Ad Hoc On-demand Distance Vector (AODVv2) Routing
draft-ietf-manet-aodvv2-10

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.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on January 7, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents
Table of Contents

1. Overview ................................................. 4
2. Terminology ............................................. 5
3. Applicability Statement ................................. 10
4. Data Structures .......................................... 11
   4.1. Interface List ...................................... 11
   4.2. Router Client List ................................. 11
   4.3. Neighbor Table .................................... 12
   4.4. Sequence Numbers .................................. 13
   4.5. Multicast Route Message Table .................... 14
   4.6. Route Table Entry ................................ 15
5. Metrics .................................................. 17
   5.1. Cost Function ...................................... 17
   5.2. LoopFree Function .................................. 18
   5.3. Default Metric Type ................................. 18
   5.4. Alternate Metric Types .............................. 19
6. AODVv2 Protocol Operations ............................... 19
   6.1. Initialization ....................................... 19
   6.2. Adjacency Monitoring ............................... 20
   6.3. Message Transmission ............................... 22
   6.4. Route Discovery, Retries and Buffering ............ 22
   6.5. Processing Received Route Information .............. 24
      6.5.1. Evaluating Route Information ................ 25
      6.5.2. Applying Route Updates ....................... 26
   6.6. Suppressing Redundant Messages .................... 28
   6.7. Route Maintenance .................................. 29
      6.7.1. Route State .................................. 29
      6.7.2. Reporting Invalid Routes ..................... 31
7. AODVv2 Protocol Messages ................................. 31
   7.1. Route Request (RREQ) Message ...................... 31
      7.1.1. RREQ Generation ............................... 33
      7.1.2. RREQ Reception ............................... 34
      7.1.3. RREQ Regeneration ............................ 35
   7.2. Route Reply (RREP) Message ....................... 36
      7.2.1. RREP Generation ............................... 38
      7.2.2. RREP Reception ............................... 39
      7.2.3. RREP Regeneration ............................ 41
   7.3. Route Reply Acknowledgement (RREP_Ack) Message .... 42
      7.3.1. RREP_Ack Generation ......................... 42
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 hoc networks [MANETs] [RFC2501]. The basic operations of the AODVv2 protocol are route discovery and route maintenance.
Route discovery is performed when an AODVv2 router needs to forward a packet for one of its clients, but does not have a valid route to the packet’s destination. AODVv2 routers use Route Request (RREQ) and Route Reply (RREP) messages to carry route information between the originator of the route discovery and the target node, establishing a route to both endpoints on all intermediate routers.

A metric is included in RREQ and RREP messages to represent the cost of the route to the originator or target of the route discovery. AODVv2 compares route metrics in a way that ensures loop avoidance. AODVv2 also uses sequence numbers to assure loop freedom, enabling identification of stale routing information so that it can be discarded.

Route maintenance involves monitoring the router’s links and routes for changes. This includes confirming adjacencies with other AODVv2 routers, issuing Route Error messages if link failures invalidate routes, extending and enforcing route timeouts, and reacting to received Route Error messages.

AODVv2 control plane messages use the Generalized MANET Packet/Message Format defined in [RFC5444] and the parameters in [RFC5498]. AODVv2 defines a set of data elements which map to RFC 5444 Address Blocks, Address Block TLVs, and Message TLVs.

Security for authentication of AODVv2 routers and encryption of control messages is dealt with by using the recommendations in [RFC7182].

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]. In addition, this document uses terminology from [RFC5444], and defines the following terms:

AddressList  
An AODVv2 Data Element (see Table 1).

Adjacency  
A bi-directional relationship between neighboring AODVv2 routers for the purpose of exchanging routing information.

AckReq  
An AODVv2 Data Element (see Table 1).

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

CurrentTime
The current time as maintained by the AODVv2 router.

Data Element
A named object used within AODVv2 protocol messages (see Table 1).

Disregard
Ignore for further processing.

Invalid route
A route that cannot be used for forwarding.

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

MetricType
An AODVv2 Data Element (see Table 1).

MetricTypeList
An AODVv2 Data Element (see Table 1).

Node
An IP addressable device in the ad hoc network. All nodes in this document are either AODVv2 Routers or Router Clients.

OrigAddr (Originator Address)
An AODVv2 Data Element (see Table 1).

OrigMetric
An AODVv2 Data Element (see Table 1).

OrigNode (Originating Node)
The node that launched the application requiring communication with the Target Address.

OrigPrefixLen
The prefix length, in bits, associated with OrigAddr.

OrigSeqNum
An AODVv2 Data Element (see Table 1).

PktSource
An AODVv2 Data Element (see Table 1).

PrefixLengthList
An AODVv2 Data Element (see Table 1).

Reactive
A protocol operation is called "reactive" if it is performed only in reaction to specific events. In this document, "reactive" is synonymous with "on-demand".

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

RERR_Gen (RERR Generating Router)
The AODVv2 router generating a Route Error message.

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.

Router Client
A node that requires the services of an AODVv2 router. An AODVv2 router is also its own client.

RREP (Route Reply)
The AODVv2 message type used to reply to a Route Request message.

RREP_Gen (RREP Generating Router)
The AODVv2 router responsible for the Target Node of a Route Request message, i.e., the router that creates the Route Reply message.

RREQ (Route Request)
The AODVv2 message type used to discover a route to the Target Address and distribute information about the route to the Originator Address.

RREQ_Gen (RREQ Generating Router)
The AODVv2 router that creates the Route Request message on behalf of the Originating Node to discover a route for Target Address.

RteMsg (Route Message)
A Route Request (RREQ) or Route Reply (RREP) message.

Sequence Number (SeqNum)
One of the sequence numbers maintained by an AODVv2 router to indicate freshness of route information. Used as an AODVv2 Data Element (see Table 1).
SeqNumList
   An AODVv2 Data Element (see Table 1).

TargAddr (Target Address)
   An AODVv2 Data Element (see Table 1).

Target Node
   The node hosting the IP address toward which a route is needed.

TargMetric
   An AODVv2 Data Element (see Table 1).

TargPrefixLen
   The prefix length, in bits, associated with TargAddr.

TargSeqNum
   An AODVv2 Data Element (see Table 1).

Unreachable Address
   An address for which a valid route is not known.

Upstream
   In the direction from destination to source (from TargAddr to OrigAddr).

Valid route
   A route that can be used for forwarding.

ValidityTime
   An AODVv2 Data Element (see Table 1).

This document defines a set of Data Elements in Table 1 which are used in AODVv2 messages. These data elements contain the message data which is transferred into RFC 5444 formatted messages (Section 8) before sending.
<table>
<thead>
<tr>
<th>Data Element</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>AckReq</td>
<td>Presence in RREP means acknowledgement is requested from the router with the address indicated</td>
</tr>
<tr>
<td>AddressList</td>
<td>A list of IP addresses</td>
</tr>
<tr>
<td>MetricType</td>
<td>The metric type for a metric value</td>
</tr>
<tr>
<td>MetricTypeList</td>
<td>Metric types associated with routes to addresses in AddressList, used in RERR</td>
</tr>
<tr>
<td>msg_hop_limit</td>
<td>Number of hops the message is allowed to traverse</td>
</tr>
<tr>
<td>msg_hop_count</td>
<td>Number of hops traversed so far by the message</td>
</tr>
<tr>
<td>OrigMetric</td>
<td>Metric value associated with the route to OrigAddr</td>
</tr>
<tr>
<td>OrigSeqNum</td>
<td>Sequence number associated with OrigAddr, used in RREQ</td>
</tr>
<tr>
<td>OrigAddr</td>
<td>IP address of the Originating Node, the source address of the packet triggering route discovery</td>
</tr>
<tr>
<td>PktSource</td>
<td>Source address of a packet triggering a RERR message</td>
</tr>
<tr>
<td>PrefixLengthList</td>
<td>A list of routing prefixes associated with addresses in AddressList</td>
</tr>
<tr>
<td>SeqNum</td>
<td>Sequence number, when used in RERR</td>
</tr>
<tr>
<td>SeqNumList</td>
<td>A list of generic sequence numbers associated with addresses in an AddressList, used in RERR</td>
</tr>
<tr>
<td>TargAddr</td>
<td>IP address of the Target Node, the destination address for which a route is requested</td>
</tr>
<tr>
<td>TargMetric</td>
<td>Metric value associated with the route to TargAddr</td>
</tr>
<tr>
<td>TargSeqNum</td>
<td>Sequence number associated with TargAddr, used in RREQ (optional) and RREP</td>
</tr>
<tr>
<td>ValidityTime</td>
<td>Length of time a route is offered</td>
</tr>
</tbody>
</table>

**Table 1: Data Elements**

This document uses the notational conventions in Table 2 to simplify the text.
Table 2: Notational Conventions

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Route[Address]</td>
<td>A route table entry toward Address</td>
</tr>
<tr>
<td>Route[Address].Field</td>
<td>A field in a route table entry toward Address</td>
</tr>
<tr>
<td>RteMsg</td>
<td>Either RREQ or RREP</td>
</tr>
<tr>
<td>RteMsg.Field</td>
<td>A field in either RREQ or RREP</td>
</tr>
<tr>
<td>AdvRte</td>
<td>A route advertised in an incoming RteMsg</td>
</tr>
</tbody>
</table>

3. Applicability Statement

The AODVv2 routing protocol is a reactive routing protocol designed for stub or disconnected mobile ad hoc networks, i.e., non-transit networks or those not connected to the internet.

AODVv2 handles a wide variety of mobility and traffic patterns by determining routes on-demand. In networks with a large number of routers, AODVv2 is best suited for relatively sparse traffic scenarios where each router forwards packets to a small percentage of other AODVv2 routers in the network. In this case fewer routes are needed, and therefore less control traffic is produced.

AODVv2 is well suited to reactive scenarios such as emergency and disaster relief, where the ability to communicate is more important than being assured of secure operations. For other ad hoc networking applications, in which insecure operation could negate the value of establishing communication paths, it is important for neighboring AODVv2 nodes to establish security associations with one another.

AODVv2 will not make use of uni-directional links. Route requests from routers which cannot confirm bidirectional connectivity should be ignored to avoid persistent packet loss or protocol failures.

AODVv2 is applicable to memory constrained devices, since only a little routing state is maintained in each AODVv2 router. In contrast to proactive routing protocols, which maintain routing information for all destinations within the MANET, AODVv2 routes that are not needed for forwarding data do not need to be maintained. On routers unable to store persistent AODVv2 state, recovery can impose a performance penalty (e.g., in case of AODVv2 router reboot).

AODVv2 supports routers with multiple interfaces, as long as each interface configured for AODVv2 has its own unicast routable IP address. Address assignment procedures are out of scope for AODVv2.
Multi-homing is not supported by AODVv2, and therefore a Router Client SHOULD NOT be served by more than one AODVv2 router at any one time. Appendix B contains some notes on this topic.

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

The routing algorithm in AODVv2 may be operated at layers other than the network layer, using layer-appropriate addresses.

AODVv2 can be configured to perform gateway functions when attached to the internet. Such a gateway router is referred to as an Internet AODVv2 Router (IAR) as discussed in Section 9. The IAR will reply to each route request generated inside the AODVv2 network for an internet destination as if they were responsible for the target address, and may advertise the AODVv2 network to the internet using procedures out of scope for this specification.

4. Data Structures

4.1. Interface List

A list of the interfaces supporting AODVv2 should be configured in the AODVv2_INTERFACES list.

4.2. Router Client List

An AODVv2 router may provide routing services for its own local applications and for other non-routing nodes that are directly reachable via its network interfaces. These nodes, including the AODVv2 router itself, are referred to as Router Clients.

Each AODVv2 router MUST be configured with information about the IP addresses of its clients. If a subnet is configured as a Router Client, the AODVv2 router MUST serve every node in that subnet.

A CLIENT_ADDRESSES list should exist to store information about Router Clients, with the following information:

RouterClient.IPAddress
  The IP address of the client node or subnet that requires routing services from the AODVv2 router.

RouterClient.PrefixLength
  The length, in bits, of the routing prefix associated with the client IP address or subnet.
The list of Router Clients for an AODVv2 router is never empty, since an AODVv2 router is always its own client. The IP Addresses of the router’s interfaces will appear in the Router Client list.

The router MUST respond to Route Requests for all Router Clients. In the initial state, an AODVv2 router is not required to have information about the Router Clients of any other AODVv2 router.

A Router Client SHOULD NOT be served by more than one AODVv2 router at any one time. Shifting responsibility for a Router Client to a different AODVv2 router is discussed in Appendix C.

4.3. Neighbor Table

A neighbor table MUST be maintained with information about neighboring AODVv2 routers, including an indication of the state of the adjacency to the router. Section 6.2 discusses how to monitor adjacency.

Neighboring routers which cannot confirm adjacency should be marked as blacklisted. Certain AODVv2 messages received from a blacklisted router should be ignored. Routers should be blacklisted for a maximum of MAX_BLACKLIST_TIME, but can be removed from the blacklist before this time if an indication of adjacency is received.

Neighbor entries should contain:

Neighbor.IPAddress
   An IP address of the neighboring router.

Neighbor.State
   The state of the adjacency (Confirmed, Unknown, or Blacklisted)

Neighbor.ResetTime
   The time at which this router SHOULD no longer be considered blacklisted. By default this value is calculated at the time the router is blacklisted and is equal to CurrentTime + MAX_BLACKLIST_TIME. After this time, the state should be reset to Unknown. While the neighbor is not marked as blacklisted, this value SHOULD be set to MAX_TIME.

Before a neighbor is confirmed, any routes learned through that neighbor are marked as Unconfirmed. When neighbor state is set to Confirmed, the Unconfirmed routes using the neighbor as a next hop can transition to Idle state (see Section 6.7.1).

If a neighbor is blacklisted, any valid routes installed which use that neighbor for their next hop should become Invalid.
When the link to a neighbor breaks, the neighbor entry should be removed and all routes using the neighbor as next hop should become invalid.

4.4. Sequence Numbers

Sequence numbers enable AODVv2 routers to determine the temporal order of route discovery messages, identifying stale routing information so that it can be discarded. The sequence number fulfills the same roles as the "Destination Sequence Number" of DSDV [Perkins94], and the AODV Sequence Number in [RFC3561].

Each AODVv2 router in the network MUST maintain its own sequence number as a 16-bit unsigned integer. All route messages (Route Request and Route Reply messages) created by an AODVv2 router include the router’s sequence number.

If the router has multiple AODVv2 interfaces, it can maintain different sequence numbers for each interface IP address, but the router MUST NOT use multiple sequence numbers for any one interface IP address. All route messages created on behalf of Router Clients on a particular interface MUST include the sequence number of that interface’s IP address.

Each AODVv2 router MUST ensure that its sequence number is strictly increasing. It can ensure this by incrementing the sequence number by one (1) whenever a route message is created, except when the sequence number is 65,535 (the maximum value of a 16-bit unsigned integer), in which case it MUST be reset to one (1). The value zero (0) is reserved to indicate that the sequence number for an address is unknown.

A router receiving a route message uses the sequence number to determine the freshness of the route information in the message in comparison with any existing information about the same route. If the sequence number stored in the route table is higher than the sequence number in the message, the received information is considered stale and MUST NOT be used to update the route table.

As a consequence, loop freedom is assured.

An AODVv2 router SHOULD maintain its sequence number(s) in persistent storage. If a sequence number is lost, the router must follow the procedure in Section 6.1 to safely resume routing operations with a new sequence number.
4.5. Multicast Route Message Table

A route message (RteMsg) is either a Route Request and Route Reply message. The Multicast Route Message Table (RteMsg Table) contains information about previously received multicast route messages, so that when a route message is received, an AODVv2 router can determine if the incoming information is redundant, and avoid unnecessary regeneration of the route message. RREQ messages are usually multicast. Future extensions to AODVv2 MAY enable RREP messages to be multicast.

Each entry in the RteMsg Table stores the following information, copied from the route message:

RteMsg.MessageType
- Either RREQ or RREP.

RteMsg.OrigAddr
- The IP address of the originating node wishing to send a packet.

RteMsg.OrigPrefixLen
- The prefix length associated with OrigAddr.

RteMsg.TargAddr
- The IP address of the target node, the destination of the packet.

RteMsg.TargPrefixLen
- The prefix length associated with TargAddr.

RteMsg.OrigSeqNum
- The sequence number associated with the originator, if present in RteMsg.

RteMsg.TargSeqNum
- The sequence number associated with the target, if present in RteMsg.

RteMsg.MetricType
- The metric type of the route requested.

RteMsg.Metric
- The metric value received in the RteMsg.

RteMsg.Timestamp
- The last time this entry was updated.

RteMsg.RemoveTime
- The time at which this entry MUST be removed.
Multicast RteMsgs are considered to be comparable if they have the same MessageType, OrigAddr, TargAddr, and MetricType. The RteMsg Table is maintained so that no two entries are comparable, i.e., all entries either have different MessageType, different OrigAddr, different TargAddr, or different MetricType. See Section 6.6 for details on updating this table.

Entries in the RteMsg Table MUST be deleted when the sequence number is no longer valid, i.e., after MAX_SEQNUM_LIFETIME. RemoveTime should be set when the sequence number of the entry is updated, i.e., when OrigSeqNum is updated for a RREQ, and when TargSeqNum is updated for a RREP. RemoveTime should be set to CurrentTime + MAX_SEQNUM_LIFETIME. Memory-constrained devices MAY remove the entry before this time, but the entry should be maintained for at least RteMsg_ENTRY_TIME after the last Timestamp update in order to account for long-lived RREQs traversing the network.

4.6. Route Table Entry

The route table entry is a conceptual data structure. Implementations MAY use any internal representation that provides access to the following information:

Route.Address
An address or address prefix of a node.

Route.PrefixLength
The prefix length, in bits, associated with the address. If it is less than the length of Route.Address, this is a route to the subnet on which Route.Address resides.

Route.SeqNum
The sequence number associated with Route.Address, obtained from the last route message that successfully updated this route table entry.

