Mobile Ad hoc Networks Working Group C. Perkins
Internet-Draft Futurewei
Intended status: Standards Track S. Ratliff
Expires: April 30, 2015 Cisco
J. Dowdell
Airbus Defence and Space
October 27, 2014

Dynamic MANET On-demand (AODVv2) Routing
draft-ietf-manet-aodvv2-05

Abstract

The revised Ad Hoc On-demand Distance Vector (AODVv2) routing protocol is intended for use by mobile routers in wireless, multihop networks. AODVv2 determines unicast routes among AODVv2 routers within the network in an on-demand fashion, offering rapid convergence in dynamic topologies.

Table of Contents

1. Overview

The revised Ad Hoc On-demand Distance Vector (AODVv2) routing protocol [formerly named DYMO] enables on-demand, multihop unicast routing among AODVv2 routers in mobile ad hod networks [MANETs][RFC2501]. The basic operations of the AODVv2 protocol are route discovery and route maintenance. Route discovery is performed when an AODVv2 router must transmit a packet towards a destination for which it does not have a route. Route maintenance is performed to avoid prematurely expunging routes from the route table, and to avoid dropping packets when an active route breaks.

During route discovery, the originating AODVv2 router (RREQ_Gen) multicasts a Route Request message (RREQ) to find a route toward some target destination. Using a hop-by-hop regeneration algorithm, each AODVv2 router receiving the RREQ message records a route toward the originator. When the target's AODVv2 router (RREP_Gen) receives the RREQ, it records a route toward RREQ_Gen and generates a Route Reply (RREP) unicast toward RREQ_Gen. Each AODVv2 router that receives the RREP stores a route toward the target, and again unicasts the RREP toward the originator. When RREQ_Gen receives the RREP, routes have then been established between RREQ_Gen (the originating AODVv2 router) and RREP_Gen (the target's AODVv2 router) in both directions.

Route maintenance consists of two operations. In order to maintain active routes, AODVv2 routers extend route lifetimes upon successfully forwarding a packet. When a data packet is received to be forwarded but there is no valid route for the destination, then the AODVv2 router of the source of the packet is notified via a Route Error (RERR) message. Each upstream router that receives the RERR marks the route as broken. Before such an upstream AODVv2 router could forward a packet to the same destination, it would have to perform route discovery again for that destination. RERR messages are also used to notify upstream routers when routes break (say, due to loss of a link to a neighbor).

AODVv2 uses sequence numbers to assure loop freedom [Perkins99], similarly to AODV. Sequence numbers enable AODVv2 routers to determine the temporal order of AODVv2 route discovery messages, thereby avoiding use of stale routing information. Unlike AODV, AODVv2 uses RFC 5444 message and TLV formats.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

This document uses terminology from [RFC5444].

This document defines the following terms:

Adjacency
   A bi-directional relationship between neighboring AODVv2 routers for the purpose of exchanging routing information. Not every pair of neighboring routers will necessarily form an adjacency. Monitoring of adjacencies where packets are being forwarded is required (see Section 8.2).

AODVv2 Router
   An IP addressable device in the ad-hoc network that performs the AODVv2 protocol operations specified in this document.

AODVv2 Sequence Number (SeqNum)
   Same as Sequence Number.

Client Interface
   An interface that directly connects Router Clients to the Router.

Current_Time
   The current time as maintained by the AODVv2 router.

Disregard
   Ignore for further processing (see Section 5.4).

Handling Router (HandlingRtr)
   HandlingRtr denotes the AODVv2 router receiving and handling an AODVv2 message.

Incoming Link
   A link over which an AODVv2 Router has received a message from an adjacent router.

MANET
   A Mobile Ad Hoc Network as defined in [RFC2501].

Node
   An IP addressable device in the ad-hoc network. A node may be an AODVv2 router, or it may be a device in the network that does not perform any AODVv2 protocol operations. All nodes in this document are either AODVv2 Routers or else Router Clients.

Originating Node (OrigNode)
   The Originating Node is the node that launched the application requiring communication with the Target Node. If OrigNode is a Router Client, its AODVv2 router (RREQ_Gen) has the responsibility to generate a AODVv2 RREQ message on behalf of OrigNode as necessary to discover a route.

Reactive

A protocol operation is said to be "reactive" if it is performed only in reaction to specific events. As used in this document, "reactive" is synonymous with "on-demand".

Routable Unicast IP Address
: A routable unicast IP address is a unicast IP address that is scoped sufficiently to be forwarded by a router. Globally-scoped unicast IP addresses and Unique Local Addresses (ULAs) [RFC4193] are examples of routable unicast IP addresses.

Route Error (RERR)
: A RERR message is used to indicate that an AODVv2 router does not have a route toward one or more particular destinations.

Route Reply (RREP)
: A RREP message is used to establish a route between the Target Node and the Originating Node, at all the AODVv2 routers between them.

Route Request (RREQ)
: An AODVv2 router uses a RREQ message to discover a valid route to a particular destination address, called the Target Node. An AODVv2 router processing a RREQ receives routing information for the Originating Node.

Router Client
: A node that requires the services of an AODVv2 router for route discovery and maintenance. An AODVv2 router is always its own client, so that its list of client IP addresses is never empty.

Router Interface
: An interface supporting the transmission or reception of Router Messages.

RREP Generating Router (RREP_Gen)
: The RREP Generating Router is the AODVv2 router that serves TargNode. RREP_Gen generates the RREP message to advertise a route towards TargNode from OrigNode.

RREQ Generating Router (RREQ_Gen)
: The RREQ Generating Router is the AODVv2 router that serves OrigNode. RREQ_Gen generates the RREQ message to discover a route for TargNode.

Sequence Number (SeqNum)
: Each AODVv2 router MUST maintain an unsigned integer known as the router's "Sequence Number". The Sequence Number guarantees the temporal order of routing information to maintain loop-free routes, and fulfills the same role as the "Destination Sequence Number" of DSDV [Perkins94], and as the AODV Sequence Number in RFC 3561[RFC3561]. The value zero (0) is reserved to indicate that the Sequence Number for an address is unknown.

Target Node (TargNode)
: The Target Node denotes the node towards which a route is needed.

Type-Length-Value structure (TLV)
: A generic way to represent information as specified in [RFC5444].

Unreachable Node (UnreachableNode)

An UnreachableNode is a node for which a valid route is not known.

upstream
: In the direction from TargNode to OrigNode.

Valid route
: A route that can be used for forwarding; in other words a route that is not Broken or Expired.

3. Notational Conventions

This document uses the conventions found in Table 1 to describe information in the fields from [RFC5444].

Notation	Information Location and/or Meaning
Route[Address]	A route table entry towards Address
Route[Address].{field}	A field in a route table entry
--	--
<msg-hop-count>	RFC 5444 Message Header <msg-hop-count>
<msg-hop-limit>	RFC 5444 Message Header <msg-hop-limit>
AddrBlk	an RFC 5444 Address TLV Block
AddrBlk[1]	The first address slot in AddrBlk
AddrBlk[N]	The Nth address slot in AddrBlk
OrigNdx	The index of OrigNode within the AddrBlk
TargNdx	The index of TargNode within the AddrBlk
AddrTLV	an RFC 5444 Address Block TLV
AddrTLV[1]	the first item in AddrTLV
AddrTLV[N]	the Nth item in AddrTLV
Metric_TLV	Metric AddrTLV for AddrBlk
SeqNum_TLV	Sequence Number AddrTLV for AddrBlk
OrigSeqNum_TLV	Originating Node Sequence Number AddrTLV
TargSeqNum_TLV	Target Node Sequence Number AddrTLV
--	--
OrigNode	Originating Node
RREQ_Gen	AODVv2 router originating an RREQ
RREP_Gen	AODVv2 router responding to an RREQ
RteMsg	Either RREQ or RREP
RteMsg.{field}	Field in RREQ or RREP
AdvRte	a route advertised in an incoming RteMsg
HandlingRtr	Handling Router
TargNode	Target Node
UnreachableNode	Unreachable Node

Table 1

4. Applicability Statement

The AODVv2 routing protocol is a reactive routing protocol designed for stub (i.e., non-transit) or disconnected (i.e., from the Internet) mobile ad hoc networks (MANETs). AODVv2 handles a wide variety of mobility patterns by determining routes on-demand. AODVv2 also handles a wide variety of traffic patterns. In networks with a large number of routers, AODVv2 is best suited for relatively sparse traffic scenarios where any particular router forwards packets to only a small percentage of the AODVv2 routers in the network, due to the on-demand nature of route discovery and route maintenance. AODVv2 supports routers with multiple interfaces, as long as each interface has its own (unicast routeable) IP address; the set of all

network interfaces supporting AODVv2 is administratively configured in a list (namely, AODVv2_INTERFACES).

Although AODVv2 is closely related to AODV [RFC3561], and shares some features of DSR [RFC4728], AODVv2 is not interoperable with either of those other two protocols.

AODVv2 is applicable to memory constrained devices, since only a little routing state is maintained in each AODVv2 router. Routes that are not needed for forwarding data do not have to be maintained, in contrast to proactive routing protocols that require routing information to all routers within the MANET be maintained.

In addition to routing for its own local applications, each AODVv2 router can also route on behalf of other non-routing nodes (in this document, "Router Clients"), reachable via Client Interfaces. Each AODVv2 router, if serving router clients other than itself, SHOULD be configured with information about the IP addresses of its clients, using any suitable method. In the initial state, no AODVv2 router is required to have information about the relationship between any other AODVv2 router and its Router Clients (see Section 5.3).

The coordination among multiple AODVv2 routers to distribute routing information correctly for a shared address (i.e. an address that is advertised and can be reached via multiple AODVv2 routers) is not described in this document. The AODVv2 router operation of shifting responsibility for a routing client from one AODVv2 router to another is described in Appendix G. Address assignment procedures are entirely out of scope for AODVv2. A Router Client SHOULD NOT be served by more than one AODVv2 router at any one time.

AODVv2 routers perform route discovery to find a route toward a particular destination. AODVv2 routers MUST must be configured to respond to RREQs for themselves and their clients. When AODVv2 is the only protocol interacting with the forwarding table, AODVv2 MAY be configured to perform route discovery for all unknown unicast destinations.

AODVv2 only supports bidirectional links. In the case of possible unidirectional links, blacklists (see Section 5.2) SHOULD be used, or other means (e.g. adjacency establishment with only neighboring routers that have bidirectional communication as indicated by NHDP [RFC6130]) of assuring and monitoring bi-directionality are recommended. Otherwise, persistent packet loss or persistent protocol failures could occur. The cost of bidirectional link L (denoted Cost(L)) may depend upon the direction across the link for which the cost is measured. If received over a link that is

unidirectional, metric information from incoming AODVv2 messages MUST NOT be used for route table updates.

The routing algorithm in AODVv2 may be operated at layers other than the network layer, using layer-appropriate addresses. The routing algorithm makes use of some persistent state; if there is no persistent storage available for this state, recovery can impose a performance penalty (e.g., in case of AODVv2 router reboots).

5. Data Structures

5.1. Route Table Entry

The route table entry is a conceptual data structure. Implementations MAY use any internal representation so long as it provides access to the information specified below.

A route table entry has the following fields:

Route.Address
: The address or address prefix of one or more TargNode(s)

Route.PrefixLength
: The length of the address or prefix. If the value of Route.PrefixLength is less than the length of addresses in the address family used by the AODVv2 routers, the associated address is an address prefix, rather than an address. A PrefixLength is stored for every route in the route table.

Route.SeqNum
: The Sequence Number associated with Route.Address, as obtained from the last packet that successfully updated this route table entry.

Route.NextHopAddress
: The IP address of the adjacent AODVv2 router used for the path toward the Route.Address

Route.NextHopInterface
: The interface used to send packets toward Route.Address

Route.LastUsed
: The time that this route was last used

Route.ExpirationTime
: The time at which this route must expire

Route.MetricType
: The type of the metric for the route towards Route.Address

Route.Metric
: The cost of the route towards Route.Address expressed in units consistent with Route.MetricType

Route.State
: The last *known* state of the route. Route.State is one of the following: Active, Idle, Expired, Broken or Timed.

Route.Precursors (optional)
A list of upstream nodes using the route.

A route table entry (i.e., a route) is in one of the following states:

Active
An Active route is in current use for forwarding packets. The route's state determines the operations that can be performed on the route table entry. During use, an Active route is maintained continuously by AODVv2 and is considered to remain active as long as it is used at least once during every ACTIVE_INTERVAL. When a route is no longer Active, it becomes an Idle route.

Idle
An Idle route can be used for forwarding packets, even though it is not in current use. If an Idle route is used to forward a packet, it becomes an Active route once again. After an Idle route remains idle for MAX_IDLETIME, it becomes an Expired route.

Expired
After a route has been idle for too long, it expires, and may no longer be used for forwarding packets. An Expired route is not used for forwarding, but the sequence number information can be maintained until the destination sequence number has had no updates for MAX_SEQNUM_LIFETIME; after that time, old sequence number information is considered no longer valuable and the Expired route MUST BE expunged.

Broken
A route marked as Broken cannot be used for forwarding packets but still has valid destination sequence number information. When the link to a route's next hop is broken, the route is marked as being Broken, and afterwards the route MAY NOT be used.

Timed
The expiration of a Timed route is controlled by the Route.ExpirationTime time of the route table entry (instead of MAX_IDLETIME). Until that time, a Timed route can be used for forwarding packets. Afterwards, the route must be Expired (or expunged).

MAX_SEQNUM_LIFETIME is the time after a reboot during which an AODVv2 router MUST NOT transmit any routing messages. Thus, if all other AODVv2 routers expunge routes to the rebooted router after that time interval, the rebooted AODVv2 router's sequence number will not be considered stale by any other AODVv2 router in the MANET.

