Mobile Ad hoc Networks Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 15, 2016

C. Perkins
Futurewei
S. Ratliff
Idirect
J. Dowdell
Airbus Defence and Space
L. Steenbrink
HAW Hamburg, Dept. Informatik
V. Mercieca
Airbus Defence and Space

October 13, 2015

Ad Hoc On-demand Distance Vector Routing Version 2 (AODVv2) draft-ietf-manet-aodvv2-12

Abstract

The Ad Hoc On-demand Distance Vector Version 2 (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 April 15, 2016.

Table of Contents

1. Overview

The Ad Hoc On-demand Distance Vector Version 2 (AODVv2) routing protocol (formerly named DYMO) enables on-demand, multihop unicast routing among AODVv2 routers in mobile ad hoc networks (MANETs) [RFC2501].

Compared to AODV [RFC3561], AODVv2 makes some features optional, notably intermediate route replies, expanding ring search, and precursor lists. Hello messages and local repair have been removed. Message formats have been updated and made compliant with [RFC5444]. AODVv2 also provides a mechanism for the use of multiple metric types.

The basic operations of the AODVv2 protocol are route discovery and route maintenance.

Route discovery is performed when an AODVv2 router needs to forward an IP 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, establishing a

route to both endpoints on all intermediate routers. A metric value is included to represent the cost of the route contained within the message.

AODVv2 uses sequence numbers to identify stale routing information, and compares route metric values to determine if advertised routes could form loops.

Route maintenance involves monitoring the router's links and routes for changes. This includes confirming bidirectionality of links to 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 [RFC5444] Address Blocks, Address Block TLVs, and Message TLVs.

Security for authentication of AODVv2 routers and encryption of control messages is accomplished using the TIMESTAMP and ICV TLVs defined 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 containing a list of IP addresses.

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

AckReq
: An AODVv2 Data Element used in a Route Reply message to request that the Route Reply message is acknowledged by returning a Route Reply Ack message. This Data Element contains the address of the AODVv2 router that should acknowledge the Route Reply message.

AdvRte
: A route advertised in an incoming route message.

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 field used within AODVv2 protocol messages.

ENAR (External Network Access Router)
An AODVv2 router with an interface to an external, non-AODVv2 network.

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 indicating the metric type for a metric value included in a message.

MetricTypeList
An AODVv2 Data Element used in a Route Error message, containing a list of metric types associated with the addresses in the AddressList of the message.

Neighbor
An AODVv2 router from which an AODVv2 message has been received. Neighbors exchange routing information and attempt to verify bidirectionality of the link to a neighbor before installing a route via that neighbor.

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 containing the source IP address of the IP packet triggering route discovery.

OrigMetric
An AODVv2 Data Element containing the metric value associated with the route to the OrigAddr in a message.

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

OrigSeqNum
An AODVv2 Data Element used in a Route Request message, containing the sequence number of the AODVv2 router which originated the Route Request.

PktSource
An AODVv2 Data Element used in a Route Error message, containing the source address of the IP packet which triggered the Route Error message.

PrefixLengthList
An AODVv2 Data Element containing a list of routing prefix lengths associated with the addresses in the AddressList of the message.

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
An address or address range configured on an AODVv2 router, corresponding to one or more nodes which require that router to initiate and respond to route discoveries on their behalf, so that they can send and receive IP traffic to and from remote destinations. The AODVv2 router's interface addresses are also configured as Router Clients.

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

RREP_Gen (RREP Generating Router)
The AODVv2 router configured with TargAddr as a Router Client, 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 a Router Client.

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

Sequence Number (SeqNum)
An AODVv2 Data Element containing the sequence number maintained by an AODVv2 router to indicate freshness of route information.

SeqNumList
An AODVv2 Data Element containing a list of sequence numbers associated with the addresses in the AddressList of a message.

TargAddr (Target Address)
An AODVv2 Data Element containing the destination address of the IP packet triggering route discovery.

TargMetric
An AODVv2 Data Element containing the metric value associated with the route to the TargAddr in a message.

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

TargSeqNum
An AODVv2 Data Element used in a Route Reply message, containing the sequence number of the AODVv2 router which originated the Route Reply.

Valid route
A route that can be used for forwarding.

Unreachable Address
An address reported in an RERR message, either the destination address of an IP packet that could not be forwarded because a valid route to the destination is not known, or the address on a route which became Invalid.

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

ValidityTime
An AODVv2 Data Element containing the length of time the route described by the message is offered.

The AODVv2 Data Elements are used to create AODVv2 messages. Their contents are transferred into [RFC5444] formatted messages (see Section 8) before sending.

This document uses the notational conventions in Table 1 to simplify the text.

Notation	Meaning
Route[Address]	A route toward Address
Route[Address].Field	A field in a route toward Address
RteMsg.Field	A field in either RREQ or RREP

Table 1: Notational Conventions

3. Applicability Statement

The AODVv2 routing protocol is a reactive routing protocol. Certain interactions with the forwarding plane are required, and these are discussed in Section 6.4.

AODVv2 is designed for stub or disconnected mobile ad hoc networks, i.e., non-transit networks or those not connected to the internet. AODVv2 can, however, be configured to perform gateway functions when attached to external networks, as discussed in Section 9.

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 IP 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.

Providing security for a reactive routing protocol can be difficult. AODVv2 provides for message integrity and security against replay attacks by using integrity check values, timestamps and sequence numbers, as described in Section 13. If security associations can be established, encryption can be used for AODVv2 messages to ensure that only trusted routers participate in routing operations.

Since the route discovery process typically results in a route being established in both directions along the same path, uni-directional

links are not suitable. AODVv2 will detect and exclude those links from route discovery. The route discovered is optimised for the requesting router, and the return path may not be the optimal route.

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), since if a router loses its sequence number, there is a delay before the router can resume full operations. This is described in Section 6.1.

AODVv2 supports routers with multiple interfaces, as long as each interface configured for AODVv2 has a unicast IP address. A router may use the same IP address on multiple interfaces. Address assignment procedures are out of scope for AODVv2.

AODVv2 supports hosts with multiple interfaces, as long as each interface is configured with its own unicast IP address. Multi-homing of an IP address is not supported by AODVv2, and therefore a Router Client, i.e. an IP Address, SHOULD NOT be served by more than one AODVv2 router at any one time. Appendix A 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.

4. Data Structures

4.1. Interface List

If multiple interfaces of the AODVv2 router are configured for use by AODVv2, a list of the interfaces SHOULD be configured in the AODVv2_INTERFACES list.

4.2. Router Client Table

An AODVv2 router MUST provide route discovery services for its own local applications and for other non-routing nodes that are reachable without traversing another AODVv2 router. These nodes, and the AODVv2 router itself, are referred to as Router Clients. An AODVv2

router will only originate Route Request and Route Reply messages on behalf of configured Router Clients.

Router Client Table entries MUST contain:

RouterClient.IPAddress
: An IP address or the start of an address range that requires route discovery services from the AODVv2 router.

RouterClient.PrefixLength
: The length, in bits, of the routing prefix associated with the RouterClient.IPAddress. If a prefix length is included, the AODVv2 router MUST provide connectivity for all addresses within that prefix.

RouterClient.Cost
: The cost associated with reaching this Router Client. This cost will also appear as the metric in a route table entry for the Router Client address.

The Router Client Table 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 Table.

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 address MUST NOT be served by more than one AODVv2 router at any one time, i.e. a Router Client of one AODVv2 router MUST NOT be configured as a Router Client on another AODVv2 router using the same Router Client IP address. Shifting responsibility for a Router Client to a different AODVv2 router is discussed in Appendix B.

4.3. Neighbor Table

A neighbor table MUST be maintained with information about neighboring AODVv2 routers which are used in discovered routes.

Neighbor Table entries MUST contain:

Neighbor.IPAddress
: An IP address of the neighboring router, learned from the source IP address of a received route message.

Neighbor.State
: The state of the adjacency with the neighbor (Confirmed, Unknown, or Blacklisted). The Unknown state is the initial state. The

Confirmed state indicates that the link to the neighbor has been confirmed as bidirectional. The Blacklisted state indicates that the link to the neighbor is uni-directional. Section 6.2 discusses how to monitor adjacency.

Neighbor.ResetTime
When the State is Blacklisted, this time indicates the point at which the State reverts to Unknown. By default this value is calculated at the time the router is blacklisted and is equal to CurrentTime + MAX_BLACKLIST_TIME. When the neighbor State is not Blacklisted, this time is set to INFINITY_TIME.

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 RREQ and RREP messages created by an AODVv2 router include the router's sequence number. Each AODVv2 router MUST ensure that its sequence number is strictly increasing. It is incremented by one (1) whenever an RREQ or RREP 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.

An AODVv2 router can only attach its own sequence number to information about a route to one of its configured router clients. All route messages regenerated by other routers retain the originator's sequence number. Therefore, when two pieces of information about a route are received, they both contain a sequence number from the originating router. Comparing the sequence number will identify which information is stale. The currently stored sequence number is subtracted from the incoming sequence number. The result of the subtraction is to be interpreted as a signed 16-bit integer, and if less than zero, then the information in the AODVv2 message is stale and MUST be discarded.

As a consequence, loop freedom is assured.

An AODVv2 router SHOULD maintain its sequence number in persistent storage. If the 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 or Route Reply message. The Multicast Route Message Table is a conceptual table which 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.

A Multicast Route Message Table entry MUST contain the following information:

RteMsg.MessageType
: Either RREQ or RREP.

RteMsg.OrigAddr
: An IP address of the node which requires the route, i.e., the source address of the IP packet triggering the route request.

RteMsg.OrigPrefixLen
: The prefix length associated with OrigAddr.

RteMsg.TargAddr
: An IP address of the target, i.e., the destination address of the IP packet triggering the route request.

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, MAX_SEQNUM_LIFETIME after the last update of RteMsg.OrigSeqNum for an RREQ, or RteMsg.TargSeqNum for an RREP.

The Multicast Route Message Table is maintained so that no two entries have the same MessageType, OrigAddr, TargAddr, and MetricType. See Section 6.8 for details on updating this table.

