IETF MANET Working Group — Josh Broch
INTERNET-DRAFT — David B. Johnson
David A. Maltz
Carnegie Mellon University
25 June 1999

The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks

<draft-ietf-manet-dsr-02.txt>

Status of This Memo

Abstract

Dynamic Source Routing (DSR) is a routing protocol designed specifically for use in mobile ad hoc networks. The protocol allows nodes to dynamically discover a source route across multiple network hops to any destination in the ad hoc network. When using source routing, each packet to be routed carries in its header the complete, ordered list of nodes through which the packet must pass. A key advantage of source routing is that intermediate hops do not need to maintain routing information in order to route the packets they receive, since the packets themselves already contain all of the necessary routing information. This, coupled with the dynamic, on-demand nature of DSR's Route Discovery, completely eliminates the need for periodic router advertisements and link status packets, significantly reducing the overhead of DSR, especially during periods when the network topology is stable and these packets serve only as keep-alives.

Contents

1. Introduction

This document describes Dynamic Source Routing (DSR) [7, 8], a protocol developed by the Monarch Project [9, 16] at Carnegie Mellon University for routing packets in a mobile ad hoc network [4].

Source routing is a routing technique in which the sender of a packet determines the complete sequence of nodes through which to forward the packet; the sender explicitly lists this route in the packet's header, identifying each forwarding "hop" by the address of the next node to which to transmit the packet on its way to the destination node.

DSR offers a number of potential advantages over other routing protocols for mobile ad hoc networks. First, DSR uses no periodic routing messages of any kind (e.g., no router advertisements and no link-level neighbor status messages), thereby significantly reducing network bandwidth overhead, conserving battery power, reducing the probability of packet collision, and avoiding the propagation of potentially large routing updates throughout the ad hoc network. Our Dynamic Source Routing protocol is able to adapt quickly to changes such as node movement, yet requires no routing protocol overhead during periods in which no such changes occur.

In addition, DSR has been designed to compute correct routes in the presence of asymmetric (uni-directional) links. In wireless networks, links may at times operate asymmetrically due to sources of interference, differing radio or antenna capabilities, or the intentional use of asymmetric communication technology such as satellites. Due to the existence of asymmetric links, traditional link-state or distance vector protocols may compute routes that do not work. DSR, however, will always find a correct route even in the presence of asymmetric links.

2. Assumptions

We assume that all nodes wishing to communicate with other nodes within the ad hoc network are willing to participate fully in the protocols of the network. In particular, each node participating in the network should also be willing to forward packets for other nodes in the network.

We refer to the minimum number of hops necessary for a packet to reach from any node located at one extreme edge of the network to another node located at the opposite extreme, as the diameter of the network. We assume that the diameter of an ad hoc network will be small (e.g., perhaps 5 or 10 hops), but may often be greater than 1.

Packets may be lost or corrupted in transmission on the wireless network. A node receiving a corrupted packet can detect the error and discard the packet.

We assume that nodes can enable promiscuous receive mode on their wireless network interface hardware, causing the hardware to deliver every received packet to the network driver software without filtering based on link-layer destination address. Although we do not require this facility, it is for example common in current LAN hardware for broadcast media including wireless, and some of our optimizations take advantage of its availability. Use of promiscuous mode does increase the software overhead on the CPU, but we believe that wireless network speeds are more the inherent limiting factor to performance in current and future systems. We also believe that portions of the protocol are also suitable for implementation directly within a programmable network interface unit to avoid this overhead on the CPU.

3. Terminology

3.1. General Terms

link

A communication facility or medium over which nodes can communicate at the link layer, such as an Ethernet (simple or bridged). A link is the layer immediately below IP.

interface

A node's attachment to a link.

prefix

A bit string that consists of some number of initial bits of an address.

interface index

An 7-bit quantity which uniquely identifies an interface among a given node's interfaces. Each node can assign interface indices to its interfaces using any scheme it wishes.

The index IF_INDEX_MA is reserved for use by Mobile IP [11] mobility agents (home or foreign agents) to indicate that they believe they can reach a destination via a connected internet infrastructure. The index IF_INDEX_ROUTER is reserved for use by routers not acting as Mobile IP mobility agents to indicate that they believe they can reach the destination via a connected internet infrastructure.

The distinction between the index for mobility agents and the index for routers, allows mobility agents to advertise their existence ``for free''. A node that processes a routing header listing the interface index IF_INDEX_MA, can then send a unicast Agent Solicitation to the corresponding address in the routing header to obtain complete information about the mobility services being provided.

link-layer address

A link-layer identifier for an interface, such as IEEE 802 addresses on Ethernet links.

packet

An IP header plus payload.

piggybacking

Including two or more conceptually different types of data in the same packet so that all data elements move through the network together.

home address

An IP address that is assigned for an extended period of time to a mobile node. It remains unchanged regardless of where the node is attached to the Internet [11]. If a node has more than one home address, it SHOULD select and use a single home address when participating in the ad hoc network.

source route

A source route from a node S to some node D is an ordered list of home addresses and interface indexes that contains all the information that would be needed to forward a packet through the ad hoc network. For each node that will transmit the packet, the source route provides the index of the interface over which the packet should be transmitted, and the address of the node which is intended to receive the packet.

DSR Routing Headers as described in Section 6.3 use a more compact encoding of the source route and do not explicitly list address S in the Routing Header`, since it is carried as the IP Source Address of the packet.

A source route is described as ``broken'' when the specific path it describes through the network is not actually viable.

Route Discovery

The method in DSR by which a node S dynamically obtains a source route to some node D that will be used by S to route packets through the network to D. Performing a Route Discovery involves sending one or more Route Request packets.

Route Maintenance

The process in DSR of monitoring the status of a source route while in use, so that any link-failures along the source route can be detected and the broken link removed from use.

3.2. Specification Language

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [2].

4. Protocol Overview

4.1. Route Discovery and Route Maintenance

A source routing protocol must solve two challenges, which DSR terms Route Discovery and Route Maintenance. Route Discovery is the mechanism whereby a node S wishing to send a packet to a destination D obtains a source route to D.

Route Maintenance is the mechanism whereby S is able to detect, while using a source route to D, if the network topology has changed such that it can no longer use its route to D because a link along the route no longer works. When Route Maintenance indicates a source route is broken, S can attempt to use any other route it happens to know to D, or can invoke Route Discovery again to find a new route.

To perform Route Discovery, the source node S link-layer broadcasts a Route Request packet. Here, node S is termed the initiator of the Route Discovery, and the node to which S is attempting to discover a source route, say D, is termed the target of the Discovery.

Each node that hears the Route Request packet forwards a copy of the Request, if appropriate, by adding its own address to a source route being recorded in the Request packet and then rebroadcasting the Route Request.

The forwarding of Route Requests is constructed so that copies of the Request propagate hop-by-hop outward from the node initiating the Route Discovery, until either the target of the Request is found or until another node is found that can supply a route to the target.

The basic mechanism of forwarding Route Requests forwards the Request if the node (1) is not the target of the Request, (2) is not already listed in the recorded source route in this copy of the Request, and (3) has not recently seen another Route Request packet belonging to this same Route Discovery. A node can determine if it has recently seen such a Route Request, since each Route Request packet contains a unique identifier for this Route Discovery, generated by the initiator of the Discovery. Each node maintains an LRU cache of the unique identifier from each recently received Route Request. By not propagating any copies of a Request after the first, the overhead of forwarding additional copies that reach this node along different paths is avoided.

In addition, the Time-to-Live field in the IP header of the packet carrying the Route Request MAY be used to limit the scope over which the Request will propagate, using the normal behavior of Time-to-Live defined by IP [14, 1]. Additional optimizations on the handling and forwarding of Route Requests are also used to further reduce the

Route Discovery overhead.