5.2. Bidirectional Connectivity and Blacklists

To avoid repeated failure of Route Discovery, an AODVv2 router (HandlingRtr) handling a RREP message MUST attempt to verify connectivity towards RREQ_Gen. This MAY be done by including the Acknowledgement Request (AckReq) message TLV (see Section 15.2) in the RREP. In reply to an AckReq, an RREP_ACK message message MUST be sent. If the verification is not received within UNICAST_MESSAGE_SENT_TIMEOUT, HandlingRtr MUST put the upstream neighbor in the blacklist. RREQs received from a blacklisted router, or any router over a link that is known to be incoming-only, MUST NOT be regenerated by HandlingRtr. However, the upstream neighbor SHOULD NOT be permanently blacklisted; after a certain time (MAX_BLACKLIST_TIME), it SHOULD once again be considered as a viable upstream neighbor for route discovery operations.

For this purpose, a list of blacklisted routers along with their time of removal SHOULD be maintained:

Blacklist.Router
: The IP address of the router that did not verify bidirectional connectivity.

Blacklist.RemoveTime
: The time at which Blacklist.Router MAY be removed from the blacklist.

5.3. Router Clients and Client Networks

An AODVv2 router may offer routing services to other nodes that are not AODVv2 routers; such nodes are defined as Router Clients in this document.

For this purpose, CLIENT_ADDRESSES must be configured on each AODVv2 router with the following information:

Client IP address
: The IP address of the node that requires routing service from the AODVv2 router.

Client Prefix Length
: The length of the routing prefix associated with the client IP address.

If the Client Prefix Length is not the full length of the Client IP address, then the prefix defines a Client Network. If an AODVv2 router is configured to serve a Client Network, then the AODVv2 router MUST serve every node that has an address within the range defined by the routing prefix of the Client Network. The list of

Routing Clients for an AODVv2 router is never empty, since an AODVv2 router is always its own client as well.

5.4. AODVv2 Message Header Fields and Information Elements

In its default mode of operation, AODVv2 sends messages using the parameters for port number and IP protocol specified in [RFC5498] to carry protocol packets. By default, AODVv2 messages are sent with the IP destination address set to the link-local multicast address LL-MANET-Routers [RFC5498] unless otherwise specified. Therefore, all AODVv2 routers MUST subscribe to LL-MANET-Routers [RFC5498] to receive AODVv2 messages. In order to reduce multicast overhead, regenerated multicast packets in MANETs SHOULD be done according to methods specified in [RFC6621]. AODVv2 does not specify which method should be used to restrict the set of AODVv2 routers that have the responsibility to regenerate multicast packets. Note that multicast packets MAY be sent via unicast. For example, this may occur for certain link-types (non-broadcast media), for manually configured router adjacencies, or in order to improve robustness.

The IPv4 TTL (IPv6 Hop Limit) field for all packets containing AODVv2 messages is set to 255. If a packet is received with a value other than 255, any AODVv2 message contained in the packet MUST be disregarded by AODVv2. This mechanism, known as "The Generalized TTL Security Mechanism" (GTSM) [RFC5082] helps to assure that packets have not traversed any intermediate routers.

IP packets containing AODVv2 protocol messages SHOULD be given priority queuing and channel access.

AODVv2 messages are transmitted in messages that conform to the packet and message format specified in [RFC5444]. Here is a brief summary of the format.

> A packet formatted according to RFC 5444 contains zero or more messages.
> A message contains a message header, message TLV block, and zero or more address blocks.
> Each address block MAY also have an associated TLV block; this TLV block MAY encode multiple TLVs. Each such TLV may include an array of values. The list of TLV values may be associated with various subsets of the addresses in the address block.

If a packet contains only a single AODVv2 message and no packet TLVs, it need only include a minimal Packet-Header [RFC5444]. The length of an address (32 bits for IPv4 and 128 bits for IPv6) inside an AODVv2 message is indicated by the msg-addr-length (MAL) in the msg-header, as specified in [RFC5444].

5.5. Sequence Numbers

Sequence Numbers allow AODVv2 routers to evaluate the freshness of routing information. Each AODVv2 router in the network MUST maintain its own sequence number. Each RREQ and RREP generated by an AODVv2 router includes that sequence number. Each AODVv2 router MUST make sure that its sequence number is unique and monotonically increasing. This can be achieved by incrementing it with every RREQ or RREP it generates.

Every router receiving a RREQ or RREP can thus use the Sequence Number of a RREQ or RREP as information concerning the freshness of the packet's route update: if the new packet's Sequence Number is lower than the one already stored in the route table, its information is considered stale.

As a consequence, loop freedom is assured.

An AODVv2 router increments its SeqNum as follows. Most of the time, SeqNum is incremented by simply adding one (1). But when the SeqNum has the value of the largest possible number representable as a 16-bit unsigned integer (i.e., 65,535), it MUST be incremented by setting to one (1). In other words, the sequence number after 65,535 is 1.

An AODVv2 router SHOULD maintain its SeqNum in persistent storage. If an AODVv2 router's SeqNum is lost, it MUST take the following actions to avoid the danger of routing loops. First, the AODVv2 router MUST invalidate all route table entries, by setting Route.State = Broken for each entry. Furthermore the AODVv2 router MUST wait for at least MAX_SEQNUM_LIFETIME before transmitting or regenerating any AODVv2 RREQ or RREP messages. If an AODVv2 protocol message is received during this waiting period, the AODVv2 router SHOULD perform normal route table entry updates, but not forward the message to other nodes. If a data packet is received for forwarding to another destination during this waiting period, the AODVv2 router MUST transmit a RERR message indicating that no route is available. At the end of the waiting period the AODVv2 router sets its SeqNum to one (1) and begins performing AODVv2 protocol operations again.

5.6. Enabling Alternate Metrics

AODVv2 route selection in MANETs depends upon associating metric information with each route table entry. When presented with candidate route update information, deciding whether to use the update involves evaluating the metric. Some applications may require metric information other than Hop Count, which has traditionally been the default metric associated with routes in MANET. Unfortunately,

it is well known that reliance on Hop Count can cause selection of the worst possible route in many situations.

It is beyond the scope of this document to describe how applications specify route selection at the time they launch processing. One possibility would be to provide a route metric preference as part of the library routines for opening sockets. In view of the above considerations, it is important to enable route selection based on metric information other than Hop Count -- in other words, based on "alternate metrics". Each such alternate metric measures a "cost" of using the associated route, and there are many different kinds of cost (latency, delay, monetary, energy, etc.). The range and data type of each such alternate metric may be different. For instance, the data type might be integers, or floating point numbers, or restricted subsets thereof.

The most significant change when enabling use of alternate metrics is to require the possibility of multiple routes to the same destination, where the "cost" of each of the multiple routes is measured by a different metric. Moreover, the method by which route updates are tested for usefulness has to be slightly generalized to depend upon a more abstract method of evaluation which, in this document, is named "Cost(R)", where 'R' is the route for which the Cost is to be evaluated. From the above, the route table information for 'R' must always include the type of metric by which Cost(R) is evaluated, so the metric type does not have to be shown as a distinct parameter for Cost(R). Since determining loop freedom is known to depend on comparing the Cost(R) of route update information to the Cost(R) of an existing stored route using the same metric, AODVv2 must also be able to invoke an abstract routine which in this document is called "LoopFree(R1, R2)". LoopFree(R1, R2) returns TRUE when, (under the assumption of nondecreasing SeqNum during Route Discovery) given that R2 is loop-free and Cost(R2) is the cost of route R2, Cost(R1) is known to guarantee loop freedom of the route R1. In this document, an AODVv2 router will only invoke LoopFree (AdvRte, Route), for routes AdvRte and Route which use the same metric to the same destination. AdvRte is the route advertised in an incoming RREQ or RREP, and is used as parameter R1 for LoopFree. Route is a route already existing in the AODVv2 router's route table, and is used as parameter R2 for LoopFree.

Generally, HopCount may still be considered the default metric for use in MANETs, notwithstanding the above objections. Each metric has to have a Metric Type, and the Metric Type is allocated by IANA as specified in [RFC6551]. Each Route has to include the Metric Type as part of the route table entry for that route. Hop Count has Metric Type assignment 3. The Cost of a route using Metric Type 3 is simply the hop count between the router and the destination. Using Metric

Type 3, LoopFree (AdvRte, Route) is TRUE when Cost(AdvRte) <= Cost(Route). The specification of Cost(R) and LoopFree(AdvRte, Route) for metric types other than 3 is beyond the scope of this document.

Whenever an AODV router receives metric information in an incoming message, the value of the metric is as measured by the transmitting router, and does not reflect the cost of traversing the incoming link. In order to simplify the description of storing accrued route costs in the route table, the Cost() function is also defined to return the value of traversing a link 'L'. In other words, the domain of the Cost() function is enlarged to include links as well as routes. For Metric Type 3, (i.e., the HopCount metric) Cost(L) = 1 for all links L. The specification of Cost(L) for metric types other than 3 is beyond the scope of this document. Whether the argument of the Cost() function is a link or a route will, in this document, always be clear. As a natural result of the way routes are looked up according to conformant metric type, all intermediate routers handling a RteMsg will assign the same metric type to all metric information in the RteMsg.

For some metrics, a maximum value is defined, namely MAX_METRIC[i] where 'i' is the Metric Type. AODVv2 does not store routes that cost more than MAX_METRIC[i]. MAX_METRIC[3] is defined to be MAX_HOPCOUNT, where as before 3 is the Metric Type of the HopCount metric. MAX_HOPCOUNT MUST be larger than the AODVv2 network diameter. Otherwise, AODVv2 protocol messages may not reach their intended destinations.

5.7. RREQ Table: Received RREQ Messages

Two incoming RREQ messages are considered to be "comparable" if they were generated by the same AODVv2 router in order to discover a route for the same destination with the same metric type. According to that notion of comparability, when RREQ messages are flooded in a MANET, an AODVv2 router may well receive comparable RREQ messages from more than one of its neighbors. A router, after receiving an RREQ message, MUST check against previous RREQs to assure that its response message would contain information that is not redundant (see Section 7.6 regarding suppression of redundant RREQ messages). Otherwise, multicast RREQs are likely to be regenerated again and again with almost no additional benefit, but generating a great deal of unnecessary signaling traffic and interference.

To avoid transmission of redundant RREQ messages, while still enabling the proper handling of earlier RREQ messages that may have somehow been delayed in the network, it is needed for each AODVv2

router to keep a list of the certain information about RREQ messages which it has recently received.

This list is called the AODVv2 Received RREQ Table -- or, more briefly, the RREQ Table. Two AODVv2 RREQ messages are comparable if:

- they have the same metric type
- they have the same OrigNode and TargNode addresses

Each entry in the RREQ Table has the following fields:

- OrigNode address
- TargNode address
- OrigNode Sequence Number
- TargNode Sequence Number (if present in RREQ)
- Metric Type
- Metric
- Timestamp

The RREQ Table is maintained so that no two entries in the RREQ Table are comparable -- that is, all RREQs represented in the RREQ Table either have different OrigNode addresses, different TargNode addresses, or different metric types. If two RREQs have the same metric type and OrigNode and Targnode addresses, the information from the one with the older Sequence Number is not needed in the table; in case they have the same Sequence Number, the one with the greater Metric value is not needed; in case they have the same Metric as well, it does not matter which table entry is maintained. Whenever a RREQ Table entry is updated, its Timestamp field should also be updated to reflect the Current_Time.

When optional multicast RREP (see Section 13.4) is used to enable selection from among multiple possible return routes, an AODVv2 router can eliminate redundant RREP messages using the analogous mechanism along with a RREP Table. The description in this section only refers to RREQ multicast messages.

Protocol handling of RERR messages eliminates the need for tracking RERR messages, since the rules for RERR regeneration prevent the phenomenon of redundant retansmission that affects RREQ and RREP multicast.

6. AODVv2 Operations on Route Table Entries

In this section, operations are specified for updating the route table due to timeouts and route updates within AODVv2 messages. Route update information in AODVv2 messages includes IP addresses, along with the SeqNum and prefix length associated with each IP

address, and including the Metric measured from the node transmitting the AODVv2 message to the IP address in the route update. IP addresses and prefix length are encoded within an RFC 5444 AddrBlk, and the SeqNum and Metric associated with each address in the AddrBlk are encoded in RFC 5444 AddrTLVs. A RREQ message advertises a route to OrigNode, and a RREP message analogously advertises a route to TargNode. In this section, RteMsg is either RREQ or RREP, and AdvRte is the route advertised by the RteMsg. All SeqNum comparisons use signed 16-bit arithmetic.

6.1. Evaluating Incoming Routing Information

If the incoming RteMsg does not have a Metric Type Message TLV, then the metric information contained by AdvRte is considered to be of type DEFAULT_METRIC_TYPE -- in other words, 3 (for HopCount) unless changed by administrative action. The AODVv2 router (HandlingRtr) checks the advertised route (AdvRte) to see whether the AdvRte should be used to update an existing route table entry. HandlingRtr searches its route table to see if there is a route table entry with the same Metric Type as the AdvRte, matching AdvRte.Address. If not, HandlingRtr creates a route table entry for AdvRte.Address as described in Section 6.2. Otherwise, HandlingRtr compares the incoming routing information for AdvRte against the already stored routing information in the route table entry (Route) for AdvRte.Address, as described next.