4.6. Route Table

All AODVv2 routers MUST maintain a route table. The route table entry is a conceptual data structure. Implementations MAY use any internal representation but MUST contain the following information:

Route.Address
An address, which, when combined with Route.PrefixLength, describes the set of destination addresses this route includes.

Route.PrefixLength
The prefix length, in bits, associated with Route.Address.

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

Route.NextHop
The source IP address of the message advertising the route to Route.Address, i.e. 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 IP packets toward Route.Address.

Route.LastUsed
The time this route was last used to forward an IP 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 IP packets.

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

Invalid
An Invalid route cannot be used for forwarding IP packets. Invalid routes have sequence number information, which allows incoming information to be assessed for freshness.

Unconfirmed
An Unconfirmed route cannot be used for forwarding IP 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 6.9.1.

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 INFINITY_TIME.

5. Metrics

Metrics measure a cost or quality associated with a route or a link, e.g., latency, delay, financial cost, energy, etc. Metric values are reported in route messages, where the goal is to determine a route between OrigAddr and TargAddr. In Route Request messages, the metric describes the cost of the route from OrigAddr (the router client) to the router sending the Route Request. The receiving router calculates the cost from OrigAddr to itself, combining the metric value from the message with knowledge of the link cost from the

sender to the receiver, i.e., the incoming link cost. This updated route cost is included when regenerating the Route Request message. In Route Reply messages, the metric reflects the cost of the route from TargAddr (the router client) to the router sending the Route Reply. Routes to OrigAddr and TargAddr are installed at intermediate routers for the purposes of forwarding a Route Reply message and subsequent data traffic between OrigAddr and TargAddr. Assuming link metrics are symmetric, the cost of the routes to OrigAddr and TargAddr installed at each router will be correct.

AODVv2 enables the use of multiple metric types. Each route discovery attempt indicates the metric type which is requested for the route. Only one metric type may be used in each route discovery attempt. However, routes to a single destination might be requested for different metric types. The decision of which of these routes to use for forwarding is outside the scope of AODVv2.

For each MetricType, AODVv2 requires:

- A MetricType number, to indicate the metric type of a route. MetricType numbers allocated are detailed in Section 11.6.

- A maximum value, denoted MAX_METRIC[MetricType]. If the cost of a route exceeds MAX_METRIC[MetricType], the route is ignored. AODVv2 cannot store routes that cost more than MAX_METRIC[MetricType].

- A function for incoming link cost, denoted Cost(L). Using incoming link costs means that the route learned has a path optimized for the direction from OrigAddr to TargAddr.

- A function for route cost, denoted Cost(R).

- A function to analyze routes for potential loops, denoted LoopFree(R1, R2). LoopFree verifies that a route R2 is not a sub-section of another route R1. An AODVv2 router invokes LoopFree() as part of the process in Section 6.7.1, when an advertised route (R1) and an existing route (R2) have the same destination address, metric type, and sequence number. LoopFree returns FALSE to indicate that an advertised route is not to be used to update a stored route, if it may cause a routing loop. In the case where the existing route is Invalid, it is possible that the advertised route includes the existing route and came from a router which did not yet receive notification of the route becoming Invalid, so the advertised route should not be used in case it forms a loop to a broken route.

AODVv2 currently supports cost metrics where Cost(R) is strictly increasing, by defining:

- Cost(R) := Sum of Cost(L) of each link in the route

- LoopFree(R1, R2) := (Cost(R1) <= Cost(R2))

Implementers MAY consider other metric types, but the definitions of Cost and LoopFree functions for such types are undefined, and interoperability issues need to be considered.

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 an AODVv2 router does not have information about its previous sequence number, or if its sequence number is lost at any point, the router resets its sequence number to one (1). However, other AODVv2 routers may still hold sequence number information that this router previously issued. Since sequence number information is removed if there has been no update to the sequence number in MAX_SEQNUM_LIFETIME, the initializing router must wait for MAX_SEQNUM_LIFETIME before it creates any messages containing its new 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 of that route discovery

- Regenerate a received RREQ or RREP

- Send an RREP_Ack

- Maintain valid routes in order that the forwarding process can forward IP packets to Router Clients and to other routers

- Create, process and regenerate RERR messages

6.2. Adjacency Monitoring

AODVv2 routers MUST NOT establish routes over uni-directional links. Consider the following. An RREQ is forwarded toward TargAddr, and intermediate routers install a route to OrigAddr. If, at one of those routers, the link to the next hop toward OrigAddr was uni-directional, and this route was used to forward data traffic, the data packets would be lost. Further, an RREP sent toward OrigAddr using this link will not reach the next hop, and will therefore not be regenerated, and will never reach RREQ_Gen, so end-to-end route establishment will fail. AODVv2 routers MUST verify that the link to the next hop is bidirectional when establishing a route, and before allowing data traffic to be forwarded on that route. If bidirectionality cannot be verified, this link MUST be excluded from the route discovery procedure.

AODVv2 refers to a bidirectional link with a neighboring router as an adjacency. AODVv2 routers do not need to monitor adjacency to all neighboring AODVv2 routers at all times, but MUST determine if there is an adjacency to the chosen next-hop AODVv2 router during route discovery.

- For the next hop toward OrigAddr, the approach for testing bidirectional connectivity 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. If a link to a neighbor is determined to be unidirectional because a requested acknowledgement is not received within RREP_Ack_SENT_TIMEOUT, the neighbor MUST be marked as blacklisted (see below).

- For the next hop toward TargAddr, receipt of the Route Reply message containing the route to TargAddr is confirmation of bidirectionality, since a Route Reply message is a reply to a Route Request message which previously crossed the link in the opposite direction.

To assist with adjacency monitoring, a Neighbor Table (Section 4.3) is maintained. Each entry contains a neighbor IP address and an indication of the state of the adjacency with that neighbor (Unknown, Blacklisted, or Confirmed). When an RREQ or RREP is received from an IP address which does not already have an entry in the Neighbor Table, a new entry is created as described in Section 6.3. While neighbor state is Unknown, acknowledgement of RREP messages MUST be

requested. While neighbor state is Confirmed, the request for an acknowledgement is unnecessary.

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

- NHDP HELLO Messages [RFC6130]
- Route timeout
- Lower layer triggers, e.g. message reception or link status notifications
- TCP timeouts
- Promiscuous listening
- Other monitoring mechanisms or heuristics

If such an external process signals that the link is bidirectional, the neighbor state MAY be set to Confirmed. If an external process signals that a link is not bidirectional, the AODVv2 router MAY update the matching Neighbor Table entry by changing the neighbor state to Blacklisted. If an external process signals that the link might not be bidirectional, and the neighbor state is currently Confirmed, the state MAY be set to Unknown.

For example, receipt of a Neighborhood Discovery Protocol HELLO message with the receiving router listed as a neighbor is a signal of bidirectional connectivity. The AODVv2 router MAY update the matching Neighbor Table entry by changing the neighbor state to Confirmed.

Similarly, if AODVv2 receives notification of a timeout, for example, from TCP or some other protocol, this may be due to a disconnection. The AODVv2 router MAY update the matching Neighbor Table entry by resetting the neighbor state to Unknown.

6.3. Neighbor Table Update

On receipt of an RREQ or RREP message, the neighbor table MUST be checked for an entry with Neighbor.IPAddress which matches the source IP address of the message. If no matching entry is found, a new entry is created.

A new Neighbor Table entry is created as follows:

- Neighbor.IPAddress := Source IP address of the message

- Neighbor.State := Unknown

- Neighbor.ResetTime := INFINITY_TIME

When the link to the neighbor is determined to be bidirectional, the Neighbor Table entry is updated as follows:

- Neighbor.State := Confirmed

When the link to the neighbor is determined to be uni-directional, the Neighbor Table entry is updated as follows:

- Neighbor.State := Blacklisted

- Neighbor.ResetTime := CurrentTime + MAX_BLACKLIST_TIME

When the Neighbor.ResetTime is reached, the Neighbor Table entry is updated as follows:

- Neighbor.State := Unknown

When a link to a neighbor is determined to be broken, the Neighbor Table entry SHOULD be removed.

Route requests from neighbors with Neighbor.State set to Blacklisted are ignored to avoid persistent IP packet loss or protocol failures. However, the reset time allows the neighbor to again be allowed to participate in route discoveries after MAX_BLACKLIST_TIME, in case the link between the routers has become bidirectional.

6.4. Interaction with Forwarding Plane

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.

AODVv2 requires signals from the forwarding plane:

- A packet cannot be forwarded because a route is unavailable: AODVv2 needs to know the source and destination IP addresses of the packet, to determine whether it should initiate route discovery, and include this information in a Route Request message, or create a Route Error message.

- A packet is to be forwarded: AODVv2 needs to check the state of the route to deal with timeouts. If the implementation uses timers to enforce route timeouts, this signal is unnecessary.

- Packet forwarding failure occurs: AODVv2 needs to initiate route error reports.

- Packet forwarding succeeds: AODVv2 needs to update the record of when a route was last used to forward a packet.

AODVv2 needs to send signals to the forwarding plane:

- A route discovery is in progress: packets awaiting a route may be buffered while route discovery is attempted.

- A route discovery was not attempted: any buffered packets requiring that route should be discarded.

- A route discovery failed: any buffered packets requiring that route should be discarded, and the source of the packet should be notified that the destination is unreachable (using an ICMP Destination Unreachable message).

- A route discovery succeeded: install a route which AODVv2 has determined to be valid and begin transmitting any buffered packets.

- A route has been lost: remove an installed route which AODVv2 has determined to be invalid.

- A route has been updated: update an installed route when AODVv2 receives new information about the route.

These are conceptual signals, and can be implemented in various ways. Conformant implementations of AODVv2 are not mandated to implement the forwarding plane separately from the control plane or data plane; these signals and interactions are identified simply as assistance for implementers who may find them useful.

6.5. Message Transmission

AODVv2 sends [RFC5444] formatted messages using the parameters for port number and IP protocol specified in [RFC5498]. Mapping of AODVv2 Data Elements to [RFC5444] is detailed in Section 8.

