Dynamic MANET On-demand (AODVv2) Routing
draft-ietf-manet-aodvv2-08

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on September 25, 2015.

Copyright Notice

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

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

1. Overview ................................................. 4
2. Terminology .............................................. 5
3. Data Elements and Notational Conventions ..................... 9
4. Applicability Statement .................................... 10
5. AODVv2 Message Transmission ............................... 12
6. Data Structures ........................................... 12
   6.1. Route Table Entry ...................................... 12
   6.2. Next-hop Router Adjacency Monitoring and Blacklists .... 14
   6.3. Router Clients and Client Networks ...................... 15
   6.4. Sequence Numbers ....................................... 16
   6.5. Table for Multicast RteMsgs ............................. 17
7. Metrics .................................................... 18
   7.1. The Cost() function .................................... 19
   7.2. The LoopFree() function ............................... 19
   7.3. Default Metric type .................................... 19
   7.4. Alternate Metrics ...................................... 19
8. AODVv2 Protocol Operations .................................. 20
   8.1. Evaluating Incoming Routing Information ................. 20
   8.2. Applying Route Updates To Route Table Entries .......... 22
   8.3. Route Maintenance ..................................... 23
   8.4. Route Table Entry Timeouts ............................. 23
   8.5. Route Discovery, Retries and Buffering .................. 24
   8.6. Suppressing Redundant RteMsgs .......................... 25
9. AODVv2 Protocol Messages ................................... 26
   9.1. RREQ Messages .......................................... 27
      9.1.1. RREQ Generation ..................................... 28
      9.1.2. RREQ Reception ...................................... 29
      9.1.3. RREQ Regeneration ................................... 31
   9.2. RREP Messages .......................................... 32
      9.2.1. RREP Generation ...................................... 33
      9.2.2. RREP Reception ...................................... 34
      9.2.3. RREP Regeneration ................................... 35
   9.3. RERR Messages .......................................... 36
      9.3.1. RERR Generation ..................................... 37
      9.3.2. RERR Reception ...................................... 39
      9.3.3. RERR Regeneration ................................... 40
   9.4. RREP_Ack Messages ...................................... 41
      9.4.1. RREP_Ack Generation .................................. 41
9.4.2. RREP_Ack Reception .................................. 41
10. Representing AODVv2 data elements using RFC 5444 ........... 42
11. Simple Internet Attachment ................................ 43
12. Optional Features ......................................... 44
  12.1. Expanding Rings Multicast .............................. 45
  12.2. Precursor Lists and Notifications ...................... 45
    12.2.1. Overview ...................................... 45
    12.2.2. Precursor Notification Details .................... 45
  12.3. Multicast RREP Response to RREQ ........................ 46
  12.4. Intermediate RREP .................................... 46
  12.5. Message Aggregation Delay ............................. 46
13. Administratively Configurable Parameters and Timer Values ... 47
  13.1. Timers ............................................... 47
  13.2. Protocol Constants ................................... 48
  13.3. Administrative (functional) controls .................. 49
  13.4. Other administrative parameters and lists .............. 49
14. IANA Considerations ....................................... 49
  14.1. AODVv2 Message Types Specification .................... 50
  14.2. Message TLV Type Specification ........................ 50
  14.3. Address Block TLV Specification ........................ 50
  14.4. MetricType Number Allocation .......................... 50
15. Security Considerations ................................... 51
16. Acknowledgments ........................................... 52
17. References ................................................ 52
  17.1. Normative References ................................ 52
  17.2. Informative References ............................... 53
Appendix A. Example Algorithms for AODVv2 Protocol Operations ... 54
  A.1. Subroutines for AODVv2 Operations ....................... 58
    A.1.1. Process_Routing_Info ................................ 58
    A.1.2. Fetch_Route_Table_Entry ............................. 59
    A.1.3. Update_Route_Table_Entry ............................ 60
    A.1.4. Create_Route_Table_Entry ............................ 60
    A.1.5. LoopFree .......................................... 61
    A.1.6. Fetch_Rte_Msg_Table_Entry ......................... 62
    A.1.7. Update_Rte_Msg_Table .............................. 62
    A.1.8. Build_RFC_5444_message_header ...................... 64
  A.2. Example Algorithms for AODVv2 RREQ Operations ............ 64
    A.2.1. Generate_RREQ ..................................... 64
    A.2.2. Receive_RREQ ..................................... 65
    A.2.3. Regenerate_RREQ ................................... 66
  A.3. Example Algorithms for AODVv2 RREP Operations ............ 68
    A.3.1. Generate_RREP ..................................... 68
    A.3.2. Receive_RREP ..................................... 69
    A.3.3. Regenerate_RREP ................................... 71
  A.4. Example Algorithms for AODVv2 RERR Operations ............ 72
    A.4.1. Generate_RERR ..................................... 73
    A.4.2. Receive_RERR ..................................... 74
    A.4.3. Regenerate_RERR ................................... 76
The revised Ad Hoc On-demand Distance Vector (AODVv2) routing protocol [formerly named DYMO] enables on-demand, multihop unicast routing among AODVv2 routers in mobile ad hoc networks [MANETs][RFC2501]. The basic operations of the AODVv2 protocol are route discovery and route maintenance. Route discovery is performed when an AODVv2 router must transmit a packet towards a destination for which it does not have a route. Route maintenance is performed to avoid prematurely expunging routes from the route table, and to avoid dropping packets when a route breaks.

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

Route maintenance consists of two operations: continuously extending the lifetime of active routes, and using Route Error (RERR) message to invalidate routes that cannot be used to forward packets. In order to maintain routes, AODVv2 routers extend route lifetimes upon successfully forwarding a packet. When a data packet is received to be forwarded and no valid route exists, then the upstream routers and AODVv2 router of the source of the packet is notified of the error by way of an RERR message. Route discovery would re-establish the
route. RERR messages are also used to notify upstream routers when routes break (say, due to loss of a link to a neighbor).

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

See Section 10 for the mapping of AODVv2 data elements to RFC 5444 Address Block, Address TLV, and Message TLV formats. Security for authentication of AODVv2 routers, and/or encryption of traffic is dealt with by the underlying transport mechanism (e.g., by using the techniques for Authentication, Integrity, and Confidentiality documented in [RFC5444]).

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:

Adjacency
A bi-directional relationship between neighboring AODVv2 routers for the purpose of exchanging routing information. Not every pair of neighboring routers will necessarily form an adjacency. Monitoring of adjacencies where packets are being forwarded is required (see Section 6.2).

AckReq
Request for acknowledgement (of an RREP message).

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

Current_Time
The current time as maintained by the AODVv2 router.

Data Element
A named object used within AODVv2 protocol messages

Disregard
Ignore for further processing.

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

Invalid route
A route that cannot be used for forwarding.

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

MetricList
The metrics associated with the addresses in an AddressList.

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

OrigAddr
An IP address of the Originating Node used as a data element within AODVv2 messages.

OrigAddrMetric
The metric associated with the route to OrigAddr.

OrigSeqNum
The Sequence Number maintained by OrigNode for OrigAddr.

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

PktSource
The source address of a packet sent to an unreachable address.

PrefixLengthList
The prefix lengths associated with addresses in an AddressList.

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

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

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

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

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

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

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

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

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

Sequence Number (SeqNum)
A Sequence Number is an unsigned integer maintained by an AODVv2 router to avoid re-use of stale messages. The router associates SeqNum with an IP address of one or more of its network interfaces. The value zero (0) is reserved to indicate that the Sequence Number for an address is unknown.
The list of Sequence Numbers associated with addresses in an AddressList, used in RERR messages.

TargAddr
An IP address of the Target Node used as a data element within AODVv2 messages.

TargAddrMetric
The metric associated with the route to TargAddr.

TargSeqNum
The Sequence Number maintained by TargNode for TargAddr.

Target Node (TargNode)
The node hosting the IP address towards which a route is needed.

Type-Length-Value structure (TLV)
A generic way to represent information, for example as used in [RFC5444].

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

upstream
In the direction from TargAddr to OrigAddr.

Valid route
A route that can be used for forwarding.

ValidityTime
The duration of time for which a route should be considered to be a valid route.
### 3. Data Elements and Notational Conventions

This document uses the Data Elements and conventions found in Table 1 and Table 2.

<table>
<thead>
<tr>
<th>Data Elements</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>msg_hop_limit</td>
<td>Number of hops allowable for the message</td>
</tr>
<tr>
<td>msg_hop_count</td>
<td>Number of hops traversed so far by the message</td>
</tr>
<tr>
<td>AckReq</td>
<td>Acknowledgement Requested for RREP</td>
</tr>
<tr>
<td>PktSource</td>
<td>Source address of a data packet</td>
</tr>
<tr>
<td>AddressList</td>
<td>A list of IP addresses</td>
</tr>
<tr>
<td>OrigAddr</td>
<td>IP address of the Originating Node</td>
</tr>
<tr>
<td>TargAddr</td>
<td>IP address of the Target Node</td>
</tr>
<tr>
<td>UnreachableAddress</td>
<td>An unreachable IP address</td>
</tr>
<tr>
<td>PrefixLengthList</td>
<td>Routing prefixes associated with addresses in AddressList</td>
</tr>
<tr>
<td>SeqNum</td>
<td>Sequence Number, used in RERR messages</td>
</tr>
<tr>
<td>SeqNumList</td>
<td>A list of SeqNums</td>
</tr>
<tr>
<td>OrigSeqNum</td>
<td>Originating Node Sequence Number</td>
</tr>
<tr>
<td>TargSeqNum</td>
<td>Target Node Sequence Number</td>
</tr>
<tr>
<td>MetricType</td>
<td>The metric type for values in MetricList</td>
</tr>
<tr>
<td>MetricList</td>
<td>Metric values for routes to addresses in AddressList</td>
</tr>
<tr>
<td>OrigAddrMetric</td>
<td>Metric value for route to OrigAddr</td>
</tr>
<tr>
<td>TargAddrMetric</td>
<td>Metric value for route to TargAddr</td>
</tr>
<tr>
<td>ValidityTime</td>
<td>Included in ValidityTimeList</td>
</tr>
<tr>
<td>ValidityTimeList</td>
<td>ValidityTime values for routes to Addresses in AddressList</td>
</tr>
</tbody>
</table>

Table 1
4. Applicability Statement

The AODVv2 routing protocol is a reactive routing protocol designed for stub (i.e., non-transit) or disconnected (i.e., from the Internet) mobile ad hoc networks (MANETs). AODVv2 handles a wide variety of mobility patterns by determining routes on-demand. AODVv2 also handles a wide variety of traffic patterns. In networks with a large number of routers, AODVv2 is best suited for relatively sparse traffic scenarios where any particular router forwards packets to only a small percentage of the AODVv2 routers in the network, due to the on-demand nature of route discovery and route maintenance. AODVv2 supports routers with multiple interfaces, as long as each interface has its own (unicast routeable) IP address; the set of all network interfaces supporting AODVv2 is administratively configured in a list (namely, AODVv2_INTERFACES).

Ad Hoc networks have been deployed in many circumstances, including for emergency and disaster relief. In those circumstances, it is sometimes the case that the simple ability to communicate is much more important than being assured of secure operations. AODVv2 is very well suited for such reactive scenarios. For other ad hoc networking applications, in which insecure operation could negate the value of establishing communication paths, it is important for neighboring AODVv2 nodes to establish security associations with one another.

Although AODVv2 is closely related to AODV [RFC3561], and shares some features of DSR [RFC4728], AODVv2 is not interoperable with either of those other two protocols.
AODVv2 is applicable to memory constrained devices, since only a little routing state is maintained in each AODVv2 router. Routes that are not needed for forwarding data do not have to be maintained, in contrast to proactive routing protocols that require routing information to all routers within the MANET be maintained.

In addition to routing for its own local applications, each AODVv2 router can also route on behalf of other non-routing nodes (in this document, "Router Clients") that are directly reachable via its network interfaces. Each AODVv2 router, if serving router clients other than itself, SHOULD be configured with information about the IP addresses of its clients, using any suitable method. In the initial state, no AODVv2 router is required to have information about the relationship between any other AODVv2 router and its Router Clients (see Section 6.3).

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

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

By default, AODVv2 only supports bidirectional links. In the case of possible unidirectional links, blacklists (see Section 6.2) SHOULD be used, or other means (e.g. adjacency establishment with only neighboring routers that have bidirectional communication as indicated by NHDP HELLO messages [RFC6130]) of assuring and monitoring bi-directionality are recommended. Otherwise, persistent packet loss or persistent protocol failures could occur. If received over a link that is unidirectional, metric information from incoming AODVv2 messages MUST NOT be used for route table updates.

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

In its default mode of operation, AODVv2 sends messages using the parameters for port number and IP protocol specified in [RFC5498]. Unless otherwise specified, the address for AODVv2 multicast messages (for example, RREQ or RERR) is the link-local multicast address LL-MANET-Routers [RFC5498]. All AODVv2 routers MUST subscribe to LL-MANET-Routers [RFC5498] to receive AODVv2 messages. Implementations are free to choose their own heuristics for reducing multicast overhead. Some methods for doing so are described in [RFC6621]. AODVv2 does not specify which method should be used to restrict the set of AODVv2 routers that have the responsibility to regenerate multicast packets. Note that multicast packets MAY be sent via unicast. For example, this may occur for certain link-types (non-broadcast media), for manually configured router adjacencies, or in order to improve robustness.

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

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

6. Data Structures

6.1. Route Table Entry

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

A route table entry has the following fields:

Route.Address  
An address or address prefix of a node

Route.PrefixLength  
The length of the address or prefix. If the value of Route.PrefixLength is less than the length of Route.Address, the route can be thought of as a route to the subnet on which Route.Address resides. A PrefixLength is stored for every route in the route table.

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

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

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

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

Route.LastSeqNum
The time that the destination SeqNum for this route was last updated

Route.ExpirationTime
The time at which this route must be marked as Invalid

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

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

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

Route.Timed
TRUE if the route was specified to have a ValidityTime

Route.Precursors (optional)
A list of upstream neighbors using the route (see Section 12.2)

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

Active
An Active route is in current use for forwarding packets. An Active route is maintained continuously by AODVv2 and is considered to remain active as long as it is used at least once during every ACTIVE_INTERVAL, or if the Route.Timed flag is true. When a route that is not a timed route is no longer active the route becomes an Idle route.
Idle
An Idle route can be used for forwarding packets, even though it is not in current use. If an Idle route is used to forward a packet, it becomes an Active route once again. After an Idle route remains idle for MAX_IDLETIME, it becomes an Invalid route.

Invalid
A route marked as Invalid cannot be used for forwarding, but the sequence number information MAY be maintained until the destination sequence number has not had any updates for MAX_SEQNUM_LIFETIME; after that time, old sequence number information may no longer be valid and the Invalid route MUST be expunged.

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

The invalidation of a Timed route is controlled by the ExpirationTime time of the route table entry (instead of MAX_IDLETIME). Until that time, a Timed route can be used for forwarding packets. A route is indicated to be a Timed route by the setting of the Timed flag in the route table entry. Afterwards, the route MAY be expunged; otherwise the route must be marked as Invalid.

6.2. Next-hop Router Adjacency Monitoring and Blacklists

Neighboring routers MAY form an adjacency based on AODVv2 messages, other protocols (e.g. NDP [RFC4861] or NHDP [RFC6130]), or manual configuration. Loss of a routing adjacency may also be indicated similarly. AODVv2 routers SHOULD monitor connectivity to adjacent routers along active routes. In the absence of other information about bidirectional connectivity, the default approach for AODVv2 routers to monitor connectivity to neighboring AODVv2 routers is to include the AckReq data element in RREP messages, and send RREP_Ack messages to fulfill the requests (see Sections 9.2 and 9.4). However, when routers perform other operations such as those from the list below, these can also be used as indications of connectivity.

- NHDP HELLO Messages [RFC6130], if is implemented by its neighbors
- Route timeout
- Lower layer triggers, e.g. message reception or link status notifications
o  TCP timeouts

o  Promiscuous listening

o  Other monitoring mechanisms or heuristics

For example, receipt of a Neighborhood Discovery message would signal a connection to the sender. In this case, the AODVv2 router doesn’t need to request an acknowledgement in the RREP. Similarly, if AODVv2 received notification of a timeout, this may possibly be due to a disconnection, and the AODVv2 router SHOULD attempt to verify connectivity by including AckReq data element when sending a RREP to that neighbor.

When a link to a neighbor is determined to be unidirectional, either by failure to respond with a RREP_Ack as requested, or by some other means, the neighbor MUST be placed in a blacklist. However, the blacklisted neighbor SHOULD NOT be permanently blacklisted; after a certain time (MAX_BLACKLIST_TIME), it SHOULD once again be considered as a viable neighbor for route discovery operations.

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

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

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

RREQs received from a blacklisted router, or any router over a link that is known to be incoming-only, MUST be disregarded. If other indications are received that a blacklisted router has restored bidirectional connectivity, for instance receiving NHDP HELLO messages, then the router SHOULD be immediately removed from the blacklist.

6.3. Router Clients and Client Networks

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

For this purpose, CLIENT_ADDRESSES must be configured on each AODVv2 router with the following information:
Client IP address
The IP address of the node that requires routing service from the AODVv2 router.

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

The list of Routing Clients for an AODVv2 router is never empty, since an AODVv2 router is always its own client as well. If the Client Prefix Length is not the full length of the Client IP address, then the prefix defines a Client Network. If an AODVv2 router is configured to serve a Client Network, then the AODVv2 router MUST serve every node that has an address within the range defined by the routing prefix of the Client Network.

6.4. Sequence Numbers

Sequence Numbers allow AODVv2 routers to evaluate the freshness of routing information. Each AODVv2 router in the network MUST maintain its own sequence number (SeqNum). Each RREQ and RREP generated by an AODVv2 router includes its SeqNum. Each AODVv2 router MUST ensure that its SeqNum is monotonically increasing. The router can ensure this by incrementing SeqNum whenever it generates RREQ or RREP.

A router receiving a RREQ or RREP message uses the Sequence Number in the message to determine the freshness of a route update: if a new Sequence Number in the message is lower than the one stored in the route table, the stored information for that route is considered stale.

As a consequence, loop freedom is assured.

If the router has multiple network interfaces, it can use the same SeqNum for the IP addresses of all of them, or it can assign different SeqNums for use with different IP addresses. However, the router MUST NOT use multiple SeqNums for any particular IP address. A Router Client has the same SeqNum as the IP address of the network interface that the AODVv2 router uses to forward packets to that Router Client. Similarly, a route to a subnet has the same SeqNum as the IP address of the network interface that the AODVv2 router uses to forward packets to that subnet. The Sequence Number fulfills the same role as the "Destination Sequence Number" of DSDV [Perkins94], and as the AODV Sequence Number in RFC 3561 [RFC3561].

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

An AODVv2 router SHOULD maintain its SeqNum in persistent storage. If an AODVv2 router’s SeqNum is lost, it MUST take the following actions to avoid the danger of routing loops. First, the AODVv2 router MUST set Route.State := Invalid for each entry. Furthermore the AODVv2 router MUST wait for at least MAX_SEQNUM_LIFETIME before transmitting or regenerating any AODVv2 RREQ or RREP messages. If an AODVv2 protocol message is received during this waiting period, the AODVv2 router SHOULD perform normal route table entry updates, but not forward the message to other nodes. If, during this waiting period, a data packet is received to be forwarded to another destination that is not among the router’s Clients, then the AODVv2 router MUST transmit a RERR message indicating that no route is available. However, packets destined to a Client are forwarded as usual. At the end of the waiting period the AODVv2 router sets its SeqNum to one (1) and begins performing AODVv2 protocol operations again.

6.5. Table for Multicast RteMsgs

Two multicast RteMsgs (i.e., RREQ or RREP) are considered to be "comparable" if they have the same Message Type, OrigAddr, TargAddr, and MetricType. When RteMsgs are flooded in a MANET, an AODVv2 router may well receive such comparable RteMsgs from its neighbors. A router, after receiving a RteMsg, MUST check against previous RteMsgs to assure that its response message would contain information that is not redundant. Otherwise, multicast RteMsgs are likely to be regenerated repeatedly with almost no additional benefit, but generating a great deal of unnecessary signaling traffic and interference. See Section 8.6 regarding suppression of redundant RteMsgs.

To avoid transmission of redundant RteMsgs, while still enabling the proper handling of earlier RteMsgs that may have somehow been delayed in the network, each AODVv2 router keeps a list of certain information about recently received RteMsgs. This list is called the AODVv2 Multicast RteMsg Table -- or, more briefly, the RteMsg Table.

