flow before confirmation of successful path setup, the agent piggybacks the setup information for the two paths and links p and q in its next-hop entry for p in its forwarding database. Otherwise, the agent sends a separate setup message for q, waits for receipt of an accept message for q before linking p and q in its next-hop entry for p in its forwarding database, and then sends the setup message for p to the next-hop agent, using q.
  - A feasible lower-level path with the necessary resources cannot be established, either because no route exists or the path setup fails. In this case, the agent generates a teardown message of type 7.2 (insufficient path resources) and sends the message to the previous-hop agent on p.

A teardown message of type 7.1 (insufficient space in forwarding database) or 7.2 (insuffcient path resources) received from an upstream neighbor during path setup may result in the recipient agent resending the setup message to another next-hop agent, if one is available. Only if there are no suitable next-hop agents does the agent propagate the teardown message toward the initiator.

8 Discovery Protocols

Each Nimrod agent acting on behalf of a node requires knowledge about its immediately neighboring agents and about the other agents acting on behalf of the same node, in order to perform its routing functions properly. Agents continually execute a neigbor discovery protocol and an agent discovery protocol in order to obtain current information about each other. Neighbor discovery and agent discovery can be performed prior to hierarchical locator assignment, because forwarding of these messages requires knowledge of NIDs and EIDs only. Hence, the discovery protocols may be used to bootstrap the Nimrod routing system.

8.1 Neighbor Discovery

Neighbor discovery is performed by activated Nimrod agents within a node and provides the initial information for the formation of adjacencies between nodes and the formation of associations between endpoints and nodes. It enables an agent to determine whether there are other Nimrod agents reachable within one "hop" and if so which agents. A hop may constitute a direct link (point-to-point or multiaccess) between two agents or a virtual link (composed of many intervening links and routers that are not Nimrod-capable).

When the hop is a virtual link, each agent at the end of the link must be configured with location information (e.g., IP address) for the agent at the other end, so that the neighbor discovery messages are distributed to the appropriate agent. Note that even when the hop is a direct link, the agent may require some location information configuration or discovery (e.g., IP router discovery) prior to Nimrod neighbor discovery; these requirements depend upon the specific network in which these Nimrod agents reside. Specifically, an agent must know the type of link attached to each of the interfaces of the device in which it resides, so that it can invoke the appropriate low-level protocols. For example, if an agent resides in a device attached to an Ethernet that also uses IP, that agent may use a combination of ARP, dynamic host configuration, and router discovery to determine its IP address and the IP address of a neighboring router. These functions must be executed prior to Nimrod neighbor discovery.

A Nimrod agent periodically issues a neighbor discovery message over each of its interfaces through which a Nimrod-capable neighbor is expected to exist. Neighbor discovery messages are transported using T/TCP (with IP destination address set equal to the "all devices" multicast address) but are never retransmitted. Each neighbor discovery message contains the following information concerning the issuing agent:

1. Type.
2. The time at which the message was generated.
3. Node on whose behalf it acts, expressed as an NID.
4. EID.
5. Locator(s), when available.
6. Set of nodes in which it resides, expressed as a hierarchical sequence of NIDs.
7. Additional location information (e.g., IP address contained in the IP header section).

8. Integrity and authentication information covering all but the additional location information.

When a Nimrod agent X receives a neighbor discovery message, it knows that it has a neighboring agent Y. From then on, X expects to continue to receive periodic neighbor discovery messages from Y. Failure to receive neighbor discovery messages from a known neighbor indicates a reachability problem, caused by a physical component's malfunction or movement. An agent detecting a failure to reach a neighboring agent must alert the agent discovery protocol (see section 8.2) and path management protocol (see section 7) operating within it. Agent discovery uses this information to determine which agents are reachable, and path management uses this information to determine which paths require repair.

8.1.1 Neighbor Reachability

The periodic exchange of neighbor discovery messages works as an "up/down" protocol, such that from an agent's perspective the link to the neighbor is in one of two states, "up" or "down". Each agent maintains a sliding window for each neighboring agent. Each window slot corresponds to one period and contains either a "hit" for receipt of a message or a "miss" for failure to receive a message. In addition to the sliding window, the agent also records the number of hits during the current period and number of misses over the current window.

The neighbor link is always considered "down" initially until the agent receives a sufficient number of messages from it to consider it to be "up". Whenever the neighbor link is down, the values of the protocol parameters are set as follows:

- The sliding window size is equal to m.
- Each window slot contains a miss.
- The number of hits during the current period is equal to 0.
- The number of misses within the window is equal to m.

When the neighbor link moves from down to up, the values of the protocol parameters are set as follows:

- The sliding window size is equal to n.
- Each window slot contains a hit.
- The number of hits during the current period is equal to 0.
- The number of misses within the window is equal to 0.

At the conclusion of each period, an agent counts the misses and determines whether there has been a state transition in the neighbor link. When the neighbor link is down, a miss count of no more than $m - j$ signals a transition to the up state. In the up state, a miss count of no less than $n - k$ signals a transition to the down state.

Counting the misses involves several steps. First, the agent prepares to slide the window by one slot so that the oldest slot disappears, making room for the newest slot. However, before sliding the window, the agent checks the contents of the oldest window slot. If this slot contains a miss, the agent decrements the number of misses by 1, as this slot is no longer part of the current window. After sliding the window, the agent initially records a miss in the newest window slot and then determines what the proper slot contents should be. If the number of hits for the current period equals 0, a miss is the correct value for the newest slot, and so the agent increases the number of misses by 1. Otherwise, if the number of hits for the current period is greater than 0, the agent applies the hits to any slot containing a miss, beginning with the newest and progressing to the oldest such slot. For each such slot, the agent records a hit in that slot and reduces the number of hits by 1. If the selected slot is not the newest slot, the hit cancels out an actual miss, and so the agent reduces the number of misses by 1 as well. The agent continues to apply each remaining hit to any slot containing a miss, until either all such hits are exhausted or all such slots are accounted for. Before beginning the next up/down period, the agent resets the number of hits to 0.

Although we expect the number of hits, within any given period, to be no greater than 1, we do anticipate the occasional period in which an agent receives more than one message from a neighbor. The most common reasons for this occurrence are message delay and clock drift. When a message is delayed, the recipient agent observes a miss in one period followed by two hits in the next period, one of which cancels the previous miss. Excess hits remaining in the tally after miss cancellation indicate a problem, such as clock drift. Thus, whenever an agent accumulates excess hits, it logs the event for potential network management action.

When clock drift occurs between two agents, it causes the period of one agent to grow with respect to the period of the other. Let p_X be the period for agent X, let p_Y be the period for agent Y, and let g and h be the smallest positive integers such that $gp_X = hp_Y$. Suppose that $p_X < p_Y$ because of clock drift. In this case, X observes $g - h$ misses in g consecutive periods, while Y observes $g - h$ surplus hits in h consecutive periods. As long as $\frac{g-h}{g} < \frac{n-k}{n}$ and $\frac{g-h}{g} \leq \frac{m-j}{m}$, the clock drift itself will not cause the neighbor link to enter or remain in the down state.

8.2 Agent Discovery

Agent discovery is performed by all Nimrod agents within a node. It enables agents acting on behalf of the same node to learn how to reach each other and hence to communicate before Nimrod inter-node routing is fully operational. Note that agents resident within a node may be configured in advance with location information pertaining to other agents in that node instead of participating in agent discovery.

During agent discovery, each agent acting on behalf of a node periodically generates and distributes an advertisement about itself. Agent discovery messages are transported using T/TCP (with IP destination address set equal to the next-hop IP address learned through neighbor discovery). Each advertisement message contains the following "link-state" information for the issuing agent:

1. Type.
2. The time at which the message was generated.
3. Node on whose behalf it acts, expressed as an NID.
4. EID.
5. Locator(s), when available.

6. Set of nodes in which it resides, expressed as a hierarchical sequence of NIDs.
7. Additional location information (e.g., IP address).
8. The following information for each neighboring agent:
   (a) EID.
   (b) Services offered on the link to that neighbor.
9. Integrity and authentication information covering all of the above except the additional location information and also covering the following information.

Each boundary forwarding agent also includes in its advertisements the following information for each neighboring boundary forwarding agent external to the node:

1. Node on whose behalf it acts, expressed as an NID.
2. Locator(s), when available.
3. Set of nodes in which it resides, expressed as a hierarchical sequence of NIDs.
4. Whether the advertising agent currently participates in an adjacency (either inbound or outbound) with that neighboring node.

In a Nimrod node that does not execute Nimrod routing internally, the agent discovery procedure should be tailored to the information distribution mechanisms native to that node. Consider the following examples. Broadcasting agent advertisements is the method of choice, if the node is a multiaccess network with broadcast capability (e.g., an Ethernet). Flooding agent advertisements may be appropriate, provided the node supports a native flooding mode. Multicasting agent advertisements may be the most efficient strategy, if the node supports multicast distribution (e.g., an IP network that supports IP multicast). In this case, all agents in the node should be configured with a multicast address specific to those agents, which they use when distributing agent advertisements. Moreover, specific types of agents may also have their own multicast addresses (e.g., all boundary forwarding agents for a node may share a multicast address).

8.2.1 Flooding Agent Advertisements

In a Nimrod node that supports Nimrod routing internally, the agent discovery procedure is a hop-by-hop reliable flooding procedure among agents within a node. Each agent periodically generates an advertisement and distributes it to each of its neighboring agents that acts on behalf of the same node as itself or on behalf of a child node. Periodic distribution ensures that all agents acting on behalf of the same node keep current with respect to their mutual reachability.

Upon receipt of an advertisement, an agent decides either to accept and forward the advertisement or to reject the advertisement. An advertisement is acceptable if it is more recent than any thus far received from the advertising agent and if the recipient agent's node is the same as or is a child of the advertising agent's node. If the advertisement is acceptable, the agent stores the advertised agent information locally and determines where to redistribute the advertisement. Otherwise, the agent discards the advertisement. Forwarding agents are the only type of agents that redistribute advertisements received from neighboring agents.