Messages may travel a maximum of MAX_HOPCOUNT hops.

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.

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.

Implementations MAY choose to employ techniques to reduce the number of multicast messages sent. Use of [RFC6621] in deployments is recommended. Employing [RFC6621] in a subset of the operational AODVv2 routers in a network, or configuring different algorithms on different routers, will not cause interoperability issues, but will reduce the effectiveness of the multicast reduction scheme.

When multiple interfaces are available, an AODVv2 router 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 SHOULD be limited (CONTROL_TRAFFIC_LIMIT) and administratively configurable. To prioritize transmission of AODVv2 control messages in order to respect the CONTROL_TRAFFIC_LIMIT:

- Highest priority SHOULD be given to RREP_Ack messages. This allows learned routes to be confirmed as bidirectional and avoids undesirable blacklisting of next hop routers.

- Second priority SHOULD be given to RERR messages for undeliverable IP packets, so that broken routes that are still being used are reported, and to avoid IP data packets being repeatedly forwarded to AODVv2 routers which cannot forward to their destination.

- Third priority SHOULD be given to RREP messages in order that RREQs do not time out.

- RREQ messages SHOULD be given priority over RERR messages for newly invalidated routes, since the invalidated routes may not still be in use, and if there is an attempt to use the route, a new RERR message will be generated.

- Lowest priority SHOULD be given to RERR messages generated in response to RREP messages which cannot be regenerated. In this case the route request will be retried at a later point.

6.6. Route Discovery, Retries and Buffering

AODVv2's RREQ and RREP messages are used for route discovery. The main difference between the two messages is that, usually, RREQ messages are multicast to solicit an RREP, whereas RREP is unicast as a response to the RREQ. The constants used in this section are defined in Section 11.

When an AODVv2 router needs to forward an IP 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. Each new RREQ results in an increment to the sequence number. The AODVv2 router is referred to as RREQ_Gen.

IP packets awaiting a route MAY be buffered by RREQ_Gen. Buffering of IP packets can have both positive and negative effects. 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.

Determining which packets to discard first when the buffer is full is a matter of policy at each AODVv2 router. Routers without sufficient memory available for buffering SHOULD have buffering disabled. 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. An RREQ from TargAddr would also fulfil 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. To reduce congestion in a network, repeated attempts at route discovery for a particular target address SHOULD utilize a binary exponential backoff: for each additional attempt, the waiting time for receipt of the RREP is multiplied by 2. If the requested route is not learned within the wait period, another RREQ MAY be sent, up to a total of DISCOVERY_ATTEMPTS_MAX. This is the same technique used in AODV [RFC3561].

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 unicasts it back toward RREQ_Gen using the potential 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 link is unavailable in the direction toward OrigAddr, an RREP is not received at the next hop, so cannot be regenerated, and it 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. Later, a timeout occurs at RREQ_Gen, and a new RREQ MAY be regenerated. If the new RREQ arrives via the blacklisted router, it will be ignored, enabling the RREQ to discover a different path toward TargAddr.

Route discovery SHOULD be considered to have failed after DISCOVERY_ATTEMPTS_MAX and the corresponding wait time for an RREP response to the final RREQ, in order to avoid repeatedly generating control traffic that is unlikely to discover a route. After the attempted route discovery has failed, RREQ_Gen MUST wait at least RREQ_HOLDDOWN_TIME before attempting another route discovery to the same destination, to avoid generating more multicast messages which are unlikely to discover a route. Any IP 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 packet, so that the application knows about the failure. The source can 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 MUST process the message according to Section 7. When a valid route is installed, the router can begin sending the buffered IP packets. Any retry timers for the corresponding RREQ MUST be cancelled.

During route discovery, all routers on the path learn a route to both OrigAddr and TargAddr, so that routes are constructed in both directions. The route is optimized for the forward route, and the return route uses the same path in reverse.

6.7. 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 6.7.1. The route table MAY then be updated according to Section 6.7.2.

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 an existing route which matches AdvRte on address, prefix length, and metric type.

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 := RteMsg.IPSourceAddress (an address of the router from which the AdvRte was received)
- AdvRte.MetricType := RteMsg.MetricType
- AdvRte.Metric := RteMsg.Metric
- AdvRte.Cost := Cost(R) using the cost function associated with the route's metric type, i.e. Cost(R) = AdvRte.Metric + Cost(L), as described in Section 5, where L is the link from the advertising router.
- AdvRte.ValidityTime := RteMsg.ValidityTime, if included

6.7.1. Evaluating Route Information

An incoming route advertisement (AdvRte) is compared to existing routes to determine whether the advertised route is to be used to update the routing table. The incoming route information MUST be processed as follows:

1. Search for a route (Route) matching AdvRte's address, prefix length and metric type

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

* If all matching routing table entries have State set to Unconfirmed, AdvRte SHOULD be added to the routing table. This may result in multiple Unconfirmed routes to the same address. In this case, the best route from the set of Unconfirmed routes SHOULD be used to forward future RREPs. If the link to the next hop is found to be bidirectional, and the Unconfirmed route becomes valid, any remaining Unconfirmed routes which would not offer improvement MUST be expunged.

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

2. Compare sequence numbers using the technique described in Section 4.4

   * If AdvRte is more recent, AdvRte MUST be used to update the routing table.

   * If AdvRte is stale, 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)

   * 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 4.

4. Compare route costs

   * If AdvRte is better, it SHOULD be used to update the routing table because it offers improvement. If it is not used to update the existing route, the existing non-optimal route will continue to be used, causing data flows to use a route with a worse cost where this could have been avoided.

   * If AdvRte is equal in cost and Route is Valid, AdvRte MAY be used to update the routing table but will offer no improvement.

   * If AdvRte is worse 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 6.7.2 MUST be followed. If the route is not used, non-optimal routes will remain in the routing table.

6.7.2. Applying Route Updates

If AdvRte is from an RREQ message, the next hop neighbor may not be confirmed as adjacent (see Section 4.3). If Neighbor.State is Unknown, the route to AdvRte.Address might not be viable, but it MUST be stored to allow a corresponding RREP to be sent. However, the route's State will be set to Unconfirmed to indicate that this route SHOULD NOT yet be used to forward data, since the link may be uni-directional and packet losses may occur. If a valid route already exists for this destination, this Unconfirmed route SHOULD be stored as an additional entry. If the link to the next hop is later confirmed to be bidirectional, the route will offer improvement over the existing valid route.

The route update is applied as follows:

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

2. If a matching route exists:

   * If AdvRte has a different next hop to the existing route (Route), and both 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 the next hop can be confirmed as bidirectional. Continue processing from Step 3 to create a new route.

   * If AdvRte.NextHop's Neighbor.State is Unknown and Route.State is Invalid, continue processing from Step 4 to update the existing route (Route).

   * If AdvRte.NextHop's Neighbor.State is Confirmed, continue processing from Step 4 to update the existing route.

3. Create a route and initialize as follows:

   * Route.Address := AdvRte.Address

   * Route.PrefixLength := AdvRte.PrefixLength

* Route.MetricType := AdvRte.MetricType

4. Update the route as follows:

   * Route.SeqNum := AdvRte.SeqNum

   * Route.NextHop := AdvRte.NextHop

   * Route.NextHopInterface := interface on which RteMsg was received

   * Route.Metric := AdvRte.Cost

   * Route.LastUsed := CurrentTime

   * Route.LastSeqNumUpdate := CurrentTime

   * Route.ExpirationTime := CurrentTime + AdvRte.ValidityTime if a validity time exists, otherwise INFINITY_TIME

5. If a new route was created, or if the existing Route.State is Invalid or Unconfirmed, update the route as follows:

   * Route.State := Unconfirmed (if the next hop's Neighbor.State is Unknown) or Idle (if the next hop's Neighbor.State is Confirmed)

6. If an existing route changed from Invalid or Unconfirmed to become Idle, any matching route table entries with worse metric values SHOULD be expunged.

7. If this update results in a route with Route.State set to Active or Idle, which matches an outstanding route request, the associated route request retry timers can be cancelled and any associated buffered IP packets MUST be forwarded.

6.8. Suppressing Redundant Messages Using the Multicast Route Message Table

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 Route Message Table (Section 4.5).

To create a Multicast Route Message Table Entry:

- RteMsg.MessageType := RREQ or RREP
- RteMsg.OrigAddr := OrigAddr from the message
- RteMsg.OrigPrefixLen := the prefix length associated with OrigAddr
- RteMsg.TargAddr := TargAddr from the message
- RteMsg.TargPrefixLen := the prefix length associated with TargAddr
- RteMsg.OrigSeqNum := the sequence number associated with OrigAddr, if present in the message
- RteMsg.TargSeqNum := the sequence number associated with TargAddr, if present in the message
- RteMsg.MetricType := the metric type of the route requested
- RteMsg.Metric := the metric value associated with OrigAddr in an RREQ or TargAddr in an RREP
- RteMsg.Timestamp := CurrentTime
- RteMsg.RemoveTime := CurrentTime + MAX_SEQNUM_LIFETIME

Entries in the Multicast Route Message Table 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. An entry MUST be deleted when the sequence number is no longer valid, i.e., after MAX_SEQNUM_LIFETIME. Memory-constrained devices MAY remove the entry before this time.

To update a Multicast Route Message Table Entry, set:

- RteMsg.OrigSeqNum := the sequence number associated with OrigAddr, if present in the message
- RteMsg.TargSeqNum := the sequence number associated with TargAddr, if present in the message
- RteMsg.Metric := the metric value associated with OrigAddr in an RREQ or TargAddr in an RREP
- RteMsg.Timestamp := CurrentTime
- RteMsg.RemoveTime := CurrentTime + MAX_SEQNUM_LIFETIME

Received route messages are tested against previously received route messages, and if determined to be redundant, regeneration or response can be avoided.

To determine if a received message is redundant:

1. Search for an entry in the Multicast Route Message Table with the same MessageType, OrigAddr, TargAddr, and MetricType

    * If there is none, the message is not redundant.

    * If there is an entry, continue to Step 2.

