Mobile Ad hoc Networking (MANET)
Internet-Draft
Expires: December 28, 2006

T. Clausen
LIX, Ecole Polytechnique, France
C. Dearlove
BAE Systems Advanced Technology Centre
P. Jacquet
Project Hipercom, INRIA
The OLSRv2 Design Team
MANET Working Group
June 26, 2006

The Optimized Link-State Routing Protocol version 2
draft-ietf-manet-olsrv2-02

Status of this Memo

Copyright Notice

Abstract

This document describes version 2 of the Optimized Link State Routing (OLSRv2) protocol for mobile ad hoc networks. The protocol embodies

an optimization of the classical link state algorithm tailored to the requirements of a mobile wireless LAN.

The key optimization of OLSRv2 is that of multipoint relays, providing an efficient mechanism for network-wide broadcast of link-state information (i.e. reducing the cost of performing a network-wide link-state broadcast). A secondary optimization is that OLSRv2 employs partial link-state information: each node maintains information about all destinations, but only a subset of links. This allows that only select nodes diffuse link-state advertisements (i.e. reduces the number of network-wide link-state broadcasts) and that these advertisements contain only a subset of links (i.e. reduces the size of each network-wide link-state broadcast). The partial link-state information thus obtained still allows each OLSRv2 node to at all times maintain optimal (in terms of number of hops) routes to all destinations in the network.

OLSRv2 imposes minimum requirements to the network by not requiring sequenced or reliable transmission of control traffic. Furthermore, the only interaction between OLSRv2 and the IP stack is routing table management.

OLSRv2 is particularly suitable for large and dense networks as the technique of MPRs works well in this context.

Table of Contents

1. Introduction

The Optimized Link State Routing protocol version 2 (OLSRv2) is an update to OLSRv1 as published in RFC3626 [1]. Compared to RFC3626, OLSRv2 retains the same basic mechanisms and algorithms, while providing an even more flexible signaling framework and some simplification of the messages being exchanged. Also, OLSRv2 takes care to accommodate both IPv4 and IPv6 addresses in a compact fashion.

OLSRv2 is developed for mobile ad hoc networks. It operates as a table driven, proactive protocol, i.e. it exchanges topology information with other nodes of the network regularly. Each node selects a set of its neighbor nodes as "MultiPoint Relays" (MPRs). Only nodes that are selected as such MPRs are then responsible for forwarding control traffic intended for diffusion into the entire network. MPRs provide an efficient mechanism for flooding control traffic by reducing the number of transmissions required.

Nodes selected as MPRs also have a special responsibility when declaring link state information in the network. Indeed, the only requirement for OLSRv2 to provide shortest path routes to all destinations is that MPR nodes declare link-state information for their MPR selectors. Additional available link-state information may be utilized, e.g. for redundancy.

Nodes which have been selected as multipoint relays by some neighbor node(s) announce this information periodically in their control messages. Thereby a node announces to the network that it has reachability to the nodes which have selected it as an MPR. Thus, as well as being used to facilitate efficient flooding, MPRs are also used for route calculation from any given node to any destination in the network.

A node selects MPRs from among its one hop neighbors with "symmetric", i.e. bi-directional, linkages. Therefore, selecting routes through MPRs automatically avoids the problems associated with data packet transfer over uni-directional links (such as the problem of not getting link-layer acknowledgments for data packets at each hop, for link-layers employing this technique for unicast traffic).

OLSRv2 is developed to work independently from other protocols. Likewise, OLSRv2 makes no assumptions about the underlying link-layer. However, OLSRv2 may use link-layer information and notifications when available and applicable.

OLSRv2, as OLSRv1, inherits the concept of forwarding and relaying from HIPERLAN (a MAC layer protocol) which is standardized by ETSI

[6].

1.1. Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC2119 [2].

MANET specific terminology is to be interpreted as described in [3] and [4].

Additionally, this document uses the following terminology:

node - A MANET router which implements the Optimized Link State Routing protocol version 2 as specified in this document.

OLSRv2 interface - A MANET interface, running OLSRv2.

symmetric strict 2-hop neighbor - A symmetric 2-hop neighbor which is not a symmetric 1-hop neighbor and is not a 2-hop neighbor only through a symmetric 1-hop neighbor with willingness WILL_NEVER. (If node Z is a symmetric 2-hop neighbor of node X then there is a node Y such that node Z is a symmetric 1-hop neighbor of node Y and node Y is a symmetric 1-hop neighbor of node X. If node Z is a symmetric strict 2-hop neighbor of node X then there is such a node Y with willingness which is not WILL_NEVER.)

symmetric strict 2-hop neighborhood - The set of the symmetric strict 2-hop neighbors of node X.

multipoint relay (MPR) - A node which is selected by its symmetric 1-hop neighbor, node X, to "re-transmit" all the broadcast messages that it receives from node X, provided that the message is not a duplicate, and that the hop limit field of the message is greater than one.

MPR selector - A node which has selected its symmetric 1-hop neighbor, node X, as one of its MPRs is an MPR selector of node X.

1.2. Applicability Statement

OLSRv2 is a proactive routing protocol for mobile ad hoc networks (MANETs) [7], [8]. It is well suited to large and dense networks of mobile nodes, as the optimization achieved using the MPRs works well in this context. The larger and more dense a network, the more optimization can be achieved as compared to the classic link state algorithm. OLSRv2 uses hop-by-hop routing, i.e. each node uses its local information to route packets.

As OLSRv2 continuously maintains routes to all destinations in the network, the protocol is beneficial for traffic patterns where the traffic is random and sporadic between a large subset of nodes, and where the [source, destination] pairs are changing over time: no additional control traffic need be generated in this situation since routes are maintained for all known destinations at all times. Also, since routes are maintained continuously, traffic is subject to no delays due to buffering/route-discovery. This continued route maintenance may be done using periodic message exchange, as detailed in this specification, or triggered by external events if available.

OLSRv2 supports nodes which have multiple interfaces which participate in the MANET. OLSRv2, additionally, supports nodes which have non-MANET interfaces which can serve as (if configured to do so) gateways towards other networks.

The message exchange format, contained in previous versions of this specification, has been factored out to an independent specification [3], which is used for carrying OLSRv2 control signals. OLSRv2 is thereby able to allow for extensions via "external" and "internal" extensibility. External extensibility implies that a protocol extension may specify and exchange new message types, formatted according to [3], which can be forwarded and delivered correctly. Internal extensibility implies that a protocol extension may define additional attributes to be carried embedded in the standard OLSRv2 control messages detailed in this specification, using the TLV mechanism specified in [3], while OLSRv2 control messages with additional attributes can still be correctly understood by all OLSRv2 nodes.

The OLSRv2 neighborhood discovery protocol using HELLO messages has likewise been factored out to an independent specification [4]. This neighborhood discovery protocol serves to ensure that each OLSRv2 node has available continuously updated information repositories describing the node's 1-hop and 2-hop neighbors. [4] uses the message format specified in [3], and hence is extensible as described above.

2. Protocol Overview and Functioning

OLSRv2 is a proactive routing protocol for mobile ad hoc networks. The protocol inherits the stability of a link state algorithm and has the advantage of having routes immediately available when needed due to its proactive nature. OLSRv2 is an optimization over the classical link state protocol, tailored for mobile ad hoc networks. The main tailoring and optimizations of OLSRv2 are:

- periodic, unacknowledged transmission of all control messages;
- optimized flooding for global link-state information diffusion;
- partial topology maintenance - each node knows only a subset of the links in the network, sufficient for a minimum hop route to all destinations.

Using the message exchange format [3] and the neighborhood discovery protocol [4], OLSRv2 also contains the following main components:

- a TLV, to be included within the HELLO messages of [4], allowing a node to signal MPR selection;
- an optimized flooding mechanism for global information exchange, denoted "MPR flooding";
- a specification of global signaling, denoted TC (Topology Control) messages. TC messages in OLSRv2 serve to:
  - inject link-state information into the entire network;
  - inject addresses of hosts and networks for which they may serve as a gateway into the entire network.

  TC messages are emitted periodically, thereby allowing nodes to continuously track global changes in the network.

The use of [4] allows a node to continuously track changes to its local topology up to two hops away. This allows a node to make provisions for ensuring optimized flooding, denoted "MPR flooding", as well as injection of link-state information into the network. This is done through the notion of MultiPoint Relays (MPRs).

The idea of MPRs is to minimize the overhead of flooding messages in the network by reducing redundant retransmissions of messages in the same region. Each node in the network selects an MPR Set, a set of nodes in its symmetric 1-hop neighborhood which may retransmit its messages. The 1-hop neighbors of a node which are not in its MPR set

receive and process broadcast messages, but do not retransmit broadcast messages received from that node. The MPR Set of a node X may be any subset of its symmetric 1-hop neighborhood such that every node in its symmetric strict 2-hop neighborhood has a symmetric link to a node in the MPR Set of node X. The MPR Set of a node may thus be said to "cover" the node's symmetric strict 2-hop neighborhood. The smaller a MPR Set, the fewer times messages are forwarded and the less resulting control traffic overhead. [8] gives an analysis and example of MPR selection algorithms. Note that as long as the condition above is satisfied, any algorithm selecting MPR Sets is acceptable in terms of implementation interoperability.

Each node maintains information about the set of symmetric 1-hop neighbors that have selected it as MPR. This set is called the MPR Selector Set of the node. A node obtains this information from an MPR TLV which is inserted into the HELLO message exchange of [4].

Each node also maintains a Relay Set, which is the set of nodes for which a node is to relay broadcast traffic. The Relay Set is derived from the MPR Selector Set in that the Relay Set MUST contain all the nodes in the MPR Selector set and MAY contain additional nodes.

Using the MPR flooding mechanism, link-state information can be injected into the network. For this purpose, a node maintains an Advertised Neighbor Set which MUST contain all the nodes in the MPR selector set and MAY contain additional nodes. If the Advertised Neighbor Set of a node is non-empty, it is reported in TC messages generated by that node. If the Advertised Neighbor Set is empty, TC messages are not generated by that node, unless needed for gateway reporting, or for a short period to accelerate the removal of unwanted links.

OLSRv2 is designed to work in a completely distributed manner and does not depend on any central entity. The protocol does not require reliable transmission of control messages: each node sends control messages periodically, and can therefore sustain a reasonable loss of some such messages. Such losses may occur frequently in radio networks due to collisions or other transmission problems.

OLSRv2 does not require sequenced delivery of messages. Each control message contains a sequence number which is incremented for each message. Thus the recipient of a control message can, if required, easily identify which information is more recent - even if messages have been re-ordered while in transmission.

OLSRv2 does not require any changes to the format of IP packets, any existing IP stack can be used as is: OLSRv2 only interacts with routing table management. OLSR sends its control messages using UDP.

2.1. Protocol Extensibility

OLSRv2 uses the neighborhood discovery mechanism of [4], and specifies additionally one message type, TC, and a number of TLVs. All messages exchanged by [4] and by OLSRv2 use and comply with the extensible message exchange format of [3], thus OLSR provides both "external" extensibility (addition of new message types as in OLSRv1 [1]) and "internal" extensibility (addition of information to existing messages through TLVs) as described in [3].

Those nodes which do not recognize a new message type ("external extensibility") will ignore this message type for processing, but will correctly forward the message, if specified in the message header. Those nodes which do not recognize a newly defined TLV type ignore the added TLV, but process (if the message type is recognized) the message correctly, as well as forwards the message, if specified in the message header.