When a forwarding agent receives an acceptable advertisement, it includes its EID (referred to as the "next-hop" EID) in the message. As we explain below, this information will be used by the next recipient to determine how to reach the agent that generated the advertisement. The agent then distributes the advertisement to all agents, for which it has forwarding entries, that act on behalf of the same node as the advertising agent or that are boundary forwarding agents acting on behalf of child nodes of the advertising agent. These prospective receipients may be neighboring agents, or they may be other more distant agents learned about through agent discovery. To reach more distant agents, the agent uses special forwarding hints discussed below. Note that the agent never transmits the message back to the "next-hop" agent from which the advertisement was received.

For each acceptable advertisement received, an agent records the next-hop EID contained in the advertisement, thus establishing forwarding information indicating how to reach the agent in the advertisement. Note that this procedure is similar to distance-vector routing, with the exception that the routes thus learned are the first and not necessarily the shortest. With this procedure, no persistent routing loops may form. The reasons are as follows: each agent knows of at most one way to reach a particular agent, and no agent distributes an advertisement to the next-hop agent from which it was received. Temporary reachability problems may occur, however, if an agent or a link between agents fails.

Node representative and route agents acting on behalf of a node receive advertisements from all agents acting on behalf of the same node. The node representatives use the link-state information contained in these advertisements to construct the node's maps. The route agents use the information to generate intra-node routes at the request of other agents in the same node and may tailor these routes to the specific service requirements of the requesting agents.

8.2.2 Distribution of Advertisements to Distant Agents

Suppose a boundary forwarding agent A acts on behalf of node X contained in node Y. The agent discovery procedure operates at two different levels, one for each nodes. Through agent discovery within X, all agents acting on behalf of X learn about each other. Moreover, through neighbor discovery, A learns of any neighbors that act on behalf of child nodes of X. Furthermore, boundary forwarding agents for X, such as A, learn about agents that act on behalf of Y. Hence, A can build up hierarchical forwarding information for reaching agents that act on behalf of X, Y, or children of X.

Suppose that A receives from outside X an advertisement for an agent acting on behalf of Y. A must then distribute a copy of this advertisement to boundary forwarding agent in X. For each such agent B, A includes the EID of the destination agent in the route portion of the forwarding header (see section 9, SETUP). It then determines, using forwarding information obtained through agent discovery within X, which neighboring agent C, acting on behalf of X, is the next hop on the way to B and sends the message to C. When C receives the message, it ignores the payload and simply acts on the forwarding information, determining its next hop toward B; it does not act on the agent advertisement information itself. Thus, advertisements for agents in higher-level nodes can tunnel through lower-level nodes.

8.2.3 Unreachable Agents

The following procedure ensures that agents acting on behalf of the same node quickly learn when an agent becomes unreachable. If an agent Q loses connectivity with a next-hop agent R, all agents learned through R become unreachable from the perspective of Q. In this case, Q floods a message indicating the set of agents that are now unreachable from its perspective. Note that all of these unreachable agents will be acting on behalf of the same node N.

Each agent X receiving this unreachability message compares the list of unreachable agents contained in the message and with the list of agents supposedly reachable through the next-hop agent from which the message was received. If no agent is present on both lists, then X reaches the listed agents through other next-hop agents and so X may safely discard the unreachability message. If at least one agent is present on both lists, then X can no longer reach those agents. In this case, X constructs an unreachability message containing all such agents. It distributes the message to all agents, for which it has forwarding entries, that either act on behalf of N or are boundary forwarding agents acting on behalf of children of N, but it does not redistribute the message to the agent from which it was received. X also removes its forwarding entries pertaining to the unreachable agents.

With the above procedure, all agents affected by a connectivity problem between agents learn which agents they can no longer reach. The periodic distribution of agent discovery messages ensures that agents will learn of alternate routes to currently unreachable agents, if such routes exist. Moreover, by installing multiple agents of each type throughout a node, a network manager may minimize the probability that a particular agent is unable to reach any agents of a particular type acting on behalf of the same node. Furthermore, all route agents acting on behalf of a particular node have a complete picture of all other agents acting on behalf of the same node, and they can attempt to generate alternate routes when reachability problems occur. As long as an agent can reach at least one route agent acting on behalf of the same node, it can request a route to agents that appear to be currently unreachable.

A boundary forwarding agent detecting a failure to reach another boundary forwarding agent for the same node must alert the management protocol (see section 7) operating within it. Path management uses this information to determine which paths require repair. Node representatives that learn of unreachable boundary forwarding agents through agent discovery may need to construct a new abstract map if the loss of agent reachability affects the existing abstract map. We discuss this case in more detail in the next section.

8.2.4 Node Partitions

Failure of physical assets in a Nimrod internetwork may lead to the partitioning of a Nimrod node, such that assets in one part of the node cannot communicate with assets in the other part of the node by traversing assets within the node. There are two cases:

1. A group of assets in a node are completely isolated from the rest of the internetwork.
2. No assets are isolated from the internetwork, but assets in one part of a node must use assets external to the node in order to communicate with assets in another part of the node. This is the more interesting case and the one discussed below.

A node's representatives detect node partitions of the second type, using information received through agent discovery. Specifically, each node representative tracks the reachability of all boundary forwarding agents and all node representatives. When a node representative detects that some boundary forwarding agents are no longer reachable, it concludes that the node has partitioned into multiple pieces.

These multiple pieces of the same node are not new nodes. Each piece has the same locator and NID of the original node. However, each piece may have a different map. All node representatives within a given piece have the same knowledge of that piece of the node. A single mutually agreed upon node representative resident in that piece of the node may generate a new map for that piece, if the loss of reachability of a boundary forwarding agent has resulted in a change in the node's abstract map. Note that if the node representative cannot reach any boundary forwarding agents, there is no point in generating a new map for the node, because no one outside that piece of the node will be able to obtain the map.

Representatives resident in each such piece of the node generate a different map for the node. These maps are distinguished by the EID of the node representative generating the map and are distributed throughout the parent node. Route agents receiving these maps maintain multiple maps for the node. In fact, when generating routes that include the partitioned node, a route agent uses the node's maps as though they were constructed by different nodes.

During a partition, when an endpoint in one piece of the node wishes to communicate with another endpoint in the same node, it may have to use a route that goes outside of the node. First, the representative acting on behalf of the source endpoint determines the locator and EID of the destination endpoint (and its representative), by querying the DNS using the name of the destination endpoint.

If the source and destination locators are different, the source's representative then attempts to obtain a route (usually from a route agent in the parent node) to the destination. This route agent may need to ask each piece of the node for a more detailed map, containing its component nodes, in order to determine which piece contains the source and which piece contains the destination. Once it determines which pieces contain which session endpoints, the route agent can generate a route connecting them and passing through external nodes to do so.

If the source and destination locators are the same, the source endpoint's representative determines whether it can reach the destination's endpoint representative, using information obtained through agent discovery. Only if it cannot reach the destination endpoint's representative, does the source endpoint's representative ask a route agent for a route. This route agent does not know which piece each resides in, and generates multiple routes, one from each of the two pieces of the node to the other. The endpoint representative can determine which route is feasible by attempting to set up paths; one path setup is likely to fail because a node along the way will be unreachable. This trial-end-error setup, while not the most efficient, is reasonable, because we expect that nodes will seldom partition into more than two pieces.

When the failed resources work again and the node partition disappears, a node representative generates a new abstract map for the node. Maps generated during the partition might still exist in the route agent's map database after the partition disappears. These maps will eventually be removed from that database, because all maps have finite lifetimes. Moreover, the new map for the whole node is available for use immediately, and even when the old maps are usable. The only drawbacks to keeping the old maps around for awhile are the storage space they consume, and the slight amount of extra work they cause during route generation.

8.3 Route Tracing

By participating in the agent discovery protocol, each agent acting on behalf of a node constructs forwarding entries for all other agents acting on behalf of the same node. This forwarding information is sufficient to enable a node representative to perform such tasks as locator acquisition for its node (see section 6.4. Recall that locator acquisition involves a query to the parent node representative. Agents within a node can forward such a query to a representative of the parent node, using only the forwarding information obtained during neighbor and agent discovery. This forwarding information, however, is not always sufficient to enable the parent node's representative to return a response to the requesting node representative; additional information may be required.

Nimrod includes a route tracing feature that enables agents handling a message, such as a locator request, to include their NID as a part of a message trailer. Thus, as the message progresses to its destination agent, the route (expressed in terms of NIDs of nodes traversed) is accumulated in the message. The destination agent can then use this route to return its response to the source agent. (Refer to section 9 for the specific format of traced routes and their use as forwarding directives). Although this feature is available to all Nimrod messages, we expect that its use will be confined to node locator request and response messages.