2. Compare sequence numbers using the technique described in Section 4.4

    * For RREQ messages, use OrigSeqNum of the entry for comparison. For RREP messages, use TargSeqNum of the entry for comparison.

    * If the entry has an older sequence number than the received message, the message is not redundant.

    * If the entry has a newer sequence number than the received message, the message is redundant.

    * If the entry has the same sequence number, continue to Step 3.

3. Compare the metric values

    * If the entry has a Metric value that is worse than or equal to the metric in the received message, the message is redundant.

    * If the entry has a Metric value that is better than the metric in the received message, the message is not redundant.

If the message is redundant, update the timestamp on the entry, since matching route messages are still traversing the network and this entry should be maintained. This message SHOULD NOT be regenerated or responded to.

If the message is not redundant, create an entry or update the existing entry. Where the message is determined not redundant before Step 3, it MUST be regenerated or responded to. Where the message is determined not redundant in Step 3, it MAY be suppressed to avoid extra control traffic. However, since the processing of the message will result in an update to the route table, the message SHOULD be regenerated or responded to, to ensure other routers have up-to-date information and the best metrics. If not regenerated, the best route

may not be found. Where necessary, regeneration or response is performed using the processes in Section 7.

6.9. 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 an IP packet, an AODVv2 router MUST check firstly if there is a route, and secondly the status of the route (Section 6.9.1). If the route exists and is valid, it MUST be marked as Active and its LastUsed timestamp MUST be updated, before forwarding the IP packet to the route's next hop. If there is no valid route, and if the source address of the IP packet is a Router Client, the RREQ generation procedure MUST be followed. Otherwise, the absence of a route MUST be reported to the packet's source (see Section 6.9.2).

6.9.1. Route State

During normal operation, AODVv2 does not require any explicit timeouts to manage the lifetime of a route. At any time, any route MAY be examined and updated according to the rules below. If timers are not used to prompt route state updates, route state MUST be checked before IP packet forwarding and before any operation based on route state.

The four possible states for an AODVv2 route are Active, Idle, Invalid, and Unconfirmed:

Active
: If Route.State is Active and the route is not timed (i.e., if Route.ExpirationTime is INFINITY_TIME), Route.State MUST become Idle if Route is not used to forward IP packets within ACTIVE_INTERVAL. Route.State for a timed route (i.e., Route.ExpirationTime is not equal to INFINITY_TIME) remains Active until its expiration time, after which it MUST become Invalid.

Idle
: If Route.State is Idle, and the route is used to forward an IP packet, Route.State MUST become Active. If the route is not used to forward an IP packet within MAX_IDLETIME, Route.State MUST become Invalid.

Invalid
: If Route.State is Invalid, the route SHOULD be maintained until MAX_SEQNUM_LIFETIME after Route.LastSeqNumUpdate, after which it

MUST be expunged. Route.SeqNum is used to classify future information about Route.Address as stale or fresh.

Unconfirmed
If Route.State is Unconfirmed, the route MUST become Idle when an adjacency with Route.NextHop is confirmed, or MUST be expunged if the neighbor is blacklisted, or at MAX_SEQNUM_LIFETIME after Route.LastSeqNumUpdate.

In all cases, if the time since Route.LastSeqNumUpdate exceeds MAX_SEQNUM_LIFETIME, Route.SeqNum must be set to zero. This is required to ensure that any AODVv2 routers following the initialization procedure can safely begin routing functions using a new sequence number, and that their messages will not be classified as stale and ignored. A route with Route.State set to Active or Idle can continue to be used to forward IP packets, but if Route.State later becomes Invalid, the route MUST be expunged.

Appendix C.2.1.1 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 for Route.NextHop MUST immediately have Route.State set to Invalid.

- If a Route Error (RERR) message containing the route is received, either from Route.NextHop, or with PktSource set to a Router Client address, Route.State MUST immediately be set to Invalid.

When Route.State changes from Unconfirmed to Idle as a result of the adjacency with Route.NextHop being Confirmed (see Section 4.3), any matching routes with metric values worse than Route.Metric MUST be expunged.

Memory constrained devices MAY choose to expunge routes from the AODVv2 route table before Route.ExpirationTime, but MUST adhere to the following rules:

- An Active route MUST NOT be expunged, as this will result in generation of a Route Error message followed by a necessary Route Request to re-establish the route.

- An Idle route SHOULD NOT be expunged, as the route is still valid for forwarding IP traffic, and if deleted, this could result in dropped IP packets and a Route Request could be generated to re-establish the route.

- Any Invalid route MAY be expunged; least recently used Invalid routes SHOULD be expunged first, since these are less likely to be reused.

- An Unconfirmed route MUST NOT be expunged if it was installed within the last RREQ_WAIT_TIME, because it may correspond to a route discovery in progress. A Route Reply message might be received which needs to use the Route.NextHop information. Otherwise, it MAY be expunged.

Route table entries are updated when Neighbor State is updated:

- While Neighbor.State is set to Unknown, 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 SHOULD be marked as valid (see Section 6.9.1).

- When Neighbor.State is set to Blacklisted, any valid routes installed which use that neighbor for their next hop are marked as Invalid.

- When a Neighbor Table entry is removed, all routes using the neighbor as next hop MUST be marked as Invalid.

6.9.2. Reporting Invalid Routes

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

An RERR message SHOULD also be sent when an AODVv2 router receives an IP packet to forward on behalf of another router but does not have a valid route for the destination of the packet.

An RERR message SHOULD also be sent when an AODVv2 router receives an RREP message to regenerate, but the route to the OrigAddr in the RREP has been lost and is marked as Invalid.

The packet or message triggering the RERR MUST be discarded.

Generation of an 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 [RFC5444] 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 contents:

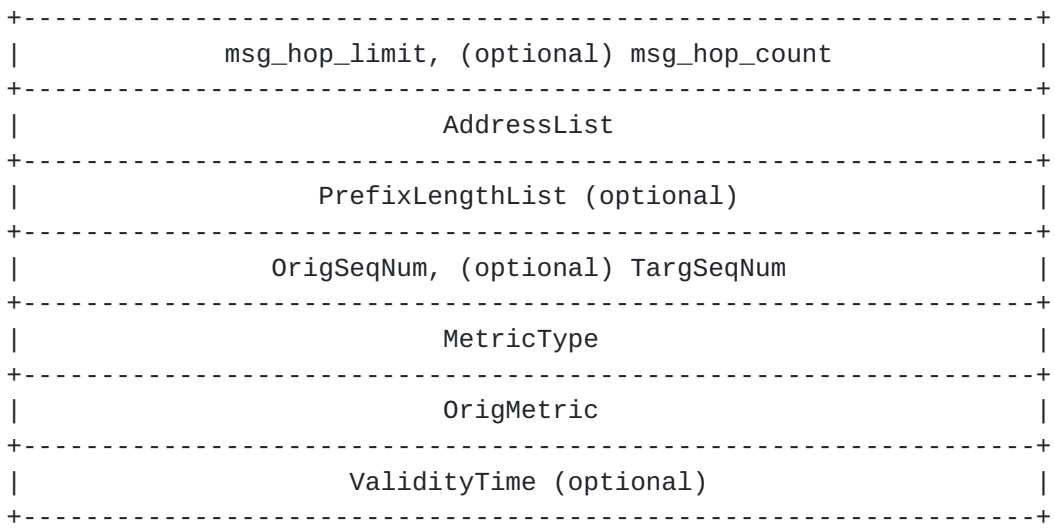

Figure 1: RREQ message contents

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 IP 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.

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 metric type associated with OrigMetric.

OrigMetric
The metric value 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

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

Before creating an RREQ, the router SHOULD check if an 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 IP 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. If approaching the limit, the message should be sent if the priorities in Section 6.5 allow it.

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 is part of an address range configured as a Router Client, set PrefixLengthList := {OrigPrefixLen, null}.

   * Otherwise, omit PrefixLengthList.

5. For OrigSeqNum:

   * Increment the router SeqNum 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 and set the type accordingly

8. Set OrigMetric := Route[OrigAddr].Metric, i.e., RouterClient.Cost

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 [RFC5444] 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 an 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. Check that the MetricType is supported and configured for use

   * 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 6.7.1

7. Check if the message is redundant by comparing to entries in the Multicast Route Message table, following the procedure in (Section 6.8)

   * If redundant, ignore this RREQ for further processing.

   * If not redundant, continue processing.

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

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

   * If not, continue to RREQ regeneration.

7.1.3. RREQ Regeneration

By regenerating an RREQ, a router advertises that it will forward IP 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 might avoid regenerating an 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. If approaching the limit, the message should be sent if the priorities in Section 6.5 allow it.

The procedure for RREQ 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 included, otherwise omit msg_hop_count

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

5. Set OrigMetric := 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 time limit the previous AODVv2 router specified, or the time limit this router wishes to impose, whichever is lower.

This AODVv2 message is used to create a corresponding [RFC5444] 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

When a Route Request message is received, requesting a route to a Target Address which is configured as a Router Client, a Route Reply message is sent in response. The RREP offers a route to the Target Address.

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 to request acknowledgement to confirm the neighbor adjacency.

