T. Clausen
LIX, Ecole Polytechnique, France
C. Dearlove
BAE Systems Advanced Technology Centre
The OLSRv2 Design Team
MANET Working Group
March 6, 2006

The Optimized Link-State Routing Protocol version 2
draft-ietf-manet-olsrv2-01

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on September 7, 2006.

Copyright Notice

Copyright (C) The Internet Society (2006).

Abstract

This document describes version 2 of the Optimized Link State Routing (OLSRv2) protocol for mobile ad hoc networks. The protocol is 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. A secondary optimization is, that OLSRv2 employs partial link-state information: each node maintains information of 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 broadcasts) and that these advertisements contain only a subset of links (i.e. reduces the size of each network-wide broadcast). The partial link-state information thus obtained 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 accomodate 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). In OLSRv2, 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 the route 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 [5].

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].

Additionally, this document uses the following terminology:

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

OLSRv2 interface - A network device participating in a MANET running OLSRv2. A node may have several OLSRv2 interfaces, each interface assigned one or more IP addresses.

neighbor - A node X is a neighbor of node Y if node Y can hear node X (i.e., a link exists from an OLSRv2 interface on node X to an OLSRv2 interface on node Y). A neighbor may also be called a 1-hop neighbor.

2-hop neighbor - A node X is a 2-hop neighbor of node Y if node X is a neighbor of a neighbor of node Y, but is not node Y itself.

strict 2-hop neighbor - a 2-hop neighbor which is not a neighbor of the node, and is not a 2-hop neighbor only through a neighbor with willingness WILL_NEVER.

multipoint relay (MPR) - A node which is selected by its 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 time to live field of the message is greater than one.

multipoint relay selector (MPR selector, MS) - A node which has selected its 1-hop neighbor, node X, as one of its multipoint relays, will be called an MPR selector of node X.

link - A link is a pair of OLSRv2 interfaces from two different nodes, where at least one interface is able to hear (i.e. receive traffic from) the other.

symmetric link - A link where both interfaces are able to hear (i.e. receive messages from) the other.

asymmetric link - A link which is not symmetric.

symmetric 1-hop neighborhood - The symmetric 1-hop neighborhood of any node X is the set of nodes which have at least one symmetric link to node X.

symmetric 2-hop neighborhood - The symmetric 2-hop neighborhood of node X is the set of nodes, excluding node X itself, which have a symmetric link to the symmetric 1-hop neighborhood of X.

symmetric strict 2-hop neighborhood - The symmetric strict 2-hop neighborhood of node X is the set of nodes in its symmetric 2-hop neighborhood that are neither in its symmetric 1-hop neighborhood nor reachable only through a symmetric 1-hop neighbor of node X with willingness WILL_NEVER.

1.2 Applicability Statement

OLSRv2 is a proactive routing protocol for mobile ad hoc networks (MANETs) [6], [7]. 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 continously, 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 independant specification [4], which is used for carrying OLSRv2 control signals. OLSRv2 is thereby able to accommodate for extensions via "external" and "internal" extensibility. External extensibility implies that a

protocol extension may specify and exchange new message types which can be forwarded and delivered correctly according to [4]. Internal extensibility implies, that a protocol extension may define additional attributes to be carried embedded in the OLSRv2 control messages, detailed in this specification, while these OLSRv2 control messages with additional attributes can still be correctly understood by all OLSRv2 nodes.

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 will know of all destinations and a subset of links in the network.

More specifically, OLSRv2 consists of the following main components:

- A general and flexible signaling framework, allowing for information exchange between OLSRv2 nodes. This framework allows for both local information exchange (between neighboring nodes) and global information exchange using an optimized flooding mechanism denoted "MPR flooding".

- A specification of local signaling, denoted HELLO messages. HELLO messages in OLSRv2 serve to:

  * discover links to adjacent OLSR nodes;

  * perform bidirectionality check on the discovered links;

  * advertise neighbors and hence discover 2-hop neighbors;

  * signal MPR selection.

  HELLO messages are emitted periodically, thereby allowing nodes to continuously track changes in their local neighborhoods.

- A specification of global signaling, denoted TC 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.

  * allow nodes with multiple interface addresses to ensure that nodes within two hops can associate these addresses with a

single node for efficient MPR Set determination.

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

Thus, through periodic exchange of HELLO messages, a node is able to acquire and maintain information about its immediate neighborhood. This includes information about immediate neighbors, as well as nodes which are two hops away. By HELLO messages being exchanged periodically, a node learns about changes in the neighborhood (new nodes emerging, old nodes disappearing) without requiring explicit mechanisms for doing so.

Based on the local topology information, acquired through the periodic exchange of HELLO messages, an OLSRv2 node is able 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.

The idea of multipoint relays is to minimize the overhead of flooding messages in the network by reducing redundant retransmissions in the same region. Each node in the network selects a set of nodes in its symmetric 1-hop neighborhood which may retransmit its messages. This set of selected neighbor nodes is called the "Multipoint Relay" (MPR) Set of that node. The neighbors of node N which are *NOT* in its MPR set, receive and process broadcast messages but do not retransmit broadcast messages received from node N. The MPR Set of a node is selected such that it covers (in terms of radio range) all symmetric strict 2-hop nodes. The MPR Set of N, denoted as MPR(N), is then an arbitrary subset of the symmetric 1-hop neighborhood of N which satisfies the following condition: every node in the symmetric strict 2-hop neighborhood of N MUST have a symmetric link towards MPR(N). The smaller a MPR Set, the less control traffic overhead results from the routing protocol. [7] gives an analysis and example of MPR selection algorithms. Notice, 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 neighbors that have selected it as MPR. This set is called the "Multipoint Relay Selector Set" (MPR Selector Set) of a node. A node obtains this information from periodic HELLO messages received from the neighbors. 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.

A broadcast message, intended to be diffused in the whole network,

coming from any of the nodes in the Relay Set of node N is assumed to be retransmitted by node N, if N has not received it yet. This set can change over time (e.g., when a node selects another MPR Set) and is indicated by the selector nodes in their HELLO messages.

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, TC messages, containing the links between the node and the nodes in the Advertised Neighbor Set, are not generated, unless needed for gateway reporting or multiple interface address association (if the latter case only, with minimal scope).

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 occur frequently in radio networks due to collisions or other transmission problems.

Also, 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. Furthermore, OLSRv2 provides support for protocol extensions such as sleep mode operation, multicast-routing etc. Such extensions may be introduced as additions to the protocol without breaking backwards compatibility with earlier versions.

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

2.1 Protocol Extensibility

This specification defines and uses two OLSRv2 message types, HELLO and TC. As for OLSRv1 [1] extensions to OLSRv2 may define new message types to carry additional information. This may be considered as "external" extensibility. New message types are divided into two ranges, those which may be added by standards actions (with types up to 127) and those made available for private/ local use (with types 128 to 255).

All new messages must be syntactically OLSRv2 messages, as defined in

[4]. (Some additional constraints to that specification are added for OLSRv2 packets and messages, requiring full packet and message headers.) Note that if it is required to include one or more blocks of unstructured data in such a message (possibly as its only content) this may be achieved by including each block as a single message TLV block, with an appropriately defined message TLV. (Like message types, TLV types are divided into those up to 127 which may be added by standards action, and those from 128 to 255 available for private/ local use.)