9 Packet Formats

9.1 Overview

The "syntax" of a Nimrod packet is given below in Backus-Naur notation. Each "terminal" is shown in uppercase and without the delimiters <, >, and its packet format with field values is illustrated in the sections as indicated at the end of that line. The formats below assume IP (V.4 or V.6) as the network layer protocol.

```
<Nimrod Pkt> ::= <header><payload>

<header>     ::= <IP hdr>  AUTH-HDR  ESP-HDR                    (section 9.2)

<IP hdr>     ::=  IP-V.4-HDR | IP-V.6-HDR                       (section 9.2)

<payload>    ::=  <forw hdr> ( <data pkt> | <path pkt> |
                             <uqr pkt>  | <disc pkt> ) <forw-tlr>

<forw hdr>   ::=  PATH-STK SETUP | PATH-STK                     (section 9.3)

<forw tlr>   ::=  TRACE | MONITOR | TRACE MONITOR               (section 9.3)

<data pkt>   ::=  DATA                                          (section 9.3)

<path pkt>   ::=  ACK | ACCEPT | TEARDOWN | STATUS              (section 9.3)

<uqr pkt>    ::= <ttcp hdr> ( <upd pkt> | <q-r pkt> )

<disc pkt>   ::= <ttcp hdr> ( <n-d pkt> | <a-d pkt> )

<ttcp hdr>   ::=  TTCP-HDR NIM-XACT-HDR                         (section 9.4)

<upd pkt>    ::=  UPDATE-HDR <upd oper>                         (section 9.5)

<q-r pkt>    ::=  Q-R-HDR <q-r oper>                            (section 9.5)

<upd oper>   ::=  MAP-UPDATE | LOC-UPDATE | ADJ-UPDATE          (section 9.6)

<q-r oper>   ::=  MAP-REQ | LOC-REQ | ADJ-REQ | ROUT-REQ | PATH-REQ (sec 9.7)

<n-d pkt>    ::=  DISC-HDR                                      (section 9.8)

<a-d pkt>    ::=  DISC-HDR AGT-ADV                              (section 9.8)
```

In the packet formats given in subsequent sections, fixed length fields are bordered by "|", and variable length fields are bordered by ":". In case of the latter, a structure name is attached in parentheses along with the field, and the exact packet format of the structure name is given in Appendix 2.

9.2 IP and Security Headers

Nimrod does not require all routers and hosts to be Nimrod capable. Only Nimrod agents need to be able to recognize Nimrod-specific information in messages. Nimrod information exchaned between Nimrod agents is encapsulated within packets prefaced by the native forwarding header of the network. For this version of Nimrod, we have specified encapsulation within IPv4 [7] and IPv6 [13]. In the future, one might consider migration of some Nimrod-specific information into the IPv6 header as options (e.g. route specifications might appear as a route option). For now, we have elected to keep the Nimrod-specific information largely separate from the IPv6 header until we have gained more experience with Nimrod.

```
IP-V.4-HDR ::=

        +=======+=======+===============+===============================+
        |   4   |HdrLen |Type of Service|       Packet Length           |
        +-------+-------+---------------+-------------------------------+
        |               IP ID           |           Fragment            |
        +---------------+---------------+-------------------------------+
        | Time to Live  |   Protocol    |           Checksum            |
        +---------------+---------------+-------------------------------+
        |                      Source IPv4 Address                      |
        +---------------------------------------------------------------+
        |                    Destination IPv4 Address                   |
        +===============================================================+


IP-V.6-HDR ::=

        +=======+=======+===============================================+
        |   6   | Prio  |               Flow Label                      |
        +-------+-------+---------------+---------------+---------------+
        |       Payload Length          |  Next Header  |   Hop Limit   |
        +-------------------------------+---------------+---------------+
        |                                                               |
        |                      Source IPv6 Address                      |
        |                                                               |
        |                                                               |
        +---------------------------------------------------------------+
        |                                                               |
        |                      Destination IPv6 Address                 |
        |                                                               |
        |                                                               |
```

```
+==============================================================+
```

There exists a new IP option (protocol number 60) to carry the EIDs for the source and destination of the message. This option is described in draft-ietf-nimrod-eid-00.txt and is pictured below.

```
+---------------+---------------+---------------+---------------+
|  Next Header  | Ext Len = 2   |   PadN = 1    | PadN Len = 0  |
+---------------+---------------+---------------+---------------+
| Opt EID = A3  | EID Len = 18  |  Src Len = 8  |  Dst Len = 8  |
+---------------+---------------+---------------+---------------+
|                          Source EID                           |
|                                                               |
+---------------------------------------------------------------+
|                        Destination EID                        |
|                                                               |
+---------------------------------------------------------------+
```

The IP security headers, including IP authentication [11] (protocol number 51) and encryption [12] (protocol number 50), follow immediately after the IP forwarding headers, thus enabling the recipient to authenticate the message before parsing the rest of it. These headers are not required but instead are inserted at the discretion of the agent sending the message.

```
AUTH-HDR ::=

        +---------------+---------------+-------------------------------+
        |  Next Header  |    Length     |           RESERVED            |
        +---------------+---------------+-------------------------------+
        |                              SPI                              |
        +---------------------------------------------------------------+
        |                                                               |
        |                      Authentication Data                      |
        |                                                               |
        |                                                               |
        +---------------------------------------------------------------+


ESP-HDR ::=

Clear   +---------------------------------------------------------------+
 Text   |                              SPI                              |
======  +---------------------------------------------------------------+
```

```
Cypher  |                                InitVec                         |
 Text   +----------------------------------------------------------------+
        |                                Data                            :
        +                                +---------------+---------------+
        :       Padding                  |  Pad Length   |  Next Header  |
        +--------------------------------+---------------+---------------+
```

9.3 Nimrod Forwarding Information

Nimrod forwarding information (protocol number 43) is contained in every Nimrod message, following the IP-related headers. This forwarding information consists of four possible sections: a set of paths to be setup or used; a set of routes that the message should follow; the route traversed thus far; and monitored information collected or reported along the path. Not all Nimrod messages contain all portions, as we describe in more detail below, but all Nimrod messages carry a path-stack section. Moreover, a message might carry no path labels in its path stack section (e.g., a response to a query that requires source-directed forwarding). There are several special bits in the path-stack section that indicate which other sections are present and how to parse them.

The S bit indicates whether the path indicated by the label already exists and is to be used (S = 0) or does not yet exist and should be installed (S = 1). Only data messages and path management messages carry path labels.

The R bit indicates whether the path direction is from source to destination (R = 1) or from destination to source (R = 0). At each forwarding agent along a path, there is a forwarding entry for each direction of the path (previous hop and next hop). Data messages flow in the source-to-destination direction. Path management messages may flow in either direction, but the direction must be indicated in the path label.

The C bit indicates whether there has been a path label collision (C = 1) during path setup. If there has been a collision, the first path label with the C bit set is the original path label and the second path label with the C bit set is the replacement path label. The next agent to receive this message will use the replacement path label when forwarding messages toward the previous agent along the path.

The T bit indicates whether or not there is route tracing in progress. Routing tracing is used to collect routing information during queries that occur prior to locator assignment, so that the response can return to the agent that placed the query, even if the queried agent does not know the locator of the requesting agent. Route tracing is normally only used in query messages, and then only prior to locator assignment.

The M bit indicates whether there is monitored information being collected in the message (M = 1). Monitored information is normally only collected in path management messages.

```
PATH-STK ::=

+------ (Header Length + 1) * 8
| +---- (Section Length + 1) * 8
| | +-- Path Offset
| | |
```

```
v v v
=====    +===============+===============+===============+===============+
^ ^ ^    |  Next Header  | Header Length |  Version = 1  |  Hops to Go   |
| | |    +---------------+---------------+---------------+-+-+-+-+-+-----+
| | |    |  Type=Forward | Section Length|  Path Offset  |S|C|M|R|T|  0  |
| | |    +---------------+---------------+---------------+-+-+-+-+-+-----+
| | v    :              Available for additional Path Labels             :
| | -    +---------------+-----------------------------------------------+
| |      |S:0 1 1:C:0 0 0:R:      Path Label n                           |
| |      +---------------+-----------------------------------------------+
| |      |S:0 1 1:C:0 0 0:R:            ...                              :
| |      +---------------+-----------------------------------------------+
| |      |S:0 1 1:0:0 0 0:R:      Path Label 1                           |
| |      +---------------+-----------------------------------------------+
| v      |S:0 1 1:0:0 1 0:R:      Path Label 0                           |
|====== +===============================================================+
|
```

All messages that require source-directed forwarding contain the following section, which carries routes. These messages include setup messages themselves, datagram-mode messages, and responses to queries prior to locator assignment (in this case, the route contains a sequence of NIDs and "best effort" connectivity specifications). The authentication field carried in each route covers all of the common information except the route offset, destination EID, destination endpoint representative EID, and all of the information pertaining to that route.

In a setup messages, the flags indicate whether the path is source-initiated or destination-initiated, unicast or multicast, sharable among sessions or dedicated to a particular session, and whether the requested services are preferences or requirements. The destination endpoint and endpoint representative may be designated as "unknown" in some cases. Setup message may simultaneously set up more than one path. In this case, multiple routes are carried in a single setup message.

```
SETUP ::=

|
| +------- (Section Length + 1) * 8
| | +----- (Common Length + 1) * 8
| | | +--- route offset
| v v v
|====== +===============+===============+===============+===============+
| ^ ^    |  Type=Paths   |Section Length | Subtype=Setup |     Flags     |
| | | |  +---------------+---------------+---------------+---------------+
| | | |  |<route offset> | Common Length |              0                |
| | | |  +---------------+---------------+-------------------------------+
| | | | |                   Message Generation Timestamp                  |
| | | |  +---------------------------------------------------------------+
```

```
| | | | :                     Requested Services                      :
| | | | +-------------------------------------------------------------+
| | | | |                         Source EID                          |
| | | | |                                                             |
| | | | +-------------------------------------------------------------+
| | | | |              Source Endpoint Representative EID             |
| | | | |                                                             |
| | | | +-------------------------------------------------------------+
| | | | |                       Destination EID                       |
| | | | |                                                             |
| | | | +-------------------------------------------------------------+
| | | | |           Destination Endpoint Representative EID           |
| | | | |                                                             |
| | | | +-------------------------------------------------------------+
| | v | ?                         64-bit Pad                          ?
| |---| +=============================================================+
| |   v :                       Free/64-bit pad                       :
| |  -- +---------------+---------------+---------------+-------------+
| |     |  Type=Route   | Route Length  |       0       | <hop offset> |
| |     +---------------+---------------+---------------+-------------+
| |     :                       Authentication                        :
| |     +-------------------------------------------------------------+
| |     :                           Route n                           :
| |     :            a sequence of hops each represented as           :
| |     :  length, connectivity specification, and node locator or nid :
| |     +-------------------------------------------------------------+
| |     :                             ...                             :
| |     +---------------+---------------+---------------+-------------+
| |     |  Type=Route   | Stack Length  |       0       | <hop offset> |
| |     +---------------+---------------+---------------+-------------+
| |     :                       Authentication                        :
| |     +-------------------------------------------------------------+
| v     :                           Route 0                           :
|=====  +---------------+---------------+---------------+-------------+
|
```