Route[AdvRte.Address] uses the same metric type as the incoming routing information, and the route entry contains Route.SeqNum, Route.Metric, and Route.State. Define AdvRte.SeqNum and AdvRte.Metric to be the corresponding routing information for Route.Address in the incoming RteMsg. Define AdvRte.Cost to be (AdvRte.Metric + Cost(L)), where L is the link from which the incoming message was received. The incoming routing information is classified as follows:

1. Stale:: AdvRte.SeqNum < Route.SeqNum :
   If AdvRte.SeqNum < Route.SeqNum the incoming information is stale. Using stale routing information is not allowed, since that might result in routing loops. In this case, HandlingRtr MUST NOT update the route table entry using the routing information for AdvRte.Address.
2. Unsafe against loops:: (TRUE != LoopFree (AdvRte, Route)) :
   If AdvRte is not Stale (as in (1) above), AdvRte.Cost is next considered to insure loop freedom. If (TRUE != LoopFree (AdvRte, Route)) (see Section 5.6), then the incoming AdvRte information is not guaranteed to prevent routing loops, and it MUST NOT be used to update any route table entry.
3. More costly::

(AdvRte.Cost >= Route.Metric) && (Route.State != Broken)
When AdvRte.SeqNum is the same as in a valid route table entry, and LoopFree (AdvRte, Route) assures loop freedom, incoming information still does not offer any improvement over the existing route table information if AdvRte.Cost >= Route.Metric. Using such incoming routing information to update a route table entry is not recommended.

4. Offers improvement::
Advertised routing information that does not match any of the above criteria is better than existing route table information and SHOULD be used to improve the route table. The following pseudo-code illustrates whether advertised routing information should be used to update an existing route table entry as described in Section 6.2.

```
(AdvRte.SeqNum > Route.SeqNum) OR
    ((AdvRte.SeqNum == Route.SeqNum) AND
           [(AdvRte.Cost < Route.Metric) OR
            ((Route.State == Broken) && LoopFree (AdvRte, Route))])
```

The above logic corresponds to placing the following conditions (compared to the existing route table entry) on the advertised route update before it can be used:

* it is more recent, or
* it is not stale and is less costly, or
* it can safely repair a broken route.

6.2. Applying Route Updates To Route Table Entries

To apply the route update, a route table entry for AdvRte.Address is either found to already exist in the route table, or else a new route table entry for AdvRte.Address is created and inserted into the route table. If the route table entry already exists, and the state is Expired or Broken, then the state is reset to be Idle. If the route table entry had to be created, the state is set to be Active. The route table entry is populated with the following information:

- If AdvRte.PrefixLength exists, then Route.PrefixLength := AdvRte.PrefixLength. Otherwise, Route.PrefixLength := maximum length for address family (either 32 or 128).
- Route.SeqNum := AdvRte.SeqNum
- Route.NextHopAddress := IP.SourceAddress (i.e., an address of the node from which the RteMsg was received)
- Route.NextHopInterface is set to the interface on which RteMsg was received
- Route.MetricType := AdvRte.MetricType.
- Route.Metric := AdvRte.Cost

o Route.LastUsed := Current_Time
o If RteMsg.VALIDITY_TIME is included, then Route.ExpirationTime := Current_Time + RteMsg.VALIDITY_TIME, otherwise, Route.ExpirationTime := Current_Time + (ACTIVE_INTERVAL + MAX_IDLETIME).

With these assignments to the route table entry, a route has been made available, and the route can be used to send any buffered data packets and subsequently to forward any incoming data packets for Route.Address. An updated route entry also fulfills any outstanding route discovery (RREQ) attempts for Route.Address.

6.3. Route Table Entry Timeouts

During normal operation, AODVv2 does not require any explicit timeouts to manage the lifetime of a route. However, the route table entry MUST be examined before using it to forward a packet, as discussed in Section 8.1. Any required expiry or deletion can occur at that time. Nevertheless, it is permissible to implement timers and timeouts to achieve the same effect.

At any time, the route table can be examined and route table entries can be expunged according to their current state at the time of examination, as follows.

o An Active route MUST NOT be expunged.
o An Idle route SHOULD NOT be expunged.
o An Expired route MAY be expunged (least recently used first).
o A route MUST be expunged if (Current_Time - Route.LastUsed) >= MAX_SEQNUM_LIFETIME.
o A route MUST be expunged if Current_Time >= Route.ExpirationTime

If precursor lists are maintained for the route (as described in Section 13.3) then the precursor lists must also be expunged at the same time that the route itself is expunged.

7. Routing Messages RREQ and RREP (RteMsgs)

AODVv2 message types RREQ and RREP are together known as Routing Messages (RteMsgs) and are used to discover a route between an Originating and Target Node, denoted here by OrigNode and TargNode. The constructed route is bidirectional, enabling packets to flow between OrigNode and TargNode. RREQ and RREP have similar information and function, but have some differences in their rules for handling. When a node receives a RREQ or a RREP, the node then creates or updates a route to the OrigNode or the TargNode respectively. The main difference between the two messages is that

RREQ messages are typically multicast to solicit a RREP, whereas RREP is typically unicast as a response to RREQ.

When an AODVv2 router needs to forward a data packet from a node (OrigNode) in its set of router clients, and it does not have a forwarding route toward the packet's IP destination address (TargNode), the AODVv2 router (RREQ_Gen) generates a RREQ (as described in Section 7.3) to discover a route toward TargNode. Subsequently RREQ_Gen awaits reception of an RREP message (see Section 7.4) or other route table update (see Section 6.2) to establish a route toward TargNode. The RREQ message contains routing information to enable RREQ recipients to route packets back to OrigNode, and the RREP message contains routing information enabling RREP recipients to route packets to TargNode.

7.1. Route Discovery Retries and Buffering

After issuing a RREQ, as described above RREQ_Gen awaits a RREP providing a bidirectional route toward Target Node. If the RREP is not received within RREQ_WAIT_TIME, RREQ_Gen MAY retry the Route Discovery by generating another RREQ. Route Discovery SHOULD be considered to have failed after DISCOVERY_ATTEMPTS_MAX and the corresponding wait time for a RREP response to the final RREQ. After the attempted Route Discovery has failed, RREQ_Gen MUST wait at least RREQ_HOLDDOWN_TIME before attempting another Route Discovery to the same destination.

To reduce congestion in a network, repeated attempts at route discovery for a particular Target Node SHOULD utilize a binary exponential backoff.

Data packets awaiting a route SHOULD be buffered by RREQ_Gen. This buffer SHOULD have a fixed limited size (BUFFER_SIZE_PACKETS or BUFFER_SIZE_BYTES). Determining which packets to discard first is a matter of policy at each AODVv2 router; in the absence of policy constraints, by default older data packets SHOULD be discarded first. Buffering of data packets can have both positive and negative effects (albeit usually positive). Nodes without sufficient memory available for buffering SHOULD be configured to disable buffering by configuring BUFFER_SIZE_PACKETS == 0 and BUFFER_SIZE_BYTES == 0. Doing so will affect the latency required for launching TCP applications to new destinations.

If a route discovery attempt has failed (i.e., DISCOVERY_ATTEMPTS_MAX attempts have been made without receiving a RREP) to find a route toward the Target Node, any data packets buffered for the corresponding Target Node MUST BE dropped and a Destination Unreachable ICMP message (Type 3) SHOULD be delivered to the source

of the data packet. The code for the ICMP message is 1 (Host unreachable error). If RREQ_Gen is not the source (OrigNode), then the ICMP is sent over the interface from which OrigNode sent the packet to the AODVv2 router.

7.2. RteMsg Structure

AODVv2 specifies that all control plane messages between Routers SHOULD use the Generalised Mobile Ad-hoc Network Packet and Message Format [RFC5444], which provides a multiplexed transport for multiple protocols. AODVv2 therefore specifies Route Messages that have components that map to message elements in RFC5444 but, in line with the concept of use, does not specify which order the messages should be arranged in an RFC5444 packet. An implementation of an RFC5444 parser may choose to optimise the content of certain message elements to reduce control plane overhead.

AODVv2 uses the following RFC5444 message elements:

- o Address of the originating node, OrigNode, which should be mapped to the <msg-orig-addr> element in <msg-header>.
- o Message Hop Count, <msg-hop-count>, which should be mapped to the <msg-hop-count> element in <msg-header>.
- o Message Hop Limit, <msg-hop-limit>, which should be mapped to the <msg-hop-limit> element in <msg-header>.

RteMsgs have the following general format:

```
+-----------------------------------------------------------------+
|                   RFC 5444 Message Header                       |
+-----------------------------------------------------------------+
|                      MsgTLVs (optional)                         |
+-----------------------------------------------------------------+
|                 AddrBlk := {OrigNode,TargNode}                  |
+-----------------------------------------------------------------+
|      AddrBlk.PrefixLength[OrigNode OR TargNode] (Optional)      |
+-----------------------------------------------------------------+
|             OrigSeqNum_TLV AND/OR TargSeqNum_TLV                |
+-----------------------------------------------------------------+
|                 Metric TLV {OrigNode, TargNode}                 |
+-----------------------------------------------------------------+
```

Figure 1: RREQ and RREP (RteMsg) message structure

Required RFC 5444 Message Header Fields

* <msg-hop-limit>

* Metric Type Message TLV, if Metric Type != 3

Optional RFC 5444 Message Header Fields

* <msg-hop-count>
* Metric Type TLV (Metric Type for Metric AddrTLV)
* AckReq TLV (Acknowledgement Requested)

AddrBlk
The Address Block contains the IP addresses for RREQ Originating and Target Node (OrigNode and TargNode). For both RREP and RREQ, OrigNode and TargNode are as identified in the context of the RREQ message originator.

OrigSeqNum AND/OR TargSeqNum AddrTLV
At least one of OrigSeqNum or TargSeqNum Address Block TLV is REQUIRED and carries the destination sequence numbers associated with OrigNode or TargNode respectively.

Metric AddrTLV
The Metric AddrTLV is REQUIRED and carries the route metric information associated with either OrigNode or TargNode.

RteMsgs carry information about OrigNode and TargNode. Since their addresses may appear in arbitrary order within the RFC 5444 AddrBlk, the OrigSeqNum and/or TargSeqNum TLVs must be used to distinguish the nature of the node addresses present in the AddrBlk. In each RteMsg, either the OrigSeqNum TLV or TargSeqNum TLV MUST appear. Both TLVs MAY appear in the same RteMsg when SeqNum information is available for both OrigNode and TargNode, but each one MUST NOT appear more than once, because there is only one OrigNode and only one TargNode address in the AddrBlk. The TLV flag thassingleindex MUST be set for these TLVs.

If the OrigSeqNum TLV appears, then the address range for the OrigSeqNum TLV MUST be limited to a single position in the AddrBlk. That position is used as the OrigNdx, identifying the OrigNode address. The other address in the AddrBlk is, by elimination, the TargNode address, and TargNdx is set appropriately.

Otherwise, if the TargSeqNum TLV appears, then the address range for the TargSeqNum TLV MUST be limited to a single position in the AddrBlk. That position is used as the TargNdx, identifying the TargNode address. The other address in the AddrBlk is, by elimination, the OrigNode address, and OrigNdx is set appropriately.

7.3. RREQ Generation

The AODVv2 router generating the RREQ (RREQ_Gen) on behalf of its client OrigNode follows the steps in this section. OrigNode MUST be a unicast address. The order of protocol elements is illustrated schematically in Figure 1.

1. RREQ_Gen MUST increment its SeqNum by one (1) according to the rules specified in Section 5.5. This assures that each node receiving the RREQ will update its route table using the information in the RREQ.
2. <msg-hop-limit> SHOULD be set to MAX_HOPCOUNT.
3. <msg-hop-count>, if included, MUST be set to 0.

   * This RFC 5444 constraint causes certain RREQ payloads to incur additional enlargement (otherwise, <msg-hop-count> could often be used as the metric).
4. RREQ.AddrBlk := {OrigNode.Addr, TargNode.Addr}

   Let OrigNdx and TargNdx denote the indexes of OrigNode and TargNode respectively in the RREQ.AddrBlk list.
5. If Route[OrigNode].PrefixLength/8 is equal to the number of bytes in the addresses of the RREQ (4 for IPv4, 16 for IPv6), then no <prefix-length> is included with the RREQ.AddrBlk. Otherwise, RREQ.PrefixLength[OrigNdx] := Route[OrigNode].PrefixLength according to the rules of RFC 5444 AddrBlk encoding.
6. RREQ.OrigSeqNum_TLV[OrigNdx] := RREQ_Gen's SeqNum
7. RREQ.TargSeqNum_TLV[TargNdx] := TargNode's SeqNum (only if known)

   RREQ_Gen SHOULD include TargNode's SeqNum, if a previous value of the TargNode's SeqNum is known (e.g., from an invalid route table entry using longest-prefix matching). If TargNode's SeqNum is not included, AODVv2 routers handling the RREQ assume that RREQ_Gen does not have that information.
8. RREQ.Metric_TLV[OrigNdx] := Route[OrigNode].Metric

An example RREQ message format is illustrated in Appendix B.1.

7.4. RREP Generation

This section specifies the generation of an RREP by an AODVv2 router (RREP_Gen) that provides connectivity for the Target Node (TargNode) of a RREQ, thus enabling the establishment of a route between OrigNode and TargNode. If TargNode is not a unicast IP address the RREP MUST NOT be generated, and processing for the RREQ is complete. Before transmitting a RREP, the routing information of the RREQ is processed as specified in Section 6.2; after such processing, RREP_Gen has an updated route to OrigNode as well as TargNode. The basic format of an RREP conforms to the structure for RteMsgs as shown in Figure 1.

RREP_Gen generates the RREP as follows:

1. RREP_Gen checks the RREQ against recently received RREQ information as specified in Section 7.6. If a previously

received RREQ has made the information in the incoming RREQ to be redundant, no RREP is generated and processing is complete.

2. RREP_Gen MUST increment its SeqNum by one (1) according to the rules specified in Section 5.5.
3. RREP.AddrBlk := {OrigNode.Addr, TargNode.Addr}

   Let OrigNdx and TargNdx denote the indexes of OrigNode and TargNode respectively in the RREQ.AddrBlk list.
4. RREP.TargSeqNum_TLV[TargNdx] := RREP_Gen's SeqNum
5. If Route[TargNode].PrefixLength/8 is equal to the number of bytes in the addresses of the RREQ (4 for IPv4, 16 for IPv6), then no <prefix-length> is included with the RREP.AddrBlk. Otherwise, RREP.PrefixLength[TargNdx] := Route[TargNode].PrefixLength according to the rules of RFC 5444 AddrBlk encoding.
6. If (DEFAULT != Route[TargNode].MetricType) then include the Metric Type message TLV and assign RREP.MetricType[TargNdx] := Route[TargNode].MetricType
7. RREP.Metric_TLV[TargNdx] := Route[TargNode].Metric
8. <msg-hop-count>, if included, MUST be set to 0.
9. <msg-hop-limit> SHOULD be set to RREQ.<msg-hop-count>.
10. IP.DestinationAddress := Route[OrigNode].NextHop

An example message format for RREP is illustrated in Appendix B.2.