Each entry in the RteMsg Table has the following fields:

- Message Type (either RREQ or RREP)
- OrigAddr
- TargAddr
The RteMsg Table is maintained so that no two entries in the RteMsg Table are comparable -- that is, all RteMsgs represented in the RteMsg Table either have different Message Types, different OrigAddr, different TargAddr, or different metric types. If two RteMsgs have the same Message Type, MetricType, OrigAddr, and TargAddr, the information from the one with the older Sequence Number is not needed in the table; in case they have the same Sequence Number, the one with the greater Metric value is not needed; in case they have the same Metric as well, it does not matter which table entry is maintained. Whenever a RteMsg Table entry is updated, its Timestamp field MUST also set to be the Current_Time.

7. Metrics

Metrics measure a cost or quality associated to a route or a link. They can account for various characteristics such as latency, delay, financial, energy, etc. A metric value is included in each routing table entry. Determining whether to use incoming information about a route requires comparing metric values. Whenever an AODV router receives metric information in an incoming message, the received value of the metric is as measured by the neighbor router, and does not reflect the cost of traversing the link to that neighbor.

Each metric has a MetricType, which is allocated by IANA as specified in [RFC6551]. Apart from its default metric type as detailed in Section 7.3, AODVv2 enables the use of monotonically increasing metrics, whose data type depends on the metric used. Using non-default metrics in a RteMsg requires the inclusion of the MetricType data element. Routes are looked up according to metric type, and intermediate routers handling a RteMsg assign the same metric type to all metric information in the RteMsg.

For each type of metric, a maximum value is defined, denoted MAX_METRIC[i] where ‘i’ is the MetricType. AODVv2 cannot store routes in its route table that cost more than MAX_METRIC[i].
7.1. The Cost() function

In order to simplify the description of storing accumulated route costs in the route table, a Cost() function is defined. This function returns the Cost of traversing a Route ('Cost(R)') or a Link ('Cost(L)'). Cost(L) for DEFAULT_METRIC_TYPE is specified in Section 7.3. The Cost() function for other metrics is beyond the scope of this document.

7.2. The LoopFree() function

Since determining loop freedom is known to depend on comparing the Cost(R1) of advertised route update information to the Cost(R2) of an existing stored route using the same metric type, AODVv2 invokes a function called "LoopFree(R1, R2)". LoopFree(R1, R2) returns TRUE when R1 is guaranteed to not rely on the route R2, i.e. R2 is not a subroute of the route R1. An AODVv2 router invokes LoopFree() to compare an advertised route to a stored route. The advertised route is referred to as AdvRte and is used as parameter R1. The stored route is referred to as Route and is used as parameter R2.

7.3. Default Metric type

The default MetricType (DEFAULT_METRIC_TYPE) is HopCount (but see Section 7.4). HopCount is the only metric described in detail in this document. For the HopCount metric, Cost(L) is always 1, and Cost(R) is the hop count between the router and the destination.

MAX_METRIC[DEFAULT_METRIC_TYPE] is defined to be MAX_HOPCOUNT. MAX_HOPCOUNT MUST be larger than the AODVv2 network diameter. Otherwise, AODVv2 protocol messages may not reach their intended destinations.

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

7.4. Alternate Metrics

Some applications may require metric information other than HopCount, which has traditionally been the default metric associated with routes in MANET. It is well known that reliance on HopCount can cause selection of the worst possible route in some situations. For this reason, AODVv2 enables route selection based on metric information other than HopCount -- in other words, based on "alternate metrics".
The range and data type of each such alternate metric may be different. For instance, the data type might be integers, or floating point numbers, or restricted subsets thereof. It is out of the scope of this document to specify for alternate metrics the Cost(L) and Cost(R) functions, or their return type. Where necessary these should take into account any differences in the link cost in each direction.

8. AODVv2 Protocol Operations

In this section, operations are specified for updating the route table using information within AODVv2 RteMsgs (either RREQ or RREP), and due to timeouts. AdvRte is the route advertised by the RteMsg. RteMsgs include IP addresses as well as possibly the SeqNum and the prefix lengths associated with those IP addresses. The AdvRte also includes the metric measured from the neighbor transmitting the RteMsg to the IP address originating the route update. All SeqNum comparisons use signed 16-bit arithmetic.

8.1. Evaluating Incoming Routing Information

After determining that the incoming information is correctly formatted and contains values in the correct ranges, the AODVv2 router will use the information to update local routing information if possible. This section explains how to determine whether the incoming information should be used to update the route table, and how to perform the update.

The incoming RteMsg may be a RREQ or a RREP. If it is a RREQ, it contains information about a route to OrigAddr. Prefix length information in a RREQ, if present, describes the subnet on which OrigAddr resides. If it is a RREP, it contains information about a route to TargAddr. AdvRte is used to denote the route information contained in the RteMsg. AdvRte has the following properties:

- AdvRte.Address = OrigAddr (in RREQ) or TargAddr (in RREP).
- AdvRte.SeqNum = OrigSeqNum (in RREQ) or TargSeqNum (in RREP).
- AdvRte.MetricType = RteMsg.MetricType, if present, else DEFAULT_Metric_TYPE.
- AdvRte.Cost = AdvRte.Metric + Cost(L) according to the indicated MetricType, where L is the link from the advertising router.
In the description below, Route denotes the stored routing table entry and HandlingRtr is the router receiving the RteMsg. HandlingRtr MUST process the incoming information as follows. If the routing table does not contain an entry matching AdvRte’s Address and MetricType, create a new route table entry according to the procedure in Section 8.2. Otherwise determine whether or not to use AdvRte for updating the route entry (Route) matching the AdvRte’s Address and MetricType as follows:

   * If AdvRte’s sequence number is newer, HandlingRtr MUST use AdvRte to update the Route.
   * If stale, using the incoming information might result in a routing loops. In this case the HandlingRtr MUST NOT use AdvRte to update the Route.
   * If the SeqNums are equal, continue checking as below.

   * If the advertised route’s cost is the same or greater than the stored route, and the stored route is valid, the incoming information does not offer any improvement and SHOULD NOT be used to update the stored route table entry.
   * If the advertised route’s cost is lower than the stored route, AdvRte offers improvement and SHOULD be used to update the stored route table entry.
   * If the advertised route’s cost is the same or greater than the stored route, but the stored route’s state is Invalid, continue processing to see whether there is a danger of a routing loop.

3. Check whether the information is safe against loops (LoopFree (AdvRte, Route) == TRUE).
   * If LoopFree (see Section 7.2) returns false, using the incoming information might cause a routing loop. AdvRte MUST NOT be used to update the stored route table entry.

4. If the advertised route can be used to update the route table entry, follow the procedure in Section 8.2.
To briefly summarize, AdvRte must satisfy the following conditions compared to the existing route table entry before it can be used:

- AdvRte is more recent, (i.e., AdvRte.SeqNum > Route.SeqNum) OR
- AdvRte is not stale and can safely restore an invalid route (i.e. LoopFree (AdvRte, Route) == TRUE), OR
- AdvRte is not stale and is less costly.

Also see the pseudocode in Appendix A.1.1.

If the route has been updated based on information in a received RREQ, the AODVv2 router MAY force regeneration of the RREQ, to ensure the most recent information is propagated to other routers, but it MAY suppress this to avoid extra control traffic.

8.2. Applying Route Updates To Route Table Entries

To apply the route update, a route table entry for AdvRte.Address is either found to already exist in the route table, or else a new route table entry for AdvRte.Address is created and inserted into the route table. If the route table entry had to be created, or if the state is Invalid, the state is set to be Idle. The fields of route table entry are assigned as follows:

- Route.SeqNum := AdvRte.SeqNum
- Route.NextHop := IP.SourceAddress (i.e., the address from which the RteMsg was received)
- Route.NextHopInterface is set to the interface on which RteMsg was received
- Route.MetricType := AdvRte.MetricType
- Route.LastUsed := Current_Time
- Route.LastSeqnum := Current_Time
- If RteMsg.ValidityTime is included, then
Route.ExpirationTime := Current_Time + RteMsg.ValidityTime and  
Route.Timed := TRUE. Otherwise, Route.Timed := FALSE and  
Route.ExpirationTime := MAXTIME.

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

8.3. Route Maintenance

AODVv2 routers attempt to maintain active routes. Before using a  
route to forward a packet, an AODVv2 router MUST check the status of  
the route as specified in Section 8.4. If the route has been marked  
as Invalid, it cannot be used for forwarding. Otherwise, set  
Route.LastUsed := Current_Time, Route.State := Active, and forward  
the packet to the route’s next hop.

When a routing problem is encountered, an AODVv2 router (denoted  
RERR_Gen) sends the RERR to quickly notify upstream routers. Two  
kinds of routing problems can trigger generation of a RERR message.  
The first happens when the router receives a packet but does not have  
a valid route for the destination of the packet. The second case  
happens immediately upon detection of a broken link (see Section 6.2)  
for an valid route.

Optionally, if a precursor list is maintained for the route, see  
Section 12.2 for precursor lifetime operations.

8.4. Route Table Entry Timeouts

During normal operation, AODVv2 does not require any explicit  
timeouts to manage the lifetime of a route. At any time, any route  
table entry can be examined and then either expunged or marked as  
Invalid according to the following rules.

The following rules are used to manage the state of route table  
entries:

- If Current_Time > Route.ExpirationTime, set Route.State :=  
  Invalid.

- If (Current_Time - Route.LastUsed) > (ACTIVE_INTERVAL +  
  MAX_IDLETIME), and if (Route.Timed == FALSE), set Route.State :=  
  Invalid.
If \((\text{Current\_Time} - \text{Route}\_\text{LastUsed}) > \text{ACTIVE\_INTERVAL}\), and if
\((\text{Route}\_\text{Timed} == \text{FALSE})\), set \(\text{Route}\_\text{State} := \text{Idle}\).

If \((\text{Current\_Time} - \text{Route}\_\text{LastSeqNum} > \text{MAX\_SEQNUM\_LIFETIME})\), and
the route is Invalid, the route table entry MUST be expunged. If
the route is not invalid and \(\text{MAX\_SEQNUM\_LIFETIME}\) has expired, the
SeqNum information should be removed from the route, to avoid
problems with boot sequence and lost SeqNum behaviour.

Memory constrained devices MAY choose to expunge routes from the
AODVv2 route table at other times, but MUST adhere to the following
rules:

- An Active route MUST NOT be expunged.
- An Idle route SHOULD NOT be expunged.
- Any Invalid route MAY be expunged; least recently used Invalid
  routes SHOULD be expunged first.

If precursor lists are maintained for the route (as described in
Section 12.2) then the precursor lists must also be expunged at the
same time that the route itself is expunged.

8.5. Route Discovery, Retries and Buffering

AODVv2 message types RREQ and RREP are together known as Routing
Messages (RteMsgs) and are used to discover a route between an
Originating and Target Address, denoted by OrigAddr and TargAddr.
The constructed route is bidirectional, enabling packets to flow
between OrigAddr and TargAddr. RREQ and RREP have similar
information and function, but have some differences in their rules
for handling. When a node receives a RREQ or a RREP, the node then
creates or updates a route to the OrigAddr or the TargAddr
respectively (see Section 8.1). The main difference between the two
messages is that, by default, RREQ messages are multicast to solicit
a RREP, whereas RREP is unicast as a response to RREQ.

When an AODVv2 router needs to forward a data packet from a node
(with IP address OrigAddr) in its set of router clients, and it does
not have a forwarding route toward the packet’s IP destination
address (TargAddr), the AODVv2 router (RREQ_Gen) generates a RREQ (as
described in Section 9.1.1) to discover a route toward TargAddr.
Subsequently RREQ_Gen awaits reception of an RREP message (see
Section 9.2.1) or other route table update (see Section 8.2) to
establish a route toward TargAddr. The RREQ message contains routing
information to enable RREQ recipients to route packets one hop
between the OrigAddr, and the RREP message contains routing
information to enable RREP recipients to route packets one hop towards the TargAddr.

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

To reduce congestion in a network, repeated attempts at route discovery for a particular Target Address SHOULD utilize a binary exponential backoff, as described in [RFC3561], where the initial wait time is RREQ_WAIT_TIME and the wait time is doubled for each retry based.

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

If a route discovery attempt has failed (i.e., DISCOVERY_ATTEMPTS_MAX attempts have been made without receiving a RREP) to find a route toward the Target Address, any data packets buffered for the corresponding Target Address MUST BE dropped and a Destination Unreachable ICMP message (Type 3) SHOULD be delivered to the source of the data packet. The code for the ICMP message is 1 (Host unreachable error). If RREQ_Gen is not the source (OrigNode), then the ICMP is sent to OrigAddr.

8.6. Suppressing Redundant RteMsgs

When RREQ messages are flooded in a MANET, an AODVv2 router may receive similar RREQ messages from more than one of its neighbours. To avoid processing and transmission associated with redundant RteMsgs, while still enabling proper handling of earlier RteMsgs that may have somehow been delayed in the network, it is necessary for
each AODVv2 router store information about RteMsgs which it has recently received (see the RteMsg table defined in Section 6.5).

When a RREQ is received, it is checked against the RteMsg Table to see if it contains redundant information. If so it does not need to be processed.

For RREQ messages, the process for comparison is as follows:

- Look for a "comparable" entry in the RteMsg Table with the same MsgType, OrigAddr, TargAddr, and MetricType.
- If there is none, create an entry to store information about the received RREQ, and continue to regenerate the RREQ.
- If there is an entry, and it has a lower SeqNum for OrigAddr than the received RREQ, update it using the new RREQ and continue to regenerate the RREQ.
- If there is an entry and it has a higher SeqNum for OrigAddr than the received RREQ, do not replace the entry and do not process the RREQ.
- If there is an entry and it has the same SeqNum for OrigAddr and a higher Metric than the received RREQ, update it with the new RREQ information.
- If there is an entry and it has the same SeqNum for OrigAddr and a Metric less than or equal to the received RREQ, do not replace the entry and do not regenerate the RREQ.
- In all cases, update the timestamp field, since other comparable RREQs may still be traversing the network.

The process of comparison for optional multicast RREP messages is analogous, substituting RREP for RREQ, and TargAddr for OrigAddr. Entries in the RteMsg Table MUST be deleted after MAX_SEQNUM_LIFETIME, but should be maintained for at least RteMsg_ENTRY_TIME in order to account for long-lived RREQs traversing the network.

9. AODVv2 Protocol Messages

This section specifies the data elements and values required in AODVv2 protocol messages, namely RREQ, RREP, RERR, and RREP_Ack.

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

See Section 10 for the mapping of AODVv2 data elements to RFC 5444 Message TLVs, Address Blocks, and Address TLVs.

9.1. RREQ Messages

RREQ messages are used in Route Discovery operations to request a route to a specified Target address. RREQ messages have the following general structure:

```
+-----------------------------------------------------------------+
|                   msg_hop_limit, msg_hop_count                  |
+-----------------------------------------------------------------+
|                 AddressList := {OrigAddr, TargAddr}             |
+-----------------------------------------------------------------+
| PrefixLengthList := {PrefixLength for OrigAddr, null}(optional) |
+-----------------------------------------------------------------+
|                 OrigSeqNum, (optional) TargSeqNum               |
+-----------------------------------------------------------------+
|                      MetricType (optional)                      |
+-----------------------------------------------------------------+
|             MetricList := {Metric for OrigAddr, null}           |
+-----------------------------------------------------------------+
| ValidityTimeList := {ValidityTime for OrigAddr, null}(optional) |
+-----------------------------------------------------------------+
```

Figure 1: RREQ message structure

RREQ Data Elements

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

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

`AddressList`
- AddressList contains OrigAddr and TargAddr.

`PrefixLengthList`
- PrefixLengthList contains the length of the prefix for OrigAddr, if OrigAddr resides on a Client Network with a prefix.
length shorter than the number of bits of the address family for OrigAddr.

OrigSeqNum
OrigSeqNum is REQUIRED and carries the destination sequence number associated with OrigNode.

TargSeqNum
TargSeqNum is optional and carries a destination sequence number associated with TargNode.

MetricList
The MetricList data element is REQUIRED, and carries the route metric information associated with OrigAddr.

MetricType
The MetricType element defines the type of Metric associated with the entries in the MetricList.

ValidityTimeList
The ValidityTimeList is optional and carries the length of time that the sender is willing to offer a route towards OrigAddr.

RREQ messages carry information about OrigAddr and TargAddr, as identified in the context of the RREQ_Gen. The OrigSeqNum MUST appear. Both MAY appear in the same RREQ when SeqNum is available for both OrigAddr and TargAddr.

The OrigSeqNum data element in a RteMsg MUST apply only to OrigAddr. The other address in the AddressList is TargAddr.

If the TargSeqNum data element appears, then it MUST apply only to TargAddr. The other address in the AddressList is OrigAddr.

9.1.1. RREQ Generation

Upon receiving an IP packet from one of its Router Clients, it often happens that an AODVv2 router has no valid route to the destination. In this case the AODVv2 router is responsible for generating a RREQ and associated data elements on behalf of its client OrigNode. The router is referred to as RREQ_Gen. Before creating a RREQ, RREQ_Gen should check if an RREQ has recently been sent for this destination and a response is awaited, or if the limit of AODVv2 RREQ retries has been reached.

In constructing the RREQ, RREQ_Gen uses AddressList, OrigSeqNum, MetricList, and optionally PrefixLengthList, TargSeqNum, MetricType, and ValidityTime.
RREQ_Gen follows the steps in this section. OrigAddr MUST be a unicast address. The order of data elements is illustrated schematically in Figure 1. RREQ_Gen SHOULD include TargSeqNum, if a previous value of the TargAddr’s SeqNum is known (e.g. from an invalid route table entry using longest-prefix matching). If TargSeqNum is not included, AODVv2 routers handling the RREQ assume that RREQ_Gen does not have that information.

1. Set msg_hop_limit to MAX_HOPCOUNT.

2. Set msg_hop_count to zero, if including it.

3. Set AddressList := {OrigAddr, TargAddr}.

4. For the PrefixLengthList:
   * If OrigAddr resides on a subnet of Router Clients, set PrefixLengthList := { OrigAddr subnet’s prefix, null }.
   * Otherwise, the PrefixLengthList is omitted.

5. For the Sequence Number List:
   * Increment the SeqNum as specified in Section 6.4.
   * Set OrigSeqNum to the new value of SeqNum.
   * If an Invalid route exists matching TargAddr using longest prefix matching, include TargSeqNum and set it to the sequence number on the Invalid route. Otherwise omit TargSeqNum.


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

8. If the RREQ_Gen wishes to limit the time that the route to OrigAddr may be used, include the ValidityTime data element.

9.1.2. RREQ Reception

Upon receiving an RREQ, an AODVv2 router performs the following steps.

1. A router MUST handle RREQs only from neighbors. RREQs from nodes that are not neighbors MUST be disregarded.
2. Check whether the sender is on the blacklist of AODVv2 routers (see Section 6.2). If not, continue processing. Otherwise, check the Blacklist Remove Time.
   * If Current_Time < Remove Time, ignore this RREQ for further processing.
   * If Current_Time >= Remove Time, remove the Blacklist entry and continue processing.

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

4. If the MetricType data element is present, check that the MetricType is known.
   * If not, ignore this RREQ for further processing.
   * Otherwise continue processing.

5. Verify that OrigAddrMetric <= (MAX_METRIC[MetricType] - Cost(Link)).
   * If not, ignore this RREQ for further processing.
   * Otherwise continue processing.

6. Process the route to OrigAddr as specified in Section 8.1.

7. Check if the message is a duplicate or redundant by comparing to entries in the RteMsg table as described in Section 8.6.
   * If duplicate or redundant, ignore this RREQ for further processing.
   * Otherwise save the information in the RteMsg table to identify future duplicates and continue processing.

8. Check if the TargAddr belongs to one of the Router Clients.
   * If so, generate a RREP as specified in Section 9.2.1.
   * Otherwise, continue to RREQ regeneration.
9.1.3. RREQ Regeneration

Unless the router is prepared to advertise the new route, it halts processing. By sending a RREQ, a router advertises that it will forward packets to the OrigAddr contained in the RREQ according to the information enclosed. The router MAY choose not to regenerate the RREQ, though this could decrease connectivity in the network or result in non-optimal paths.

The circumstances under which a router MAY choose not to regenerate a RREQ are not specified in this document. Some examples may include the router being heavily loaded and not advertising routing for more traffic, or being low on energy and having to reduce energy expended for sending AODVv2 messages or packet forwarding.

The procedure for RREQ regeneration is as follows:

1. Check the msg_hop_limit.
   * If it is zero, do not regenerate.
   * Otherwise, decrement the value by one.

2. Check if msg_hop_count is present and greater than or equal to MAX_HOPCOUNT
   * If so, do not regenerate.
   * Otherwise, increment msg_hop_count by one.