3. Processing and Forwarding Repositories

The following data structures are employed in order to ensure that a message is processed at most once and is forwarded at most once per interface of a node, and that fragmented content is treated correctly.

3.1. Received Set

Each node maintains, for each OLSRv2 interface, a set of signatures of messages, which have been received over that interface, in the form of "Received Tuples":

(RX_type, RX_orig_addr, RX_seq_number, RX_time)

where:

RX_type is the received message type, or zero if the received message sequence number is not type-specific;

RX_orig_addr is the originator address of the received message;

RX_seq_number is the message sequence number of the received message;

RX_time specifies the time at which this Received Tuple expires and *MUST* be removed.

In a node, this is denoted the "Received Set" for that interface.

3.2. Fragment Set

Each node stores messages containing fragmented content until all fragments are received and the message processing can be completed, in the form of "Fragment Tuples":

(FG_message, FG_time)

where:

FG_message is the message containing fragmented content;

FG_time specifies the time at which this Fragment Tuple expires and MUST be removed.

In a node, this is denoted the "Fragment Set".

3.3. Processed Set

Each node maintains a set of signatures of messages which have been processed by the node, in the form of "Processed Tuples":

(P_type, P_orig_addr, P_seq_number, P_time)

where:

P_type is the processed message type, or zero if the processed message sequence number is not type-specific;

P_orig_addr is the originator address of the processed message;

P_seq_number is the message sequence number of the processed message;

P_time specifies the time at which this Processed Tuple expires and *MUST* be removed.

In a node, this is denoted the "Processed Set".

3.4. Forwarded Set

Each node maintains a set of signatures of messages which have been retransmitted/forwarded by the node, in the form of "Forwarded Tuples":

(FW_type, FW_orig_addr, FW_seq_number, FW_time)

where:

FW_type is the forwarded message type, or zero if the forwarded message sequence number is not type-specific;

FW_orig_addr is the originator address of the forwarded message;

FW_seq_number is the message sequence number of the forwarded message;

FW_time specifies the time at which this Forward Tuple expires and *MUST* be removed.

In a node, this is denoted the "Forwarded Set".

3.5. Relay Set

Each node maintains a set of neighbor interface addresses for which it is to relay flooded messages, in the form of "Relay Tuples":

(RY_iface_addr)

where:

RY_iface_addr is the address of a neighbor interface for which the node SHOULD relay flooded messages. This MUST include a prefix length.

In a node, this is denoted the "Relay Set".

4. Packet Processing and Message Forwarding

On receiving a basic packet, as defined in [3], a node examines each of the message headers. If the message type is known to the node, the message is processed locally according to the specifications for that message type. The message is also independently evaluated for forwarding.

4.1. Actions when Receiving an OLSRv2 Packet

On receiving a packet, a node MUST perform the following tasks:

1. The packet may be fully parsed on reception, or the packet and its messages may be parsed only as required. (It is possible to parse the packet header, or determine its absence, without parsing any messages. It is possible to divide the packet into messages without even fully parsing their headers. It is possible to determine whether a message is to be forwarded, and to forward it, without parsing its body. It is possible to determine whether a message is to be processed without parsing its body. It is possible to determine if that processing may be delayed because the message is a fragment by inspecting the first few octets of the message body without fully parsing it.)

2. If parsing fails at any point the relevant entity (packet or message) MUST be silently discarded, other parts of the packet (up to the whole packet) MAY be silently discarded;

3. Otherwise if the packet header is present and it contains a packet TLV block, then each TLV in it is processed according to its type;

4. Otherwise each message in the packet, if any, is treated according to Section 4.2.

4.2. Actions when Receiving an OLSRv2 Message

A node MUST perform the following tasks for each received OLSRv2 message:

1. If the received OLSRv2 message header cannot be correctly parsed according to the specification in [3], or if the node recognizes from the originator address of the message that the message is one which the receiving node itself originated, then the message MUST be silently discarded;

2. Otherwise:

1. If the received message is of a known type then the message is considered for processing according to Section 4.3, AND;

2. If for the received message (<hop-limit> + <hop-count>) > 1, then the message is considered for forwarding according to Section 4.4.

4.3. Message Considered for Processing

If a message (the "current message") is considered for processing, the following tasks MUST be performed:

1. If an entry exists in the Processed Set where:

   * P_type == the message type of the current message, or 0 if the typedep bit in the message semantics octet (in the message header) of the current message is cleared ('0'), AND;

   * P_orig_addr == the originator address of the current message, AND;

   * P_seq_number == the message sequence number of the current message.

   then the current message MUST NOT be processed.

2. Otherwise:

   1. Create an entry in the Processed Set with:

      + P_type = the message type of the current message, or 0 if the typedep bit in the message semantics octet (in the message header) of the current message is cleared ('0');

      + P_orig_addr = originator address of the current message;

      + P_seq_number = sequence number of the current message;

      + P_time = current time + P_HOLD_TIME.

   2. If the current message does not contain a valid message TLV with Type == FRAGMENTATION (or if it does and the indicated number of fragments is one) then process the message fully according to its type.

   3. Otherwise:

1. If the current message does not contain a valid message TLV with Type == CONT_SEQ_NUM then the current message MUST be silently discarded;

2. Otherwise if the current message is "partially or wholly self-contained", as indicated by the notselfcont bit in the Value field of the TLV with Type == FRAGMENTATION being cleared ('0'), then process the current message as far as possible according to its type;

3. If the Fragment Set includes any Fragment Tuples with:

   - either the typedepseq bit in the Value field of the TLV with Type == FRAGMENTATION in the current message is cleared ('0') OR message type of FG_message == message type of current message, AND;

   - originator address of FG_message == originator address of current message, AND;

   - content sequence number (the Value field of the message TLV with Type == CONT_SEQ_NUM) of FG_message == content sequence number of current message; AND

   - either fragment number (from the Value field of the TLV with Type == FRAGMENTATION) in FG_message == fragment number of current message OR number of fragments (from the Value field of the TLV with Type == FRAGMENTATION) of FG_message != number of fragments of current message;

   then remove these Fragment Tuples from the Fragment Set;

4. If the Fragment Set includes Fragment Tuples containing all the remaining fragments of the same overall message as the current message, i.e. if the number of Fragment Tuples such that:

   - either the typedepseq bit in the Value field of the TLV with Type == FRAGMENTATION in the current message is cleared ('0') OR message type of FG_message == message type of current message, AND;

   - originator address of FG_message == originator address of current message, AND;

   - content sequence number (the Value field of the message TLV with Type == CONT_SEQ_NUM) of FG_message

== content sequence number of current message

is equal to (number of fragments of current message, less one) then all of these Fragment Tuples are removed from the Fragment Set and their messages processed according to their type (taking account of any previous processing of any which are partially or wholly self-contained);

5. Otherwise, a Fragment Tuple is added to the Fragment Set with

   - FG_message = current message;

   - FG_time = current time + FG_HOLD_TIME;

   possibly replacing a previously received instance of the same fragment.

4.4. Message Considered for Forwarding

If a message is considered for forwarding, and it is either of a message type defined in this document or of an unknown message type, then it MUST use the following algorithm. A message type not defined in this document MAY specify the use of this, or another algorithm. (Such an other algorithm MAY use the Received Set for the receiving interface, it SHOULD use the Forwarded Set similarly to the following algorithm.)

If a message is considered for forwarding according to this algorithm, the following tasks MUST be performed:

1. If the sending interface (as indicated by the source interface of the IP datagram containing the message) does not match (taking into account any address prefix of) any N_neighbor_iface_addr in any Symmetric Neighbor Tuple, then the message MUST be silently discarded.

2. Otherwise:

   1. If an entry exists in the Received Set for the receiving interface, where:

      + RX_type == the message type, or 0 if the typedep bit in the message semantics octet (in the message header) is cleared ('0'), AND;

      + RX_orig_addr == the originator address of the received message, AND;

+ RX_seq_number == the sequence number of the received message.

then the message MUST be silently discarded.

2. Otherwise:

1. Create an entry in the Received Set for the receiving interface with:

- RX_type = the message type, or 0 if the typedep bit in the message semantics octet (in the message header) is cleared ('0');

- RX_orig_addr = originator address of the message;

- RX_seq_number = sequence number of the message;

- RX_time = current time + RX_HOLD_TIME.

2. If an entry exists in the Forwarded Set where:

- FW_type == the message type, or 0 if the typedep bit in the message semantics octet (in the message header) is cleared ('0');

- FW_orig_addr == the originator address of the received message, AND;

- FW_seq_number == the sequence number of the received message.

then the message MUST be silently discarded.

3. Otherwise if a Relay Tuple exists whose RY_iface_addr matches (taking into account any address prefix) the sending interface (as indicated by the source interface of the IP datagram containing the message):

1. Create an entry in the Forwarded Set with:

o FW_type = the message type, or 0 if the typedep bit in the message semantics octet (in the message header) is cleared ('0');

o FW_orig_addr = originator address of the message;

- FW_seq_number = sequence number of the message;
- FW_time = current time + FW_HOLD_TIME.

2. The message header is modified as follows:
   - Decrement <hop-limit> in the message header by 1;
   - Increment <hop-count> in the message header by 1;
3. Transmit the message on all OLSRv2 interfaces of the node.

Messages are retransmitted in the format specified by [3] with the ALL-MANET-NEIGHBORS address (see Section 16.1) as destination IP address.

5. Information Repositories

The purpose of OLSRv2 is to determine the Routing Set, which may be used to update IP's Routing Table, providing "next hop" routing information for IP datagrams. In order to accomplish this, OLSRv2 uses a number of protocol sets: the Neighborhood Information Base, provided by [4], is in OLSRv2 augmented by information allowing MPR selection and signaling. Additionally, OLSRv2 specifies a Topology Information Base, which describes the information used for and acquired through TC message exchange - in other words: the topology base represents the network topology graph as seen from each node.

Addresses (other than originator addresses) recorded in the Neighborhood Information Base and the Topology Information Base MUST all be recorded with prefix lengths, in order to allow comparison with addresses received in HELLO and TC messages. For the Topology Information Base this applies to A_neighbor_iface_addr, T_dest_iface_addr, T_last_iface_addr, AN_net_addr, AN_gw_iface_addr, R_dest_addr, R_dest_addr, R_next_iface_addr and R_local_iface_addr, but not AH_orig_addr. For the Neighborhood Information Base see [4].

5.1. Neighborhood Information Base

The neighborhood information base stores information about links between local interfaces and interfaces on adjacent nodes. In addition to the sets described in [4], OLSRv2 adds an element to each Link Tuple to allow a node to record the willingness of a 1-hop neighbor node to be selected as an MPR. Also, OLSRv2 adds an MPR Set and an MPR Selector Set to the Neighborhood Information Base. The MPR Set is used by a node to record which of its symmetric 1-hop neighbors are selected as MPRs, and the MPR Selector Set is used by a node to record which of its symmetric 1-hop neighbors have selected it as MPR. Thus the MPR Set is used, in addition to what is specified in [4], when generating HELLO messages, and the MPR Selector Set is populated, in addition to what is specified in [4] when processing HELLO messages.

5.1.1. Link Set

The Link Tuples, specified in [4] are augmented by an element, L_willingness:

L_willingness is the node's willingness to be selected as an MPR;

The remaining elements of the Link Tuples are as specified in [4].

5.1.2. MPR Set

A node maintains a set of all of the OLSRv2 interface addresses with which the node has a symmetric link and which are of 1-hop symmetric neighbors which the node has selected as MPRs. This is denoted the "MPR Set".