7.5. Handling a Received RteMsg

Before an AODVv2 router can make use of a received RteMsg (i.e., RREQ or RREP), the router first must verify that the RteMsg is permissible according to the following steps. OrigNdx and TargNdx are set according to the rules in Section 7.2. For RREQ, RteMsg.Metric is Metric_TLV[OrigNdx]. For RREP, RteMsg.Metric is Metric_TLV[TargNdx]. In this section (unless qualified by additional description such as "upstream" or "neighboring") all occurrences of the term "router" refer to the AODVv2 router handling the received RteMsg.

1. A router MUST handle RteMsgs only from neighbors as specified in Section 5.4. RteMsgs from other sources MUST be disregarded.
2. The router examines the RteMsg to ascertain that it contains the required information: <msg-hop-limit>, TargNode.Addr, OrigNode.Addr, RteMsg.Metric, and either RteMsg.OrigSeqNum or RteMsg.TargSeqNum. If the required information does not exist, the message is disregarded.
3. The router checks that OrigNode.Addr and TargNode.Addr are valid routable unicast addresses. If not, the message is disregarded.
4. The router checks the Metric Type MsgTLV (if present) to assure that the Metric Type associated with the Metric AddrTLV

information in the RREQ or RREP is known. If not, the message is disregarded.

* DISCUSSION: or, can change the AddrBlk metric to use HopCount, e.g., measured from <msg-hop-count>.

5. If (MAX_METRIC[RteMsg.MetricType] - Cost(L)) <= RteMsg.Metric, the RteMsg is disregarded, where Cost(L) denotes the cost of traversing the incoming link (i.e., as measured by the network interface receiving the incoming RteMsg).

An AODVv2 router handles a permissible RteMsg according to the following steps.

1. The router MUST process the routing information for OrigNode and TargNode contained in the RteMsg as specified in Section 6.1.
2. If RteMsg.<msg-hop-limit> is zero (0), no further action is taken, and the RteMsg is not regenerated. Otherwise, the router MUST decrement RteMsg.<msg-hop-limit>.
3. If the RteMsg.<msg-hop-count> is present, and MAX_HOPCOUNT <= <msg-hop-count>, then no further action is taken. Otherwise, the router MUST increment RteMsg.<msg-hop-count>

Further actions to transmit an updated RteMsg depend upon whether the incoming RteMsg is an RREP or an RREQ.

7.5.1. Additional Handling for Incoming RREQ

o By sending a RREQ, a router advertises that it will route for addresses contained in the RteMsg based on the information enclosed. The router MAY choose not to send the RREQ, though not resending the RREQ could decrease connectivity in the network or result in nonoptimal paths. The circumstances under which a router might choose not to re-transmit a RREQ are not specified in this document. Some examples might include the following:

  * The router is already heavily loaded and does not want to advertise routing for more traffic
  * The router recently transmitted identical routing information (e.g. in a RREQ advertising the same metric) Section 7.6
  * The router is low on energy and has to reduce energy expended for sending protocol messages or packet forwarding

  Unless the router is prepared to send a RREQ, it halts processing.

o If the upstream router sending a RREQ is in the Blacklist, and Current_Time < Blacklist.RemoveTime, then the router receiving that RREQ MUST NOT transmit any outgoing RteMsg, and processing is complete.

- o Otherwise, if the upstream router is in the Blacklist, and Current_Time >= Blacklist.RemoveTime, then the upstream router SHOULD be removed from the Blacklist, and message processing continued.
- o The incoming RREQ MUST be checked against previously received information from the RREQ Table (Section 7.6). If the information in the incoming RteMsg is redundant, then then no further action is taken.
- o If TargNode is a client of the router receiving the RREQ, then the router generates a RREP message as specified in Section 7.4, and subsequently processing for the RREQ is complete. Otherwise, processing continues as follows.
- o If (DEFAULT != Route[OrigNode].MetricType) then include the Metric Type message TLV and assign RREQ.MetricType := Route[OrigNode].MetricType
- o RREQ.Metric_TLV[OrigNdx] := Route[OrigNode].Metric
- o The RREQ (with updated fields as specified above>) SHOULD be sent to the IP multicast address LL-MANET-Routers [RFC5498]. If the RREQ is unicast, the IP.DestinationAddress is set to Route[RREQ.TargNode].NextHopAddress.

7.5.2. Additional Handling for Incoming RREP

As always, OrigNode and TargNode are named in the context of RREQ_Gen (i.e., the router originating the RREQ for which the RREP was generated) (see Table 1). OrigNdx and TargNdx are set according to the rules in Section 7.2.

- o If no forwarding route exists to OrigNode, then a RERR SHOULD be transmitted to RREP.AddrBlk[TargNdx]. Otherwise, if HandlingRtr is not RREQ_Gen then the outgoing RREP is sent to the Route.NextHopAddress for the RREP.AddrBlk[OrigNdx].
- o If HandlingRtr is RREQ_Gen then the RREP satisfies RREQ_Gen's earlier RREQ, and RREP processing is completed. Any packets buffered for OrigNode should be transmitted.

7.6. Suppressing Redundant RREQ messages

Since RREQ messages are multicast, there are common circumstances in which an AODVv2 router might transmit a redundant response (RREQ or RREP), duplicating the information transmitted in response to some other recent RREQ (see Section 5.7). Before responding, an AODVv2 router MUST suppress such RREQ messages. This is done by checking the list of recently received RREQs to determine whether the incoming RREQ is redundant, as follows:

- o The AODVv2 router searches the RREQ Table for recent entries with the same OrigNode, TargNode, and Metric Type. If there is no such

entry, the incoming RREQ message is not suppressed. A new entry for the incoming RREQ is created in the RREQ Table.

- If there is such an entry, and the incoming RREQ has a newer sequence number, the incoming RREQ is not suppressed, and the existing table entry MUST be updated to reflect the new Sequence Number and Metric.
- Similarly, if the Sequence Numbers are the same, and the incoming RREQ offers a better Metric, the incoming RREQ is not suppressed, and the RREQ Table entry MUST be updated to reflect the new Metric.
- Otherwise, the incoming RREQ is suppressed.

8. Route Maintenance and RERR Messages

AODVv2 routers attempt to maintain active routes. When a routing problem is encountered, an AODVv2 router (denoted RERR_Gen) attempts to quickly notify upstream routers. Two kinds of routing problems may trigger generation of a RERR message. The first case happens when the router receives a packet but does not have a route for the destination of the packet. The second case happens immediately upon detection of a broken link (see Section 8.2) of an Active route, to quickly notify upstream AODVv2 routers that that route is no longer available.

8.1. Maintaining Route Lifetimes During Packet Forwarding

Before using a route to forward a packet, an AODVv2 router MUST check the status of the route as follows.

- If the route is marked has been marked as Broken, it cannot be used for forwarding.
- If Current_Time > Route.ExpirationTime, the route table entry has expired, and cannot be used for forwarding.
- Similarly, if (Route.ExpirationTime == MAXTIME), and if (Current_Time - Route.LastUsed) > (ACTIVE_INTERVAL + MAX_IDLETIME), the route has expired, and cannot be used for forwarding.
- Furthermore, if Current_Time - Route.LastUsed > (MAX_SEQNUM_LIFETIME), the route table entry MUST be expunged.

If any of the above route error conditions hold true, the route cannot be used to forward the packet, and an RERR message MUST be generated (see Section 8.3).

Otherwise, Route.LastUsed := Current_Time, and the packet is forwarded to the route's next hop.

Optionally, if a precursor list is maintained for the route, see Section 13.3 for precursor lifetime operations.

8.2. Active Next-hop Router Adjacency Monitoring

Neighboring routers MAY form an adjacency based on various information or other protocols; for example, exchange of AODVv2 routing messages, other protocols (e.g. NDP [RFC4861] or NHDP [RFC6130]), or manual configuration. Loss of a routing adjacency may also be indicated by similar information. AODVv2 routers SHOULD monitor connectivity to adjacent routers along active routes. This monitoring can be accomplished by one or several mechanisms, including:

- o Neighborhood discovery [RFC6130]
- o Route timeout
- o Lower layer trigger that a link is broken
- o TCP timeouts
- o Promiscuous listening
- o Other monitoring mechanisms or heuristics

If a next-hop AODVv2 router has become unreachable, RERR_Gen follows the procedures specified in Section 8.3.2.

8.3. RERR Generation

An RERR message is generated by a AODVv2 router (i.e., RERR_Gen) in order to notify upstream routers that packets cannot be delivered to certain destinations. An RERR message has the following general structure:

```
+-----------------------------------------------------------------+
|     RFC 5444 Message Header <msg-hoplimit> <msg-hopcount>       |
+-----------------------------------------------------------------+
|      UnreachableNode AddrBlk (Unreachable Node addresses)       |
+-----------------------------------------------------------------+
|        AddrBlk.PrefixLength[UnreachableNodes] (Optional)        |
+-----------------------------------------------------------------+
|               UnreachableNode SeqNum AddrBlk TLV                |
+-----------------------------------------------------------------+
|               UnreachableNode PfxLen AddrBlk TLV                |
+-----------------------------------------------------------------+
```

Figure 2: RERR message structure

Required Message Header Fields
   The RERR MUST contain the following:

* <msg-hop-limit>
* PktSource Message TLV (see Section 15), if the RERR is unicast
* Metric Type Message TLV (see Section 15), if Metric Type != 3

Optional Message Header Fields
The RERR SHOULD contain the following:

* <msg-hop-count>

UnreachableNode AddrBlk
This Address Block contains the IP addresses unreachable by AODVv2 router transmitting the RERR.

UnreachableNode.PrefixLength
If needed, the Address Block can also carry the prefix length associated with each UnreachableNode.

Sequence Number AddrBlk TLV
This Address Block TLV carries the destination sequence number associated with each UnreachableNode when that information is available.

There are two kinds of events indicating that packets cannot be delivered to certain destinations. The two cases differ in the way that the neighboring IP destination address for the RERR is chosen, and in the way that the set of UnreachableNodes is identified.

In both cases, the <msg-hop-limit> MUST be included and SHOULD be set to MAX_HOPCOUNT. <msg-hop-count> SHOULD be included and set to 0, to facilitate use of various route repair strategies including expanding rings multicast and Intermediate RREP [I-D.perkins-irrep].

8.3.1. Case 1: Undeliverable Packet

The first case happens when the router receives a packet from another AODVv2 router but does not have a valid route for the destination of the packet. In this case, there is exactly one UnreachableNode to be included in the RERR's AddrBlk (either IP.DestinationAddress from a data packet or the OrigNode address found in the AddrBlk of an RREP message). The RERR SHOULD be sent to the multicast address LL-MANET-Routers, but RERR_Gen MAY instead send the RERR to the next hop towards the source IP address of the packet which was undeliverable. For unicast RERR, the PktSource Message TLV MUST be included, containing the the source IP address of the undeliverable packet, or the IP address of TargRtr in case the undeliverable packet was an RREP message generated by TargRtr. If a Sequence Number for UnreachableNode is known, that Sequence Number SHOULD be included in a Seqnum AddrTLV the RERR. Otherwise all nodes handling the RERR will assume their route through RERR_Gen towards the UnreachableNode is no longer valid and mark those routes as broken, regardless of the Sequence Number information for those routes. RERR_Gen MUST discard the packet or message that triggered generation of the RERR.

If an AODVv2 router receives an ICMP packet from the address of one of its client nodes, it simply relays the packet to the ICMP packet's destination address, and does not generate any RERR message.

8.3.2. Case 2: Broken Link

The second case happens when the link breaks to an active adjacent AODVv2 router (i.e., the next hop of an active route). In this case, the RERR MUST be sent to the multicast address LL-MANET-Routers, except when the optional feature of maintaining precursor lists is used as specified in Section 13.3. All routes (Active, Idle and Expired) that use the broken link MUST be marked as Broken. The set of UnreachableNodes is initialized by identifying those Active routes which use the broken link. For each such Active Route, Route.Dest is added to the set of Unreachable Nodes. After the Active Routes using the broken link have all been included as UnreachableNodes, Idle routes MAY also be included, if allowed by the setting of ENABLE_IDLE_IN_RERR, as long as the packet size of the RERR does not exceed the MTU (interface "Maximum Transfer Unit") of the physical medium.

If the set of UnreachableNodes is empty, no RERR is generated. Otherwise, RERR_Gen generates a new RERR, and the address of each UnreachableNode is inserted into an AddrBlock. If any UnreachableNode.Addr entry is associated with a routing prefix (i.e., a prefix length shorter than the maximum length for the address family), then the AddrBlk MUST include prefix lengths; otherwise, if no such entry, the prefix lengths NOT be included. The value for each UnreachableNode's SeqNum (UnreachableNode.SeqNum) MUST be placed in the SeqNum AddrTLV.

Every broken route reported in the RERR MUST have the same Metric Type. If the Metric Type is not 3, then the RERR message MUST contain a Metric Type MsgTLV indicating the Metric Type of the broken route(s).

8.4. Receiving and Handling RERR Messages

When an AODVv2 router (HandlingRtr) receives a RERR message, it uses the information provided to invalidate affected routes. If HandlingRtr has neighbors that are using the affected routes, then HandlingRtr subsequently sends an RERR message to those neighbors. This has the effect of regenerating the RERR information and is counted as another "hop" for purposes of properly modifying <msg-hop-limit> and <msg-hop-count> in the RERR message header.