Route.NextHop
An IP address of the AODVv2 router used for the next hop on the path toward Route.Address.

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

Route.LastUsed
The time this route was last used to forward a packet.

Route.LastSeqNumUpdate
The time the sequence number for this route was last updated.
Route.ExpirationTime
    The time at which this route must be marked as Invalid.

Route.MetricType
    The type of metric associated with this route.

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

Route.State
    The last known state (Active, Idle, Invalid, or Unconfirmed) of
    the route.

Route.Precursors (optional feature)
    A list of upstream neighbors using the route (see Section 10.2).

There are four possible states for an AODVv2 route:

Active
    An Active route is in current use for forwarding packets.

Idle
    An Idle route has not been used in the last ACTIVE_INTERVAL, but
    can still be used for forwarding packets.

Invalid
    An Invalid route cannot be used for forwarding packets, but its
    sequence number information allows incoming information to be
    assessed for freshness.

Unconfirmed
    An Unconfirmed route cannot be used for forwarding packets. It is
    a route learned from a Route Request which has not yet been confirmed as
    bidirectional.

Route state changes are detailed in Section 6.7.1.

An AODVv2 route may be offered for a limited time. In this case, the
route is referred to as a timed route. The length of time for which
the route is valid is referred to as validity time, and is included
in messages which advertise the route. The shortened validity time
is reflected in Route.ExpirationTime. If a route is not timed, the
ExpirationTime is MAX_TIME, and the route will become Idle and then
Invalid if it is not used. Invalid routes should be maintained for
their sequence number information.
5. Metrics

Metrics measure a cost or quality associated with a route or a link, e.g., latency, delay, financial cost, energy, etc.

AODVv2 enables the use of multiple metric types. Each metric that can be used in AODVv2 has a MetricType number. Numbers are allocated by IANA as specified in [RFC6551] or detailed in Section 12.3. The default metric type is discussed in Section 5.3. Alternate metrics are discussed in Section 5.4.

An AODVv2 implementation MAY be configured to use a limited set of the supported metric types. In the processing described in Section 7, a "known" MetricType can be interpreted as a configured MetricType. If a message is received using an unknown or non-configured MetricType it MUST be ignored. Since the message will not be regenerated, other routers which do support the MetricType will not be able to route through a router which does not support the MetricType.

For each MetricType, a maximum value is defined, denoted MAX_METRIC[MetricType]. AODVv2 cannot store routes that cost more than MAX_METRIC[MetricType].

Metric information reported in incoming route messages describes the metric as measured by message sender, and does not reflect the cost of traversing the link to that sender. The receiving router calculates the cost of the route from its perspective. This cost is used to determine whether to use incoming information to update an existing route. If the cost exceeds MAX_METRIC[MetricType], the route is ignored.

5.1. Cost Function

This document uses the following notation to represent costs:

- Cost(L) for link cost
- Cost(R) for route cost

These functions return the cost of traversing a link or a route. Cost(L) and Cost(R) for the default metric type are detailed in Section 5.3. The Cost() functions for other metric types are beyond the scope of this document.
5.2. LoopFree Function

When comparing an advertised route to an existing route for the same destination and the same metric type, the metric value should be examined to ensure that using the advertised route does not create any routing loops.

The function LoopFree(R1, R2) is defined to verify that a route R2 is not a part of another route R1. LoopFree returns TRUE if R2 cannot be a sub-section of the route R1.

An AODVv2 router invokes LoopFree() as part of the process in Section 6.5.1. The advertised route (AdvRte) is used as parameter R1, and the stored route (Route) is used as parameter R2.

The LoopFree(R1, R2) function for the default metric type is detailed in Section 5.3. The LoopFree(R1, R2) functions for other metric types are beyond the scope of this document.

5.3. Default Metric Type

AODVv2’s default metric type (DEFAULT_METRIC_TYPE) is HopCount, and is the only metric described in detail in this document. Alternate metrics are discussed in Section 5.4.

For the HopCount metric:

- Cost(L) := 1
- Cost(R) := Sum (Cost(L) of each link in the route), i.e., the hop count between the router calculating the cost, and the destination of the route
- LoopFree(R1, R2) returns TRUE when the cost of R1 is less than or equal to the cost of R2, i.e., Cost(R1) <= Cost(R2)
- MAX_METRIC[HopCount] := MAX_HOPCOUNT

The LoopFree function for the HopCount metric is derived from the fact that route cost increases with number of hops. When examining two routes, the route with higher cost may include the route with lower cost as a sub-section of its route. Therefore, an advertised route with higher cost than the corresponding stored route could include the stored route as a sub-section. Replacing the stored route with the received route could form a routing loop. LoopFree returns FALSE in this case to indicate that an advertised route with higher cost is not to be used to update a stored route, even if the stored route is Invalid.
MAX_HOPCOUNT is a constant defined in Section 11.2. It MUST be larger than the AODVv2 network diameter, in order that AODVv2 protocol messages may reach their intended destinations.

5.4. Alternate Metric Types

Some applications may require metric information other than hop count. For this reason, AODVv2 enables route selection based on alternate metric types.

Using non-default metrics in an AODVv2 message requires the inclusion of the MetricType data element.

Alternate metrics may have different types and ranges, for example integers or floating point numbers, or restricted subsets thereof. Therefore the size of the metric field in route messages may vary. See Section 12.3 for further information on MetricType number allocation and size.

Metrics might be classified as additive, concave, convex, or multiplicative as discussed in [RFC6551]. Where Cost and LoopFree functions can be developed for a metric type, it can be supported by AODVv2.

AODVv2 can support additive metrics using the Cost(R) and LoopFree(R1, R2) functions defined for the default metric. Furthermore, any strictly increasing metric can be supported using the LoopFree function defined. It is, however, out of the scope of this document to specify for alternate metrics the correct Cost(L), Cost(R), and LoopFree() functions. Where possible these should take into account differences in the link cost in each direction.

6. AODVv2 Protocol Operations

The AODVv2 protocol’s operations include managing sequence numbers, monitoring adjacent AODVv2 routers, performing route discovery and dealing with requests from other routers, processing incoming route information and updating the route table, suppressing redundant messages, maintaining the route table and reporting broken routes. These processes are discussed in detail in the following sections.

6.1. Initialization

During initialization where the previous sequence number is unknown, or if the sequence number is lost at any point, an AODVv2 router resets its sequence number(s) to one (1). However, other AODVv2 routers may still hold sequence number information this router previously issued. Since sequence number information will be removed...
if there have been no sequence number updates in MAX_SEQNUM_LIFETIME, the initializing router must wait for MAX_SEQNUM_LIFETIME before it creates any messages containing its sequence number. It can then be sure that the information it sends will not be considered stale.

Until MAX_SEQNUM_LIFETIME after its sequence number is reset, the router SHOULD not create RREQ or RREP messages.

During this wait period, the router can do the following:

- Process information in a received RREQ or RREP message to learn a route to the originator or target
- Send a RREP_Ack
- Regenerate a received RREQ or RREP
- Forward data packets to Router Clients, and to other routers, if a route exists
- Create, process and regenerate RERR messages

6.2. Adjacency Monitoring

An adjacency is a bidirectional relationship between neighboring AODVv2 routers for the purpose of exchanging routing information. Not every pair of neighboring routers will necessarily form an adjacency, but AODVv2 routers MUST monitor connectivity to neighboring AODVv2 routers along potential routes and MUST NOT establish routes over uni-directional links, since packet losses are likely to occur and route establishment can fail.

The default approach for monitoring bidirectional connectivity to the next hop toward OrigAddr is to request acknowledgement of Route Reply messages. Receipt of an acknowledgement proves that bidirectional connectivity exists. All AODVv2 routers MUST support this process, which is explained in Section 7.2 and Section 7.3.

Bidirectionality to the next hop toward TargAddr is confirmed by receipt of the Route Reply message, since a Route Reply message is a reply to a Route Request message which previously crossed the link in the opposite direction.

When routers perform other operations such as those from the list below, these can be used as additional indications of connectivity:

- NHDP HELLO Messages [RFC6130]
For example, receipt of a Neighborhood Discovery Protocol HELLO message with the receiving router listed as a neighbor is a signal of bidirectional connectivity. In this case, acknowledgement of a RREP message sent to that neighbor is unnecessary. Similarly, if AODVv2 receives notification of a timeout, this may be due to a disconnection. The AODVv2 router SHOULD attempt to verify connectivity by requesting acknowledgement of the next RREP sent to that neighbor.

The Neighbor Table (Section 4.3) gives the last known state of the neighbor adjacency, either Confirmed, Unknown, or Blacklisted. Until bidirectionality is confirmed, the state is Unknown, and acknowledgement of RREP messages MUST be requested. If the state is Confirmed, the acknowledgement request is unnecessary. If bidirectionality cannot be confirmed, the state is Blacklisted. RREQs received from a blacklisted router, or any router over a link that is known to be incoming-only, MUST be disregarded.

Neighbor state is updated as follows:

- If a link to a neighbor is determined to be unidirectional, either by failure to acknowledge a RREP, or by some other means, the neighbor MUST be marked as blacklisted.

- If a notification indicates that there may be a problem with bidirectionality, and the neighbor state is currently Confirmed, the state SHOULD be set to Unknown to force acknowledgement of the next RREP sent to the neighbor.

- If an indication of bidirectional connectivity is received, the neighbor state SHOULD be set to Confirmed.

- If the neighbor state is Blacklisted and the reset time is reached, the neighbor state SHOULD be reset to Unknown and the neighbor SHOULD again be allowed to participate in route discovery.
If a link to a neighbor is determined to be broken, the neighbor entry should be removed.

6.3. Message Transmission

In its default mode of operation, AODVv2 sends [RFC5444] formatted messages using the parameters for port number and IP protocol specified in [RFC5498]. Mapping of AODVv2 data elements to RFC 5444 is detailed in Section 8.

Unless otherwise specified, AODVv2 multicast messages are sent to the link-local multicast address LL-MANET-Routers [RFC5498]. All AODVv2 routers MUST subscribe to LL-MANET-Routers [RFC5498] to receive AODVv2 messages.

Implementations are free to choose their own heuristics for reducing multicast overhead. Some methods for doing so are described 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 messages. Note that multicast messages 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.

When multiple interfaces are available, a node transmitting a multicast message to LL-MANET-Routers MUST send the message on all interfaces that have been configured for AODVv2 operation, as given in the AODVv2_INTERFACES list (Section 4.1). Similarly, AODVv2 routers MUST subscribe to LL-MANET-Routers on all their AODVv2 interfaces.

To avoid congestion, each AODVv2 router’s rate of message generation (CONTROL_TRAFFIC_LIMIT) SHOULD be limited and administratively configurable. The implementation is free to choose the algorithm for limiting messages, including prioritizing messages when approaching the limit. AODVv2 messages SHOULD be discarded in the following order: RERR for invalidated routes, RREQ, RREP, RERR for undeliverable packet, RREP_Ack.

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

6.4. Route Discovery, Retries and Buffering

AODVv2’s RREQ and RREP messages are used for route discovery and are together known as route messages (RteMsgs). The main difference between the two messages is that, by default, RREQ messages are multicast to solicit a RREP, whereas RREP is unicast as a response to
When an AODVv2 router needs to forward a data packet (with source address OrigAddr and destination address TargAddr) from one of its Router Clients, it needs a route to the packet's destination. If no route exists, the AODVv2 router generates and multicasts a Route Request message (RREQ) using OrigAddr and TargAddr. The procedure for this is described in Section 7.1.1. The AODVv2 router is referred to as RREQ_Gen.

Data packets awaiting a route MAY be buffered by RREQ_Gen. Buffering of data packets can have both positive and negative effects (albeit usually positive). Real-time traffic, voice, and scheduled delivery may suffer if packets are buffered and subjected to delays, but TCP connection establishment will benefit if packets are queued while route discovery is performed.

The packet buffer is configured with a fixed limited size of BUFFER_SIZE_PACKETS or BUFFER_SIZE_BYTES. Determining which packets to discard first when the buffer is full is a matter of policy at each AODVv2 router. Nodes without sufficient memory available for buffering SHOULD have buffering disabled by configuring BUFFER_SIZE_PACKETS := 0 and BUFFER_SIZE_BYTES := 0. This will affect the latency for launching TCP applications to new destinations.

RREQ_Gen awaits reception of a Route Reply message (RREP) containing a route toward TargAddr. A RREQ from TargAddr would also fulfill the request, if adjacency to the next hop is already confirmed. If a route to TargAddr is not learned within RREQ_WAIT_TIME, RREQ_Gen MAY retry the route discovery by generating another RREQ with a new sequence number. To reduce congestion in a network, repeated attempts at route discovery for a particular target address SHOULD utilize a binary exponential backoff, as described in [RFC3561], where the wait time is doubled for each retry.

The RREQ is received by neighboring AODVv2 routers, and processed and regenerated as described in Section 7.1. Intermediate routers learn a potential route to OrigAddr from the RREQ. The router responsible for TargAddr responds by generating a Route Reply message (RREP) and sends it back toward RREQ_Gen using the route to OrigAddr learned from the RREQ. Each intermediate router regenerates the RREP and unicasts toward OrigAddr.

Links which are not bidirectional cause problems. If a RREP is not received at an intermediate router, the RREP cannot be regenerated and will never reach RREQ_Gen. However, since routers monitor...
adjacencies (Section 6.2), the loss of the RREP will cause the last router which regenerated the RREP to blacklist the router which did not receive it. When a timeout occurs at RREQ_Gen, a new RREQ may be regenerated. When the new RREQ arrives again via the blacklisted router, it will be ignored, and the RREQ should discover a different path.

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. Any data packets buffered for TargAddr MUST also be dropped and a Destination Unreachable ICMP message (Type 3) with a code of 1 (Host Unreachable Error) SHOULD be delivered to the source of the data packet. The source may be an application on RREQ_Gen itself, or on a Router Client with address OrigAddr.

If RREQ_Gen does receive a route message containing a route to TargAddr within the timeout, it processes the message according to Section 7. When a valid route is installed, the router can begin sending the buffered packets. Any retry timers for the corresponding RREQ SHOULD be cancelled.

During route discovery, all routers on the path learn a route to both OrigAddr and TargAddr, making the constructed route bidirectional.

6.5. Processing Received Route Information

All AODVv2 route messages contain a route. A Route Request (RREQ) includes a route to OrigAddr, and a Route Reply (RREP) contains a route to TargAddr.

All AODVv2 routers that receive a route message can store the route contained within it. Incoming information is first checked to verify that it is both safe to use and offers an improvement to existing information. This process is explained in Section 6.5.1. The route table may then be updated according to Section 6.5.2.

In the processes below, RteMsg is used to denote the route message, AdvRte is used to denote the route contained within it, and Route denotes a stored routing table entry which matches AdvRte.

AdvRte has the following properties:

- AdvRte.Address := RteMsg.OrigAddr (in RREQ) or RteMsg.TargAddr (in RREP)
AdvRte.PrefixLength := RteMsg.OrigPrefixLen (in RREQ) or RteMsg.TargPrefixLen (in RREP) if included, or if no prefix length was included in RteMsg, the address length, in bits, of AdvRte.Address

AdvRte.SeqNum := RteMsg.OrigSeqNum (in RREQ) or RteMsg.TargSeqNum (in RREP)

AdvRte.NextHop := IP.SourceAddress (the address of the router from which the AdvRte was received)

AdvRte.MetricType := RteMsg.MetricType if included, or DEFAULT_METRIC_TYPE if not

AdvRte.Metric := RteMsg.Metric

AdvRte.Cost := Cost(R) using the cost function associated with the route's metric type, where L is the link from the advertising router, i.e. Cost(R) = AdvRte.Metric + Cost(L) for the default metric type

AdvRte.ValidityTime := RteMsg.ValidityTime, if included

If prefix length information is present, the route describes the subnet on which the address resides.

6.5.1. Evaluating Route Information

To determine whether the advertised route should be used to update the routing table, the incoming route information MUST be processed as follows:

1. Search for a routing table entry (Route) matching AdvRte’s address, prefix length and metric type

   * If no matching route exists, AdvRte SHOULD be used to update the routing table. Multiple routes to the same destination may exists with different metric types.

   * If all matching routes have State set to Unconfirmed, AdvRte SHOULD be used to update the routing table, so that it contains multiple Unconfirmed routes. If an Unconfirmed route becomes valid in future, any remaining Unconfirmed routes which would not offer improvement will be expunged.

   * If a matching route exists with State set to Active, Idle, or Invalid, continue to Step 2.
2. Compare sequence numbers

* If AdvRte is more recent (AdvRte.SeqNum > Route.SeqNum), AdvRte MUST be used to update the routing table.

* If AdvRte is stale (AdvRte.SeqNum < Route.SeqNum), AdvRte MUST NOT be used to update the routing table.

* If the sequence numbers are equal, continue to Step 3.

3. Check that AdvRte is safe against routing loops (see Section 5.2)

* If LoopFree(AdvRte, Route) returns FALSE, AdvRte MUST NOT be used to update the routing table because using the incoming information might cause a routing loop.

* If LoopFree(AdvRte, Route) returns TRUE, continue to Step 3.

4. Compare route costs

* For some metric types, including the default metric specified in Section 5.3, the best route is the route with the lowest metric value. For other metric types, the best route may be the route with the highest metric.

* If AdvRte is better, it SHOULD be used to update the routing table because it offers improvement.

* If AdvRte is not better (i.e., it is worse or equal) and Route is valid, AdvRte MUST NOT be used to update the routing table because it does not offer any improvement.

* If AdvRte is not better (i.e., it is worse or equal) but Route is Invalid, AdvRte SHOULD be used to update the routing table because it can safely repair the existing Invalid route.

If the advertised route SHOULD be used to update the routing table, the procedure in Section 6.5.2 is followed.