A network may contain nodes both aware of, and unaware of, any new message types. The originator of a message can control whether a message flooded through the network is forwarded by nodes which are unaware of the message type, thus reaching all nodes in the network, or is only flooded by nodes which recognise the message type.

OLSRv2 also supports an alternative, and more powerful, extension mechanism which was not supported by OLSRv1, that of adding new information to an already defined message type, whilst still leaving the predefined information unchanged and usable, including by a node which does not recognise the new information. This may be considered to be "internal" extensibility of a message.

The mechanism for this extensibility is the use of TLV (type-length-value) structures in the message format defined in [4] to carry information associated with either the message as a whole, or with one or more addresses carried in the message. The messages defined in this specification carry two types of addresses, those of the originating node's own interfaces participating in OLSRv2, and those of neighbouring nodes or networks to which it has a route. (New message types may define other relationships to addresses which they carry.) All information associated with these addresses, or the message as a whole, in messages defined in this specification is in TLV format; additional TLVs may be defined and added to these messages.

Those nodes which do not recognise newly defined TLV types ignore the added TLVs. (This is facilitated by that the TLVs defined in this specification, or in [4], have the lowest type numbers and that TLVs must be included in type order, as specified in [4].) It is important that newly defined TLV types permit this behaviour.

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 Message Set

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

(RX_type, RX_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_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 record expires and *MUST* be removed.

In a node, this is denoted the "Received Message 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 record 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_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_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 record 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_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_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 record 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_if_addr)

where:

RY_if_addr is the address of a neighbor interface for which the node SHOULD relay flooded messages.

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

4. Packet Processing and Message Forwarding

Upon receiving a basic packet, 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

Upon receiving a packet, a node MUST perform the following task:

1. If the packet contains no messages (i.e. the packet length is less than or equal to the size of the packet header) or if the packet cannot be parsed into messages, the packet MUST be silently discarded.

2. Otherwise, each message in the packet 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 [4], or if the originator address of the message is an interface address of the receiving node then the message MUST be silently discarded;

2. Otherwise:

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

    2. If for the received message TTL > 0, and if either the message is of a known type, or bit 3 of the message semantics octet in the message header is clear, as indicated in [4], then the message is considered for forwarding according to Section 4.4.

4.3 Message Considered for Processing

If a 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, or 0 if bit 2 of the message semantics octet (in the message header) is clear, AND;

   * P_addr == the originator address of the message, AND;

   * P_seq_number == the sequence number of the message.

   then the message MUST NOT be processed.

2. Otherwise:

   1. Create an entry in the Processed Set with:

      + P_type = the message type, or 0 if bit 2 of the message semantics octet (in the message header) is clear;

      + P_addr = originator address of the message;

      + P_seq_number = sequence number of the message;

      + P_time = current time + P_HOLD_TIME.

   2. If the message does not contain a message TLV of type Fragment (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 message is "wholly or partially self-contained" as indicated by its Fragment TLV then process the current message as far as possible according to its type;

      2. If the Fragment Set includes any messages with the same originator address and content sequence number as the current message, and either the same fragment number or a different number of fragments, then remove these messages are from the Fragment Set;

      3. If the Fragment Set includes messages containing all the remaining fragments of the same overall message as the current message (i.e. if the number of messages in the Fragment Set with the same originator address and content sequence number as the current message is equal to the current message's number of fragments, less one) then all of these messages are removed from the Fragment Set and processed according to their type (taking account of any

previous processing if any or all were wholly or partially self-contained);

4. Otherwise, the current message is added to the Fragment Set with a FG_time of FG_HOLD_TIME (possibly replacing an identical and previous received instance of the same fragment of the same content).

4.4 Message Considered for Forwarding

If a message is considered for forwarding then if it is either of a message type defined in this document, or of an unknown message type 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 there is no symmetric link in the Link Set between the receiving interface and the sending interface (as indicated by the source interface of the IP datagram containing the message) 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 bit 2 of the message semantics octet (in the message header) is clear, AND;

        + RX_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 bit 2 of the message semantics octet (in the message header) is clear;

- RX_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 bit 2 of the message semantics octet (in the message header) is clear;

   - FW_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 an entry exists in the Relay Set, where RY_if_addr == source address of the message (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 bit 2 of the message semantics octet (in the message header) is clear;

      o FW_addr = originator address of the message;

      o FW_seq_number = sequence number of the message;

      o FW_time = current time + FW_HOLD_TIME.

   2. The message header is modified as follows:

      o Decrement the message TTL by 1;

      o Increment the message hop count by 1;

3. Transmit the message on all OLSRv2 interfaces of the node.

Messages are retransmitted in the format specified by [4] with the All-OLSRv2-Multicast address (see Section 17.1) 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 maintains a number of protocol sets, the information repository of the protocol. These sets are updated, directly or indirectly, by the exchange of messages between nodes in the network. In turn the contents of these messages are largely determined by the contents of a part of the information repositories, the Neighbourhood Information Base, which contains information about the 1- and 2- hop neighbourhoods of the node. The remaining part of the information repository, the Topology Information Base (including the Routing Set) contains information about the network which is not constrained to the node's neighbourhood. The Topology Information Base is updated by the OLSRv2 messages defined in this document, it is not used to define their contents. The process of information exchange which leads to the population of the Neighbourhood Information Base and the Topology Information Base is started using only the node's own OLSRv2 interface addresses and host and network associated addresses. These are not affected by the exchange of the OLSRv2 messages defined in this document.

5.1 Neighborhood Information Base

The neighborhood information base stores information about links between local interfaces and interfaces on adjacent nodes.

5.1.1 Link Set

A node records a set of "Link Tuples":

```
(L_local_iface_addr, L_neighbor_iface_addr,
 L_SYM_time, L_ASYM_time, L_willingness, L_time).
```

where:

L_local_iface_addr is the interface address of the local node;

L_neighbor_iface_addr is the interface address of the neighbor node;

L_SYM_time is the time until which the link is considered symmetric;

L_ASYM_time is the time until which the neighbor interface is considered heard;

L_willingness is the nodes willingness to be selected as MPR;

L_time specifies when this record expires and *MUST* be removed.

L_SYM_time	L_ASYM_time	L_STATUS
Expired	Expired	LOST
Not Expired	Expired	SYMMETRIC
Not Expired	Not Expired	SYMMETRIC
Expired	Not Expired	ASYMMETRIC

Table 1

The status of the link, denoted L_STATUS, can be derived based on the fields L_SYM_time and L_ASYM_time as defined in Table 1.

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

5.1.2 2-Hop Neighbor Set

A node records a set of "2-Hop Neighbor Tuples"

(N2_local_iface_addr, N2_neighbor_iface_addr, N2_2hop_iface_addr, N2_time)

describing symmetric links between its neighbors and the symmetric 2-hop neighborhood.

N2_local_iface_addr is the address of the local interface over which the information was received;

N2_neighbor_iface_addr is the interface address of a neighbor;

N2_2hop_iface_addr is the interface address of a 2-hop neighbor with a symmetric link to the node with interface address N_neighbor_iface_addr;

N2_time specifies the time at which the tuple expires and *MUST* be removed.

In a node, the set of 2-Hop Neighbor Tuples is denoted the "2-Hop Neighbor Set".