HandlingRtr examines the incoming RERR to assure that it contains <msg-hop-limit> and at least one UnreachableNode.Address. If the

required information does not exist, the incoming RERR message is disregarded and further processing stopped. Otherwise, for each UnreachableNode.Address, HandlingRtr searches its route table for a route using longest prefix matching. If no such Route is found, processing is complete for that UnreachableNode.Address. Otherwise, HandlingRtr verifies the following:

1. The UnreachableNode.Address is a routable unicast address.
2. Route.NextHopAddress is the same as RERR IP.SourceAddress.
3. Route.NextHopInterface is the same as the interface on which the RERR was received.
4. The UnreachableNode.SeqNum is unknown, OR Route.SeqNum <= UnreachableNode.SeqNum (using signed 16-bit arithmetic).

If the Route satisfies all of the above conditions, HandlingRtr checks whether Route.PrefixLength is the same as the prefix length for UnreachableNode.Address. If so, HandlingRtr simply sets the state for that Route to be Broken. Otherwise, HandlingRtr creates a new route (call it BrokenRoute) with the same PrefixLength as the prefix length for UnreachableNode.Address, and sets Route.State == Broken for BrokenRoute. If the prefix length for the new route is shorter than Route.PrefixLength, then Route MUST be expunged from the route table (since it is a subroute of the larger route which is reported to be broken). Furthermore, if <msg-hop-limit> is greater than 0, then HandlingRtr adds the UnreachableNode address and TLV information to an AddrBlk for delivery in the outgoing RERR message.

If there are no UnreachableNode addresses to be transmitted in an RERR to upstream routers, HandlingRtr MUST discard the RERR, and no further action is taken.

Otherwise, <msg-hop-limit> is decremented by one (1) and processing continues as follows:

- (Optional) If precursor lists are maintained, the outgoing RERR SHOULD be sent to the active precursors of the broken route as specified in Section 13.3.
- Otherwise, if the incoming RERR message was received at the LL-MANET-Routers [RFC5498] multicast address, the outgoing RERR SHOULD also be sent to LL-MANET-Routers.
- Otherwise, if the PktSource Message TLV is present, and HandlingRtr has a Route to PktSource.Addr, then HandlingRtr MUST send the outgoing RERR to Route[PktSource.Addr].NextHop.
- Otherwise, the outgoing RERR MUST be sent to LL-MANET-Routers.

9. Unknown Message and TLV Types

For handling of messages that contain unknown TLV types, ignore the information for processing, but preserve it unmodified for forwarding.

10. Simple Internet Attachment

Simple Internet attachment means attachment of a stub (i.e., non-transit) network of AODVv2 routers to the Internet via a single Internet AODVv2 router (called IAR).

As in any Internet-attached network, AODVv2 routers, and their clients, wishing to be reachable from hosts on the Internet MUST have IP addresses within the IAR's routable and topologically correct prefix (e.g. 191.0.2.0/24).

```
     /-------------------------\
    / +----------------+        \
   /  |  AODVv2 Router |         \
   |  |  191.0.2.2/32  |         |
   |  +----------------+         |            Routable
   |                       +-----+--------+   Prefix
   |                       |   Internet   |  /191.0.2/24
   |                       | AODVv2 Router| /
   |                       |  191.0.2.1   |/      /---------------\
   |                       | serving net  +------+    Internet     \
   |                       |  191.0.2/24  |      \                 /
   |                       +-----+--------+       \---------------/
   |         +----------------+  |
   |         |  AODVv2 Router |  |
   |         |  191.0.2.3/32  |  |
   \         +----------------+  /
    \                           /
     \-------------------------/
```

Figure 3: Simple Internet Attachment Example

When an AODVv2 router within the AODVv2 MANET wants to discover a route toward a node on the Internet, it uses the normal AODVv2 route discovery for that IP Destination Address. The IAR MUST respond to RREQ on behalf of all Internet destinations.

When a packet from a node on the Internet destined for a node in the AODVv2 MANET reaches the IAR, if the IAR does not have a route toward that destination it will perform normal AODVv2 route discovery for that destination.

11. Multiple Interfaces

AODVv2 MAY be used with multiple interfaces; therefore, the particular interface over which packets arrive MUST be known whenever a packet is received. Whenever a new route is created, the interface through which the route's destination can be reached is also recorded in the route table entry.

When multiple interfaces are available, a node transmitting a multicast packet to LL-MANET-Routers MUST send the packet on all interfaces that have been configured for AODVv2 operation.

Similarly, AODVv2 routers MUST subscribe to LL-MANET-Routers on all their AODVv2 interfaces.

12. AODVv2 Control Message Generation Limits

To avoid congestion, each AODVv2 router's rate of packet/message generation SHOULD be limited. The rate and algorithm for limiting messages (CONTROL_TRAFFIC_LIMITS) is left to the implementor and should be administratively configurable. AODVv2 messages SHOULD be discarded in the following order of preference: RREQ, RREP, and finally RERR.

13. Optional Features

Some optional features of AODVv2, associated with AODV, are not required by minimal implementations. These features are expected to apply in networks with greater mobility, or larger node populations, or requiring reduced latency for application launches. The optional features are as follows:

- Expanding Rings Multicast
- Intermediate RREPs (iRREPs): Without iRREP, only the destination can respond to a RREQ.
- Precursor lists.
- Reporting Multiple Unreachable Nodes. An RERR message can carry more than one Unreachable Destination node for cases when a single link breakage causes multiple destinations to become unreachable from an intermediate router.
- RREP_ACK.
- Message Aggregation.

13.1. Expanding Rings Multicast

For multicast RREQ, <msg-hop-limit> MAY be set in accordance with an expanding ring search as described in [RFC3561] to limit the RREQ

propagation to a subset of the local network and possibly reduce route discovery overhead.

13.2. Intermediate RREP

This specification has been published as a separate Internet Draft [I-D.perkins-irrep].

13.3. Precursor Lists and Notifications

This section specifies an interoperable enhancement to AODVv2 (and possibly other reactive routing protocols) enabling more economical notifications to traffic sources upon determination that a route needed to forward such traffic to its destination has become Broken.

13.3.1. Overview

In many circumstances, there can be several sources of traffic for a certain destination. Each such source of traffic is known as a "precursor" for the destination, as well as all upstream routers between the forwarding AODVv2 router and the traffic source. For each active destination, an AODVv2 router MAY choose to keep track of the upstream neighbors that have provided traffic for that destination; there is no need to keep track of upstream routers any farther away than the next hop.

Moreover, any particular link to an adjacent AODVv2 router may be a path component of multiple routes towards various destinations. The precursors for all destinations using the next hop across any link are collectively known as the precursors for that next hop.

When an AODVv2 router determines that an active link to one of its neighbors has broken, the AODVv2 router detecting the broken link must mark multiple routes as Broken, for each of the newly unreachable destinations, as described in Section 8.3. Each route that relies on the newly broken link is no longer valid. Furthermore, the precursors of the broken link should be notified (using RERR) about the change in status of their route to a destination relying upon the broken next hop.

13.3.2. Precursor Notification Details

During normal operation, each AODVv2 router wishing to maintain precursor lists as described above, maintains a precursor table and updates the table whenever the node forwards traffic to one of the destinations in its route table. For each precursor in the precursor list, a record must be maintained to indicate whether the precursor has been used for recent traffic (in other words, whether the

precursor is an Active precursor). So, when traffic arrives from a precursor, the Current_Time is used to mark the time of last use for the precursor list element associated with that precursor.

When an AODVv2 router detects that a link is broken, then for each precursor using that next hop, the node MAY notify the precursor using either unicast or multicast RERR:

unicast RERR to each Active precursor
  This option is applicable when there are few Active precursors compared to the number of neighboring AODVv2 routers.

multicast RERR to RERR_PRECURSORS
  RERR_PRECURSORS is, by default, LL-MANET-Routers [RFC5498]. This option is typically preferable when there are many precursors, since fewer packet transmissions are required.

Each active upstream neighbor (i.e., precursor) MAY then execute the same procedure until all active upstream routers have received the RERR notification.

13.4. Multicast RREP Response to RREQ

The RREQ Target Router (RREP_Gen) MAY, as an alternative to unicasting a RREP, be configured to distribute routing information about the route toward the RREQ TargNode (RREP_Gen's client) more widely. That is, RREP_Gen MAY be configured respond to a route discovery by generating a RREP, using the procedure in Section 7.4, but multicasting the RREP to LL-MANET-Routers [RFC5498] (subject to similar suppression algorithm for redundant RREP multicasts as described in Section 7.6). The redundant message suppression must occur at every router handling the multicast RREP. Afterwards, RREP_Gen processing for the incoming RREQ is complete.

Broadcast RREP response to incoming RREQ was originally specified to handle unidirectional links, but it is expensive. Due to the significant overhead, AODVv2 routers MUST NOT use multicast RREP unless configured to do so by setting the administrative parameter USE_MULTICAST_RREP.

13.5. RREP_ACK

Instead of relying on existing mechanisms for requesting verification of link bidirectionality during Route Discovery, RREP_Ack is provided as an optional feature and modeled on the RREP_Ack message type from AODV [RFC3561].

Since the RREP_ACK is simply echoed back to the node from which the RREP was received, there is no need for any additional RFC 5444

address information (or TLVs). Considerations of packet TTL are as specified in Section 5.4. An example message format is illustrated in section Appendix B.4.

13.6. Message Aggregation

The aggregation of multiple messages into a packet is specified in RFC 5444 [RFC5444].

Implementations MAY choose to briefly delay transmission of messages for the purpose of aggregation (into a single packet) or to improve performance by using jitter [RFC5148].

14. Administratively Configurable Parameters and Timer Values

AODVv2 uses various configurable parameters of various types:

- Timers
- Protocol constants
- Administrative (functional) controls
- Other administrative parameters and lists

The tables in the following sections show the parameters along their definitions and default values (if any).

Note: several fields have limited size (bits or bytes). These sizes and their encoding may place specific limitations on the values that can be set. For example, <msg-hop-count> is a 8-bit field and therefore MAX_HOPCOUNT cannot be larger than 255.

14.1. Timers

AODVv2 requires certain timing information to be associated with route table entries. The default values are as follows, subject to future experience:

Name	Default Value
ACTIVE_INTERVAL	5 second
MAX_IDLETIME	200 seconds
MAX_BLACKLIST_TIME	200 seconds
MAX_SEQNUM_LIFETIME	300 seconds
RREQ_WAIT_TIME	2 seconds
UNICAST_MESSAGE_SENT_TIMEOUT	1 second
RREQ_HOLDDOWN_TIME	10 seconds

Table 2: Timing Parameter Values

The above timing parameter values have worked well for small and medium well-connected networks with moderate topology changes.

The timing parameters SHOULD be administratively configurable for the network where AODVv2 is used. Ideally, for networks with frequent topology changes the AODVv2 parameters should be adjusted using either experimentally determined values or dynamic adaptation. For example, in networks with infrequent topology changes MAX_IDLETIME may be set to a much larger value.

14.2. Protocol constants

AODVv2 protocol constants typically do not require changes. The following table lists these constants, along with their values and a reference to the specification describing their use.

Name	Default Value	Description
DISCOVERY_ATTEMPTS_MAX	3	Section 7.1
MAX_HOPCOUNT	20 hops	Section 5.6
MAX_METRIC[i]	Specified only for HopCount	Section 5.6
MAXTIME	[TBD]	Maximum expressible clock time

Table 3: Parameter Values

14.3. Administrative (functional) controls

The following administrative controls may be used to change the operation of the network, by enabling optional behaviors. These options are not required for correct routing behavior, although they

may potentially reduce AODVv2 protocol messaging in certain situations. The default behavior is to NOT enable most such options, options. Packet buffering is enabled by default.

Name	Description
DEFAULT_METRIC_TYPE	3 (i.e, Hop Count (see [RFC6551]))
ENABLE_IDLE_IN_RERR	Section 8.3.2
ENABLE_IRREP	Section 7.3
USE_MULTICAST_RREP	Section 13.4

Table 4: Administratively Configured Controls

14.4. Other administrative parameters and lists

The following table lists contains AODVv2 parameters which should be administratively configured for each specific network.

Name	Default Value	Cross Reference
AODVv2_INTERFACES		Section 4
BUFFER_SIZE_PACKETS	2	Section 7.1
BUFFER_SIZE_BYTES	MAX_PACKET_SIZE [TBD]	Section 7.1
CLIENT_ADDRESSES	AODVv2_INTERFACES	Section 5.3
CONTROL_TRAFFIC_LIMIT	TBD [50 packets/sec?]	Section 12

Table 5: Other Administrative Parameters

15. IANA Considerations

This section specifies several message types, message tlv-types, and address tlv-types. Also, a new registry of 16-bit alternate metric types is specified.

15.1. AODVv2 Message Types Specification

Name	Type (TBD)
Route Request (RREQ)	10
Route Reply (RREP)	11
Route Error (RERR)	12
Route Reply Acknowledgement (RREP_ACK)	13

Table 6: AODVv2 Message Types

15.2. Message TLV Type Specification

Name	Type (TBD)	Length in octets	Cross Reference
Acknowledgment Request (AckReq)	10	0	Section 5.2
Packet Source (PktSource)	11	4 or 16	Section 8.3
Metric Type	12	1	Section 7.2

Table 7: Message TLV Types

15.3. Address Block TLV Specification

Name	Type (TBD)	Length	Value
Metric	10	depends on Metric Type	Section 7.2
Sequence Number (SeqNum)	11	2 octets	Section 7.2
Originating Node Sequence Number (OrigSeqNum)	12	2 octets	Section 7.2
Target Node Sequence Number (TargSeqNum)	13	2 octets	Section 7.2
VALIDITY_TIME	1	1 octet	[RFC5497]

Table 8: Address Block TLV (AddrTLV) Types

15.4. Metric Type Number Allocation

Metric types are identified according to the assignments as specified in [RFC6551]. The metric type of the Hop Count metric is assigned to be 3, in order to maintain compatibility with that existing table of

values from RFC 6551. Non-addititve metrics are not supported in this draft.