5.1.3. MPR Selector Set

A node maintains a set of "MPR Selector Tuples" containing all of the OLSRv2 interface addresses with which the node has a symmetric link and which are of 1-hop symmetric neighbors which have selected the node as an MPR.

(MS_neighbor_iface_addr, MS_time)

MS_neighbor_iface_addr specifies an OLSRv2 interface address with which the node has a symmetric link and which is of a 1-hop symmetric neighbor which has selected the node as an MPR;

MS_time specifies the time at which this MPR Selector Tuple expires and *MUST* be removed.

In a node, the set of MPR Selector Tuples is denoted the "MPR Selector Set".

5.2. Topology Information Base

The Topology Information Base stores information, required for the generation and processing of TC messages. The Advertised Neighbor Set contains OLSRv2 interface addresses of symmetric 1-hop neighbors which are to be reported in TC messages. The Topology Set and Attached Network Set both record information received through TC messages. Thus the Advertised Neighbor Set is used for generating TC messages, while the Topology Set and Attached Network Set are populated when processing TC messages.

Additionally, a Routing Set is maintained, derived from the information recorded in the Neighborhood Information Base, Topology Set and Attached Network Set.

5.2.1. Advertised Neighbor Set

A node maintains a set of OLSRv2 interface addresses of symmetric 1-hop neighbors, which are to be advertised through TC messages:

(A_neighbor_iface_addr)

For this set, an Advertised Neighbor Set Sequence Number (ANSN) is maintained. Each time the Advertised Neighbor Set is updated, the ANSN MUST be incremented. The ANSN MUST also be incremented if any locally advertised attached networks are added or removed.

5.2.2. ANSN History Set

A node records a set of "ANSN History Tuples", recording information about the freshness of the topology information received from each other node:

(AH_orig_addr, AH_seq_number, AH_time)

AH_orig_addr is the originator address of a received TC message;

AH_seq_number is the highest ANSN in any TC message received which originated from AH_orig_addr;

AH_time is the time at which this ANSN History Tuple expires and *MUST* be removed.

In a node, the set of ANSN History Tuples is denoted the "ANSN History Set".

5.2.3. Topology Set

Each node in the network maintains topology information about the network in the form of "Topology Tuples":

(T_dest_iface_addr, T_last_iface_addr, T_seq_number, T_time)

T_dest_iface_addr is an OLSRv2 interface address of a destination node, which may be reached in one hop from the node with the OLSRv2 interface address T_last_iface_addr;

T_last_iface_addr is, conversely, an OLSRv2 interface address of a node which is the last hop on a path towards the node with OLSRv2 interface address T_dest_iface_addr. Typically, the node with OLSRv2 interface address T_last_iface_addr is a MPR of the node with OLSRv2 interface address T_dest_iface_addr;

T_seq_number is the highest received ANSN associated with the information contained in this Topology Tuple;

T_time specifies the time at which this Topology Tuple expires and *MUST* be removed.

In a node, the set of Topology Tuples is denoted the "Topology Set".

5.2.4. Attached Network Set

Each node in the network maintains information about attached networks in the form of "Attached Network Tuples":

(AN_net_addr, AN_gw_iface_addr, AN_seq_number, AN_time)

AN_net_addr is the network address (including prefix length) of an attached network, which may be reached via the node with the OLSRv2 interface address AN_gw_iface_addr;

AN_gw_iface_addr is the address of an OLSRv2 interface of a node which can act as gateway to the network identified by AN_net_addr;

AN_seq_number is the highest received ANSN associated with the information contained in this Attached Network Tuple;

AN_time specifies the time at which this Attached Network Tuple expires and *MUST* be removed.

In a node, the set of Attached Network Tuples is denoted the "Attached Network Set".

5.2.5. Routing Set

A node records a set of "Routing Tuples" describing the selected path to each destination in the network for which a route is known:

(R_dest_addr, R_next_iface_addr, R_dist, R_local_iface_addr)

R_dest_addr is the address of the destination, either the address of an OLSRv2 interface of a destination node, or the network address of an attached network;

R_next_iface_addr is the OLSRv2 interface address of the "next hop" on the selected path to the destination;

R_dist is the number of hops on the selected path to the destination;

R_local_iface_addr is the address of the local interface over which a packet MUST be sent to reach the destination.

In a node, the set of Routing Tuples is denoted the "Routing Set".

6. OLSRv2 Control Message Structures

Nodes using OLSRv2 exchange information through messages. One or more messages sent by a node at the same time are combined into a packet. These messages may have originated at the sending node, or have originated at another node and forwarded by the sending node. Messages with different originators may be combined in the same packet.

The packet and message format used by OLSRv2 is defined in [3]. However this specification contains some options which are not used by OLSRv2. In particular (using the syntactical elements defined in the packet format specification):

- All OLSRv2 packets, not limited to those defined in this document, include a <packet-header>.
- All OLSRv2 packets, not limited to those defined in this document, have the pseqnum bit of <packet-semantics> cleared ('0'), i.e. they include a packet sequence number.
- OLSRv2 packets MAY include packet TLVs, however OLSRv2 itself does not specify any packet TLVs.
- All OLSRv2 messages, not limited to those defined in this document, include a full <msg-header> and hence have the noorig and nohops bits of <msg-semantics> cleared ('0').
- All OLSRv2 message defined in this document have the typedep bit, and all reserved bits of <msg-semantics> cleared ('0').

Other options defined in [3] may be freely used, in particular any other values of <packet-semantics> or <tlv-semantics> consistent with its specification.

OLSRv2 messages are sent using UDP, see Appendix C.

The remainder of this section defines, within the framework of [3], message types and TLVs specific to OLSRv2.

6.1. General OLSRv2 Message TLVs

This document specifies two message TLVs, which can be applied to any OLSRv2 control messages, VALIDITY_TIME and INTERVAL_TIME.

6.1.1. VALIDITY_TIME TLV

All OLSRv2 messages specified in this document MUST include a

VALIDITY_TIME TLV, specifying a period following receipt of a message, after which the receiving node MUST consider the message content to no longer be valid (unless repeated in a later message). The validity time of a message MAY be specified to depend on the distance from the originator (i.e. the <hop-count> field in the message header as defined in [3]). Thus, a VALIDITY_TIME TLV's value field MAY contain a sequence of pairs (time, hop limit) in increasing hop limit order; it MUST contain a final default value.

This is an extended, and compatible, version of the VALIDITY_TIME TLV defined in [4].

Thus, an instance of a VALIDITY_TIME TLV MAY have the following value field:

<t_1><hl_1><t_2><hl_2> ... <t_i><hl_i> .... <t_n><hl_n><t_default>

Which would mean that the message carrying this VALIDITY_TIME TLV would have the following validity times:

- <t_1> in the interval from 0 (exclusive) to <hl_1> (inclusive) hops away from the originator;
- <t_i> in the interval from <hl_(i-1)> (exclusive) to <hl_i> (inclusive) hops away from the originator;
- <t_default> in the interval from <hl_n> (exclusive) to 255 (inclusive) hops away from the originator.

The VALIDITY_TIME message TLV specification is given in Table 1.

Name	Type	Length	Value
VALIDITY_TIME	TBD	(2*n+1) * 8 bits	{<time><hop_limit>}* <t_default>

Table 1

where n is the number of (time, hop_limit) pairs in the TLV (i.e. is equal to (<length>-1)/2, where <length> is the length of the TLV value field) and where <time> and <t_default> are represented as specified in [3].

6.2. HELLO Messages

A HELLO message in OLSRv2 is generated as specified in [4].

Additionally, an OLSRv2 node:

- MUST include TLV(s) with Type == MPR associated with all OLSRv2 interface addresses included in the HELLO message with a TLV with Type == LINK_STATUS and Value == SYMMETRIC if that address is also included in the node's MPR Set (if there is more than one copy of the address, this applies to the specific copy of the address to which the TLV is associated);

- MUST NOT include any TLVs with Type == MPR associated with any other addresses;

- MAY include a message TLV with Type == WILLINGNESS, indicating the node's willingness to be selected as MPR.

6.2.1. HELLO Message OLSRv2 Message TLVs

In a HELLO message, a node MAY include a WILLINGNESS message TLV as specified in Table 2.

Name	Type	Length	Value
WILLINGNESS	TBD	8 bits	The node's willingness to be selected as MPR, any unused bits (based on the maximum willingness value WILL_ALWAYS) are RESERVED and SHOULD be set to zero.

Table 2

A node's willingness to be selected as MPR ranges from WILL_NEVER (indicating that a node MUST NOT be selected as MPR by any node) to WILL_ALWAYS (indicating that a node MUST always be selected as MPR).

If a node does not advertise a Willingness TLV in HELLO messages, the node MUST be assumed to have a willingness of WILL_DEFAULT.

6.2.2. HELLO Message OLSRv2 Address Block TLVs

In a HELLO message, a node MAY include MPR address block TLV(s) as specified in Table 3.

Name	Type	Length	Value
MPR	TBD	0 bits	No value, i.e. novalue bit ([3]) set

Table 3

6.3. TC Messages

A TC message MUST contain:

- a message TLV with Type == CONT_SEQ_NUM, as specified in [3];
- a message TLV with Type == VALIDITY_TIME, as specified in Section 6.1.1;
- a first address block containing all of the node's OLSRv2 interface addresses. This is similar to the Local Interface Block specified in [4], however these addresses MUST be included in the same order in all copies of a given TC message, regardless of which interface it is transmitted on, and no OTHER_IF address block TLV(s) are required;
- additional address block(s) containing all addresses in the Advertised Address Set and Attached Network Set, the latter (only) with associated GATEWAY address block TLV(s), as specified in Section 6.4, both with associated PREFIX_LENGTH TLV(s), as specified in [3], as necessary.

A TC message MAY contain:

- a message TLV INTERVAL_TIME, as specified in [4].

6.4. TC Message: OLSRv2 Address Block TLVs

In a TC message, a node MAY include GATEWAY address block TLV(s) as specified in Table 4.

Name	Type	Length	Value
GATEWAY	TBD	0 bits	No value, i.e. novalue bit ([3]) set

Table 4

7. HELLO Message Generation

An OLSRv2 HELLO message is composed as defined in [4], with the following TLVs added:

- a message TLV with Type == WILLINGNESS and Value == the node's willingness to act as an MPR, MAY be included in the message;

- for each symmetric 1-hop neighbor OLSRv2 interface address which is included in the HELLO message with an associated TLV with Type == LINK_STATUS and is selected as an MPR (i.e. is present in the MPR Set), an address TLV with Type == MPR MUST be included, this SHOULD be associated with the same copy of the address as the TLV with Type == LINK_STATUS;

- for each 1-hop neighbor OLSRv2 interface address which is included in the HELLO message but is not selected as an MPR (i.e. is not present in the MPR Set), an address TLV with Type == MPR MUST NOT be included.

7.1. HELLO Message: Transmission

Messages are retransmitted in the packet/message format specified by [3] with the ALL-MANET-NEIGHBORS address as destination IP address and with TTL (IPv4) or hop limit (IPv6) equal to 1.

8. HELLO Message Processing

Subsequent to the processing of HELLO messages, as specified in [4], the node MUST:

1. Determine the willingness of the originating node to be an MPR by:

   * if the HELLO message contains a message TLV with Type == WILLINGNESS then the willingness is the value of that TLV, ignoring the reserved bits in that field;

   * otherwise the willingness is WILL_DEFAULT.