All path management messages are transmitted reliably, hop by hop along the path. The ack message indicates the type of message acknowledged. If the type of message acknowledged is a setup message and the C bit is set in the path label acknowledged, then the message contains a replacement path label to use in the opposite direction along the path. In the ack, accept, status, and teardown messages, the authentication field covers the entire section of the message.

```
ACK ::=

|
| +------- (Section Length + 1) * 8
```

```
| v
|===    +==============+==============+==============+==============+
| ^     |  Type=Paths   | Section Length|  Subtype=Ack  |     Flags     |
| |     +---------------+---------------+---------------+---------------+
| |     |Type Msg Acked |                       0                       |
| |     +---------------+-----------------------------------------------+
| |     |                  Message Generation Timestamp                 |
| |     +---------------------------------------------------------------+
| |     |                Path Label of Acknowledged Message             |
| |     +---------------------------------------------------------------+
| |     |                     Replacement Path Label                    |
| |     +---------------------------------------------------------------+
| |     :                         Authentication                        :
| |     +---------------------------------------------------------------+
| v     ?                           64-bit Pad                          ?
|===    +---------------------------------------------------------------+
|
```

ACCEPT ::=

```
|
| +------- (Section Length + 1) * 8
| v
|===    +==============+==============+==============+==============+
| ^     |  Type=Paths   | Section Length| Subtype=Accept|     Flags     |
| |     +---------------+---------------+---------------+---------------+
| |     |                               0                               |
| |     +---------------------------------------------------------------+
| |     |                  Message Generation Timestamp                 |
| |     +---------------------------------------------------------------+
| |     |                      Accepted Path Label                      |
| |     +---------------------------------------------------------------+
| |     :                         Authentication                        :
| |     +---------------------------------------------------------------+
| v     ?                           64-bit Pad                          ?
|===    +---------------------------------------------------------------+
|
```

Status and teardown message contain information pertaining to the reason for the status or teardown message.

STATUS ::=

```
|
| +------- (Section Length + 1) * 8
| v
|===    +==============+==============+==============+==============+
| ^     |  Type=Paths   | Section Length| Subtype=Status|    Flags      |
| |     +---------------+---------------+---------------+---------------+
| |     |  Status Code  |  Code Length  |          Code Data            :
| |     +---------------+-----------------------------------------------+
| |     |                 Message Generation Timestamp                  |
| |     +---------------------------------------------------------------+
| |     |                          Path Label                           |
| |     +---------------------------------------------------------------+
| |     :                        Authentication                         :
| |     +---------------------------------------------------------------+
| v     ?                          64-bit Pad                           ?
|===    +---------------------------------------------------------------+
|
```

TEARDOWN ::=

```
|
| +------- (Section Length + 1) * 8
| v
|===    +==============+==============+==============+==============+
| ^     |  Type=Paths   | Section Length|Subtype=Teardwn|     Flags     |
| |     +---------------+---------------+---------------+---------------+
| |     | Teardown Code |  Code Length  |          Code Data            :
| |     +---------------+-----------------------------------------------+
| |     |                 Message Generation Timestamp                  |
| |     +---------------------------------------------------------------+
| |     |                          Path Label                           |
| |     +---------------------------------------------------------------+
| |     :                        Authentication                         :
| |     +---------------------------------------------------------------+
| v     ?                          64-bit Pad                           ?
|===    +---------------------------------------------------------------+
|
```

The route tracing section is optional and usually only contained in query messages transmitted prior to locator assignment. The route collected in the message can be reversed to send the response back to the requesting agent. In this case, the traced route is expressed in terms of NIDs instead of locators.

```
TRACE ::=

|
| +------- (Section Length + 1) * 8
| | +----- free offset
| v v
|====== +==============+==============+=============+===============+
| ^ ^   |  Type=Trace   | Section Length|      0       | <free offset> |
| | |   +--------------+--------------+--------------+---------------+
| | v   :                         Free/64-bit pad                     :
| |---  +-------------------------------------------------------------+
| |     :<locator id/len>                <locator/nid n>              :
| |     +-------------------------------------------------------------+
| |     :                              ...                            :
| |     +-------------------------------------------------------------+
| v     :<locator id/len>                <locator/nid 0 (source)>     :
|===    +-------------------------------------------------------------+
|
```

Monitored information is normally carried only in path management messages. There are two types of monitored information, collection-mode and report-mode. Setup messages carry only collection-mode information and accept and teardown carry only report-mode information; status messages may carry either type. Each of the monitored items is expressed in terms of a service specification (e.g., delay, MTU, throughput).

```
MONITOR ::=

|
| +------- (Section Length + 1) * 8
| v
|===    +==============+==============+==============================+
| ^     |  Type=Monitor | Section Length|Subtype=Cl/Rp |     Flags     |
| |     +--------------+--------------+------------------------------+
| |     |                       <monitored item 1>                    |
| |     +-------------------------------------------------------------+
| |     :                              ...                            :
| |     +-------------------------------------------------------------+
| |     |                       <monitored item n>                    |
| |     +-------------------------------------------------------------+
v v     ?                          64-bit Pad                         ?
====    +-------------------------------------------------------------+
```

9.4 Transaction Headers

```
TTCP-HDR ::=

        (Transaction) TCP Header (IPPROTO_TCP = 6)

        +-------------------------------+-------------------------------+
        |               Src Port        |       Dest Port = 1617        |
        +-------------------------------+-------------------------------+
        |                       Sequence Number                         |
        +---------------------------------------------------------------+
        |                       Acknowledgement Number                  |
        +--------+-----------+-+-+-+-+-+-+-------------------------------+
        |TCP Len|           |U|A|P|R|S|F|               Window          |
        +--------+-----------+-+-+-+-+-+-+-------------------------------+
        |               Checksum        |       Urgent Pointer          |
        +---------------+---------------+-------------------------------+
        |    MSS = 2    |    Len = 4    |       Maximum Segment Size    |
        +---------------+---------------+-------------------------------+
        | CC=11/.NEW=12 |    Len = 6    |               Seg.cc
        +---------------+---------------+---------------+---------------+
                                        |  CC.Echo = 13 ?    Len = 6    ?
        +-------------------------------+---------------+---------------+
        ?                               Seg.cc                          ?
----    +---------------------------------------------------------------+


NIM-XACT-HDR ::= (Nimrod) Transaction Header (TCP Dest Port = 1617)

        +---------------------------------------------------------------+
        |                       Transaction Length                      |
        +---+---+-------+---------------+-------------------------------+
        |Ver|Prt| Oper  |     Phase     |     Transaction Identifier    |
        +---+---+-------+---------------+-------------------------------+
        |                          NTP Seconds                          |
----    +---------------------------------------------------------------+

          Ver value  : 0

          Prt values :

                   0 = Discovery Protocol
                   1 = Update Protocol
                   2 = Query Protocol
                   3 = Response Protocol


         Oper values :
```

```
                When Prt = 1 :

                    1 = Adjacency update
                    3 = Endpoint locator update
                    4 = Node locator update
                    5 = Map update

                When Prt = 2 or 3 :

                    1 = Adjacency Activation
                    2 = Adjacency Release
                    3 = Adjacency Request
                    4 = Endpoint locator acquisition
                    5 = Node locator acquisition
                    6 = Locator release
                    7 = Map request
                    8 = Map update termination
                    9 = Path request
                   10 = Route request

     Phase values : 10 = forward map update to parent node FA
                    12 = distribute map update to NR and RA
                    13 = forward locator update to child node FA
                    14 = notify all agents in node of new locator
                    16 = notify adjacency to NR
```

9.5 Update, Query, and Response Protocol Headers

```
UPD-HDR ::=

    +---------------+---------------+-------------------------------+
    |  org agt typ  |  dst agt typ  |             flags             |
    +---------------+---------------+-------------------------------+
    |    db_type    |                  db sequence                  |
    +---------------+-----------------------------------------------+
    :                     authenticator    (NimAuth)                :
    +---------------------------------------------------------------+
    :                     dst NID          (NimNID)                 :
    +---------------------------------------------------------------+
    :                     originator's EID    (NimEID)              :
    +---------------------------------------------------------------+
    :                     originator's Locator  (NimLoc)            :
    +---------------- ----------------------------------------------+
```

```
agt typ values :

     0    = Any endpoint, not a Nimrod agent.
     0x01 = Association Agent (for future use).
     0x02 = Endpoint Representative.
     0x04 = All Forwarding Agents.
     0x08 = Boundary to child Forwarding Agent.
     0x10 = Boundary to parent FA.
     0x20 = Node Representative.
     0x40 = Designated Node Representative.
     0x80 = Route Agent.
     0xA0 = Node representative and Route Agents.
     0xA6 = All agents in the node.
   0x1000 = Neighboring FA.
   0x2000 = Neighboring FA in parent.
   0x4000 = Neighboring FA in child.

db type values :

 1 = Abstraction DB                2 = Agent DB
 3 = Adjacency DB                  4 = ARP DB
 5 = Association DB                6 = DNS DB
 7 = Endpoint DB                   8 = End-to-End Session DB
 9 = Forwarding Agt DB            10 = Forwarding path branch DB
11 = Forw. Conn. Spec DB          12 = Forwarding flood DB
13 = Filtering DB                 14 = Forwarding neighbor DB
15 = Forwarding path DB           16 = Forwarding route DB
17 = Intercept endpoint DB        18 = Intercept session DB
19 = Key DB                       20 = Locator DB
21 = IP Multicast forwarding DB   22 = Map DB
23 = Map registration DB          24 = Node DB
25 = Route DB                     26 = IP unicast forwarding DB
```