6.5.2. Applying Route Updates

If AdvRte is from a RREQ message, the next hop neighbor may not be confirmed as adjacent (see Section 4.3). If Neighbor.State is Unknown, the route may not be viable, but it MUST be stored to allow a corresponding RREP to be sent. It SHOULD NOT yet be used to forward data. Route.State will be set to Unconfirmed to indicate this. If a valid route already exists for this destination, the Unconfirmed route should be stored as an additional entry.
The route update is applied as follows:

1. If no existing entry in the route table matches AdvRte on address, prefix length and metric type, continue to Step 3 and create a new route table entry.

2. If a matching entry exists:
   * If AdvRte.NextHop is not equal to Route.NextHop, and AdvRte.NextHop’s Neighbor.State is Unknown and Route.State is Active or Idle, the current route is valid but the advertised route may offer improvement, if it can be confirmed. Continue to Step 3 and create a new route table entry. It can replace the original route when Neighbor.State is set to Confirmed.
   * If AdvRte.NextHop’s Neighbor.State is Unknown and Route.State is Invalid, continue to Step 4 and update the existing route table entry.
   * If AdvRte.NextHop’s Neighbor.State is Confirmed, continue to Step 4 and update the existing route table entry.

3. Create a route table entry and initialize as follows:
   * Route.Address := AdvRte.Address
   * Route.PrefixLength := AdvRte.PrefixLength (if included), or the length, in bits, of Route.Address (32 for IPv4 or 128 for IPv6)
   * Route.MetricType := AdvRte.MetricType

4. Update the route table entry as follows:
   * Route.SeqNum := AdvRte.SeqNum
   * Route.NextHop := AdvRte.NextHop
   * Route.NextHopInterface := interface on which RteMsg was received
   * Route.LastUsed := CurrentTime
   * Route.LastSeqNumUpdate := CurrentTime
Route.ExpirationTime := CurrentTime + AdvRte.ValidityTime if a validity time exists, otherwise MAX_TIME

5. If a new route was created, or if the existing Route.State is Invalid or Unconfirmed, update as follows:
   * Route.State := Unconfirmed (if the next hop’s Neighbor.State is Unknown) or Idle

6. If an existing route changed from Invalid or Unconfirmed to become Idle, any matching inferior routes should be expunged.

7. If this update results in a valid route which fulfills an outstanding route discovery, the associated timers can be cancelled and any buffered packets for this route can be forwarded.

6.6. Suppressing Redundant Messages

When route messages are flooded in a MANET, an AODVv2 router may receive multiple similar messages. Regenerating every one of these gives little additional benefit, and generates unnecessary signaling traffic and interference.

Each AODVv2 router stores information about recently received route messages in the AODVv2 Multicast RteMsg Table (Section 4.5). Received RteMsgs are tested against previously received RteMsgs, and if determined to be redundant, regeneration can be avoided. Where necessary, regeneration is performed using the processes in Section 7.

To determine if a received RREQ is redundant:

1. Search for an entry in the RteMsg Table with the same MessageType, OrigAddr, TargAddr, and MetricType
   * If there is none, create an entry to store information about the received RREQ and regenerate the RREQ.

   * If there is an entry, update the timestamp field, since comparable RteMsgs are still traversing the network, and continue to Step 2.

2. Compare the sequence numbers
   * If the entry has a lower OrigSeqNum than the received RREQ, update the entry using information from the new RREQ and regenerate the RREQ.
3. Compare the metric values

* If the entry has a Metric that is worse than the received RREQ, update the entry using information from the new RREQ.

* If the entry has a Metric that is better than the received RREQ, do not update the entry.

* In both cases, the RREQ MAY be suppressed to avoid extra control traffic. However, if the processing of the RREQ results in an update to the route table, the RREQ MAY be regenerated to ensure other routers have up-to-date information.

6.7. Route Maintenance

Route maintenance involves monitoring and updating route state, handling route timeouts and reporting routes that become Invalid.

Before using a route to forward a packet, an AODVv2 router MUST check the status of the route (Section 6.7.1). If the route is valid, it MUST be marked as Active and its LastUsed timestamp MUST be updated, before forwarding the packet to the route's next hop. If there is no valid route, this MUST be reported to the packet's source (see Section 6.7.2).

6.7.1. Route State

During normal operation, AODVv2 does not require any explicit timeouts to manage the lifetime of a route. At any time, any route table entry can be examined and updated according to the rules below. Route state should be checked before packet forwarding and before any operation based on route state.

The four possible states for an AODVv2 route are Active, Idle, Invalid, and Unconfirmed, as defined in Section 4.6.

Active
If an Active route is not timed (i.e., its ExpirationTime is MAX_TIME), it becomes Idle if not used within ACTIVE_INTERVAL. A timed route (i.e., a route with ExpirationTime not equal to MAX_TIME) remains Active until its expiration time, after which it MAY either be expunged or marked as Invalid.
Idle
An Idle route becomes Active if it is used to forward a packet.
If not used, an Idle route remains idle for MAX_IDLETIME before
becoming an Invalid route.

Invalid
An Invalid route MAY be maintained until MAX_SEQNUM_LIFETIME after
the last sequence number update. This allows incoming information
to be assessed for freshness. After this time it should be
expunged.

Unconfirmed
An Unconfirmed route becomes Idle when adjacency with the next hop
router is confirmed, or will be expunged if the neighbor is
blacklisted, or at MAX_SEQNUM_LIFETIME after the last sequence
number update.

In all cases, if the time since the route’s last sequence number
update exceeds MAX_SEQNUM_LIFETIME, the sequence number must be
removed from the route. If the route is Invalid or Unconfirmed at
this time, it MUST be expunged. Active and Idle routes can continue
to be used to forward packets. The removal of the sequence number is
required to ensure that any AODVv2 routers following the
initialization procedure can safely begin routing functions using a
new sequence number.

Appendix D.1.1 contains an algorithmic representation of this timeout
behavior.

Routes can become Invalid before a timeout occurs:

- If a link breaks, all routes using that link as the next hop MUST
  immediately be marked as Invalid.

- If a Route Error (RERR) message containing the route is received
  from the route’s next hop, the route MUST immediately be marked as
  Invalid.

When an Unconfirmed route is set as Idle as a result of the adjacency
with Route.NextHop being Confirmed (see Section 4.3), any inferior
matching routes MUST be expunged.

Memory constrained devices MAY choose to expunge routes from the
AODVv2 route table before their expiration time, but MUST adhere to
the following rules:

- An Active route MUST NOT be expunged.
An Idle route SHOULD NOT be expunged.

Any Invalid route MAY be expunged; least recently used Invalid routes SHOULD be expunged first.

An Unconfirmed route MUST NOT be expunged if it was installed within the last 
RREQ_WAIT_TIME. Otherwise, it MAY be expunged.

6.7.2. Reporting Invalid Routes

When an Active route becomes Invalid as a result of a broken link or a received Route Error (RERR) message, other routers should be informed by sending a RERR message containing details of the invalidated route.

A RERR message should also be sent when an AODVv2 router receives a packet to forward on behalf of another router but does not have a valid route for the destination of the packet. This packet may be a data packet or, in rare cases, a RREP message, if the route to the request originator has been lost. The packet or message triggering the RERR MUST be discarded.

Generation of a RERR message is described in Section 7.4.1.

7. AODVv2 Protocol Messages

AODVv2 defines four message types: Route Request (RREQ), Route Reply (RREP), Route Reply Acknowledgement (RREP_Ack), and Route Error (RERR).

Each AODVv2 message is defined as a set of data elements. Rules for the generation, reception and regeneration of each message type are described in the following sections. Section 8 discusses how the data elements map to RFC 5444 Message TLVs, Address Blocks, and Address TLVs.

7.1. Route Request (RREQ) Message

Route Request messages are used in route discovery operations to request a route to a specified target address. RREQ messages have the following general structure:
Figure 1: RREQ message structure

RREQ Data Elements

**msg_hop_limit**
- The remaining number of hops allowed for dissemination of the RREQ message.

**msg_hop_count**
- The number of hops already traversed during dissemination of the RREQ message.

**AddressList**
- Contains OrigAddr and TargAddr, the source and destination addresses of the packet for which a route is requested. OrigAddr and TargAddr MUST be routable unicast addresses.

**PrefixLengthList**
- Contains OrigPrefixLen, i.e., the length, in bits, of the prefix associated with OrigAddr. If omitted, the prefix length is equal to OrigAddr’s address length in bits. If OrigAddr resides on a subnet configured as a Router Client, the prefix length represents the number of bits in the subnet mask.

**OrigSeqNum**
- The sequence number associated with OrigAddr.

**TargSeqNum**
- A sequence number associated with TargAddr. This may be included if an Invalid route exists to the target. This is useful for the optional Intermediate RREP feature (see Section 10.3).
MetricType
The type of metric associated with OrigMetric. This element
 can be omitted if the default metric type is used.

OrigMetric
The metric associated with the route to OrigAddr, as measured
by the sender of the message.

ValidityTime
The length of time that the message sender is willing to offer
a route toward OrigAddr. Omitted if no time limit is imposed.

7.1.1.  RREQ Generation

A RREQ is generated when a packet needs to be forwarded for a Router
Client, and no valid route currently exists for the packet’s
destination.

Before creating a RREQ, the router should check if a RREQ has
recently been sent for the requested destination. If so, and the
wait time for a reply has not yet been reached, the router should
continue to await a response without generating a new RREQ. If the
timeout has been reached, a new RREQ may be generated. If buffering
is configured, the incoming packet SHOULD be buffered until the route
discovery is completed.

If the limit for the rate of AODVv2 control message generation has
been reached, no message should be generated.

To generate the RREQ, the router (referred to as RREQ_Gen) follows
this procedure:

1.  Set msg_hop_limit := MAX_HOPCOUNT
2.  Set msg_hop_count := 0, if including it
3.  Set AddressList := {OrigAddr, TargAddr}
4.  For the PrefixLengthList:
   *  If OrigAddr resides on a Router Client subnet, set
      PrefixLengthList := {OrigPrefixLen, null}.
   *  Otherwise, omit PrefixLengthList.
5.  For OrigSeqNum:
* Increment the SeqNum associated with OrigAddr as specified in
  Section 4.4.

* Set OrigSeqNum := SeqNum.

6. For TargSeqNum:

* If an Invalid route exists matching TargAddr using longest
  prefix matching and has a valid sequence number, set
  TargSeqNum := route’s sequence number.

* If no Invalid route exists matching TargAddr, or the route
  doesn’t have a sequence number, omit TargSeqNum.

7. Include the MetricType data element if requesting a route for a
   non-default metric type, and set the type accordingly

8. Set OrigMetric := Route[OrigAddr].Metric

9. Include the ValidityTime data element if advertising that the
   route to OrigAddr via this router is offered for a limited time,
   and set ValidityTime accordingly

This AODVv2 message is used to create a corresponding RFC 5444
message (see Section 8) which is multicast, by default, to LL-MANET-
Routers on all interfaces configured for AODVv2 operation.

7.1.2. RREQ Reception

Upon receiving a RREQ, an AODVv2 router performs the following steps:

1. If the sender is blacklisted (Section 4.3), check the entry’s
   reset time

   * If CurrentTime < Remove Time, ignore this RREQ for further
     processing.

   * If CurrentTime >= Remove Time, reset the neighbor state to
     Unknown and continue to Step 2.

2. Verify that the message hop count, if included, hasn’t exceeded
   MAX_HOPCOUNT

   * If so, ignore this RREQ for further processing.

3. Verify that the message contains the required data elements:
   msg_hop_limit, OrigAddr, TargAddr, OrigSeqNum, and OrigMetric,
and that OrigAddr and TargAddr are valid addresses (routable and unicast)

* If not, ignore this RREQ for further processing.

4. If the MetricType data element is present, check that the metric type is known

* If not, ignore this RREQ for further processing.

5. Verify that the cost of the advertised route will not exceed the maximum allowed metric value for the metric type (Metric <= MAX_METRIC[MetricType] - Cost(L))

* If it will, ignore this RREQ for further processing.

6. Process the route to OrigAddr as specified in Section 6.5.1

7. Check if the message is redundant by comparing to entries in the RteMsg table (Section 6.6)

* If redundant, ignore this RREQ for further processing.

* If not redundant, save the information in the RteMsg table to identify future duplicates and continue processing.

8. Check if the TargAddr belongs to one of the Router Clients

* If so, generate a RREP as specified in Section 7.2.1.

* If not, continue to RREQ regeneration.

7.1.3. RREQ Regeneration

By regenerating a RREQ, a router advertises that it will forward packets to the OrigAddr contained in the RREQ according to the information enclosed. The router MAY choose not to regenerate the RREQ, though this could decrease connectivity in the network or result in non-optimal paths. The full set of circumstances under which a router may avoid regenerating a RREQ are not declared in this document, though examples include the router being heavily loaded or low on energy and therefore unwilling to advertise routing capability for more traffic.

The RREQ should not be regenerated if the limit for the rate of AODVv2 control message generation has been reached.

The procedure for RREQ regeneration is as follows:
1. Set \( \text{msg\_hop\_limit} := \text{received msg\_hop\_limit} - 1 \)

2. If \( \text{msg\_hop\_limit} \) is now zero, do not continue the regeneration process

3. Set \( \text{msg\_hop\_count} := \text{received msg\_hop\_count} + 1 \), if included, otherwise omit \( \text{msg\_hop\_count} \)

4. Set AddressList, PrefixLengthList, sequence numbers and MetricType to the values in the received RREQ

5. Set \( \text{OrigMetric} := \text{Route[OrigAddr].Metric} \)

6. If the received RREQ contains a ValidityTime, or if the regenerating router wishes to limit the time that it offers a route to OrigAddr, the regenerated RREQ MUST include a ValidityTime data element

   * The ValidityTime is either the ValidityTime the previous AODVv2 router specified, or the ValidityTime this router wishes to impose, whichever is lower.

This AODVv2 message is used to create a corresponding RFC 5444 message (see Section 8) which is multicast, by default, to LL-MANET-Routers on all interfaces configured for AODVv2 operation. However, the regenerated RREQ can be unicast to the next hop address of the route toward TargAddr, if known.

7.2. Route Reply (RREP) Message

A Route Reply message is sent in response to a Route Request message and offers a route to the Target Address in the RREQ.

The RREP is sent by unicast to the next hop router on the route to OrigAddr, if there is a Confirmed entry in the Neighbor Table for the next hop. Otherwise, the RREP is sent multicast to LL-MANET-Routers, including the AckReq data element in the message to indicate the intended next hop address and request acknowledgement to confirm the neighbor adjacency.

RREP messages have the following general structure:
Figure 2: RREP message structure

RREP Data Elements

msg_hop_limit
The remaining number of hops allowed for dissemination of the RREP message.

msg_hop_count
The number of hops already traversed during dissemination of the RREP message.

AckReq
The address of the intended next hop of the RREP. This data element is used when the RREP is multicast because the next hop toward OrigAddr is a neighbor with Unknown state. It indicates that an acknowledgement to the RREP is requested by the sender from the intended next hop (see Section 6.2).

AddressList
Contains OrigAddr and TargAddr, the source and destination addresses of the packet for which a route is requested. OrigAddr and TargAddr MUST be routable unicast addresses.

PrefixLengthList
Contains TargPrefixLen, i.e., the length, in bits, of the prefix associated with TargAddr. If omitted, the prefix length is equal to TargAddr’s address length, in bits. If TargAddr resides on a subnet configured as a Router Client, the prefix length represents the number of bits in the subnet mask.
TargSeqNum
The sequence number associated with TargAddr.

MetricType
The type of metric associated with TargMetric. This element can be omitted if the default metric type is used.

TargMetric
The metric associated with the route to TargAddr, as seen from the sender of the message.

ValidityTime
The length of time that the message sender is willing to offer a route toward TargAddr. Omitted if no time limit is imposed.

7.2.1. RREP Generation

A RREP is generated when a RREQ arrives for one of the AODVv2 router’s Router Clients.

Before creating a RREP, the router should check if the corresponding RREQ is redundant, i.e., a response has already been generated, or if the limit for the rate of AODVv2 control message generation has been reached. If so, the RREP should not be created.

If the next hop neighbor on the route to OrigAddr is not yet confirmed as adjacent (as described in Section 6.2), the RREP MUST include an AckReq data element including the intended next hop address, in order to perform adjacency monitoring. If the adjacency is already confirmed, it can be omitted. The AckReq data element indicates that an acknowledgement to the RREP is requested from the intended next hop router in the form of a Route Reply Acknowledgement (RREP_Ack).

Implementations may allow a number of retries of the RREP if an acknowledgement is not received within RREP_Ack_SENT_TIMEOUT, doubling the timeout with each retry, up to a maximum of RREP_RETRIES, using the same exponential backoff described in Section 6.4 for RREQ retries. Adjacency confirmation MUST be considered to have failed after the wait time for a RREP_Ack response to the final RREP. The next hop router MUST be marked as blacklisted (Section 4.3), and any installed routes with next hop set to the newly blacklisted router should become Invalid.

To generate the RREP, the router (also referred to as RREP_Gen) follows this procedure:
1. Set msg_hop_limit := msg_hop_count from the received RREQ message, if it was included, or MAX_HOPCOUNT if it was not included.

2. Set msg_hop_count := 0, if including it.

3. If adjacency with the next hop toward OrigAddr is not already confirmed, include the AckReq data element with the address of the intended next hop router.

4. Set Address List := (OrigAddr, TargAddr).

5. For the PrefixLengthList:
   * If TargAddr resides on a Router Client subnet, set PrefixLengthList := (null, TargPrefixLen).
   * Otherwise, omit PrefixLengthList.

6. For the TargSeqNum:
   * Increment the SeqNum associated with TargAddr as specified in Section 4.4.
   * Set TargSeqNum := SeqNum.

7. Include the MetricType data element if the route requested is for a non-default metric type, and set the type accordingly.


9. Include the ValidityTime data element if advertising that the route to TargAddr via this router is offered for a limited time, and set ValidityTime accordingly.