2. Update each Link Tuple whose L_neighbor_iface_addr is present in the Local Interface Block of the HELLO message, with:

   * L_willingness = the willingness of the originating node;

3. Update its MPR Selector Set, according to Section 8.1.

8.1. Populating the MPR Selector Set

On receiving a HELLO message, a node MUST:

1. If a node finds one of its own interface addresses with an associated TLV with Type == MPR in the HELLO message (indicating that the originator node has selected the receiving node as an MPR), the MPR Selector Set MUST be updated as follows:

   1. For each address, henceforth neighbor address, in the Local Interface Block of the received HELLO message, where the neighbor address is present as an N_neighbor_iface_addr in a Symmetric Neighbor Tuple with N_STATUS == SYMMETRIC:

      1. If there exists no MPR Selector Tuple with:

         - MS_neighbor_iface_addr == neighbor address

         then a new MPR Selector Tuple is created with:

         - MS_neighbor_iface_addr = neighbor address

      2. The MPR Selector Tuple (new or otherwise) with:

         - MS_neighbor_iface_addr == neighbor address

         is then modified as follows:

- MS_time = current time + validity time

2. Otherwise if a node finds one of its own interface addresses with an associated TLV with Type == LINK_STATUS and Value == SYMMETRIC in the HELLO message (indicating, since there is no TLV with Type == MPR, that originator node has de-selected the receiving node as an MPR), the MPR Selector Set MUST be updated as follows:

   1. All MPR Selector Tuples whose N_neighbor_iface_addr is in the Local Interface Block of the HELLO message are removed.

MPR Selector Tuples are also removed upon expiration of MS_time, or upon symmetric link breakage as described in Section 8.2.

8.2. Symmetric Neighborhood and 2-Hop Neighborhood Changes

A node MUST also perform the following:

1. If a Link Tuple with L_STATUS == SYMMETRIC is removed, or its L_STATUS changes from SYMMETRIC to HEARD or LOST, and if that Link Tuple's L_neighbor_iface_addr is an MS_iface_addr of an MPR Selector Tuple, then that MPR Selector Tuple MUST be removed.

2. If any of:

   * a Link Tuple is added with L_STATUS == SYMMETRIC, OR;

   * a Link Tuple with L_STATUS == SYMMETRIC is removed, or its L_STATUS changes from SYMMETRIC to HEARD or LOST, or vice versa, OR;

   * a 2-Hop Neighbor Tuple is added or removed, OR;

   * the Neighbor Address Association Set is changed such that the subset of any NA_neighbor_iface_addr_list consisting of those addresses which are the L_neighbor_iface_addr of a Link Tuple with L_STATUS == SYMMETRIC is changed, including the cases of removal or addition of a Neighbor Address Association Tuple containing any such addresses;

   then the MPR Set MUST be recalculated.

An additional HELLO message MAY be sent when the MPR Set changes, in addition to the cases specified in [4], and subject to the same constraints.

9. TC Message Generation

A node with one or more OLSRv2 interfaces, and with a non-empty Advertised Neighbor Set or which acts as a gateway to an associated network which is to be advertised in the MANET, MUST generate TC messages. A node with an empty Advertised Neighbor Set and which is not acting as such a gateway SHOULD also generate "empty" TC messages for a period A_HOLD_TIME after it last generated a non-empty TC message. TC messages (non-empty and empty) are generated according to the following:

1. the TC message MUST contain a message TLV with Type == CONT_SEQ_NUM and Value == ANSN from the Advertised Neighbor Set;

2. the TC message MUST contain a message TLV with Type == VALIDITY_TIME and Value == T_HOLD_TIME as specified in Section 6.1.1;

3. the TC message MAY contain a message TLV with Type == INTERVAL_TIME and Value == TC_INTERVAL as specified in [4];

4. the TC message MUST contain the addresses of all of its OLSRv2 interfaces in its first address block, note that the TC message generated on all OLSRv2 interfaces MUST be identical (including having identical message sequence number) and hence these addresses are not ordered or otherwise identified according to the interface on which the TC message is transmitted;

5. the TC message MUST contain, in address blocks other than its first:

    1. A_neighbor_iface_addr from each Advertised Neighbor Tuple;

    2. the addresses of all associated hosts and networks for which this node is to act as a gateway and which is to be advertised in the MANET, each associated with a TLV with Type == GATEWAY.

6. the TC message MAY be fragmented, only by its originator. It SHOULD be fragmented only if necessary; if the TC message is fragmented, a FRAGMENTATION TLV MUST be included, and each fragment SHOULD be indicated as "partially or wholly self contained" in it, and MUST indicate that the content sequence number (ANSN) is message type specific.

9.1. TC Message: Transmission

TC messages are generated and transmitted periodically on all OLSRv2 interfaces, with a default interval between two consecutive TC emissions by the same node of TC_INTERVAL. TC messages MAY be generated in response to a change of contents (a change in ANSN, due to a change in the Advertised Neighbor Set or the advertised locally attached networks) but a node must respect a minimum interval of TC_MIN_INTERVAL between generated TC messages.

TC messages SHOULD be generated with a message hop limit [3] greater than or equal to the expected network diameter (by default with a hop limit of 255).

TC messages are transmitted with the ALL-MANET-NEIGHBORS multicast address as destination IP address and are forwarded according to the specification in Section 4.4.

10. TC Message Processing

When according to Section 4.3 a TC message is to be processed according to its type, this means that processing is carried out according to Section 10.1 and Section 10.2. The timing of this processing depends on whether the TC message is a fragment, and if so whether it is "partially or wholly self-contained":

- if the message is not a fragment, then first Section 10.1 and then Section 10.2 are carried out when the message is received;

- if the message is a fragment which is "partially or wholly self-contained", then processing according to Section 10.1 is carried out when the message is received, and processing according to Section 10.2 is carried out when all matching fragments have been received and all processing according to Section 10.1 has been carried out;

- if the message is a fragment which is not "partially or wholly self-contained", then processing according to Section 10.1 is carried out when all matching fragments have been received, and processing according to Section 10.2 is carried out when all matching fragments have been received and all processing according to Section 10.1 has been carried out.

For all processing purposes, "ANSN" is defined as being the value of the message TLV with Type == CONT_SEQ_NUM in the TC message. If a TC message has no such TLV then the processing of Section 10.1 and Section 10.2 MUST NOT be carried out. (Note that if the message is a fragment it will have already been discarded according to Section 4.3.) If more than one TC message is processed according to Section 10.2 then these must have the same ANSN to be recognized as fragments of the same message.

10.1. Single TC Message Processing

For the purpose of this section, note the following:

- "validity time" is calculated from the VALIDITY_TIME message TLV in the TC message according to the specification in Section 6.1.1;

- "originator address" refers to the originator address in the TC message header;

- comparisons of sequence numbers are carried out as specified in Section 15.

The TC message is processed as follows:

1. the ANSN History Set is updated according to Section 10.1.1; if the TC message is indicated as discarded in that processing then the following steps are not carried out;

2. the Topology Set is updated according to Section 10.1.2;

3. the Attached Network Set is updated according to Section 10.1.3.

10.1.1. Populating the ANSN History Set

The node MUST update its ANSN History Set as follows:

1. if there is an ANSN History Tuple with:

   * AH_orig_addr == originator address; AND

   * AH_seq_number > ANSN

   then the TC message MUST be discarded;

2. otherwise create a new ANSN History Tuple with:

   * AH_orig_addr = originator address;

   * AH_seq_number = ANSN;

   * AH_time = current time + validity time.

   possibly replacing an existing ANSN History Tuple with the same AH_orig_addr.

10.1.2. Populating the Topology Set

The node SHOULD update its Topology Set as follows:

1. for each address, henceforth local address, in the first address block in the TC message:

   1. for each address, henceforth advertised address, in an address block other than the first in the TC message, and which does not have an associated TLV with Type == GATEWAY:

      1. if there is a Topology Tuple with:

         T_dest_iface_addr == advertised address; AND

T_last_iface_addr == local address

then update this Topology Tuple to have:

T_seq_number = ANSN;

T_time = current time + validity time

2. otherwise create a new Topology Tuple with:

T_dest_iface_addr = advertised address;

T_last_iface_addr = local address;

T_seq_number = ANSN;

T_time = current time + validity time.

10.1.3. Populating the Attached Network Set

The node SHOULD update its Attached Network Set as follows:

1. for each address, henceforth gateway address, in the first address block in the TC message:

    1. for each address, henceforth network address, in an address block other than the first in the TC message, and which has an associated TLV with Type == GATEWAY:

        1. if there is a Attached Network Tuple with:

            AN_net_addr == network address; AND

            AN_gw_iface_addr == gateway address

            then update this Attached Network Tuple to have:

            AN_seq_number = ANSN;

            AN_time = current time + validity time

        2. otherwise create a new Attached Network Tuple with:

            AN_net_addr = network address;

AN_gw_iface_addr = gateway address

AN_seq_number = ANSN;

AN_time = current time + validity time

10.2. Completed TC Message Processing

The TC message(s) are processed as follows:

1. the Topology Set is updated according to Section 10.2.1;

2. the Attached Network Set is updated according to Section 10.2.2.

10.2.1. Purging the Topology Set

The Topology Set MUST be updated as follows:

1. for each address, henceforth local address, in the first address block of any of the TC messages, all Topology Tuples with:

   T_last_iface_addr == local address; AND

   T_seq_number < ANSN

   MUST be removed.

10.2.2. Purging the Attached Network Set

The Attached Network Set MUST be updated as follows:

1. for each address, henceforth local address, in the first address block of any of the TC messages, all Attached Network Tuples with:

   AN_gw_iface_addr == local address; AND

   AN_seq_number < ANSN

   MUST be removed.

11. Populating the MPR Set

Each node MUST select, from among its symmetric 1-hop neighbors, a subset of nodes as MPRs. This subset MUST be selected such that a message transmitted by the node, and retransmitted by all its MPRs, will be received by all of its symmetric strict 2-hop neighbors.

Each node selects its MPR Set individually, utilizing the information in the Symmetric Neighbor Set, the 2-Hop Neighbor Set and the Neighborhood Address Association Set. Initially these sets will be empty, as will be the MPR Set. A node SHOULD recalculate its MPR Set when a relevant change is made to the Symmetric Neighbor Set, the 2-Hop Neighbor Set or the Neighborhood Address Association Set.

More specifically, a node MUST calculate MPRs per interface, the union of the MPR Sets of each interface make up the MPR Set for the node. All OLSRv2 interfaces of nodes selected as MPRs with which the node has a symmetric link MUST be added to the MPR Set. Also symmetric 1-hop neighbor nodes with willingness WILL_NEVER (as recorded in the Link Set) MUST NOT be considered as MPRs.

MPRs are used to flood control messages from a node into the network while reducing the number of retransmissions that will occur in a region. Thus, the concept of MPR is an optimization of a classical flooding mechanism. While it is not essential that the MPR Set is minimal, it is essential that all symmetric strict 2-hop neighbors can be reached through the selected MPR nodes. A node MUST select an MPR Set such that any strict 2-hop neighbor is "covered" by at least one MPR node. A node MAY select additional MPRs beyond the minimum set. Keeping the MPR Set small ensures that the overhead of OLSRv2 is kept at a minimum.

Appendix A contains an example heuristic for selecting MPRs.

12. Populating Derived Sets

The Relay Set and the Advertised Neighbor Set of OLSRv2 are denoted derived sets, since updates to these sets are not directly a function of message exchanges, but rather are derived from updates to other sets, in particular the MPR Selector Set.