5.1.3 Neighborhood Address Association Set

A node maintains, for each 1-hop and 2-hop neighbor with multiple addresses participating in the OLSRv2 network, a "Neighborhood Address Association Tuple", representing that "these addresses belong to the same node".

    (NA_neighbor_addr_list, NA_time)

NA_neighbor_iface_addr_list is the list of interface addresses of the 1-hop or 2-hop neighbor node;

NA_time specifies the time at which the tuple expires and *MUST* be removed.

In a node, the set of Neighborhood Address Association Tuples is denoted the "Neighborhood Address Association Set".

5.1.4 MPR Set

A node maintains a set of neighbors which are selected as MPRs. Their interface addresses are listed in the MPR Set.

5.1.5 MPR Selector Set

A node maintains, for each interface of an 1-hop neighbor which has selected it as MPR, an "MPR Selector Tuple", representing the an interface of the neighbor node which have selected it as an MPR.

    (MS_neighbor_if_addr, MS_time)

MS_neighbor_if_addr specifies the interface address of a 1-hop neighbor, which has selected the node as MPR;

MS_time specifies the time at which the tuple expires and *MUST* be removed.

Notice that if a MPR selector node has multiple interface addresses, the MPR Selector Set will contain one tuple for each interface address of the MPR selector.

5.1.6 Advertised Neighbor Set

A node maintains a set of neighbor interface addresses, which are to be advertised through TC messages:

        (A_neighbor_iface_addr)

For this set, an Advertised Neighbor Set Sequence Number (ASSN) is maintained. Each time the Advertised Neighbor Set is updated, the ASSN MUST be incremented.

5.2 Topology Information Base

The Topology Information Base stores topological information describing the network beyond the nodes neighborhood (i.e. beyond the Neighborhood Information Base of the node).

5.2.1 Topology Set

Each node in the network maintains topology information about the network.

For each destination in the network, at least one "Topology Tuple"

    (T_dest_iface_addr, T_last_iface_addr, T_seq, T_time)

is recorded.

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

T_last_iface_addr is, conversely, the last hop towards T_dest_iface_addr. Typically, T_last_iface_addr is a MPR of T_dest_iface_addr;

T_seq is a sequence number, and

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

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

5.2.2 Attached Network Set

Each node in the network maintains information about attached networks.

For each attached network, at least one "Attached Network Tuple"

    (AN_net_addr, AN_prefix_lenght, AN_gw_addr, AN_seq_no, AN_time)

is recorded.

AN_net_addr is the network address (prefix) of a network, which may be reached via the node with the OLSRv2 interface address AN_gw_addr;

AN_prefix_length is the length of the prefix of the network address AN_net_addr;

AN_gw_addr is the address of an OLSRv2 interface of a node which can act as gateway to the network identified by the AD_net_addr/ AD_prefix_length;

AN_seq_no is a sequence number, and;

AN_time specifies the time at which this tuple expires and *MUST* be removed.

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

5.2.3 Routing Set

A node records a set of "Routing Tuples":

(R_dest_iface_addr, R_next_iface_addr, R_dist, R_iface_addr)

describing the next hop and distance of the path to each destination in the network for which a route is known.

R_dest_iface_addr is the interface address of the destination node;

R_next_iface_addr is the interface address of the "next hop" on the path towards R_dest_iface_addr;

R_dist is the number of hops on the path to R_dest_iface_addr;

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

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

- o All OLSRv2 packets include a <packet-header>.
- o All OLSRv2 messages, not limited to those defined in this document, include a full <msg-header> and hence have bits 0 and 1 of <msg-semantics> cleared.
- o All OLSRv2 message defined in this document have all remaining bits of <msg-semantics> cleared.

Other options defined in [4] may be freely used, in particular any values of <tlv-semantics> consistent with its specification. An implementation of OLSRv2 MAY take full advantage of the features of the message specification in [4] allowing decisions relating to whether a message should be forwarded and/or processed to be taken parsing only the message header (plus, if a message is to be processed but may be fragmented, only the first octets of the message body).

OLSRv2 messages are sent using UDP, see Appendix C.

The remainder of this section defines, within the framework of [4], 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 message, VALIDITY_TIME and INTERVAL_TIME, detailed in this section.

6.1.1 VALIDITY_TIME TLV

All OLSRv2 messages specified in this specification MUST include a VALIDITY_TIME TLV, specifying for how long a node may, upon receiving a message, consider the message content to be valid. 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 [4]). Thus, the VALIDITY_TIME TLV contains a sequence of pairs (time, hop-limit) in increasing hop-limit order, followed by a default value.