RREP messages have the following contents:

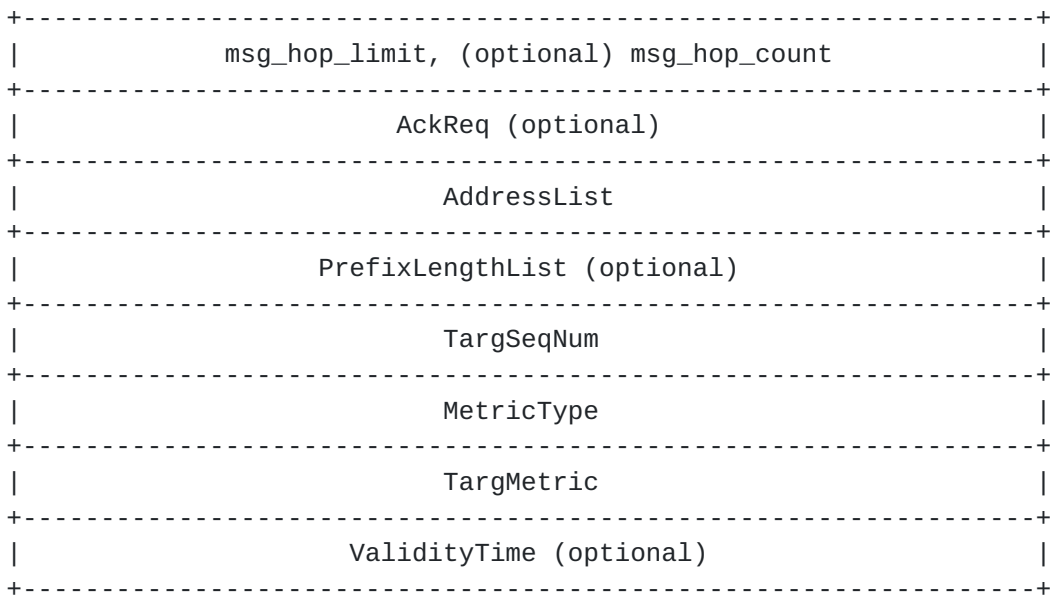

Figure 2: RREP message contents

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 to be multicast because the next hop toward OrigAddr is a neighbor with Unknown state. It indicates that an acknowledgement of 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 IP 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.

TargSeqNum
The sequence number associated with TargAddr.

MetricType
The metric type associated with TargMetric.

TargMetric
The metric value 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

An RREP is generated when an RREQ arrives requesting a route to one of the AODVv2 router's Router Clients.

Before creating an 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 approaching the limit, the message should be sent if the priorities in Section 6.5 allow it.

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 next hop neighbor is already confirmed as adjacent, the AckReq Data Element 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 6.6 for RREQ retries. Adjacency confirmation MUST be considered to have failed after the wait time for an 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 is part of an address range configured as a Router Client, set PrefixLengthList := {null, TargPrefixLen}.

   * Otherwise, omit PrefixLengthList.

6. For the TargSeqNum:

   * Increment the router SeqNum as specified in Section 4.4.

   * Set TargSeqNum := SeqNum.

7. Include the MetricType Data Element and set the type to match the MetricType in the received RREQ message

8. Set TargMetric := Route[TargAddr].Metric, i.e., RouterClient.Cost

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 [RFC5444] 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 an 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 Neighbor Table entry for this sender SHOULD have State set to Confirmed since an 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. Check that the MetricType is supported and configured for use

   * If not, ignore this RREP for further processing.

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

   * If it does, 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 6.7.1

   * If the route to TargAddr fulfills a previously sent RREQ, any associated timeouts will be cancelled and buffered IP 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 Multicast Route Message table (Section 6.8)

   * If redundant, ignore this RREP for further processing.

   * If not redundant, save the information in the Multicast Route Message 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 IP 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 an 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 approaching the limit, the message should be sent if the priorities in Section 6.5 allow it.

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 time limit the previous AODVv2 router specified, or the time limit this router wishes to impose, whichever is lower.

This AODVv2 message is used to create a corresponding [RFC5444] 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

An 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 an RREP_Ack. The [RFC5444] representation is discussed in Section 8. The RREP_Ack is unicast, by default, to the source IP address of the RREP message that requested it.

7.3.2. RREP_Ack Reception

Upon receiving an RREP_Ack, an AODVv2 router performs the following steps:

1. If an 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. An RERR message has the following contents:

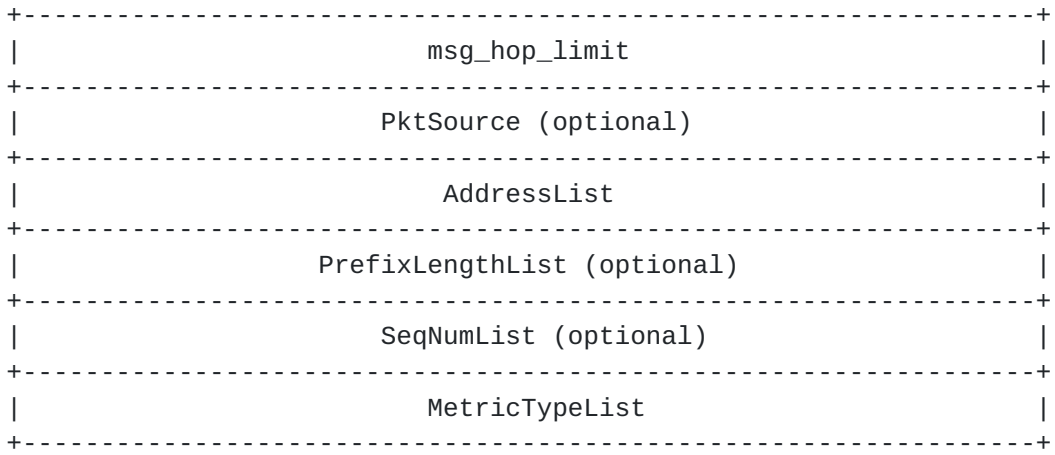

Figure 3: RERR message contents

RERR Data Elements

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

PktSource
The source address of the IP 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. These values indicate whether routes represent a single device or an address range.

SeqNumList
The sequence numbers of the routes no longer available through RERR_Gen (where known).

MetricTypeList
The metric types associated with the routes no longer available through RERR_Gen.

7.4.1. RERR Generation

An 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 an IP packet arrives that cannot be forwarded because no valid route exists for its destination, or if an RREP arrives which cannot be regenerated because no route exists to OrigAddr, the RERR generated MUST contain the PktSource Data Element and will contain only one unreachable address. The contents of PktSource and AddressList are set as follows:

  * For an IP packet that cannot be forwarded, PktSource is set to the source address of the IP packet, and the AddressList contains the destination address of the IP packet.

  * For an RREP message when 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 PktSource. The MetricTypeList MUST also be included if a MetricType can be determined from the IP packet or an existing Invalid route to PktSource.

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

In order to avoid flooding the network with RERR messages when a stream of IP packets to an unreachable address arrives, an AODVv2 router SHOULD determine whether an 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. 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.

  An 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 IP 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. If approaching the limit, the message should be sent if the priorities in Section 6.5 allow it.

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 IP packet triggering the RERR, or the TargAddr of an RREP that cannot be regenerated toward OrigAddr

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.

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

The AODVv2 message is used to create a corresponding [RFC5444] message (see Section 8).

If the RERR is sent in response to an undeliverable IP packet or RREP 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 an 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 is supported and configured for use

   * 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 existing route, if the reported sequence number is determined to be newer using the comparison technique described in Section 4.4.

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. If approaching the limit, the message should be sent if the priorities in Section 6.5 allow it.

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.

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

The AODVv2 message is used to create a corresponding [RFC5444] 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's route messages comprise Data Elements that map to message elements in [RFC5444].

[RFC5444] provides a multiplexed transport for multiple protocols. An [RFC5444] multiplexer MAY choose to optimize the content of certain message elements to reduce control plane overhead.

A brief summary of the [RFC5444] 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, or with a contiguous set of addresses, or with a single address in the Address Block

AODVv2 does not require access to the [RFC5444] packet header.

In the message header, AODVv2 uses <msg-hop-limit>, <msg-hop-count>, <msg-type> and <msg-addr-length>. The <msg-addr-length> field 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 by the [RFC5444] implementation. 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 an RERR. [RFC5444] maps this information to Address Block TLVs associated with the relevant addresses in the Address Block.

The following sections show how AODVv2 Data Elements are represented in [RFC5444] messages. AODVv2 makes use of the VALIDITY_TIME TLV from [RFC5497], and defines (in Section 12) a number of new TLVs.

Where the extension type of a TLV is set to zero, this is the default [RFC5444] value and the extension type will not be included in the message.

8.1. Route Request Message Representation

8.1.1. Message Header

Data Element	Header Field	Value
None	<msg-type>	RREQ
msg_hop_limit	<msg-hop-limit>	MAX_HOPCOUNT, reduced by number of hops traversed so far by the message.
msg_hop_count	<msg-hop-count>	Number of hops traversed so far by the message.

8.1.2. Message TLV Block

An RREQ contains no Message TLVs.

8.1.3. Address Block

An 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.

Data Elements	Address Block
OrigAddr/OrigPrefixLen	<address> + <prefix-length>
TargAddr/TargPrefixLen	<address> + <prefix-length>

8.1.4. Address Block TLV Block

Address Block TLVs are always associated with one or more 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

Data Element	TLV Type	Extension Type	Value
None	ADDRESS_TYPE	0	ADDRTYPE_ORIGADDR
OrigSeqNum	SEQ_NUM	0	Sequence Number of RREQ_Gen, the router which initiated route discovery.
OrigMetric /MetricType	PATH_METRIC	MetricType	Metric value for the route to OrigAddr, using MetricType.
ValidityTime	VALIDITY_TIME	0	ValidityTime for route to OrigAddr.

8.1.4.2. Address Block TLVs for TargAddr

Data Element	TLV Type	Extension Type	Value
None	ADDRESS_TYPE	0	ADDRTYPE_TARGADDR
TargSeqNum	SEQ_NUM	0	The last known TargSeqNum for TargAddr.

8.2. Route Reply Message Representation

8.2.1. Message Header

Data Element	Header Field	Value
None	<msg-type>	RREP
msg_hop_limit	<msg-hop-limit>	<msg-hop-count> from corresponding RREQ, reduced by number of hops traversed so far by the message.
msg_hop_count	<msg-hop-count>	Number of hops traversed so far by the message.

8.2.2. Message TLV Block

An RREP contains no Message TLVs.

8.2.3. Address Block

An 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.