3. Change OrigAddrMetric to match the route table entry for OrigAddr, which should match the advertised value in the received RREQ plus the cost of the link to the router which forwarded the RREQ.

4. If the incoming RREQ contains a ValidityTimeList, it MUST be copied into the regenerated RREQ. If not present, and the regenerating router wishes to limit the time that its route to OrigAddr may be used, set ValidityTimeList := {ValidityTime for OrigAddr, null}.

If the received RREQ was unicast, the regenerated RREQ can be unicast to the next hop address of the route towards TargAddr, if known. Otherwise, the RREQ SHOULD be multicast to the LL-MANET-Routers IP and MAC address [RFC5498], [RFC4291].
9.2. RREP Messages

RREP messages are used to offer a route to a target address, and are sent in response to a RREQ message. RREP messages have the following general structure:

```
+-----------------------------------------------------+
| msg_hop_limit, msg_hop_count                          |
| AckReq (optional)                                    |
| AddressList := {OrigAddr, TargAddr}                  |
| PrefixLengthList := {null, PrefixLength for TargAddr}|
| TargSeqNum                                           |
| MetricList := {null, metric for TargAddr}            |
| MetricType (optional)                                |
| ValidityTimeList := {null, ValidityTime for TargAddr}|
+-----------------------------------------------------+
```

Figure 2: RREP message structure

RREP Data Elements

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

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

AckReq
Acknowledgement Requested by sender (optional).

AddressList
AddressList contains OrigAddr and TargAddr.

PrefixLengthList
PrefixLengthList contains the length of the prefix for TargAddr, if TargAddr resides on a Client Network with a prefix length shorter than the number of bits of the address family for TargAddr.
TargSeqNum
TargSeqNum is REQUIRED and carries the destination sequence number associated with TargNode.

MetricList
The MetricList data element is REQUIRED, and carries the route metric information associated with TargAddr.

MetricType
The MetricType element defines the type of Metric associated with the entries in the MetricList.

ValidityTimeList
The ValidityTimeList is optional and carries the length of time that the sender is willing to offer a route towards TargAddr.

RREP messages carry information about OrigAddr and TargAddr, as known in the context of the RREP_Gen. The TargSeqNum MUST appear. It MUST apply only to TargAddr. The other address in the AddressList is OrigAddr.

9.2.1. RREP Generation

This section specifies the generation of an RREP by an AODVv2 router (RREP_Gen) that provides connectivity for TargAddr, thus enabling the establishment of a route between OrigAddr and TargAddr. In constructing the RREP, AODVv2 uses AddressList, TargSeqNumber List, MetricList, and optionally AckReq, PrefixLengthList and/or ValidityTimeList. These elements are then used to create a RFC5444 message; see Section 10 for details.

The AckReq data element indicates that an acknowledgement to the RREP has been requested. If no corresponding RREP_Ack is received within the RREP_Ack_SENT_TIMEOUT, the next hop is added to the blacklist as discussed in Section 6.2.

The procedure for RREP generation is as follows:

1. Set msg_hop_limit to the msg_hop_count from the received RREQ message.
2. Set msg_hop_count, if including it, to zero.
3. Include the AckReq data element if RREP_Ack is requested from the next hop (as described in Section 6.2).
4. Include the MetricType data element and set the type accordingly.
5. Set the Address List := {OrigAddr, TargAddr}.

6. For the PrefixLengthList:
   * If TargAddr resides on a subnet of Router Clients, set PrefixLengthList := {null, TargAddr subnet’s prefix}.
   * Otherwise, no PrefixLengthList is needed.

7. For the TargSeqNum:
   * RREP_Gen increments its SeqNum as specified in Section 6.4.
   * Set TargSeqNum := the new value of SeqNum.

8. Set MetricList := { null, Route[TargAddr].Metric }.

9. If the RREP_Gen wishes to limit the time that the route to TargAddr may be used, set ValidityTimeList := {null, TargAddr ValidityTime}.

By default, the RREP is sent by unicast to the IP address of the next hop of the RREP_Gen’s route to OrigAddr.

9.2.2. RREP Reception

Upon receiving an RREP, an AODVv2 router performs the following steps.

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

2. Check that the MetricType is known.
   * If not, ignore this RREP for further processing.
   * Otherwise continue processing.

3. Verify that TargAddrMetric <= (MAX_METRIC[MetricType] - Cost(Link)).
   * If not, ignore this RREP for further processing.
   * Otherwise continue processing.
4. Process the route to TargAddr as specified in Section 8.1.

5. If the AckReq data element is present, send a RREP_Ack as specified in Section 9.4.

6. Check if the message is a duplicate or redundant by comparing to entries in the RREP table as described in Section 8.6.
   * If duplicate or redundant, ignore this RREP for further processing.
   * Otherwise save the information in the RREP table to identify future duplicates and continue processing.

7. Check if the OrigAddr belongs to one of the Router Clients.
   * If so, the RREP satisfies a previously sent RREQ. Processing is complete and data can now be forwarded along the route. Any packets from OrigAddr that were buffered for later delivery SHOULD be transmitted.
   * Otherwise, continue to RREP regeneration.

9.2.3. RREP Regeneration

Similar to rules for RREQ regeneration, unless the router is prepared to advertise the route to TargAddr, it halts processing. By forwarding a RREP, a router advertises that it will forward packets to the TargAddr contained in the RREP according to the information enclosed. The router MAY choose not to regenerate the RREP, for the same reasons as mentioned under RREQ regeneration Section 9.1.3, though this could decrease connectivity in the network or result in non-optimal paths.

If no valid route exists to OrigAddr, a RERR SHOULD be transmitted to TargAddr as specified in Section 9.3.1 and the RREP should not be regenerated.

The procedure for RREP regeneration is as follows:

1. Check the msg_hop_limit.
   * If it is zero, do not regenerate.
   * Otherwise, decrement the value by one.

2. If msg_hop_count is present, then:
* If msg_hop_count >= MAX_HOPCOUNT, do not regenerate.
* Otherwise, increment msg_hop_count by one.

3. The RREP SHOULD be unicast to the next hop on the route to OrigAddr. If no valid route exists to OrigAddr, a RERR SHOULD be transmitted to TargAddr as specified in Section 9.3.1.

4. Change TargAddrMetric to match the route table entry for TargAddr, which should match the advertised value in the received RREP plus the cost of the link to the router which forwarded the RREP.

5. Include the AckReq data element if this device requires acknowledgement of the RREP message.

6. If the incoming RREP contains a ValidityTimeList, it MUST be copied into the regenerated RREP. If not present, and the regenerating router wishes to limit the time that its route to TargAddr may be used, set ValidityTimeList := {null, ValidityTime for TargAddr}.

   The RREP SHOULD be unicast to the next hop on the route to OrigAddr.

9.3. RERR Messages

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

```
+---------------------------------------------+     |
| msg_hop_limit                               |
+---------------------------------------------+     |
| PktSource (optional)                        |
+---------------------------------------------+     |
| RERR AddressList                            |
+---------------------------------------------+     |
| PrefixLengthList for UnreachableAddresses (optional) |
+---------------------------------------------+     |
| SeqNumList (one entry per address)          |
+---------------------------------------------+     |
| MetricType (optional)                       |
+---------------------------------------------+     |
```

Figure 3: RERR message structure

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

PktSource
The IP address of the unreachable destination triggering RERR generation. If this RERR message was triggered by a broken link, the PktSource data element is not required.

RERR AddressList
A list of IP addresses not reachable by the AODVv2 router transmitting the RERR.

PrefixLengthList
PrefixLengthList contains the prefix lengths associated with the addresses in the RERR AddressList, if any of them reside on a Client Network with a prefix length shorter than the number of bits of their address family.

MetricType
If MetricType != DEFAULT_METRIC_TYPE, the MetricType associated with routes affected by a broken link.

SeqNumList
The list of sequence numbers associated with the UnreachableAddresses in the RERR AddressList.

9.3.1. RERR Generation

There are two types of events which trigger generation of a RERR message. The first is the arrival of a packet for which there is no route to the destination address. This can be a packet forwarded by the routing process, or a RREP when there is no route to OrigAddr. In this case, exactly one UnreachableAddress will be included in RERR’s AddressList (either the Destination Address of the IP header from a data packet, or the OrigAddr found in the AddressList of an RREP message). RERR_Gen MUST discard the packet or message that triggered generation of the RERR.

The second type of event happens when a link breaks. All routes (whether valid or not) that use the broken link MUST be marked as Invalid. If the broken link was not used by any Active route, no RERR message is generated. Every Invalid route reported in the RERR MUST have the same MetricType. If the broken link affects routes to destinations that have different MetricTypes, multiple RERR messages must be generated.
If an AODVv2 router receives an ICMP packet to or from the address of one of its client nodes, it simply forwards the ICMP packet, and does not generate any RERR message.

In constructing the RERR, AODVv2 uses MetricType, AddressList, SeqNumList, and in some cases PktSource and PrefixLengthList. These elements are then used to create a RFC5444 message; see Section 10 for details.

The procedure for RERR generation is as follows:

1. Set msg_hop_limit to MAX_HOPCOUNT.

2. If the RERR was triggered by an Undeliverable Packet, the PktSource data element MUST be included, containing the source IP address of the Undeliverable Packet.

3. Include the MetricType data element if reporting a Invalid route for a non-default metric type.

4. For the RERR AddressList:
   * If the RERR was triggered by an undeliverable packet, insert the destination IP address of the undeliverable packet, or if the packet was a RREP, insert the OrigAddr.
   * If the RERR was triggered by a broken link, include the addresses of all previously Active routes which are now Invalid, up to the limit imposed by the MTU (interface "Maximum Transfer Unit") of the physical medium. If there are too many such previously Active routes, additional RERR messages should be constructed and transmitted to contain the remaining addresses. If the configuration option ENABLE_IDLE_IN_RERR is enabled, include any previously Idle routes which are now Invalid, as long as the packet size of the RERR does not exceed the MTU.

5. If there are destinations reported in the RERR AddressList that have associated subnet prefixes in the route table, insert those prefixes in the PrefixLengthList; otherwise, omit the PrefixLengthList.

6. If known, the sequence numbers associated with the routes to the addresses in the RERR AddressList SHOULD be included in the SeqNumList; otherwise, omit the SeqNumList.

If the RERR is sent in response to an Undeliverable Packet:
o It SHOULD be sent unicast to the next hop towards the source IP address of the packet which triggered the RERR.

o Otherwise the RERR MUST be sent to the multicast IP and MAC address for LL-MANET-Routers.

If the RERR is sent in response to a broken link:

o If precursor lists are maintained for the addresses in the RERR AddressList (see Section 12.2), the RERR SHOULD be unicast to the precursors.

o Otherwise the RERR MUST be sent to the multicast IP and MAC address for LL-MANET-Routers.

9.3.2. RERR Reception

Upon receiving an RERR, the following steps are performed.

1. If the message does not contain the msg_hop_limit and at least one UnreachableAddress, do not process the RERR.

2. If the MetricType data element is present, check that the MetricType is known.
   * If not, ignore this RERR for further processing.
   * Otherwise continue processing.

3. For each UnreachableAddress,
   * Check that the address is valid (routable and unicast).
   * Check that there is a valid route with the same MetricType matching the address using longest prefix matching.
   * Check that the route’s next hop is the sender of the RERR.
   * Check that the route’s next hop interface is the interface on which the RERR was received.
   * Check that the Unreachable Address SeqNum is either unknown, or is greater than the route’s SeqNum.
   * If any of the above are false, the UnreachableAddress 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 UnreachableAddress’s prefix length, set the route state to Invalid.

+ 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 UnreachableAddress and its prefix, and set the state to Invalid.

If there are no UnreachableAddresses which need to be advertised in a regenerated RERR, take no further action.

Otherwise regenerate the RERR as specified in Section 9.3.3.

9.3.3. RERR Regeneration

The procedure for RERR regeneration is as follows:

1. Check the msg_hop_limit.
   * If it is zero, do not regenerate.
   * Otherwise, decrement the value by one.

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

3. For the RERR AddressList, include all UnreachableAddresses which have been determined to need regeneration.

4. For the PrefixLengthList, insert the prefix lengths associated with the addresses in the RERR AddressList.

5. For the SeqNumList, include the sequence numbers corresponding to the addresses in the RERR AddressList.

If the original RERR contained the PktSource data element, and a route exists to the source address, the regenerated RERR MUST be sent unicast to the next hop of the route towards PktSource.

Otherwise, if precursor lists are maintained, the regenerated RERR SHOULD be sent to the active precursors of the Invalid routes as specified in Section 12.2.
Otherwise the regenerated RERR MUST be sent to the multicast IP and MAC address for LL-MANET-Routers.

9.4. RREP_Ack Messages

RREP_Ack is modeled on the RREP_Ack message type from AODV [RFC3561]. RREP_Ack messages have the following general structure:

```
+-----------------------------------------------------------------+
|                       msg_hop_limit := 1                        |
+-----------------------------------------------------------------+
```

Figure 4: RREP_Ack message structure

RREP_Ack Data Elements

- `msg_hop_limit`: The remaining number of hops allowed for dissemination of the RREP_Ack message.

9.4.1. RREP_Ack Generation

This section specifies the generation of an RREP_Ack by an AODVv2 router. The procedure is as follows:

1. Set `msg_hop_limit := 1`.

   The RREP_Ack is sent by unicast to the IP address of the router that inserted a AckReq data element into a RREP message.

9.4.2. RREP_Ack Reception

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

1. The router checks whether the sender’s IP address is in the blacklist. If so, the IP address is deleted from the blacklist.

2. The router checks whether an RREP_Ack message was expected from the sending IP address, in response to an AckReq data element that the router included in a preceding RREP message as specified in Section 9.2.1. If so, the router records that the required RREP_Ack has been received and cancels the associated timeout.
10. Representing AODVv2 data elements using RFC 5444

AODVv2 specifies that all control plane messages between Routers SHOULD use the Generalised Mobile Ad-hoc Network Packet and Message Format [RFC5444], which provides a multiplexed transport for multiple protocols. AODVv2 therefore specifies Route Messages comprising data elements that map to message elements in RFC5444 but, in line with the concept of use, does not specify which order the messages should be arranged in an RFC5444 packet. An implementation of an RFC5444 multiplexer may choose to optimise the content of certain message elements to reduce control plane overhead. For handling of messages that contain unknown TLV types, the multiplexer SHOULD ignore the information for processing, but preserve it unmodified for forwarding.

Here is a brief summary of the RFC 5444 format.

1. A packet formatted according to RFC 5444 contains zero or more messages.

2. A message contains a message header, message TLV block, and zero or more address blocks.

3. Each address block MAY also have one TLV blocks; each TLV block MAY encode any number of TLVs (including zero). Each TLV value in an Address TLV block is associated with exactly one of the addresses in the address block.

The following table shows how AODVv2 data elements are represented in RFC 5444 messages.
AODVv2 neither requires any inclusion nor uses any information from the packet header. The length of an address (32 bits for IPv4 and 128 bits for IPv6) inside an AODVv2 message is indicated by the msg-addr-length (MAL) in the msg-header. Although the addresses in an Address Block may appear in any order, each TLV value in a TLV Block is associated with exactly one Address in the Address Block. So, for instance, the ordering of the OrigAddrMetric and TargAddrMetric values in the MetricList is determined by the order of OrigAddr and TargAddr in the preceding RteMsg Address List. See Section 14.2 for more information about AODVv2 Message TLVs. See Section 14.3 for more information about AODVv2 Address Block TLVs.

### 11. Simple Internet Attachment

Simple Internet attachment means attachment of a stub (i.e., non-transit) network of AODVv2 routers to the Internet via a single Internet AODVv2 router (called IAR).
As in any Internet-attached network, AODVv2 routers, and their clients, wishing to be reachable from hosts on the Internet MUST have IP addresses within the IAR’s routable and topologically correct prefix (e.g. 191.0.2.0/24).

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

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

12. Optional Features

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

- Expanding Rings Multicast
- Precursor lists.
- Multicast RREP Response to RREQ
- Intermediate RREPs (iRREPs): Without iRREP, only the destination can respond to a RREQ.
- Message Aggregation Delay.

12.1. 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.

12.2. Precursor Lists and Notifications

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

12.2.1. Overview

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

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

When an AODVv2 router marks a route as Invalid, the precursors of the Invalid route should be notified (using RERR) about the change in status of their route to the destination of that Invalid route.

12.2.2. Precursor Notification Details

During normal operation, each AODVv2 router wishing to maintain precursor lists as described above, maintains a precursor table and updates the table whenever the node forwards traffic to one of the destinations in its route table. For each precursor in the precursor list, a record must be maintained to indicate whether the precursor
has been used for recent traffic (in other words, whether the precursor is an Active precursor). So, when traffic arrives from a precursor, the Current_Time is used to mark the time of last use for the precursor list element associated with that precursor.

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

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

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

Each neighbor receiving the RERR MAY then execute the same procedure until all upstream routers have received the RERR notification.

### 12.3. Multicast RREP Response to RREQ

The RREQ Target Router (RREP_Gen) MAY, as an alternative to uncasting a RREP, be configured to use multicast to distribute routing information about the route toward TargAddr. RREP_Gen does this as described in Section 9.2.1, but multicasting the RREP to LL-MANET-Routers [RFC5498]. Routers receiving the multicast RREP must perform RteMsg suppression (see Section 8.6).

Broadcast RREP response to incoming RREQ was originally specified to handle unidirectional links, but it is expensive. Due to the significant overhead, AODVv2 routers MUST NOT use multicast RREP unless configured to do so by setting the administrative parameter USE_MULTICAST_RREP. This technique can be used to find the best return path rather than follow the same path as the RREQ took.

### 12.4. Intermediate RREP

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

### 12.5. Message Aggregation Delay

The aggregation of multiple messages into a packet is specified in RFC 5444 [RFC5444].
Implementations MAY choose to briefly delay transmission of messages for the purpose of aggregation (into a single packet) or to improve performance by using jitter [RFC5148].

13. Administratively Configurable Parameters and Timer Values

AODVv2 uses various configurable parameters of various types:

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

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

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

13.1. Timers

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

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

Table 4: Timing Parameter Values

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

13.2. Protocol Constants

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISCOVERY_ATTEMPTS_MAX</td>
<td>3</td>
<td>Section 8.5</td>
</tr>
<tr>
<td>MAX_HOPCOUNT</td>
<td>20 hops</td>
<td>Section 7</td>
</tr>
<tr>
<td>MAX_METRIC[i]</td>
<td>Specified only for HopCount [TBD]</td>
<td>Section 7</td>
</tr>
<tr>
<td>MAXTIME</td>
<td>[TBD]</td>
<td>Maximum expressible clock time Section 8.4</td>
</tr>
</tbody>
</table>

Table 5: Parameter Values

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

- DISCOVERY_ATTEMPTS_MAX: some nodes are likely to be more successful at finding routes, but at the cost of additional control traffic for unsuccessful attempts.

- MAX_HOPCOUNT: If some nodes use a value that is too small, they would not be able to discover routes to distant addresses.

- MAX_METRIC[DEFAULT_METRIC_TYPE]: MUST always be the maximum expressible metric of type DEFAULT_METRIC_TYPE. No interoperability problems due to variations on different nodes, but if a lesser value is used, route comparisons may exhibit overly restrictive behavior.