Name	Type	Metric Size
Unallocated	0 -- 2	TBD
Hop Count	3 - TBD	1 octet
Unallocated	4 -- 254	TBD
Reserved	255	Undefined

Table 9: Metric Types

16. Security Considerations

The objective of the AODVv2 protocol is for each router to communicate reachability information about addresses for which it is responsible. Positive routing information (i.e. a route exists) is distributed via RREQ and RREP messages. Negative routing information (i.e. a route does not exist) is distributed via RERRs. AODVv2 routers store the information contained in these messages in order to properly forward data packets, and they generally provide this information to other AODVv2 routers.

This section does not mandate any specific security measures. Instead, this section describes various security considerations and potential avenues to secure AODVv2 routing.

The most important security mechanisms for AODVv2 routing are integrity/authentication and confidentiality.

In situations where routing information or router identity are suspect, integrity and authentication techniques SHOULD be applied to AODVv2 messages. In these situations, routing information that is distributed over multiple hops SHOULD also verify the integrity and identity of information based on originator of the routing information.

A digital signature could be used to identify the source of AODVv2 messages and information, along with its authenticity. A nonce or timestamp SHOULD also be used to protect against replay attacks. S/MIME and OpenPGP are two authentication/integrity protocols that could be adapted for this purpose.

In situations where confidentiality of AODVv2 messages is important, cryptographic techniques can be applied.

In certain situations, for example sending a RREP or RERR, an AODVv2 router could include proof that it has previously received valid routing information to reach the destination, at one point of time in the past. In situations where routers are suspected of transmitting maliciously erroneous information, the original routing information along with its security credentials SHOULD be included.

Note that if multicast is used, any confidentiality and integrity algorithms used MUST permit multiple receivers to handle the message.

Routing protocols, however, are prime targets for impersonation attacks. In networks where the node membership is not known, it is difficult to determine the occurrence of impersonation attacks, and security prevention techniques are difficult at best. However, when the network membership is known and there is a danger of such attacks, AODVv2 messages must be protected by the use of authentication techniques, such as those involving generation of unforgeable and cryptographically strong message digests or digital signatures. While AODVv2 does not place restrictions on the authentication mechanism used for this purpose, IPsec Authentication Message (AH) is an appropriate choice for cases where the nodes share an appropriate security association that enables the use of AH.

In particular, routing messages SHOULD be authenticated to avoid creation of spurious routes to a destination. Otherwise, an attacker could masquerade as that destination and maliciously deny service to the destination and/or maliciously inspect and consume traffic intended for delivery to the destination. RERR messages SHOULD be authenticated in order to prevent malicious nodes from disrupting active routes between communicating nodes.

If the mobile nodes in the ad hoc network have pre-established security associations, the purposes for which the security associations are created should include that of authorizing the processing of AODVv2 control packets. Given this understanding, the mobile nodes should be able to use the same authentication mechanisms based on their IP addresses as they would have used otherwise.

If the mobile nodes in the ad hoc network have pre-established security associations, the purposes for which the security associations Most AODVv2 messages are transmitted to the multicast address LL-MANET-Routers [RFC5498]. It is therefore required for security that AODVv2 neighbors exchange security information that can be used to insert an ICV [RFC6621] into the AODVv2 message block [RFC5444]. This enables hop-by-hop security. For destination-only RREP discovery procedures, AODVv2 routers that share a security association SHOULD use the appropriate mechanisms as specified in RFC

6621. The establishment of these security associations is out of scope for this document.

17. Acknowledgments

AODVv2 is a descendant of the design of previous MANET on-demand protocols, especially AODV [RFC3561] and DSR [RFC4728]. Changes to previous MANET on-demand protocols stem from research and implementation experiences. Thanks to Elizabeth Belding-Royer for her long time authorship of AODV. Additional thanks to Derek Atkins, Emmanuel Baccelli, Abdussalam Baryun, Ramon Caceres, Thomas Clausen, Christopher Dearlove, Ulrich Herberg, Henner Jakob, Luke Klein-Berndt, Lars Kristensen, Tronje Krop, Koojana Kuladinithi, Kedar Namjoshi, Alexandru Petrescu, Henning Rogge, Fransisco Ros, Pedro Ruiz, Christoph Sommer, Lotte Steenbrink, Romain Thouvenin, Richard Trefler, Jiazi Yi, Seung Yi, and Cong Yuan, for their reviews AODVv2 and DYMO, as well as numerous specification suggestions.

18. References

18.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC5082] Gill, V., Heasley, J., Meyer, D., Savola, P., and C. Pignataro, "The Generalized TTL Security Mechanism (GTSM)", RFC 5082, October 2007.

[RFC5444] Clausen, T., Dearlove, C., Dean, J., and C. Adjih, "Generalized Mobile Ad Hoc Network (MANET) Packet/Message Format", RFC 5444, February 2009.

[RFC5497] Clausen, T. and C. Dearlove, "Representing Multi-Value Time in Mobile Ad Hoc Networks (MANETs)", RFC 5497, March 2009.

[RFC5498] Chakeres, I., "IANA Allocations for Mobile Ad Hoc Network (MANET) Protocols", RFC 5498, March 2009.

[RFC6551] Vasseur, JP., Kim, M., Pister, K., Dejean, N., and D. Barthel, "Routing Metrics Used for Path Calculation in Low-Power and Lossy Networks", RFC 6551, March 2012.

18.2. Informative References

[I-D.perkins-irrep]
Perkins, C. and I. Chakeres, "Intermediate RREP for dynamic MANET On-demand (AODVv2) Routing", draft-perkins-irrep-02 (work in progress), November 2012.

[Perkins94]
Perkins, C. and P. Bhagwat, "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers", Proceedings of the ACM SIGCOMM '94 Conference on Communications Architectures, Protocols and Applications, London, UK, pp. 234-244, August 1994.

[Perkins99]
Perkins, C. and E. Royer, "Ad hoc On-Demand Distance Vector (AODV) Routing", Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications, New Orleans, LA, pp. 90-100, February 1999.

[RFC2501] Corson, M. and J. Macker, "Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations", RFC 2501, January 1999.

[RFC3561] Perkins, C., Belding-Royer, E., and S. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing", RFC 3561, July 2003.

[RFC4193] Hinden, R. and B. Haberman, "Unique Local IPv6 Unicast Addresses", RFC 4193, October 2005.

[RFC4728] Johnson, D., Hu, Y., and D. Maltz, "The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4", RFC 4728, February 2007.

[RFC4861] Narten, T., Nordmark, E., Simpson, W., and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, September 2007.

[RFC5148] Clausen, T., Dearlove, C., and B. Adamson, "Jitter Considerations in Mobile Ad Hoc Networks (MANETs)", RFC 5148, February 2008.

[RFC6130] Clausen, T., Dearlove, C., and J. Dean, "Mobile Ad Hoc Network (MANET) Neighborhood Discovery Protocol (NHDP)", RFC 6130, April 2011.

[RFC6621] Macker, J., "Simplified Multicast Forwarding", RFC 6621, May 2012.

Appendix A. Example Algorithms for AODVv2 Protocol Operations

The following subsections show example algorithms for protocol operations required by AODVv2, including RREQ, RREP, RERR, and RREP-ACK.

Processing for RREQ, RREP, and RERR messages follows the following general outline:

1. Receive incoming message.
2. Update route table as appropriate.
3. Respond as needed, often regenerating the incoming message with updated information.

Once the route table has been updated, the information contained there is known to be the most recent available information for any fields in the outgoing message. For this reason, the algorithms are written as if outgoing message field values are assigned from the route table information, even though it is often equally appropriate to use fields from the incoming message.

AODVv2_algorithms:

- o Process_Routing_Info
- o Generate_RREQ
- o Receive_RREQ
- o Regenerate_RREQ
- o Generate_RREP
- o Receive_RREP
- o Regenerate_RREP
- o Generate_RERR
- o Receive_RERR
- o Regenerate_RERR
- o Generate_RREP_Ack
- o Consume_RREP_Ack()
- o Timeout RREP_Ack()

The following lists indicate the meaning of the field names used in subsequent sections to describe message processing for the above algorithms.

Incoming RREQ message parameters:

```
inRREQ.origIP := originator IP address
inRREQ.origSeq := originator IP sequence #
```

```
inRREQ.metType := metric type
inRREQ.origMet := metric to originator
inRREQ.targIP := target IP address
inRREQ.targSeq := target sequence # (if known)
inRREQ.hopLim := msg-hop-limit /* from RFC 5444 header */
inRREQ.nbrIP := IP address of the neighbor that sent the RREQ
```

Outgoing RREQ message parameters:

```
outRREQ.origIP := originator IP address
outRREQ.origSeq := originator IP sequence #
outRREQ.metType := metric type
outRREQ.origMet := metric to origNode {initially
MIN_METRIC[MetType]}
outRREQ.targIP := target IP address
outRREQ.targSeq := target sequence # (if known)
outRREQ.hopLim /* initially MAX_HOPCOUNT at originator */
```

Incoming RREP message parameters:

```
inRREP.hoplim /* msg-hop-limit from RFC 5444 header */
inRREP.origIP := originator's IP address
inRREP.metType := metric type
inRREP.targIP := target IP address
inRREP.targSeq := target sequence #
inRREP.targMet := target's metric {initially MIN_METRIC[MetType]}
inRREP.PfxLen
```

Outgoing RREP message parameters:

```
outRREP.origIP := originator's IP address
outRREP.metType := metric type
outRREP.targIP := target IP address
outRREP.targSeq := target sequence #
outRREP.targMet := target's metric {starting with zero}
outRREP.PfxLen
outRREP.hopLim /* initially MAX_HOPCOUNT at originator */
```

Incoming RERR message parameters:

```
inRERR.PktSrc := source IP of unforwardable packet (if present)
inRERR.metType := metric type for routes to unreachable
destinations
inRERR.PfxLen[] := prefix lengths for unreachable destinations
inRERR.LostDest[] := unreachable destinations
inRERR.LostSeq[] := sequence #s for unreachable destinations
```

Outgoing RERR message parameters:

```
outRERR.PktSrc := source IP of unforwardable packet (if present)
outRERR.metType := metric type for routes to unreachable
destinations
outRERR.PfxLen[] := prefix lengths for unreachable destinations
outRERR.LostDest[] := unreachable destinations
outRERR.LostSeq[] := sequence #s for unreachable destinations
```

A.1. Subroutines for AODVv2 Protocol Operations

```
/* Compare incoming route information to current route, maybe use */
Process_Routing_Info (dest, seq#, metric_type, metric,
                                                    last_hop_metric)
  /* last_hop_metric: either Cost(inRREQ.netif) or (inRREP.netif) */
{
new_metric := metric + last_hop_metric;
rte := Fetch_Route_Table_Entry (dest, seq#, metric_type);
if (NULL == rte) {
    rte := Create_Route_Table_Entry
            (dest, seq#, metric_type, new_metric);
} else if (seq# > rte.seq#) { /* stale rte route entry */
    Update_Route_Table_Entry (rte, seq#, metric_type, new_metric);
} else if (seq# < rte.seq#) { /* stale incoming route infor */
    return(NULL);
} else if (rte.state == broken) {  /* when (seq# == rte.seq#) */
    Update_Route_Table_Entry (rte, seq#, metric_type, new_metric);
} else if (rte.metric > (new_metric) { /* and (seq# == rte.seq#) */
    Update_Route_Table_Entry (rte, seq#, metric_type, new_metric);
} else {    /* incoming route information is not useful */
    return(NULL);
}
return (rte);
}
```

A.2. Example Algorithms for AODVv2 RREQ Operations

A.2.1. Generate_RREQ

```
Generate_RREQ {
/* Marshall parameters */
outRREQ.origIP := IP address used by application
outRREQ.origSeq := originating router's sequence #
outRREQ.metType := (if included) metric type needed by application
outRREQ.origMet := 0 (default) or MIN_METRIC(Metric_type)
outRREQ.targIP := target IP address
outRREQ.targSeq := target sequence # /* if known from route table */
outRREQ.hopLim := msg-hop-limit     /* RFC 5444 */

/* build RFC 5444 message header fields */
{
    msg-type=RREQ (message is of type RREQ)
    MF=4 (Message Flags = 4 [only msg-hop-limit field is present])
    MAL=3 or 15 (Message Address Length [3 for IPv4, 15 for IPv6])
    msg-size=NN (octets -- counting MsgHdr, AddrBlk, and AddrTLVs)
    msg-hop-limit := MAX_HOPCOUNT
    if (Metric_type == DEFAULT) {
        msg.tlvs-length=0
    } else { /* Metric_type != HopCount */
        /* Build Metric_type Message TLV */
    }
}

/* build AddrBlk */
num-addr := 2
AddrBlk := {outRREQ.origIP and outRREQ.targIP addresses}

/* Include each available Sequence Number in appropriate AddrTLV */
    /* put outRREQ.origSeq in OrigSeqNum AddrTLV */
    if (NULL != targSeq) {
        /* put outRREQ.targSeq in TargSeqNum AddrTLV */
    }

/* Build Metric AddrTLV containing OrigNode metric */
    /* use MIN_METRIC(metric type) [==0 for default metric type */
}
```

A.2.2. Receive_RREQ

```
Receive_RREQ (inRREQ) {
/* Extract inRREQ values */
origRTE = Process_Routing_Info (inRREQ.origIP, inRREQ.origSeq, ...)
if (inRREQ.targIP belongs to me or my client subnet) {
    Generate_RREP()
} else if (inRREQ present in RREQ_table) {
    return;     /* don't regenerate RREQ... */
} else if (inRREQ.nbrIP not present in blacklist) {
    Regenerate_RREQ(origRTE, inRREQ)
} else if (blacklist_expiration_time > current_time) {
    return;     /* don't regenerate RREQ... */
} else {
    Remove nbrIP from blacklist;
    Regenerate_RREQ(origRTE, inRREQ)
}
}
```