When the target of the Request (e.g., node D) receives the Route Request, the recorded source route in the Request identifies the sequence of hops over which this copy of the Request reached D. Node D copies this recorded source route into a Route Reply packet and sends this Route Reply back to the initiator of the Route Request (e.g., node S).

All source routes learned by a node are kept in a Route Cache, which is used to further reduce the cost of Route Discovery. When a node wishes to send a packet, it examines its own Route Cache and performs Route Discovery only if no suitable source route is found in its Cache.

Further, when some intermediate node B receives a Route Request from S for some target node D, B not equal D, B searches its own Route Cache for a route to D. If B finds such a route, it might not have to propagate the Route Request, but instead return a Route Reply to node S based on the concatenation of the recorded source route from S to B in the Route Request and the cached route from B to D. The details of replying from a Route Cache in this way are discussed in Section 8.1.

As a node overhears routes being used by others, either on data packets or on control packets used by Route Discovery or Route Maintenance, the node MAY insert those routes into its Route Cache, leveraging the Route Discovery operations of the other nodes in the network. Such route information MAY be learned either by promiscuously snooping on packets or when forwarding packets.

4.2. Packet Forwarding

To represent a source route within a packet's header, DSR uses a Routing Header similar to the Routing Header format specified for IPv6, adapted to the needs of DSR and to the use of DSR in IPv4 (or in IPv6 in the future). The DSR Routing Header uses a unique Routing Type field value to distinguish it from the existing Type 0 Routing Header defined within IPv6 [5].

To forward a packet, a receiving node N simply processes the Routing Header as specified in Section 7.3 and transmits the packet to the next hop. If a forwarding error occurs along the link to the next hop in the route, this node N sends a Route Error back to the originator S of this packet informing S that this link is "broken". If node N's Route Cache contains a different route to the destination of the original packet, then the packet is salvaged using the new source route (Section 7.5.5). Otherwise, the packet is dropped.

Each node overhearing or forwarding a Route Error packet also removes from its Route Cache the link indicated to be broken, thereby cleaning the stale cache data from the network.

4.3. Multicast Routing

At this time DSR does not support true multicasting. However, it does support the controlled flooding of a data packet to all nodes in the network that are within some number of hops of the originator. While this mechanism does not support pruning of the broadcast tree to conserve network resources, it can be used to distribute information to nodes in the network.

When an application on a DSR node sends a packet to a multicast address, DSR piggybacks the data from the packet inside a Route Request packet targeted at the multicast address. The normal Route Request distribution scheme described in Sections 4.1 and 7.4.2 will result in this packet being efficiently distributed to all nodes in the network within the specified TTL of the originator. The receiving nodes can then do destination address filtering on the packet, discarding it if they do not wish to receive multicast packets destined to this multicast address.

5. Conceptual Data Structures

In order to participate in the Dynamic Source Routing Protocol, a node needs four conceptual data structures: a Route Cache, a Route Request Table, a Send Buffer, and a Retransmission Buffer. These data structures MAY be implemented in any manner consistent with the external behavior described in this document.

5.1. Route Cache

All routing information needed by a node participating in an ad hoc network using DSR is stored in a Route Cache. Each node in the network maintains its own Route Cache. The node adds information to the Cache as it learns of new links between nodes in the ad hoc network, for example through packets carrying either a Route Reply or a Routing Header. Likewise, the node removes information from the cache as it learns existing links in the ad hoc network have broken, for example through packets carrying a Route Error or through the link-layer retransmission mechanism reporting a failure in forwarding a packet to its next-hop destination. The Route Cache is indexed logically by destination node address, and supports the following operations:

void Insert(Route RT)

Inserts information extracted from source route RT into the Route Cache.

Route Get(Node DEST)

Returns a source route from this node to DEST (if one is known).

void Delete(Node FROM, Interface INDEX, Node TO)

Removes from the route cache any routes which assume that a packet transmitted by node FROM over its interface with the given INDEX will be received by node TO.

Each implementation MAY choose the cache replacement and cache search strategies for its Route Cache that are most appropriate for its particular network environment. For example, some environments may choose to return the shortest route to a node (the shortest sequence of hops), while others may select an alternate metric for the Get() operation.

The Route Cache SHOULD support storing more than one source route for each destination.

If there are multiple cached routes to a destination, the Route Get() operation SHOULD prefer routes that do not traverse a hop with an interface index of IF_INDEX_MA or IF_INDEX_ROUTER. This will prefer routes that lead directly to the target node over routes that attempt to reach the target via any internet infrastructure connected to the ad hoc network.

If a node S is using a source route to some destination D that includes intermediate node N, S SHOULD shorten the route to destination D when it learns of a shorter route to node N than the one that is listed as the prefix of its current route to D.

A node S using a source route to destination D through intermediate node N, MAY shorten the source route if it learns of a shorter path from node N to node D.

The Route Cache replacement policy SHOULD allow routes to be categorized based upon "preference", where routes with a higher preferences are less likely to be removed from the cache. For example, a node could prefer routes for which it initiated a Route Discovery over routes that it learned as the result of promiscuous snooping on other packets. In particular, a node SHOULD prefer routes that it is presently using over those that it is not.

5.2. Route Request Table

The Route Request Table is a collection of records about Route Request packets that were recently originated or forwarded by this node. The table is indexed by the home address of the target of the route discovery. A record maintained on node S for node D contains the following:

- The time that S last originated a Route Discovery for D.

- The remaining amount of time that S must wait before the next attempt at a Route Discovery for D.

- The Time-to-live (TTL) field in the IP header of last Route Request originated by S for D.

- A FIFO cache of the last ID_FIFO_SIZE Identification values from Route Request packets targeted at node D that were forwarded by this node.

Nodes SHOULD use an LRU policy to manage the entries of in their Route Request Table.

ID_FIFO_SIZE MUST NOT be set to an unlimited value, since, in the worst case, when a node crashes and reboots the first ID_FIFO_SIZE Route Request packets it sends might appear to be duplicates to the other nodes in the network.

5.3. Send Buffer

The Send Buffer of some node is a queue of packets that cannot be transmitted by that node because it does not yet have a source route to each respective packet's destination. Each packet in the Send Buffer is stamped with the time that it is placed into the Buffer, and SHOULD be removed from the Send Buffer and discarded SEND_BUFFER_TIMEOUT seconds after initially being placed in the Buffer. If necessary, a FIFO strategy SHOULD be used to evict packets before they timeout to prevent the buffer from overflowing.

Subject to the rate limiting defined in Section 7.4, a Route Discovery SHOULD be initiated as often as possible for the destination address of any packets residing in the Send Buffer.

5.4. Retransmission Buffer

The Retransmission Buffer of a node is a queue of packets sent by this node that are awaiting the receipt of an acknowledgment from the next hop in the source route (Section 6.3).

For each packet in the Retransmission Buffer, a node maintains (1) a count of the number of retransmissions and (2) the time of the last retransmission.

Packets are removed from the buffer when an acknowledgment is received, or when the number of retransmissions exceeds DSR_MAXRXTSHIFT. In the later case, the removal of the packet from the Retransmission Buffer SHOULD result in a Route Error being returned to the initial source of the packet (Section 7.5).

6. Packet Formats

Dynamic Source Routing makes use of four options carrying control information that can be piggybacked in any existing IP packet.

The mechanism used for these options is based on the design of the Hop-by-Hop and Destination Options mechanisms in IPv6 [5]. The ability to generate and process such options must be added to an IPv4 protocol stack. Specifically, the Protocol field in the IP header is used to indicate that a Hop-by-Hop Options or Destination Options extension header exists between the IP header and the remaining portion of a packet's payload (such as a transport layer header). The Next Header field in each extension header will then indicate the type of header that follows it in a packet.