This AODVv2 message is used to create a corresponding RFC 5444 message (see Section 8). If there is a Confirmed entry in the Neighbor Table for the next hop router on the route to OrigAddr, the RREP is sent by unicast to the next hop. Otherwise, the RREP is sent multicast to LL-MANET-Routers.

### 7.2.2. RREP Reception

Upon receiving a RREP, an AODVv2 router performs the following steps:

1. If the sender is blacklisted (Section 4.3), but the RREP answers a recently sent RREQ, the sender state should be set to Confirmed since a RREP is an indication of adjacency.
2. Verify that the message hop count, if included, hasn’t exceeded MAX_HOPCOUNT
   * If so, ignore this RREQ for further processing.

3. Verify that the message contains the required data elements: msg_hop_limit, OrigAddr, TargAddr, TargSeqNum, and TargMetric, and that OrigAddr and TargAddr are valid addresses (routable and unicast)
   * If not, ignore this RREP for further processing.

4. If the MetricType data element is present, check that the metric type is known
   * If not, ignore this RREP for further processing.

5. Verify that the cost of the advertised route will not exceed the maximum allowed metric value for the metric type (Metric <= MAX_Metric[MetricType] - Cost(L))
   * If it will, ignore this RREP for further processing.

6. If the AckReq data element is present, check the intended recipient of the received RREP
   * If the receiving router is the intended recipient, send an acknowledgement as specified in Section 7.3 and continue processing.
   * If the receiving router is not the intended recipient, ignore this RREP for further processing.

7. Process the route to TargAddr as specified in Section 6.5.1
   * If the route to TargAddr fulfills a previously sent RREQ, any associated timeouts will be cancelled and buffered packets will be forwarded to TargAddr, but processing continues to Step 8.

8. Check if the message is redundant by comparing to entries in the RteMsg table (Section 6.6)
   * If redundant, ignore this RREP for further processing.
   * If not redundant, save the information in the RteMsg table to identify future redundant RREP messages and continue processing.
9. Check if the OrigAddr belongs to one of the Router Clients
   * If so, no further processing is necessary.

10. Check if a valid (Active or Idle) or Unconfirmed route exists to
    OrigAddr
    * If so, continue to RREP regeneration.
    * If not, a Route Error message SHOULD be transmitted to
      TargAddr according to Section 7.4.1 and the RREP should
      be discarded and not regenerated.

7.2.3. RREP Regeneration

A received Route Reply message is regenerated toward OrigAddr. Unless the
router is prepared to advertise the route contained within the received RREP,
it halts processing. By regenerating a RREP, a router advertises that it will
forward packets to TargAddr according to the information enclosed. The router
MAY choose not to regenerate the RREP, in the same way it may choose not to
regenerate a RREQ (see Section 7.1.3), though this could decrease connectivity
in the network or result in non-optimal paths.

The RREP should not be regenerated if the limit for the rate of
AODVv2 control message generation has been reached.

If the next hop neighbor on the route to OrigAddr is not yet
confirmed as adjacent (as described in Section 6.2), the RREP MUST
include an AckReq data element including the intended next hop
address, in order to perform adjacency monitoring. If the adjacency
is already confirmed, the AckReq data element can be omitted. The
AckReq data element indicates that an acknowledgement to the RREP is
requested in the form of a Route Reply Acknowledgement (RREP_Ack)
from the intended next hop router.

The procedure for RREP regeneration is as follows:

1. Set msg_hop_limit := received msg_hop_limit - 1

2. If msg_hop_limit is now zero, do not continue the regeneration process

3. Set msg_hop_count := received msg_hop_count + 1, if it was included, otherwise omit msg_hop_count
4. If an adjacency with the next hop toward OrigAddr is not already confirmed, include the AckReq data element with the address of the intended next hop router

5. Set AddressList, PrefixLengthList, TargSeqNum and MetricType to the values in the received RREP

6. Set TargMetric := Route[TargAddr].Metric

7. If the received RREP contains a ValidityTime, or if the regenerating router wishes to limit the time that it will offer a route to TargAddr, the regenerated RREP MUST include a ValidityTime data element

   * The ValidityTime is either the ValidityTime the previous AODVv2 router specified, or the ValidityTime this router wishes to impose, whichever is lower.

This AODVv2 message is used to create a corresponding RFC 5444 message (see Section 8). If there is a Confirmed entry in the Neighbor Table for the next hop router on the route to OrigAddr, the RREP is sent by unicast to the next hop. Otherwise, the RREP is sent multicast to LL-MANET-Routers.

7.3. Route Reply Acknowledgement (RREP_Ack) Message

The Route Reply Acknowledgement MUST be sent in response to a received Route Reply which includes an AckReq data element with an address matching one of the receiving router’s IP addresses. When the RREP_Ack message is received, it confirms the adjacency between the two routers. The RREP_Ack has no data elements.

7.3.1. RREP_Ack Generation

A RREP_Ack MUST be generated when a received RREP includes the AckReq data element with the address of the receiving router. The RREP_Ack should not be generated if the limit for the rate of AODVv2 control message generation has been reached.

There are no data elements in a RREP_Ack. The RFC 5444 representation is discussed in Section 8. The RREP_Ack is unicast, by default, to the IP address of the router that requested it.

7.3.2. RREP_Ack Reception

Upon receiving a RREP_Ack, an AODVv2 router performs the following steps:
1. If a RREP_Ack message was expected from the IP source address of the RREP_Ack, the router cancels any associated timeouts.

2. If the RREP_Ack was expected, ensure the router sending the RREP_Ack is marked with state Confirmed in the Neighbor Table (Section 4.3).

7.4. Route Error (RERR) Message

A Route Error message is generated by an AODVv2 router to notify other AODVv2 routers of routes that are no longer available. A RERR message has the following general structure:

```
+-----------------------------------------------------------------+
|                          msg_hop_limit                          |
+-----------------------------------------------------------------+
|                       PktSource (optional)                      |
+-----------------------------------------------------------------+
|                           AddressList                           |
+-----------------------------------------------------------------+
|                   PrefixLengthList (optional)                   |
+-----------------------------------------------------------------+
|                       SeqNumList (optional)                     |
+-----------------------------------------------------------------+
|                     MetricTypeList (optional)                   |
+-----------------------------------------------------------------+
```

Figure 3: RERR message structure

**RERR Data Elements**

**msg_hop_limit**

The remaining number of hops allowed for dissemination of the RERR message.

**PktSource**

The source IP address of the packet triggering the RERR. If the RERR is triggered by a broken link, the PktSource data element is not required.

**AddressList**

The addresses of the routes no longer available through RERR_Gen.

**PrefixLengthList**

The prefix lengths, in bits, associated with the routes no longer available through RERR_Gen, indicating whether a route represents a single device or a subnet.
SeqNumList
The sequence numbers of the routes no longer available through RERR_Gen (where known).

MetricTypeList
The types of metric associated with the routes no longer available through RERR_Gen. This element can be omitted if all routes use the default metric type.

7.4.1. RERR Generation

A RERR is generated when an AODVv2 router (also referred to as RERR_Gen) needs to report that a destination is no longer reachable. There are two events that cause this response:

- If a packet arrives that cannot be forwarded because no valid route exists for its destination, the RERR generated MUST contain the PktSource data element and will contain only one unreachable address. The contents of PktSource and AddressList depend on the packet that triggered the RERR:
  - If the packet is a data packet forwarded by another AODVv2 router, PktSource is set to the source IP address of the packet, and the AddressList contains the destination IP address of the packet.
  - If the packet contains a RREP message and the route to OrigAddr has been lost, PktSource is set to the TargAddr of the RREP, and the AddressList contains the OrigAddr from the RREP.

The prefix length and sequence number MAY be included if known from an Invalid route entry to the destination of the packet. The MetricTypeList MAY also be included if a MetricType can be determined from the packet itself, or if an Invalid route exists for the packet’s destination and the metric type is not DEFAULT_METRIC_TYPE.

RERR_Gen MUST discard the packet or message that triggered generation of the RERR.

In order to avoid flooding the network with RERR messages when a stream of packets to an unreachable address arrives, an AODVv2 router SHOULD determine whether a RERR has recently been sent with the same unreachable address and PktSource, and SHOULD avoid creating duplicate RERR messages.

- When a link breaks, multiple routes may become Invalid, and the RERR generated MAY contain multiple unreachable addresses. If the
message contents would cause the MTU to be exceeded, multiple RERR messages must be sent. The RERR MUST include the MetricTypeList data element when it contains routes which do not use the DEFAULT_METRIC_TYPE. The PktSource data element is omitted.

All previously Active routes that used the broken link MUST be reported. The AddressList, PrefixLengthList, SeqNumList, and MetricTypeList will contain entries for each route which has become Invalid.

A RERR message is only sent if an Active route becomes Invalid, though an AODVv2 router can also include Idle routes that become Invalid if the configuration parameter ENABLE_IDLE_IN_RERR is set (see Section 11.3).

Incidentally, if an AODVv2 router receives an ICMP error packet to or from the address of one of its Router Clients, it simply forwards the ICMP packet in the same way as any other data packet, and will not generate any RERR message based on the contents of the ICMP packet.

The RERR should not be generated if the limit for the rate of AODVv2 control message generation has been reached.

To generate the RERR, the router follows this procedure:

1. Set msg_hop_limit := MAX_HOPCOUNT

2. If necessary, include the PktSource data element and set the value to the source address of the packet triggering the RERR

3. For each route that needs to be reported, while respecting the interface MTU:
   * Insert the route address into the AddressList.
   * Insert the prefix length into PrefixLengthList, if known and not equal to the address length.
   * Insert the sequence number into SeqNumList, if known.
   * Insert the metric type into MetricTypeList, if known and not equal to DEFAULT_METRIC_TYPE.

4. If interface MTU would be exceeded, create additional RERR messages

The AODVv2 message is used to create a corresponding RFC 5444 message (see Section 8).
If the RERR is sent in response to an undeliverable packet or message, it SHOULD be sent unicast to the next hop on the route to PktSource, or alternatively it MUST be multicast to LL-MANET-Routers.

If the RERR is sent in response to a broken link, the RERR is, by default, multicast to LL-MANET-Routers.

If the optional precursor lists feature (see Section 10.2) is enabled, the RERR is unicast to the precursors of the routes being reported.

7.4.2. RERR Reception

Upon receiving a RERR, an AODVv2 router performs the following steps:

1. Verify that the message contains the required data elements:
   msg_hop_limit and at least one unreachable address

   * If not, ignore this RREP for further processing.

2. For each address in the AddressList, check that:

   * The address is valid (routable and unicast)

   * The MetricType, if present, is known (assume DEFAULT_METRIC_TYPE if not present)

   * There is a valid route with the same MetricType matching the address using longest prefix matching

   * Either the route’s next hop is the sender of the RERR and route’s next hop interface is the interface on which the RERR was received, or PktSource is present in the RERR and is a Router Client address

   * The unreachable address’ sequence number is either unknown, or is greater than the route’s sequence number

   If any of the above are false, the route does not need to be made Invalid and the unreachable address does not need to be advertised in a regenerated RERR.

   If all of the above are true:

   * If the route’s prefix length is the same as the unreachable address’ prefix length, set the route state to Invalid, and note that the route should be advertised in a regenerated RERR.
* If the prefix length is shorter than the original route, the route MUST be expunged from the routing table, since it is a sub-route of the larger route which is reported to be Invalid.

* If the prefix length is different, create a new route with the unreachable address, and its prefix and sequence number, set the state to Invalid, and note that the route should be advertised in a regenerated RERR.

* Update the sequence number on the stored route, if the reported sequence number is greater.

3. If PktSource is included and is a Router Client, do not regenerate the RERR.

4. Check if there are unreachable addresses which need to be advertised in a regenerated RERR

   * If so, regenerate the RERR as detailed in Section 7.4.3.

   * If not, take no further action.

7.4.3. RERR Regeneration

The RERR should not be generated if the limit for the rate of AODVv2 control message generation has been reached.

The procedure for RERR regeneration is as follows:

1. Set msg_hop_limit := received msg_hop_limit − 1

2. If msg_hop_limit is now zero, do not continue the regeneration process

3. If the PktSource data element was included in the original RERR, copy it into the regenerated RERR

4. For each route that needs to be reported, while respecting the interface MTU:

   * Insert the unreachable address into the AddressList.

   * Insert the prefix length into PrefixLengthList, if known and not equal to the address length.

   * Insert the sequence number into SeqNumList, if known.
* Insert the MetricType into MetricTypeList if known, and not equal to DEFAULT_METRIC_TYPE.

5. If interface MTU would be exceeded, create additional RERR messages

The AODVv2 message is used to create a corresponding RFC 5444 message (see Section 8). If the RERR contains the PktSource data element, the regenerated RERR SHOULD be sent unicast to the next hop on the route to PktSource, or alternatively it MUST be multicast to LL-MANET-Routers. If the RERR is sent in response to a broken link, the RERR is, by default, multicast to LL-MANET-Routers.

8. RFC 5444 Representation

AODVv2 specifies that all control plane messages between routers SHOULD use the Generalized Mobile Ad Hoc Network Packet/Message Format [RFC5444], and therefore AODVv2 defines route messages comprising data elements that map to message elements in RFC 5444.

RFC 5444 provides a multiplexed transport for multiple protocols. An RFC 5444 multiplexer may choose to optimize the content of certain message elements to reduce control plane overhead.

A brief summary of the RFC 5444 format:

1. A packet contains zero or more messages

2. A message contains a Message Header, one Message TLV Block, zero or more Address Blocks, and one Address Block TLV Block per Address Block

3. The Message TLV Block MAY contain zero or more Message TLVs

4. An Address Block TLV Block MAY include zero or more Address Block TLVs

5. Each TLV value in an Address Block TLV Block can be associated with all of the addresses, a contiguous set of addresses, or a single address in the Address Block

AODVv2 does not require access to the RFC 5444 packet header.

In the message header, AODVv2 uses <msg-hop-limit>, <msg-hop-count>, <msg-type> and <msg-addr-length>. <msg-addr-length> indicates the length of any addresses in the message (using <msg-addr-length> := address length in octets - 1, i.e. 3 for IPv4 and 15 for IPv6).
Each address included in the Address Block is identified as OrigAddr, TargAddr, PktSource, or Unreachable Address by including an ADDRESS_TYPE TLV in the Address Block TLV Block.

The addresses in an Address Block may appear in any order, and values in a TLV in the Address Block TLV Block must be associated with the correct address in the Address Block. To indicate which value is associated with each address, the AODVv2 message representation uses lists where the order of the addresses in the AODVv2 AddressList data element matches the order of values in other list-based data elements, e.g., the order of SeqNums in the SeqNumList in a RERR.

The following sections show how AODVv2 data elements are represented in RFC 5444 messages. See Section 12 for more information about the Message TLVs and Address Block TLVs AODVv2 defines, and the type numbers allocated.

Where the extension type of a TLV is set to zero, this is the default RFC 5444 value and the extension type will not be included in the message.

8.1. RREQ

8.1.1. Message Header

<table>
<thead>
<tr>
<th>Data Element</th>
<th>Header Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>&lt;msg-type&gt;</td>
<td>RREQ</td>
</tr>
<tr>
<td>msg_hop_limit</td>
<td>&lt;msg-hop-limit&gt;</td>
<td>MAX_HOPCOUNT</td>
</tr>
<tr>
<td>msg_hop_count</td>
<td>&lt;msg-hop-count&gt;</td>
<td>Number of hops traversed so far by the message.</td>
</tr>
</tbody>
</table>

8.1.2. Message TLV Block

A RREQ contains no Message TLVs.

8.1.3. Address Block

A RREQ contains two Addresses, OrigAddr and TargAddr, and each address has an associated prefix length. If the prefix length has not been included in the AODVv2 message, it is equal to the address length in bits.
8.1.4. Address Block TLV Block

Address Block TLVs are always associated with addresses in the Address Block. The following sections show the TLVs that apply to each address.

8.1.4.1. Address Block TLVs for OrigAddr

<table>
<thead>
<tr>
<th>Data Element</th>
<th>TLV Type</th>
<th>Extension Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>ADDRESS_TYPE</td>
<td>0</td>
<td>ADDRTYPE_ORIGADDR</td>
</tr>
<tr>
<td>OrigSeqNum</td>
<td>SEQ_NUM</td>
<td>0</td>
<td>Sequence Number of RREQ_Gen, the router which initiated route discovery.</td>
</tr>
<tr>
<td>OrigMetric</td>
<td>PATH_METRIC</td>
<td>MetricType</td>
<td>Metric for the route to OrigAddr, using MetricType.</td>
</tr>
<tr>
<td>ValidityTime</td>
<td>VALIDITY_TIME</td>
<td>0</td>
<td>ValidityTime for route to OrigAddr.</td>
</tr>
</tbody>
</table>

In the AODVv2 representation, if the message relates to DEFAULT_METRIC_TYPE, MetricType is not included in the message. The RFC 5444 representation will set the extension type in the PATH_METRIC TLV to 0. AODVv2 interprets a MetricType of 0 as DEFAULT_METRIC_TYPE.

8.1.4.2. Address Block TLVs for TargAddr

<table>
<thead>
<tr>
<th>Data Element</th>
<th>TLV Type</th>
<th>Extension Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>ADDRESS_TYPE</td>
<td>0</td>
<td>ADDRTYPE_TARGADDR</td>
</tr>
<tr>
<td>TargSeqNum</td>
<td>SEQ_NUM</td>
<td>0</td>
<td>The last known TargSeqNum for TargAddr.</td>
</tr>
</tbody>
</table>
8.2. RREP

8.2.1. Message Header

<table>
<thead>
<tr>
<th>Data Element</th>
<th>Header Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>&lt;msg-type&gt;</td>
<td>RREP</td>
</tr>
<tr>
<td>msg_hop_limit</td>
<td>&lt;msg-hop-limit&gt;</td>
<td>&lt;msg-hop-count&gt; from corresponding RREQ.</td>
</tr>
<tr>
<td>msg_hop_count</td>
<td>&lt;msg-hop-count&gt;</td>
<td>Number of hops traversed so far by the message.</td>
</tr>
</tbody>
</table>

8.2.2. Message TLV Block

A RREP contains no Message TLVs.