Data Elements	Address Block
OrigAddr/OrigPrefixLen	<address> + <prefix-length>
TargAddr/TargPrefixLen	<address> + <prefix-length>
AckReq	<address> + <prefix-length>

8.2.4. Address Block TLV Block

Address Block TLVs are always associated with one or more 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

Data Element	TLV Type	Extension Type	Value
None	ADDRESS_TYPE	0	ADDRTYPE_ORIGADDR

8.2.4.2. Address Block TLVs for TargAddr

Data Element	TLV Type	Extension Type	Value
None	ADDRESS_TYPE	0	ADDRTYPE_TARGADDR
TargSeqNum	SEQ_NUM	0	Sequence number of RREP_Gen, the router which created the RREP.
TargMetric /MetricType	PATH_METRIC	MetricType	Metric value for the route to TargAddr, using MetricType.
ValidityTime	VALIDITY_TIME	0	ValidityTime for route to TargAddr.

8.2.4.3. Address Block TLVs for AckReq Intended Recipient Address

Data Element	TLV Type	Extension Type	Value
None	ADDRESS_TYPE	0	ADDRTYPE_INTEND

8.3. Route Reply Acknowledgement Message Representation

8.3.1. Message Header

Data Element	Header Field	Value
None	<msg-type>	RREP_Ack

8.3.2. Message TLV Block

An RREP_Ack contains no Message TLVs.

8.3.3. Address Block

An RREP_Ack contains no Address Block.

8.3.4. Address Block TLV Block

An RREP_Ack contains no Address Block TLV Block.

8.4. Route Error Message Representation

8.4.1. Message Header

Data Element	Header Field	Value
None	<msg-type>	RERR
msg_hop_limit	<msg-hop-limit>	MAX_HOPCOUNT, reduced by number of hops traversed so far by the message.

8.4.2. Message TLV Block

An RERR contains no Message TLVs.

8.4.3. Address Block

The Address Block in an RERR MAY contain PktSource, the source address of the IP 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.

Data Element	Address Block
PktSource	<address> + <prefix-length> for PktSource
AddressList/PrefixLengthList	<address> + <prefix-length> for each unreachable address in AddressList

8.4.4. Address Block TLV Block

Address Block TLVs are always associated with one or more 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

Data Element	TLV Type	Extension Type	Value
PktSource	ADDRESS_TYPE	0	ADDRTYPE_PKTSOURCE

8.4.4.2. Address Block TLVs for Unreachable Addresses

Data Element	TLV Type	Extension Type	Value
None	ADDRESS_TYPE	0	ADDRTYPE_UNREACHABLE
SeqNumList	SEQ_NUM	0	Sequence Number associated with invalid route to the unreachable address.
MetricTypeList	PATH_METRIC	MetricType	None. Extension Type set to MetricType of the route to the unreachable address.

9. Simple External Network Attachment

Figure 4 shows a stub (i.e., non-transit) network of AODVv2 routers which is attached to an external network via a single External Network Access Router (ENAR). The interface to the external network MUST NOT be configured in the AODVv2_INTERFACES list.

As in any externally-attached network, AODVv2 routers and Router Clients that wish to be reachable from hosts on the external network MUST have IP addresses within the ENAR's routable and topologically correct prefix (i.e., 191.0.2.0/24 in Figure 4). This AODVv2 network and subnets within it will be advertised to the external network using procedures which are out of scope for this specification.

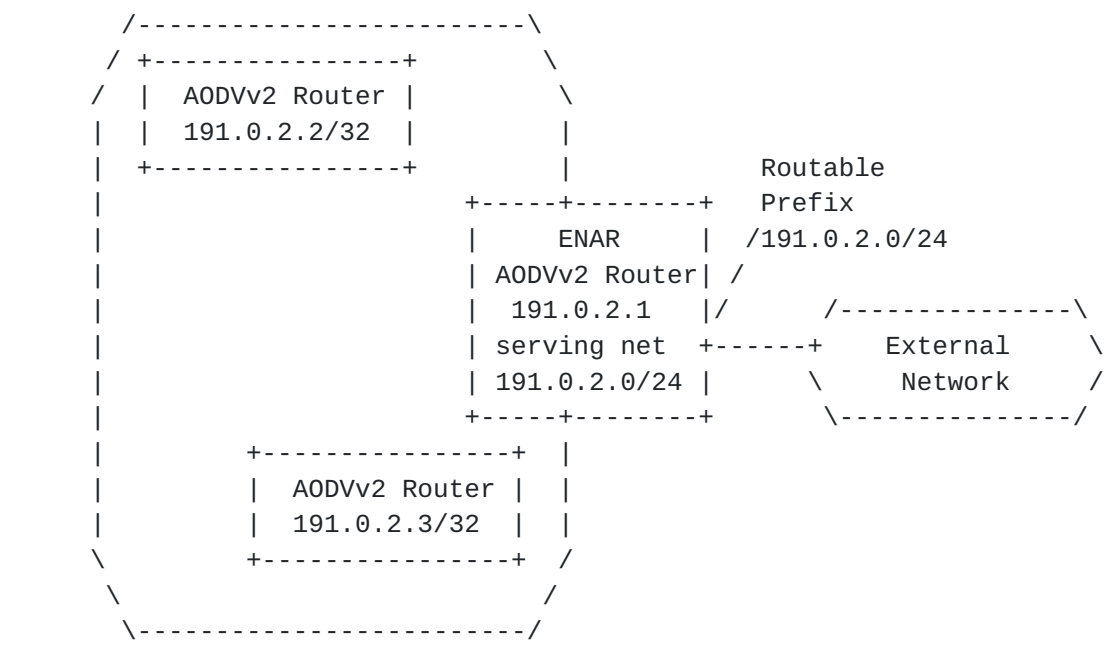

Figure 4: Simple External Network Attachment Example

When an AODVv2 router within the AODVv2 MANET wants to discover a route toward an address on the external network, it uses the normal AODVv2 route discovery for that IP Destination Address. The ENAR MUST respond to RREQ on behalf of all external network destinations, i.e., destinations not on the configured 191.0.2.0/24 subnet. RREQs for addresses inside the AODVv2 network, i.e. destinations on the configured 191.0.2.0/24 subnet, are handled using the standard processes described in Section 7.

When an IP packet from an address on the external network destined for an address in the AODVv2 MANET reaches the ENAR, if the ENAR does not have a route toward that exact destination it will perform normal AODVv2 route discovery for that destination.

Configuring the ENAR 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 an 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 IP packet's source address as the precursor. In this case, it is not guaranteed that an 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 IP 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 IP packet, determine which router forwarded the packet, and save the router address as a precursor. This ensures that when an 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 IP 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 an 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 an 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 [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

o Administrative parameters and controls

This section show 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:

Name	Default Value
ACTIVE_INTERVAL	5 second
MAX_IDLETIME	200 seconds
MAX_BLACKLIST_TIME	200 seconds
MAX_SEQNUM_LIFETIME	300 seconds
RteMsg_ENTRY_TIME	12 seconds
RREQ_WAIT_TIME	2 seconds
RREP_Ack_SENT_TIMEOUT	1 second
RREQ_HOLDDOWN_TIME	10 seconds

Table 2: Timing Parameter Values

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

If MAX_SEQNUM_LIFETIME was configured differently across the network, and any of the routers lost their sequence number or rebooted, this could result in their next route messages being classified as stale at any AODVv2 router using a greater value for MAX_SEQNUM_LIFETIME. This would delay route discovery from and to the re-initializing router.

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.

Name	Default	Description
DISCOVERY_ATTEMPTS_MAX	3	Section 6.6
RREP_RETRIES	2	Section 7.2.1
MAX_METRIC[MetricType]	[TBD]	Section 5
MAX_METRIC[HopCount]	20 hops	Section 5 and Section 7
MAX_HOPCOUNT	20	Same as MAX_METRIC[HopCount]
INFINITY_TIME	[TBD]	Maximum expressible clock time (Section 6.7.2)

Table 3: AODVv2 Constants

Note that <msg-hop-count> is an 8-bit field in the [RFC5444] message header and therefore MAX_HOPCOUNT cannot be larger than 255.

MAX_METRIC[MetricType] MUST always be the maximum expressible metric value of type MetricType. Field lengths associated with metric values are found in Section 11.6.

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

- RREP_RETRIES: Routers with lower values are more likely to blacklist neighbors when there is a

- MAX_METRIC[MetricType]: No interoperability problems due to variations on different routers, but routers with lower values may exhibit overly restrictive behavior during route comparisons. temporary fluctuation in link quality.

- MAX_HOPCOUNT: Routers with a value too small would not be able to discover routes to distant addresses.

- INFINITY_TIME: No interoperability problems due to variations on different routers, but if a lower value is used, route state management may exhibit overly restrictive behavior.

11.3. Local Settings

The following table lists AODVv2 parameters which SHOULD be administratively configured for each router:

Name	Default Value	Description
AODVv2_INTERFACES		Section 3
BUFFER_SIZE_PACKETS	2	Section 6.6
BUFFER_SIZE_BYTES	MAX_PACKET_SIZE [TBD]	Section 6.6
CONTROL_TRAFFIC_LIMIT	[TBD - 50 pkts/sec?]	Section 7

Table 4: Configuration for Local Settings

11.4. Network-Wide 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 routers in the network will not result in protocol errors, but poor performance may result.

Name	Default	Description
ENABLE_IDLE_IN_RERR	Disabled	Section 7.4.1

Table 5: Configuration for Network-Wide Settings

11.5. Optional Feature 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.

Name	Default	Description
PRECURSOR_LISTS	Disabled	Local (Section 10.2)
MSG_AGGREGATION	Disabled	Local (Section 10.4)
ENABLE_IRREP	Disabled	Network-wide (Section 10.3)
EXPANDING_RINGS_MULTICAST	Disabled	Network-wide (Section 10.1)

Table 6: Configuration for Optional Features

11.6. MetricType Allocation

The metric types used by AODVv2 are identified according to the assignments in [RFC6551]. All implementations MUST use these values.

Name of MetricType	Type	Metric Value Size
Unassigned	0	Undefined
Hop Count	3 [TBD]	1 octet
Unallocated	9 - 254	TBD
Reserved	255	Undefined

Table 7: AODVv2 Metric Types

11.7. AddressType Allocation

These values are used in the [RFC5444] Address Type TLV discussed in Section 8. All implementations MUST use these values.