6.1. Destination Options Headers

The Destination Options header is used to carry optional information that need be examined only by a packet's destination node(s). The Destination Options header is identified by a Next Header (or Protocol) value of 60 in the immediately preceding header, and has the following format:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Next Header  |  Hdr Ext Len  |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
|                                                               |
.                                                               .
.                            Options                            .
.                                                               .
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Next Header

8-bit selector. Identifies the type of header immediately following the Destination Options header. Uses the same values as the IPv4 Protocol field [17].

Hdr Ext Len

8-bit unsigned integer. Length of the Destination Options header in 4-octet units, not including the first 8 octets.

Options

Variable-length field, of length such that the complete Destination Options header is an integer multiple of 4 octets long. Contains one or more TLV-encoded options.

The following destination option is used by the Dynamic Source Routing protocol:

- DSR Route Request option (Section 6.1.1)

This destination option MUST NOT appear multiple times within a single Destination Options header.

6.1.1. DSR Route Request Option

The DSR Route Request destination option is encoded in type-length-value (TLV) format as follows:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Option Type  | Option Length |         Identification        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Target Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|C| IN Index[1] |C| IN Index[2] |C| IN Index[3] |C| IN Index[4] |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|C|OUT Index[1] |C|OUT Index[2] |C|OUT Index[3] |C|OUT Index[4] |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Address[1]                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Address[2]                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Address[3]                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Address[4]                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|C| IN Index[5] |C| IN Index[6] |C|  IN Index[7] |C| IN Index[8]|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|C|OUT Index[5] |C|OUT Index[6] |C| OUT Index[7] |C|OUT Index[8]|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Address[5]                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               ...                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

IP fields:

Source Address

MUST be the home address of the node originating this packet. Intermediate nodes that repropagate the request do not change this field.

Destination Address

MUST be the limited broadcast address (255.255.255.255).

Hop Limit (TTL)

Can be varied from 1 to 255, for example to implement expanding-ring searches.

Route Request fields:

Option Type

???. A node that does not understand this option MUST discard the packet and the Option Data may change en-route (the top three bits are 011).

Option Length

8-bit unsigned integer. Length of the option, in octets, excluding the Option Type and Option Length fields.

Identification

A unique value generated by the initiator (original sender) of the Route Request. This value allows a recipient to determine whether or not it has recently seen this a copy of this Request; if it has, the packet is simply discarded. When propagating a Route Request, this field MUST be copied from the received copy of the Request being forwarded.

Target Address

The home address of the node that is the target of the Route Request.

Change Interface (C) bit[1..n]

A flag associated with each interface index that indicates whether or not the corresponding node repropagated the Request over a different physical interface type than over which it received the Request.

IN Index[1..n]

IN Index[i] is the index of the interface over which the node indicated by Address[i] received the Route Request option. These are used to record a reverse route from the target of the request to the originator, over which a Route Reply MAY be sent.

OUT Index[1..n]

OUT Index[i] is the interface index that the node indicated by Address[i-1] used when rebroadcasting the Route Request option.

Address[1..n]

Address[i] is the home address of the ith hop recorded in the Route Request option.

6.2. Hop-by-Hop Options Headers

The Hop-by-Hop Options header is used to carry optional information that must be examined by every node along a packet's delivery path. The Hop-by-Hop Options header is identified by a Next Header (or Protocol) value of ??? in the IP header, and has the following format:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Next Header  |  Hdr Ext Len  |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
|                                                               |
.                                                               .
.                            Options                            .
.                                                               .
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Next Header

8-bit selector. Identifies the type of header immediately following the Hop-by-Hop Options header. Uses the same values as the IPv4 Protocol field [17].

Hdr Ext Len

8-bit unsigned integer. Length of the Hop-by-Hop Options header in 4-octet units, not including the first 8 octets.

Options

Variable-length field, of length such that the complete Hop-by-Hop Options header is an integer multiple of 4 octets long. Contains one or more TLV-encoded options.

The following hop-by-hop options are used by the Dynamic Source Routing protocol:

- DSR Route Reply option (Section 6.2.1)

- DSR Route Error option (Section 6.2.2)

- DSR Acknowledgment option (Section 6.2.3)

All of these destination options MAY appear one or more times within a single Hop-by-Hop Options header.

6.2.1. DSR Route Reply Option

The DSR Route Reply hop-by-hop option is encoded in type-length-value (TLV) format as follows:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
                +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                |  Option Type  | Option Length |   Reserved    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Target Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|C|OUT Index[1] |C|OUT Index[2] |C|OUT Index[3] |C|OUT Index[4] |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Address[1]                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Address[2]                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Address[3]                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Address[4]                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|C|OUT Index[5] |C|OUT Index[6] |C|OUT Index[7] |C|OUT Index[8] |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Address[5]                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               ...                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Option Type

???. A node that does not understand this option should ignore this option and continue processing the packet, and the Option Data does not change en-route (the top three bits are 000).

Option Length

8-bit unsigned integer. Length of the option, in octets, excluding the Option Type and Option Length fields.

Reserved

Sent as 0; ignored on reception.

Target Address

The home address of the node to which the Route Reply must be delivered.

Change Interface (C) bit[1..n]

If the C bit associated with a node N is set, it implies N will be forwarding the packet out a different interface than the one over which it was received (i.e., the node sending the packet to N should not expect a passive acknowledgment).

OUT Index[1..n]

OUT Index[i] is the interface index of the ith hop listed in the Route Reply option. It denotes the interface that should be used by Address[i-1] to reach Address[i] when using the specified source route.

Address[1..n]

Address[i] is the home address of the ith hop listed in the Route Reply option.

6.2.2. DSR Route Error Option

The DSR Route Error hop-by-hop option is encoded in type-length-value (TLV) format as follows:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
                +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                |  Option Type  | Option Length |     Index     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Error Source Address                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Error Destination Address                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Unreachable Node Address                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Option Type

???. A node that does not understand this option should ignore the option and continue processing the packet, and the Option Data does not change en-route (the top three bits are 000).

Option Length

8-bit unsigned integer. Length of the option, in octets, excluding the Option Type and Option Length fields.

Index

The interface index of the network interface over which the node designated by Error Source Address tried to forward a packet to the node designated by Unreachable Node Address.

Error Source Address

The home address of the node originating the Route Error (e.g., the node that attempted to forward a packet and discovered the link failure).

Error Destination Address

The home address of the node to which the Route Error must be delivered (e.g, the node that generated the routing information claiming that the hop Error Source Address to Unreachable Node Address was a valid hop).

Unreachable Node Address

The home address of the node that was found to be unreachable (the next hop neighbor to which the node at ``Error Source Address'' was attempting to transmit the packet).

6.2.3. DSR Acknowledgment Option

The DSR Acknowledgment hop-by-hop option is encoded in type-length-value (TLV) format as follows:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
                                +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                                |  Option Type  | Option Length |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Identification                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       ACK Source Address                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     ACK Destination Address                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Data Source Address                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Option Type

???. A node that does not understand this option should ignore the option and continue processing the packet, and the Option Data does not change en-route (the top three bits are 000).

Option Length

8-bit unsigned integer. Length of the option, in octets, excluding the Option Type and Option Length fields.

Identification

A 32-bit value that when taken in conjunction with Data Source Address, uniquely identifies the packet being acknowledged.

The Identification value is computed as ((ip_id << 16) | ip_off) where ip_id is the value of the 16-bit Identification field in the IP header of the packet being acknowledged, and ip_off is the value of the 13-bit Fragment Offset field in the IP header of the packet being acknowledged.

When constructing the Identification, ip_id and ip_off MUST be in host byte-order. The entire Identification value MUST then

be converted to network byte-order before being placed in the Acknowledgment option.

ACK Source Address

The home address of the node originating the Acknowledgment.

ACK Destination Address

The home address of the node to which the Acknowledgment must be delivered.

Data Source Address

The IP Source Address of the packet being acknowledged.