8.2.3. Address Block

A RREP contains a minimum of two Addresses, OrigAddr and TargAddr, and each address has an associated prefix length. If the prefix length has not been included in the AODVv2 message, it is equal to the address length in bits.

It may also contain the address of the intended next hop, in order to request acknowledgement to confirm adjacency, as described in Section 6.2. The prefix length associated with this address is equal to the address length in bits.

<table>
<thead>
<tr>
<th>Data Elements</th>
<th>Address Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>OrigAddr/OrigPrefixLen</td>
<td>&lt;address&gt; + &lt;prefix-length&gt;</td>
</tr>
<tr>
<td>TargAddr/TargPrefixLen</td>
<td>&lt;address&gt; + &lt;prefix-length&gt;</td>
</tr>
<tr>
<td>AckReq</td>
<td>&lt;address&gt; + &lt;prefix-length&gt;</td>
</tr>
</tbody>
</table>

8.2.4. Address Block TLV Block

Address Block TLVs are always associated with addresses in the Address Block. The following sections show the TLVs that apply to each address.
### 8.2.4.1. Address Block TLVs for OrigAddr

<table>
<thead>
<tr>
<th>Data Element</th>
<th>TLV Type</th>
<th>Extension Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>ADDRESS_TYPE</td>
<td>0</td>
<td>ADDRTYPE_ORIGADDR</td>
</tr>
</tbody>
</table>

### 8.2.4.2. Address Block TLVs for TargAddr

<table>
<thead>
<tr>
<th>Data Element</th>
<th>TLV Type</th>
<th>Extension Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>ADDRESS_TYPE</td>
<td>0</td>
<td>ADDRTYPE_TARGADDR</td>
</tr>
<tr>
<td>TargSeqNum</td>
<td>SEQ_NUM</td>
<td>0</td>
<td>Sequence number of RREP_Gen, the router which created the RREP.</td>
</tr>
<tr>
<td>TargMetric</td>
<td>PATH_METRIC</td>
<td>MetricType</td>
<td>Metric for the route to TargAddr, using MetricType.</td>
</tr>
<tr>
<td>ValidityTime</td>
<td>VALIDITY_TIME</td>
<td>0</td>
<td>ValidityTime for route to TargAddr.</td>
</tr>
</tbody>
</table>

In the AODVv2 representation, if the message relates to DEFAULT_METRIC_TYPE, MetricType is not included in the message. The RFC 5444 representation will set the extension type in the PATH_METRIC TLV to 0. AODVv2 interprets a MetricType of 0 as DEFAULT_METRIC_TYPE.

### 8.2.4.3. Address Block TLVs for AckReq Intended Recipient Address

<table>
<thead>
<tr>
<th>Data Element</th>
<th>TLV Type</th>
<th>Extension Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>ADDRESS_TYPE</td>
<td>0</td>
<td>ADDRTYPE_INTEND</td>
</tr>
</tbody>
</table>

### 8.3. RREP_Ack

#### 8.3.1. Message Header
8.3.2. Message TLV Block

A RREP_Ack contains no Message TLVs.

8.3.3. Address Block

A RREP_Ack contains no Address Block.

8.3.4. Address Block TLV Block

A RREP_Ack contains no Address Block TLV Block.

8.4. RERR

8.4.1. Message Header

<table>
<thead>
<tr>
<th>Data Element</th>
<th>Header Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>&lt;msg-type&gt;</td>
<td>RERR</td>
</tr>
<tr>
<td>msg_hop_limit</td>
<td>&lt;msg-hop-limit&gt;</td>
<td>MAX_HOPCOUNT</td>
</tr>
</tbody>
</table>

8.4.2. Message TLV Block

A RERR contains no Message TLVs.

8.4.3. Address Block

The Address Block in a RERR may contain PktSource, the source IP address of the packet triggering RERR generation, as detailed in Section 7.4. Prefix Length associated with PktSource is equal to the address length in bits.

Address Block always contains one Address per route that is no longer valid, and each address has an associated prefix length. If a prefix length has not been included for this address, it is equal to the address length in bits.
### 8.4.4. Address Block TLV Block

Address Block TLVs are always associated with addresses in the Address Block. The following sections show the TLVs that apply to each type of address in the RERR.

#### 8.4.4.1. Address Block TLVs for PktSource

<table>
<thead>
<tr>
<th>Data Element</th>
<th>TLV Type</th>
<th>Extension Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PktSource</td>
<td>ADDRESS_TYPE</td>
<td>0</td>
<td>ADDRTYPE_PKTSOURCE</td>
</tr>
</tbody>
</table>

#### 8.4.4.2. Address Block TLVs for Unreachable Addresses

<table>
<thead>
<tr>
<th>Data Element</th>
<th>TLV Type</th>
<th>Extension Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>ADDRESS_TYPE</td>
<td>0</td>
<td>ADDRTYPE_UNREACHABLE</td>
</tr>
<tr>
<td>SeqNumList</td>
<td>SEQ_NUM</td>
<td>0</td>
<td>Sequence Number associated with invalid route to the unreachable address.</td>
</tr>
<tr>
<td>MetricTypeList</td>
<td>PATH_METRIC</td>
<td>MetricType</td>
<td>None. Extension Type set to MetricType of the route to the unreachable address.</td>
</tr>
</tbody>
</table>

Using the PATH_METRIC TLV without a value is a mechanism used in RERR messages to indicate the MetricType associated with the route being reported, without the need to include a Metric value. Multiple PATH_METRIC TLVs may be necessary if routes with multiple MetricTypes are included in the RERR.
In the AODVv2 representation, if the RERR message includes only routes with DEFAULT_METRIC_TYPE, MetricType is not included in the message. In this case, the RFC 5444 representation does not need to include a PATH_METRIC TLV to indicate the DEFAULT_METRIC_TYPE. If the RERR message includes both routes with DEFAULT_METRIC_TYPE and other MetricTypes, only the routes with non-default MetricType need to be marked with a PATH_METRIC TLV using Extension Type to indicate MetricType. AODVv2 interprets the absence of MetricType information as an indication of DEFAULT_METRIC_TYPE.

9. Simple Internet Attachment

Figure 4 shows a stub (i.e., non-transit) network of AODVv2 routers which is attached to the Internet via a single Internet AODVv2 Router (abbreviated IAR). The interface to the Internet MUST NOT be configured in the AODVv2_INTERFACES list.