Address Type	Value
ADDRTYPE_ORIGADDR	0
ADDRTYPE_TARGADDR	1
ADDRTYPE_UNREACHABLE	2
ADDRTYPE_PKTSOURCE	3
ADDRTYPE_INTEND	4
ADDRTYPE_UNSPECIFIED	255

Table 8: AODVv2 Address Types

12. IANA Considerations

This section specifies several [RFC5444] message types and address tlv-types required for AODVv2. A registry of metric types is specified, in addition to a registry of address types.

12.1. RFC 5444 Message Types

This specification defines four Message Types, to be allocated from the 0-223 range of the "Message Types" namespace defined in [RFC5444], as specified in Table 9.

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

Table 9: AODVv2 Message Types

12.2. RFC 5444 Address Block TLV Types

This specification defines three Address Block TLV Types, to be allocated from the "Address Block TLV Types" namespace defined in [RFC5444], as specified in Table 10.

Name of TLV	Type	Length (octets)	Reference
PATH_METRIC	10 (TBD)	depends on MetricType	Section 7
SEQ_NUM	11 (TBD)	2	Section 7
ADDRESS_TYPE	15 (TBD)	1	Section 8

Table 10: AODVv2 Address Block TLV 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. AODVv2 routers store the information contained in these messages in order to properly forward IP packets, and they generally provide this information to other AODVv2 routers. Negative routing information (i.e. a route does not exist) is distributed via RERR messages. AODVv2 routers process these messages and remove routes, and forward 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 routers 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 an RREQ message with false information is that traffic to OrigAddr could be disrupted. Since RREQ is multicast and likely to be received by all routers 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 an 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 an RREP_Ack message with false information is that the route advertised to a target address in an 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 an 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 of AODVv2 and DYMO, as well as numerous specification suggestions.

15. References

15.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3561] Perkins, C., Belding-Royer, E., and S. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing", RFC 3561, DOI 10.17487/RFC3561, July 2003, <http://www.rfc-editor.org/info/rfc3561>.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <http://www.rfc-editor.org/info/rfc4291>.

[RFC5082] Gill, V., Heasley, J., Meyer, D., Savola, P., Ed., and C. Pignataro, "The Generalized TTL Security Mechanism (GTSM)", RFC 5082, DOI 10.17487/RFC5082, October 2007, <http://www.rfc-editor.org/info/rfc5082>.

[RFC5444] Clausen, T., Dearlove, C., Dean, J., and C. Adjih, "Generalized Mobile Ad Hoc Network (MANET) Packet/Message Format", RFC 5444, DOI 10.17487/RFC5444, February 2009, <http://www.rfc-editor.org/info/rfc5444>.

[RFC5497] Clausen, T. and C. Dearlove, "Representing Multi-Value Time in Mobile Ad Hoc Networks (MANETs)", RFC 5497, DOI 10.17487/RFC5497, March 2009, <http://www.rfc-editor.org/info/rfc5497>.

[RFC5498] Chakeres, I., "IANA Allocations for Mobile Ad Hoc Network (MANET) Protocols", RFC 5498, DOI 10.17487/RFC5498, March 2009, <http://www.rfc-editor.org/info/rfc5498>.

[RFC6551] Vasseur, JP., Ed., Kim, M., Ed., Pister, K., Dejean, N., and D. Barthel, "Routing Metrics Used for Path Calculation in Low-Power and Lossy Networks", RFC 6551, DOI 10.17487/RFC6551, March 2012, <http://www.rfc-editor.org/info/rfc6551>.

[RFC7182] Herberg, U., Clausen, T., and C. Dearlove, "Integrity Check Value and Timestamp TLV Definitions for Mobile Ad Hoc Networks (MANETs)", RFC 7182, DOI 10.17487/RFC7182, April 2014, <http://www.rfc-editor.org/info/rfc7182>.

15.2. Informative References

[I-D.perkins-irrep]
Perkins, C., "Intermediate RREP for dynamic MANET On-demand (AODVv2) Routing", draft-perkins-irrep-03 (work in progress), May 2015.

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

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

[RFC2501] Corson, S. and J. Macker, "Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations", RFC 2501, DOI 10.17487/RFC2501, January 1999, <http://www.rfc-editor.org/info/rfc2501>.

[RFC4193] Hinden, R. and B. Haberman, "Unique Local IPv6 Unicast Addresses", RFC 4193, DOI 10.17487/RFC4193, October 2005, <http://www.rfc-editor.org/info/rfc4193>.

[RFC4728] Johnson, D., Hu, Y., and D. Maltz, "The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4", RFC 4728, DOI 10.17487/RFC4728, February 2007, <http://www.rfc-editor.org/info/rfc4728>.

[RFC4861] Narten, T., Nordmark, E., Simpson, W., and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, DOI 10.17487/RFC4861, September 2007, <http://www.rfc-editor.org/info/rfc4861>.

[RFC5148] Clausen, T., Dearlove, C., and B. Adamson, "Jitter Considerations in Mobile Ad Hoc Networks (MANETs)", RFC 5148, DOI 10.17487/RFC5148, February 2008, <http://www.rfc-editor.org/info/rfc5148>.

[RFC6130] Clausen, T., Dearlove, C., and J. Dean, "Mobile Ad Hoc Network (MANET) Neighborhood Discovery Protocol (NHDP)", RFC 6130, DOI 10.17487/RFC6130, April 2011, <http://www.rfc-editor.org/info/rfc6130>.

[RFC6621] Macker, J., Ed., "Simplified Multicast Forwarding", RFC 6621, DOI 10.17487/RFC6621, May 2012, <http://www.rfc-editor.org/info/rfc6621>.

[Sholander02] Sholander, P., Coccoli, P., Oakes, T., and S. Swank, "A Portable Software Implementation of a Hybrid MANET Routing Protocol", 2002.

Appendix A. Multi-homing Considerations

Multi-homing is not supported by the AODVv2 specification. A Router Client, i.e., an IP Address, can only be served by one AODVv2 router at any time. The coordination between multiple AODVv2 routers to distribute routing information correctly for a shared address is not defined. See Appendix B for information about how to move a router client to a different AODVv2 router.

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 B. 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 C. 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, and sometimes values and algorithms specific to each metric type.

The following table indicates the field names used in subsequent sections and their meaning.

Parameter	Description
RteMsg	A route message
	(inRREQ/outRREQ/inRREP/outRREP)
RteMsg.HopLimit	Hop limit for the message
RteMsg.HopCount	Hop count for the message
RteMsg.AckReq	True/False, optional in RREP
RteMsg.MetricType	The type of metric included, optional
RteMsg.OrigAddr	Address of source of queued data
RteMsg.TargAddr	Address route is requested for
RteMsg.OrigPrefixLen	Prefix length of OrigAddr, optional
RteMsg.TargPrefixLen	Prefix length of TargAddr, optional
RteMsg.OrigSeqNum	SeqNum of OrigAddr, in RREQ only
RteMsg.TargSeqNum	SeqNum of TargAddr, in RREP, optional
	in RREQ
RteMsg.OrigMetric	Metric to OrigAddr, in RREQ only
RteMsg.TargMetric	Metric to TargAddr, in RREP only
RteMsg.ValidityTime	Time limit for route advertised
RteMsg.NbrIP	Sender of the RteMsg
RteMsg.Netif	Interface on which the RteMsg arrived
AdvRte	Derived from a RteMsg (see Section 6.7)
AdvRte.Address	Route destination address
AdvRte.PrefixLength	Route destination prefix length
AdvRte.SeqNum	SeqNum associated with route
AdvRte.MetricType	MetricType associated with route
AdvRte.Metric	Advertised metric of route
AdvRte.Cost	Cost from receiving router
AdvRte.ValidityTime	Time limit for route advertised
AdvRte.NextHopIP	Sender of the RteMsg
AdvRte.NextHopIntf	Interface on which the RteMsg arrived
AdvRte.HopCount	Number of hops traversed
AdvRte.HopLimit	Allowed number of hops remaining
Route	A route table entry (see Section 4.6)
Route.Address	Route destination address
Route.PrefixLength	Route destination prefix length
Route.SeqNum	SeqNum associated with route

Route.NextHop	Address of router which advertised the route
Route.NextHopInterface	Interface on which next hop is reachable
Route.LastUsed	Time this route was last used for packet forwarding
Route.LastSeqNumUpdate	Time the SeqNum of the route was last updated
Route.ExpirationTime	Time at which the route will expire
Route.MetricType	MetricType associated with route
Route.Metric	Cost from receiving router
Route.State	Active/Idle/Invalid
Route.Precursors	Optional (see Section 10.2)
RERR	Route Error message (inRERR/outRERR)
RERR.HopLimit	Hop limit for the message
RERR.PktSource	Source address of packet which triggered RERR
RERR.AddressList[]	List of unreachable route addresses
RERR.PrefixLengthList[]	List of PrefixLengths for AddressList
RERR.SeqNumList[]	List of SeqNums for AddressList
RERR.MetricTypeList[]	MetricType for the invalid routes
RERR.Netif	Interface on which the RERR arrived

Table 11: Notation used in Appendix

C.1. HopCount MetricType

The HopCount MetricType defines:

- MAX_METRIC[HopCount] := MAX_HOPCOUNT. A constant defined in Section 11.2. MAX_HOPCOUNT is also used to limit the number of hops an AODVv2 message can travel, regardless of the MetricType in use. It MUST be larger than the AODVv2 network diameter, in order that AODVv2 protocol messages may reach their intended destinations.

- Cost(L) := 1

- Cost(R) := Sum of 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 (OrigAddr if RREQ, TargAddr if RREP)

- LoopFree(R1, R2) := (Cost(R1) <= Cost(R2)). This is derived from the fact that route cost increases with number of hops. Therefore, an advertised route with higher cost than the corresponding existing route could include the existing route as a

sub-section. Replacing the existing route with the advertised route could form a routing loop.

C.2. General Operations

General AODVv2 operations involve the comparisons of incoming and current data, and updates to local data sets.