12.1. Populating the Relay Set

The Relay Set contains the set of neighbor addresses, for which a node is supposed to relay broadcast traffic. This set SHOULD at least contain all addresses in the MPR Selector Set. This set MAY contain additional symmetric 1-hop neighbor addresses.

12.2. Populating the Advertised Neighbor Set

The Advertised Neighbor Set contains the set of OLSRv2 interface addresses of those 1-hop neighbors to which a node advertises a symmetric link in TC messages. This set SHOULD at least contain all of the OLSRv2 interface addresses of the nodes in the MPR Selector Set (i.e. all addresses associated with an MPR Selector node through the Neighborhood Address Association Set, that is, appearing in the same NA_neighbor_iface_addr_list as any MS_neighbor_iface_addr). This set MAY also contain OLSRv2 interface addresses of other symmetric 1-hop neighbors.

Whenever an address is removed from the Advertised Neighbor Set, the ANSN MUST be incremented. Whenever an address is added to the Advertised Neighbor Set, the ANSN MUST be incremented.

13. Routing Table Calculation

The Routing Set is updated when a change (an entry appearing or disappearing, or changing between SYMMETRIC and LOST) is detected in:

- the Link Set, OR;
- the Neighbor Address Association Set, OR;
- the 2-Hop Neighbor Set, OR;
- the Topology Set, OR;
- the Attached Network Set.

Note that some changes to these sets do not necessitate a change to the Routing Set, in particular changes to the Link Set which do not involve Link Tuples with L_STATUS == SYMMETRIC (either before or after the change), similar changes to the Neighbor Address Association Set. A node MAY avoid updating the Routing Set in such cases.

Updates to the Routing Set does not generate or trigger any messages to be transmitted. The state of the Routing Set SHOULD, however, be reflected in the IP routing table by adding and removing entries from the routing table as appropriate.

To construct the Routing Set of node X, a shortest path algorithm is run on the directed graph containing

- the arcs X -> Y where there exists a Link Tuple with Y as L_neighbor_iface_addr and L_STATUS == SYMMETRIC (i.e. Y is a symmetric 1-hop neighbor of X), AND;
- the arcs Y -> Z where Y is added as above and the Link Tuple with Y as L_neighbor_iface_addr has L_willingness not equal to WILL_NEVER, and there exists a 2-Hop Neighbor Tuple with Y as N2_neighbor_iface_addr and Z as N2_2hop_iface_addr (i.e. Z is a symmetric 2-hop neighbor of Z through Y, which does not have willingness WILL_NEVER), AND;
- the arcs U -> V, where there exists a Topology Tuple with U as T_last_iface_addr and V as T_dest_iface_addr (i.e. this is an advertised link in the network).

The graph is complemented with:

- arcs Y -> W where there exists a Link Tuple with Y as L_neighbor_iface_addr and L_STATUS == SYMMETRIC and a Neighborhood Address Association Tuple with Y and W both contained in NA_neighbor_iface_addr_list (i.e. Y and W are both addresses of the same symmetric 1-hop neighbor), AND;

- arcs U -> T where there exists an Attached Network Tuple with U as AN_net_addr and T as AN_gw_iface_addr (i.e. U is a gateway to network T).

The following procedure is given as an example for (re-)calculating the Routing Set using a variation of Dijkstra's algorithm. Thus:

1. All Routing Tuples are removed.

2. For each Link Tuple with L_STATUS == SYMMETRIC, a new Routing Tuple is added with:

   * R_dest_addr = L_neighbor_iface_addr of the Link Tuple;

   * R_next_iface_addr = L_neighbor_iface_addr of the Link Tuple;

   * R_dist = 1;

   * R_local_iface_addr = L_local_iface_addr of the Link Tuple.

3. For each Neighbor Address Association Tuple, for which two addresses A1 and A2 are in NA_neighbor_iface_addr_list where:

   * there is a Routing Tuple with:

     + R_dest_addr == A1

   * and there is no Routing Tuple with:

     + R_dest_addr == A2

   then a Routing Tuple is added with:

   * R_dest_addr = A2;

   * R_next_iface_addr = R_next_iface_addr of the Routing Tuple in which R_dest_addr == A1;

   * R_dist = 1;

   * R_local_iface_addr = R_local_iface_addr of the Routing Tuple in which R_dest_addr == A1.

4. The following procedure, which adds Routing Tuples for destination nodes h+1 hops away, MUST be executed for each value of h, starting with h=2 and incrementing by 1 for each iteration. The execution MUST stop if no new Routing Tuples are added in an iteration.

   1. For each Topology Tuple, if

      + T_dest_iface_addr is not equal to R_dest_addr of any Routing Tuple, AND;

      + T_last_iface_addr is equal to R_dest_addr of a Routing Tuple whose R_dist == h;

      then a new Routing Tuple MUST be added, with:

      + R_dest_addr = T_dest_iface_addr;

      + R_next_iface_addr = R_next_iface_addr of the Routing Tuple whose R_dest_addr == T_last_iface_addr;

      + R_dist = h+1;

      + R_local_iface_addr = R_local_iface_addr of the Routing Tuple whose R_dest_addr == T_last_iface_addr.

      Several Topology Tuples may be used to select a next hop R_next_iface_addr for reaching the address R_dest_addr. When h == 1, ties should be broken such that nodes with highest willingness are preferred, and between nodes of equal willingness, MPR selectors are preferred over non-MPR selectors.

   2. After the above iteration has completed, if h == 1, for each 2-Hop Neighbor Tuple where:

      + N2_2hop_iface_addr is not equal to R_dest_addr of any Routing Tuple, AND;

      + N2_neighbor_iface_addr has a willingness (i.e. the L_willingness of the Link Tuple of which L_neighbor_iface_addr == N2_neighbor_iface_addr) which is not equal to WILL_NEVER;

      a Routing Tuple is added with:

      + R_dest_addr = N2_2hop_iface_addr of the 2-Hop Neighbor Tuple;

+ R_next_iface_addr = R_next_iface_addr of the Routing Tuple in which R_dest_addr == N2_neighbor_iface_addr;

+ R_dist = 2;

+ R_local_iface_addr = R_local_iface_addr of the Routing Tuple in which R_dest_addr == N2_neighbor_iface_addr.

5. For each Attached Network Tuple, if

* AN_net_addr is not equal to R_dest_addr of any Routing Tuple, AND;

* AN_gw_iface_addr is equal to R_dest_addr of a Routing Tuple;

then a new Routing Tuple MUST be added, with:

* R_dest_addr = AN_net_addr;

* R_next_iface_addr = R_next_iface_addr of the Routing Tuple whose R_dest_addr == AN_gw_iface_addr;

* R_dist = R_dist of the Routing Tuple whose R_dest_addr == AN_gw_iface_addr;

* R_local_iface_addr = R_local_iface_addr of the Routing Tuple whose R_dest_addr == AN_gw_iface_addr.

If more than one Attached Network Tuple has the same AN_net_addr, then more than one Routing Tuple MUST NOT be added, and the added Routing Tuple MUST have minimum R_dist.

14. Proposed Values for Constants

This section list the values for the constants used in the description of the protocol.

14.1. Neighborhood Discovery Constants

The constants HELLO_INTERVAL, REFRESH_INTERVAL, HELLO_MIN_INTERVAL, H_HOLD_TIME, L_HOLD_TIME, N_HOLD_TIME and C are used as in [4].

14.2. Message Intervals

- TC_INTERVAL = 5 seconds
- TC_MIN_INTERVAL = TC_INTERVAL/4

14.3. Holding Times

- T_HOLD_TIME = 3 x TC_INTERVAL
- A_HOLD_TIME = T_HOLD_TIME
- P_HOLD_TIME = 30 seconds
- FG_HOLD_TIME = 30 seconds
- RX_HOLD_TIME = 30 seconds
- FW_HOLD_TIME = 30 seconds

14.4. Willingness

- WILL_NEVER = 0
- WILL_DEFAULT = 3
- WILL_ALWAYS = 7

15. Sequence Numbers

Sequence numbers are used in OLSRv2 with the purpose of discarding "old" information, i.e. messages received out of order. However with a limited number of bits for representing sequence numbers, wrap-around (that the sequence number is incremented from the maximum possible value to zero) will occur. To prevent this from interfering with the operation of OLSRv2, the following MUST be observed when determining the ordering of sequence numbers.

The term MAXVALUE designates in the following one more than the largest possible value for a sequence number. For a 16 bit sequence number (as are those defined in this specification) MAXVALUE is 65536.

The sequence number S1 is said to be "greater than" the sequence number S2 if:

- S1 > S2 AND S1 - S2 <= MAXVALUE/2 OR

- S2 > S1 AND S2 - S1 > MAXVALUE/2

Thus when comparing two messages, it is possible - even in the presence of wrap-around - to determine which message contains the most recent information.

16. IANA Considerations

16.1. Multicast Addresses

A well-known multicast address, ALL-MANET-NEIGHBORS, must be registered and defined for both IPv6 and IPv4. The addressing scope is link-local, i.e. this address is similar to the all nodes/routers multicast address of IPv6 in that it targets all OLSRv2 capable nodes adjacent to the originator of an IP datagram.

16.2. Message Types

OLSRv2 defines one message type, which must be allocated from the "Assigned Message Types" repository of [3]

Mnemonic	Value	Description
TC	TBD	Topology Control (global signaling)

Table 5

16.3. TLV Types

OLSRv2 defines one Message TLV type, which must be allocated from the "Assigned message TLV Types" repository of [3]

Mnemonic	Value	Description
WILLINGNESS	TBD	Specifies a node's willingness to act as a relay and to partake in network formation

Table 6

OLSRv2 defines one Address Block TLV type, which must be allocated from the "Assigned address block TLV Types" repository of [3]

Mnemonic	Value	Description
MPR	TBD	Specifies that a given address is selected as MPR

Table 7

17. References

17.1. Normative References

[1] Clausen, T. and P. Jacquet, "The Optimized Link State Routing Protocol", RFC 3626, October 2003.

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

[3] Clausen, T., Dean, J., Dearlove, C., and C. Adjih, "Generalized MANET Packet/Message Format", work in progress draft-ietf-manet-packetbb-01.txt, June 2006.

[4] Clausen, T., Dean, J., and C. Dearlove, "MANET Neighborhood Discovery Protocol (NHDP)", work in progress draft-ietf-manet-nhdp-00.txt, June 2006.

17.2. Informative References

[5] Atkins, D., Stallings, W., and P. Zimmermann, "PGP Message Exchange Formats", RFC 1991, August 1996.

[6] ETSI, "ETSI STC-RES10 Committee. Radio equipment and systems: HIPERLAN type 1, functional specifications ETS 300-652", June 1996.

[7] Jacquet, P., Minet, P., Muhlethaler, P., and N. Rivierre, "Increasing reliability in cable free radio LANs: Low level forwarding in HIPERLAN.", 1996.

[8] Qayuum, A., Viennot, L., and A. Laouiti, "Multipoint relaying: An efficient technique for flooding in mobile wireless networks.", 2001.

Appendix A. Example Heuristic for Calculating MPRs

The following specifies a proposed heuristic for selection of MPRs.

In graph theory terms, MPR computation is a "set cover" problem, which is a difficult optimization problem, but for which an easy and efficient heuristics exist: the so-called "Greedy Heuristic", a variant of which is described here. In simple terms, MPR computation constructs an MPR Set that enables a node to reach any symmetric 2-hop neighbors by relaying through an MPR node.