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

```
<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:

- o <t_1> in the interval from 0 (exclusive) to <hl_1> (inclusive) hops away from the originator;

- o <t_i> in the interval from <hl_(i-1)> (exclusive) to <hl_i> (inclusive) hops away from the originator; and

- o <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 2.

VALIDITY_TIME message TLV specification overview

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

Table 2

where <n> is the number of (time, hop_limit) pairs in the TLV, and where <time> and <t_default> are represented as specified in section Section 16.

6.1.2 INTERVAL_TIME TLV

OLSRv2 messages of a given type MAY include an INTERVAL_TIME message TLV, specifying the interval at which messages of this type are being generated by the originator node.

The INTERVAL_TIME message TLV specification is given in Table 3.

INTERVAL_TIME TLV specification overview

Name	Type	Length	Value
INTERVAL_TIME	TBD	8 bits	<time>

Table 3

where <time> is the time between two successive emissions of messages of the type, represented as specified in section Section 16.

6.2 Local Interface Blocks

The first address block, plus following TLV block in a HELLO or TC message is known as a Local Interface Block. A Local Interface Block is not distinguished in any way other than by being the first address block in the message.

A Local Interface Block contains the addresses of all of the interfaces of the originating node that support OLSRv2 and participate in the MANET, using the standard <address-block> syntax from [4]. In a TC message this is sufficient; in a HELLO message, those addresses, if any, which correspond to interfaces other than that on which the HELLO message is sent must have a corresponding OTHER_IF TLV. In this case (only) this OTHER_IF TLV SHALL NOT have a <value> field.

Note that a Local Interface Block may include more than one address for each interface, and hence in a HELLO message may contain more than one address without an OTHER_IF TLV.

6.3 HELLO Messages

A HELLO message MUST contain:

- a message TLV VALIDITY_TIME Section 6.1.1

- one or more address blocks with associated address block TLVs

The first (mandatory) address block is a Local Interface Block, as specified in Section 6.2. Other (optional) address blocks contain 1-hop neighbors' interface addresses.

A HELLO message MAY optionally contain:

o a message TLV INTERVAL_TIME as specified in Section 6.1.2

o a message TLV WILLINGNESS, as specified in Section 6.3.1

6.3.1 HELLO Message: Message TLVs

In a HELLO message, a node MAY include a message TLV as specified in Table 4.

VALIDITY_TIME message TLV specification overview

Name	Type	Length	Value
WILLINGNESS	TBD	8 bits	<The node's willingness to be selected as MPR>

Table 4

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.

6.3.2 HELLO Message: Address Blocks TLVs

HELLO message address block TLV specification overview

Name	Type	Length	Value
LINK_STATUS	TBD	8 bits	One of HEARD, SYMMETRIC, LOST.
MPR	TBD	0 bits	No value, i.e. novalue bit (see [4]) set

OTHER_IF	TBD	0 or 8 bits	In a Local Interface Block none, otherwise either of SYMMETRIC or LOST

Table 5

6.4 TC Messages

A TC message MUST contain:

o a message TLV VALIDITY_TIME Section 6.1.1

o a message TLV CONTENT_SEQUENCE_NUMBER [4]

o one or more address blocks with associated address block TLVs.

The first (mandatory) address block is a Local Interface Block, as specified in Section 6.2. Other (optional) address blocks contain 1-hop neighbors' interface addresses and/or host or network addresses for which this node may act as a gateway. In the latter case they may use PREFIX_LENGTH TLV(s) as specified in [4].

A TC message MAY optionally contain:

o a message TLV INTERVAL_TIME as specified in Section 6.1.2

7. HELLO Message Generation

An OLSRv2 HELLO message is composed of a set of message TLVs, describing general properties of the message and the node emitting the HELLO, and a set of address blocks (with associated TLV sets), describing the links and their associated properties.

OLSRv2 HELLO messages are generated and transmitted per interface, i.e. different HELLO messages are generated and transmitted per OLSRv2 interface of a node.

OLSRv2 HELLO messages are generated and transmitted periodically, with a default interval between two consecutive HELLO emissions on the same interface of HELLO_INTERVAL.

This section specifies the requirements, which HELLO message generation MUST fulfill. An example algorithm is proposed in Appendix B.1.

For each OLSRv2 interface a node MUST generate a HELLO message with a Local Interface Block as the first address block, as specified in Section 6.2, followed by address blocks and address TLVs according to Table 6.

The set of neighbor interfaces which are ...	TLV(s) (Type = Value)
HEARD, but not SYMMETRIC over the interface over which the HELLO message is being transmitted	LINK_STATUS=HEARD
SYMMETRIC over the interface over which the HELLO message is being transmitted	LINK_STATUS=SYMMETRIC
LOST over the interface over which the HELLO message is being transmitted	LINK_STATUS=LOST

Not SYMMETRIC over the interface over which the HELLO message is being transmitted, but SYMMETRIC over one or more other interfaces of the node	OTHER_IF=SYMMETRIC
Not SYMMETRIC over any interface or LOST over the interface over which the HELLO message is being transmitted, but previously reported as OTHER_IF=SYMMETRIC and still HEARD or LOST over one or more interfaces of the node other than the interface over which the HELLO message is being transmitted	OTHER_IF=LOST
Selected as MPR for the interface over which the HELLO message is transmitted	MPR

Table 6

In order that an address can be reported as OTHER_IF=LOST by a node with more than one interface participating in the MANET, such a node MAY maintain an Other Interface Set of addresses for each interface. The Other Interface Set for an interface is updated when a HELLO message is to be transmitted over that interface, and used to determine which addresses are reported as OTHER_IF=LOST in that message. The Other Interface Set of addresses is updated and used as follows:

1. Each address that the HELLO message is to include with a corresponding TLV with Type=LINK_STATUS and Value=SYMMETRIC is removed from the set.

2. Each address that the HELLO message is to include with a corresponding TLV with Type=OTHER_IF and Value=SYMMETRIC is added to the set if not already present.

3. Each other address in the set (not included in the HELLO message with a corresponding TLV with Type=OTHER_IF and Value=SYMMETRIC)

    1. Is removed if the HELLO message is to include it with a corresponding TLV with Type=LINK_STATUS and Value=LOST.

    2. Is removed if it is not HEARD or LOST over an interface other than the interface over which the HELLO message is to be transmitted.

    3. Otherwise is included in the HELLO message with a TLV with Type=OTHER_IF and Value=LOST. (Note that the address may also have a corresponding TLV with Type=LINK_STATUS and Value=HEARD if appropriate.)

7.1 HELLO Message: Transmission

Messages are retransmitted in the packet/message format specified by [4] with the All-OLSRv2-Multicast address as destination IP address and with a TTL=1.

8. HELLO Message Processing

Upon receiving a HELLO message, a node will update its local link information base according to the specification given in this section.

For the purpose of this section, please notice the following:

- the "validity time" of a message is calculated from the VALIDITY-TIME TLV of the message as specified in Section 6.1.1;

- the "Source Address" is the source address as indicated by the source interface of the IP datagram containing the message;

- a HELLO message MUST neither be forwarded nor be recorded in the Processing and Forwarding Repositories;

- the address blocks considered exclude the Local Interface Block, unless explicitly specified;

- a HELLO message is only valid when, for each address listed in the address blocks:

  * the address is associated with a TLV with Type=Link Status OR a TLV with Type=Other Interface Status OR both, the latter either when the TLV with Type=Link Status has Value=HEARD, or when the the TLV with Type=Link Status has Value=LOST and the TLV with Type=Other Interface Status has Value=SYMMETRIC, AND

  * if the address is associated with a TLV with Type=MPR, then it MUST also be associated with a TLV with Type=Link Status and Value=SYMMETRIC.

  Invalid HELLO messages are not processed.

8.1 Populating the Link Set

Upon receiving a HELLO message, a node SHOULD update its Link Set with the information contained in the HELLO. Thus, for the Local Interface Block (see Section 6.2) the Neighbor Address Association Set is updated as specified by Section 13. For each address, listed in the subsequent HELLO message address blocks (see Section 6):

1. if there exists no link tuple with:

   * L_neighbor_iface_addr == Source Address

a new tuple is created with

* L_neighbor_iface_addr = Source Address;

* L_local_iface_addr = Address of the interface which received the HELLO message;

* L_SYM_time = current time - 1 (expired);

* L_time = current time + validity time.

2. The tuple (existing or new) with L_neighbor_iface_addr == Source Address is then modified as follows:

   1. if the node finds the address of the interface, which received the HELLO message, in one of the address blocks included in message, then the tuple is modified as follows:

      1. if the occurrence of L_local_iface_addr in the HELLO message is:

         - associated with a TLV with (Type == "LINK_STATUS", Value == LOST)

         then

         - L_SYM_time = current time - 1 (i.e., expired)

      2. else if the occurrence of L_local_iface_addr in the HELLO message:

         - is associated with:

           o a TLV with (Type == "LINK_STATUS", Value == SYMMETRIC);

           OR;

           o a TLV with (Type == "LINK_STATUS", Value == HEARD);

         then

         - L_SYM_time = current time + validity time,

         - L_time = L_SYM_time + L_HOLD_TIME.

2. L_ASYM_time = current time + validity time;

3. L_time = max(L_time, L_ASYM_time)

3. Additionally, the willingness field is updated as follows:

If a TLV with Type=="WILLINGNESS" is present in the message TLVs, then:

+ L_willingness = Value of the TLV

otherwise:

+ L_willingness = WILL_DEFAULT

The rule for setting L_time is the following: a link losing its symmetry SHOULD still be advertised in HELLOs (with the remaining status as defined by Table 1) during at least the duration of the "validity time". This allows neighbors to detect the link breakage. Thus, the Local Link Set must maintain information, also about LOST links, until the link would otherwise expire.

8.2 Populating the 2-Hop Neighbor Set

Upon receiving a HELLO message from a symmetric neighbor interface, a node SHOULD update its 2-hop Neighbor Set.

If the Source Address is the L_local_iface_addr from a link tuple included in the Link Set with L_STATUS equal to SYMMETRIC (in other words: if the Source Address is a symmetric neighbor interface) then the 2-hop Neighbor Set SHOULD be updated as follows:

1. for each address (henceforth: 2-hop neighbor address), listed in the HELLO message:

1. if the 2-hop neighbor address is an interface address of the receiving node silently discard the 2-hop neighbor address (in other words: a node is not its own 2-hop neighbor).

2. else if the 2-hop neighbor address has a TLV with:

+ (Type=LINK_STATUS, Value == SYMMETRIC); OR

+ (Type=OTHER_IF, Value=SYMMETRIC);

a 2-hop tuple is created with:

+ N2_local_iface_addr = address of the interface over which the HELLO message was received;

+ N2_neighbor_iface_addr = source address of the message;

+ N2_2hop_iface_addr = 2-hop neighbor address;

+ N2_time = current time + validity time.

This tuple may replace an older similar tuple with the same N2_local_iface_addr, N2_neighbor_iface_addr and N2_2hop_iface_addr values.

3. else if the 2-hop neighbor address has a TLV with:

+ (Type == LINK_STATUS, Value == LOST); OR

+ (Type == OTHER_IF, Value == LOST),

then any 2-hop tuple with:

+ N2_local_iface_addr equal to the address of the interface over which the HELLO message was received; AND

+ N2_neighbor_iface_addr equal to the source address of the message; AND

+ and N2_2hop_iface_addr equal to the 2-hop neighbour address

MUST be deleted.

8.3 Populating the MPR Selector Set

Upon receiving a HELLO message, if a node finds one of its own interface addresses, listed with an MPR TLV (indicating that the originator node has selected one of the receiving node's interfaces as MPR), the MPR Selector Set SHOULD be updated as follows:

For each address in the Local Interface Block of the received message:

1. If there exists no MPR Selector tuple with:

    * MS_if_addr == that address

    then a new tuple is created with:

* MS_if_addr = that address

2. The tuple (new or otherwise) with:

   * MS_if_addr == that address

   is then modified as follows:

   * MS_time = current time + validity time.

MPR Selector tuples are removed upon expiration of MS_time, or upon link breakage as described in Section 8.4.

8.4 Neighborhood and 2-Hop Neighborhood Changes

A change in the neighborhood is detected when:

o Link Loss: the L_SYM_time field of a link tuple expires (either due to time out, or as a result of processing a TLV (Type == LINK_STATUS, Value == LOST)).

o Link Acquisition: a new link tuple is inserted in the Link Set with a non expired L_SYM_time or a tuple with expired L_SYM_time is modified so that L_SYM_time becomes non-expired. This is considered as a link acquisition if there was previously no such link tuple.

o Neighbor Loss: all links to a neighbor node have have been lost.

A change in the 2-hop neighborhood is detected when a 2-Hop Neighbor Tuple expires or is deleted according to section Section 8.2.

The following processing occurs when changes in the neighborhood or the 2-hop neighborhood are detected:

o In case of link loss, all 2-Hop Neighbor Tuples with

  * N2_local_iface_addr == interface address of the node where the link was lost

  * N2_neighbor_iface_addr == interface address of the neighbor

  MUST be deleted.

o In case of neighbor loss, all MPR Selector tuples associated with that neighbor are deleted. More precisely:

* all MPR selector tuples with MS_iface_addr == interface address of the neighbor MUST be deleted, along with any interface addresses associated in the Neighbor Address Association Set.

- o The MPR Set MUST be re-calculated when a link acquisition or loss is detected, or when a change in the 2-hop neighborhood is detected.

- o An additional HELLO message MAY be sent when the MPR Set or the neighborhood changes.

Additionally, proper update of the sets describing local topology should be made when a Neighbor Association Address Tuple has a list of addresses which is modified.

9. TC Message Generation

TC messages are, in OLSRv2, transmitted with the purpose of populating the Topology Set, the Attached Network Set and the Neighborhood Address Association Set:

- o Topology Discovery: ensure that information is present in each node describing all destinations and a sufficient subset of links in order to provide least-hop paths to all destinations.

- o Multiple Interface Declaration: ensure that nodes, up to two hops away from the originator, are aware of the interface configuration of the originator node.

Thus, nodes with a non-empty Advertised Neighbor Set, or which are specifically reporting an empty Advertised Neighbor Set (for a period of T_HOLD_TIME following reporting a non-empty Advertised Neighbor Set) or with more than one interface which supports OLSRv2 and participates in the MANET, MUST generate TC messages, according to the following:

1. The node includes, in its first address block of the TC message, a Local Interface Block as specified in Section 6.2

2. If the node has a non-empty Advertised Neighbor Set or is specifically reporting an empty Advertised Neighbor Set, or it has a one or more attached non-OLSRv2 networks, to which it wishes to advertise routes to the network, it furthermore:

    1. includes a message TLV (Type = CONTENT_SEQ_NUMBER TLV, Value = the Advertised Neighbor Set Sequence Number);

    2. includes address blocks, containing its Advertised Neighbor Set (if non-empty);

    3. includes address blocks and PREFIX_LENGTH TLVs, describing attached non-OLSRv2 networks;

    4. sets the TTL of the message to the network diameter.

3. Otherwise, the node:

    1. sets the TTL of the message to 2.

OLSRv2 TC messages are generated and transmitted periodically, with a default interval between two consecutive TC emissions by the same node of TC_INTERVAL.

9.1 TC Message: Transmission

Messages are retransmitted in the packet/message format specified by [4] with the All-OLSRv2-Multicast address as destination IP address and is forwarded according to the specification in section Section 4.4. If fragmentation is necessary, a FRAGMENTATION TLV MUST be included, and each fragment SHOULD be flagged as partially or wholly self contained as specified in [4].

10. TC Message Processing

Upon receiving a TC message, a node MUST update its topology information base according to the specification given in this section.

For the purpose of this section, note the following:

- o the "validity time" of a message is calculated from the VALIDITY_TIME message TLV according to the specification in Section 16;
- o the "originator address" refers to the address, contained in the "originator address" field of the OLSRv2 message header specified in [4];
- o the ASSN of the node, originating the TC message, is recovered as the value of the CONTENT_SEQ_NO message TLV in the TC message, if any.

10.1 Checking Freshness & Validity of a TC message

In order to be able to ensure that only valid and fresh information is recorded in the Topology Set, each node maintains an ASSN History Set, recording the highest ASSN received from each node in the network, in the form of a "ASSN History Tuples":

(AS_Address, AS_seq, AS_time)

AS_Address is the originator address of a received TC message;

AS_seq is the highest received ASSN seen in a TC message from AS_Address;

AS_time is the time at which this tuple expires and MUST be removed.

Upon receiving a TC message, a node MUST check if the TC message is fresh and valid as follows:

1. If the TC message has more than one address block (i.e. not just a Local Interface Block) and does not contain a message-TLV of type CONTENT_SEQ_NO. then the message MUST be discarded;

2. otherwise, if the ASSN History Set contains a tuple where:

   * AS_Address == Originator Address of the TC message; AND

* AS_seq > the ASSN recovered from the TC message,