6.3. DSR Routing Header

As specified for IPv6 [5], a Routing header is used by a source to list one or more intermediate nodes to be ``visited'' on the way to a packet's destination. This function is similar to IPv4's Loose Source and Record Route option, but the Routing header does not record the route taken as the packet is forwarded. The specific processing steps required to implement the Routing header must be added to an IPv4 protocol stack. The Routing header is identified by a Next Header value of 43 in the immediately preceding header, and has the following format:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Next Header  |  Hdr Ext Len  |  Routing Type | Segments Left |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
.                                                               .
.                       type-specific data                      .
.                                                               .
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The type specific data for a Routing Header carrying a DSR Source Route is:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|R|S|                        Reserved                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|C|OUT Index[1] |C|OUT Index[2] |C|OUT Index[3] |C|OUT Index[4] |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Address[1]                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Address[2]                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Address[3]                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Address[4]                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|C|OUT Index[5] |C|OUT Index[6] |C|OUT Index[7] |C|OUT Index[8] |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Address[5]                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               ...                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Routing Header Fields:

Next Header

8-bit selector. Identifies the type of header immediately following the Routing header.

Hdr Ext Len

8-bit unsigned integer. Length of the Routing header in 4-octet units, not including the first 8 octets.

Routing Type

???

Segments Left

Number of route segments remaining, i.e., number of explicitly listed intermediate nodes still to be visited before reaching the final destination.

Type Specific Fields:

Acknowledgment Request (R)

The Acknowledgment Request (R) bit is set to request an explicit acknowledgment from the next hop. After processing the Routing Header, The IP Destination Address lists the address of the next hop.

Salvaged Packet (S)

The Salvaged Packet (S) bit indicates that this packet has been salvaged by an intermediate node, and thus that this Routing Header was generated by Address[1] and not the IP Source Address (Section 7.5.5).

Reserved

Sent as 0; ignored on reception.

Change Interface (C) bit[1..n]

If the C bit associated with a node N is set, it implies N will be forwarding the packet out a different interface than the one over which it was received (i.e., the node sending the packet to N should not expect a passive acknowledgment and MAY wish to set the R bit).

OUT Index[1..n]

Index[i] is the interface index that the node indicated by Address[i-1] must use when transmitting the packet to Address[i]. Index[1] indicates which interface the node indicated by the IP Source Address uses to transmit the packet.

Address[1..n]

Address[i] is the home address of the ith hop in the Routing header.

Note that Address[1] is the first intermediate hop along the route. The address of the originating node is the IP Source Address. The only exception to this rule is for packets that are salvaged, as described in Section 7.5.5. A packet that has been salvaged has an alternate route placed on it by an intermediate node in the network, and in this case, the address of the originating node (the salvaging node) is Address[1]. Salvaged packets are indicated by setting the S bit in the DSR Routing header.

7. Detailed Operation

7.1. Originating a Data Packet

When node A originates a packet, the following steps MUST be taken before transmitting the packet:

1. If the destination address is a multicast address, piggyback the data packet on a Route Request targeting the multicast address. The following fields MUST be initialized as specified:

```
IP.Source_Address      = Home address of node A
IP.Destination_Address = 255.255.255.255
Request.Target_Address = Multicast destination address
```

   DONE.

2. Otherwise, call Route_Cache.Get() to determine if there is a cached source route to the destination.

3. If the cached route indicates that the destination is directly reachable over one hop, no Routing Header should be added to the packet. Initialize the following fields:

```
IP.Source_Address      = Home address of node A
IP.Destination_Address = Home address of the Destination
```

   DONE.

4. Otherwise, if the cached route indicates that multiple hops are required to reach the destination, insert a Routing Header into the packet as described in Section 7.2. DONE.

5. Otherwise, if no cached route to the destination is found, insert the packet into the Send Buffer and initiate Route Discovery as described in Section 7.4.

7.2. Originating a Packet with a DSR Routing Header

When a node originates a packet with a Routing Header, the address of the first hop in the source route MUST be listed as the IP Destination Address as well as Address[1] in the Routing Header. The final destination of the packet is listed as the last hop in the Routing Header (Address[n]). At each intermediate hop i, Address[i] is copied into the IP Destination Address and the packet is retransmitted.

For example, suppose node A originates a packet destined for node D that should pass through intermediate hops B and C. The packet MUST be initialized as follows:

```
IP.Source_Address      = Home address of node A
IP.Destination_Address = Home address of node B
RT.Segments_Left       = 2
RT.Out_Index[1]        = Interface index used by A to reach B
RT.Out_Index[2]        = Interface index used by B to reach C
RT.Out_Index[3]        = Interface index used by C to reach D
RT.Address[1]          = Home address of node B
RT.Address[2]          = Home address of node C
RT.Address[3]          = Home address of node D
```

7.3. Processing a Routing Header

Excluding the exceptions listed here, a DSR Routing Header is processed using the same rules as outlined for Type 0 Routing Headers in IPv6 [5]. The Routing Header is only processed by the node whose address appears as the IP destination of the packet. The following additional rules apply to processing the type specific data of a DSR Source Route:

Let

SegLft = the value of Segments Left when the packet was received
NumAddrs = the total number of addresses in the Routing Header

1. The address of the next hop, Address[NumAddrs - SegLft + 1], is copied into the IP.Destination_Address of the packet. The existing IP.Destination_Address is NOT copied back into the Address list of the Routing Header.

2. The interface used to transmit the packet to its next hop from this node MUST be the interface denoted by Index[NumAddrs - SegLft + 1].

3. If the Acknowledgment Request (R) bit is set, the node MUST transmit a packet containing the DSR Acknowledgment option to the previous hop, Address[NumAddrs - SegLft - 1], performing Route Discovery if necessary. (Address[0] is taken as the IP.Source_Address)

4. Perform Route Maintenance by verifying that the packet was received by the next hop as described in Section 7.5.

7.4. Route Discovery

Route Discovery is the on-demand process by which nodes actively obtain source routes to destinations to which they are actively attempting to send packets. The destination node for which a Route Discovery is initiated is known as the "target" of the Route Discovery. A Route Discovery for a destination SHOULD NOT be initiated unless the initiating node has a packet in the Send Buffer requiring delivery to that destination. A Route Discovery for a given target node MUST NOT be initiated unless permitted by the rate-limiting information contained in the Route Request Table. After each Route Discovery attempt, the interval between successive Route Discoveries for this target must be doubled, up to a maximum of MAX_REQUEST_PERIOD.

Route Discoveries for a multicast address SHOULD NOT be rate limited, and SHOULD always be permitted.

7.4.1. Originating a Route Request

The basic Route Discovery algorithm for a unicast destination is as follows:

1. Originate a Route Request packet with the IP header Time-to-Live field initialized to 1. This type of Route Request is called a non-propagating Route Request and allows the originator of the Request to inexpensively query the route caches of each of its neighbors for a route to the destination.

2. If a Route Reply is received in response to the non-propagating Request, use the returned source route to transmit all packets for the destination that are in the Send Buffer. DONE.

3. Otherwise, if no Route Reply is received within RING0_REQUEST_TIMEOUT seconds, transmit a Route Request with the IP header Time-to-Live field initialized to MAX_ROUTE_LEN. This type of Route Request is called a propagating Route Request. Update the information in the Route Request Table, to double the amount of time before any subsequent Route Discovery attempt to this target.

4. If no Route Reply is received within the time interval indicated by the Route Request Table, GOTO step 1.

The Route Request option SHOULD be initialized as follows:

```
IP.Source_Address      = This node's home address
IP.Destination_Address = 255.255.255.255
Request.Target         = Home address of intended destination
```

Request.OUT_Index[1] = Index of interface used to transmit the Request

The behavior of a node processing a packet containing both a Routing Header and a Route Request Destination option is unspecified. Packets SHOULD NOT contain both a Routing Header and a Route Request Destination option. [This is not exactly true: A Route Request option appearing in the second Destination Options header that IPv6 allows after the Routing Header would probably do-what-you-mean, though we have not triple-checked it yet. Namely, it would allow the originator of a route discovery to unicast the request to some other node, where it would be released and begin the flood fill. We call this a Route Request Blossom since the unicast portion of the path looks like a stem on the blossoming flood-fill of the request.]