There are several peripheral issues that the algorithm needs to address. The first one is that some nodes have some willingness WILL_NEVER. The second one is that some nodes may have several interfaces.

The algorithm hence can be summarized by:

- All 1-hop neighbor nodes with willingness equal to WILL_NEVER MUST ignored in the following algorithm: they are not considered as 1-hop neighbors (hence not used as MPRs).

- Because link sensing is performed by interface, the local network topology is best described in terms of links: hence the algorithm is considering 1-hop neighbor OLSRv2 interfaces, and 2-hop neighbor OLSRv2 interfaces (and their addresses). Additionally, asymmetric links are ignored. This is reflected in the definitions below.

- MPR computation is performed on each interface of the node: on each interface I, the node MUST select some neighbor interfaces, so that all 2-hop neighbor interfaces are reached.

From now on, MPR calculation will be described for one interface I on the node, and the following terminology will be used in describing the heuristics:

neighbor interface (of I) - An OLSRv2 interface of a 1-hop neighbor to which there exist a symmetric link using interface I.

N - the set of such neighbor interfaces

2-hop neighbor interface (of I) An interface of a symmetric strict 2-hop neighbor which can be reached from a neighbor interface for I.

N2 - the set of such 2-hop neighbor interfaces

D(y): - the degree of a 1-hop neighbor interface y (where y is a member of N), is defined as the number of symmetric neighbor interfaces of node y which are in N2

MPR Set - the set of the neighbor interfaces selected as MPRs.

The proposed heuristic selects iteratively some interfaces from N as MPRs in order to cover 2-hop neighbor interfaces from N2, as follows:

1. Start with an MPR Set made of all members of N with L_willingness equal to WILL_ALWAYS

2. Calculate D(y), where y is a member of N, for all interfaces in N.

3. Add to the MPR Set those interfaces in N, which are the *only* nodes to provide reachability to an interface in N2. For example, if interface B in N2 can be reached only through a symmetric link to interface A in N, then add interface B to the MPR Set. Remove the interfaces from N2 which are now covered by a interface in the MPR Set.

4. While there exist interfaces in N2 which are not covered by at least one interface in the MPR Set:

    1. For each interface in N, calculate the reachability, i.e., the number of interfaces in N2 which are not yet covered by at least one node in the MPR Set, and which are reachable through this neighbor interface;

    2. Select as an MPR the interface with highest L_willingness among the interfaces in N with non-zero reachability. In case of multiple choice select the interface which provides reachability to the maximum number of interfaces in N2. In case of multiple interfaces providing the same amount of reachability, select the interface as MPR whose D(y) is greater. Remove the interfaces from N2 which are now covered by an interface in the MPR Set.

Other algorithms, as well as improvements over this algorithm, are possible. For example:

- Assume that in a multiple interface scenario there exists more than one link between nodes 'a' and 'b'. If node 'a' has selected node 'b' as MPR for one of its interfaces, then node 'b' can be selected as MPR with minimal performance loss by any other

interfaces on node 'a'.

- In a multiple interface scenario MPRs are selected for each interface of the selecting node, providing full coverage of all 2-hop nodes accessible through that interface. The overall MPR Set is then the union of these sets. These sets do not however have to be selected independently, if a node is selected as an MPR for one interface it may be automatically added to the MPR selection for other interfaces.

Appendix B. Heuristics for Generating Control Traffic

A node creates HELLO messages and TC messages as specified in Section 7 and Section 9, the former being a modification of the specification in [4]. The heuristics for creation of HELLO messages in [4] remain applicable, with the division of the address TLVs with Type == LINK_STATUS and Value == SYMMETRIC into separate ranges with and without an associated TLV with Type == MPR. The heuristics for collection of addresses are also generally applicable to TC messages, excepting that the first address block is not sorted as the Local Interface Block of a HELLO message is, and that other addresses recorded in TC messages are divided into those with and without a TLV with Type == GATEWAY. These should be ordered so that the range of addresses without that TLV is continuous (and it is suggested that the range without is also continuous).

Appendix C. Protocol and Port Number

Packets in OLSRv2 are communicated using UDP. Port 698 has been assigned by IANA for exclusive usage by the OLSR (v1 and v2) protocol.

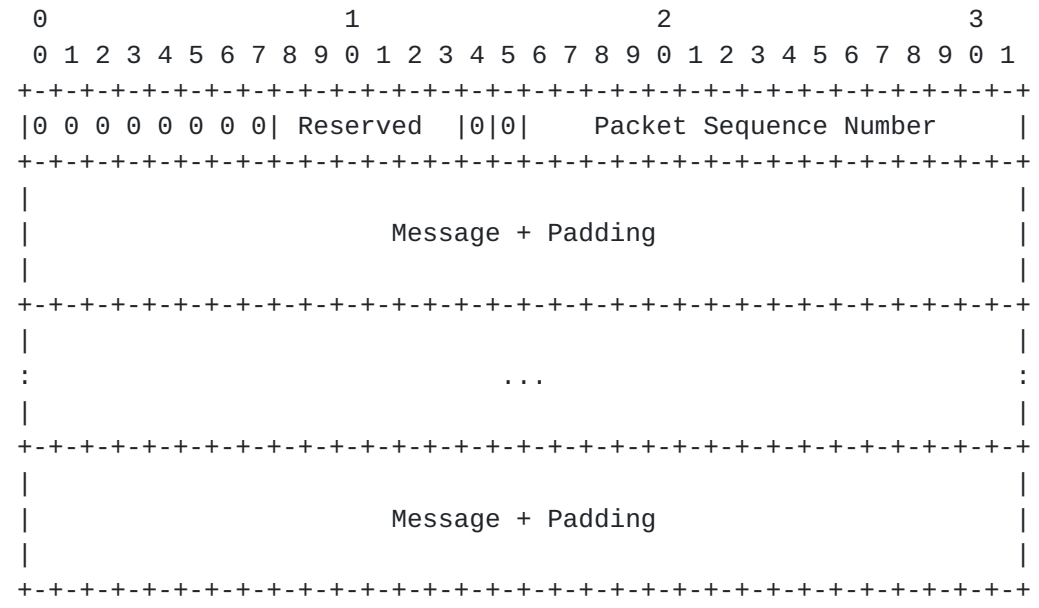
Appendix D. Packet and Message Layout

This section specifies the translation from the abstract descriptions of packets employed in the protocol specification, and the bit-layout packets actually exchanged between the nodes.

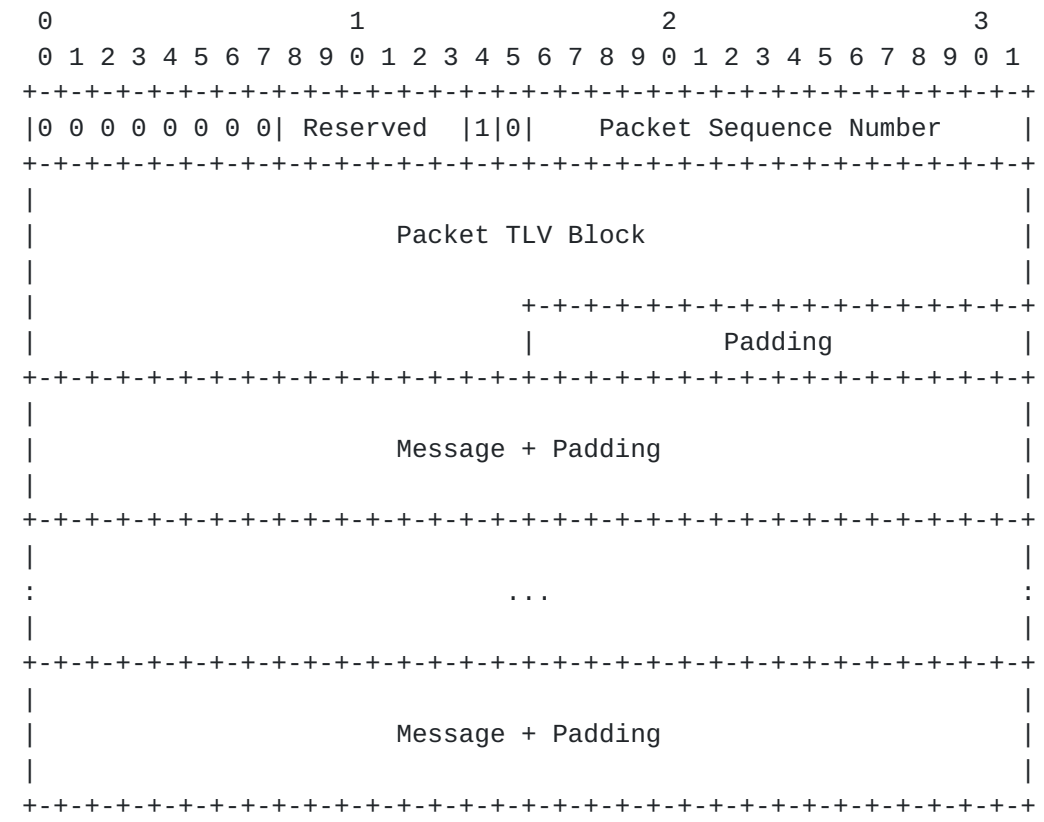
Appendix D.1. OLSRv2 Packet Format

The basic layout of an OLSRv2 packet is as described in [3]. However the following points should be noted.

OLSRv2 uses only packets with a packet header including a packet sequence number, either with or without a packet TLV block. Thus all OLSRv2 packets have the layout of either

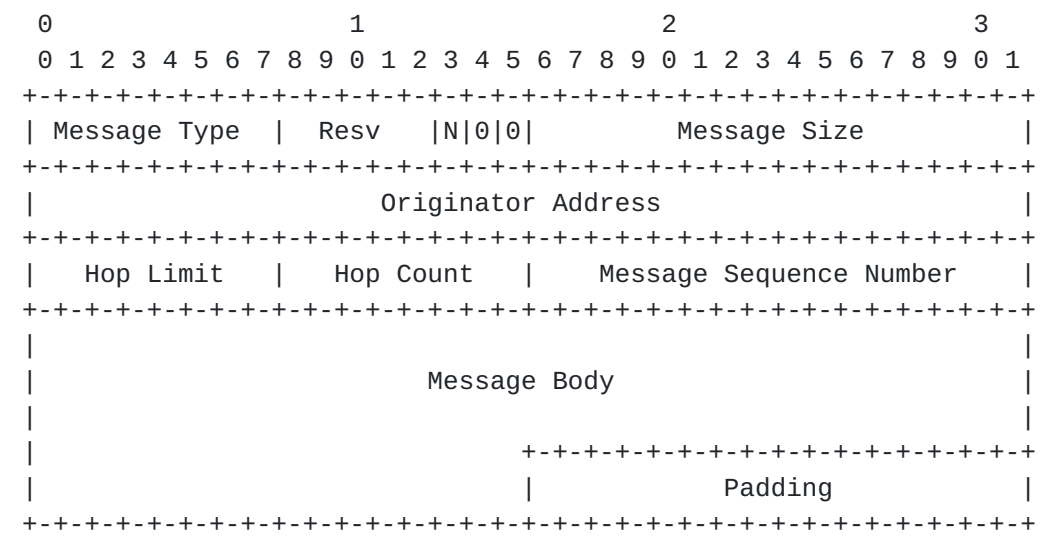


or



The reserved bits marked Resv SHOULD be cleared ('0'). The octets indicated as Padding are optional and MAY be omitted; if not omitted they SHOULD be used to pad to a 32 bit boundary and MUST all be zero.