As in any Internet-attached network, AODVv2 routers and clients that wish to be reachable from hosts on the Internet MUST have IP addresses within the IAR’s routable and topologically correct prefix (i.e., 191.0.2.0/24). This AODVv2 network and subnets within it will be advertised to the Internet using procedures which are out of scope for this specification.

```
/-------------------------\
/ +----------------+        \
/  |  AODVv2 Router |         \ 
/  |  191.0.2.2/32  |         \ Routable
/  +----------------+         \ Prefix
\  | Internet        |/191.0.2.0/24
\  | AODVv2 Router |\  
\  | 191.0.2.1  |\ ---+-+-------+ Internet \
\  | serving net  |\    \  
\  | 191.0.2.0/24 |\     / 
\  +-------------+\   / 
\  | AODVv2 Router |\ 
\  | 191.0.2.3/32 |\ 
\  \-----------------+ 
\ 
Figure 4: 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, i.e., destinations not on the configured 191.0.2.0/24 subnet.

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 exact destination it will perform normal AODVv2 route discovery for that destination.

Configuring the IAR as a default router is outside the scope of this specification.

10. Optional Features

A number of optional features for AODVv2, associated initially with AODV, MAY be useful in networks with greater mobility or larger node populations, or networks requiring reduced latency for application launches. These features are not required by minimal implementations.

10.1. 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.

10.2. Precursor Lists

This section specifies an interoperable enhancement to AODVv2 enabling more economical RERR notifications.

There can be several sources of traffic for a certain destination. Each source of traffic and each upstream router between the forwarding AODVv2 router and the traffic source is known as a "precursor" for the destination. For each destination, an AODVv2 router MAY choose to keep track of precursors that have provided traffic for that destination. Route Error messages about that destination can be sent unicast to these precursors instead of multicast to all AODVv2 routers.

Since a RERR will be regenerated if it comes from a next hop on a valid route, the RERR should ideally be sent backwards along the route that the source of the traffic uses, to ensure it is regenerated at each hop and reaches the traffic source. If the reverse path is unknown, the RERR should be sent toward the source along some other route. Therefore, the options for saving precursor information are as follows:
- Save the next hop on an existing route to the packet’s source address as the precursor. In this case, it is not guaranteed that a RERR that is sent will follow the reverse of the source’s route. In rare situations, this may prevent the route from being invalidated at the source of the data traffic.

- Save the packet’s source address as the precursor. In this case, the RERR can be sent along any existing route to the source of the data traffic, and should include the PktSource data element to ensure that the route will be invalidated at the source of the traffic, in case the RERR does not follow the reverse of the source’s route.

- By inspecting the MAC address of each forwarded packet, determine which router forwarded the packet, and save the router address as a precursor. This ensures that when a RERR is sent to the precursor router, the route will be invalidated at that router, and the RERR will be regenerated toward the source of the packet.

During normal operation, each AODVv2 router maintaining precursor lists for a route must update the precursor list whenever it uses this route to forward traffic to the destination. Precursors are classified as Active if traffic has recently been forwarded by the precursor. The precursor is marked with a timestamp to indicate the time it last forwarded traffic on this route.

When an AODVv2 router detects that one or more routes are broken, it MAY notify each Active precursor using a unicast Route Error message instead of creating multicast traffic. Unicast is applicable when there are few Active precursors compared to the number of neighboring AODVv2 routers. However, the default multicast behavior is still preferable when there are many precursors, since fewer message transmissions are required.

When an AODVv2 router supporting precursor lists receives a RERR message, it MAY identify the list of its own affected Active precursors for the routes in the RERR, and choose to send a unicast RERR to those, rather than send a multicast RERR.

When a route is expunged, any precursor list associated with it must also be expunged.

### 10.3. Intermediate RREP

Without iRREP, only the AODVv2 router responsible for the target address can respond to a RREQ. Using iRREP, route discoveries can be faster and create less control traffic. This specification has been published as a separate Internet Draft [I-D.perkins-irrep].
10.4. Message Aggregation Delay

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].

11. Configuration

AODVv2 uses various parameters which can be grouped into the following categories:

- Timers
- Protocol constants
- Administrative parameters and controls

This section shows the parameters along with their definitions and default values (if any).

Note that several fields have limited size (bits or bytes). These sizes and their encoding may place specific limitations on the values that can be set.

11.1. Timers

AODVv2 requires certain timing information to be associated with route table entries and message replies. The default values are as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTIVE_INTERVAL</td>
<td>5 second</td>
</tr>
<tr>
<td>MAX_IDLETIME</td>
<td>200 seconds</td>
</tr>
<tr>
<td>MAX_BLACKLIST_TIME</td>
<td>200 seconds</td>
</tr>
<tr>
<td>MAX_SEQNUM_LIFETIME</td>
<td>300 seconds</td>
</tr>
<tr>
<td>RteMsg_ENTRY_TIME</td>
<td>12 seconds</td>
</tr>
<tr>
<td>RREQ_WAIT_TIME</td>
<td>2 seconds</td>
</tr>
<tr>
<td>RREP_Ack_SENT_TIMEOUT</td>
<td>1 second</td>
</tr>
<tr>
<td>RREQ_HOLDDOWN_TIME</td>
<td>10 seconds</td>
</tr>
</tbody>
</table>

Table 3: 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. Ideally, for networks with frequent topology changes the AODVv2 parameters should be adjusted using experimentally determined values or dynamic adaptation. For example, in networks with infrequent topology changes MAX_IDLETIME may be set to a much larger value.

11.2. 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 section describing their use.

<table>
<thead>
<tr>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISCOVERY_ATTEMPTS_MAX</td>
<td>3</td>
<td>Section 6.4</td>
</tr>
<tr>
<td>RREP_RETRIES</td>
<td>2</td>
<td>Section 7.2.1</td>
</tr>
<tr>
<td>MAX_METRIC[MetricType]</td>
<td>[TBD]</td>
<td>Section 5</td>
</tr>
<tr>
<td>MAX_METRIC[HopCount]</td>
<td>20 hops</td>
<td>Section 5 and Section 7</td>
</tr>
<tr>
<td>MAX_HOPCOUNT</td>
<td>20</td>
<td>Same as MAX_METRIC[HopCount]</td>
</tr>
<tr>
<td>MAX_TIME</td>
<td>[TBD]</td>
<td>Maximum expressible clock time</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Section 6.5.2)</td>
</tr>
</tbody>
</table>

Table 4: AODVv2 Constants

Note that <msg-hop-count> is an 8-bit field in the RFC 5444 message header and therefore MAX_HOPCOUNT cannot be larger than 255. Field lengths associated with metrics are to be found in Section 12.3.

MAX_METRIC[MetricType] MUST always be the maximum expressible metric of type MetricType.

These protocol constants MUST have the same values for all AODVv2 routers in the ad hoc network. If the values were configured differently, the following consequences may be observed:

- DISCOVERY_ATTEMPTS_MAX: Nodes with higher values are likely to be more successful at finding routes, at the cost of additional control traffic.

- RREP_RETRIES: Nodes with lower values are more likely to blacklist neighbors when there is a temporary fluctuation in link quality.

- MAX_HOPCOUNT: Nodes with a value too small would not be able to discover routes to distant addresses.
The following table lists AODVv2 parameters which should be administratively configured for each node:

<table>
<thead>
<tr>
<th>Name</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AODVv2_INTERFACE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BUFFER_SIZE_PACKETS</td>
<td>2</td>
<td>Section 3</td>
</tr>
<tr>
<td>BUFFER_SIZE_BYTES</td>
<td>MAX_PACKET_SIZE [TBD]</td>
<td>Section 6.4</td>
</tr>
<tr>
<td>CLIENT_ADDRESSES</td>
<td>AODVv2_INTERFACE</td>
<td>Section 4.2</td>
</tr>
<tr>
<td>CONTROL_TRAFFIC_LIMIT</td>
<td>[TBD - 50 pkts/sec?]</td>
<td>Section 7</td>
</tr>
</tbody>
</table>

Table 5: Configuration for Local Settings

The following administrative controls may be used to change the operation of the network. The same settings should be used across the network. Inconsistent settings at different nodes in the network will not result in protocol errors, but poor performance may result, especially if metrics are misinterpreted because DEFAULT_METRIC_TYPE is configured differently at different nodes.

<table>
<thead>
<tr>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT_METRIC_TYPE</td>
<td>3 (i.e., Hop Count)</td>
<td>[RFC6551]</td>
</tr>
<tr>
<td>ENABLE_IDLE_IN_RERR</td>
<td>Disabled</td>
<td>Section 7.4.1</td>
</tr>
</tbody>
</table>

Table 6: Configuration for Network-Wide Settings

These options are not required for correct routing behavior, although they may reduce AODVv2 protocol overhead in certain situations. The default behavior is to leave these options disabled.
Table 7: Configuration for Optional Features

12. IANA Considerations

This section specifies several RFC 5444 message types, message tlv-types, and address tlv-types required for AODVv2. Also, a new registry of 16-bit metric types is specified.

12.1. RFC 5444 Message Types

<table>
<thead>
<tr>
<th>Name of Message</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Route Request (RREQ)</td>
<td>10 (TBD)</td>
</tr>
<tr>
<td>Route Reply (RREP)</td>
<td>11 (TBD)</td>
</tr>
<tr>
<td>Route Error (RERR)</td>
<td>12 (TBD)</td>
</tr>
<tr>
<td>Route Reply Acknowledgement (RREP_Ack)</td>
<td>13 (TBD)</td>
</tr>
</tbody>
</table>

Table 8: AODVv2 Message Types

12.2. RFC 5444 Address Block TLV Types

<table>
<thead>
<tr>
<th>Name of TLV</th>
<th>Type</th>
<th>Length (octets)</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>PATH_METRIC</td>
<td>10 (TBD)</td>
<td>depends on MetricType</td>
<td>Section 7</td>
</tr>
<tr>
<td>SEQ_NUM</td>
<td>11 (TBD)</td>
<td>2</td>
<td>Section 7</td>
</tr>
<tr>
<td>ADDRESS_TYPE</td>
<td>15 (TBD)</td>
<td>1</td>
<td>Section 8</td>
</tr>
<tr>
<td>VALIDITY_TIME</td>
<td>1 (TBD)</td>
<td>1</td>
<td>[RFC5497]</td>
</tr>
</tbody>
</table>

Table 9: AODVv2 Address Block TLV Types
12.3. MetricType Allocation

Metric types are identified according to the assignments in [RFC6551].

<table>
<thead>
<tr>
<th>Name of MetricType</th>
<th>Type</th>
<th>Metric Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unassigned</td>
<td>0</td>
<td>Undefined</td>
</tr>
<tr>
<td>Currently Unsupported</td>
<td>1 - 2</td>
<td>TBD</td>
</tr>
<tr>
<td>Hop Count</td>
<td>3 [TBD]</td>
<td>1 octet</td>
</tr>
<tr>
<td>Currently Unsupported</td>
<td>4 - 8</td>
<td>TBD</td>
</tr>
<tr>
<td>Unallocated</td>
<td>9 - 254</td>
<td>TBD</td>
</tr>
<tr>
<td>Reserved</td>
<td>255</td>
<td>Undefined</td>
</tr>
</tbody>
</table>

Table 10: AODVv2 Metric Types

When creating AODVv2 messages which relate to the DEFAULT_METRIC_TYPE, MetricType is not reported in the message. In the RFC 5444 message representation, the PATH_METRIC TLV, if included, will not include an extension type. While RFC 5444 would interpret the lack of an extension type value as indication that extension type is zero, AODVv2 will interpret an extension type of zero to mean the DEFAULT_METRIC_TYPE configured on the router. This is possible because zero is not assigned to any metric type ([RFC6551]). In RERR, the absence of the PATH_METRIC TLV also indicates use of the DEFAULT_METRIC_TYPE.

12.4. AddressType Allocation

The values used in the Address Type TLV used in Section 8 are given in the table below:

<table>
<thead>
<tr>
<th>Address Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDRTYPE_ORIGADDR</td>
<td>0</td>
</tr>
<tr>
<td>ADDRTYPE_TARGADDR</td>
<td>1</td>
</tr>
<tr>
<td>ADDRTYPE_UNREACHABLE</td>
<td>2</td>
</tr>
<tr>
<td>ADDRTYPE_PKTSOURCE</td>
<td>3</td>
</tr>
<tr>
<td>ADDRTYPE_INTEND</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 11: AODVv2 Address Types
13. Security Considerations

This section describes various security considerations and potential avenues to secure AODVv2 routing. The objective of the AODVv2 protocol is for each router to communicate reachability information about addresses for which it is responsible, and for routes it has learned from other AODVv2 routers. 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 RERR messages. 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.

Networks using AODVv2 to maintain connectivity and establish routes on demand may be vulnerable to certain well-known types of threats. Flooding attacks using RREQ amount to a denial of service for route discovery. Valid route table entries can be replaced by maliciously constructed RREQ and RREP messages. Links could be erroneously treated as bidirectional if malicious unsolicited RREP or RREP_Ack messages were to be accepted. Replay attacks using RERR messages could, in some circumstances, be used to disrupt active routes. Passive inspection of AODVv2 control messages could enable unauthorized devices to gain information about the network topology, since exchanging such information is the main purpose of AODVv2.

The on-demand nature of AODVv2 route discovery reduces the vulnerability to route disruption. Since control traffic for updating route tables is diminished, there is less opportunity for failure. Processing requirements for AODVv2 are typically quite small, and would typically be dominated by calculations to verify integrity. This has the effect of reducing (but by no means eliminating) AODVv2’s vulnerability to denial of service attacks.

Encryption MAY be used for AODVv2 messages. If the routers share a packet-level security association, the message data can be encrypted prior to message transmission. The establishment of such security associations is outside the scope of this specification. Encryption will not only protect against unauthorized devices obtaining information about network topology but will ensure that only trusted routers participate in routing operations.

Message integrity checking is enabled by the Integrity Check Value mechanisms defined in [RFC7182]. The data contained in AODVv2 routing protocol messages SHOULD be verified using ICV values, to avoid the use of message data if the message has been tampered with or replayed. Otherwise, it would be possible to disrupt communications by injecting nonexistent or malicious routes into the
route tables of nodes within the ad hoc network. This can result in loss of data or message processing by unauthorized devices.

The remainder of this section provides specific recommendations for the use of the integrity checking and timestamp functions defined in [RFC7182] to ensure the integrity of each AODVv2 message. The calculation used for the Integrity Check Value will depend on the message type. Sequence numbers can be used as timestamps to protect against replay, since they are known to be strictly increasing.

RREQ messages advertise a route to OrigAddr, and impose very little processing requirement for receivers. The main threat presented by sending a RREQ message with false information is that traffic to OrigAddr could be disrupted. Since RREQ is multicast and likely to be received by all nodes in the ad hoc network, this threat could have serious impact on applications communicating by way of OrigAddr. The actual threat to disrupt routes to OrigAddr is reduced by the AODVv2 mechanism of marking RREQ-derived routes as "Unconfirmed" until adjacency with the next hop is confirmed. If AODVv2 routers always verify the integrity of the RREQ message data, then the threat of disruption is minimized. The ICV mechanisms offered in [RFC7182] are sufficient for this purpose. Since OrigAddr is included as a data element of the RREQ, the ICV can be calculated and verified using message contents. The ICV should be verified at every step along the dispersal path of the RREQ to mitigate the threat. Since RREQ_Gen’s sequence number is incremented for each new RREQ, replay protection is already afforded and no extra timestamp mechanism is required.

RREP messages advertise a route to TargAddr, and impose very little processing requirement for receivers. The main threat presented by sending a RREP message with false information is that traffic to TargAddr could be disrupted. Since RREP is unicast, this threat is restricted to receivers along the path from OrigAddr to TargAddr. If AODVv2 routers always verify the integrity of the RREP message data, then this threat is minimized. This facility is offered by the ICV mechanisms in [RFC7182]. Since TargAddr is included as a data element of the RREP, the ICV can be calculated and verified using message contents. The ICV should be verified at every step along the unicast path of the RREP. Since RREP_Gen’s sequence number is incremented for each new RREP, replay protection is afforded and no extra timestamp mechanism is required.

RREP_Ack messages are intended to verify bidirectional neighbor connectivity, and impose very little processing requirement for receivers. The main threat presented by sending a RREP_Ack message with false information is that the route advertised to a target node in a RREP might be erroneously accepted even though the route would
contain a unidirectional link and thus not be suitable for most traffic. Since RREP_Ack is unicast, this threat is strictly local to the RREP transmitter expecting the acknowledgement. A malicious router could also attempt to send an unsolicited RREP_Ack to convince another router that a bidirectional link exists and subsequently use further messages to divert traffic along a route which is not valid. If AODVv2 routers always verify the integrity of the RREP_Ack message data, then this threat is minimized. This facility is offered by the ICV mechanisms in [RFC7182]. The RREP_Gen SHOULD use the source IP address of the RREP_Ack to identify the sender, and so the ICV should be calculated using the message contents and the IP source address. The message must also include the Timestamp defined in [RFC7182] to protect against replay attacks, using TargSeqNum from the RREP as the value in the TIMESTAMP TLV.

RERR messages remove routes, and impose very little processing requirement for receivers. The main threat presented by sending a RERR message with false information is that traffic to the advertised destinations could be disrupted. Since RERR is multicast and can be received by many routers in the ad hoc network, this threat could have serious impact on applications communicating by way of the sender of the RERR message. However, since the sender of the RERR message with erroneous information may be presumed to be either malicious or broken, it is better that such routes not be used anyway. Another threat is that a malicious RERR message may be sent with a PktSource data element included, to disrupt PktSource’s ability to send to the addresses contained in the RERR. If AODVv2 routers always verify the integrity of the RERR message data, then this threat is reduced. This facility is offered by the ICV mechanisms in [RFC7182]. The receiver of the RERR SHOULD use the source IP address of the RERR to identify the sender. The message must also include the Timestamp defined in [RFC7182] to protect against replay attacks, using SeqNum from RERR_Gen as the value in the TIMESTAMP TLV.

14. 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 and Ian Chakeres for their long time authorship of AODV. Additional thanks to Derek Atkins, Emmanuel Baccelli, Abdussalam Baryun, Ramon Caceres, Thomas Clausen, Justin Dean, Christopher Dearlove, Ulrich Herberg, Henner Jakob, Luke Klein-Berndt, Lars Kristensen, Tronje Krop, Koojana Kuladinithi, Kedar Namjoshi, Keyur Patel, Alexandru Petrescu, Henning Rogge, Fransisco Ros, Pedro Ruiz, Christoph Sommer, Romain Thouvenin, Richard Trefler, Jiazi Yi, Seung Yi, and Cong Yuan, for
their reviews AODVv2 and DYMO, as well as numerous specification suggestions.

15. References

15.1. Normative References


15.2. Informative References


Appendix A. Features Required of IP

AODVv2 needs the following:

- information that IP routes are requested
- information that packets are flowing
A reactive protocol reacts when a route is needed. A route is requested when an application tries to send a packet. The fundamental concept of reactive routing is to avoid creating routes that are not needed. The trigger for route discovery is an application trying to send a packet. If a route is not available to forward the packet, the packet is queued while the route is requested.

Appendix B. Multi-homing Considerations

Multi-homing is not supported by the AODVv2 specification. The coordination between multiple AODVv2 routers to distribute routing information correctly for a shared address is not defined.