then the TC message MUST be discarded;

3. otherwise a tuple is inserted in the ASSN History Set with:

   * AS_Address = Originator Address in the message;

   * AS_seq = The ASSN, extracted from the message;

   * AS_time = current time + AS_HOLD_TIME.

   possibly replacing an existing tuple with the same AS_Address.

10.2 Updating the Topology Set

A node SHOULD update its Topology Set as follows:

1. For each address, LocAddr, from the Local Interface Block in the TC message:

   1. For each advertised neighbor address, listed in an address block other than the Local Interface Block in the TC message, which does NOT have an associated PREFIX_LENGTH TLV:

      1. if there exists a tuple in the Topology Set where:

         T_dest_iface_addr == advertised neighbor address; AND

         T_last_iface_addr == LocAddr.

         then the tuple is updated as follows:

         T_time = current time + validity time

         T_seq = ASSN

      2. Otherwise, a new topology tuple is created with:

         T_dest_iface_addr = advertised neighbor address, AND

         T_last_iface_addr = LocAddr; AND

         T_seq = ASSN.

10.3 Purging Old Entries from the Topology Set

Old entries from the Topology Set MUST be purged as follows:

1. For each address, LocAddr, from the Local Interface Block in the TC message:

    1. all tuples in the Topology Set where:

        T_last_iface_addr == LocAddr AND

        T_seq < ASSN

        MUST be removed.

10.4 Updating the Attached Networks Set

A node SHOULD update its Attached Networks Set as follows:

1. For each address, LocAddr, from the Local Interface Block in the TC message:

    1. For each advertised neighbor address, listed in an address block other than the Local Interface Block in the TC message, which does have an associated PREFIX_LENGTH TLV:

        1. if there exists a tuple in the Attached Networks Set where:

            AN_net_addr == advertised neighbor address; AND

            AN_prefix_length == the prefix length as recoveredf from the PREFIX_LENGTH TLV; AND

            AN_gw_addr == LocAddr.

            then the tuple is updated as follows:

            AN_time = current time + validity time

            AN_seq = ASSN

        2. Otherwise, a new topology tuple is created with:

AN_net_addr == advertised neighbor address; AND

AN_prefix_length == the prefix length as recoveredf from the PREFIX_LENGTH TLV; AND

AN_gw_addr == LocAddr.

AN_time = current time + validity time

AN_seq = ASSN

10.5 Purging Old Entries from the Attached Network Set

TBD

10.6 Processing Unfragmented TC Messages

If an unfragmented TC message, i.e. a TC message without a FRAGMENTATION message TLV, is received, it MUST be processed as follows:

1. Verify freshness and validity of the TC message (see Section 10.1). If the message is not discarded, then continue;

2. Update the Topology Set (see Section 10.2);

3. Purge old entries from the Topology Set (see Section 10.3);

4. Update the Attached Networks Set (see Section 10.4;

5. Purge old entries from the Attached Networks Set (see Section 10.5);

6. Update the Neighborhood Address Association Set (see Section 13).

10.7 Processing Partially or Wholly Self-Contained Fragmented TC Messagess

If a TC message contains a FRAGMENTATION message TLV which indicates that the fragment is a partially or wholly self-contained message, then the following processing SHOULD be carried out immediately upon receipt of each received fragment (if not then it MUST be carried out for each fragment once all fragments have been received):

1. Verify freshness and validity of the TC message (see Section 10.1). If the message is not discarded, then continue;

2. Update the Topology Set (see Section 10.2);

3. Update the Neighborhood Address Association Set (see Section 13).

4. Update the Attached Networks Set (see Section 10.4;

Once all fragments have been received, the following processing MUST be carried out once:

1. Purge old entries from the Topology Set (see Section 10.3);

2. Purge old entries from the Attached Networks Set (see Section 10.5);

11. Populating the MPR Set

Each node MUST select, from among its one-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 MPR nodes, will be received by all nodes 2 hops away.

Each node selects its MPR Set individually, utilizing the information in the Link Set, 2-Hop Neighbor Set and 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 Link Set, 2-Hop Neighbor Set or 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.

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 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 the addresses of the MPR Selector set (i.e. all addresses, associated with a MPR selector through the Neighborhood Address Association Set). This set MAY contain additional neighbor addresses.

12.2 Populating the Advertised Neighbor Set

The Advertised Neighbor Set contains the set of neighbor addresses, to which a node advertises links through TC messages. This set SHOULD at least contain the addresses of the MPR Selector Set (i.e. all addresses, associated with a MPR selector through the Neighborhood Address Association Set). This set MAY contain additional neighbor addresses.

Each time an address is removed from the Advertised Neighbor Set, the ASSN MUST be incremented. When an address is added to the Advertised Neighbor Set, the ASSN MUST be incremented.

13. Populating the Neighborhood Address Association Set

All OLSRv2 messages containing a Local Interface Block (including HELLO and TC messages) SHOULD be used to update the Neighborhood Address Association Set as follows:

1. If there is a Neighborhood Address Association Tuple, any of whose addresses are in the Local Interface Block being processed, then discard that tuple.

2. A tuple is added to the Neighborhood Address Association Set, where:

   * NA_neighbor_addr_list = all addresses from the Local Interface Block;

   * NA_time = current time + NA_HOLD_TIME.

14. Routing Table Calculation

The Routing Set is updated when a change (an entry appearing/ disappearing) is detected in:

- o the Link Set,
- o the Neighbor Address Association Set,
- o the 2-hop Neighbor Set,
- o the Topology Set,

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 Y is any symmetric neighbor of X (with Link Type equal to SYM), the arcs Y -> Z where Y is a neighbor node with willingness different of WILL_NEVER and there exists an entry in the 2-hop Neighbor Set with Y as N2_neighbor_iface_addr and Z as N2_2hop_iface_addr, and the arcs U -> V, where there exists an entry in the Topology Set with V as T_dest_iface_addr and U as T_last_iface_addr. The graph is complemented with the arcs W0 -> W1 where W0 and W1 are two addresses of interfaces of a same neighbor (in a neighbor address association tuple).

The following procedure is given as an example for (re-)calculating the Routing Set (with a breadth-first algorithm):

1. All the tuples from the Routing Set are removed.

2. The new routing tuples are added starting with the symmetric neighbors (h=1) as the destinations. Thus, for each tuple in the Link Set where:

   * L_STATUS == SYMMETRIC (L_STATUS is calculated as indicated in Table 1)

   a new routing tuple is recorded in the Routing Set with:

   * R_dest_iface_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_iface_addr = L_local_iface_addr of the link tuple.

3. for each neighbor address association tuple, for which two addresses A1 and A2 exist in I_neighbor_iface_addr_list where:

   * there exists a routing tuple with:

     + R_dest_iface_addr == A1

   * there is no routing tuple with:

     + R_dest_iface_addr == A2

   then a tuple in the Routing Set is created with:

   * R_dest_iface_addr = A2;

   * R_next_iface_addr = R_next_iface_addr of the route tuple of A1;

   * R_dist = R_dist of the route tuple of A1 (e.g. 1);

   * R_iface_addr = R_iface_addr of the route tuple of A1.

4. for each symmetric strict 2-hop neighbor where the N2_neighbor_iface_addr has a willingness different from WILL_NEVER a tuple in the Routing Set is created with:

   * R_dest_iface_addr = N2_2hop_iface_addr of the 2-hop neighbor;

   * R_next_iface_addr = the R_next_iface_addr of the route tuple with:

     + R_dest_iface_addr == N2_neighbor_iface_addr of the 2-hop tuple;

   * R_dist = 2;

   * R_iface_addr = the R_iface_addr of the route tuple with:

     + R_dest_iface_addr == N2_neighbor_iface_addr of the 2-hop tuple;

5. The new route tuples for the destination nodes h+1 hops away are recorded in the routing table. The following procedure MUST be executed for each value of h, starting with h=2 and incrementing

by 1 for each iteration. The execution will stop if no new tuple is recorded in an iteration.

1. For each topology tuple in the Topology Set, if its T_dest_iface_addr does not correspond to R_dest_iface_addr of any route tuple in the Routing Set AND its T_last_iface_addr corresponds to R_dest_iface_addr of a route tuple whose R_dist is equal to h, then a new route tuple MUST be recorded in the Routing Set (if it does not already exist) where:

   + R_dest_iface_addr = T_dest_iface_addr;

   + R_next_iface_addr = R_next_iface_addr of the route tuple where:

     - R_dest_iface_addr == T_last_iface_addr

   + R_dist = h+1; and

   + R_iface_addr = R_iface_addr of the route tuple where:

     - R_dest_iface_addr == T_last_iface_addr.