Q-R-HDR ::=

```
+---------------+---------------+-------------------------------+
|   org agt typ |   dst agt typ |              flags            |
+---------------+---------------+-------------------------------+
|    db_type    |       count   |              opcd             |
+---------------------------------------------------------------+
:                      authenticator      (NimAuth)             :
+---------------------------------------------------------------+
:       Q: Originator's or R: Responder's EID       (NimEID)    :
+---------------------------------------------------------------+
:       Q: Originator's or R: Responder's Locator   (NimLoc)    :
```

```
+---------------------------------------------------------------+
:                       credentials                (NimCred)   :
+---------------------------------------------------------------+
```

9.6 Update Operation Messages

ADJ-UPDATE ::=

```
+---------------+---------------+---------------+---------------+
:                 Neighbor node id (NimNID)                     :
+---------------+---------------+---------------+---------------+
:                 Neighbor node loc (NimLoc)                    :
+---------------+---------------+---------------+---------------+
:                Boundary forwarding agent locator (NimLoc)     :
+---------------+---------------+---------------+---------------+
```

LOC-UPDATE ::=

```
+---------------+---------------+---------------+---------------+
:            Originator Credentials (NimCred)                   :
+---------------+---------------+---------------+---------------+
:                     Old NR EID    (NimEID)                    :
+---------------+---------------+---------------+---------------+
:                     Old NR locator (NimLoc)                   :
+---------------+---------------+---------------+---------------+
:                     Old locator    (NimLoc)                   :
+---------------+---------------+---------------+---------------+
:                     New locator    (NimLoc)                   :
+---------------+---------------+---------------+---------------+
:                     Valid until    (NimSecs)                  :
+---------------+---------------+---------------+---------------+

  Defined Flag (in UPD-HDR) values :

               1 = Depreciate use of old locator
               2 = Terminate use of old locator.
```

MAP-UPDATE ::=

```
+---------------+---------------+---------------+---------------+
:                     Abstract Map   (NimMap)                   :
```

```
+---------------+---------------+---------------+---------------+
:                    Agent specific map 1 (NimMap)              :
+---------------+---------------+---------------+---------------+
:                    Agent specific map 2 (NimMap)              :
+---------------+---------------+---------------+---------------+
:                         .......                               :
+---------------+---------------+---------------+---------------+
:                    Agent specific map n (NimMap)              :
+---------------+---------------+---------------+---------------+

   Defined Flag (in UPD-HDR) values :

               1 = One or more component nodes not in map
               2 = Map is of a partitioned node.
```

9.7 Query/Response Operation Messages

For most databases, we have the request, release, and response messages (for adjacencies, we additionally have the activation message). We only give the formats for the request messages for each database. The other message formats are similar and can be easily reconstructed from the contents description given earlier in section 6.4.

```
ADJ-REQ ::=

        +---------------+---------------+---------------+---------------+
        :              Requesting Node Locator  (NimLoc)                :
        +---------------+---------------+---------------+---------------+
        :              Requesting Node Identifier (NimNID)              :
        +---------------+---------------+---------------+---------------+
        :              Requesting Node Circuit ID (NimLkID)             :
        +---------------+---------------+---------------+---------------+
        :              Neighbor Node ID            (NimNID)             :
        +---------------+---------------+---------------+---------------+


              Defined Flag (in Q-R-HDR) values :

                        1 = Adjacency from parent to child
                        2 = Other adjacencies
                        4 = Adjacency from child to parent
                        8 = Adjacency to siblings



NODE-LOC-REQ ::=
```

```
+---------------+---------------+---------------+---------------+
|               Old NR  EID   (NimEID)                          :
+---------------+---------------+---------------+---------------+
:               Old NR locator (NimLoc)                         :
+---------------+---------------+---------------+---------------+
:               Previously assigned loc (NimLoc)                :
+---------------+---------------+---------------+---------------+
:               Parent NID      (NimNID)                        :
+---------------+---------------+---------------+---------------+
:               Child NID       (NimNID)                        :
+---------------+---------------+---------------+---------------+
```

```
EP-LOC-REQ ::=

+---------------+---------------+---------------+---------------+
|               Old NR  EID   (NimEID)                          :
+---------------+---------------+---------------+---------------+
:               Old NR locator (NimLoc)                         :
+---------------+---------------+---------------+---------------+
:               Previously assigned loc (NimLoc)                :
+---------------+---------------+---------------+---------------+
:               NID to provided loc  (NimNID)                   :
+---------------+---------------+---------------+---------------+
:               Endpoint name       (NimFQDN)                   :
+---------------+---------------+---------------+---------------+
:               Endpoint ID         (NimEID)                    :
+---------------+---------------+---------------+---------------+
```

```
MAP-REQ  ::=

+---------------+---------------+---------------+---------------+
|               Node loc of desired map  (NimLoc)               :
+---------------+---------------+---------------+---------------+
:               Child elements if partial (NimLElem)            :
+---------------+---------------+---------------+---------------+
:               Credentials of requesting node (NimCred)        :
+---------------+---------------+---------------+---------------+

  Defined Flag (in Q-R-HDR) values :

             0x8000  = Response must be authoritative
             0x4000  = Abbreviated map desired
```

```
                    0x2000  = Complete maps desired
                    0x1000  = Automatic updates desired
```

ROUT-REQ ::=

```
+---------------+---------------+---------------+---------------+
|               Sources        (NimLEpt)                       :
+---------------+---------------+---------------+---------------+
:               Destinations   (NimLEpt)                       :
+---------------+---------------+---------------+---------------+
:               Initiating EPs service reqs (NimServ)          :
+---------------+---------------+---------------+---------------+
:               Target EPs service reqs (NimServ)              :
+---------------+---------------+---------------+---------------+
:               Child NID       (NimNID)                       :
+---------------+---------------+---------------+---------------

    Defined Flag (in Q-R-Hdr) values :

              0x10 = Want maximally disjoint routes
```

9.8 Discovery Message Header

Neighbor discovery messages and agent discovery messages contain much of the same information. This information is represented in a single header pictured below. The authentication field covers the header and any enclosed information.

DISC-HDR ::=

```
+---------------+---------------+-------------------------------+
| Type=Nbr/Agt  |  Agent Type   |       0       |    Flags      |
+---------------+---------------+-------------------------------+
|                  Message Generation Timestamp                 |
+---------------------------------------------------------------+
:                        Authentication                         :
+---------------------------------------------------------------+
|                        Agent's EID                            |
|                                                               |
+---------------------------------------------------------------+
|                    NID of Agent's Node                        |
|                                                               |
+---------------------------------------------------------------+
:                      Agent's Locator                          :
```

```
+--------------------------------------------------------------+
:                       Sequence of NIDs                        :
:                  of nodes containing agent                    :
:                    formatted as a locator                     :
+--------------------------------------------------------------+
```

The neighbor discovery message includes only the discovery header, while the agent discovery message contains the EID and services to each neighbor. Furthermore, if the advertising agent is a boundary forwarding agent, the advertisement also contains the neighbor's NID, locator, sequence of NIDs in which the neighbor is contained, and an indication of whether the agent participates in an adjacency (A = 1) with the neighbor and if so which types: outbound (T = 1), inbound (T = 2), or both (T = 3).

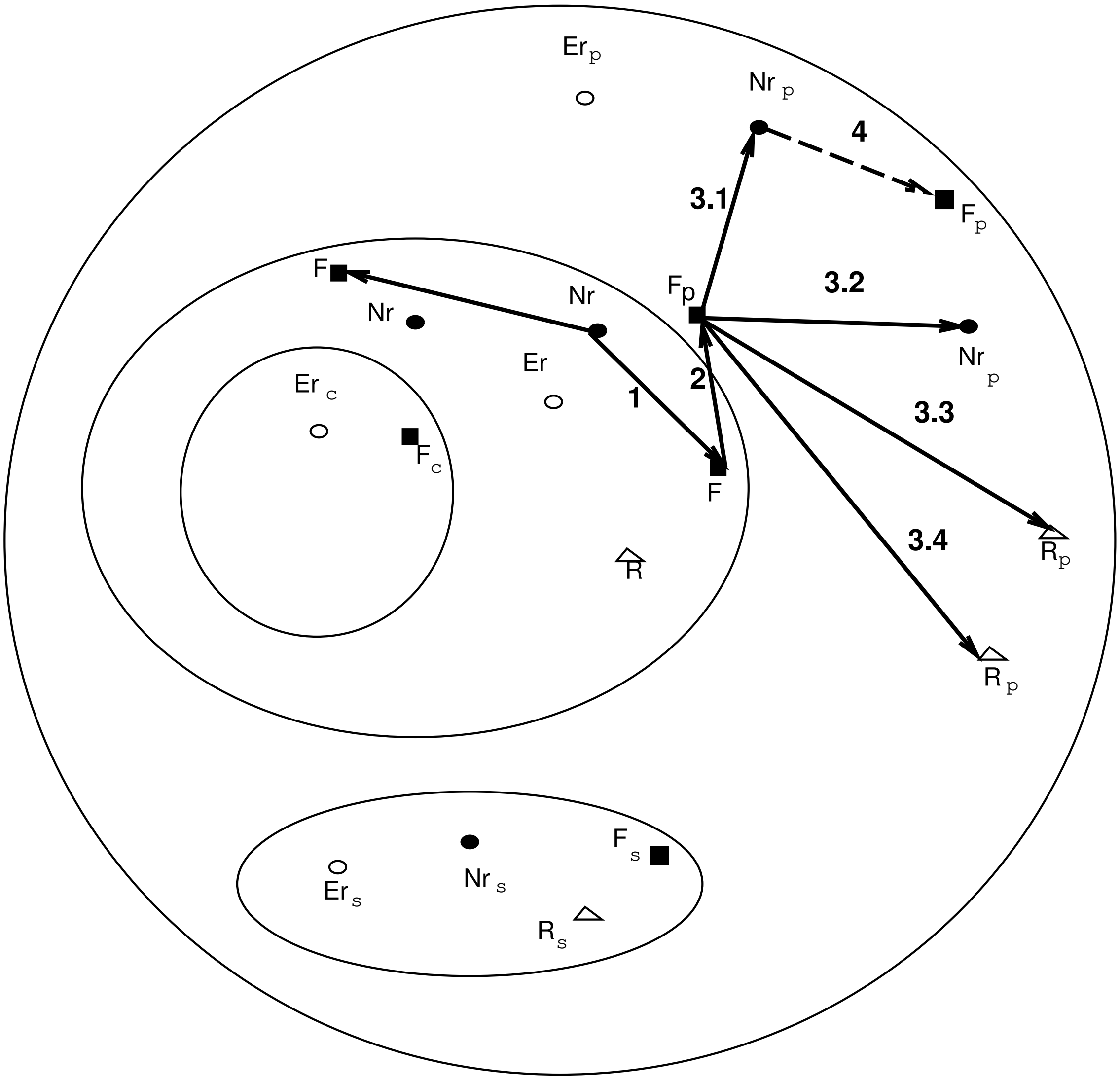
AGT-ADV ::=

```
+--------------------------------------------------------------+
|                      EID of Neighbor n                        |
|                                                               |
+--------------------------------------------------------------+
:                   Services to Neighbor n                      :
+--------------------------------------------------------------+
|                   NID of Neighbor n's Node                    |
|                                                               |
+--------------------------------------------------------------+
:                 Locator of Neighbor n's Node                  :
+--------------------------------------------------------------+
:                       Sequence of NIDs                        :
:                 of nodes containing neighbor n                :
:                    formatted as a locator                     :
+--------------------------------------------------------------+
|A|T |       type of adjacency with neighbor n's node          |
+--------------------------------------------------------------+
:                             ...                               :
+--------------------------------------------------------------+
|                      EID of Neighbor 0                        |
|                                                               |
+--------------------------------------------------------------+
:                   Services to Neighbor 0                      :
+--------------------------------------------------------------+
|                   NID of Neighbor 0's Node                    |
|                                                               |
+--------------------------------------------------------------+
:                 Locator of Neighbor 0's Node                  :
+--------------------------------------------------------------+
:                       Sequence of NIDs                        :
:                 of nodes containing neighbor 0                :
```

```
:                     formatted as a locator                       :
+-----------------------------------------------------------------+
|A|T |        type of adjacency with neighbor 0's node            |
+-----------------------------------------------------------------+
```