Previous work indicates 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. Without this, comparing sequence numbers would not work to evaluate freshness. Even when the IP address is included, there is no good way to compare sequence numbers from different IP addresses, but 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 necessary for reactive protocol operation.

Appendix C. Router Client Relocation

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, i.e., it must wait at least MAX_SEQNUM_LIFETIME after the previous AODVv2 router’s last SeqNum update for this network prefix.

Appendix D. Example Algorithms for AODVv2 Operations

The following subsections show example algorithms for protocol operations required by AODVv2. AODVv2 requires general algorithms for manipulating and comparing table entries, and algorithms specific to each message type.
Processing for 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.

The following table indicates the field names used in subsequent sections to describe the AODVv2 algorithms.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RteMsg</td>
<td>A route message (inRREQ/outRREQ/inRREP/outRREP)</td>
</tr>
<tr>
<td>RteMsg.HopLimit</td>
<td>Hop limit for the message</td>
</tr>
<tr>
<td>RteMsg.HopCount</td>
<td>Hop count for the message</td>
</tr>
<tr>
<td>RteMsg.AckReq</td>
<td>True/False, optional in RREP</td>
</tr>
<tr>
<td>RteMsg.MetricType</td>
<td>The type of metric included, optional</td>
</tr>
<tr>
<td>RteMsg.OrigAddr</td>
<td>Address of source of queued data</td>
</tr>
<tr>
<td>RteMsg.TargAddr</td>
<td>Address route is requested for</td>
</tr>
<tr>
<td>RteMsg.OrigPrefixLen</td>
<td>Prefix length of OrigAddr, optional</td>
</tr>
<tr>
<td>RteMsg.TargPrefixLen</td>
<td>Prefix length of TargAddr, optional</td>
</tr>
<tr>
<td>RteMsg.OrigSeqNum</td>
<td>SeqNum of OrigAddr, in RREQ only</td>
</tr>
<tr>
<td>RteMsg.TargSeqNum</td>
<td>SeqNum of TargAddr, in RREP, optional in RREQ</td>
</tr>
<tr>
<td>RteMsg.OrigMetric</td>
<td>Metric to OrigAddr, in RREQ only</td>
</tr>
<tr>
<td>RteMsg.TargMetric</td>
<td>Metric to TargAddr, in RREP only</td>
</tr>
<tr>
<td>RteMsg.ValidityTime</td>
<td>Time limit for route advertised</td>
</tr>
<tr>
<td>RteMsg.NbrIP</td>
<td>Sender of the RteMsg</td>
</tr>
<tr>
<td>RteMsg.Netif</td>
<td>Interface on which the RteMsg arrived</td>
</tr>
<tr>
<td>AdvRte</td>
<td>Derived from a RteMsg (see Section 6.5)</td>
</tr>
<tr>
<td>AdvRte.Address</td>
<td>Route destination address</td>
</tr>
<tr>
<td>AdvRte.PrefixLength</td>
<td>Route destination prefix length</td>
</tr>
<tr>
<td>AdvRte.SeqNum</td>
<td>SeqNum associated with route</td>
</tr>
<tr>
<td>AdvRte.MetricType</td>
<td>MetricType associated with route</td>
</tr>
<tr>
<td>AdvRte.Metric</td>
<td>Advertised metric of route</td>
</tr>
<tr>
<td>AdvRte.Cost</td>
<td>Cost from receiving router</td>
</tr>
<tr>
<td>AdvRte.ValidityTime</td>
<td>Time limit for route advertised</td>
</tr>
<tr>
<td>AdvRte.NextHopIP</td>
<td>Sender of the RteMsg</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>AdvRte.NextHopIntf</td>
<td>Interface on which the RteMsg arrived</td>
</tr>
<tr>
<td>AdvRte.HopCount</td>
<td>Number of hops traversed</td>
</tr>
<tr>
<td>AdvRte.HopLimit</td>
<td>Allowed number of hops remaining</td>
</tr>
<tr>
<td>Route</td>
<td>A route table entry (see Section 4.6)</td>
</tr>
<tr>
<td>Route.Address</td>
<td>Route destination address</td>
</tr>
<tr>
<td>Route.PrefixLength</td>
<td>Route destination prefix length</td>
</tr>
<tr>
<td>Route.SeqNum</td>
<td>SeqNum associated with route</td>
</tr>
<tr>
<td>Route.NextHop</td>
<td>Address of router which advertised the route</td>
</tr>
<tr>
<td>Route.NextHopInterface</td>
<td>Interface on which next hop is reachable</td>
</tr>
<tr>
<td>Route.LastUsed</td>
<td>Time this route was last used for packet forwarding</td>
</tr>
<tr>
<td>Route.LastSeqNumUpdate</td>
<td>Time the SeqNum of the route was last updated</td>
</tr>
<tr>
<td>Route.ExpirationTime</td>
<td>Time at which the route will expire</td>
</tr>
<tr>
<td>Route.State</td>
<td>Active/Idle/Invalid</td>
</tr>
<tr>
<td>Route.Precursors</td>
<td>Optional (see Section 10.2)</td>
</tr>
<tr>
<td>RERR</td>
<td>Route Error message (inRERR/outRERR)</td>
</tr>
<tr>
<td>RERR.HopLimit</td>
<td>Hop limit for the message</td>
</tr>
<tr>
<td>RERR.PktSource</td>
<td>Source address of packet which triggered RERR</td>
</tr>
<tr>
<td>RERR.AddressList[]</td>
<td>List of unreachable route addresses</td>
</tr>
<tr>
<td>RERR.PrefixLengthList[]</td>
<td>List of PrefixLengths for AddressList</td>
</tr>
<tr>
<td>RERR.SeqNumList[]</td>
<td>List of SeqNums for AddressList</td>
</tr>
<tr>
<td>RERR.MetricTypeList[]</td>
<td>MetricType for the invalid routes</td>
</tr>
<tr>
<td>RERR.Netif</td>
<td>Interface on which the RERR arrived</td>
</tr>
</tbody>
</table>

Table 12: Notation used in Appendix

D.1. General Operations

D.1.1. Check_Route_State
/* Update the state of the route entry based on timeouts. Return whether the route can be used for forwarding a packet. */

Check_Route_State(route)
{
    if (CurrentTime > route.ExpirationTime)
        route.State := Invalid;
    if (((CurrentTime - route.LastUsed > ACTIVE_INTERVAL + MAX_IDLETIME)
        AND (route.State != Unconfirmed)
        AND (route.ExpirationTime == MAX_TIME)) //not a timed route
        route.State := Invalid;
    if (((CurrentTime - route.LastUsed > ACTIVE_INTERVAL)
        AND (route.State != Unconfirmed)
        AND (route.ExpirationTime == MAX_TIME)) //not a timed route
        route.State := Idle;
    if ((CurrentTime - route.LastSeqNumUpdate > MAX_SEQNUM_LIFETIME)
        AND (route.State == Invalid OR route.State == Unconfirmed)) /* remove route from route table */
        if ((CurrentTime - route.LastSeqNumUpdate > MAX_SEQNUM_LIFETIME)
            AND (route.State != Invalid)
            route.SeqNum := 0;
    if (route still exists AND route.State != Invalid
        AND Route.State != Unconfirmed)
        return TRUE;
    else
        return FALSE;
}

D.1.2. Process_Routing_Info

(See Section 6.5.1)
/* Compare incoming route information to stored route, and if better, 
use to update stored route. */

Process_Routing_Info (advRte)
{
    rte := Fetch_Route_Table_Entry (advRte);
    if (!rte exists)
    {
        rte := Create_Route_Table_Entry(advRte);
        return rte;
    }

    if (AdvRte.SeqNum > Route.SeqNum        /* stored route is stale */
    OR
    (AdvRte.SeqNum == Route.SeqNum         /* same SeqNum */
    AND
    ((Route.State == Invalid AND LoopFree(advRte, rte))
    /* advRte can repair stored */
    OR AdvRte.Cost < Route.Metric)))       /* advRte is better */
    {
        if (advRte is from a RREQ)
            rte := Create_Route_Table_Entry(advRte);
        else
            Update_Route_Table_Entry (rte, advRte);
    }
    return rte;
}

D.1.3. Fetch_Route_Table_Entry
/* Lookup a route table entry matching an advertised route */

Fetch_Route_Table_Entry (advRte) {
    foreach (rteTableEntry in rteTable) {
        if (rteTableEntry.Address == advRte.Address AND rteTableEntry.MetricType == advRte.MetricType) return rteTableEntry;
    }
    return null;
}

/* Lookup a route table entry matching address and metric type */

Fetch_Route_Table_Entry (destination, metricType) {
    foreach (rteTableEntry in rteTable) {
        if (rteTableEntry.Address == destination AND rteTableEntry.MetricType == metricType) return rteTableEntry;
    }
    return null;
}

D.1.4. Update_Route_Table_Entry

/* Update a route table entry using AdvRte in received RteMsg */

Update_Route_Table_Entry (rte, advRte); {
    rte.SeqNum := advRte.SeqNum;
    rte.NextHop := advRte.NextHopIp;
    rte.NextHopInterface := advRte.NextHopIntf;
    rte.LastUsed := CurrentTime;
    rte.LastSeqNumUpdate := CurrentTime;
    else rte.ExpirationTime := MAX_TIME;

    rte.Metric := advRte.Cost;
    if (rte.State == Invalid)
        rte.State := Idle (if advRte is from RREP); or Unconfirmed (if advRte is from RREQ);
D.1.5. Create_Route_Table_Entry

/* Create a route table entry from address and prefix length */

Create_Route_Table_Entry (address, prefixLength, seqNum, metricType)
{
    rte := allocate_memory();
    rte.Address := address;
    rte.PrefixLength := prefixLength;
    rte.SeqNum := seqNum;
    rte.MetricType := metricType;
}

/* Create a route table entry from the advertised route */

Create_Route_Table_Entry(advRte)
{
    rte := allocate_memory();
    rte.Address := advRte.Address;
    if (advRte.PrefixLength)
        rte.PrefixLength := advRte.PrefixLength;
    else
        rte.PrefixLength := maxPrefixLenForAddressFamily;
    rte.SeqNum := advRte.SeqNum;
    rte.NextHop := advRte.NextHopIp;
    rte.NextHopInterface := advRte.NextHopIntf;
    rte.LastUsed := CurrentTime;
    rte.LastSeqNumUpdate := CurrentTime;
    if (validityTime)
    else
        rte.ExpirationTime := MAX_TIME;
    rte.MetricType := advRte.MetricType;
    rte.Metric := advRte.Metric;
    rte.State := Idle (if advRte is from RREP);
        or Unconfirmed (if advRte is from RREQ);
}

D.1.6. LoopFree
/* Return TRUE if the route advRte is LoopFree compared to rte */

LoopFree(advRte, rte)
{
    if (advRte.Cost <= rte.Cost)
        return TRUE;
    else
        return FALSE;
}

D.1.7. Fetch_Rte_Msg_Table_Entry

/* Find an entry in the RteMsg table matching the given message’s msg-type, OrigAddr, TargAddr, MetricType */

Fetch_Rte_Msg_Table_Entry (rteMsg)
{
    foreach (entry in RteMsgTable)
    {
        if (entry.msg-type == rteMsg.msg-type
            AND entry.OrigAddr == rteMsg.OrigAddr
            AND entry.TargAddr == rteMsg.TargAddr
            AND entry.MetricType == rteMsg.MetricType)
            return entry;
    }
    return NULL;
}

D.1.8. Update_Rte_Msg_Table
(See Section 4.5)

/* Update the multicast route message suppression table based on the received RteMsg, return true if it was created or the SeqNum was updated (i.e. it needs to be regenerated) */

Update_Rte_Msg_Table(rteMsg)
{
    /* search for a comparable entry */
    entry := Fetch_Rte_Msg_Table_Entry(rteMsg);

    /* if there is none, create one */
    if (entry does not exist)
    {
        entry.MessageType := rteMsg.msg_type;
        entry.OrigAddr := rteMsg.OrigAddr;
        entry.TargAddr := rteMsg.TargAddr;
        entry.OrigSeqNum := rteMsg.origSeqNum; // (if present)
entry.TargSeqNum := rteMsg.targSeqNum; // (if present)
entry.MetricType := rteMsg.MetricType;
entry.Metric := rteMsg.OrigMetric; // (for RREQ)
or rteMsg.TargMetric; // (for RREP)
entry.Timestamp := CurrentTime;
return TRUE;

/* if current entry is stale */
if (
    (rteMsg.msg-type == RREQ AND entry.OrigSeqNum < rteMsg.OrigSeqNum)
    OR
    (rteMsg.msg-type == RREP AND entry.TargSeqNum < rteMsg.TargSeqNum))
{
    entry.OrigSeqNum := rteMsg.OrigSeqNum; // (if present)
    entry.TargSeqNum := rteMsg.TargSeqNum; // (if present)
    entry.Timestamp := CurrentTime;
    return TRUE;
}

/* if received rteMsg is stale */
if (
    (rteMsg.msg-type == RREQ AND entry.OrigSeqNum > rteMsg.OrigSeqNum)
    OR
    (rteMsg.msg-type == RREP AND entry.TargSeqNum > rteMsg.TargSeqNum))
{
    entry.Timestamp := CurrentTime;
    return FALSE;
}

/* if same SeqNum but rteMsg has lower metric */
if (entry.Metric > rteMsg.Metric)
    entry.Metric := rteMsg.Metric;
entry.Timestamp := CurrentTime;
return FALSE;

D.1.9.  Build_RFC_5444_Message_Header
/* This pseudocode shows possible RFC 5444 actions, and would not
be performed by the AODVv2 implementation. It is shown only to
provide more understanding about the AODVv2 message that will be
constructed by RFC 5444.
MAL := Message Address Length
MF := Message Flags
Size := number of octets inMsgHdr, AddrBlk, AddrTLVs */

Build_RFC_5444_Message_Header (msgType, Flags, AddrFamily, Size,
hopLimit, hopCount, tlvLength)
{
    /* Build RFC 5444 message header fields */
    msg-type := msgType;
    MF := Flags;
    MAL := 3 or 15; // for IPv4 or IPv6
    msg-size := Size;
    msg-hop-limit := hopLimit;
    if (hopCount != 0) /* if hopCount is 0, do not include */
        msg-hop-count := hopCount;
    msg.tlvs-length := tlvLength;
}

D.2. RREQ Operations

D.2.1. Generate_RREQ

/* Generate a route request message to find a route from OrigAddr
to TargAddr using the given MetricType
origAddr := IP address of Router Client which generated the
packet to be forwarded
origPrefix := prefix length associated with the Router Client
targAddr := destination IP address in the packet to be forwarded
targSeqNum := sequence number in existing route to targAddr
mType := metric type for the requested route */

Generate_RREQ(origAddr, origPrefix, targAddr, targSeqNum, mType)
{
    /* Increment sequence number in nonvolatile storage */
    mySeqNum := (1 + mySeqNum);

    /* Marshall parameters */
    outRREQ.HopLimit := MAX_HOPCOUNT;
    outRREQ.HopCount := 0; // if included
    outRREQ.MetricType := mType; // include if not DEFAULT_METRIC_TYPE
    outRREQ.OrigAddr := origAddr;
    outRREQ.TargAddr := targAddr;
    outRREQ.OrigPrefixLen := origPrefix; // include if not address length
    outRREQ.OrigSeqNum := mySeqNum;

outRREQ.TargSeqNum := targSeqNum;            //included if available
outRREQ.OrigMetric := Route[OrigAddr].Metric;       //zero by default
outRREQ.ValidityTime := limit for route to OrigAddr;   //if required

/* Build Address Blk using prefix length information from
   outRREQ.OrigPrefixLen if necessary */
AddrBlk := {outRREQ.OrigAddr, outRREQ.TargAddr};

/* Include sequence numbers in appropriate Address Block TLVs */
/* OrigSeqNum Address Block TLV */
origSeqNumAddrBlkTlv.value := outRREQ.OrigSeqNum;
/* TargSeqNum Address Block TLV */
if (outRREQ.TargSeqNum is known)
   targSeqNumAddrBlkTlv.value := outRREQ.TargSeqNum;

/* Build Metric Address Block TLV, include Metric AddrBlkTlv
   Extension type if a non-default metric */
metricAddrBlkTlv.value := outRREQ.OrigMetric;
if (outRREQ.MetricType != DEFAULT_METRIC_TYPE)
   metricAddrBlkTlv.typeExtension := outRREQ.MetricType;
if (outRREQ.ValidityTime is required)
{
   /* Build VALIDITY_TIME Address Block TLV */
   VALIDITY_TIMEAddrBlkTlv.value := outRREQ.ValidityTime;
}

Build_RFC_5444_Message_Header (RREQ, 4, IPv4 or IPv6, NN,
   outRREQ.HopLimit, outRREQ.HopCount, tlvLength);

/* multicast RFC 5444 message to LL-MANET-Routers */

D.2.2.  Receive_RREQ
/* Process a RREQ received on link L */

Receive_RREQ (inRREQ, L)
{
    if (inRREQ.NbrIP present in blacklist)
    {
        if (blacklist_expiration_time < CurrentTime)
            return; // don’t process or regenerate RREQ
        else
            remove nbrIP from blacklist;
    }
    if (inRREQ does not contain msg_hop_limit, OrigAddr, TargAddr, OrigSeqNum, OrigMetric)
        return;
    if (inRREQ.OrigAddr and inRREQ.TargAddr are not valid routable and unicast addresses)
        return;
    if (inRREQ.MetricType is present but an unknown value)
        return;
    if (inRREQ.OrigMetric > MAX_METRIC[inRREQ.MetricType] - Cost(L))
        return;

    /* Extract inRREQ values */
    advRte.Address := inRREQ.OrigAddr;
    advRte.PrefixLength := inRREQ.OrigPrefixLen; (if present)
        or the address length of advRte.Address;
    advRte.SeqNum := inRREQ.OrigSeqNum;
    advRte.MetricType := inRREQ.MetricType;
    advRte.Metric := inRREQ.OrigMetric;
    advRte.Cost := inRREQ.OrigMetric + Cost(L); //according to the indicated MetricType
    advRte.ValidityTime := inRREQ.ValidityTime; //if present
    advRte.NextHopIP := inRREQ.NbrIP;
    advRte.NextHopIntf := inRREQ.Netif;
    advRte.HopCount := inRREQ.HopCount;
    advRte.HopLimit := inRREQ.HopLimit;

    rte := Process_Routing_Info (advRte);

    /* Update the RteMsgTable and determine if the RREQ needs to be regenerated */
    regenerate := Update_Rte_Msg_Table(inRREQ);

    if (inRREQ.TargAddr is in Router Client list)
        Generate_RREP(inRREQ, rte);
    else if (regenerate)
        Regenerate_RREQ(inRREQ, rte);
}
D.2.3. Regenerate_RREQ

/* Called from receive_RREQ()
   rte := the route to OrigAddr */

Regenerate_RREQ (inRREQ, rte)
{
    outRREQ.HopLimit := inRREQ.HopLimit - 1;
    if (outRREQ.HopLimit == 0)
        return; // don’t regenerate

    if (inRREQ.HopCount exists)
    {
        if (inRREQ.HopCount >= MAX_HOPCOUNT)
            return; // don’t regenerate
        outRREQ.HopCount := inRREQ.HopCount + 1;
    }

    /* Marshall parameters */
    outRREQ.MetricType := rte.MetricType;
    outRREQ.OrigAddr := rte.Address;
    outRREQ.TargAddr := inRREQ.TargAddr;
    /* include prefix length if not equal to address length */
    outRREQ.OrigPrefixLen := rte.PrefixLength;
    outRREQ.OrigSeqNum := rte.SeqNum;
    outRREQ.TargSeqNum := inRREQ.TargSeqNum; // if present
    outRREQ.OrigMetric := rte.Metric;
    outRREQ.ValidityTime := rte.ValidityTime;
    or the time limit this router wishes to put on route to OrigAddr

    /* Build Address Block using prefix length information from outRREQ.OrigPrefixLen if necessary */
    AddrBlk := {outRREQ.OrigAddr, outRREQ.TargAddr};

    /* Include sequence numbers in appropriate Address Block TLVs */
    /* OrigSeqNum Address Block TLV */
    origSeqNumAddrBlkTlv.value := outRREQ.OrigSeqNum;
    /* TargSeqNum Address Block TLV */
    if (outRREQ.TargSeqNum is known)
        targSeqNumAddrBlkTlv.value := outRREQ.TargSeqNum;

    /* Build Metric Address Block TLV, include Metric AddrBlkTlv Extension type if a non-default metric */
    metricAddrBlkTlv.value := outRREQ.OrigMetric;
    if (outRREQ.MetricType != DEFAULT_METRIC_TYPE)
        metricAddrBlkTlv.typeExtension := outRREQ.MetricType;
if (outRREQ.ValidityTime is required)
{
    /* Build VALIDITY_TIME Address Block TLV */
    VALIDITY_TIMEAddrBlkTlv.value := outRREQ.ValidityTime;
}

Build_RFC_5444_Message_Header (RREQ, 4, IPv4 or IPv6, NN,
outRREQ.HopLimit, outRREQ.HopCount, tlvLength);

/* Multicast RFC 5444 message to LL-MANET-Routers, or if
inRREQ was unicast, the message can be unicast to the next
hop on the route to TargAddr, if known */

D.3. RREP Operations

D.3.1. Generate_RREP

Generate_RREP(inRREQ, rte)
{
    /* Increment sequence number in nonvolatile storage */
    mySeqNum := (1 + mySeqNum);

    /* Marshall parameters */
    outRREP.HopLimit := inRREQ.HopCount;
    outRREP.HopCount := 0;
    /* Include the AckReq when:
     - previous RREP does not seem to enable any data flow, OR
     - when RREQ is received from same OrigAddr after RREP was
       unicasted to rte.NextHop     */
    outRREP.AckReq := TRUE or FALSE; //TRUE if acknowledgement required
    /* if included, set timeout RREP_Ack_SENT_TIMEOUT */

    if (rte.MetricType != DEFAULT_METRIC_TYPE)
        outRREP.MetricType := rte.MetricType;
    outRREP.OrigAddr := rte.Address;
    outRREP.TargAddr := inRREQ.TargAddr;
    outRREP.TargPrefixLen := rte.PrefixLength; //if not address length
    outRREP.TargSeqNum := mySeqNum;
    outRREP.TargMetric := Route[TargAddr].Metric;
        //zero by default
    outRREP.ValidityTime := limit for route to TargAddr;   //if required

    if (outRREP.AckReq == TRUE)
        /* include AckReq Message TLV */

        /* Build Address Block using prefix length information from
         outRREP.TargPrefixLen if necessary */
    AddrBlk := {outRREP.OrigAddr, outRREP.TargAddr};
/* TargSeqNum Address Block TLV */
targSeqNumAddrBlkTlv.value := outRREP.TargSeqNum;

/* Build Metric Address Block TLV include Metric AddrBlkTlv 
   Extension type if a non-default metric */
metricAddrBlkTlv.value := outRREP.TargMetric;
if (outRREP.MetricType != DEFAULT_METRIC_TYPE)
    metricAddrBlkTlv.typeExtension := outRREP.MetricType;

if (outRREP.ValidityTime is required)
{
    if (outRREP.OrigAddr and inRREQ.TargAddr are not 
        valid routable and unicast addresses)
        return;
    if (inRREP.MetricType is present but an unknown value)
        return;
    if (inRREP.TargMetric > MAX_METRIC[outRREP.MetricType] - Cost(L))
        return;

    /* Extract inRREP values */
    advRte.Address := inRREP.TargAddr;

    /* Build VALIDITY_TIME Address Block TLV */
    VALIDITY_TIMEAddrBlkTlv.value := outRREP.ValidityTime;
}

Build_RFC_5444_Message_Header (RREP, 4, IPv4 or IPv6, NN,
    outRREP.HopLimit, outRREQ.HopCount, tlvLength);

/* unicast RFC 5444 message to rte[OrigAddr].NextHop */

D.3.2. Receive_RREP

Receive_RREP (inRREP, L)
{
    if (inRREP.NbrIP present in blacklist)
    {
        if (blacklist_expiration_time < CurrentTime)
            return;  // don’t process or regenerate RREP
        else
            remove NbrIP from blacklist;
    }

    if (inRREP does not contain msg_hop_limit, OrigAddr,
        TargAddr, TargSeqNum, TargMetric)
        return;
    if (inRREP.OrigAddr and inRREQ.TargAddr are not 
        valid routable and unicast addresses)
        return;
    if (inRREP.MetricType is present but an unknown value)
        return;
    if (inRREP.TargMetric > MAX_METRIC[inRREP.MetricType] - Cost(L))
        return;

    /* Extract inRREP values */
    advRte.Address := inRREP.TargAddr;
advRte.PrefixLength := inRREP.TargPrefixLen; //if present
or the address length of advRte.Address;
advRte.SeqNum := inRREP.TargSeqNum;
advRte.MetricType := inRREP.MetricType;
advRte.Metric := inRREP.TargMetric;
advRte.Cost := inRREP.TargMetric + Cost(L);
    //according to the indicated MetricType
advRte.ValidityTime := inRREP.ValidityTime; //if present
advRte.NextHopIP := inRREP.NbrIP;
advRte.NextHopIntf := inRREP.Netif;
advRte.HopCount := inRREP.HopCount;
advRte.HopLimit := inRREP.HopLimit; //if included

rte := Process_Routing_Info (advRte);

    '   if (inRREP includes AckReq data element)
        Generate_RREP_Ack(inRREP);

    /*  Update the RteMsgTable and determine if the RREP needs