Packets containing a Route Request Destination option SHOULD NOT be retransmitted, SHOULD NOT request an explicit DSR Acknowledgment by setting the R bit, SHOULD NOT expect a passive acknowledgment, and SHOULD NOT be placed in the Retransmission Buffer. The repeated transmission of packets containing a Route Request Destination option is controlled solely by the logic described in this section.

7.4.2. Processing a Route Request Option

When a node A receives a packet containing a Route Request option, the Route Request option is processed as follows:

1. If Request.Target_Address matches the home address of this node, then the Route Request option contains a complete source route describing the path from the initiator of the Route Request to this node.

   (a) Send a Route Reply as described in Section 7.4.4.

   (b) Continue processing the packet in accordance with the Next Header value contained in the Destination Option extension header. DONE.

2. Otherwise, if the combination (IP.Source_Address, Request.Identification) is found in the Route Request Table, then discard the packet, since this is a copy of a recently seen Route Request. DONE.

3. Otherwise, if Request.Target_Address is a multicast address then:

   (a) If node A is a member of the multicast group indicated by Request.Target_Address, then create a copy of the packet, setting IP.Destination_Address = REQUEST.Target_Address, and continue processing the copy of the packet in accordance with the Next Header field of the Destination option.

(b) If IP.TTL is non-zero, decrement IP.TTL, and retransmit the packet. DONE.

(c) Otherwise, discard the packet. DONE.

4. Otherwise, if the home address of node A is already listed in the Route Request (IP.Source_Address or Request.Address[]), then discard the packet. DONE.

5. Let

   m = number of addresses currently in the Route Request option
   n = m + 1

6. Otherwise, append the home address of node A to the Route Request option (Request.Address[n]).

7. Set Request.IN_Index[n] = index of interface packet was received on.

8. If a source route to Request.Target_Address is found in our Route Cache and the rules of Section 7.4.3 permit it, return a Cached Route Reply as described in Section 7.4.3. DONE.

9. Otherwise, for each interface on which the node is configured to participate in a DSR ad hoc network:

   (a) Make a copy of the packet containing the Route Request.

   (b) Set Request.OUT_Index[n+1] = index of the interface.

   (c) If the outgoing interface is different from the incoming interface, then set the C bit on both Request.OUT_Index[n+1] and Request.IN_Index[n]

   (d) Link-layer re-broadcast the packet containing the Route Request on the interface jittered by T milliseconds, where T is a uniformly distributed, random number between 0 and BROADCAST_JITTER. DONE.

7.4.3. Generating Route Replies using the Route Cache

A node SHOULD use its Route Cache to avoid propagating a Route Request packet received from another node. In particular, suppose a node receives a Route Request packet for which it is not the target and which it does not discard based on the logic of Section 7.4.2. If the node has a Route Cache entry for the target of the Request, it SHOULD append this cached route to the accumulated route record in the packet and return this route in a Route Reply packet to

the initiator without propagating (re-broadcasting) the Route Request. Thus, for example, if node F in the example network shown in Figure 7.4.3 needs to send a packet to node D, it will initiate a Route Discovery and broadcast a Route Request packet. If this broadcast is received by node A, node A can simply return a Route Reply packet to F containing the complete route to D consisting of the sequence of hops: A, B, C, and D.

Before transmitting a Route Reply packet that was generated using information from its Route Cache, a node MUST verify that:

1. The resulting route contains no loops.

2. The node issuing the Route Reply is listed in the route that it specifies in its Reply. This increases the probability that the route is valid, since the node in question should have received a Route Error if this route stopped working. Additionally, this requirement means that a Route Error traversing the route will pass through the node that issued the Reply based on stale cache data, which is critical for ensuring stale data is removed from caches in a timely manner. Without this requirement, the next Route Discovery initiated by the original requester might also be contaminated by a Route Reply from this node containing the same stale route.

7.4.4. Originating a Route Reply

Let REQPacket denote a packet received by node A that contains a Route Request option which lists node A as the REQPacket.Request.Target_Address. Let REPPacket be a packet transmitted by node A that contains a corresponding Route Reply. The Route Reply option transmitted in response to a Route Request MUST be initialized as follows:

```
B->C->D
+---+     +---+     +---+     +---+
| A |---->| B |---->| C |---->| D |
+---+     +---+     +---+     +---+

+---+
| F |                       +---+
+---+                       | E |
                            +---+
```

Figure 1: An example network where A knows a route to D via B and C.

1. If REQPacket.Request.Address[] does not contain any hops, then node A is only a single hop from the originator of the Route Request. Build a Route Reply packet as follows:

```
REPPacket.IP.Source_Address    = REQPacket.Request.Target_Address
REPPacket.Reply.Target         = REQPacket.IP.Source_Address
REPPacket.Reply.OUT_Index[1]   = REQPacket.Request.OUT_index[1]
REPPacket.Reply.OUT_C_bit[1]   = REQPacket.Request.OUT_C_bit[1]
REPPacket.Reply.Address[1]     = The home address of node A
```

   GOTO step 3.

2. Otherwise, build a Route Reply packet as follows:

```
REPPacket.IP.Source_Address    = The home address of node A
REPPacket.Reply.Target         = REQPacket.IP.Source_Address
REPPacket.Reply.OUT_Index[1..n]=
REQPacket.Request.OUT_index[1..n]
REPPacket.Reply.OUT_C_bit[1..n]=
REQPacket.Request.OUT_C_bit[1..n]
REPPacket.Reply.Address[1..n]  = REQPacket.Request.Address[1..n]
```

3. Send the Route Reply jittered by T milliseconds, where T is a uniformly distributed random number between 0 and BROADCAST_JITTER. DONE.

If sending a Route Reply packet to the originator of the Route Request requires performing a Route Discovery, the Route Reply hop-by-hop option MUST be piggybacked on the packet that contains the Route Request. This prevents a loop wherein the target of the new Route Request (which was itself the originator of the original Route Request) must do another Route Request in order to return its Route Reply.

If sending the Route Reply to the originator of the Route Request does not require performing Route Discovery, a node SHOULD send a unicast Route Reply in response to every Route Request targeted at it.

7.4.5. Processing a Route Reply Option

Upon receipt of a Route Reply, a node should extract the source route (Target_Address, OUT_Index[1]:Address[1], .. OUT_Index[n]:Address[n]) and insert this route into its Route Cache. All the packets in the Send Buffer SHOULD be checked to see whether the information in the Reply allows them to be sent immediately.

7.5. Route Maintenance

Route Maintenance requires that whenever a node transmits a data packet, a Route Reply, or a Route Error, it must verify that the next hop (indicated by the Destination IP Address) correctly receives the packet.

If the sender cannot verify that the next hop received the packet, it MUST decide that its link to the next hop is broken and MUST send a Route Error to the node responsible for generating the Routing Header that contains the broken link (Section 7.5.3).

The following ways may be used to verify that the next hop correctly received a packet:

- The receipt of a passive acknowledgment (Section 7.5.1).

- The receipt of an explicitly requested acknowledgment (Section 7.5.1).

- By the presence of positive feedback from the link layer indicating that the packet was acknowledged by the next hop (Section 7.5.2).

- By the absence of explicit failure notification from the link layer that provides reliable hop-by-hop delivery such as MACAW or 802.11 (Section 7.5.2).