A.2.3. Regenerate_RREQ

```
Regenerate_RREQ (origRTE, inRREQ) { /* called from receive_RREQ() */
outRREQ.hopLim := inRREQ.hopLim - 1
if (outRREQ.hopLim == 0) {  /* don't regenerate */
    return()
}
/* Marshall parameters */
outRREQ.origIP := origRTE.origIP
outRREQ.origSeq := origRTE.origSeq
outRREQ.origMet := origRTE.origMet
outRREQ.metType := origRTE.metType
outRREQ.targIP := inRREQ.targIP
outRREQ.targSeq := inRREQ.targSeq   /* if present */

/* build RFC 5444 message header fields */
{
    msg-type=RREQ (message is of type RREQ)
    MF=4 (Message Flags = 4 [only msg-hop-limit field is present])
    MAL=3 or 15 (Message Address Length [3 for IPv4, 15 for IPv6])
    msg-size=NN (octets -- counting MsgHdr, AddrBlk, and AddrTLVs)
    msg-hop-limit := MAX_METRIC(Metric Type) (default, MAX_HOPCOUNT)
    if (Metric_type == DEFAULT) {
        msg.tlvs-length=0
    } else { /* Metric_type != HopCount */
        /* Build Metric_type Message TLV */
    }
}

/* build AddrBlk */
num-addr := 2
AddrBlk := {outRREQ.origIP and outRREQ.targIP addresses}

/* Include each available Sequence Number in its proper AddrTLV */
/* put outRREQ.origSeq in OrigSeqNum AddrTLV */
if (NULL != targSeq) {
    /* put outRREQ.targSeq in TargSeqNum AddrTLV */
}

/* Build Metric AddrTLV to contain outRREQ.origMet */

}
```

A.3. Example Algorithms for AODVv2 RREP Operations

A.3.1. Generate_RREP

```
Generate_RREP {
/* Marshall parameters */
outRREP.origIP := origRTE.origIP
metric_type := origRTE.metType  /* if not default */
if (DEFAULT != metric_type)
    outRREP.metType := metric_type
outRREP.targIP := inRREQ.targIP
outRREP.targMet := MIN_METRIC(outRREP.metType) (0  by default)
my_sequence_# := (1 + my_sequence_#) /* from nonvolatile storage */
outRREP.targSeq := my_sequence_#

/* build RFC 5444 message header fields */
{
    msg-type=RREP
    MF=4 (Message Flags = 4 [only msg-hop-limit field is present])
    MAL=3 or 15 (Message Address Length [3 for IPv4, 15 for IPv6])
    msg-size=NN (octets -- counting MsgHdr, AddrBlk, and AddrTLVs)
    msg-hop-limit := MAX_HOPCOUNT
    /* Include the AckReq TLV when:
       - previous RREP does not seem to enable any data flow, OR
       - when RREQ is received from same OrigNode after RREP was
         unicast to targRTE.nextHop
     */
    if (DEFAULT != metric_type) {
        msg.tlvs-length=0
    } else { /* Metric_type != HopCount */
        /* Build Metric_type Message TLV */
    }
}

/* build AddrBlk */
num-addr := 2
AddrBlk := {outRREQ.origIP and outRREQ.targIP addresses}

/* put outRREP.TargSeq in TargSeqNum AddrTLV */

/* Build Metric AddrTLV containing TargNode metric */
/* use MIN_METRIC(origRTE.metType) */
}
```

A.3.2. Receive_RREP

```
Receive_RREP (inRREP)
{
If (RREP includes AckReq TLV) {
    Generate_RREP_Ack()
}
/* Extract inRREP values */
targRTE := Process_Routing_Info (inRREP.targIP, inRREP.targSeq, ...)
if (inRREP.targIP belongs to me, a client, or a client subnet) {
    Consume_RREP(inRREP)
} else {
    Regenerate_RREP(targRTE, inRREP)
}
}
```

A.3.3. Regenerate_RREP

```
Regenerate_RREP(targRTE, inRREP) {
outRREP.hopLim := inRREP.hopLim - 1
if (outRREP.hopLim == 0) {  /* don't regenerate */
    return()
}
/* Marshall parameters */
outRREP.targIP := targRTE.targIP
outRREP.targSeq := targRTE.targSeq
outRREP.targMet := targRTE.targMet
metric_type := origRTE.metType  /* if not default */
if (DEFAULT != metric_type)
    outRREP.metType := metric_type
outRREP.origIP := inRREP.origIP
outRREP.nextHop := targRTE.nextHop

/* build RFC 5444 message header fields */
{
    msg-type=RREP (message is of type RREP)
    MF=4 (Message Flags = 4 [only msg-hop-limit field is present])
    MAL=3 or 15 (Message Address Length [3 for IPv4, 15 for IPv6])
    msg-size=NN (octets -- counting MsgHdr, AddrBlk, and AddrTLVs)
    /* Include the AckReq TLV when:
       - previous RREP does not seem to enable any data flow, OR
       - when RREQ is received from same OrigNode after RREP was
         unicast to targRTE.nextHop
     */
    msg-hop-limit := outRREP.hopLim;
    if (metric_type == DEFAULT) {
        msg.tlvs-length=0
    } else { /* Metric_type != HopCount */
        /* Build Metric_type Message TLV */
    }
}

/* build AddrBlk */
num-addr := 2
AddrBlk := {outRREQ.origIP and outRREQ.targIP addresses}

/* put outRREP.targSeq in TargSeqNum AddrTLV */

/* Build Metric AddrTLV containing TargNode metric */
}
```

A.3.4. Consume_RREP

```
/* executed by RREQ_Gen */
/* TargNode route table entry was updated by Receive_RREP() */
Consume_RREP() {
    /* Transmit buffered packet(s) (if any) to TargNode */
}
```

A.4. Example Algorithms for AODVv2 RERR Operations

A.4.1. Generate_RERR

```
Generate_RERR()
{
metric_type := DEFAULT;
switch (error_type) in {
case (broken_link):
    num-broken-addr=0
    /* find unreachable destinations, seqNums, prefixes */
    for (every rte (route table entry) in route table) {
        if (broken_link == rte.next_hop) {
        rte.state := broken;
        outRERR.LostDest[num-broken-addr] := rte.dest
        outRERR.LostSeq[num-broken-addr] := rte.seq#
        outRERR.PfxLen[num-broken-addr] := rte.pfx
        metric_type := rte.metType
        num-broken-addr := (num-broken-addr+1)
        }
    }
    /* No offending-src for this case */
case (undeliverable packet):
    offending-src := undeliverable_packet.srcIP
    outRERR.LostDest[] := undeliverable_packet.destIP
    outRERR.LostPfxSiz[] := MAX_PFX_SIZE   /* 31 or 127 */
    num-broken-addr=1
}

/* build RFC 5444 message header fields */
{
    msg-type=RERR (message is of type RERR)
    MF=4 (Message Flags = 4 [only msg-hop-limit field is present])
    MAL=3 or 15 (Message Address Length [3 for IPv4, 15 for IPv6])
    msg-size=NN (octets -- counting MsgHdr, AddrBlk, and AddrTLVs)
    msg-hop-limit := outRERR.hopLim;
    if (NULL != offending-src) {
```

```
            /* Build PktSource Message TLV */
        }
        if (metric_type != DEFAULT) { /* Metric_type != HopCount */
            /* Build Metric_type Message TLV */
        }
    }

    /* build AddrBlk */
    num-addr := num-broken-addr;
    AddrBlk := outRERR.LostDest[];

    /* Add AddrBlk Seq# TLV */
    Seq#TLV := outRERR.LostSeq[]

    /* only add AddrBlk PfxSiz TLV if prefixes are nondefault */
    for (pfx in outRERR.LostPfx[]) {
        if (pfx != Max_Prefix_Size) { /* 31 for IPv4, 127 for IPv6 */
            PfxSizTLV := outRERR.LostPfx[]
            return;
        }
    }
    }
```

A.4.2. Receive_RERR

```
Receive_RERR (inERR)
{
/* Extract inERR values */
next_hop := inRERR.nbrIP
offending-src := inRERR.offending-src;  /* NULL if not present */

precursors[] := NULL;
num-broken-addr := 0;
in-broken-addr := 0;
for (IPaddr := inRERR.LostDest[in-broken-addr]) {
    rte := Fetch_Route_Table_Entry (dest, metric_type);
    if (NULL == rte) {
        continue;
    } else if (rte.nextHop != inRERR.fromIP) {
        continue;
    } else if (NULL != rte.precursors) {
        /* add rte.precursors to precursors */
    } else if (rte.PfxSiz < inRERR.PfxSiz) {
    /***********************************************************
     If the reported prefix from the incoming RERR is *longer*
       than the prefix from Route Table, then create a new route
       with the longer prefix.
       The newly created route will be marked as broken, and used
       to regenerate RERR, NOT using shorter the routing prefix.
     This avoids unnecessarily invalidating the larger subnet.
     ***********************************************************/
        rte := Create_Route_Table_Entry (IPaddr, seq#,
            metric_type, new_metric, inRERR.PfxSiz);
    }
    LostDest[num-broken-addr] := rte.Dest;
    Seq#[num-broken-addr] := rte.Seq#;
    PfxSiz[num-broken-addr] := rte.PfxSiz;
    rte.state = broken;
    num-broken-addr := (num-broken-addr + 1);
    in-broken-addr := (in-broken-addr + 1);
}
if (num-broken-addr > 0) {
    Regenerate_RERR (offending-src, precursors,
        LostDest[], Seq#[], PfxSiz[])
}
}
```

A.4.3. Regenerate_RERR

```
Regenerate_RERR (offending-src, precursors,
    LostDest[], LostSeq#[], PfxSiz[])
{
/* build RFC 5444 message header fields */
{
    msg-type=RERR (message is of type RERR)
    MF=4 (Message Flags = 4 [only msg-hop-limit field is present])
    MAL=3 or 15 (Message Address Length [3 for IPv4, 15 for IPv6])
    msg-size=NN (octets -- counting MsgHdr, AddrBlk, and AddrTLVs)
    outRERR.hopLim := inRERR.hopLim - 1
    msg-hop-limit := outRERR.hopLim;

    if (NULL != offending-src) {
        /* Build PktSource Message TLV */
    }
    if (metric_type != DEFAULT) { /* Metric_type != HopCount */
        /* Build Metric_type Message TLV */
    }
}

/* build AddrBlk */
num-addr := num-broken-addr;
AddrBlk := LostDest[];

/* Add AddrBlk Seq# TLV */
Seq#TLV := LostSeq[]

/* only add AddrBlk PfxSiz TLV if prefixes are nondefault */
for (pfx in PfxSiz[]) {
    if (pfx != Max_Prefix_Size) { /* 31 for IPv4, 127 for IPv6 */
        PfxSizTLV := PfxSiz[]
    }
}   /* If all are default, don't include PfxSize AddrTLV */

if (#precursors == 1) {
    unicast RERR to precursor[0];
} else if (#precursors > 1) {
    multicast RERR to RERR_PRECURSORS;
} else if (offending-src != NULL) {
    unicast RERR to offending-src;
} else {
    multicast RERR to RERR_PRECURSORS;
}
}
```

A.5. Example Algorithms for AODVv2 RREP-Ack Operations

A.5.1. Generate_RREP_Ack

```
/* To be sent when RREP includes the AckReq TLV */
Generate_RREP_Ack()
{
/* assign RFC 5444 fields */
msgtype := RREPAck
MF := 0
MAL := 3
msg-size := 4
}
```

A.5.2. Consume_RREP_Ack

```
Consume_RREP_Ack()
{
/* turn off timeout event for the node sending RREP_Ack */
}
```

A.5.3. Timeout_RREP_Ack

```
Timeout_RREP_Ack()
{
/* insert unresponsive node into blacklist */
}
```

Appendix B. Example RFC 5444-compliant packet formats

The following subsections show example RFC 5444-compliant packets for AODVv2 message types RREQ, RREP, RERR, and RREP-Ack. These proposed message formats are designed based on expected savings from IPv6 addressable MANET nodes, and a layout for the Address TLVs that may be viewed as natural, even if perhaps not the absolute most compact possible encoding.

For RteMsgs, the msg-hdr fields are followed by at least one and optionally two Address Blocks. The first AddrBlk contains OrigNode and TargNode. For each AddrBlk, there must be AddrTLVs of type Metric and one of the SeqNum types (i.e, OrigSeqNum, TargSeqNum, or Seqnum).

There is no Metric Type Message TLV present, so the Metric AddrTLV measures HopCount. The Metric AddrTLV also provides a way for the AODV router generating the RREQ or RREP to supply an initial nonzero cost for the route to its client node (OrigNode or TargNode, for RREQ or RREP respectively).

In all cases, the length of an address (32 bits for IPv4 and 128 bits for IPv6) inside an AODVv2 message is indicated by the msg-addr-length (MAL) in the msg-header, as specified in [RFC5444].