        to be regenerated */
    regenerate := Update_Rte_Msg_Table(inRREP);

    if (inRREP.TargAddr is in the Router Client list)
        send_buffered_packets(rte); /* start to use the route */
    else if (regenerate)
        Regenerate_RREP(inRREP, rte);
}

D.3.3.  Regenerate_RREP

Regenerate_RREP(inRREP, rte)
{
    if (rte does not exist)
    {
        Generate_RERR(inRREP);
        return;
    }

    outRREP.HopLimit := inRREP.HopLimit - 1;
    if (outRREP.HopLimit == 0) /* don’t regenerate */
        return;

    if (inRREP.HopCount exists)
    {
        if (inRREP.HopCount >= MAX_HOPCOUNT)
            return; // don’t regenerate the RREP
    }
    outRREP.HopCount := inRREP.HopCount + 1;
}
/* Marshall parameters */
/* Include the AckReq when:
   - previous unicast RREP seems not to enable data flow, OR
   - when RREQ is received from same OrigAddr after RREP
     was unicast to rte.NextHop */
outRREP.AckReq := TRUE or FALSE; //TRUE if acknowledgement required
/* if included, set timeout RREP_Ack_SENT_TIMEOUT */

if (rte.MetricType != DEFAULT_METRIC_TYPE)
   outRREP.MetricType := rte.MetricType;
outRREP.OrigAddr := inRREP.OrigAddr;
outRREP.TargAddr := rte.Address;
outRREP.TargPrefixLen := rte.PrefixLength; //if not address length
outRREP.TargSeqNum := rte.SeqNum;
outRREP.TargMetric := rte.Metric;
outRREP.ValidityTime := limit for route to TargAddr; //if required
outRREP.NextHop := rte.NextHop

if (outRREP.AckReq == TRUE)
   /* include AckReq Message TLV */

   /* Build Address Block using prefix length information from
   outRREP.TargPrefixLen if necessary */
   AddrBlk := {outRREP.OrigAddr, outRREP.TargAddr};

   /* TargSeqNum Address Block TLV */
   targSeqNumAddrBlkTlv.value := outRREP.TargSeqNum;

   /* Build Metric Address Block TLV include Metric AddrBlkTlv
   Extension type if a non-default metric */
   metricAddrBlkTlv.value := outRREP.TargMetric;
   if (outRREP.MetricType != DEFAULT_METRIC_TYPE)
      metricAddrBlkTlv.typeExtension := outRREP.MetricType;

   if (outRREP.ValidityTime is required)
      { /* Build VALIDITY_TIME Address Block TLV */
         VALIDITY_TIMEAddrBlkTlv.value := outRREP.ValidityTime;
      }

Build_RFC_5444_Message_Header (RREP, 4, IPv4 or IPv6, NN,
   outRREP.HopLimit, 0, tlvLength);

   /* unicast RFC 5444 message to rte[OrigAddr].NextHop */
}
D.4.  RREP_Ack Operations

D.4.1.  Generate_RREP_Ack

/* To be sent when a received RREP includes the AckReq data element */

Generate_RREP_Ack(inRREP)
{
    Build_RFC_5444_Message_Header (RREP_Ack, 4, IPv4 or IPv6, NN,
    1, 0, 0);
    /* unicast RFC 5444 message to inRREP.NbrIP */
}

D.4.2.  Receive_RREP_Ack

Receive_RREP_Ack(inRREP_Ack)
{
    /* cancel timeout event for the node sending RREP_Ack */
}

D.4.3.  Timeout_RREP_Ack

Timeout_RREP_Ack(outRREP)
{
    if (numRetries < RREP_RETRIES)
        /* resend RREP and double the previous timeout */
    else
        /* insert unresponsive node into blacklist */
}

D.5.  RERR Operations

D.5.1.  Generate_RERR

There are two parts to this function, based on whether it was
triggered by an undeliverable packet or a broken link to neighboring
AODVv2 router.

/* Generate a Route Error message.
   errorType := undeliverablePacket or brokenLink */

Generate_RERR(errorType, triggerPkt, brokenLinkNbrIp)
{
    switch (errorType)
    {
        case (brokenLink):
            doGenerate := FALSE;
            num-broken-addr := 0;
            break;
        case (undeliverablePacket):
            doGenerate := FALSE;
            num-unknown-addr := 0;
            break;
        default:
            doGenerate := FALSE;
            break;
    }
}
precursors[] := new empty precursor list;
outRERR.HopLimit := MAX_HOPCOUNT;
/* find routes which are now Invalid */
foreach (rte in route table)
{
    if (brokenLinkNbrIp == rte.NextHop
        AND (rte.State == Active
        OR
            (rte.State == Idle AND ENABLE_IDLE_IN_RERR)))
    {
        if (rte.State == Active)
            doGenerate := TRUE;
        rte.State := Invalid;
precursors += rte.Precursors (if any);
outRERR.AddressList[num-broken-addr] := rte.Address;
outRERR.PrefixLengthList[num-broken-addr] :=
    rte.PrefixLength;
outRERR.SeqNumList[num-broken-addr] := rte.SeqNum;
outRERR.MetricTypeList[num-broken-addr] := rte.MetricType
num-broken-addr := num-broken-addr + 1;
    }
}
}

/* The remaining steps add address, prefix length, sequence
number and metric type information for each unreachable address,
while conforming to the allowed MTU. If the MTU is reached, a new
message MUST be created. */
/* Build Address Block using prefix length information from
outRERR.PrefixLengthList[] if necessary */
AddrBlk := outRERR.AddressList[];

/* Optionally, add SeqNum Address Block TLV, including index values */
seqNumAddrBlkTLV := outRERR.SeqNumList[];

if (outRERR.MetricTypeList contains non-default MetricTypes)
/* include Metric Address Block TLVs with Type Extension set to
MetricType, including index values if necessary */
metricAddrBlkTlv.typeExtension := outRERR.MetricTypeList[];

Build_RFC_5444_Message_Header (RERR, 4, IPv4 or IPv6, NN,
outRERR.HopLimit, 0, tlvLength);

if (undeliverablePacket)
/* unicast outRERR to rte[outRERR.PktSource].NextHop */
else if (brokenLink)
/* unicast to precursors, or multicast to LL-MANET-Routers */
}

D.5.2. Receive_RERR

Receive_RERR (inRERR)
{
  if (inRERR does not contain msg_hop_limit and at least
    one unreachable address)
    return;

  /* Extract inRERR values, copy relevant unreachable addresses,
    their prefix lengths, and sequence numbers to outRERR */
  num-broken-addr := 0;
  precursors[] := new empty precursor list;
  foreach (unreachableAddress in inRERR.AddressList)
  {
    if (unreachableAddress is not valid routable and unicast)
      continue;
    if (unreachableAddress MetricType is present but an unknown value)
      return;

    /* Find a matching route table entry, assume
       DEFAULT_METRIC_TYPE if no MetricType included */
    rte := Fetch_Route_Table_Entry (unreachableAddress,
                                       unreachableAddress MetricType)
    if (rte does not exist)
      continue;
    if (rte.State == Invalid)/* ignore already invalid routes */
      continue;
  }
if ((rte.NextHop != inRERR.NbrIP
    OR
    rte.NextHopInterface != inRERR.Netif)
    AND (PktSource is not present OR is not a Router Client))
   continue;
if (unreachableAddress SeqNum (if known) < rte.SeqNum)
   continue;

/* keep a note of all precursors of newly Invalid routes */
precursors += rte.Precursors; // if any

/* assume prefix length is address length if not included */
if (rte.PrefixLength != unreachableAddress prefixLength)
{
   /* create new route with unreachableAddress information */
   invalidRte := Create_Route_Table_Entry(unreachableAddress,
                                            unreachableAddress PrefixLength,
                                            unreachableAddress SeqNum,
                                            unreachableAddress MetricType);

   invalidRte.State := Invalid;

   if (rte.PrefixLength > unreachableAddress prefixLength)
       expunge_route(rte);
   rte := invalidRte;
}
else if (rte.PrefixLength == unreachableAddress prefixLength)
   rte.State := Invalid;

outRERR.AddressList[num-broken-addr] := rte.Address;
outRERR.SeqNumList[num-broken-addr] := rte.SeqNum;
outRERR.MetricTypeList[num-broken-addr] := rte.MetricType;
num-broken-addr := num-broken-addr + 1;

if (num-broken-addr AND (PktSource is not present OR PktSource is not
   a Router Client))
   Regenerate_RERR(outRERR, inRERR, precursors);
}

D.5.3. Regenerate_RERR
Regenerate_RERR (outRERR, inRERR, precursors)
{
    /* Marshal parameters */
    outRERR.HopLimit := inRERR.HopLimit - 1;
    if (outRERR.HopLimit == 0) // don’t regenerate
        return;

    outRERR.PktSource := inRERR.PktSource; //if included
    /* AddressList[], SeqNumList[], and PrefixLengthList[] are
       already up-to-date */

    if (outRERR.PktSource exists)
    {
        /* Build PktSource Message TLV */
        pktSourceMessageTlv.value := outRERR.PktSource;
    }

    /* Build Address Block using prefix length information from
       outRERR.PrefixLengthList[] if necessary */
    AddrBlk := outRERR.AddressList[];

    /* Optionally, add SeqNum Address Block TLV, including index values */
    seqNumAddrBlkTlv := outRERR.SeqNumList[];

    if (outRERR.MetricTypeList contains non-default MetricTypes)
        /* include Metric Address Block TLVs with Type Extension set to
           MetricType, including index values if necessary */
        metricAddrBlkTlv.typeExtension := outRERR.MetricTypeList[];

    Build_RFC_5444_Message_Header (RERR, 4, IPv4 or IPv6, NN,
           outRERR.HopLimit, 0, tlvLength);

    if (outRERR.PktSource exists)
        /* unicast RFC 5444 message to next hop towards
           outRERR.PktSource */
    else if (number of precursors == 1)
        /* unicast RFC 5444 message to precursors[0] */
    else if (number of precursors > 1)
        /* unicast RFC 5444 message to all precursors, or multicast
           RFC 5444 message to RERR_PRECURSORS if preferable */
    else
        /* multicast RFC 5444 message to LL-MANET-Routers */
}
Appendix E. AODVv2 Draft Updates

E.1. Changes between revisions 9 and 10

This section lists the changes between AODVv2 revisions ...-09.txt and ...-10.txt.

- Updated RFC 5444 Representation section to add "Address Type" TLV, which explicitly declares the meaning of addresses in the RFC 5444 Address Block.

- Relocated route state definitions. Minor improvements to clarity throughout.

- Updated definition of timed routes.

- More consistent use of OrigPrefixLen, TargPrefixLen, and Invalid.

- Mandated use of neighbor adjacency checking and support of AckReq and RREP_Ack and clarified related text.

- Changed order of LoopFree checking and route cost comparisons in Evaluating Route Information.

- Updated structure of section on Applying Route Updates.

- Updated AckReq to include intended next hop address, and RREP to be multicast if intended next hop is not a confirmed neighbor.

- Clarified that gateway router is not default router.

E.2. Changes between revisions 8 and 9

This section lists the changes between AODVv2 revisions ...-08.txt and ...-09.txt.

- Numerous editorial improvements were made, including relocation/removal/renaming/adding of some sections and text, collection and tidying of scattered text on same topic, formatting made more consistent to improve readability.

- Removed mentions of precursors from main text, except one mention in Route Table Entry.

- Removed use of MIN_METRIC which was not defined.

- Changed Current_Time to CurrentTime for consistency.
- Changed OrigAddrMetric and TargAddrMetric to OrigMetric and TargMetric respectively.
- Updated Overview to simplify and provide a broader summary.
- Updated Terminology definitions, Data Elements tables and combined sections.
- Updated Applicability Statement to move some of the non-applicability text and to simplify what remains.
- Updated TLV names to conform to existing naming style.
- Updated Blacklist to be a NeighborList to include neighbors that have confirmed bidirectional connectivity.
- Updated messages processed if router on blacklist and which are indicators of bidirectional links.
- Added RemoveTime to RteMsg Table section.
- Added short description of timed route to Route Table Entry section but removed Route.Timed flag. Route is timed if its expiration time is not MAX_TIME.
- Added Unconfirmed route state for route to OrigAddr learned from RREQ.
- Updated AODVv2 Protocol Operations section and subsections, including Initialization, Adjacency Monitoring, making algorithms easier to read and making notation consistent, general improvements to the text.
- Updated Route Discovery, Retries and Buffering to include a more complete description of the route discovery process.
- Updated wording relating to different metric types.
- Added text regarding control message limit in Message Transmission section.
- Added short explanation of positive/negative effects of buffering.
- Simplified the packet diagrams, since some of their contents was already explained in the text below and then again as part of generation, reception and regeneration processes.
- Clarified some elements of the message content descriptions.
Moved MetricType above MetricList in message sections, for consistency.

Mirrored structure throughout AODVv2 Protocol Messages.

Changed RREQ and RREP’s use of Lists when only one entry is necessary.

Added some pre-message-generation checks.

Ensured consistency in regeneration (if msg-hop-limit is reduced to zero, do not regenerate).

Removed statements about neighbors but added blacklist checks where necessary.

Noted that RREQ retries should increase the SeqNum.

Added statement that implementations SHOULD retry sending RREP.

Added text explaining what happens if RREP is lost, regarding blacklisting and RREQ retries.

Removed hop limit from RREP_Ack. Changed order of blacklist check.

Updated RERR so that multiple metric types can be reported in the same message.

Updated RERR reception processing to ensure PktSource deletes the contained route.

Added text to show that if a router is the destination of a RERR, the RERR is not regenerated.

Added text that RERRs should not be created if the same RERR has recently been sent.

Updated RFC 5444 overview and simplified/rearranged text in this section.

Major update to RFC 5444 representation section

Updated RERR’s RFC 5444 representation so that PktSource is placed in Address Block, and updated IANA section to make PktSource an Address Block TLV to indicate which address is PktSource.
o Described use of extension type in Metric TLV to represent MetricType, and the interpretation when using the default metric type.

o Removed Multicast RREP as an optional feature.

o Updated Precursor Lists section to include options for precursor information to store.

o Updated Security Considerations.

E.3. Changes between revisions 7 and 8

This section lists the changes between AODVv2 revisions ...-07.txt and ...-08.txt.

o MetricType is now an Address Block TLV. Minor changes to the text. By using an extension type in the Metric TLV we can represent MetricType more elegantly in the RFC 5444 message.

o Updated Overview to be slightly more concise.

o Moved MetricType next to Metric when mentioned for better flow.

o Added text to Applicability to address comments on mailing list regarding gateway behavior and NHDP HELLO messages.

o Removed paragraph in AODVv2 Message Transmission section regarding TTL.

o Added reference where precursors are mentioned in route table entry.

o Added text to bidirectionality explanation regarding NHDP HELLO messages and lower layer triggers.

o Clarified blacklist removal with SHOULD rather than MAY.

o Removed pseudo-code from section on evaluating incoming routing information.

o Clarified rules for expunging route entries on memory-constrained devices.

o Clarified the use of exponential backoff for route discovery attempts.
o Small updates to message sections. Removed steps about checking if neighbors.

o Renamed RFC 5444 parser to multiplexer in Section 10.

o Removed "optional feature" to include multiple addresses in RERR.

o Removed MetricType from the Message TLV Type Specification.

o Updated Security Considerations.

o Added reference to RFC 7182.

o Small updates to message algorithms, including moving MetricType from Message TLV to the Metric TLV in the Address Block TLV Block, and only generating RERR if an Active route was made Invalid.

E.4. Changes between revisions 6 and 7
This section lists the changes since AODVv2 revision ...-06.txt

o Added Victoria Mercieca as co-author.

o Reorganized protocol message descriptions into major subsections for each protocol message. For protocol messages, organized processing into Generation, Reception, and Regeneration subsections.

o Separated RREQ and RREP message processing description into separate major subsection which had previously been combined into RteMsg description.

o Enlarged RREQ Table function to include similar processing for optional flooded RREP messages. The table name has been correspondingly been changed to be the Table for Multicast RteMsgs.

o Moved sections for Multiple Interfaces and AODVv2 Control Message Generation Limits to be major subsections of the AODVv2 Protocol Operations section.

o Reorganized the protocol message processing steps into the subsections as previously described, adopting a more step-by-step presentation.

o Coalesced the router states Broken and Expired into a new combined state named the Invalid state. No changes in processing are required for this.
Merged the sections describing Next-hop Router Adjacency Monitoring and Blacklists.

Specified that routes created during Route Discovery are marked as Idle routes. If they are used for carrying data they become Active routes.

Added Route.LastSeqNumUpdate information to route table, so that route activity and sequence number validity can be tracked separately. An active route can still forward traffic even if the sequence number has not been refreshed within MAX_SEQNUM_LIFETIME.

Mandated implementation of RREP_Ack as response to AckReq Message TLV in RREP messages. Added field to RREP_Ack to ensure correspondence to the correct AckReq message.

Added explanations for what happens if protocol constants are given different values on different AODVv2 routers.

Specified that AODVv2 implementations are free to choose their own heuristics for reducing multicast overhead, including RFC 6621.

Added appendix to identify AODVv2 requirements from OS implementation of IP and ICMP.

Deleted appendix showing example RFC 5444 packet formats.

Clarification on the use of RFC 5497 VALIDITY_TIME.

In Terminology, deleted superfluous definitions, added missing definitions.

Numerous editorial improvements and clarifications.

E.5. Changes between revisions 5 and 6

This section lists the changes between AODVv2 revisions ...-05.txt and ...-06.txt.

 Added Lotte Steenbrink as co-author.

Reorganized section on Metrics to improve readability by putting specific topics into subsections.

Introduced concept of data element, which is used to clarify the method of enabling RFC 5444 representation for AODVv2 data elements. A list of Data Elements was introduced in section 3,
which provides a better understanding of their role than was previously supplied by the table of notational devices.

- Replaced instances of OrigNode by OrigAddr whenever the more specific meaning is appropriate. Similarly for instances of other node versus address terminology.

- Introduced concepts of PrefixLengthList and MetricList in order to avoid use of index-based terminology such as OrigNdx and TargNdx.

- Added section 5, "AODVv2 Message Transmission", describing the intended interface to RFC 5444.

- Included within the main body of the specification the mandatory setting of the TLV flag thassingleindex for TLVs OrigSeqNum and TargSeqNum.

- Removed the Route.Timed state. Created a new flag for route table entries known as Route.Timed. This flag can be set when the route is in the active state. Previous description would require that the route table entry be in two states at the same time, which seems to be misleading. The new flag is used to clarify other specification details for Timed routes.

- Created table 3 to show the correspondence between AODVv2 data elements and RFC 5444 message components.

- Replaced "invalid" terminology by the more specific terms "broken" or "expired" where appropriate.

- Eliminated the instance of duplicate specification for inclusion of OrigNode (now, OrigAddr) in the message.

- Corrected the terminology to be Mid instead of Tail for the trailing address bits of OrigAddr and TargAddr for the example message formats in the appendices.

- Repaired remaining instances of phraseology that could be construed as indicating that AODV only supports a single network interface.

- Numerous editorial improvements and clarifications.

E.6. Changes between revisions 4 and 5

This section lists the changes between AODVv2 revisions ...-04.txt and ...-05.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).

E.7. Changes between revisions 3 and 4

This section lists the changes between AODVv2 revisions ...-03.txt and ...-04.txt.

o An appendix was added to exhibit algorithmic code for implementation of AODVv2 functions.

o 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.

o Issues were resolved as discussed on the mailing list.

o Appropriate instances of "may" were changed to "MAY".

o 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 MetricType TLV to allow for omission when MetricType == 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.

E.8. Changes between revisions 2 and 3

This section lists the changes between AODVv2 revisions ...-02.txt and ...-03.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 2.

- 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.

Authors' Addresses

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

Phone: +1-408-330-4586
Email: charliep@computer.org

Stan Ratliff
Idirect
13861 Sunrise Valley Drive, Suite 300
Herndon, VA 20171
USA

Email: ratliffstan@gmail.com

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

Email: john.dowdell@airbus.com

Lotte Steenbrink
HAW Hamburg, Dept. Informatik
Berliner Tor 7
D-20099 Hamburg
Germany

Email: lotte.steenbrink@haw-hamburg.de