10 Appendix 1: Figures for Update and Q-R protocols

Figure 1 : Map Update



Key :

Agents :
Er = endpoint rep.
Nr = node rep.
F = Forwarding agent
R = Route Agent

Subscripts :
p = parent
c = child
s = sibling

Figure 2 : Locator Request (Phase 1 : Acquisition)

Arrows 1−4 denote node locator acquisition, 5−6 denote endpoint locator acquistion.

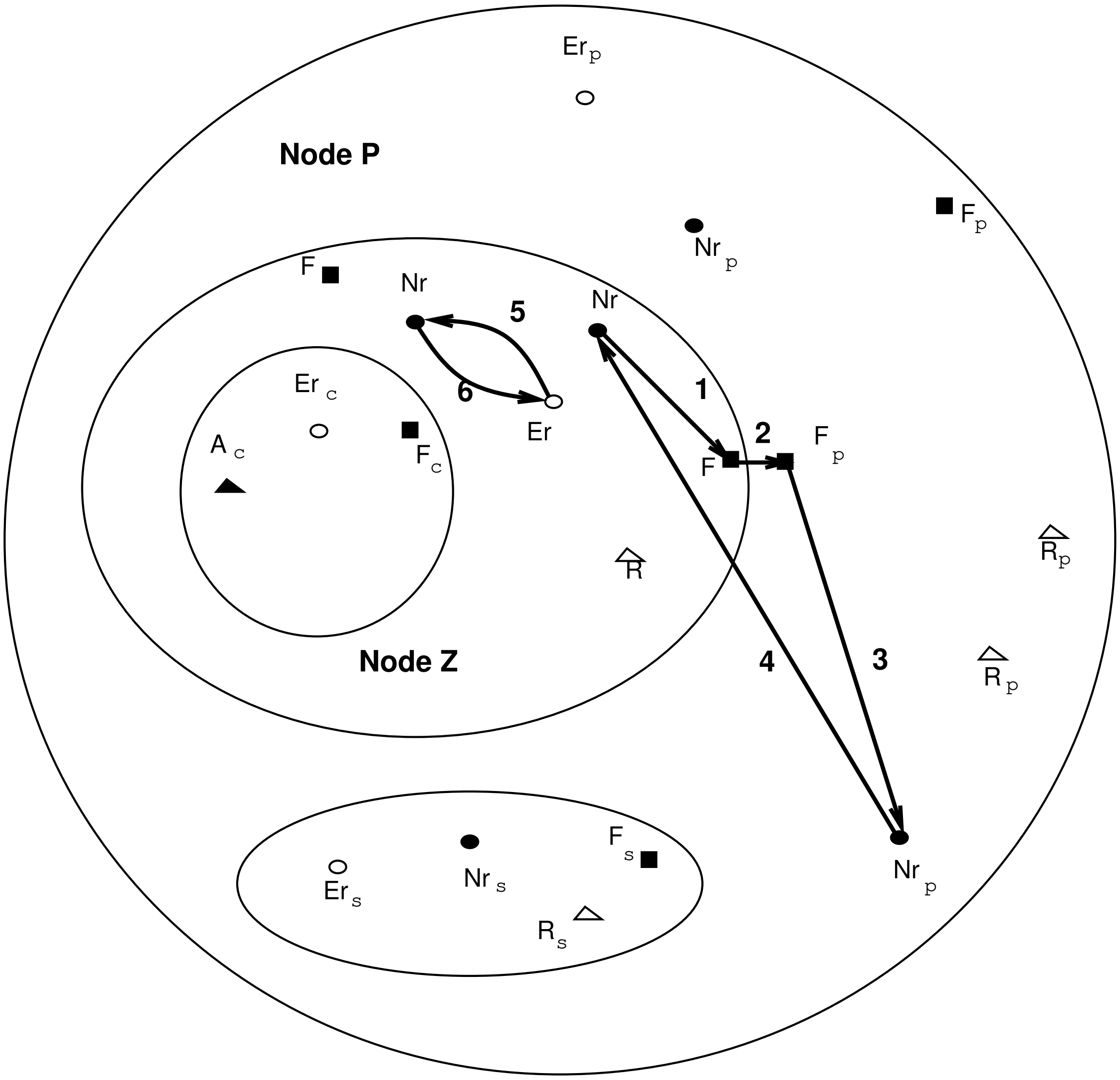


Figure 3 : Locator Request (Phase 2 : Notification)