C.2.1. Route Operations

C.2.1.1. 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 == INFINITY_TIME)) //not a timed route
        route.State := Invalid;
    if ((CurrentTime - route.LastUsed > ACTIVE_INTERVAL)
        AND (route.State != Unconfirmed)
        AND (route.ExpirationTime == INFINITY_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;
}
```

C.2.1.2. Process_Routing_Info

(See Section 6.7.1)

```
/* 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;
}
```

C.2.1.3. 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;
   }
```

C.2.1.4. 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;
       if (validityTime)
           rte.ExpirationTime := CurrentTime + advRte.ValidityTime;
       else
           rte.ExpirationTime := INFINITY_TIME;

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

C.2.1.5. 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)
           rte.ExpirationTime := CurrentTime + advRte.ValidityTime;
       else
           rte.ExpirationTime := INFINITY_TIME;
       rte.MetricType := advRte.MetricType;
       rte.Metric := advRte.Metric;
       rte.State := Idle (if advRte is from RREP);
                    or Unconfirmed (if advRte is from RREQ);
   }
```

C.2.2. 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;
   }
```

C.2.3. Multicast Route Message Table Operations

C.2.3.1. 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;
   }
```

C.2.3.2. 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;
}
```

C.3. Message Algorithms

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.

After processing a message, the most recent information is stored in the route table. For this reason, it is equally appropriate to set outgoing message field values using route table information or using fields from the incoming message.

C.3.1. 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 in MsgHdr, 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;
}
```

C.3.2. RREQ Operations

C.3.2.1. 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 */
}
```

C.3.2.2. 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 (msg_hop_count > MAX_HOPCOUNT)
         return;
     if (msg_hop_limit < 0)
         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);
```

```
 }
```

C.3.2.3. 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 */
}
```

C.3.3. RREP Operations

C.3.3.1. 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
         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 := inRREQ.Address;
    outRREP.TargAddr := rte.TargAddr;
    outRREP.TargPrefixLen := rte.PrefixLength; //if not address length
    outRREP.TargSeqNum := mySeqNum;
    outRREP.TargMetric := rte.Metric;
    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)
    {
        /* 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 */
}
```

C.3.3.2. Receive_RREP

```
   /*  Process a RREP received on link L */

   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 (msg_hop_count > MAX_HOPCOUNT)
           return;
       if (msg_hop_limit < 0)
           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])
            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);
    }
```

C.3.3.3. 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 */
}
```

C.3.4. RREP_Ack Operations

C.3.4.1. 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 */
  }
```

C.3.4.2. Receive_RREP_Ack

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

C.3.4.3. Timeout_RREP_Ack

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

C.3.5. RERR Operations

C.3.5.1. 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;
        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;
            }
        }
    }
    case (undeliverablePacket):
        doGenerate := TRUE;
        num-broken-addr := 1;
        outRERR.HopLimit := MAX_HOPCOUNT;
        outRERR.PktSource := triggerPkt.SrcIP;
                          or triggerPkt.TargAddr; //if pkt was a RREP
        outRERR.AddressList[0] := triggerPkt.DestIP;
                               or triggerPkt.OrigAddr; //if pkt was RREP
        /* optional to include outRERR.PrefixLengthList, outRERR.SeqNumList
           and outRERR.MetricTypeList */
    }

    if (doGenerate == FALSE)
        return;

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

```
    /*  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 */
}
```

C.3.5.2. 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.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;
    }

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

C.3.5.3. 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 D. AODVv2 Draft Updates

D.1. Changes between revisions 11 and 12

This section lists the changes between AODVv2 revisions ...-11.txt and ...-12.txt.

- Avoided use of "node" and "subnet" where possible.
- Improved separation of data structure information from protocol operation.
- Updated uses of the terms "IP address" and "packet" to be clearer.
- More consistent and accurate use of MUST, SHOULD, SHOULD NOT, and MAY, and added explanations of consequences of not implementing SHOULDs.
- Used consistent references to [RFC5444].
- Updated title to include "Version 2".
- Updated Overview to state differences from AODV, text about loop freedom and RFC 7182 in Overview.
- Updated Terminology and removed the Data Element table. Gave clearer definition of Router Client and Unreachable Address.
- Updated Applicability Statement to draw attention to requirements of the forwarding plane, handling of uni-directional links, usage of IP addresses on multiple interfaces, and description of gateway functionality. Added note about penalty for not storing persistent state.
- Updated Router Client section and added cost to Router Client entry.
- Clarified that Neighbor Table needs only information on neighboring routers on discovered routes.
- Updated Sequence Number section. Use only one sequence number per router. Added description of sequence number comparison.
- Updated descriptions of route states.
- Improved clarity of Metrics section, generic metric instead of hopcount, removed default metric type, added explanation of LoopFree.

- Improved Initialization section.
- Major update to Adjacency Monitoring section. Made it clear that if bidirectional connectivity is already confirmed, requesting acknowledgement is unnecessary. Separated Neighbor Table Updates into separate section.
- Updated description of message prioritization near the control message generation limit.
- Updated wording regarding [RFC6621].
- Added description of backoff used for message retries.
- Improved description of how unidirectional links are handled.
- Improved text regarding creation of Unconfirmed route entries.
- Improved section on determining redundancy of received multicast messages.
- Added section on interactions with the forwarding plane.
- Improved Route State section. Clarified action when Active route expires. Separated information on expunging routes on memory constrained routers.
- Updated RERR description to be clearer about triggers.
- Updated IANA section to include only newly defined Messages and TLVs, and define an Unspecified value for AddressType.
- Updated references.
- Updated section on Gateway behaviour.
- Updated Appendix D to include more checks on msg_hop_limit and msg_hop_count.
- Renamed MAX_TIME to INFINITY_TIME to make meaning clearer.

D.2. Changes between revisions 10 and 11

This section lists the changes between AODVv2 revisions ...-10.txt and ...-11.txt.

- Updated Simple Internet Attachment section to clarify behaviour of IAR for incoming RREQ messages.

D.3. Changes between revisions 9 and 10

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

- Updated [RFC5444] Representation section to add "Address Type" TLV, which explicitly declares the meaning of addresses in the [RFC5444] 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.

D.4. 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 [RFC5444] overview and simplified/rearranged text in this section.
- Major update to [RFC5444] representation section
- Updated RERR's [RFC5444] 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.
- Described use of extension type in Metric TLV to represent MetricType, and the interpretation when using the default metric type.

- Removed Multicast RREP as an optional feature.
- Updated Precursor Lists section to include options for precursor information to store.
- Updated Security Considerations.

D.5. Changes between revisions 7 and 8

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

- 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 [RFC5444] message.
- Updated Overview to be slightly more concise.
- Moved MetricType next to Metric when mentioned for better flow.
- Added text to Applicability to address comments on mailing list regarding gateway behavior and NHDP HELLO messages.
- Removed paragraph in AODVv2 Message Transmission section regarding TTL.
- Added reference where precursors are mentioned in route table entry.
- Added text to bidirectionality explanation regarding NHDP HELLO messages and lower layer triggers.
- Clarified blacklist removal with SHOULD rather than MAY.
- Removed pseudo-code from section on evaluating incoming routing information.
- Clarified rules for expunging route entries on memory-constrained devices.
- Clarified the use of exponential backoff for route discovery attempts.
- Small updates to message sections. Removed steps about checking if neighbors.
- Renamed [RFC5444] parser to multiplexer in Section 10.

- Removed "optional feature" to include multiple addresses in RERR.
- Removed MetricType from the Message TLV Type Specification.
- Updated Security Considerations.
- Added reference to RFC 7182.
- 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.

D.6. Changes between revisions 6 and 7

This section lists the changes since AODVv2 revision ...-06.txt

- Added Victoria Mercieca as co-author.
- Reorganized protocol message descriptions into major subsections for each protocol message. For protocol messages, organized processing into Generation, Reception, and Regeneration subsections.
- Separated RREQ and RREP message processing description into separate major subsection which had previously been combined into RteMsg description.
- 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.
- Moved sections for Multiple Interfaces and AODVv2 Control Message Generation Limits to be major subsections of the AODVv2 Protocol Operations section.
- Reorganized the protocol message processing steps into the subsections as previously described, adopting a more step-by-step presentation.
- 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 [RFC5444] packet formats.

- Clarification on the use of RFC 5497 VALIDITY_TIME.

- In Terminology, deleted superfluous definitions, added missing definitions.

- Numerous editorial improvements and clarifications.

D.7. 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 [RFC5444] 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 [RFC5444].

- 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 [RFC5444] 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.

D.8. Changes between revisions 4 and 5

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

- Normative text moved out of definitions into the relevant section of the body of the specification.

- Editorial improvements and improvements to consistent terminology were made. Replaced "retransmit" by the slightly more accurate term "regenerate".

- Issues were resolved as discussed on the mailing list.

- 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.

- Added text to clarify that different metrics may have different data types and different ranges of acceptable values.

- Added text to section "RteMsg Structure" to emphasize the proper use of [RFC5444].

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

- 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).

D.9. Changes between revisions 3 and 4

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

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

- Numerous editorial improvements and improvements to consistent terminology were made. Terminology related to prefix lengths was made consistent. Some items listed in "Notational Conventions" were no longer used, and so deleted.

- Issues were resolved as discussed on the mailing list.

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

- Definition inserted for "upstream".

- Route.Precursors included as an *optional* route table field

- Reworded text to avoid use of "relevant".
- Deleted references to "DestOnly" flag.
- Refined statements about 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.

D.10. 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 1.
- Example RREQ and RREP message formats shown in the Appendices were changed to use OrigSeqNum and TargSeqNum message TLVs instead of using the SeqNum message TLV.
- Inclusion of the OrigNode's SeqNum in the RREP message is not specified. The processing rules for the OrigNode's SeqNum were incompletely specified in previous versions of the draft, and very little benefit is foreseen for including that information, since reverse path forwarding is used for the RREP.
- Additional acknowledgements were included, and contributors names were alphabetized.

- Definitions in the Terminology section capitalize the term to be defined.
- Uncited bibliographic entries deleted.
- Ancient "Changes" sections were deleted.

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

Victoria Mercieca
Airbus Defence and Space
Celtic Springs
Newport, Wales NP10 8FZ
United Kingdom

Email: victoria.mercieca@airbus.com