2. Several topology tuples may be used to select a next hop R_next_iface_addr for reaching the node R_dest_iface_addr. When h==1, ties should be broken such that nodes with highest willingness and MPR selectors are preferred as next hop.

15. Proposed Values for Constants

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

15.1 Message Intervals

- o HELLO_INTERVAL = 2 seconds
- o REFRESH_INTERVAL = 2 seconds
- o TC_INTERVAL = 5 seconds

15.2 Holding Times

- o L_HOLD_TIME = 3 x HELLO_INTERVAL
- o N2_HOLD_TIME = 3 x REFRESH_INTERVAL
- o NA_HOLD_TIME = 3 x TC_INTERVAL
- o T_HOLD_TIME = 3 x TC_INTERVAL
- o RX_HOLD_TIME = 30 seconds
- o FW_HOLD_TIME = 30 seconds
- o P_HOLD_TIME = 30 seconds
- o FG_HOLD_TIME = 30 seconds

15.3 Willingness

- o WILL_NEVER = 0
- o WILL_LOW = 1
- o WILL_DEFAULT = 3
- o WILL_HIGH = 6
- o WILL_ALWAYS = 7

15.4 Time

o C = 0.0625 seconds (1/16 second)

16. Representing Time

OLSRv2 specifies several TLVs, where time, in seconds, is to be represented via an 8 bit field.

Of these 8 bits, the highest four bits represent the mantissa (a) and the four lowest bits represent the exponent (b), yielding that:

- time = C*(1+a/16)* 2^b [in seconds]

where a is the integer represented by the four highest bits of the time field and b the integer represented by the four lowest bits of the time field. The proposed value of the scaling factor C is specified in Section 15. All nodes in the network MUST use the same value of C.

17. IANA Considerations

17.1 Multicast Addresses

A well-known multicast address, All-OLSRv2-Multicast, 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.

17.2 Message Types

OLSRv2 defines two message types, which must be allocated from the "Assigned Message Types" repository of [4]

Mnemonic	Value	Description
HELLOv2	TBD	Local Signaling
TCv2	TBD	Global Signaling

Table 7

17.3 TLV Types

OLSRv2 defines three Message TLV types, which must be allocated from the "Assigned message TLV Types" repository of [4]

Mnemonic	Value	Description
VALIDITY_TIME	TBD	The time (in seconds) from receipt of the message during which the information contained in a message is to be valid
INTERVAL_TIME	TBD	The time (in seconds) between two successive transmissions of messages of a given type
WILLINGNESS	TBD	Specifies a node's willingness [0-7] to act as a relay and to partake in network formation

Table 8

OLSRv2 defines three Address Block TLV types, which must be allocated from the "Assigned address block TLV Types" repository of [4]

Mnemonic	Value	Description
OTHER_IF	TBD	Specifies that an address is associated to an interface other than the one where the message is transmitted, and may specify its status (verified bidirectional or lost)
LINK_STATUS	TBD	Specifies a given link's status (asymmetric, verified bidirectional, lost)
MPR	TBD	Specifies that a given address is selected as MPR

Table 9

18. References

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

[4] Clausen, T., Dean, J., and C. Dearlove, "Generalized MANET Packet/Message Format", Work In Progress draft-ietf-manet-packetbb-00.txt, February 2006.

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

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

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

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

The OLSRv2 Design Team
MANET Working Group

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 2-hop interfaces by relaying through an MPR node.

There are several peripheral issues that the algorithm need 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 need to be precised in the following way:

- o All neighbor nodes with willingness equal to WILL_NEVER MUST ignored in the following algorithm: they are not considered as neighbors (hence not used as MPRs), nor as 2-hop neighbors (hence no attempt to cover them is made).

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

- o 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 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 interface of a neighbor to which there exist a symmetrical link on interface I.

N - the set of such neighbor interfaces

2-hop neighbor interface (of I) An interface of a symmetric strict 2-hop neighbor and 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 N_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 N_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:

- Some 2-hop neighbors may have several interfaces. The described algorithm attempts to reach every such interface of the nodes. However, whenever information that several 2-hop interfaces are, in fact, interfaces of the same 2-hop neighbor, is available, it

can be used: only one of the interfaces of the 2-hop neighbor needs to be covered. This information is provided in the Neighborhood Address Association Set.

- 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. Example Algorithms for Generating Control Traffic

The proposed generation of the control messages proceeds in four steps. HELLO messages and TC messages both essentially consist of a list of advertised addresses of neighbors (some part of the topology).

Hence, a first step is to collect the set of relevant addresses which are to be advertised. Because there are a number of TLVs which can be associated with each address (including mandatory ones), this step results in a list of addresses, each associated with a certain number of TLVs.

The second step is then to regroup the addresses which share exactly the same TLVs (same Type and same Value), into an address block which will be associated with a list of TLVs.

The third step is to pack the message header and message TLVs into a sequence of octets.

The fourth step consists of packing every address block obtained in the second step by finding the longest common prefix of the addresses in the address block (the head), then, packing the list of the tails of the addresses into a sequence of octets, followed by the TLVs of the address block.

This generation method can be used for TC generation and HELLO generation: in each case, all what need to be specified is the message headers, message TLVs, and the list of each address with its associated TLVs.

The Local Interface Block MUST include all of the participating interface addresses of the node (including the one of chosen as the node's originator address and included in the message header).

Appendix B.1 Example Algorithm for Generating HELLO messages

This section proposes an algorithm for generating HELLO messages. Periodically, on each interface I, the node generates a HELLO message specific to that interface, as follows:

1. First, the list of the links of the interface is collected. It is the list of the Link Tuples where:

   * L_local_iface_addr == address of the interface

   Each corresponding address L_neighbor_iface_addr is then advertised with the following TLVs:

* Type="LINK-STATUS", Value=L_STATUS, the status of the link (see Section 5.1.1);

* Type="OTHER_IF", if and only if as specified in Section 7);

* Type="MPR", if and only of the address L_neighbor_iface_addr is an interface address in the MPR Set.

2. Second, if the node has more than one interface, for each address which was not previously advertised and for which there exists a Link Tuple on another interface where:

   * L_local_iface_addr is different from address of the interface I; AND

   * L_STATUS == SYMMETRIC

   the corresponding address L_neighbor_iface_addr is advertised with the following TLV:

   * Type="OTHER_IF", Value=SYMMETRIC.

3. Third, if the node has more than one interface, for each interface address which is to be reported as LOST as specified in Section 7) the interface address is advertised with the following TLV:

   * Type="OTHER_IF", Value=LOST.

4. Then a HELLO message is generated using the previous method, with the specified headers and TLVs:

   * a message TLV with Type="VALIDITY_TIME" and Value=encoding of L_HOLD_TIME, SHALL be added

   * a message TLV with Type="INTERVAL_TIME" and Value=encoding of HELLO_INTERVAL, SHOULD be added

   * a message TLV with Type="WILLINGNESS" and Value=the willingness of the node. This SHOULD NOT be included if this value is WILL_DEFAULT, it SHALL be included otherwise.

Appendix B.2 Example Algorithm for Generating TC messages

Periodically, the node generates TC messages, broadcast on all the interfaces of the node, as follows:

1. Each A_iface_addr in the Advertised Neighbor Set, SHALL be included in the TC message.

2. The TC message is generated with the proper headers, and (except where the Advertised Neighbor Set is empty and the TC message is not specifically reporting this, see Section 9) including the message TLV, Type="CONTENT_SEQUENCE_NUMBER", Value=the current ASSN of the node.

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 [4]. However the following points should be noted.

OLSRv2 uses only packets with a packet header. Thus all OLSRv2 packets have the following layout.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0 0 0 0 0 0 0 0|   Reserved    |    Packet Sequence Number     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                            Message                            |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
:                              ...                              :
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                            Message                            |
|                                                               |
```