- MAXTIME: Variations on different nodes would not cause problems for interoperability. If a lesser value is used, route state management may exhibit overly restrictive behavior.
13.3. Administrative (functional) controls

The following administrative controls may be used to change the operation of the network, by enabling optional behaviors. These options are not required for correct routing behavior, although they may potentially reduce AODVv2 protocol messaging in certain situations. The default behavior is typically to NOT enable the options. Inconsistent settings at different nodes in the network will not result in protocol errors. In the case of inconsistent settings for DEFAULT_METRIC_TYPE, inconsistent setting might result in messages specifying metric types unknown to some nodes and consequent poor performance.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT_METRIC_TYPE</td>
<td>3 (i.e, Hop Count (see [RFC6551]))</td>
</tr>
<tr>
<td>ENABLE_IDLE_IN_RERR</td>
<td>Section 9.3.1</td>
</tr>
<tr>
<td>ENABLE_IRREP</td>
<td>Section 9.1.1</td>
</tr>
<tr>
<td>USE_MULTICAST_RREP</td>
<td>Section 12.3</td>
</tr>
</tbody>
</table>

Table 6: Administratively Configured Controls

13.4. Other administrative parameters and lists

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Default Value</th>
<th>Cross Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>AODVv2_INTERFACES</td>
<td></td>
<td>Section 4</td>
</tr>
<tr>
<td>BUFFER_SIZE_PACKETS</td>
<td>2</td>
<td>Section 8.5</td>
</tr>
<tr>
<td>BUFFER_SIZE_BYTES</td>
<td>MAX_PACKET_SIZE [TBD]</td>
<td>Section 8.5</td>
</tr>
<tr>
<td>CLIENT_ADDRESSES</td>
<td>AODVv2_INTERFACES</td>
<td>Section 6.3</td>
</tr>
<tr>
<td>CONTROL_TRAFFIC_LIMIT</td>
<td>TBD [50 packets/sec?]</td>
<td>Section 9</td>
</tr>
</tbody>
</table>

Table 7: Other Administrative Parameters

14. IANA Considerations

This section specifies several RFC 5444 message types, message tlv-types, and address tlv-types. Also, a new registry of 16-bit alternate metric types is specified.
14.1. AODVv2 Message Types Specification

<table>
<thead>
<tr>
<th>Name of AODVv2 Message</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Route Request (RREQ)</td>
<td>10 (TBD)</td>
</tr>
<tr>
<td>Route Reply (RREP)</td>
<td>11 (TBD)</td>
</tr>
<tr>
<td>Route Error (RERR)</td>
<td>12 (TBD)</td>
</tr>
<tr>
<td>Route Reply Acknowledgement (RREP_Ack)</td>
<td>13 (TBD)</td>
</tr>
</tbody>
</table>

Table 8: AODVv2 Message Types

14.2. Message TLV Type Specification

<table>
<thead>
<tr>
<th>Name of Message TLV</th>
<th>Type</th>
<th>Length (octets)</th>
<th>Cross Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>AckReq (Acknowledgment Request)</td>
<td>10 (TBD)</td>
<td>0</td>
<td>Section 6.2</td>
</tr>
<tr>
<td>PktSource (Packet Source)</td>
<td>11 (TBD)</td>
<td>4 or 16</td>
<td>Section 9.3.1</td>
</tr>
</tbody>
</table>

Table 9: Message TLV Types

14.3. Address Block TLV Specification

<table>
<thead>
<tr>
<th>Name of Address Block TLV</th>
<th>Type</th>
<th>Length (octets)</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metric</td>
<td>10 (TBD)</td>
<td>depends on MetricType</td>
<td>Section 9.1</td>
</tr>
<tr>
<td>Sequence Number (SeqNum)</td>
<td>11 (TBD)</td>
<td>2 octets</td>
<td>Section 9.1</td>
</tr>
<tr>
<td>Originating Node Sequence Number (OrigSeqNum)</td>
<td>12 (TBD)</td>
<td>2 octets</td>
<td>Section 9.1</td>
</tr>
<tr>
<td>Target Node Sequence Number (TargSeqNum)</td>
<td>13 (TBD)</td>
<td>2 octets</td>
<td>Section 9.1</td>
</tr>
<tr>
<td>VALIDITY_TIME</td>
<td>1</td>
<td>1 octet</td>
<td>[RFC5497]</td>
</tr>
</tbody>
</table>

Table 10: Address Block TLV (AddrTLV) Types

14.4. MetricType Number Allocation

Metric types are identified according to the assignments as specified in [RFC6551]. The metric type of the Hop Count metric is assigned to

be 3, in order to maintain compatibility with that existing table of values from RFC 6551.

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

Table 11: Metric Types

15. Security Considerations

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

This section describes various security considerations and potential avenues to secure AODVv2 routing. Security for authentication of AODVv2 routers, and/or encryption of traffic is dealt with by the underlying transport mechanism (e.g., by using the techniques for Authentication, Integrity, and Confidentiality documented in [RFC5444]). The most important security mechanism for AODVv2 routing is integrity/authentication.

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

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

In certain situations, for example sending a RREP or RERR, an AODVv2 router could include proof that it has previously received valid routing information to reach the destination, at one point of time in the past. In situations where routers are suspected of transmitting maliciously erroneous information, the original routing information along with its security credentials SHOULD be included.
Note that if multicast is used, any confidentiality and integrity algorithms used MUST permit multiple receivers to handle the message [RFC7182].

Routing protocols, however, are prime targets for impersonation attacks. In networks where the node membership is not known, it is difficult to determine the occurrence of impersonation attacks, and security prevention techniques are difficult at best. However, when the network membership is known and there is a danger of such attacks, AODVv2 messages must be protected by the use of authentication techniques, such as those involving generation of unforgeable and cryptographically strong message digests or digital signatures.

Most AODVv2 messages are transmitted to the multicast address LL-MANET-Routers [RFC5498]. It is therefore required for security that AODVv2 neighbors exchange security information that can be used to insert an ICV [RFC7182] into the AODVv2 message block [RFC5444]. This enables hop-by-hop security. For destination-only RREP discovery procedures, AODVv2 routers that share a security association SHOULD use the appropriate mechanisms as specified in [RFC7182]. The establishment of these security associations is out of scope for this document.

16. 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, Christopher Dearlove, Ulrich Herberg, Henner Jakob, Luke Klein-Berndt, Lars Kristensen, Tronje Krop, Koojana Kuladinithi, Kedar Namjoshi, Alexandru Petrescu, Henning Rogge, Fransisco Ros, Pedro Ruiz, Christoph Sommer, Lotte Steenbrink, Romain Thouvenin, Richard Trefler, Jiazi Yi, Seung Yi, and Cong Yuan, for their reviews AODVv2 and DYMO, as well as numerous specification suggestions.

17. References

17.1. Normative References


17.2. Informative References


Appendix A.  Example Algorithms for AODVv2 Protocol Operations

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

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

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

Once the route table has been updated, the information contained there is known to be the most recent available information for any fields in the outgoing message.  For this reason, the algorithms are written as if outgoing message field values are assigned from the route table information, even though it is often equally appropriate to use fields from the incoming message.
AODVv2_algorithms:

- Process_Routing_Info
- Fetch_Route_Table_Entry
- Update_Route_Table_Entry
- Create_Route_Table_Entry
- LoopFree

- Update_Rte_Msg_Table

- Generate_RREQ
- Receive_RREQ
- Regenerate_RREQ

- Generate_RREP
- Receive_RREP
- Regenerate_RREP

- Generate_RERR
- Receive_RERR
- Regenerate_RERR

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

RteMsg parameters, where rteMsg can be inRREQ, outRREQ, inRREP or outRREP:

rteMsg.hopLimit
rteMsg.hopCount
rteMsg.ackReq (RREP only, optional)
rteMsg.metricType (optional)
rteMsg.origAddr
rteMsg.targAddr
rteMsg.origPrefixLen (optional)
rteMsg.targPrefixLen (optional)
rteMsg.origSeqNum (RREQ only)
rteMsg.targSeqNum (optional in RREQ)
rteMsg.origAddrMetric (optional in RREQ)
rteMsg.targAddrMetric (RREP only)
rteMsg.validityTime
rteMsg.nbrIP

AdvRte has the following properties as described in Section 8.1:

AdvRte.Address = OrigAddr (in a RREQ) or TargAddr (in a RREP)

AdvRte.PrefixLength = PrefixLength for OrigAddr (in a RREQ) or TargAddr (in a RREP), or if not present, the maximum address length for the address family of AdvRte.Address

AdvRte.SeqNum = SeqNum for OrigAddr (in a RREQ) or for TargAddr (in a RREP)

AdvRte.MetricType = RteMsg.MetricType
AdvRte.Metric = RteMsg.Metric

AdvRte.Cost = AdvRte.Metric + Cost(L) according to the indicated MetricType, where L is the link from the advertising router

AdvRte.ValidityTime = ValidityTime in the RteMsg, if present

AdvRte.NextHopIP = IP source of the RteMsg

AdvRte.NextHopIntf = interface the RteMsg was received on

AdvRte.HopCount = value from RteMsg header

AdvRte.HopLimit = value from RteMsg header

AdvRte.AckReq = true/false whether present in RteMsg (optional in RREP)

A route table entry has properties as described in Section 6.1:

Route.Address
Route.PrefixLength
Route.SeqNum
Route.NextHop
Route.NextHopInterface
Route.LastUsed
Route.LastSeqNum
Route.ExpirationTime
Route.MetricType
Route.Metric
Route.State
Route.Timed
Route.Precursors (optional)
A.1. Subroutines for AODVv2 Operations

A.1.1. Process_Routing_Info

/* Compare incoming route information to stored route, maybe use
linkMetric: either Cost(inRREQ.netif) or (inRREP.netif) */
Process_Routing_Info (advRte)
{
    rte := Fetch_Route_Table_Entry (advRte);
    if (!rte exists)
    {
        rte := Create_Route_Table_Entry(advRte);
        return rte;
    }

    /* rule from 8.1 */
    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 */
    {
        Update_Route_Table_Entry (rte, advRte);
    }
    return rte;
}
A.1.2. 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;
}
A.1.3. 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 := Current_Time;
    rte.LastSeqNum := Current_Time;
    if (validityTime)
    {
        rte.ExpirationTime := Current_Time + advRte.validityTime;
        rte.Timed := true;
    }
    else
    {
        rte.Timed := false;
        rte.ExpirationTime := MAXTIME;
    }
    rte.Metric := advRte.Cost;
    if (rte.State == Invalid)
        rte.State := Idle;
}

A.1.4. 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 := Current_Time
    rte.LastSeqnum := Current_Time
    if (validityTime)
    {
        rte.Timed := true;
    }
    else
    {
        rte.Timed := false;
        rte.ExpirationTime := MAXTIME;
    }
    rte.MetricType := advRte.MetricType;
    rte.Metric := advRte.Metric;
    rte.State := Idle;
}

A.1.5.  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;
}
A.1.6. 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;
}

A.1.7. Update_Rte_Msg_Table

/* 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 (see 6.5 and 8.6) */
    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.origAddrMetric(for RREQ) 
            or rteMsg.targAddrMetric(for RREP)
        entry.Timestamp := Current_Time
        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 := Current_Time
    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 := Current_Time
    return false;
}

/* if same SeqNum but rteMsg has lower metric */
if (entry.Metric > rteMsg.Metric)
    entry.Metric := rteMsg.Metric
    entry.Timestamp := Current_Time
    return false;
A.1.8. 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 */
Build_RFC_5444_message_header (msgType, Flags, AddrFamily, Size, hopLimit, hopCount, tlvLength)
{
    /* Build RFC 5444 message header fields */
    msg-type := msgType
    MF (Message Flags) := Flags
    MAL (Message Address Length) := 3 for IPv4, 15 for IPv6
    msg-size := Size (octets - counting MsgHdr, AddrBlk, AddrTLVs)
    msg-hop-limit := hopLimit
    if (hopCount != 0) /* hopCount == 0 means do not include */
        msg-hop-count := hopCount
    msg.tlvs-length := tlvLength
}

A.2. Example Algorithms for AODVv2 RREQ Operations
A.2.1. Generate_RREQ
Generate_RREQ
{
    /* Increment sequence number */
    mySeqNum := (1 + mySeqNum) /* from nonvolatile storage */

    /* Marshall parameters */
    outRREQ.hopLimit := MAX_HOPCOUNT /* RFC 5444 */
    outRREQ.hopCount := (if included) 0
    outRREQ.metricType := if not DEFAULT_METRIC_TYPE,
                            metric type needed by application
    outRREQ.origAddr := IP address of Router Client which generated
                        the packet to be forwarded
    outRREQ.targAddr := destination IP address in
                        the packet to be forwarded
    outRREQ.origPrefixLen := if included, the prefix length
                            associated with the Router Client
    outRREQ.origSeqNum := mySeqNum
    outRREQ.targSeqNum := if known from route table, target sequence number
    outRREQ.origAddrMetric := 0 (default) or
                            MIN_METRIC(outRREQ.metricType)
    outRREQ.validityTime := if included, the validity time
                           for route to OrigAddr

/* Build Address Blk */
AddrBlk := outRREQ.origAddr and outRREQ.targAddr addresses
  /* using prefix length information from 
    outRREQ.origPrefixLen if necessary */
/* Include each available Sequence Number in appropriate 
Address Block TLV */
/* 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 */
metricAddrBlkTlv.value := outRREQ.origAddrMetric
if (outRREQ.metricType != DEFAULT_METRIC_TYPE)
  { /* include Metric AddrBlkTlv Extension byte */
    metricAddrBlkTlv.typeExtension := outRREQ.MetricType
  }
if (outRREQ.validityTime is required)
  { /* Build VALIDITY_TIME Address Block TLV */
    VALIDITY_TIMEAddrBlkTlv.value := outRREQ.validityTime
  }
/* multicast RFC 5444 message to LL-MANET-Routers */
}

A.2.2. Receive_RREQ

Receive_RREQ (inRREQ)
{
  if (inRREQ.nbrIP present in blacklist) { 
    if (blacklistExpirationTime < current_time)
      return; /* don’t process or regenerate RREQ... */
    else
      remove nbrIP from blacklist;
  }
  if (inRREQ does not contain msg_hop_limit, OrigAddr, 
      TargAddr, OrigSeqNum, OrigAddrMetric)
    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.origAddrMetric >
    MAX_Metric[inRREQ.metricType] - Cost(Link)
    return;

/* Extract inRREQ values */
advRte.Address = inRREQ.origAddr
advRte.PrefixLength = inRREQ.origPrefixLen (if present),
    or the maximum address length for the
    address family of advRte.Address
advRte.SeqNum = inRREQ.origSeqNum
advRte.MetricType = inRREQ.metricType
advRte.Metric = inRREQ.origAddrMetric
advRte.Cost = inRREQ.origAddrMetric + Cost(L) 
    according to the indicated MetricType, where
    L is the link from the advertising router
advRte.ValidityTime = inRREQ.validityTime (if present)
advRte.NextHopIP = inRREQ.nbrIP
advRte.NextHopIntf = interface the RteMsg was received on
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)
}

A.2.3.  Regenerate_RREQ

Regenerate_RREQ (inRREQ, rte) /* called from receive_RREQ(),
    rte is the route to OrigAddr */
{
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
outRREQ.origPrefixLen := rte.PrefixLength
(outRREQ.metricType != DEFAULT_METRIC_TYPE)
outRREQ.origSeqNum := rte.SeqNum
outRREQ.targSeqNum := inRREQ.targSeqNum /* if present */
outRREQ.origAddrMetric := rte.Metric
outRREQ.validityTime := rte.ValidityTime or length of time
(HandlingRtr wishes to advertise route to OrigAddr)

/* Build Address Block */
AddrBlk := outRREQ.origAddr and outRREQ.targAddr addresses
using prefix length information from outRREQ.origPrefixLen
if necessary
/* Include available Sequence Numbers in Address Block TLV */
/* 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 */
metricAddrBlkTlv.value = outRREQ.origAddrMetric
if (outRREQ.metricType != DEFAULT_METRIC_TYPE)
{ /* include Metric AddrBlkTlv extension byte */
    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 */

A.3. Example Algorithms for AODVv2 RREP Operations

A.3.1. Generate_RREP

Generate_RREP(inRREQ, rte)
{
/* Increment Sequence Number */
    mySeqNum := (1 + mySeqNum) /* from nonvolatile storage */

/* 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 := if included, TRUE otherwise FALSE

if (rte.metricType != DEFAULT_METRIC_TYPE)
    outRREP.metricType := rte.metricType
outRREP.origAddr := rte.Address
outRREP.targAddr := inRREQ.targAddr
outRREP.targPrefixLen := rte.PrefixLength
    (if not equal to address length)
outRREP.targSeqNum := mySeqNum
outRREP.targAddrMetric := 0 (default) or
    MIN_METRIC(rte.metricType)

outRREP.validityTime := (if included) the validity time for route to TargAddr

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

/* Build Address Block */
AddrBlk := outRREP.origAddr and outRREP.targAddr addresses
    using prefix length information from outRREP.targPrefixLen
    if necessary

/* TargSeqNum Address Block TLV */
Internet-Draft                   AODVv2                       March 2015

targSeqNumAddrBlkTlv.value := outRREP.targSeqNum

/* Build Metric Address Block TLV containing TargAddr metric */
metricAddrBlkTlv.value := outRREP.targAddrMetric
if (outRREP.metricType != DEFAULT_METRIC_TYPE)
  /* include Metric AddrBlkTlv extension byte */
  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 */

A.3.2. Receive_RREP

Receive_RREP (inRREP)
{
  if (inRREP.nbrIP present in blacklist)
    { if (blacklist_expiration_time < current_time)
       return; /* don’t process or regenerate RREQ... */
    else
       remove nbrIP from blacklist;
  }

  if (inRREP does not contain msg_hop_limit, OrigAddr, TargAddr, TargSeqNum, TargAddrMetric)
    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.targAddrMetric > MAX_METRIC[MetricType] - Cost(Link)
    return;

  /* Extract inRREP values */
  advRte.Address := inRREP.targAddr
  advRte.PrefixLength := inRREP.targPrefixLen if present), or the

maximum address length for address family of advRte.Address
advRte.SeqNum := inRREP.targSeqNum
advRte.MetricType := inRREP.metricType
advRte.Metric := inRREP.targAddrMetric
advRte.Cost := inRREP.targAddrMetric + Cost(L) according to
inRREP’s MetricType. L is the link from the advertising router
advRte.ValidityTime := inRREP.validityTime (if present)
advRte.NextHopIP := inRREP.nbrIP
advRte.NextHopIntf := interface the RteMsg was received on
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)
A.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 */
        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 whether to include
    /* 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 equal to address length)
    if (rte.metricType != DEFAULT_METRIC_TYPE)
        outRREP.metricType := rte.metricType
    outRREP.targPrefixLen := rte.PrefixLength
    (if not equal to address length)
    outRREP.targSeqNum := rte.SeqNum
    outRREP.targAddrMetric := rte.Metric
    outRREP.validityTime := (if included) the validity time
                          for route to TargAddr
    outRREP.nextHop := rte.nextHop

    if (outRREP.ackReq == TRUE)
    {
        /* include AckReq Message TLV */
        /* set timeout RREP_Ack_SENT_TIMEOUT */
    }
}
/* Build Address Block */
AddrBlk := {outRREP.origAddr and outRREP.targAddr}
using prefix length information from
outRREP.targPrefixLen if necessary

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

/* Build Metric Address Block TLV containing TargAddrMetric */
metricAddrBlkTlv.value := outRREP.targAddrMetric
if (outRREP.metricType != DEFAULT_METRIC_TYPE)
{ /* include Metric AddrBlkTlv extension byte */
   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 */

A.4. Example Algorithms for AODVv2 RERR Operations

RERR message parameters, where RERR can be inRERR or outRERR:

RERR.hopLimit := the maximum number of hops this RERR can traverse
RERR.pktSource := source IP of unforwardable packet (if present)
RERR.metricType := metric type for routes to unreachable destinations
RERR.unreachableAddressList[] := addresses of unreachable destinations
RERR.prefixLengthList[] := prefix lengths of unreachable destinations
RERR.seqNumList[] := sequence numbers for unreachable destinations
RERR.intf := the interface on which the RERR was received
A.4.1. 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_RERR(errorType, triggerPkt, brokenLinkNbrIp)
    /* errorType is either undeliverablePacket or brokenLink */
{
    switch (errorType)
    {
    case (brokenLink):
        /* a RERR will be required for each MetricType */
        foreach metric type in use
        {
            doGenerate := FALSE
            num-broken-addr := 0
            precursors[] := new empty precursor list
            outRERR.hopLimit := MAX_HOPCOUNT
            outRERR.metricType := the metric type for this loop
        /* find routes which are now Invalid */
        foreach (rte in route table)
        {
            if (brokenLinkNbrIp == rte.nextHop AND
                rte.MetricType == outRERR.metricType 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.unreachableAddressList[num-broken-addr] :=
                rte.Address
                outRERR.prefixLengthList[num-broken-addr] :=
                rte.PrefixLength
                outRERR.seqNumList[num-broken-addr] := rte.SeqNum
                num-broken-addr := num-broken-addr + 1
            }
        }
        if (doGenerate == TRUE)
            { /* build and send RFC5444 message as below, then repeat loop for other MetricTypes */
        }
    }
    case (undeliverablePacket):
        num-broken-addr := 1
```
outRERR.hopLimit := MAX_HOPCOUNT
outRERR.pktSource := triggerPkt.srcIP or
    triggerPkt.targAddr if packet was a RREP
/* optional to include outRERR.metricType */
outRERR.unreachableAddressList[0] := triggerPkt.destIP or
    triggerPkt.origAddr if packet was a RREP
}
if (triggerPkt exists)
    { /* Build PktSource Message TLV */
        pktSourceMessageTlv.value := outRERR.pktSource
    }
/* The remaining steps add address, prefix length
   and sequence number information for each
   UnreachableAddress, while conforming to the allowed MTU.
   If the MTU is reached, a new message MUST be created. */
/* Build Address Block */
AddrBlk := outRERR.unreachableAddressList[]
    using prefix length information from
    outRERR.prefixLengthList[] if necessary
/* Add SeqNum Address Block TLV including index values */
seqNumAddrBlkTLV := outRERR.seqNumList[]
if (outRERR.metricType != DEFAULT_METRIC_TYPE)
    { /* include Metric AddrBlkTlv extension byte */
        metricAddrBlkTlv.typeExtension := outRERR.MetricType
    }
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 */

A.4.2.  Receive_RERR

Receive_RERR (inRERR)
{
    if (inRERR does not contain msg_hop_limit and at least
        one UnreachableAddress)
        return;
    if (inRERR.metricType is present but an unknown value)
        return;

/* Extract inRERR values, copy relevant UnreachableAddresses, */
/* their prefix lengths, and sequence numbers to outRERR */
num-broken-addr := 0;
precursors[] := new empty list of type precursors;

foreach (unreachableAddress in inRERR.unreachableAddressList)
{
    if (unreachableAddress is not valid routable
        and unicast address)
        continue;
    /* find a matching route table entry, assume */
    DEFAULT_METRIC_TYPE if no MetricType included */
    rte := Fetch_Route_Table_Entry (unreachableAddress,
        inRERR.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.intf)
        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, inRERR.metricType)
        invalidRte.State := Invalid
        if (rte.PrefixLength > unreachableAddress prefixLength)
            expunge_route(rte);
        rte := invalidRte;
    }
    else if (rte.PrefixLength == unreachableAddress prefixLength)
        rte.State := Invalid;
    outRERR.unreachableAddressList[num-broken-addr] :=rte.Address
    outRERR.prefixLengthList[num-broken-addr] := rte.PrefixLength
    outRERR.seqNumList[num-broken-addr] := rte.SeqNum
    num-broken-addr := num-broken-addr + 1
}
if (num-broken-addr)
    Regenerate_RERR(outRERR, inRERR, precursors)
}

A.4.3. 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)
outRERR.metricType := inRERR.MetricType (if included)
    or DEFAULT_METRIC_TYPE
/* UnreachableAddressList[], SeqNumList[], and
PrefixLengthList[] are already up-to-date */

if (outRERR.pktSource exists)
{
    /* Build PktSource Message TLV */
    pktSourceMessageTlv.value := outRERR.pktSource
}
if (outRERR.metricType != DEFAULT_METRIC_TYPE)
{
    /* Build MetricType Message TLV */
    metricMsgTlv.value := outRERR.metricType
}

/* Build Address Block */

AddrBlk := outRERR.unreachableAddressList[] using prefix length
    information from outRERR.prefixLengthList[] if necessary

/* Add SeqNum AddressBlock TLV including index values */
seqNumAddrTlv := outRERR.seqNumList[]

Build_RFC_5444_message_header (RERR, 4, IPv4 or IPv6, NN,
    outRERR.hopLimit, 0, tlvLength)
if (outRERR.pktSource exists) {
    /* unicast RFC 5444 message to 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 */
}
A.5. Example Algorithms for AODVv2 RREP_Ack Operations

A.5.1. Generate_RREP_Ack

/* To be sent when 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 */
}

A.5.2. Receive_RREP_Ack

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

A.5.3. Timeout_RREP_Ack

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

Appendix B. Changes since revision ...-06.txt

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

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

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

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

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

- Deleted appendix showing example RFC 5444 packet formats.

- Clarification on the use of RFC 5497 VALIDITY_TIME.

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

- Numerous editorial improvements and clarifications.
Appendix C. Changes between revisions 5 and 6

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

- Added Lotte Steenbrink as co-author.

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

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

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

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

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

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

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

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

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

- Eliminated the instance of duplicate specification for inclusion of OrigNode (now, OrigAddr) in the message.
Corrected the terminology to be Mid instead of Tail for the trailing address bits of OrigAddr and TargAddr for the example message formats in the appendices.

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

Numerous editorial improvements and clarifications.

Appendix D. Changes from revision ...-04.txt

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 RFC 5444.

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).
Appendix E. Changes from revision ...-03.txt

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.

Appendix F. Changes from revision ...-02.txt

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

- The "Added Node" feature was removed. This feature was intended to enable additional routing information to be carried within a RREQ or a RREP message, thus increasing the amount of topological information available to nodes along a routing path. However, enlarging the packet size to include information which might never be used can increase congestion of the wireless medium. The feature can be included as an optional feature at a later date.
when better algorithms are understood for determining when the inclusion of additional routing information might be worthwhile.

- Numerous editorial improvements and improvements to consistent terminology were made. Instances of OrigNodeNdx and TargNodeNdx were replaced by OrigNdx and TargNdx, to be consistent with the terminology shown in Table 2.

- Example RREQ and RREP message formats shown in the Appendices were changed to use OrigSeqNum and TargSeqNum message TLVs instead of using the SeqNum message TLV.

- Inclusion of the OrigNode’s SeqNum in the RREP message is not specified. The processing rules for the OrigNode’s SeqNum were incompletely specified in previous versions of the draft, and very little benefit is foreseen for including that information, since reverse path forwarding is used for the RREP.

- Additional acknowledgements were included, and contributors names were alphabetized.

- Definitions in the Terminology section capitalize the term to be defined.

- Uncited bibliographic entries deleted.

- Ancient "Changes" sections were deleted.

Appendix G. Features of IP needed by AODVv2

AODVv2 needs the following:

- information that IP routes are requested

- information that packets are flowing

- the ability to queue packets.

A reactive protocol reacts when a route is needed. One might say that 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, and the way that has been used to know whether a route is needed is when an application tries to send a packet.

If an application tries to send a packet, and the route is not available, the packet has to wait until the route is available.
Appendix H. Multi-homing Considerations

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

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

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

Appendix I. Shifting Network Prefix Advertisement Between AODVv2 Routers

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

Authors’ Addresses

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

Phone: +1-408-330-4586
Email: charliep@computer.org
Dynamic Link Exchange Protocol (DLEP)
draft-ietf-manet-dlep-08

Abstract

When routing devices rely on modems to effect communications over wireless links, they need timely and accurate knowledge of the characteristics of the link (speed, state, etc.) in order to make forwarding decisions. In mobile or other environments where these characteristics change frequently, manual configurations or the inference of state through routing or transport protocols does not allow the router to make the best decisions. A bidirectional, event-driven communication channel between the router and the modem is necessary.

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 August 31, 2015.
7.15. Link Characteristics Request Signal ..................... 28
7.16. Link Characteristics ACK Signal ......................... 29
8. DLEP Data Items ........................................... 30
  8.1. DLEP Version ........................................... 31
  8.2. Status .................................................. 32
  8.3. DLEP Port ................................................ 33
  8.4. Peer Type ................................................ 33
  8.5. Heartbeat Interval ....................................... 34
  8.6. Extensions Supported ................................... 35
  8.7. Experimental Definition ................................ 35
  8.8. MAC Address ............................................ 36
  8.9. IPv4 Address ............................................ 37
  8.10. IPv6 Address ........................................... 37
  8.11. IPv4 Attached Subnet ................................... 38
  8.12. IPv6 Attached Subnet ................................... 39
  8.13. Maximum Data Rate (Receive) ............................ 39
  8.14. Maximum Data Rate (Transmit) .......................... 40
  8.15. Current Data Rate (Receive) ............................ 41
  8.16. Current Data Rate (Transmit) .......................... 41
  8.17. Latency .................................................. 42
  8.18. Resources (Receive) .................................... 43
  8.19. Resources (Transmit) ................................... 43
  8.20. Relative Link Quality (Receive) ......................... 44
  8.21. Relative Link Quality (Transmit) ....................... 45
  8.22. Link Characteristics ACK Timer ......................... 45
9. Credit-Windowing ............................................ 46
  9.1. Credit-Windowing Signals ................................ 46
    9.1.1. Destination Up Signal ................................. 46
    9.1.2. Destination Up ACK Signal ............................ 47
    9.1.3. Destination Update Signal ............................ 47
  9.2. Credit-Windowing Data Items ............................. 47
    9.2.1. Credit Window Status ................................ 47
    9.2.2. Credit Grant ......................................... 48
    9.2.3. Credit Request ....................................... 49
10. Security Considerations .................................... 50
11. IANA Considerations ....................................... 50
  11.1. Registrations .......................................... 50
  11.3. Signal Type Registration ................................ 51
  11.4. DLEP Data Item Registrations ........................... 52
  11.5. DLEP Status Code Registrations ........................ 53
  11.6. DLEP Extensions Registrations ........................... 53
  11.7. DLEP Well-known Port ................................... 54
  11.8. DLEP Multicast Address ................................ 54
12. Acknowledgements .......................................... 54
13. References ................................................ 54
  13.1. Normative References ................................... 54
1. Introduction

There exist today a collection of modem devices that control links of variable datarate and quality. Examples of these types of links include line-of-sight (LOS) terrestrial radios, satellite terminals, and cable/DSL modems. Fluctuations in speed and quality of these links can occur due to configuration (in the case of cable/DSL modems), or on a moment-to-moment basis, due to physical phenomena like multipath interference, obstructions, rain fade, etc. It is also quite possible that link quality and datarate varies with respect to individual destinations on a link, and with the type of traffic being sent. As an example, consider the case of an 802.11g access point, serving 2 associated laptop computers. In this environment, the answer to the question "What is the datarate on the 802.11g link?" is "It depends on which associated laptop we’re talking about, and on what kind of traffic is being sent." While the first laptop, being physically close to the access point, may have a datarate of 54Mbps for unicast traffic, the other laptop, being relatively far away, or obstructed by some object, can simultaneously have a datarate of only 32Mbps for unicast. However, for multicast traffic sent from the access point, all traffic is sent at the base transmission rate (which is configurable, but depending on the model of the access point, is usually 24Mbps or less).

In addition to utilizing variable datarate links, mobile networks are challenged by the notion that link connectivity will come and go over time, without an effect on a router’s interface state (Up or Down). Effectively utilizing a relatively short-lived connection is problematic in IP routed networks, as routing protocols tend to rely
on interface state and independent timers at OSI Layer 3 to maintain network convergence (e.g., HELLO messages and/or recognition of DEAD routing adjacencies). These dynamic connections can be better utilized with an event-driven paradigm, where acquisition of a new neighbor (or loss of an existing one) is signaled, as opposed to a paradigm driven by timers and/or interface state.

Another complicating factor for mobile networks are the different methods of physically connecting the modem devices to the router. Modems can be deployed as an interface card in a router’s chassis, or as a standalone device connected to the router via Ethernet or serial link. In the case of Ethernet or serial attachment, with existing protocols and techniques, routing software cannot be aware of convergence events occurring on the radio link (e.g., acquisition or loss of a potential routing neighbor), nor can the router be aware of the actual capacity of the link. This lack of awareness, along with the variability in datarate, leads to a situation where finding the (current) best route through the network to a given destination is difficult to establish and properly maintain. This is especially true of demand-based access schemes such as Demand Assigned Multiple Access (DAMA) implementations used on some satellite systems. With a DAMA-based system, additional datarate may be available, but will not be used unless the network devices emit traffic at a rate higher than the currently established rate. Increasing the traffic rate does not guarantee additional datarate will be allocated; rather, it may result in data loss and additional retransmissions on the link.

Addressing the challenges listed above, the authors have developed the Data Link Exchange Protocol, or DLEP. The DLEP protocol runs between a router and its attached modem devices, allowing the modem to communicate link characteristics as they change, and convergence events (acquisition and loss of potential routing destinations). The following diagrams are used to illustrate the scope of DLEP packets.

\[
\begin{array}{c}
\text{-------Local Node-------} \\
\text{-----------} \\
\text{Router} \quad \text{Modem Device} \\
\text{-----------} \\
\text{-DLEP--} \\
\end{array}
\quad
\begin{array}{c}
\text{-------Remote Node-------} \\
\text{-----------} \\
\text{Modem Device} \quad \text{Router} \\
\text{-----------} \\
\text{Link Protocol (e.g. 802.11)} \\
\text{-DLEP--} \\
\end{array}
\]

Figure 1: DLEP Network
In Figure 1, when the local modem detects the presence of a remote node, it (the local modem) sends a signal to its router via the DLEP protocol. Upon receipt of the signal, the local router may take whatever action it deems appropriate, such as initiating discovery protocols, and/or issuing HELLO messages to converge the network. On a continuing, as-needed basis, the modem devices utilize DLEP to report any characteristics of the link (datarate, latency, etc) that have changed. DLEP is independent of the link type and topology supported by the modem. Note that the DLEP protocol is specified to run only on the local link between router and modem. Some over the air signaling may be necessary between the local and remote modem in order to provide some parameters in DLEP signals between the local modem and local router, but DLEP does not specify how such over the air signaling is carried out. Over the air signaling is purely a matter for the modem implementer.

Figure 2 shows how DLEP can support a configuration where routers are connected with different link types. In this example, Modem A implements a point-to-point link, and Modem B is connected via a shared medium. In both cases, the DLEP protocol is used to report the characteristics of the link (datarate, latency, etc.) to routers. The modem is also able to use the DLEP session to notify the router when the remote node is lost, shortening the time required to re-converge the network.
DLEP defines a set of signals used by modems and their attached routers. The signals are used to communicate events that occur on the physical link(s) managed by the modem: for example, a remote node entering or leaving the network, or that the link has changed. Associated with these signals are a set of data items — information that describes the remote node (e.g., address information), and/or the characteristics of the link to the remote node.

The protocol is defined as a collection of type-length-value (TLV) based formats, specifying the signals that are exchanged between a router and a modem, and the data items associated with the signal. This document specifies transport of DLEP signals and data items via the TCP transport, with a UDP-based discovery mechanism. Other transports for the protocol are possible, but are outside the scope of this document.

DLEP uses a session-oriented paradigm between the modem device and its associated router. If multiple modem devices are attached to a
router (as in Figure 2), a separate DLEP session MUST exist for each modem. If a modem device supports multiple connections to a router (via multiple logical or physical interfaces), or supports connections to multiple routers, a separate DLEP session MUST exist for each connection. This router/modem session provides a carrier for information exchange concerning ‘destinations’ that are available via the modem device. A ‘destination’ can be either physical (as in the case of a specific far-end router), or a logical destination (as in a Multicast group). As such, all of the destination-level exchanges in DLEP can be envisioned as building an information base concerning the remote nodes, and the link characteristics to those nodes.

Any DLEP signal that is NOT understood by a receiver MUST result in an error indication being sent to the originator, and also MUST result in termination of the session between the DLEP peers. Any data item that is NOT understood by a receiver MUST be ignored.

Multicast traffic destined for the variable-quality network (the network accessed via the DLEP modem) is handled in IP networks by deriving a Layer 2 MAC address based on the Layer 3 address. Leveraging on this scheme, Multicast traffic is supported in DLEP simply by treating the derived MAC address as any other ‘destination’ (albeit a logical one) in the network. To support these logical destinations, one of the DLEP participants (typically, the router) informs the other as to the existence of the logical neighbor. The modem, once it is aware of the existence of this logical neighbor, reports link characteristics just as it would for any other destination in the network. The specific algorithms a modem would use to report metrics on multicast (or logical) destinations is outside the scope of this specification, and is left to specific implementations to decide.

1.1. Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, RFC 2119 [RFC2119].

2. Assumptions

Routers and modems that exist as part of the same node (e.g., that are locally connected) can utilize a discovery technique to locate each other, thus avoiding a-priori configuration. The router is responsible for initializing the discovery process, using the Peer Discovery signal (Section 7.1).
DLEP utilizes a session-oriented paradigm. A router and modem form a session by completing the discovery process. This router-modem session persists unless or until it either (1) times out, based on the timeout values supplied, or (2) is explicitly torn down by one of the participants. Note that while use of timers in DLEP is OPTIONAL, it is strongly recommended that implementations choose to run with timers enabled.

DLEP assumes that the MAC address for delivering data traffic is the MAC specified in the Destination Up signal (Section 7.9). No manipulation or substitution is performed; the MAC address supplied in Destination Up is used as the OSI Layer 2 Destination MAC address. DLEP also assumes that MAC addresses MUST be unique within the context of a router-modem session.

DLEP utilizes UDP multicast for single-hop discovery, and TCP for transport of the control signals. Therefore, DLEP assumes that the modem and router have topologically consistent IP addresses assigned. It is recommended that DLEP implementations utilize IPv6 link-local addresses to reduce the administrative burden of address assignment.

This document refers to a remote node as a ‘Destination’. Destinations can be identified by either the router or the modem, and represent a specific destination (e.g., an address) that exists on the link(s) managed by the modem. A destination MUST contain a MAC address, it MAY optionally include a Layer 3 address (or addresses). Note that since a destination is a MAC address, the MAC could reference a logical destination, as in a derived multicast MAC address, as well as to a physical device. As destinations are discovered, DLEP routers and modems build an information base on destinations accessible via the modem. Changes in link characteristics are then reported as being ‘modem-wide’ (effecting ALL destinations accessed via the modem, reported via the Peer Update signal, Section 7.5) or reported for a specific neighbor (via the Destination Update signal, Section 7.13).

The DLEP signals concerning destinations thus become the way for routers and modems to maintain, and notify each other about, an information base representing the physical and logical (e.g., multicast) destinations accessible via the modem device. The information base would contain addressing information (i.e., MAC address, and OPTIONALLY, Layer 3 addresses), link characteristics (metrics), and OPTIONALLY, flow control information (credits).

DLEP assumes that security on the session (e.g., authentication of session partners, encryption of traffic, or both) is dealt with by the underlying transport mechanism (e.g., by using a transport such as TLS [RFC5246]).
This document specifies an implementation of the DLEP signals and data items running over the TCP transport. It is assumed that DLEP running over other transport mechanisms would be documented separately.

3. Core Features and Optional Extensions

DLEP has a core set of signals and data items that MUST be processed without error by an implementation in order to guarantee interoperability and therefore make the implementation DLEP compliant. This document defines the core set of signals and data items, listing them as ‘mandatory’. It should be noted that some core signals and data items might not be used during the lifetime of a single DLEP session, but a compliant implementation MUST support them.

While this document represents the best efforts of the co-authors, and the working group, to be functionally complete, it is recognized that extensions to DLEP will in all likelihood be necessary as more link types are utilized. To support future extension of DLEP, this document describes an extension negotiation capability to be used during session initialization via the Extensions Supported data item, documented in Section 8.6 of this document.

All extensions are considered OPTIONAL. Only the DLEP functionality listed as ‘mandatory’ is required by implementation in order to be DLEP compliant.

This specification defines one extension, Credit processing, exposed via the Extensions Supported mechanism that implementations MAY chose to implement, or to omit.

3.1. Negotiation of Optional Extensions

Optional extensions supported by an implementation MUST be declared to potential DLEP peers using the Extensions Supported data item (Section 8.6) during the session initialization sequence. Once both peers have exchanged initialization signals, an implementation MUST NOT emit any signal or data item associated with an optional extension that was not specified in the received initialization signal from its peer.

3.2. Protocol Extensions

If/when protocol extensions are required, they should be standardized either as an update to this document, or as an additional stand-alone specification. The requests for IANA-controlled registries in this document contain sufficient reserved space, both in terms of DLEP
signals and DLEP data items, to accommodate future extensions to the protocol and the data transferred.

3.3. Experimental Signals and Data Items

This document requests numbering space in both the DLEP signal and data item registries for experimental items. The intent is to allow for experimentation with new signals and/or data items, while still retaining the documented DLEP behavior. If a given experiment proves successful, it SHOULD be documented as an update to this document, or as a stand-alone specification.

Use of the experimental signals or data items MUST be announced by inclusion of an Experimental Definition data item (Section 8.7) with a value agreed upon (a-priori) between the participating peers. The exact mechanism for a-priori communication of the experimental definition formats is beyond the scope of this document.

Multiple Experimental Definition data items MAY appear in the Peer Initialization/Peer Initialization ACK sequence. However, use of multiple experiments in a single peer session could lead to interoperability issues or unexpected results (e.g., redefinition of experimental signals and/or data items), and is therefore discouraged. It is left to implementations to determine the correct processing path (e.g., a decision on whether to terminate the peer session, or to establish a precedence of the conflicting definitions) if such conflicts arise.

4. Metrics

DLEP includes the ability for the router and modem to communicate metrics that reflect the characteristics (e.g., datarate, latency) of the variable-quality link in use. DLEP does NOT specify how a given metric value is to be calculated, rather, the protocol assumes that metrics have been calculated with a 'best effort', incorporating all pertinent data that is available to the modem device.

As mentioned in the introduction section of this document, metrics have to be used within a context - for example, metrics to a unicast address in the network. DLEP allows for metrics to be sent within two contexts - metrics for a specific destination within the network (e.g., a specific router), and 'modem-wide' (those that apply to all destinations accessed via the modem). Metrics can be further subdivided into transmit and receive metrics. Metrics supplied on DLEP Peer signals are, by definition, modem-wide; metrics supplied on Destination signals are, by definition, used for the specific neighbor only.
DLEP modem implementations MUST announce all supported metric items, and provide default values for those metrics, in the Peer Initialization signal (Section 7.3). In order to introduce a new metric type, DLEP modem implementations MUST terminate the session with the router (via the Peer Terminate signal, Section 7.7), and re-establish the session.

It is left to implementations to choose sensible default values based on their specific characteristics. Modems having static (non-changing) link metric characteristics MAY report metrics only once for a given neighbor (or once on a modem-wide basis, if all connections via the modem are of this static nature).

The approach of allowing for different contexts for metric data increases both the flexibility and the complexity of using metric data. This document details the mechanism whereby the data is transmitted, however, the specific algorithms (precedence, etc) for utilizing the dual-context metrics is out of scope and not addressed by this document.

4.1. Mandatory Metrics

As mentioned above, DLEP modem implementations MUST announce all supported metric items during session initialization. However, an implementation MUST include the following list of metrics:

- Maximum Data Rate (Receive) (Section 8.13)
- Maximum Data Rate (Transmit) (Section 8.14)
- Current Data Rate (Receive) (Section 8.15)
- Current Data Rate (Transmit) (Section 8.16)
- Latency (Section 8.17)

5. Normal Session Flow

Normal session flow for a DLEP router has two sub-cases, depending on whether the implementation supports the discovery process. Modem implementations MUST support the Discovery case; router implementations MAY support discovery, or rely on a-priori configuration to define the address(es) of attached modems.
5.1. DLEP Router session flow - Discovery case

If the DLEP router implementation is utilizing the optional discovery mechanism, then the implementation will initialize a UDP socket, binding it to an arbitrary port. This UDP socket is used to send the Peer Discovery signal (Section 7.1) to the DLEP link-local multicast address and port (TBD). The implementation then waits on receipt of a Peer Offer signal (Section 7.2), which MUST contain the unicast address and port for TCP-based communication with a DLEP modem. The Peer Offer signal MAY contain multiple address/port combinations. If more than one address/port combination is in the Peer Offer, the DLEP router implementation SHOULD consider the list to be in priority sequence, with the ‘most desired’ address/port combination listed first. However, router implementations MAY use their own heuristics to determine the best address/port combination. At this point, the router implementation MAY either destroy the UDP socket, or continue to issue Peer Discovery signals to the link-local address/port combination. In either case, the TCP session initialization occurs as in the configured case.

5.2. DLEP Router session flow - Configured case

When a DLEP router implementation has the address and port information for a TCP connection to a modem (obtained either via configuration or via the discovery process described above), the router will initialize and bind a TCP socket. This socket is used to connect to the DLEP modem software. After a successful TCP connect, the modem implementation MUST issue a Peer Initialization signal (Section 7.3) to the DLEP router. The Peer Initialization signal MUST contain data items for ALL supported metrics from this modem, along with the default values of those metrics. After sending the Peer Initialization, the modem implementation MUST wait for receipt of a Peer Initialization ACK signal (Section 7.4) from the router. Receipt of the Peer Initialization ACK signal indicates that the router has received and processed the Peer Initialization, and the session MUST transition to the ‘in session’ state. At this point, signals regarding destinations in the network, and/or Peer Update signals (Section 7.5), can flow on the DLEP session between modem and router. The ‘in session’ state is maintained until one of the following conditions occur:

- The session is explicitly terminated (using Peer Termination), or
- The session times out, based on supplied timeout values.
5.3. DLEP Modem session flow

DLEP modem implementations MUST support the discovery mechanism. Therefore, the normal flow is as follows:

The implementation will initialize a UDP socket, binding that socket to the DLEP link-local multicast address (TBD) and the DLEP well-known port number (also TBD). The implementation will then initialize a TCP socket, on a unicast address and port. This socket is used to listen for incoming TCP connection requests.

When the modem implementation receives a Peer Discovery signal (Section 7.1) on the UDP socket, it responds by issuing a Peer Offer signal (Section 7.2) to the sender of the Peer Discovery signal. The Peer Offer signal MUST contain the unicast address and port of the TCP listen socket, described above. A DLEP modem implementation MAY respond with ALL address/port combinations that have an active TCP listen posted. If multiple address/port combinations are listed, the receiver of the Peer Offer signal MAY connect on any available address/port pair. Anything other than Peer Discovery signals received on the UDP socket MUST be silently dropped.

When the DLEP modem implementation accepts a connection via TCP, it MUST send a Peer Initialization signal (Section 7.3). The Peer Initialization signal MUST contain metric data items for ALL supported metrics. If an additional metric is to be introduced, the DLEP session between router and modem MUST be terminated and restarted, and the new metric described in a Peer Initialization signal.

5.4. Common Session Flow

In order to maintain the session between router and modem, periodic Heartbeat signals (Section 7.14) MAY be exchanged. These signals are intended to keep the session alive, and to verify bidirectional connectivity between the two participants. DLEP also provides a Peer Update signal (Section 7.5), intended to communicate some change in status (e.g., a change of layer 3 address parameters, or a modem-wide link change).