Nodes MUST NOT perform Route Maintenance for packets containing a Route Request option or packets containing only an Acknowledgment option. Sending Acknowledgments for packets containing only an Acknowledgment option would create an infinite loop whereby acknowledgments would be sent for acknowledgments. Acknowledgments should be always sent for packets containing a Routing Header with the R bit set (e.g., packets which contain only an Acknowledgment and a Routing Header for which the last forwarding hop requires an explicit acknowledgment of receipt by the final destination).

7.5.1. Using Network-Layer Acknowledgments

For link layers that do not provide explicit failure notification, the following steps SHOULD be used by a node A to perform Route Maintenance.

When receiving a packet:

- If the packet contains a Routing Header with the R bit set, send an explicit acknowledgment as described in Section 7.3.

- If the packet does not contain a Routing Header, the node MUST transmit a packet containing the DSR Acknowledgment option to the previous hop as indicated by the IP.Source_Address. Since the receiving node is the final destination, there will be no opportunity for the originator to obtain a passive acknowledgment, and the receiving node must infer the originator's request for an explicit acknowledgment.

When sending a packet:

1. Before sending a packet, insert a copy of the packet into the Retransmission Buffer and update the information maintained about this packet in the Retransmission Buffer.

2. If after processing the Routing Header, RH.Segments_Left is equal to 0, then node A MUST set the Acknowledgment Request (R) bit in the Routing Header before transmitting the packet over its final hop.

3. If after processing the Routing Header and copying RH.Address[n] to IP.Destination_Address, node A determines that RH.OUT_C_bit[n+1] is set, then node A MUST set the Acknowledgment Request (R) bit in the Routing Header before transmitting the packet (since the C bit was set during Route Discovery by the node now listed as the IP.Destination_Address to indicate that it will propagate the packet out a different interface, and that node A will not receive a passive acknowledgment).

4. Set the retransmission timer for the packet in the Retransmission Buffer.

5. Transmit the packet.

6. If a passive or explicit acknowledgment is received before the retransmission timer expires, then remove the packet from the Retransmission Buffer and disable the retransmission timer. DONE.

7. Otherwise, when the Retransmission Timer expires, remove the packet from the Retransmission Buffer.

8. If DSR_MAXRXTSHIFT transmissions have been done, then attempt to salvage the packet (Section 7.5.5). Also, generate a Route Error. DONE.

9. GOTO step 1.

7.5.2. Using Link Layer Acknowledgments

If explicit failure notifications are provided by the link layer, then all packets are assumed to be correctly received by the next hop and a Route Error is sent only when a explicit failure notification is made from the link layer.

Nodes receiving a packet without a Routing Header do not need to send an explicit Acknowledgment to the packet's originator, since the link layer will notify the originator if the packet was not received properly.

7.5.3. Originating a Route Error

If the next hop of a packet is found to be unreachable as described in Section 7.5, a Route Error packet (Section 6.2.2) MUST be returned to the node whose cache generated the information used to route the packet.

When a node A generates a Route Error for packet P, it MUST initialize the fields in the Route Error as follows:

```
Error.Source_Address      = Home address of node A
Error.Unreachable_Address = Home address of the unreachable node
```

- If the packet contains a DSR Routing Header and the S bit is NOT set, the packet has been forwarded without the need for salvaging up to this point.

  ```
  Error.Destination_Address = P.IP.Source_Address
  ```

- Otherwise, if the packet contains a DSR Routing Header and the S bit IS set, the packet has been salvaged by an intermediate node, and thus this Routing Header was placed there by the salvaging node.

  ```
  Error.Destination_Address = P.RoutingHeader.Address[1]
  ```

- Otherwise, if the packet does not contain a DSR Routing Header, the packet must have been originated by this node A.

  ```
  Error.Destination_Address = Home address of node A
  ```

Send the packet containing the Route Error to Error.Destination_Address, performing Route Discovery if necessary.

As an optimization, Route Errors that are discovered by the packet's originator (such that Error.Source_Address is equal to Error.Destination_Address) SHOULD be processed internally. Such

processing should invoke all the steps that would be taken if a Route Error option was created, transmitted, received, and processed, but an actual packet containing a Route Error option SHOULD NOT be transmitted.

7.5.4. Processing a Route Error Option

Upon receipt of a Route Error via any mechanism, a node SHOULD remove any route from its Route Cache that uses the hop (Error.Source_Address, Error.Index to Error.Unreachable_Address). This includes all Route Errors overheard, and those processed internally as described in Section 7.5.3.

When the node identified by Error.Destination_Address receives the Route Error, it SHOULD verify that the source route responsible for delivering the Route Error includes the same hops as the working prefix of the original packet's source route (Error.Destination_Address to Error.Source_Address). If any hop listed in the working prefix is not included in the Route Error's source route, then the originator SHOULD forward the Route Error back along the working prefix (Error.Destination_Address to Error.Source_Address) so that each node along the working prefix will remove the invalid route from its Route Cache.

If the node processing a Route Error option discovers its home address is Error.Destination_Address and the packet contains additional Route Error option(s) later on the inside of the Hop by Hop options header, we call the additional Route Errors nested Route Errors. The node MUST deliver the first nested Route Error to Nested_Error.Destination_Address, performing Route Discovery if needed. It does this by removing the Route Error option listing itself as the Error.Destination_Address, finding the first nested Route Error option, and originating the remaining packet to Nested_Error.Destination_Address. This mechanism allows for the proper handling of Route Errors that are discovered while delivering a Route Error.

7.5.5. Salvaging a Packet

When node A attempts to salvage a packet originated at node S and destined for node D, it MUST perform the following steps:

1. Generate and send a Route Error to A as explained in Section 7.5.3.

2. Call Route_Cache.Get() to determine if it has a cached source route to the packet's ultimate destination D (which is the last Address listed in the Routing Header).

3. If node A does not have a cached route for node D, it MUST discard the packet. DONE.

4. Otherwise, let Salvage_Address[1] through Salvage_Address[m] be the sequence of hops returned from the Route Cache. Initialize the following fields in the packet's header:

```
RT.Segments_Left   = m - 2;
RT.S               = 1
RT.Address[1]      = Home address of Node A
RT.Address[2]      = Salvage.Address[1]
...
RT.Address[n]      = Salvage.Address[m]
```

The IP Source Address of the packet MUST remain unchanged. When the Routing Header in the outgoing packet is processed, RT.Address[2], will be copied to the IP Destination Address field.

8. Optimizations

A number of optimizations can be added to the basic operation of Route Discovery and Route Maintenance as described in Sections 7.4 and 7.5 that can reduce the number of overhead packets and improve the average efficiency of the routes used on data packets. This section discusses some of those optimizations.

8.1. Leveraging the Route Cache

The data in a node's Route Cache may be stored in any format, but the active routes in its cache form a tree of routes, rooted at this node, to other nodes in the ad hoc network. For example, the illustration below shows an ad hoc network of six mobile nodes, in which mobile node A has earlier completed a Route Discovery for mobile node D and has cached a route to D through B and C:

```
B->C->D
+---+     +---+     +---+     +---+
| A |---->| B |---->| C |---->| D |
+---+     +---+     +---+     +---+

+---+
| F |                     +---+
+---+                     | E |
                          +---+
```

Since nodes B and C are on the route to D, node A also learns the route to both of these nodes from its Route Discovery for D. If A later performs a Route Discovery and learns the route to E through B and C, it can represent this in its Route Cache with the addition of the single new hop from C to E. If A then learns it can reach C in a single hop (without needing to go through B), A SHOULD use this new route to C to also shorten the routes to D and E in its Route Cache.

8.1.1. Promiscuous Learning of Source Routes

A node can add entries to its Route Cache any time it learns a new route. In particular, when a node forwards a data packet as an intermediate hop on the route in that packet, the forwarding node is able to observe the entire route in the packet. Thus, for example, when any intermediate node B forwards packets from A to D, B SHOULD add the source route information from that packet's Routing Header to its own Route Cache. If a node forwards a Route Reply packet, it SHOULD also add the source route information from the route record being returned in the Route Reply, to its own Route Cache.