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

All reserved bits are also unset (zero).

OLSRv2 uses only packets with a complete message header. Thus all OLSRv2 messages have the following layout.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Message Type | Resv  |U|N|0|0|         Message Size          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Originator Address                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time To Live |   Hop Count   |    Message Sequence Number    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                    Message Body + Padding                     |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

In standard OLSRv2 messages (HELLO and TC) the U and N bits are also unset(zero). In all OLSRv2 messages the reserved bits marked Resv above are also unset (zero).

The layouts of the message body, address block, TLV block and TLV are as in [4], allowing all options. Standard (HELLO and TC) messages

contain a first address block which contains local interface addresses, all other address blocks contain information specific to the message type. Except by being first, the local interface address block is not distinguished in any way.

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 TTL 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 single local interface address, with head length 4; thus although 1 tail is indicated, no tail octets are included. This address block has no TLVs (TLV block content length 0 octets).

The second, and last, address block reports 4 neighbour interface addresses, with address head length 3 octets. The following TLV block (content length 11 octets) includes two TLVs.

The first of these TLVs reports the link status of all four neighbours 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, since 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.

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |     HELLO     |0 0 0 0 0 0 0 0|0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0|
```

```
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                      Originator Address                       |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |0 0 0 0 0 0 0 1|0 0 0 0 0 0 0 0|    Message Sequence Number    |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0| VALIDITY-TIME |0 0 0 0 0 1 0 0|
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |0 0 0 0 0 0 0 1|     Value     | INTERVAL-TIME |0 0 0 0 0 1 0 0|
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |0 0 0 0 0 0 0 1|     Value     |  WILLINGNESS  |0 0 0 0 0 1 0 0|
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |0 0 0 0 0 0 0 1|     Value     |0 0 0 0 0 1 0 0|     Head      |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                  Head (cont)                  |0 0 0 0 0 0 0 1|
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0|0 0 0 0 0 0 1 1|     Head      |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |          Head (cont)          |0 0 0 0 0 1 0 0|     Tail      |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |     Tail      |     Tail      |     Tail      |0 0 0 0 0 0 0 0|
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |0 0 0 0 1 0 1 1|  LINK-STATUS  |0 0 0 0 1 1 0 0|0 0 0 0 0 1 0 0|
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |     HEARD     |     HEARD     |   SYMMETRIC   |   SYMMETRIC   |
```

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      MPR      |0 0 0 0 0 0 0 1|0 0 0 0 0 0 1 1|0 0 0 0 0 0 1 1|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

An example TC message, using IPv4 (four octet) addresses, is as follows. The overall message length is 67 octets, the final octet is 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 4 octets containing a single TLV with semantics value 1, indicating that the TLV has no value field, or length thereof. This TLV indicates that the second and third of these addresses (indexes 1 to 2) are for other interfaces than the one on which this TC message is transmitted.

The other two address blocks contain neighbour interface addresses, with head lengths 2 and 4 respectively. The first of these, with 3 addresses, has an empty TLV block (content length 0 octets). The second, which contains 1 address, has a TLV block (content length 4 octets) with a single TLV (semantics value 4 indicating no indexes needed) indicating that this is a network address with the given prefix length (itself with length 1 octet).

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      TC       |0 0 0 0 0 0 0 0|0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
    |                         Originator Address                    |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    | Time to Live  |   Hop Count   |    Message Sequence Number    |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1| CONT_SEQ_NUM  |0 0 0 0 0 1 0 0|
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |0 0 0 0 0 0 1 0|         Value (ASSN)          | VALIDITY_TIME |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |0 0 0 0 0 1 0 0|0 0 0 0 0 0 0 1|     Value     | INTERVAL_TIME |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |0 0 0 0 0 1 0 0|0 0 0 0 0 0 0 1|     Value     |0 0 0 0 0 0 1 0|
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |             Head              |0 0 0 0 0 0 1 1|     Tail      |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |  Tail (cont)  |             Tail              |     Tail      |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |  Tail (cont)  |0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0|   OTHER_IF    |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |0 0 0 0 0 0 0 1|0 0 0 0 0 0 0 1|0 0 0 0 0 0 1 0|0 0 0 0 0 0 1 0|
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |             Head              |0 0 0 0 0 0 1 1|     Tail      |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |  Tail (cont)  |             Tail              |     Tail      |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
|  Tail (cont)  |0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0|0 0 0 0 0 1 0 0|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             Head                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0 0 0 0 0 0 0 1|0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0| PREFIX-LENGTH |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0 0 0 0 0 1 0 0|0 0 0 0 0 0 0 1|Value (Length) |0 0 0 0 0 0 0 0|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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. 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 F.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 [3] 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 F.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 F.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 F.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 G. Flow and Congestion Control

TBD

Appendix H. Sequence Numbers

Sequence numbers are used in OLSR 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.

The term MAXVALUE designates in the following the largest possible value for a sequence number.

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.

Appendix I. Contributors

This specification is the result of the joint efforts of the following contributers -- 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, 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) 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: Kenichi Mase (Niigata University), Li Li (CRC), Louise Lamont (CRC), Joe Macker (NRL), Alan Cullen (BAE Systems), Philippe Jacquet (INRIA), Khaldoun Al Agha (LRI), Richard Ogier (?), Song-Yean Cho (Samsung Software Center), Shubhranshu Singh (Samsung AIT) and the entire IETF MANET working group.

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright Statement

Copyright (C) The Internet Society (2006). This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

Acknowledgment

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