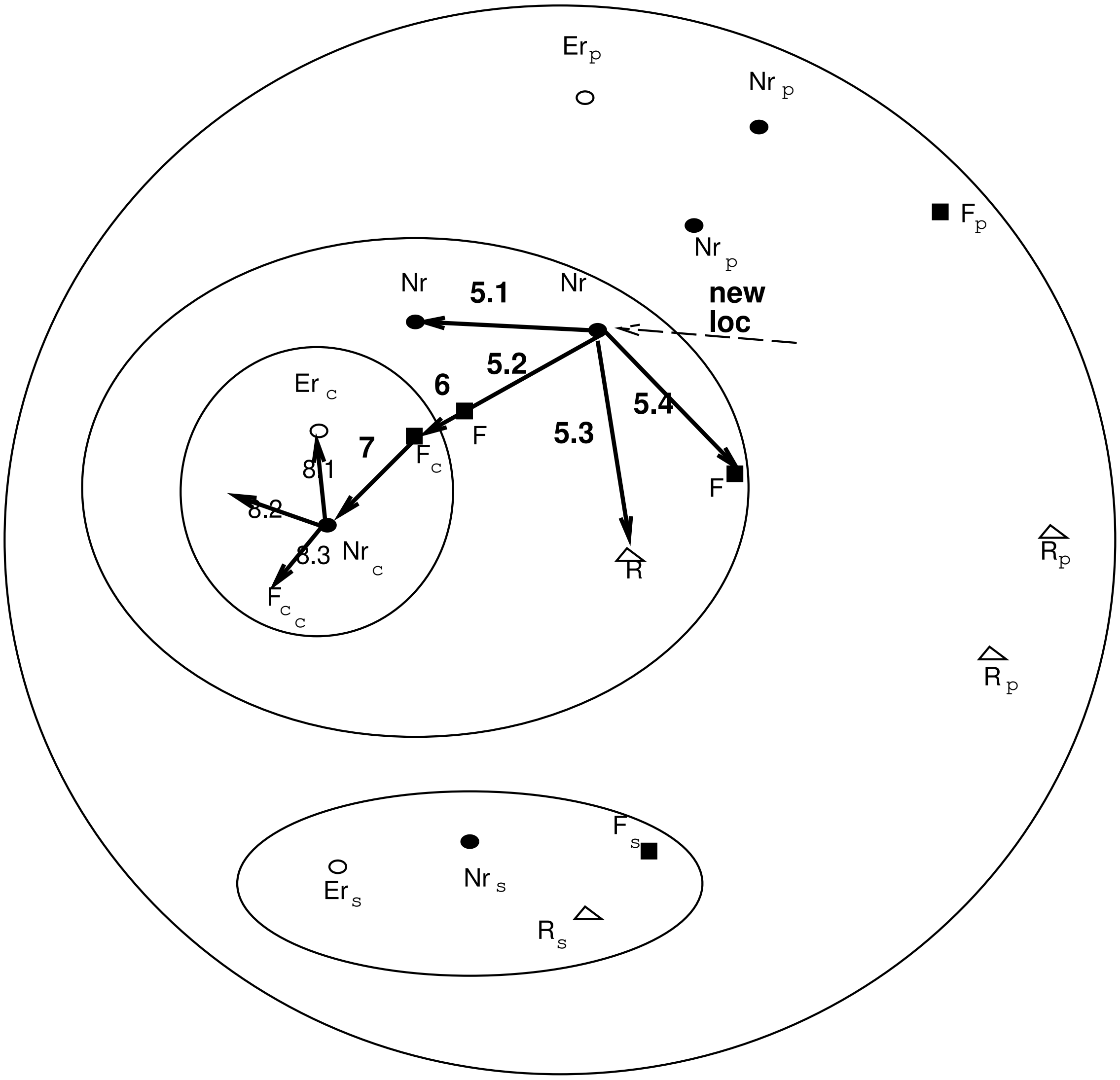
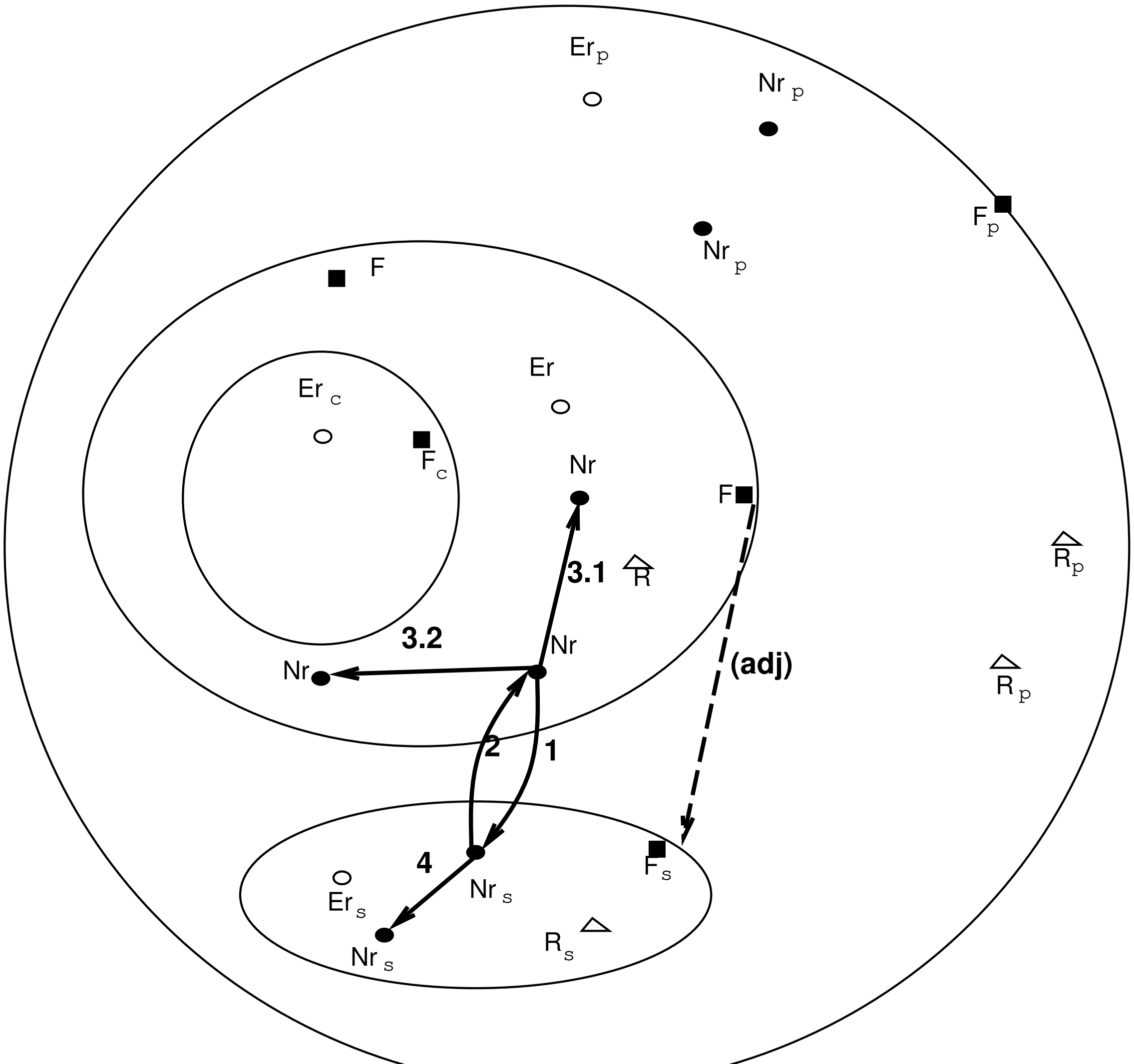


Figure 4 : Adjacency Formation



11 Appendix 2: Basic Data Formats

In section 9, a number of packet formats were pictured in terms of other, more basic, variable length data objects such as locator (NimLoc), and EID (NimEID). Here we give the formats for such basic data elements. Only the more basic and important data objects are described in detail and illustrated pictorially - namely Element (NimElem), Locator (NimLoc), EID (NimEID), Endpoint Name (NimFQDN), Node ID (NimNID). The other data objects are given in the form of C structure declarations.

11.1 Element (NimElem)

An Element consists of a variable number of octets, with the most frequently used length expected to be two octets. The most significant bit (M) is one (1) for a multicast element and zero (0) otherwise. The next few bits encodes the element length. The remaining bits may be asigned by node representatives. Different instances of a node representative might choose to use different assignment algorithms. The only constraint is that the set of node representatives for a node should cooperate to make sure that an element is not given out by more than one of the representatives.

Several formats have been predefined. One-octet elements are expected to be used to identify routers or router interfaces of small sized nodes that choose to include router identification in their locators, e.g., use Nimrod routing as their intra-node routing protocol. The format of a one-octet element is:

```
+-+-+-+-+-+-+-+-+
|M|0|           |                       1 octet, 6 free bits
+-+-+-+-+-+-+-+-+
```

Two-octet elements are expected to be used in most cases. An organization might choose to set aside some bits in an element that could be used, e.g., to identify successive numbering plans, so that internal management and monitoring might be simplified. The format of a two-octet element is:

```
+-+-+-+-+-+-+-+-+-+...+-+
|M|1 0 0|               |               2 octets, 12 free bits
+-+-+-+-+-+-+-+-+-+...+-+
```

Four-octet elements are provided for nodes that, e.g., want to include local information in an element. The format of a four-octet element is:

```
+-+-+-+-+-+-+-+-+-+...+-+
|M|1 1 1 0 0 0 0|       |               4 octets, 24 free bits
+-+-+-+-+-+-+-+-+-+...+-+
```

Nimrod uses a two-octet element to specify which agents within a node should receive a control message. The format of this element is:

```
+-+-+-+-+-+-+-+-+-+...+-+-+
|M|1 1 1 0 0 0 1| Agent Set |           2 octets, Nimrod Agent Set
+-+-+-+-+-+-+-+-+-+...+-+-+
```

Seven-octet elements are used within the Nimrod system to encode EIDs and NIDs in a locator element. These elements are used during bootstrapping to enable intra-node (including to a parent or component node) communication before globally routable locator (element)s have been acquired. The format of this element is:

```
                +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                |M|1 1 0|Rel|          EID / NID bits         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The relationship field (Rel) is used for node-relative addressing. Its values are:

```
    00      This node
    01      Parent node
    10      Child node
    11      Reserved
```

11.2 Locator (NimLoc)

A locator is a variable length object that represents a location in an internet. It consists of a header containing a flag bit, bits that identify the object as a locator and bits encoding the total length, one or more self delimiting locator elements, and optional octets of zero padding to make the overall length of the locator in octets zero mod 4. An element occupies an integral number of octets.

The bits identifying a locator are 0000 011. This is derived from the experimental IPv6 address space.

Each locator begins on a 32-bit boundary, and is padded with zero octets to maintain 32-bit overall alignment.

```
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+---------------+
        |0 0 0 0 0 1 1|Z|F|1 1 0 0|CdLen|   <element>   |               |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+---------------+
        |                 <element>                   <padding>   |
        +---------------+---------------+---------------+---------------+
```

The Z field must be zero and is reserved for future expansion.

The F field is a context-dependent flag bit.

The length, in octets, of a locator is indicated by the CdLen field as follows:

CdLen	0	1	2	3	4	5	6	7
Octets	4	8	12	16	20	24	32	Reserved

11.3 Endpoint Identifier (NimEID)

Nimrod uses Endpoint Identifiers (EIDs), globally unique, hierarchy independent, relatively fixed and short length, administratively assigned binary representations of endpoint names. Initially, EIDs will be eight-octet quantities, with fiftey bits available for assignment. The format of the EID is (HELP???) :

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|F|0 1 0 0|0 0 1|   octet-1      |  octet-2      |    octet-3    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+---------------+
|  octet-4      | octet-5       |  octet-6      |    octet-7    |
+---------------+---------------+---------------+---------------+
```

A first cut at an EID allocation policy is as follows, where "r" is the two-bit relationship code (00 = this node, 01 = parent node, 10 = child node, 11 = reserved).

```
6r0 00 00 00 00 00 00                           "Wild" Node/Endpoint
6r0 00 00 00 00 00 01                           "This" Node/Endpoint
6r0 00 00 00 00 00 02 thru 6r0 FF FF FF FF FF FF   48-bit administered
6r1 00 00 00 00 00 00 thru 6r1 FF FF FF FF FF FF   48-bit reserved
6r2 00 00 00 00 00 00 thru 6r2 FF FE|FF FF FF FF   Reserved
6r2 FF FF|00 00 00 00 thru 6r2 FF FF|FF FF FF FF   Embedded IPv4
6r3 00 00 00 00 00 00 thru 6r3 FF FF FF FF FF FF   Embedded IEEE 802
```

Note that NIDs and EIDs are taken from the same number space. EIDs have the "0x21" prefix and NIDs the "0x29" prefix.