In addition to the local (Peer level) signals above, the participants will transmit DLEP signals concerning destinations in the network. These signals trigger creation/maintenance/deletion of destinations in the information base of the recipient. For example, a modem will inform its attached router of the presence of a new destination via the Destination Up signal (Section 7.9). Receipt of a Destination Up causes the router to allocate the necessary resources, creating an entry in the information base with the specifics (i.e., MAC Address,
Latency, Data Rate, etc) of the neighbor. The loss of a destination is communicated via the Destination Down signal (Section 7.11), and changes in status to the destination (e.g., varying link quality, or addressing changes) are communicated via the Destination Update signal (Section 7.13). The information on a given neighbor will persist in the router’s information base until (1) a Destination Down signal is received, indicating that the modem has lost contact with the remote node, or (2) the router/modem session terminates, indicating that the router has lost contact with its own local modem.

Metrics can be expressed within the context of a specific neighbor via the Destination Update signal, or on a modem-wide basis via the Peer Update signal. In cases where metrics are provided on the router/modem session, the receiver MUST propagate the metrics to all destinations in its information base that are accessed via the originator. A DLEP participant MAY send metrics both in a router/modem session context (via the Peer Update signal) and a specific neighbor context (via Destination Update) at any time. The heuristics for applying received metrics is left to implementations.

In addition to receiving metrics about the link, DLEP provides a signal allowing a router to request a different datarate, or latency, from the modem. This signal is referred to as the Link Characteristics Request signal (Section 7.15), and gives the router the ability to deal with requisite increases (or decreases) of allocated datarate/latency in demand-based schemes in a more deterministic manner.

6. DLEP Message Processing

Communication between DLEP peers consists of a bidirectional stream of signals, each signal consisting of a signal header and an unordered list of data items. Both signal headers and data items are encoded as TLV (Type-Length-Value) structures. In this document, the data items following the signal header are described as being ‘contained in’ the signal.

All integer values in all TLV structures MUST be in network byte-order.

There is no restriction on the order of data items following a signal, and the multiplicity of duplicate data items is defined by the definition of the signal declared by the type in the signal header.

If an unrecognized, or unexpected signal is received, or a received signal contains unrecognized, invalid or disallowed duplicate data items, the receiving peer MUST terminate the session by issuing a
Peer Termination signal (Section 7.7) with a Status data item (Section 8.2) containing the most relevant status code, and then close the TCP connection:

6.1. DLEP Signal Header

The DLEP signal header contains the following fields:

![Figure 3: DLEP Signal Header]

Signal Type: One of the DLEP Signal Type values defined in this document.

Length: The length, expressed as a 16-bit unsigned integer, of all of the DLEP data items associated with this signal. This length does not include the length of the header itself.

Data Items: One or more DLEP data items, encoded in TLVs, as defined in this document.

6.2. DLEP Generic Data Item

All DLEP data items contain the following fields:

![Figure 4: DLEP Generic Data Item]

Data Item Type: An 8-bit unsigned integer field specifying the data item being sent.

Length: An 8-bit length of the value field of the data item.

Value: A field of length <Length> which contains data specific to a particular data item.
7. DLEP Signals

As mentioned above, all DLEP signals begin with the DLEP signal header structure. Therefore, in the following descriptions of specific signals, this header structure is assumed, and will not be replicated.

Following is the set of MANDATORY signals that must be recognized by a DLEP compliant implementation. As mentioned before, not all signals may be used during a session, but an implementation MUST correctly process these signals when received.

The mandatory DLEP signals are:

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBD</td>
<td>Peer Discovery</td>
<td>Section 7.1</td>
</tr>
<tr>
<td>TBD</td>
<td>Peer Offer</td>
<td>Section 7.2</td>
</tr>
<tr>
<td>TBD</td>
<td>Peer Initialization</td>
<td>Section 7.3</td>
</tr>
<tr>
<td>TBD</td>
<td>Peer Initialization ACK</td>
<td>Section 7.4</td>
</tr>
<tr>
<td>TBD</td>
<td>Peer Update</td>
<td>Section 7.5</td>
</tr>
<tr>
<td>TBD</td>
<td>Peer Update ACK</td>
<td>Section 7.6</td>
</tr>
<tr>
<td>TBD</td>
<td>Peer Termination</td>
<td>Section 7.7</td>
</tr>
<tr>
<td>TBD</td>
<td>Peer Termination ACK</td>
<td>Section 7.8</td>
</tr>
<tr>
<td>TBD</td>
<td>Destination Up</td>
<td>Section 7.9</td>
</tr>
<tr>
<td>TBD</td>
<td>Destination Up ACK</td>
<td>Section 7.10</td>
</tr>
<tr>
<td>TBD</td>
<td>Destination Down</td>
<td>Section 7.11</td>
</tr>
<tr>
<td>TBD</td>
<td>Destination Down ACK</td>
<td>Section 7.12</td>
</tr>
<tr>
<td>TBD</td>
<td>Destination Update</td>
<td>Section 7.13</td>
</tr>
<tr>
<td>TBD</td>
<td>Heartbeat</td>
<td>Section 7.14</td>
</tr>
<tr>
<td>TBD</td>
<td>Link Characteristics Request</td>
<td>Section 7.15</td>
</tr>
<tr>
<td>TBD</td>
<td>Link Characteristics ACK</td>
<td>Section 7.16</td>
</tr>
</tbody>
</table>

7.1. Peer Discovery Signal

A Peer Discovery signal SHOULD be sent by a router to discover DLEP routers in the network. The Peer Offer signal (Section 7.2) is required to complete the discovery process. Implementations MAY implement their own retry heuristics in cases where it is determined the Peer Discovery signal has timed out.

To construct a Peer Discovery signal, the Signal Type value in the signal header is set to DLEP_PEER_DISCOVERY (value TBD).

The Peer Discovery signal MUST contain one of each of the following data items:
7.2. Peer Offer Signal

A Peer Offer signal MUST be sent by a DLEP modem in response to a Peer Discovery signal (Section 7.1). Upon receipt, and processing, of a Peer Offer signal, the router responds by issuing a TCP connect to the address/port combination specified in the received Peer Offer.

The Peer Offer signal MUST be sent to the unicast address of the originator of Peer Discovery.

To construct a Peer Offer signal, the Signal Type value in the signal header is set to DLEP_PEER_OFFER (value TBD).

The Peer Offer signal MUST contain one of each of the following data items:

- DLEP Version (Section 8.1)
- Heartbeat Interval (Section 8.5)

The Peer Offer signal MAY contain one of each of the following data items:

- Peer Type (Section 8.4)
- DLEP Port (Section 8.3)

The Peer Offer signal MAY contain one or more of any of the following data items, with different values:

- IPv4 Address (Section 8.9), with Add/Drop indicator = 1
- IPv6 Address (Section 8.10), with Add/Drop indicator = 1

If the Peer Offer signal includes a DLEP Port data item, the port number specified MUST be used to establish the TCP session. If the DLEP Port number is omitted, the receiver MUST use the DLEP well-known port number (Section 11.7) to establish the TCP connection.

The IP Address data items indicate the unicast address the receiver of Peer Offer MUST use when connecting the DLEP TCP session. If multiple IP Address items are present in the Peer Offer signal, implementations MAY use their own heuristics to select the address to connect to. If no IP Address data items are included in the Peer
Offer signal, the receiver MUST use the origin address of the signal as the IP address to establish the TCP connection.

7.3. Peer Initialization Signal

A Peer Initialization signal MUST be sent by a router as the first signal of the DLEP TCP session. It is sent by the router after a TCP connect to an address/port combination that was obtained either via receipt of a Peer Offer, or from a-priori configuration.

If any optional extensions are supported by the implementation, they MUST be enumerated in the Extensions Supported data item. If an Extensions Supported data item does NOT exist in a Peer Initialization signal, the receiver of the signal MUST conclude that there is NO support for extensions in the sender.

If any experimental signals or data items are used by the implementation, they MUST be enumerated in one or more Experimental Definition data items. If there are no Experimental Definition data items in a Peer Initialization signal, the receiver of the signal MUST conclude that NO experimental definitions are in use by the sender.

To construct a Peer Initialization signal, the Signal Type value in the signal header is set to DLEP_PEER_INITIALIZATION (value TBD).

The Peer Initialization signal MUST contain one of each of the following data items:

- DLEP Version (Section 8.1)
- Heartbeat Interval (Section 8.5)

The Peer Initialization signal MAY contain one of each of the following data items:

- Peer Type (Section 8.4)
- Extensions Supported (Section 8.6)

The Peer Initialization signal MAY contain one or more of any of the following data items, with different values:

- Experimental Definition (Section 8.7)
7.4. Peer Initialization ACK Signal

A Peer Initialization ACK signal MUST be sent in response to a received Peer Initialization signal (Section 7.3). The Peer Initialization ACK signal completes the TCP-level DLEP session establishment; the sender of the signal should transition to an ‘in-session’ state when the signal is sent, and the receiver should transition to the ‘in-session’ state upon receipt (and successful parsing) of a Peer Initialization ACK signal.

All supported metric data items MUST be included in the Peer Initialization ACK signal, with default values to be used on a ‘modem-wide’ basis. This can be viewed as the modem ‘declaring’ all supported metrics at DLEP session initialization. Receipt of any DLEP signal containing a metric data item NOT included in the Peer Initialization ACK signal MUST be treated as an error, resulting in the termination of the DLEP session between router and modem.

If any optional extensions are supported by the modem, they MUST be enumerated in the Extensions Supported data item. If an Extensions Supported data item does NOT exist in a Peer Initialization ACK signal, the receiver of the signal MUST conclude that there is NO support for extensions in the sender.

If any experimental signals or data items are used by the implementation, they MUST be enumerated in one or more Experimental Definition data items. If there are no Experimental Definition data items in a Peer Initialization ACK signal, the receiver of the signal MUST conclude that NO experimental definitions are in use by the sender.

After the Peer Initialization/Peer Initialization ACK signals have been successfully exchanged, implementations MUST only utilize extensions and experimental definitions that are supported by BOTH peers.

To construct a Peer Initialization ACK signal, the Signal Type value in the signal header is set to DLEP_PEER_INIT_ACK (value TBD).

The Peer Initialization ACK signal MUST contain one of each of the following data items:

- DLEP Version (Section 8.1)
- Heartbeat Interval (Section 8.5)
- Maximum Data Rate (Receive) (Section 8.13)
The Peer Initialization ACK signal MAY contain one of each of the following data items:

- Status (Section 8.2)
- Peer Type (Section 8.4)
- Resources (Receive) (Section 8.18)
- Resources (Transmit) (Section 8.19)
- Relative Link Quality (Receive) (Section 8.20)
- Relative Link Quality (Transmit) (Section 8.21)
- Extensions Supported (Section 8.6)

The Peer Initialization ACK signal MAY contain one or more of any of the following data items, with different values:

- Experimental Definition (Section 8.7)

7.5. Peer Update Signal

A Peer Update signal MAY be sent by a DLEP peer to indicate local Layer 3 address changes, or for metric changes on a modem-wide basis. For example, addition of an IPv4 address to the router MAY prompt a Peer Update signal to its attached DLEP modems. Also, a modem that changes its Maximum Data Rate for all destinations MAY reflect that change via a Peer Update signal to its attached router(s).

Concerning Layer 3 addresses, if the modem is capable of understanding and forwarding this information (via proprietary mechanisms), the address update would prompt any remote DLEP modems (DLEP-enabled modems in a remote node) to issue a Destination Update signal (Section 7.13) to their local routers with the new (or deleted) addresses. Modems that do not track Layer 3 addresses SHOULD silently parse and ignore the Peer Update signal. Modems that track Layer 3 addresses MUST acknowledge the Peer Update with a Peer Update ACK signal (Section 7.6). Routers receiving a Peer Update
with metric changes MUST apply the new metric to all destinations (remote nodes) accessible via the modem. Supporting implementations are free to employ heuristics to retransmit Peer Update signals. The sending of Peer Update signals for Layer 3 address changes SHOULD cease when a either participant (router or modem) determines that the other implementation does NOT support Layer 3 address tracking.

If metrics are supplied with the Peer Update signal (e.g., Maximum Data Rate), these metrics are considered to be modem-wide, and therefore MUST be applied to all destinations in the information base associated with the router/modem session.

To construct a Peer Update signal, the Signal Type value in the signal header is set to DLEP_PEER_UPDATE (value TBD).

The Peer Update signal MAY contain one of each of the following data items:

- Maximum Data Rate (Receive) (Section 8.13)
- Maximum Data Rate (Transmit) (Section 8.14)
- Current Data Rate (Receive) (Section 8.15)
- Current Data Rate (Transmit) (Section 8.16)
- Latency (Section 8.17)
- Resources (Receive) (Section 8.18)
- Resources (Transmit) (Section 8.19)
- Relative Link Quality (Receive) (Section 8.20)
- Relative Link Quality (Transmit) (Section 8.21)

The Peer Update signal MAY contain one or more of the following data items, with different values:

- IPv4 Address (Section 8.9)
- IPv6 Address (Section 8.10)

7.6. Peer Update ACK Signal

A Peer Update ACK signal MUST be sent by implementations supporting Layer 3 address tracking and/or modem-wide metrics to indicate whether a Peer Update signal (Section 7.5) was successfully
processed. If the Peer Update ACK is issued, it MUST contain a Status data item, indicating the success or failure of processing the received Peer Update.

To construct a Peer Update ACK signal, the Signal Type value in the signal header is set to DLEP_PEER_UPDATE_ACK (value TBD).

The Peer Update ACK signal MAY contain one of each of the following data items:

- Status (Section 8.2)

A receiver of a Peer Update ACK signal without a Status data item MUST behave as if a Status data item with code ‘Success’ had been received.

7.7. Peer Termination Signal

A Peer Termination signal MUST be sent by a DLEP participant when the router/modem session needs to be terminated. Implementations receiving a Peer Termination signal MUST send a Peer Termination ACK signal (Section 7.8) to confirm the termination process. The sender of a Peer Termination signal is free to define its heuristics in event of a timeout. The receiver of a Peer Termination signal MUST release all resources allocated for the router/modem session, and MUST eliminate all destinations in the information base accessible via the router/modem pair represented by the session. Router and modem state machines are returned to the ‘discovery’ state. No Destination Down signals (Section 7.11) are sent.

To construct a Peer Termination signal, the Signal Type value in the signal header is set to DLEP_PEER_TERMINATION (value TBD).

The Peer Termination signal MAY contain one of each of the following data items:

- Status (Section 8.2)

A receiver of a Peer Termination signal without a Status data item MUST behave as if a Status data item with status code ‘Success’ had been received.

7.8. Peer Termination ACK Signal

A Peer Termination ACK signal MUST be sent by a DLEP peer in response to a received Peer Termination signal (Section 7.7). Receipt of a Peer Termination ACK signal completes the teardown of the router/modem session.
To construct a Peer Termination ACK signal, the Signal Type value in the signal header is set to DLEP_PEER_TERMINATION_ACK (value TBD).

The Peer Termination ACK signal MAY contain one of each of the following data items:

- Status (Section 8.2)

A receiver of a Peer Termination ACK signal without a Status data item MUST behave as if a Status data item with status code ‘Success’ had been received.

### 7.9. Destination Up Signal

A DLEP participant MUST send a Destination Up signal to report that a new destination has been detected. A Destination Up ACK signal (Section 7.10) is required to confirm a received Destination Up. A Destination Up signal can be sent either by the modem, to indicate that a new remote node has been detected, or by the router, to indicate the presence of a new logical destination (e.g., a Multicast group) exists in the network.

The sender of the Destination Up signal is free to define its retry heuristics in event of a timeout. When a Destination Up signal is received and successfully processed, the receiver should add knowledge of the new destination to its information base, indicating that the destination is accessible via the modem/router pair.

To construct a Destination Up signal, the Signal Type value in the signal header is set to DLEP_DESTINATION_UP (value TBD).

The Destination Up signal MUST contain one of each of the following data items:

- MAC Address (Section 8.8)

The Destination Up signal MAY contain one of each of the following data items:

- Maximum Data Rate (Receive) (Section 8.13)
- Maximum Data Rate (Transmit) (Section 8.14)
- Current Data Rate (Receive) (Section 8.15)
- Current Data Rate (Transmit) (Section 8.16)
- Latency (Section 8.17)
The Destination Up signal MAY contain one or more of the following data items, with different values:

- IPv4 Address (Section 8.9)
- IPv6 Address (Section 8.10)
- IPv4 Attached Subnet (Section 8.11)
- IPv6 Attached Subnet (Section 8.12)

If the sender has IPv4 and/or IPv6 address information for a destination it SHOULD include the relevant data items in the Destination Up signal, reducing the need for the receiver to probe for any address.

### 7.10. Destination Up ACK Signal

A DLEP participant MUST send a Destination Up ACK signal to indicate whether a Destination Up signal (Section 7.9) was successfully processed.

To construct a Destination Up ACK signal, the Signal Type value in the signal header is set to DLEP_DESTINATION_UP_ACK (value TBD).

The Destination Up ACK signal MUST contain one of each of the following data items:

- MAC Address (Section 8.8)

The Destination Up ACK signal MAY contain one of each of the following data items:

- Status (Section 8.2)

A receiver of a Destination Up ACK signal without a Status data item MUST behave as if a Status data item with status code ‘Success’ had been received.
7.11. Destination Down Signal

A DLEP peer MUST send a Destination Down signal to report when a
destination (a remote node or a multicast group) is no longer
reachable. A Destination Down ACK signal (Section 7.12) MUST be sent
by the recipient of a Destination Down signal to confirm that the
relevant data has been removed from the information base. The sender
of the Destination Down signal is free to define its retry heuristics
in event of a timeout.

To construct a Destination Down signal, the Signal Type value in the
signal header is set to DLEP_DESTINATION_DOWN (value TBD).

The Destination Down signal MUST contain one of each of the following
data items:

- MAC Address (Section 8.8)

7.12. Destination Down ACK Signal

A DLEP participant MUST send a Destination Down ACK signal to
indicate whether a received Destination Down signal (Section 7.11)
was successfully processed. If successfully processed, the sender of
the ACK MUST have removed all entries in the information base that
pertain to the referenced destination.

To construct a Destination Down ACK signal, the Signal Type value in
the signal header is set to DLEP_DESTINATION_DOWN_ACK (value TBD).

The Destination Down ACK signal MUST contain one of each of the
following data items:

- MAC Address (Section 8.8)

The Destination Down ACK signal MAY contain one of each of the
following data items:

- Status (Section 8.2)

A receiver of a Destination Down ACK signal without a Status data
item MUST behave as if a Status data item with status code 'Success'
had been received.

7.13. Destination Update Signal

A DLEP participant SHOULD send the Destination Update signal when it
detects some change in the information base for a given destination
(remote node or multicast group). Some examples of changes that would prompt a Destination Update signal are:

- Change in link metrics (e.g., Data Rates)
- Layer 3 addressing change (for implementations that support it)

To construct a Destination Update signal, the Signal Type value in the signal header is set to DLEP_DESTINATION_UPDATE (value TBD).

The Destination Update signal MUST contain one of each of the following data items:

- MAC Address (Section 8.8)

The Destination Update signal MAY contain one of each of the following data items:

- Maximum Data Rate (Receive) (Section 8.13)
- Maximum Data Rate (Transmit) (Section 8.14)
- Current Data Rate (Receive) (Section 8.15)
- Current Data Rate (Transmit) (Section 8.16)
- Latency (Section 8.17)
- Resources (Receive) (Section 8.18)
- Resources (Transmit) (Section 8.19)
- Relative Link Quality (Receive) (Section 8.20)
- Relative Link Quality (Transmit) (Section 8.21)

The Destination Update signal MAY contain one or more of the following data items, with different values:

- IPv4 Address (Section 8.9)
- IPv6 Address (Section 8.10)
- IPv4 Attached Subnet (Section 8.11)
- IPv6 Attached Subnet (Section 8.12)

A Heartbeat signal SHOULD be sent by a DLEP participant every N seconds, where N is defined in the Heartbeat Interval field of the Peer Initialization signal (Section 7.3) or Peer Initialization ACK signal (Section 7.4). Note that implementations setting the Heartbeat Interval to 0 effectively set the interval to an infinite value, therefore, in those cases, this signal SHOULD NOT be sent.

The signal is used by participants to detect when a DLEP session partner (either the modem or the router) is no longer communicating. Participants SHOULD allow two (2) heartbeat intervals to expire with no traffic on the router/modem session before initiating DLEP session termination procedures.

To construct a Heartbeat signal, the Signal Type value in the signal header is set to DLEP_PEER_HEARTBEAT (value TBD).

There are no valid data items for the Heartbeat signal.

7.15. Link Characteristics Request Signal

The Link Characteristics Request signal MAY be sent by the router to request that the modem initiate changes for specific characteristics of the link. The request can reference either a real (e.g., a remote node), or a logical (e.g., a multicast group) destination within the network.

The Link Characteristics Request signal contains either a Current Data Rate (CDRR or CDRT) data item to request a different datarate than what is currently allocated, a Latency data item to request that traffic delay on the link not exceed the specified value, or both. A Link Characteristics ACK signal (Section 7.16) is required to complete the request. Issuing a Link Characteristics Request with ONLY the MAC Address data item is a mechanism a peer MAY use to request metrics (via the Link Characteristics ACK) from its partner.

The sender of a Link Characteristics Request signal MAY attach a timer to the request using the Link Characteristics ACK Timer data item. If a Link Characteristics ACK signal is received after the timer expires, the sender MUST assume that the request failed. Implementations are free to define their retry heuristics in event of a timeout.

To construct a Link Characteristics Request signal, the Signal Type value in the signal header is set to DLEP_LINK_CHAR_REQ (value TBD).
The Link Characteristics Request signal MUST contain one of each of the following data items:
  o MAC Address (Section 8.8)

The Link Characteristics Request signal MAY contain one of each of the following data items:
  o Link Characteristics ACK Timer (Section 8.22)
  o Current Data Rate (Receive) (Section 8.15)
  o Current Data Rate (Transmit) (Section 8.16)
  o Latency (Section 8.17)

7.16. Link Characteristics ACK Signal

A DLEP participant MUST send a Link Characteristics ACK signal to indicate whether a received Link Characteristics Request signal (Section 7.15) was successfully processed. The Link Characteristics ACK signal SHOULD contain a complete set of metric data items. It MUST contain the same metric types as the request. The values in the metric data items in the Link Characteristics ACK signal MUST reflect the link characteristics after the request has been processed.

If an implementation is not able to alter the characteristics of the link in the manner requested, then a Status data item with status code ‘Request Denied’ MUST be added to the signal.

To construct a Link Characteristics Request ACK signal, the Signal Type value in the signal header is set to DLEP_LINK_CHAR_ACK (value TBD).

The Link Characteristics ACK signal MUST contain one of each of the following data items:
  o MAC Address (Section 8.8)

The Link Characteristics ACK signal MAY contain one of each of the following data items:
  o Current Data Rate (Receive) (Section 8.15)
  o Current Data Rate (Transmit) (Section 8.16)
  o Latency (Section 8.17)
A receiver of a Link Characteristics ACK signal without a Status data item MUST behave as if a Status data item with status code ‘Success’ had been received.

8. DLEP Data Items

Following is the list of MANDATORY data items that must be recognized by a DLEP compliant implementation. As mentioned before, not all data items need be used during a session, but an implementation MUST correctly process these data items when correctly associated with a signal.

The mandatory DLEP data items are:
### 8.1. DLEP Version

The DLEP Version data item MUST appear in the Peer Discovery (Section 7.1), Peer Offer (Section 7.2), Peer Initialization (Section 7.3) and Peer Initialization ACK (Section 7.4) signals. The Version data item is used to indicate the version of the protocol running in the originator. A DLEP implementation MAY use this information to decide if the potential session partner is running at a supported level.

The DLEP Version data item contains the following fields:

<table>
<thead>
<tr>
<th>Data Item Type</th>
<th>Length = 4</th>
<th>Major Version</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Minor Version</td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Data Item Type: TBD

Length: 4

Major Version: Major version of the DLEP protocol.

Minor Version: Minor version of the DLEP protocol.

Support of this draft is indicated by setting the Major Version to ‘0’, and the Minor Version to ‘8’ (i.e., Version 0.8).

8.2. Status

The Status data item is MAY appear in the Peer Initialization ACK (Section 7.4), Peer Termination (Section 7.7), Peer Termination ACK (Section 7.8), Peer Update ACK (Section 7.6), Destination Up ACK (Section 7.10), Destination Down ACK (Section 7.12) and Link Characteristics ACK (Section 7.16) signals as part of an acknowledgement from either the modem or the router, to indicate the success or failure of the previously received signal.

The Status data item contains the following fields:

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Data Item Type | Length = 1 |     Code      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Data Item Type: TBD

Length: 1

Status Code: One of the codes defined below.
8.3. DLEP Port

The DLEP Port data item MAY appear in the Peer Offer signal (Section 7.2). The DLEP Port data item indicates the TCP Port number on the DLEP server available for connections. If provided, the receiver MUST use this information to perform the TCP connect to the DLEP server.

The DLEP Port data item contains the following fields:

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Data Item Type | Length = 2 | TCP Port Number |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Data Item Type: TBD

Length: 2

TCP Port Number: TCP Port number on the DLEP server.

8.4. Peer Type

The Peer Type data item MAY appear in both the Peer Discovery (Section 7.1) and Peer Offer (Section 7.2) signals. The Peer Type data item is used by the router and modem to give additional information as to its type. The peer type is a string and is envisioned to be used for informational purposes (e.g., as output in a display command).
The Peer Type data item contains the following fields:

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Data Item Type| Length = peer | Peer Type                     |
|               |    type len   |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Data Item Type: TBD

Length: Length of peer type string.

Peer Type: UTF-8 encoded string. For example, a satellite modem might set this variable to "Satellite terminal".

An implementation MUST NOT assume the Peer Type is NUL-terminated.

8.5. Heartbeat Interval

The Heartbeat Interval data item MUST appear in the Peer Discovery (Section 7.1), Peer Offer (Section 7.2), Peer Initialization (Section 7.3) and Peer Initialization ACK (Section 7.4) signals to indicate the desired Heartbeat timeout window. The receiver MUST either accept the timeout interval supplied by the sender, or reject the Peer Initialization, and close the socket. Implementations MUST implement heuristics such that DLEP signals sent/received reset the timer interval.

The Interval is used to specify a period (in seconds) for Heartbeat signals (Section 7.14). By specifying an Interval value of 0, implementations MAY indicates the desire to disable Heartbeat signals entirely (i.e., the Interval is set to an infinite value), however, it is strongly recommended that implementations use non 0 timer values.

A DLEP session will be considered inactive, and MUST be torn down, by an implementation detecting that two (2) Heartbeat intervals have transpired without receipt of any DLEP signals.

The Heartbeat Interval data item contains the following fields:

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Data Item Type| Length = 2 | Interval           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
Data Item Type: TBD
Length: 2

Interval: 0 = Do NOT use heartbeats on this peer-to-peer session.
Non-zero = Interval, in seconds, for heartbeat signals.

8.6. Extensions Supported

The Extensions Supported data item MAY be used in both the Peer Initialization and Peer Initialization ACK signals. The Extensions Supported data item is used by the router and modem to negotiate additional optional functionality they are willing to support. The Extensions List is a concatenation of the types of each supported extension, found in the IANA DLEP Extensions repository.

The Extensions Supported data item contains the following fields:

0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Data Item Type| Length = No.  | Extensions List               |
|               |   of values   |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Data Item Type: TBD
Length: Number of Extensions supported.

Extension List: A list of extensions supported, identified by their 1-octet value as listed in the extensions registry.

8.7. Experimental Definition

The Experimental Definition data item MAY be used in both the Peer Initialization and Peer Initialization ACK signals. The Experimental Definition data item is used by the router and modem to indicate the formats to be used for experimental signals and data items for the given peer session. The formats are identified by using a string that matches the ‘name’ given to the experiment.

The Experimental Definition item contains the following fields:
Data Item Type: TBD

Length: Length of the name string for the Experiment.

Experiment Name: UTF-8 encoded string, containing the name of the experiment being utilized.

An implementation receiving this data item MUST compare the received string to a list of experiments that it supports. An implementation MUST NOT assume the Experiment Name is NUL-terminated.

8.8. MAC Address

The MAC address data item MUST appear in all destination-oriented signals (i.e., Destination Up (Section 7.9), Destination Up ACK (Section 7.10), Destination Down (Section 7.11), Destination Down ACK (Section 7.12), Destination Update (Section 7.13), Link Characteristics Request (Section 7.15), and Link Characteristics ACK (Section 7.16)). The MAC Address data item contains the address of the destination on the remote node. The MAC address MAY be either a physical or a virtual destination. Examples of a virtual destination would be a multicast MAC address, or the broadcast MAC (FF:FF:FF:FF:FF:FF).

Data Item Type: TBD

Length: 6

MAC Address: MAC Address of the destination (either physical or virtual).
8.9. IPv4 Address

The IPv4 Address data item MUST appear in the Peer Offer signal (Section 7.2), and MAY appear in the Peer Update (Section 7.5), Destination Up (Section 7.9) and Destination Update (Section 7.13) signals. When included in Destination signals, this data item contains the IPv4 address of the destination. In the Peer Offer signal, it contains the IPv4 address of the originating peer to be used to establish a DLEP session. In either case, the data item also contains an indication of whether this is a new or existing address, or is a deletion of a previously known address. When used in a Peer Offer signal the Add/Drop Indicator MUST be 1 (i.e. Add).

The IPv4 Address data item contains the following fields:

<table>
<thead>
<tr>
<th>Data Item Type</th>
<th>Length = 5</th>
<th>Add/Drop Indicator</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPv4 Address</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Data Item Type: TBD

Length: 5

Add/Drop: Value indicating whether this is a new or existing address (1), or a withdrawal of an address (0).

IPv4 Address: The IPv4 address of the destination or peer.

8.10. IPv6 Address

The IPv6 Address data item MUST appear in the Peer Offer signal (Section 7.2), and MAY appear in the Peer Update (Section 7.5), Destination Up (Section 7.9) and Destination Update (Section 7.13) signals. When included in Destination signals, this data item contains the IPv6 address of the destination. In the Peer Offer signal, it contains the IPv6 address of the originating peer to be used to establish a DLEP session. In either case, the data item also contains an indication of whether this is a new or existing address, or is a deletion of a previously known address. When used in a Peer Offer signal the Add/Drop Indicator MUST be 1 (i.e. Add).

The IPv6 Address data item contains the following fields:
Data Item Type: TBD

Length: 17

Add/Drop: Value indicating whether this is a new or existing address (1), or a withdrawal of an address (0).

IPv6 Address: IPv6 Address of the destination or peer.

8.11. IPv4 Attached Subnet

The DLEP IPv4 Attached Subnet allows a device to declare that it has an IPv4 subnet (e.g., a stub network) attached. Once an IPv4 Subnet has been declared on a device, the declaration can NOT be withdrawn without terminating the destination (via the Destination Down signal) and re-issuing the Destination Up signal.

The DLEP IPv4 Attached Subnet data item contains the following fields:

<table>
<thead>
<tr>
<th>Data Item Type</th>
<th>Length = 5</th>
<th>IPv4 Attached Subnet</th>
<th>Subnet Mask</th>
</tr>
</thead>
</table>

Data Item Type: TBD

Length: 5

IPv4 Subnet: The IPv4 subnet reachable at the destination.
Subnet Mask: A subnet mask (0-32) to be applied to the IPv4 subnet.

8.12. IPv6 Attached Subnet

The DLEP IPv6 Attached Subnet allows a device to declare that it has an IPv6 subnet (e.g., a stub network) attached. As in the case of the IPv4 attached Subnet data item above, once an IPv6 attached subnet has been declared, it can NOT be withdrawn without terminating the destination (via Destination Down) and re-issuing the Destination Up signal.

The DLEP IPv6 Attached Subnet data item contains the following fields:

```
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  | Data Item Type| Length = 17   | IPv6 Attached Subnet          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |      IPv6 Attached Subnet                                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |      IPv6 Attached Subnet                                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |      IPv6 Attached Subnet                                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |      IPv6 Attached Subnet     | Subnet Mask   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Data Item Type: TBD

Length: 17

IPv4 Subnet: The IPv6 subnet reachable at the destination.

Subnet Mask: A subnet mask (0-128) to be applied to the IPv6 subnet.

8.13. Maximum Data Rate (Receive)

The Maximum Data Rate (Receive) (MDRR) data item MUST appear in the Peer Initialization ACK signal (Section 7.4), and MAY appear in the Peer Update (Section 7.5), Destination Up (Section 7.9) and Destination Update (Section 7.13) signals to indicate the maximum theoretical data rate, in bits per second, that can be achieved while receiving data on the link.

The Maximum Data Rate (Receive) data item contains the following fields:
Data Item Type: TBD
Length: 8

Maximum Data Rate (Receive): A 64-bit unsigned integer, representing the maximum theoretical data rate, in bits per second (bps), that can be achieved while receiving on the link.

8.14. Maximum Data Rate (Transmit)

The Maximum Data Rate (Transmit) (MDRT) data item MUST appear in the Peer Initialization ACK signal (Section 7.4), and MAY appear in the Peer Update (Section 7.5), Destination Up (Section 7.9) and Destination Update (Section 7.13) signals to indicate the maximum theoretical data rate, in bits per second, that can be achieved while transmitting data on the link.

The Maximum Data Rate (Transmit) data item contains the following fields:

Data Item Type: TBD
Length: 8

Maximum Data Rate (Transmit): A 64-bit unsigned integer, representing the maximum theoretical data rate, in bits per second (bps), that can be achieved while transmitting on the link.
8.15. Current Data Rate (Receive)

The Current Data Rate (Receive) (CDRR) data item MUST appear in the Peer Initialization ACK signal (Section 7.4), and MAY appear in the Peer Update (Section 7.5), Destination Up (Section 7.9), Destination Update (Section 7.13), Link Characteristics Request (Section 7.15) and Link Characteristics ACK (Section 7.16) signals to indicate the rate at which the link is currently operating for receiving traffic. When used in the Link Characteristics Request signal, CDRR represents the desired receive rate, in bits per second, on the link.

The Current Data Rate (Receive) data item contains the following fields:

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Data Item Type| Length = 8 | CDRR (bps)                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     CDRR (bps)                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     CDRR (bps)     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Data Item Type: TBD

Length: 8

Current Data Rate (Receive): A 64-bit unsigned integer, representing the current data rate, in bits per second, that is currently be achieved while receiving traffic on the link.

If there is no distinction between current and maximum receive data rates, current data rate receive MUST be set equal to the maximum data rate receive.

8.16. Current Data Rate (Transmit)

The Current Data Rate Receive (CDRT) data item MUST appear in the Peer Initialization ACK signal (Section 7.4), and MAY appear in the Peer Update (Section 7.5), Destination Up (Section 7.9), Destination Update (Section 7.13), Link Characteristics Request (Section 7.15) and Link Characteristics ACK (Section 7.16) signals to indicate the rate at which the link is currently operating for transmitting traffic. When used in the Link Characteristics Request signal, CDRT represents the desired transmit rate, in bits per second, on the link.
The Current Data Rate (Transmit) data item contains the following fields:

+----------------------------------+-
<table>
<thead>
<tr>
<th>Data Item Type</th>
<th>Length = 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Data Rate (Transmit)</td>
<td>64-bit</td>
</tr>
<tr>
<td></td>
<td>unsigned</td>
</tr>
<tr>
<td></td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>representing the current data rate, in bits per second, that is currently be achieved while transmitting traffic on the link.</td>
</tr>
<tr>
<td>If there is no distinction between current and maximum transmit data rates, current data rate transmit MUST be set equal to the maximum data rate transmit.</td>
<td></td>
</tr>
</tbody>
</table>

8.17. Latency

The Latency data item MUST appear in the Peer Initialization ACK signal (Section 7.4), and MAY appear in the Peer Update (Section 7.5), Destination Up (Section 7.9), Destination Update (Section 7.13), Link Characteristics Request (Section 7.15) and Link Characteristics ACK (Section 7.16) signals to indicate the amount of latency, in microseconds, on the link, or in the case of the Link Characteristics Request, to indicate the maximum latency required on the link.