In addition, since all wireless network transmissions at the physical layer are inherently broadcast, it may be possible for a node to configure its network interface into promiscuous receive mode, such that the node is able to receive all packets without link layer address filtering. In this case, the node MAY add to its Route Cache the route information from any packet it can overhear.

8.2. Preventing Route Reply Storms

The ability for nodes to reply to a Route Request not targeted at them by using their Route Caches can result in a Route Reply storm. If a node broadcasts a Route Request for a node that its neighbors have in their Route Caches, each neighbor may attempt to send a Route Reply, thereby wasting bandwidth and increasing the rate of collisions in the area. For example, in the network shown in Section 8.1, if both node A and node B receive F's Route Request, they will both attempt to reply from their Route Caches. Both will send their Replies at about the same time since they receive the broadcast at about the same time. Particularly when more than the two mobile nodes in this example are involved, these simultaneous replies from the mobile nodes receiving the broadcast may create packet collisions among some or all of these replies and may cause local congestion in the wireless network. In addition, it will often be the case that the different replies will indicate routes of different lengths. For example, A's Route Reply will indicate a route to D that is one hop longer than that in B's reply.

For interfaces which can promiscuously listen to the channel, mobile nodes SHOULD use the following algorithm to reduce the number of simultaneous replies by slightly delaying their Route Reply:

1. Pick a delay period

   $$d = H * (h - 1 + r)$$

   where h is the length in number of network hops for the route to be returned in this node's Route Reply, r is a random number between 0 and 1, and H is a small constant delay to be introduced per hop.

2. Delay transmitting the Route Reply from this node for a period of d.

3. Within the delay period, promiscuously receive all packets at this node. If a packet is received by this node during the delay period that is addressed to the target of this Route Discovery (the target is the final destination address for the packet, through any sequence of intermediate hops), and if the length of the route on this packet is less than h, then cancel the delay

timer and do not transmit the Route Reply from this node; this node may infer that the initiator of this Route Discovery has already received a Route Reply, giving an equally good or better route.

8.3. Piggybacking on Route Discoveries

As described in Section 4.1, when one node needs to send a packet to another, if the sender does not have a route cached to the destination node, it must initiate a Route Discovery, buffering the original packet until the Route Reply is returned. The delay for Route Discovery and the total number of packets transmitted can be reduced by allowing data to be piggybacked on Route Request packets. Since some Route Requests may be propagated widely within the ad hoc network, though, the amount of data piggybacked must be limited. We currently use piggybacking when sending a Route Reply or a Route Error packet, since both are naturally small in size. Small data packets such as the initial SYN packet opening a TCP connection [15] could easily be piggybacked.

One problem, however, arises when piggybacking on Route Request packets. If a Route Request is received by a node that replies to the request based on its Route Cache without propagating the Request (Section 8.1), the piggybacked data will be lost if the node simply discards the Route Request. In this case, before discarding the packet, the node must construct a new packet containing the piggybacked data from the Route Request packet. The source route in this packet MUST be constructed to appear as if the new packet had been sent by the initiator of the Route Discovery and had been forwarded normally to this node. Hence, the first portion of the route is taken from the accumulated route record in the Route Request packet and the remainder of the route is taken from this node's Route Cache. The sender address in the packet MUST also be set to the initiator of the Route Discovery. Since the replying node will be unable to correctly recompute an Authentication header for the split off piggybacked data, data covered by an Authentication header SHOULD NOT be piggybacked on Route Request packets.

8.4. Discovering Shorter Routes

Once a route between a packet source and a destination has been discovered, the basic DSR protocol MAY continue to use that route for all traffic from the source to the destination as long as it continues to work, even if the nodes move such that a shorter route becomes possible. In many cases, the basic Route Maintenance procedure will discover the shorter route, since if a node moves enough to create a shorter route, it will likely also move out of transmission range of at least one hop on the existing route.

Furthermore, when a data packet is received as the result of operating in promiscuous receive mode, the node checks if the Routing Header packet contains its address in the unprocessed portion of the source route (Address[NumAddrs - SegLft] to Address[NumAddrs]). If so, the node knows that packet could bypass the unprocessed hops preceding it in the source route. The node then sends what is called a gratuitous Route Reply message to the packet's source, giving it the shorter route without these hops.

The following algorithm describes how a node A should process packets with an IP.Destination_Address not addressed to A or the IP broadcast address or a multicast address that are received as a result of A being in promiscuous receive mode:

1. If the packet is not a data packet containing a Routing Header, drop the packet. DONE.

2. If the home address of this node does not appear in the portion of the source route that has not yet been processed (indicated by Segments Left), then drop the packet. DONE.

3. Otherwise, the node B that just transmitted the packet (indicated by Address[NumAddrs - SegLft - 1]) can communicate directly with this node A. Create a Route Reply. The Route Reply MUST list the entire source route contained in the received packet with the exception of the intermediate nodes between node B and node A.

4. Send this gratuitous Route Reply to the node listed as the IP.Source_Address of the received packet. If Route Discovery is required it MAY be initiated, or the gratuitous Route Reply packet MAY be dropped.

8.5. Rate Limiting the Route Discovery Process

One common error condition that must be handled in an ad hoc network is the case in which the network effectively becomes partitioned. That is, two nodes that wish to communicate are not within transmission range of each other, and there are not enough other mobile nodes between them to form a sequence of hops through which they can forward packets. If a new Route Discovery was initiated for each packet sent by a node in this situation, a large number of unproductive Route Request packets would be propagated throughout the subset of the ad hoc network reachable from this node. In order to reduce the overhead from such Route Discoveries, we use exponential back-off to limit the rate at which new Route Discoveries may be initiated from any node for the same target. If the node attempts to send additional data packets to this same node more frequently than this limit, the subsequent packets SHOULD be buffered in the Send Buffer until a Route Reply is received, but it MUST NOT initiate a

new Route Discovery until the minimum allowable interval between new Route Discoveries for this target has been reached. This limitation on the maximum rate of Route Discoveries for the same target is similar to the mechanism required by Internet nodes to limit the rate at which ARP requests are sent to any single IP address [1].

8.6. Improved Handling of Route Errors

All nodes SHOULD process all of the Route Error messages they receive, regardless of whether the node is the destination of the Route Error, is forwarding the Route Error, or promiscuously overhears the Route Error.

Since a Route Error packet names both ends of the hop that is no longer valid, any of the nodes receiving the error packet may update their Route Caches to reflect the fact that the two nodes indicated in the packet can no longer directly communicate. A node receiving a Route Error packet simply searches its Route Cache for any routes using this hop. For each such route found, the route is effectively truncated at this hop. All nodes on the route before this hop are still reachable on this route, but subsequent nodes are not.

An experimental optimization to improve the handling of errors is to support the caching of "negative" information in a node's Route Cache. The goal of negative information is to record that a given route was tried and found not to work, so that if the same route is discovered again shortly after the failure, the Route Cache can ignore or downgrade the metric of the failed route.

We have not currently included this caching of negative information in our simulations, since it appears to be unnecessary if nodes also promiscuously receive Route Error packets.

8.7. Increasing Scalability

We recently designed and began experimenting with ways to integrate ad hoc networks with the Internet and with Mobile IP [11]. In addition to this, we are also exploring ways to increase the scalablity of ad hoc networks by taking advantage of their cooperative nature and the fact that some hierarchy can be imposed on an ad hoc network, just be assigning addresses to the nodes in a reasonable way. These ideas are described in a workshop paper [3].