The RFC 5444 header preceding AODVv2 messages in this document has the format illustrated in Figure 4.

```
 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
+-+-+-+-+-+-+-+-+-+
| PV=0 |  PF=0  |
+-+-+-+-+-+-+-+-+-+
```

Figure 4: RFC 5444 Packet Header

The fields in Figure 4 are to be interpreted as follows:

- o PV=0 (Packet Header Version = 0)
- o PF=0 (Packet Flags = 0)

B.1. RREQ Message Format

Figure 5 illustrates an example RREQ message format.

```
       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
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      | msg-type=RREQ | MF=4  | MAL=3 |          msg-size=28          |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      | msg-hop-limit |      msg.tlvs-length=0        |   num-addr=2  |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |1|0|0|0|0| Rsv | head-length=3 | Head (bytes for Orig & Target):
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      :Head(Orig&Targ)|   Orig.Tail   |  Target.Tail  |addr.TLV.len=11:
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      :addr.TLV.len=11|type=OrigSeqNum|0|1|0|1|0|0|Rsv| Index-start=0 |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      | tlv-length=2  |     Orig.Node Sequence #      |  type=Metric  |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |0|1|0|1|0|0|Rsv| Index-start=0 | tlv-length=1  | OrigNodeHopCt |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 5: Example IPv4 RREQ, with OrigSeqNum and Metric AddrTLVs

The fields in Figure 5 are to be interpreted as follows:

- msg-type=RREQ (first [and only] message is of type RREQ)
- MF=4 (Message Flags = 4 [only msg-hop-limit field is present])
- MAL=3 (Message Address Length indicator [3 for IPv4, 15 for IPv6])
- msg-size=28 (octets -- counting MsgHdr, MsgTLVs, and AddrBlks)
- msg-hop-limit (initially MAX_HOPCOUNT by default)
- msg.tlvs-length=0 (no Message TLVs)
- num-addr=2 (OrigNode and TargNode addresses in RteMsg AddrBlock)
- AddrBlk flags:
    - bit 0 (ahashead): 1
    - bit 1 (ahasfulltail): 0
    - bit 2 (ahaszerotail): 0
    - bit 3 (ahassingleprelen): 0
    - bit 4 (ahasmultiprelen): 0
    - bits 5-7: RESERVED
- head-length=3 (length of head part of each address is 3 octets)
- Head (3 initial bytes for both Originating & Target addresses)
- Orig.Tail (4th byte of Originating Node IP address)
- Target.Tail (4th byte of Target Node IP address)
- addr.TLV.len = 11 (length in bytes for OrigSeqNum and Metric TLVs
- type=OrigSeqNum (type of first AddrBlk TLV, value 2 octets)
- AddrTLV flags for the OrigSeqNum TLV:
    - bit 0 (thastypeext): 0
    - bit 1 (thassingleindex): 1
    - bit 2 (thasmultiindex): 0

* bit 3 (thasvalue): 1
* bit 4 (thasextlen): 0
* bit 5 (tismultivalue): 0
* bits 6-7: RESERVED

- o Index-start=0 (OrigSeqNum TLV value applies at index 0)
- o tlv-length=2 (so there is only one TLV value, [1 = 2/2])
- o Orig.Node Sequence # (TLV value for the OrigSeqNum TLV
- o type=Metric (AddrTLV type of second AddrBlk TLV, values 1 octet)
- o AddrTLV flags for Metric_TLV:

  * bit 0 (thastypeext): 0
  * bit 1 (thassingleindex): 1
  * bit 2 (thasmultiindex): 0
  * bit 3 (thasvalue): 1
  * bit 4 (thasextlen): 0
  * bit 5 (tismultivalue): 0
  * bits 6-7: RESERVED

- o Index-start=0 (Metric TLV values start at index 0)
- o tlv-length=1 (so there is only one TLV value, [1 = 1/1])
- o OrigNodeHopCt (first [and only] TLV value for the Metric TLV)

B.2. RREP Message Format

Figure 6 illustrates a packet format for an example RREP message.

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| msg-type=RREP | MF=4  | MAL=3 |          msg-size=28          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| msg-hop-limit |      msg.tlvs-length=0        |   num-addr=2  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1|0|0|0|0| Rsv | head-length=3 | Head (bytes for Orig & Target):
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:Head(Orig&Targ)|   Orig.Tail   |  Target.Tail  |addr.TLV.len=11:
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:addr.TLV.len=11|type=TargSeqNum|0|1|0|1|0|0|Rsv| Index-start=1 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| tlv-length=2  |     Targ.Node Sequence #      |  type=Metric  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0|1|0|1|0|0|Rsv| Index-start=1 | tlv-length=1  | TargNodeHopCt |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 6: Example IPv4 RREP, with TargSeqNum TLV and 1 Metric

The fields in Figure 6 are to be interpreted as follows:

o msg-type=RREP (first [and only] message is of type RREP)
o MF=4 (Message Flags = 4 [only msg-hop-limit field is present])
o MAL=3 (Message Address Length indicator [3 for IPv4, 15 for IPv6])
o msg-size=28 (octets -- counting MsgHdr, MsgTLVs, and AddrBlks)
o msg-hop-limit (initially MAX_HOPCOUNT by default)
o msg.tlvs-length=0 (no Message TLVs)
o num-addr=2 (OrigNode and TargNode addresses in RteMsg AddrBlock)
o AddrBlk flags:

  * bit 0 (ahashead): 1
  * bit 1 (ahasfulltail): 0
  * bit 2 (ahaszerotail): 0
  * bit 3 (ahassingleprelen): 0
  * bit 4 (ahasmultiprelen): 0
  * bits 5-7: RESERVED
o head-length=3 (length of head part of each address is 3 octets)
o Head (3 initial bytes for both Originating & Target addresses)
o Orig.Tail (4th byte of Originating Node IP address)
o Target.Tail (4th byte of Target Node IP address)
o addr.TLV.len = 11 (length in bytes for TargSeqNum TLV and Metric TLV
o type=TargSeqNum (type of first AddrBlk TLV, value 2 octets)
o AddrTLV flags for the TargSeqNum TLV:

  * bit 0 (thastypeext): 0
  * bit 1 (thassingleindex): 1
  * bit 2 (thasmultiindex): 0
  * bit 3 (thasvalue): 1
  * bit 4 (thasextlen): 0
  * bit 5 (tismultivalue): 0
  * bits 6-7: RESERVED
o Index-start=1 (TargSeqNum TLV value applies to address at index 1)
o tlv-length=2 (there is one TLV value, 2 bytes in length)
o Targ.Node Sequence # (value for the TargSeqNum TLV)
o type=Metric (AddrTLV type of second AddrBlk TLV, value 1 octet)
o AddrTLV flags for the Metric TLV [01010000, same as for TargSeqNum TLV]
o Index-start=1 (Metric TLV values start at index 1)
o tlv-length=1 (there is one TLV value, 1 byte in length)
o TargNodeHopCt (first [and only] TLV value for Metric TLV)

B.3. RERR Message Format

Figure 7 illustrates an example RERR message format.

```
     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
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    | msg-type=RERR | MF=4  | MAL=3 |          msg-size=24          |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    | msg-hop-limit |      msg.tlvs-length=0        |   num-addr=2  |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |1|0|0|0|0| Rsv | head-length=3 | Head (for both destinations)  :
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    :Head (3rd byte)|  Tail(Dest_1) | Tail(Dest_2)  | addr.TLV.len=7:
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    :addr.TLV.len=7 |  type=SeqNum  |0|0|1|1|0|1|Rsv| tlv-length=4  |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |        Dest_1 Sequence #      |        Dest_2 Sequence #      |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 7: Example IPv4 RERR with Two Unreachable Nodes

The fields in Figure 7 are to be interpreted as follows:

- msg-type=RERR (first [and only] message is of type RERR)
- MF=4 (Message Flags = 4 [only msg-hop-limit field is present])
- MAL=3 (Message Address Length indicator [3 for IPv4, 15 for IPv6])
- msg-size=24 (octets -- counting MsgHdr, MsgTLVs, and AddrBlks)
- msg-hop-limit (initially MAX_HOPCOUNT by default)
- msg.tlvs-length=0 (no Message TLVs)
- num-addr=2 (OrigNode and TargNode addresses in RteMsg AddrBlock)
- AddrBlk flags == 10000000 [same as RREQ and RREP AddrBlk examples]
- head-length=3 (length of head part of each address is 3 octets)
- Head (3 initial bytes for both Unreachable Nodes, Dest_1 and Dest_2)
- Dest_1.Tail (4th byte of Dest_1 IP address)
- Dest_2.Tail (4th byte of Dest_2 IP address)
- addr.TLV.len = 7 (length in bytes for SeqNum TLV
- type=SeqNum (AddrTLV type of AddrBlk TLV, values 2 octets each)
- AddrTLV flags for SeqNum TLV:

  * bit 0 (thastypeext): 0
  * bit 1 (thassingleindex): 0
  * bit 2 (thasmultiindex): 1
  * bit 3 (thasvalue): 1
  * bit 4 (thasextlen): 0
  * bit 5 (tismultivalue): 1
  * bits 6-7: RESERVED

- o tlv-length=4 (so there are two TLV values, [2 = 4/2])
- o Dest_1 Sequence # (first of two TLV values for the SeqNum TLV)
- o Dest_2 Sequence # (second of two TLV values for the SeqNum TLV)

B.4. RREP_ACK Message Format

The figure below illustrates a packet format for an example RREP_ACK message.

```
     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
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |msgtype=RREPAck| MF=0  | MAL=3 |          msg-size=4           |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 8: Example IPv4 RREP_ACK

Appendix C. Changes since revision ...-04.txt

This section lists the changes since AODVv2 revision ...-04.txt

- o Normative text moved out of definitions into the relevant section of the body of the specification.
- o Editorial improvements and improvements to consistent terminology were made. Replaced "retransmit" by the slightly more accurate term "regenerate".
- o Issues were resolved as discussed on the mailing list.
- o Changed definition of LoopFree as suggested by Kedar Namjoshi and Richard Trefler to avoid the failure condition that they have described. In order to make understanding easier, replaced abstract parameters R1 by RteMsg and R2 by Route to reduce the level of abstraction when the function LoopFree is discussed.
- o Added text to clarify that different metrics may have different data types and different ranges of acceptable values.
- o Added text to section "RteMsg Structure" to emphasize the proper use of RFC 5444.
- o Included within the main body of the specification the mandatory setting of the TLV flag thassingleindex for TLVs OrigSeqNum and TargSeqNum.
- o Made more extensive use of the AdvRte terminology, in order to better distinguish between the incoming RREQ or RREP message

(i.e., RteMsg) versus the route advertised by the RteMsg (i.e., AdvRte).

Appendix D. Changes since revision ...-03.txt

This section lists the changes since AODVv2 revision ...-03.txt

- An appendix was added to exhibit algorithmic code for implementation of AODVv2 functions.
- Numerous editorial improvements and improvements to consistent terminology were made. Terminology related to prefix lengths was made consistent. Some items listed in "Notational Conventions" were no longer used, and so deleted.
- Issues were resolved as discussed on the mailing list.
- Appropriate instances of "may" were changed to "MAY".
- Definition inserted for "upstream".
- Route.Precursors included as an *optional* route table field
- Reworded text to avoid use of "relevant".
- Deleted references to "DestOnly" flag.
- Refined statements about Metric Type TLV to allow for omission when Metric Type == HopCount.
- Bulletized list in section 8.1
- ENABLE_IDLE_UNREACHABLE renamed to be ENABLE_IDLE_IN_RERR
- Transmission and subscription to LL-MANET-Routers converted to MUST from SHOULD.

Appendix E. Changes since revision ...-02.txt

This section lists the changes since AODVv2 revision ...-02.txt

- The "Added Node" feature was removed. This feature was intended to enable additional routing information to be carried within a RREQ or a RREP message, thus increasing the amount of topological information available to nodes along a routing path. However, enlarging the packet size to include information which might never be used can increase congestion of the wireless medium. The feature can be included as an optional feature at a later date when better algorithms are understood for determining when the inclusion of additional routing information might be worthwhile.
- Numerous editorial improvements and improvements to consistent terminology were made. Instances of OrigNodeNdx and TargNodeNdx were replaced by OrigNdx and TargNdx, to be consistent with the terminology shown in Table 1.
- Example RREQ and RREP message formats shown in the Appendices were changed to use OrigSeqNum and TargSeqNum message TLVs instead of using the SeqNum message TLV.
- Inclusion of the OrigNode's SeqNum in the RREP message is not specified. The processing rules for the OrigNode's SeqNum were

incompletely specified in previous versions of the draft, and very little benefit is foreseen for including that information, since reverse path forwarding is used for the RREP.

- Additional acknowledgements were included, and contributors names were alphabetized.
- Definitions in the Terminology section capitalize the term to be defined.
- Uncited bibliographic entries deleted.
- Ancient "Changes" sections were deleted.

Appendix F. Multi-homing Considerations

Multi-homing is not supported by the AODVv2 specification. There has been previous work indicating that it can be supported by expanding the sequence number to include the AODVv2 router's IP address as a parsable field of the SeqNum. Otherwise, comparing sequence numbers would not work to evaluate freshness. Even when the IP address is included, there isn't a good way to compare sequence numbers from different IP addresses, but at least a handling node can determine whether the two given sequence numbers are comparable. If the route table can store multiple routes for the same destination, then multi-homing can work with sequence numbers augmented by IP addresses.

This non-normative information is provided simply to document the results of previous efforts to enable multi-homing. The intention is to simplify the task of future specification if multihoming becomes needed for reactive protocol operation.

Appendix G. Shifting Network Prefix Advertisement Between AODVv2 Routers

Only one AODVv2 router within a MANET SHOULD be responsible for a particular address at any time. If two AODVv2 routers dynamically shift the advertisement of a network prefix, correct AODVv2 routing behavior must be observed. The AODVv2 router adding the new network prefix must wait for any existing routing information about this network prefix to be purged from the network. Therefore, it must wait at least ROUTER_SEQNUM_AGE_MAX_TIMEOUT after the previous AODVv2 router for this address stopped advertising routing information on its behalf.

Authors' Addresses

Charles E. Perkins
Futurewei Inc.
2330 Central Expressway
Santa Clara, CA 95050
USA

Phone: +1-408-330-5305
Email: charliep@computer.org

Stan Ratliff
Cisco
170 West Tasman Drive
San Jose, CA 95134
USA

Email: sratliff@cisco.com

John Dowdell
Airbus Defence and Space
Celtic Springs
Newport, Wales NP10 8FZ
United Kingdom

Email: john.dowdell@cassidian.com