OLSRv2 uses only messages with a complete message header. Thus all OLSRv2 messages, plus padding if any, have the following layout.



The reserved bits marked Resv SHOULD be cleared ('0'). In standard OLSRv2 messages (HELLO and TC) the type dependent sequence number bit marked N SHOULD also be cleared ('0').

The layouts of the message body, address block, TLV block and TLV are as in [3], allowing all options. Standard (HELLO and TC) messages contain a first address block which contains local interface address information, all other address blocks contain neighbor interface address information (or for a TC message address information for which it is a gateway) specific to the message type.

An example HELLO message, using IPv4 (four octet) addresses is as follows. The overall message length is 56 octets (it does not need padding). The message has a hop limit of 1 and a hop count of 0, as sent by its originator.

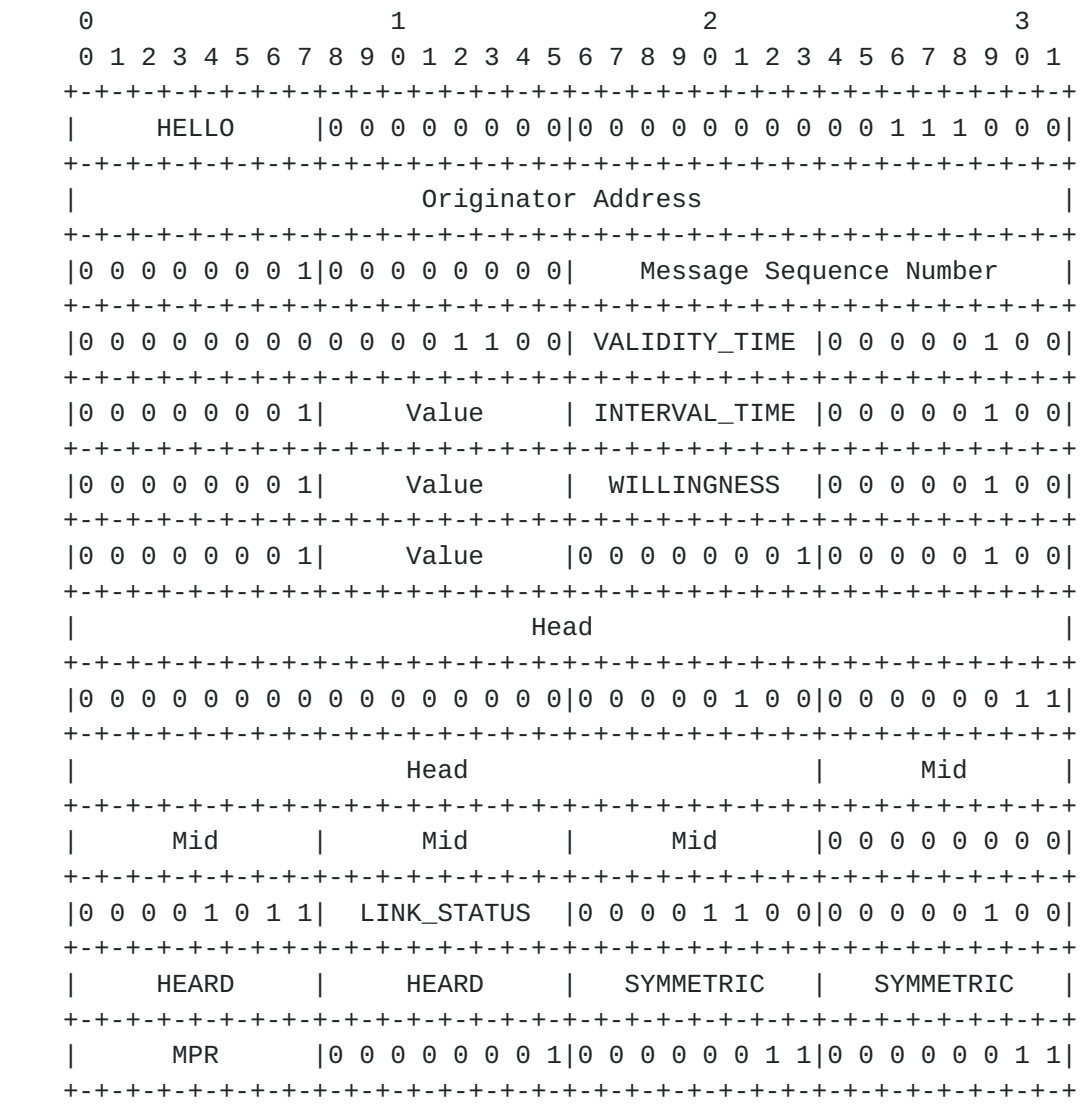
The message has a message TLV block with content length 12 octets containing three message TLVs. These TLVs represent message validity time, message interval time and willingness. Each uses a TLV with semantics value 4, indicating no start and stop indexes are included, and each has a value length of 1 octet.

The first address block contains a 1 local interface address, with head length 4. This is equal to the address length, thus no tail or mid sections of the address are included. This address block has no TLVs (the TLV block content length is 0 octets).

The second, and last, address block reports 4 neighbor interface addresses, with address head length 3 octets, and no tail octet (zero tail length). Thus each mid address section is of length one octet. The following address TLV block (content length 11 octets) includes two TLVs.

The first of these TLVs reports the link status of all four neighbors in a single multivalue TLV, the first two addresses are HEARD, the last two addresses are SYMMETRIC. The TLV semantics value of 12 indicates, in addition to that this is a multivalue TLV, that no start index and stop index are included, hence values for all addresses are included. The TLV value length of 4 octets indicates one octet per value per address.

The second of these TLV indicates that the last address (start index 3, stop index 3) is an MPR. This TLV has no value, or value length, fields, as indicated by its semantics octet being equal to 1.



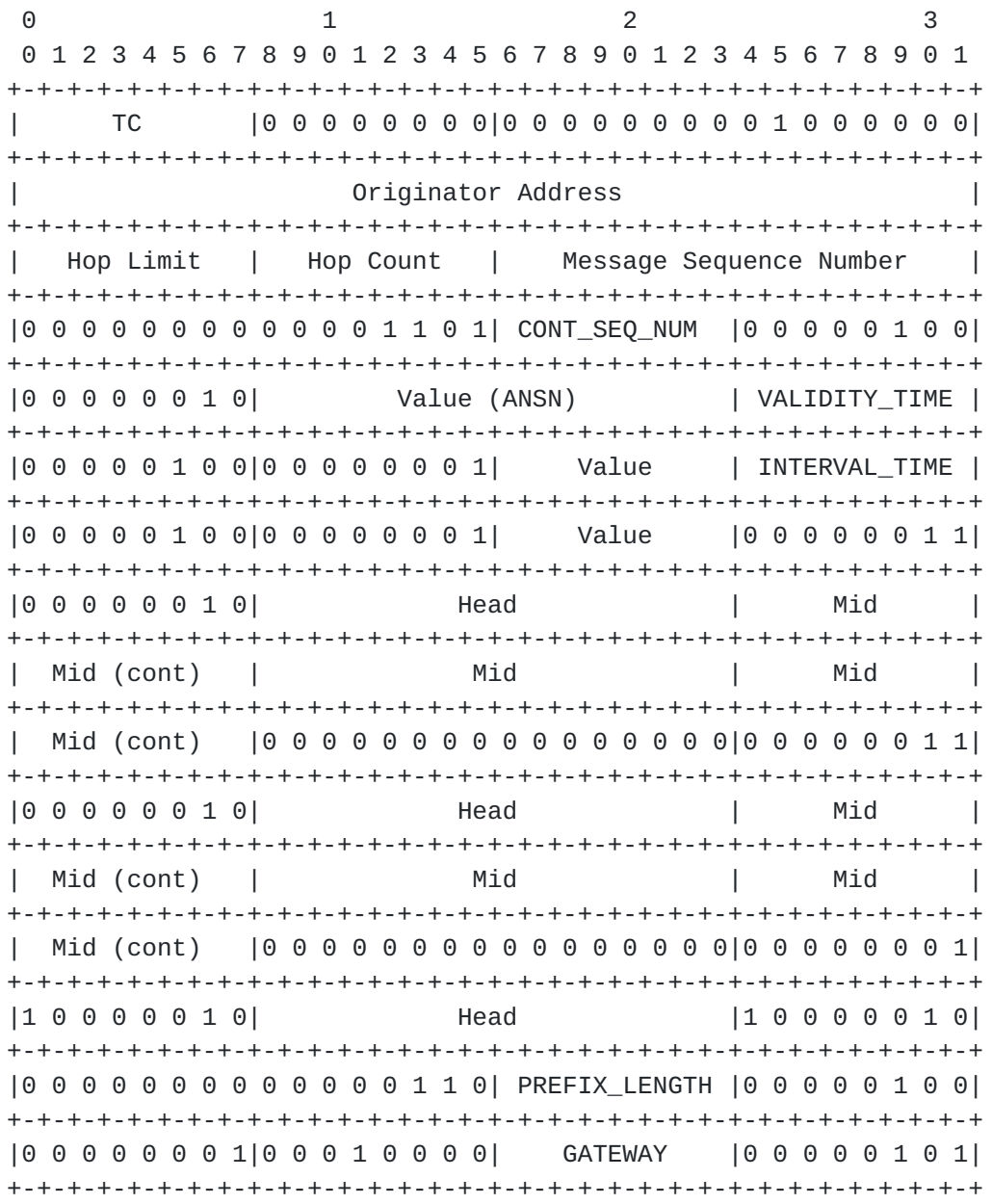
An example TC message, using IPv4 (four octet) addresses, is as follows. The overall message length is 64 octets, it also does not need padding.

The message has a message TLV block with content length 13 octets containing three TLVs. The first TLV is a content sequence number TLV used to carry the 2 octet ANSN. The semantics value is 4 indicating that no index fields are included. The other two TLVs are validity and interval times as for the HELLO message above.

The message has three address blocks. The first address block contains 3 local interface addresses (with common head length 2 octets) and has a TLV block with content length 0 octets.

The other two address blocks contain neighbor interface addresses.

The first contains 3 addresses and has an empty TLV block (content length 0 octets). The second contains 1 address. The head octet (hex 82) indicates a head length of two octets and the presence of a tail octet. The tail octet (hex 82) indicates a tail length of two octets, all zero bits and not included. The following TLV block (content length 6 octets) includes two TLVs, the first (semantics value 4 indicating no indexes are needed) indicates that the address has a netmask, with length given by the value (of length 1 octet) of 16. Thus this address is Head.0.0/16. The second TLV indicates that the originating node is a gateway to this network, the TLV semantics value of 5 indicates that neither indexes nor value are needed.



Appendix E. Node Configuration

OLSRv2 does not make any assumption about node addresses, other than that each node is assumed to have at least one a unique and routable IP address for each interface that it has which participates in the MANET.

When applicable, a recommended way of connecting an OLSRv2 network to an existing IP routing domain is to assign an IP prefix (under the authority of the nodes/gateways connecting the MANET with the routing domain) exclusively to the OLSRv2 area, and to configure the gateways statically to advertise routes to that IP sequence to nodes in the existing routing domain.

Appendix F. Jitter

In a wireless network, simultaneous packet transmission by nearby nodes is undesirable as, depending on the medium access control and other lower layer mechanisms, the interference between these messages may cause at best increased delay, and at worst complete packet loss by both nodes. This is often particularly true when using a broadcast mechanism, such as is used by OLSRv2 packets.