9. Constants

```
   BROADCAST_JITTER                           10   milliseconds

   MAX_ROUTE_LEN                              15   nodes

   Interface Indexes
       IF_INDEX_INVALID                     0x7F
       IF_INDEX_MA                          0x7E
       IF_INDEX_ROUTER                      0x7D

   Route Cache
       ROUTE_CACHE_TIMEOUT                   300   seconds

   Send Buffer
       SEND_BUFFER_TIMEOUT                    30   seconds

   Request Table
       MAX_REQUEST_ENTRIES                    32   nodes
       MAX_REQUEST_IDS                         8   identifiers
       MAX_REQUEST_REXMT                      16   retransmissions
       MAX_REQUEST_PERIOD                     10   seconds
       REQUEST_PERIOD                        500   milliseconds
       RING0_REQUEST_TIMEOUT                  30   milliseconds

   Retransmission Buffer
       DSR_RXMT_BUFFER_SIZE                   50   packets

   Retransmission Timer
       DSR_MAXRXTSHIFT                         2
```

10. IANA Considerations

This document proposes the use of the Destination Options header and the Hop-by-Hop Options header, originally defined for IPv6, in IPv4. The Next Header values indicating these two extension headers thus must be reserved within the IPv4 Protocol number space.

Furthermore, this document defines four new types of destination options, each of which must be assigned an Option Type value:

- The DSR Route Request option, described in Section 6.1.1

- The DSR Route Reply option, described in Section 6.2.1

- The DSR Route Error option, described in Section 6.2.2

- The DSR Acknowledgment option, described in Section 6.2.3

DSR also requires a routing header Routing Type be allocated for the DSR Source Route defined in Section 6.3.

In IPv4, we require two new protocol numbers be issued to identify the next header as either an IPv6-style destination option, or an IPv6-style routing header. Other protocols can make use of these protocol numbers as nodes that support them will processes any included destination options or routing headers according to the normal IPv6 semantics.

11. Security Considerations

This document does not specifically address security concerns. This document does assume that all nodes participating in the DSR protocol do so in good faith and with out malicious intent to corrupt the routing ability of the network. In mission-oriented environments where all the nodes participating in the DSR protocol share a common goal that motivates their participation in the protocol, the communications between the nodes can be encrypted at the physical channel or link layer to prevent attack by outsiders.

Location of DSR Functions in the ISO Reference Model

When designing DSR, we had to determine at what level within the protocol hierarchy to implement source routing. We considered two different options: routing at the link layer (ISO layer 2) and routing at the network layer (ISO layer 3). Originally, we opted to route at the link layer for the following reasons:

- Pragmatically, running the DSR protocol at the link layer maximizes the number of mobile nodes that can participate in ad hoc networks. For example, the protocol can route equally well between IPv4 [14], IPv6 [5], and IPX [6] nodes.

- Historically, DSR grew from our contemplation of a multi-hop ARP protocol [7, 8] and source routing bridges [12]. ARP [13] is a layer 2 protocol.

- Technically, we designed DSR to be simple enough that that it could be implemented directly in network interface cards, well below the layer 3 software within a mobile node. We see great potential for DSR running between clouds of mobile nodes around fixed base stations. DSR would act to transparently fill in the coverage gaps between base stations. Mobile nodes that would otherwise be unable to communicate with the base station due to factors such as distance, fading, or local interference sources could then reach the base station through their peers.

Ultimately, however, we decided to specify DSR as a layer 3 protocol since this is the only layer at which we could realistically support nodes with multiple interfaces of different types.

Implementation Status

We have implemented Dynamic Source Routing (DSR) under the FreeBSD 2.2.7 operating system running on Intel x86 platforms. FreeBSD is based on a variety of free software, including 4.4 BSD Lite from the University of California, Berkeley.

During the 7 months from August 1998 to February 1999, we designed and implemented a full-scale physical testbed to enable the evaluation of ad hoc network performance in the field. The last week of February and the first week of March included demonstrations of this testbed to a number of our sponsors and partners, including Lucent Technologies, Bell Atlantic, and DARPA. A complete description of the testbed is available as a Technical Report [10].

Acknowledgments

The protocol described in this draft has been designed within the CMU Monarch Project, a research project at Carnegie Mellon University which is developing adaptive networking protocols and protocol interfaces to allow truly seamless wireless and mobile node networking [9, 16]. The current members of the CMU Monarch Project include:

- Josh Broch
- Yih-Chun Hu
- Jorjeta Jetcheva
- David B. Johnson
- Qifa Ke
- David A. Maltz

References

[1] Robert Braden, editor. Requirements for Internet Hosts -- Communication Layers. RFC 1122, October 1989.

[2] Scott Bradner. Key words for use in RFCs to Indicate Requirement Levels. RFC 2119, March 1997.

[3] Josh Broch, David A. Maltz, and David B. Johnson. Supporting Hierarchy and Heterogeneous Interfaces in Multi-Hop Wireless Ad Hoc Networks. In Proceedings of I-SPAN'99, Perth, Australia, June 1999. To appear.

[4] Scott Corson and Joseph Macker. Mobile Ad Hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations. RFC 2501, January 1999.

[5] S. Deering and R. Hinden. Internet Protocol, version 6 (IPv6) Specification. RFC 2460, December 1998.

[6] IPX Router Specification. Novell Part Number 107-000029-001, Document Version 1.30, March 1996.

[7] David B. Johnson. Routing in Ad Hoc Networks of Mobile Hosts. In Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications, pages 158--163, December 1994.

[8] David B. Johnson and David A. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. In Mobile Computing, edited by Tomasz Imielinski and Hank Korth, chapter 5, pages 153--181. Kluwer Academic Publishers, 1996.

[9] David B. Johnson and David A. Maltz. Protocols for Adaptive Wireless and Mobile Networking. IEEE Personal Communications, 3(1):34--42, February 1996.

[10] David A. Maltz, Josh Broch, and David B. Johnson. Experiences Designing and Building a Multi-Hop Wireless Ad Hoc Network Testbed. Technical Report 99-116, School of Computer Science, Carnegie Mellon University, March 1999.

[11] Charles Perkins, editor. IP Mobility Support. RFC 2002, October 1996.

[12] Radia Perlman. Interconnections: Bridges and Routers. Addison-Wesley, Reading, Massachusetts, 1992.

[13] David C. Plummer. An Ethernet address resolution protocol: Or converting network protocol addresses to 48.bit Ethernet

addresses for transmission on Ethernet hardware. RFC 826, November 1982.

[14] J. Postel. Internet Protocol. RFC 791, September 1981.

[15] J. Postel. Transmission Control Protocol. RFC 793, September 1981.

[16] The CMU Monarch Project. http://www.monarch.cs.cmu.edu/. Computer Science Department, Carnegie Mellon University.

[17] J. Reynolds and J. Postel. Assigned Numbers. RFC 1700, October 1994.

Chair's Address

The Working Group can be contacted via its current chairs:

M. Scott Corson
Institute for Systems Research
University of Maryland
College Park, MD 20742
USA

Phone: +1 301 405-6630
Email: corson@isr.umd.edu

Joseph Macker
Information Technology Division
Naval Research Laboratory
Washington, DC 20375
USA

Phone: +1 202 767-2001
Email: macker@itd.nrl.navy.mil

Authors' Addresses

Questions about this document can also be directed to the authors:

Josh Broch
Carnegie Mellon University
Electrical and Computer Engineering
5000 Forbes Avenue
Pittsburgh, PA 15213-3890
USA

Phone: +1 412 268-3056
Fax: +1 412 268-7196
Email: broch@cs.cmu.edu

David B. Johnson
Carnegie Mellon University
Computer Science Department
5000 Forbes Avenue
Pittsburgh, PA 15213-3891
USA

Phone: +1 412 268-7399
Fax: +1 412 268-5576
Email: dbj@cs.cmu.edu

David A. Maltz
Carnegie Mellon University
Computer Science Department
5000 Forbes Avenue
Pittsburgh, PA 15213-3891
USA

Phone: +1 412 268-3621
Fax: +1 412 268-5576
Email: dmaltz@cs.cmu.edu