The Latency value is reported as delay. The calculation of latency is implementation dependent. For example, the latency may be a running average calculated from the internal queuing.

+----------------------------------+-
<table>
<thead>
<tr>
<th>Data Item Type</th>
<th>Length = 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency in microseconds</td>
<td></td>
</tr>
<tr>
<td>Latency (cont.) microsecs</td>
<td></td>
</tr>
</tbody>
</table>

Data Item Type: TBD
Length: 4

Latency: A 32-bit unsigned value, representing the transmission delay that a packet encounters as it is transmitted over the link.

8.18. Resources (Receive)

The Resources (Receive) (RESR) data item MAY appear in the Peer Initialization ACK signal (Section 7.4), Peer Update (Section 7.5), Destination Up (Section 7.9), Destination Update (Section 7.13) and Link Characteristics ACK (Section 7.16) signals to indicate the amount of resources for reception (with 0 meaning ‘no resources available’, and 100 meaning ‘all resources available’) at the destination. The list of resources that might be considered is beyond the scope of this document, and is left to implementations to decide.

The Resources (Receive) data item contains the following fields:

```
| Data Item Type | Length = 1 | RESR |
```

Data Item Type: TBD
Length: 1

Resources (Receive): A percentage, 0-100, representing the amount of resources allocated to receiving data.

If a device cannot calculate RESR, this data item SHOULD NOT be issued.

8.19. Resources (Transmit)

The Resources (Receive) (RESR) data item MAY appear in the Peer Initialization ACK signal (Section 7.4), Peer Update (Section 7.5), Destination Up (Section 7.9), Destination Update (Section 7.13) and Link Characteristics ACK (Section 7.16) signals to indicate the amount of resources for transmission (with 0 meaning ‘no resources available’, and 100 meaning ‘all resources available’) at the destination. The list of resources that might be considered is beyond the scope of this document, and is left to implementations to decide.
The Resources (Transmit) data item contains the following fields:

```
0                   1                   2
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Data Item Type| Length = 1    |     REST      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Data Item Type: TBD
Length: 1

Resources (Transmit): A percentage, 0-100, representing the amount of resources allocated to transmitting data.

If a device cannot calculate REST, this data item SHOULD NOT be issued.

8.20. Relative Link Quality (Receive)

The Relative Link Quality (Receive) (RLQR) data item MAY appear in the Peer Initialization ACK signal (Section 7.4), Peer Update (Section 7.5), Destination Up (Section 7.9), Destination Update (Section 7.13) and Link Characteristics ACK (Section 7.16) signals to indicate the quality of the link for receiving data as calculated by the originating peer.

The Relative Link Quality (Receive) data item contains the following fields:

```
0                   1                   2
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Data Item Type| Length = 1    |     RLQR      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Data Item Type: TBD
Length: 1

Relative Link Quality (Receive): A non-dimensional integer, 1-100, representing relative link quality. A value of 100 represents a link of the highest quality.

If a device cannot calculate the RLQR, this data item SHOULD NOT be issued.
8.21. Relative Link Quality (Transmit)

The Relative Link Quality (Transmit) (RLQT) data item MAY appear in
the Peer Initialization ACK signal (Section 7.4), Peer Update
(Section 7.5), Destination Up (Section 7.9), Destination Update
(Section 7.13) and Link Characteristics ACK (Section 7.16) signals to
indicate the quality of the link for transmitting data as calculated
by the originating peer.

The Relative Link Quality (Transmit) data item contains the following
fields:

```
0                   1                   2
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Data Item Type| Length = 1    |     RLQT      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Data Item Type: TBD

Length: 1

Relative Link Quality (Transmit): A non-dimensional integer, 1-100,
representing relative link quality. A value of 100 represents a
link of the highest quality.

If a device cannot calculate the RLQT, this data item SHOULD NOT be
issued.

8.22. Link Characteristics ACK Timer

The Link Characteristics ACK Timer data item MAY appear in the Link
Characteristics Request signal (Section 7.15) to indicate the
desired number of seconds to the sender will wait for a response to
the request. If this data item is omitted, implementations
supporting the Link Characteristics Request SHOULD choose a default
value.

The Link Characteristics ACK Timer data item contains the following
fields:

```
0                   1                   2
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Data Item Type| Length = 1    | Interval      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Data Item Type: TBD
9. Credit-Windowing

DLEP includes an OPTIONAL credit-windowing scheme analogous to the one documented in [RFC5578]. In this scheme, traffic between the router and modem is treated as two unidirectional windows. This document identifies these windows as the ‘Modem Receive Window’, or MRW, and the ‘Router Receive Window’, or RRW.

If the OPTIONAL credit-windowing scheme is used, credits MUST be granted by the receiver on a given window – that is, on the ‘Modem Receive Window’ (MRW), the modem is responsible for granting credits to the router, allowing it (the router) to send data to the modem. Likewise, the router is responsible for granting credits on the RRW, which allows the modem to send data to the router.

DLEP expresses all credit data in number of octets. The total number of credits on a window, and the increment to add to a grant, are always expressed as a 64-bit unsigned integer quantity.

If used, credits are managed on a neighbor-specific basis; that is, separate credit counts are maintained for each neighbor requiring the service. Credits do not apply to the DLEP session that exists between routers and modems.

If a peer is able to support the OPTIONAL credit-windowing scheme then it MUST include a Extensions Supported data item (Section 8.6) including the value DLEP_EXT_CREDITS (value TBD) in the appropriate Peer Initialization or Peer Initialization ACK signal.

9.1. Credit-Windowing Signals

The credit-windowing scheme introduces no additional DLEP signals. However, if a peer has advertised during session initialization that it supports the credit-windowing scheme then the following DLEP signals may contain additional credit-windowing data items:

9.1.1. Destination Up Signal

The Destination Up signal MAY contain one of each of the following data items:
9.1.2. Destination Up ACK Signal

The Destination Up ACK signal MAY contain one of each of the following data items:

- Credit Window Status (Section 9.2.1)

9.1.3. Destination Update Signal

The Destination Update signal MAY contain one of each of the following data items:

- Credit Window Status (Section 9.2.1)
- Credit Grant (Section 9.2.2)
- Credit Request (Section 9.2.3)

9.2. Credit-Windowing Data Items

The credit-windowing scheme introduces 3 additional data items. If a peer has advertised during session initialization that it supports the credit-windowing scheme then it MUST correctly process the following data items without error.

<table>
<thead>
<tr>
<th>Data Item</th>
<th>Description</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBD</td>
<td>Credit Window Status</td>
<td>Section 9.2.1</td>
</tr>
<tr>
<td>TBD</td>
<td>Credit Grant</td>
<td>Section 9.2.2</td>
</tr>
<tr>
<td>TBD</td>
<td>Credit Request</td>
<td>Section 9.2.3</td>
</tr>
</tbody>
</table>

9.2.1. Credit Window Status

If the credit-window scheme is supported by the DLEP participants (both the router and the modem), the Credit Window Status data item MUST be sent by the participant receiving a Credit Grant for a given destination.

The Credit Window Status data item contains the following fields:
Data Item Type: TBD
Length: 16
Modem Receive Window Value: A 64-bit unsigned integer, indicating the current (or initial) number of credits available on the Modem Receive Window.

Router Receive Window Value: A 64-bit unsigned integer, indicating the current (or initial) number of credits available on the Router Receive Window.

9.2.2. Credit Grant

The Credit Grant data item is sent from a DLEP participant to grant an increment to credits on a window. The Credit Grant data item MAY appear in the Destination Up (Section 7.9) or Destination Update (Section 7.13) signals. The value in a Credit Grant data item represents an increment to be added to any existing credits available on the window. Upon successful receipt and processing of a Credit Grant data item, the receiver MUST respond with a signal containing a Credit Window Status data item to report the updated aggregate values for synchronization purposes.

In the Destination Up signal, when credits are desired, the originating peer MUST set the initial credit value of the window it controls (i.e., the Modem Receive Window, or Router Receive Window) to an initial, non-zero value. If the receiver of a Destination Up signal with a Credit Grant data item supports credits, the receiver MUST either reject the use of credits, via a Destination Up ACK response containing a Status data item (Section 8.2) with a status code of ‘Request Denied’, or set the initial value from the data contained in the Credit Window Status data item. If the initialization completes successfully, the receiver MUST respond to
the Destination Up signal with a Destination Up ACK signal that contains a Credit Window Status data item, initializing its receive window.

The Credit Grant data item contains the following fields:

```
+---------------------------------------------+-
| Data Item Type| Length | Credit Increment |
+---------------------------------------------+-
| Credit Increment | Credit Increment |
+---------------------------------------------+
```

Data Item Type: TBD
Length: 8
Reserved: A 64-bit unsigned integer representing the additional credits to be assigned to the credit window.

Since credits can only be granted by the receiver on a window, the applicable credit window (either the MRW or the RRW) is derived from the sender of the grant. The Credit Increment MUST NOT cause the window to overflow; if this condition occurs, implementations MUST set the credit window to the maximum value contained in a 64-bit quantity.

9.2.3. Credit Request

The Credit Request data item MAY be sent from either DLEP participant, via the Destination Update signal (Section 7.13), to indicate the desire for the partner to grant additional credits in order for data transfer to proceed on the session. If the corresponding Destination Up signal (Section 7.9) for this session did NOT contain a Credit Window Status data item, indicating that credits are to be used on the session, then the Credit Request data item MUST be rejected by the receiver via a Destination Update ACK signal containing a Status data item (Section 8.2) with status code 'Request Denied'.

The Credit Request data item contains the following fields:
Data Item Type: TBD
Length: 1
Reserved: This field is currently unused and MUST be set to 0.

10. Security Considerations

The protocol does not contain any mechanisms for security (e.g., authentication or encryption). The protocol assumes that any security would be implemented in the underlying transport (for example, by use of DTLS or some other mechanism), and is therefore outside the scope of this document.

11. IANA Considerations

This section specifies requests to IANA.

11.1. Registrations

This specification defines:

o A new repository for DLEP signals, with sixteen values currently assigned.

o Reservation of numbering space for Experimental DLEP signals.

o A new repository for DLEP data items, with twenty-three values currently assigned.

o Reservation of numbering space in the data items repository for experimental data items.

o A new repository for DLEP status codes.

o A new repository for DLEP extensions, with one value currently assigned.

o A request for allocation of a well-known port for DLEP communication.

No additional guidelines for expert review are anticipated.

11.3. Signal Type Registration

A new repository must be created with the values of the DLEP signals. All signal values are in the range [0..255].

Valid signals are:
- Peer Discovery
- Peer Offer
- Peer Initialization
- Peer Initialization ACK
- Peer Update
- Peer Update ACK
- Peer Termination
- Peer Termination ACK
- Destination Up
- Destination Up ACK
- Destination Down
- Destination Down ACK
- Destination Update
- Heartbeat
- Link Characteristics Request
- Link Characteristics ACK
It is also requested that the repository contain space for experimental signal types.

11.4. DLEP Data Item Registrations

A new repository for DLEP data items must be created.

All data item values are in the range [0..255].

Valid data items are:
- DLEP Version
- Status
- DLEP Port
- Peer Type
- Heartbeat Interval
- Extensions Supported
- Experimental Definition
- MAC Address
- IPv4 Address
- IPv6 Address
- IPv4 Attached Subnet
- IPv6 Attached Subnet
- Maximum Data Rate (Receive)
- Maximum Data Rate (Transmit)
- Current Data Rate (Receive)
- Current Data Rate (Transmit)
- Latency
- Resources (Receive)
- Resources (Transmit)
o Relative Link Quality (Receive)

o Relative Link Quality (Transmit)

o Link Characteristics ACK Timer

o Credit Window Status

o Credit Grant

o Credit Request

It is also requested that the registry allocation contain space for experimental data items.

11.5. DLEP Status Code Registrations

A new repository for DLEP status codes must be created.

All status codes are in the range [0..255].

Valid status codes are:

o Success (value 0)

o Unknown Signal

o Invalid Signal

o Unexpected Signal

o Request Denied

o Timed Out

11.6. DLEP Extensions Registrations

A new repository for DLEP extensions must be created.

All extension values are in the range [0..255].

Valid extensions are:

o DLEP_EXT_CREDITS - Credit windowing
11.7. DLEP Well-known Port

It is requested that IANA allocate a well-known port number for DLEP communication.

11.8. DLEP Multicast Address

It is requested that IANA allocate a multicast address for DLEP discovery signals.

12. Acknowledgements

The authors would like to acknowledge and thank the members of the DLEP design team, who have provided invaluable insight. The members of the design team are: Teco Boot, Bow-Nan Cheng, John Dowdell, and Henning Rogge.

The authors would also like to acknowledge the influence and contributions of Greg Harrison, Chris Olsen, Martin Duke, Subir Das, Jaewon Kang, Vikram Kaul, and Nelson Powell.

13. References

13.1. Normative References


13.2. Informative References


Appendix A. Peer Level Signal Flows

_NB_ The following diagrams are possibly out of date. If there is a discrepancy with the text, then the text is correct.

A.1. Router Device Restarts Discovery
Router                       Modem    Signal Description
====================================================================
--------Peer Discovery-------->       Router initiates discovery

<--------Peer Offer------------ Modem detects a problem, sends
   w/ Non-zero Status TLV   Peer Offer w/Status TLV indicating
                           the error.
                           Router accepts failure, restarts
discovery process.

--------Peer Discovery-------->       Router initiates discovery

<--------Peer Offer------------ Modem accepts, sends Peer Offer
   w/Zero Status TLV indicating   w/Zero Status TLV indicating
                                   success.
                                   Discovery completed.

A.2. Router Device Detects Peer Offer Timeout

Router                       Modem    Signal Description
====================================================================
--------Peer Discovery-------->       Router initiates discovery, starts
                                   a guard timer.
                                   Router guard timer expires. Router
                                   restarts discovery process.

--------Peer Discovery-------->       Router initiates discovery, starts
                                   a guard timer.

<--------Peer Offer------------ Modem accepts, sends Peer Offer
   w/Zero Status TLV indicating   w/Zero Status TLV indicating
                                   success.
                                   Discovery completed.

A.3. Router Peer Offer Lost
### A.4. Discovery Success

<table>
<thead>
<tr>
<th>Router</th>
<th>Modem</th>
<th>Signal Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;-------Peer Discovery------&gt;</td>
<td>Modem initiates discovery</td>
<td></td>
</tr>
<tr>
<td>--------Peer Offer---------&gt;</td>
<td>Router offers availability</td>
<td></td>
</tr>
<tr>
<td>&lt;-----Peer Initialization------&gt;</td>
<td>Modem Connects on TCP Port</td>
<td></td>
</tr>
<tr>
<td>&lt;------Peer Heartbeat----------&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------Peer Heartbeat---------&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;================&gt;</td>
<td>Signal flow about destinations (i.e. Destination Up, Destination Down, Destination update)</td>
<td></td>
</tr>
<tr>
<td>&lt;------Peer Heartbeat---------&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------Peer Heartbeat---------&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;--------Peer Term Req--------&gt;</td>
<td>Terminate Request</td>
<td></td>
</tr>
<tr>
<td>&lt;-------Peer Term Res---------&gt;</td>
<td>Terminate Response</td>
<td></td>
</tr>
</tbody>
</table>
### A.5. Router Detects a Heartbeat timeout

<table>
<thead>
<tr>
<th>Router</th>
<th>Modem</th>
<th>Signal Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;-------Peer Heartbeat----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------Peer Heartbeat-----------&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>---Peer Heartbeat-------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>..................................................................................................................................................</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------Peer Heartbeat-----------</td>
<td></td>
<td>Router Heartbeat Timer expires, detects missing heartbeats. Router takes down all destination sessions and terminates the Peer association.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>---Peer Heartbeat-------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------Peer Terminate------------</td>
<td></td>
<td>Peer Terminate Request</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;-----Peer Terminate ACK--------</td>
<td></td>
<td>Peer Terminate ACK</td>
</tr>
</tbody>
</table>

### A.6. Modem Detects a Heartbeat timeout
Router                      Modem    Signal Description
====================================================================

<------Peer Heartbeat-------

<------Peer Heartbeat-------|

<------Peer Heartbeat-------

<------Peer Heartbeat-------|

<------Peer Heartbeat-------

    Modem Heartbeat Timer expires, detects missing heartbeats. Modem takes down all destination sessions

<------Peer Terminate-------

Peer Terminate Request

--------Peer Terminate------->

Router takes down all destination sessions, then acknowledges the Peer Terminate

------Peer Terminate ACK----->

Peer Terminate ACK

A.7. Peer Terminate (from Modem) Lost

| |------Peer Terminate-------

Modem Peer Terminate Request

Router Heartbeat times out, terminates association.

------Peer Terminate------->

Router Peer Terminate

<------Peer Terminate ACK------

Modem sends Peer Terminate ACK

A.8. Peer Terminate (from Router) Lost


Router                       Modem    Signal Description
====================================================================
-------Peer Terminate-------->        Router Peer Terminate Request
                                   Modem HB times out, terminates association.
<-------Peer Terminate--------         Modem Peer Terminate
-------Peer Terminate ACK----->        Peer Terminate ACK

Appendix B. Destination Specific Signal Flows

B.1. Modem Destination Up Lost

Router                       Modem    Signal Description
====================================================================
||-----Destination Up ------------   Modem sends Destination Up
                                   Modem timesout on ACK
<-----Destination Up ------------   Modem sends Destination Up
-------Destination Up ACK--------->   Router accepts the destination session
<-----Destination Update---------    Modem Destination Metrics
                                   . . . . . .
<-----Destination Update---------    Modem Destination Metrics

B.2. Router Detects Duplicate Destination Ups
Router                       Modem    Signal Description
====================================================================
<-----Destination Up ------- Modem sends Destination Up
-----Destination Up ACK------| Router accepts the destination session
                                 Modem timeout on ACK
<-----Destination Up ------- Modem resends Destination Up
                                 Router detects duplicate Destination, takes down the previous, accepts the new Destination.
-----Destination Up ACK-------> Router accepts the destination session
<-----Destination Update------- Modem Destination Metrics
                                    ...........
<-----Destination Update------- Modem Destination Metrics
B.3. Destination Up, No Layer 3 Addresses
Router                       Modem    Signal Description
====================================================================
<-----Destination Up ------- Modem sends Destination Up
-----Destination Up ACK------> Router accepts the destination session
                                 Router ARPs for IPv4 if defined. Router drives ND for IPv6 if defined.
<-----Destination Update------- Modem Destination Metrics
                                    ...........
<-----Destination Update------- Modem Destination Metrics
B.4. Destination Up with IPv4, No IPv6
<-----Destination Up ----------- Modem sends Destination Up with the IPv4 TLV

-----Destination Up ACK--------> Router accepts the destination session

Router drives ND for IPv6 if defined.

<-----Destination Update--------- Modem Destination Metrics
<-----Destination Update--------- Modem Destination Metrics

B.5.  Destination Up with IPv4 and IPv6

<-----Destination Up ----------- Modem sends Destination Up with the IPv4 and IPv6 TLVs

-----Destination Up ACK--------> Router accepts the destination session

<-----Destination Update--------- Modem Destination Metrics

B.6.  Destination Session Success
Router                       Modem    Signal Description
====================================================================
--------Peer Offer------------->       Router offers availability
--------Peer Heartbeat--------->

<------Destination Up -----------     Modem
--------Destination Up ACK-------->     Router
<------Destination Update---------    Modem

Modem initiates the terminate
<------Destination Down ----------    Modem
--------Destination Down ACK------->    Router
 or
 awkwardly phrased
--------Destination Down ---------->    Router
<------Destination Down ACK-------    Modem

Authors’ Addresses

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

Email: sratliff@idirect.net

Bo Berry
Shawn Jury
Cisco Systems
170 West Tasman Drive
San Jose, CA 95134
USA
Email: sjury@cisco.com

Darryl Satterwhite
Broadcom
Email: dsatterw@broadcom.com

Rick Taylor
Airbus Defence & Space
Quadrant House
Celtic Springs
Coedkernew
Newport NP10 8FZ
UK
Email: rick.taylor@airbus.com