The problems of simultaneous packet transmission in OLSRv2 are increased by the following features of the protocol:

- If two nodes send packets containing regularly scheduled messages of the same type at the same time, then if, as is typical, they are using the same message interval, further transmissions of these messages will also be at the same time, and will also interfere. This node synchronization could even result in complete operational failure of these nodes.

- OLSRv2 allows nodes to respond to changes in their circumstances, usually changes in the neighborhood, with immediate messages of appropriate types. Nearby nodes will have overlapping neighborhoods, and may respond to the same change in circumstances. For example a single link failure can result in a node having to change its MPR Set, and then two or more of its neighbors having changed MPR status responding simultaneously with revised TC messages, whose packets may interfere.

- When a node sends such a responsive message, there is no apparent reason why it should not restart its message schedule of the appropriate type of message. This results in nodes responding to the same change not just sending single simultaneous packets, but becoming synchronized.

- Nodes also forward messages they receive from other nodes. Two nearby nodes will thus commonly receive and forward the same message. The consequent packet transmissions can easily interfere with each other.

Because interference can easily occur, is self-reinforcing, and is anything from undesirable to catastrophic, mechanisms to minimize it, and to break synchronization of nodes, SHOULD be used in OLSRv2. These all make a deliberate adjustment to the timing, known as "jitter". Three cases exist:

- When a node generates a control message periodically, it would normally wait for a delay equal to MESSAGE_INTERVAL (e.g. HELLO_INTERVAL for HELLO messages or TC_INTERVAL for TC messages)

between two transmissions of messages of that type. This delay SHOULD be mitigated by subtracting a jitter time, so that the delay between consecutive transmissions of a messages of the same type SHOULD be equal to MESSAGE_INTERVAL - jitter, where jitter is a random value whose generation is discussed below. Note that this is a deliberately asymmetric process. It ensures that the message interval does not exceed MESSAGE_INTERVAL (which leaves MESSAGE_INTERVAL an appropriate value for reporting in an INTERVAL_TIME message TLV) and also allows different nodes to become completely desynchronized as each interval is based on the previous actual transmission time, not on a fixed clock of period MESSAGE_INTERVAL.

- When a node responds to an externally triggered change in circumstances, it SHOULD delay the transmission of a message in response by a random jitter time. It MAY restart its schedule of messages of the appropriate type based on that new time. If such a message is delayed due to the need to respect the appropriate MESSAGE_MIN_INTERVAL (e.g. HELLO_MIN_INTERVAL for HELLO messages or TC_MIN_INTERVAL for TC messages) then the node MAY reduce this minimum interval by a jitter time as the normal message interval is reduced (thus allowing MESSAGE_MIN_INTERVAL to equal MESSAGE_INTERVAL even when using jitter).

- When a node forwards a message, it SHOULD delay the message retransmission by a random jitter time.

In the first and second cases above, the maximum jitter time may be specified by a parameter MAXJITTER. It is necessary only that this be significantly less than each MESSAGE_INTERVAL, and less than each MESSAGE_MIN_INTERVAL. Normally the actual value of the jitter (reduction in message interval or delay of responsive message) SHOULD be uniformly generated in the interval 0 <= jitter <= MAXJITTER, however this may be modified as indicated below.

In the third case above, a message SHOULD be delayed by a jitter value which is significantly less than the originating node's message interval. This MAY be available in an INTERVAL_TIME message TLV in the message to be forwarded. If not so available, a node MAY estimate an acceptable maximum jitter by any other means available to it, which may be by use of its own MAXJITTER parameter for as long as this works. In a network in which this is likely to be unsuccessful, nodes SHOULD include an INTERVAL_TIME message TLV in messages which are to be forwarded.

In all cases, as well as constraints imposed by message intervals and message minimum intervals, the maximum jitter delay SHOULD only be as large as is required to achieve the required objective of minimizing

interference due to synchronization. This is because all jitter, and forwarding jitter in particular, is undesirable for otherwise ideal functioning of the network.

Because of differing parameters, or due to responsive messages with a small minimum message interval, a node may receive a message from an originating node while still waiting to forward an earlier message of the same type originating from the same node. The forwarding node SHOULD NOT allow forwarding jitter delay to reorder these messages. A node MAY discard the earlier message, transmitting the later message no later than the earlier message was due to be retransmitted, if, and only if, it can guarantee that this will not have any adverse effect.

OLSRv2 messages are transmitted in potentially multi-message packets. Whilst a packet is a hop by hop construct and it is the messages in it which are forwarded, if a number of messages are received in the same packet, they SHOULD (if their maximum jitter delays are compatible) be permitted to be forwarded in the same new packet. This may be accomplished by generating the same random delay for all messages received in a single packet. Furthermore, the opportunity to combine messages to be forwarded from different sources, and locally generated messages in a single packet SHOULD be allowed even when this means adjusting (forwards or backwards) the strictly uniformly generated random jitter times, however these SHOULD NOT be allowed to exceed their maximum value, nor allow a message interval to be exceeded, nor compromise the purpose of jitter. (It is for this reason that messages in the same packet should be given the same random jitter, as giving them independent jitter values but then, for example, allowing all to be sent with the earliest would reduce the mean jitter delay.)

Appendix G. Security Considerations

Currently, OLSRv2 does not specify any special security measures. As a proactive routing protocol, OLSRv2 makes a target for various attacks. The various possible vulnerabilities are discussed in this section.

Appendix G.1. Confidentiality

Being a proactive protocol, OLSRv2 periodically diffuses topological information. Hence, if used in an unprotected wireless network, the network topology is revealed to anyone who listens to OLSRv2 control messages.

In situations where the confidentiality of the network topology is of importance, regular cryptographic techniques, such as exchange of OLSRv2 control traffic messages encrypted by PGP [5] or encrypted by some shared secret key, can be applied to ensure that control traffic can be read and interpreted by only those authorized to do so.

Appendix G.2. Integrity

In OLSRv2, each node is injecting topological information into the network through transmitting HELLO messages and, for some nodes, TC messages. If some nodes for some reason, malicious or malfunction, inject invalid control traffic, network integrity may be compromised. Therefore, message authentication is recommended.

Different such situations may occur, for instance:

1. a node generates TC messages, advertising links to non-neighbor nodes;

2. a node generates TC messages, pretending to be another node;

3. a node generates HELLO messages, advertising non-neighbor nodes;

4. a node generates HELLO messages, pretending to be another node;

5. a node forwards altered control messages;

6. a node does not forward control messages;

7. a node does not select multipoint relays correctly;

8. a node forwards broadcast control messages unaltered, but does not forward unicast data traffic;

9. a node "replays" previously recorded control traffic from another node.

Authentication of the originator node for control messages (for situations 2, 4 and 5) and on the individual links announced in the control messages (for situations 1 and 3) may be used as a countermeasure. However to prevent nodes from repeating old (and correctly authenticated) information (situation 9) temporal information is required, allowing a node to positively identify such delayed messages.

In general, digital signatures and other required security information may be transmitted as a separate OLSRv2 message type, thereby allowing that "secured" and "unsecured" nodes can coexist in the same network, if desired, or signatures and security information may be transmitted within the OLSRv2 HELLO and TC messages, using the TLV mechanism.

Specifically, the authenticity of entire OLSRv2 control messages can be established through employing IPsec authentication headers, whereas authenticity of individual links (situations 1 and 3) require additional security information to be distributed.

An important consideration is, that all control messages in OLSRv2 are transmitted either to all nodes in the neighborhood (HELLO messages) or broadcast to all nodes in the network (TC messages).

For example, a control message in OLSRv2 is always a point-to-multipoint transmission. It is therefore important that the authentication mechanism employed permits that any receiving node can validate the authenticity of a message. As an analogy, given a block of text, signed by a PGP private key, then anyone with the corresponding public key can verify the authenticity of the text.

Appendix G.3. Interaction with External Routing Domains

OLSRv2 does, through the use of TC messages, provide a basic mechanism for injecting external routing information to the OLSRv2 domain. Appendix E also specifies that routing information can be extracted from the topology table or the routing table of OLSRv2 and, potentially, injected into an external domain if the routing protocol governing that domain permits.

Other than as described in Appendix E, when operating nodes, connecting OLSRv2 to an external routing domain, care MUST be taken not to allow potentially insecure and untrustworthy information to be injected from the OLSRv2 domain to external routing domains. Care MUST be taken to validate the correctness of information prior to it

being injected as to avoid polluting routing tables with invalid information.

A recommended way of extending connectivity from an existing routing domain to an OLSRv2 routed MANET is to assign an IP prefix (under the authority of the nodes/gateways connecting the MANET with the exiting routing domain) exclusively to the OLSRv2 MANET area, and to configure the gateways statically to advertise routes to that IP sequence to nodes in the existing routing domain.

Appendix G.4. Node Identity

OLSRv2 does not make any assumption about node addresses, other than that each node is assumed to have at least one a unique and routable IP address for each interface that it has which participates in the MANET.

Appendix H. Flow and Congestion Control

TBD

Appendix I. Contributors

This specification is the result of the joint efforts of the following contributors -- listed alphabetically.

- Cedric Adjih, INRIA, France, <Cedric.Adjih@inria.fr>
- Emmanuel Baccelli, Hitachi Labs Europe, France, <Emmanuel.Baccelli@inria.fr>
- Thomas Heide Clausen, PCRI, France<T.Clausen@computer.org>
- Justin Dean, NRL, USA<jdean@itd.nrl.navy.mil>
- Christopher Dearlove, BAE Systems, UK, <Chris.Dearlove@baesystems.com>
- Satoh Hiroki, Hitachi SDL, Japan, <h-satoh@sdl.hitachi.co.jp>
- Philippe Jacquet, INRIA, France, <Philippe.Jacquet@inria.fr>
- Monden Kazuya, Hitachi SDL, Japan, <monden@sdl.hitachi.co.jp>
- Kenichi Mase, Niigata University, Japan, <mase@ie.niigata-u.ac.jp>
- Ryuji Wakikawa, KEIO University, Japan, <ryuji@sfc.wide.ad.jp>

Appendix J. Acknowledgements

The authors would like to acknowledge the team behind OLSRv1, specified in RFC3626, including Anis Laouiti, Pascale Minet, Laurent Viennot (all at INRIA, France), and Amir Qayuum (Center for Advanced Research in Engineering, Pakistan) for their contributions.

The authors would like to gratefully acknowledge the following people for intense technical discussions, early reviews and comments on the specification and its components: Li Li (CRC), Louise Lamont (CRC), Joe Macker (NRL), Alan Cullen (BAE Systems), Philippe Jacquet (INRIA), Khaldoun Al Agha (LRI), Richard Ogier (SRI), Song-Yean Cho (Samsung Software Center), Shubhranshu Singh (Samsung AIT) and the entire IETF MANET working group.

Authors' Addresses

Thomas Heide Clausen
LIX, Ecole Polytechnique, France

Phone: +33 6 6058 9349
Email: T.Clausen@computer.org
URI: http://www.lix.polytechnique.fr/Labo/Thomas.Clausen/

Christopher M. Dearlove
BAE Systems Advanced Technology Centre

Phone: +44 1245 242194
Email: chris.dearlove@baesystems.com
URI: http://www.baesystems.com/ocs/sharedservices/atc/

Philippe Jacquet
Project Hipercom, INRIA

Phone: +33 1 3963 5263
Email: philippe.jacquet@inria.fr
URI: http://hipercom.inria.fr/test/Jacquet.htm

The OLSRv2 Design Team
MANET Working Group

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.