11.4 Endpoint Name (NimFQDN)

Endpoint names have the form of a fully qualified domain name (FQDN) and are represented by the NimFQDN object. Nimrod uses the user-friendly host name, e.g., www.nimrod.BBN.Com form.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|F|0 0 1 1 1 1 0 0 0 0 0 1 1 0|       obj len in octets         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+---------------+
:                     DNS  Name                                  :
+---------------+---------------+---------------+---------------+
:                     DNS  Name                : pad (if nec.) :
+---------------+---------------+---------------+---------------+
```

11.5 Node Identifier (NimNID)

Nimrod uses a Node Identifier to identify a node in a topologically independent manner. Node identifiers are administratively assigned and are used to identify a node before the Nimrod bootstrapping process and the associated locator acquisition and update messages have determined the node's global locator. An NID may be thought of as an EID of a node (they come from the same number space). A node's policies are tagged by the node's NID. Each interface of a forwarding agent that provides connectivity across a node boundary has an associated sequence of NIDs that identify the nodes whose policies, both incoming and outgoing, must be enforced on that interface. Such an interface is termed a boundary forwarding agent.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|F|0 1 0 1|0 0 1|   octet-1     |   octet-2     |    octet-3    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+---------------+
|  octet-4      |  octet-5      |   octet-6     |    octet-7    |
+---------------+---------------+---------------+---------------+
```

11.6 Services (NimServ)

Nimrod advertises, requests, and offers communication services through lists of communication parameters. Each parameter has its own object type code, length, and value.

```
struct NimServ {
        NimOL2          idlen;          /* Object type code and length */
        NimMgmt         mgmt;           /* Management authority */
        NimAuth         auth;           /* Authentication of authority */
        NimLSrv         orig_req;       /* Opt: Service requested by orig*/
        NimLSrv         dest_req;       /* Opt: Service requested by dest*/
        NimLSrv         offered;        /* Opt: Service offered*/
}

struct NimLSrv {                        /* Object type code and length */
        NimOL2          idlen;          /* List of Service Parameter */
        u_int32_t       skd_id;         /* Schedule id */
        NimSrvP         values[];       /* Set of parameters and values */
}


struct NimSrvP {                        /* Service Parameter */
        NimOL2          idlen;          /* Object type code and length */
        NIM_SRV_*       0x              /* Service parameter */
        u_int32_t       value[];        /* Opt: Value of parameter */
}
```

Service parameters include but are not limited to delay, throughput, probability of packet error, MTU, monetary cost of service, time at which the service is offered, organizations which may use the service, and characterization of traffic necessary to obtain the service. The availability of services is frequently restricted to certain time intervals, called schedules, especially when fees are being charged for the services.

```
struct NimSked {
       NimOL2          idlen;          /* Object type code and length */
       u_int32_t       skd_id;         /* Schedule id, within context ... */
       NimMgmt         mgmt;           /* ... of a management authority */
       NimAuth         auth;           /* Authentication of authorizing entity*/
       NimSkd1         sets[];         /* Opt: time intervals */
}                                      /* Omitted means anytime */

struct NimSkd1 {
       NimOL2          idlen;          /* Object type code and length */
       u_int16_t       flags;          /* Periodic events */
       int16_t         tzone;          /* Minutes from UTC */
       NimSecs         earliest,       /* Bounding time, or 0 ... */
                       latest;         /* ... or all 1's */
       u_int16_t       intervals[2*i]; /* Opt: Start and stop pairs, minutes
                                               from midnight */
    }
```

11.7 Maps (NimMap)

```
struct NimMap {
       NimOL2          idlen;          /* Object type code and length */

       u_int8_t        kind;           /* Kind of map */
       u_int24_t       seq;            /* 24 LSB of generation NTP seconds */
                                       /* Rollsover in 6 mo, longer than */
                                       /* signature key lifetime */
       NimLoc          locs[];         /* Locator(s) of node represntd */
                                       /* Makes component and connspec unique */
       NimNID          nid;            /* NID of node */
       NimAgnt         agent;          /* NR that produced the map */
       NimSecs         expires;        /* When map expires, advisory */
       NimLLoc         adj_in,         /* Adjacent incoming nodes */
                       adj_out;        /* Adjacent outgoing nodes */

       NimAuth         abv_auth;      /* Authenticator over abbreviated map */

/* Next 4 are only in Full map, either all present, or all omitted */
       NimLCsp         csp_in,         /* Opt1: Set of incoming conn specs */
                       csp_out,        /* Opt1: Set of outgoing conn specs */
                       csp_trnst;      /* Opt1: Set of transit conn specs */
       NimAuth         full_auth;       /*Opt1: Authenticator over full map */
                                       /*       (incl abv_auth) */
/* Next 5 are only in Basic map: */
/* appended child maps, NimMap len excludes child maps */
```

```
        NimAuth         basic_auth;     /* Opt2:Authenticator over this map */
                                        /*    and appended child maps */
        u_int32_t       num_maps;       /* Opt2: Number of appended maps */
                                        /*       of child nodes */
        NimLElm         chilren;        /* Opt2: Elements of component nodes */
|
                                        /* Context is "locs", above */
        NimLLoc         adj_child;      /* Opt2: Adjacent children nodes */
        /* Service may be omitted if it appears (for same cspec_id) above */
        NimLCsp         topology;       /* Opt2: Interconnectivity of node */
}
```

11.8 Routes (NimRute)

A route is used to identify, at the level of Nimrod nodes or forwarding agents, the communication components through which traffic from one endpoint to another endpoint should pass, where each endpoint is identified by its locator. It includes the type of route, such as unicast, multicast, sharable, and so on. Each hop in a route identifies the next node or forwarding agent through which traffic should be routed and the connectivity specification that should be used for that portion of the route.

```
struct NimRute {
        NimOL2          idlen;          /* Object type code and length */
        u_int32_t       flags;          /* multicast, unicast, etc. */
        NimAuth         auth;            /* Authentication for generating
                                           route agent */
        NimServ         serv_req;       /* Services required */
        NimDRst         drst;           /* Opt: Restrictions on reuse of route */
        NimServ         offered;        /* Services associated with route */
        NimRutH         hops[];         /* List of hops in route */
}

struct NimRutH {
        NimOL2          idlen;          /* Object type code and length */
        u_int32_t       cspec_id;       /* Connectivity Specification Id */
        NimLoc          node;           /* Next Node to pass through */
}
```

11.9 Credentials (NimCred)

```
struct NimCred {
        NimOL2          idlen;          /* Object type code and length */
        NimFQDN         name;           /* Who's credentials */
        u_int32_t       cred[];         /* Body of credentials */
}
```

11.10 Path Labels (NimPLbl)

```
struct NimPLbl {
        u_int32_t       flag:1,         /* Setup flag */
                        obj_id:7,       /* Object type code */
                        label:24;       /* Path label */
        NimEID          source;         /* Opt: source EID, top level only */
}
```

11.11 Time (NimSecs, NimNTP)

```
struct NimSecs {
        NimOL2          idlen;          /* Object type code and length */
        u_int32_t       seconds;        /* Seconds */
}

struct NimNTP {
        NimOL2          idlen;          /* Object type code and length */
        u_int32_t       seconds;        /* NTP Seconds */
        u_int32_t       fraction;       /* NTP Fractional Seconds */
}
```

11.12 Authenticator (NimAuth)

```
struct NimAuth {
        NimOL2          idlen;          /* Object type code and length */
        u_int16_t       alg_id,         /* Authentication algorithm id */
                        key_ftpt;       /* Footprint of key, for key rollover */
        NimFQDN         signer;         /* Opt: Identify of signer */
        u_int8_t        auth[16];       /* Authenticator, 16 for MD5 */
}
```

The key footprint field specifies the least significant two bytes of the key used to form the authenticator; see [13]. It is used to identify which of the possible keys was used to form the authenticator. Multiple keys may be valid when keys are being changed. Having the footprint is an optimization that makes it possible to pick the right key without having to try each one until one works, or all possible keys fail.

12 Security Considerations

Security issues for Nimrod are discussed in general in sections 2 and 3 and more specifically in the sections for the specific protocols and packet formats (section 9).

13 Contact Information

Ram Ramanathan
BBN Systems and Technologies
10 Moulton St.
Cambridge, MA 02138
Ph: (617) 873-2736
email: ramanath@bbn.com

Martha Steenstrup
BBN Systems and Technologies
10 Moulton St.
Cambridge, MA 02138
Ph: (617) 873-3192
email: msteenst@bbn.com

References

[1] I. Castineyra, J.N. Chiappa and M. Steenstrup, "The Nimrod Architecture", Working Draft, March 1996.

[2] M. Steenstrup, "A Perspective on Nimrod Functionality", Working Draft, March 1996.

[3] S. Ramanathan, "Mobility Support for Nimrod: Requirements and Solution Approaches", Working Draft, March 1996.

[4] S. Ramanathan, "Multicast Support for Nimrod: Requirements and Solution Approaches", Working Draft, March 1996.

[5] C. Lynn, "Nimrod Deployment", Working Draft, March 1995.

[6] D.L. Mills, "Network Time Protocol (Version 3) Specification, Implementation and Analysis", *Internet RFC*, Number 1305, March 1992.

[7] J. Postel, "Internet Protocol", *Internet RFC*, Number 791, September 1981.

[8] J. Postel, "Transmission Control Protocol", *Internet RFC*, Number 793, September 1981.

[9] R. Braden, "Extending TCP for Transactions – Concepts", *Internet RFC*, Number 1379, November 1992.

[10] R. Braden, "T/TCP – TCP Extensions for Transactions Functional Specification", *Internet RFC*, Number 1664, July 1994.

[11] R. Atkinson, "IP Authentication Header", *Internet RFC*, Number 1826, August 1995.

[12] R. Atkinson, "IP Encapsulating Security Payload (ESP)", *Internet RFC*, Number 1827, August 1995.

[13] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", *Internet RFC*, Number 1883, January 1996.