On MPR-OSPF Specification and Implementation in Quagga/GTNetS

Juan Antonio Cordero

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

On MPR-OSPF Specification and Implementation in Quagga/GTNetS

Juan Antonio Cordero

N° 6827

Février 2009

Thème COM

ISSN 0249-6399 ISRN INRIA/RR--6827--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
PARIS - ROCQUENCOURT

On MPR-OSPF Specification and Implementation in Quagga/GTNetS

Juan Antonio Cordero*

Thème COM — Systèmes communicants
Équipe-Projet Projets Hipercom

Rapport de recherche n° 6827 — Février 2009 — 37 pages

Abstract: This document analyses the MPR-OSPF current specification and compares it with the implemented version for the Quagga / Zebra routing suite, adapted for the GTNetS network simulator. It presents the relationship between Quagga/Zebra core and the GTNetS simulation framework, describes the inner architecture of the MPR-OSPF extension in the OSPF Quagga general implementation and identifies the different protocol main elements in the implemented code.

Key-words: Routing, protocol, OSPF, MANET, mobile networks, ad hoc networks, Multipoint Relaying, MPR-OSPF, implementation, network simulation, GTNetS, Quagga, Zebra, specification, comparison, analysis

* juan-antonio.cordero@inria.fr

Centre de recherche INRIA Paris – Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 Le Chesnay Cedex
Téléphone : +33 1 39 63 55 11 — Télécopie : +33 1 39 63 53 30

On MPR-OSPF Specification and Implementation in Quagga/GTNetS

Résumé : Ce document analyse la spécification du MPR-OSPF et la compare avec la version implementée dans le logiciel d'enroutement de réseaux Quagga / Zebra, adapté pour le simulateur GTNetS. Il présente aussi la relation entre le noyau Quagga / Zebra et le cadre de simulation GTNetS, décrit l'architecture intérieure de l'extension MPR-OSPF dans l'implementation générale du OSPF à Quagga et en identifie les différents éléments principaux du protocole mis en oeuvre dans le code.

Mots-clés : Enroutement, protocol, OSPF, MANET, réseaux mobiles, réseaux ad hoc, Multipoint Relaying, MPR-OSPF, implementation, simulation de réseaux, GTNetS, Quagga, Zebra, spécification, comparaison, analyse

Contents

1 Introduction

This report presents the implementation of MPR-OSPF extension of OSPF protocol for MANET networks and its correspondence to the official current specification. The versions considered for this comparison are the following:

- Version 03 of the *OSPF MPR Extension for Ad Hoc Networks*, known as `draft-ietf-ospf-mpr-ext-03.txt`[1].
- OSPF6 daemon for GTNetS, valid at February 21, 2008, over a Quagga/Zebra routing suite v0.98.5.

This report is organized as follows. Section 2 presents the structure of the examined OSPF6 daemon, its connection with the GTNetS simulation core and gives also an overview about the main issues of the implementation. Section 3 describes the elements of the MPR-OSPF data structure for a concrete network interface. Section 4, 5, 6 and 7 focus on different features of the MPR-OSPF extension: the neighbor sensing procedure and format (*Hello* protocol), the adjacency forming and handling algorithm, the Link State Advertisements (LSA) generation and flooding and the acknowledgement policy. For each feature, there is presented the specification (IETF-03) and the implemented extensions for the Quagga *ospf6d* daemon. It is also discussed the consistency between specification and implementation: the gaps between them are presented and some code modifications are suggested to solve them.

2 OSPF6 architecture overview

The OSPF6 code that this report is focusing on constitutes the implementation of the *ospf6d* daemon of the Quagga/Zebra network routing suite. This is linked to the GTNetS simulator code and its core deploys the mechanisms stated in RFCs 2328 [RFC2328] and 2740 [RFC2740]. Several additions to this initial core have adapted the implementation to the three main OSPF extensions over MANET, in particular MPR-OSPF. Description and analysis of this extension is the main goal of the report.

However, it is also worthy to present more in detail the kind of relationship among GTNetS, Quagga/Zebra and the MPR-OSPF extension, to expose the main principles of the OSPF6 core and the way that the different structures and algorithms provided in the specifications are developed.

2.1 Interaction between GTNetS and Quagga/Zebra

The Georgia Tech Network Simulator (GTNetS) offers a generalistic C++ based network simulation framework. It admits many scenarios, network configurations (in particular, wireless and mobility issues are accepted) and traffic patterns, different monitoring tools and provides support for a wide variety of OSI stack-based protocols (layers 2 to 5). Scheme in figure 1 shows the basic GTNetS architecture elements.

[1]The family `draft-ietf-ospf-mpr-ext-xx` continues the `draft-baccelli-ospf-mpr-ext-xx`, corresponding the IETF-00 to Baccelli-05. Hereafter I will refer, for comparability purposes, to the current draft ([IETF-03]) and to [Baccelli-04] as well.

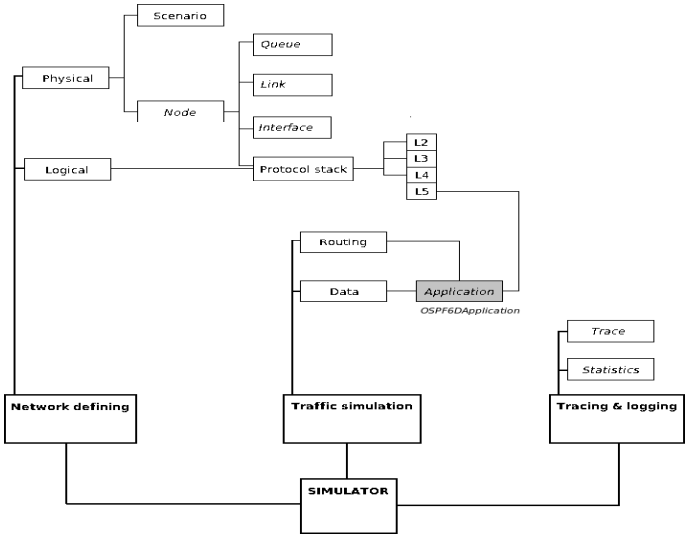

Figure 1: GTNetS architecture scheme

The Quagga project, an evolution of the GNU Zebra project, is a network routing suite for Unix that includes routing support for protocols such as RIP (v.1 and 2), OSPF (v.2 and 3) and BGP (v.4). Each of these protocols are implemented by means of a specific daemon. The daemon for OSPFv3 is called *ospf6d*. All these protocol-specific daemons are coordinated by the *zebra* daemon, which assumes IP routing handling and route redistribution among protocol daemons tasks. A diagram of the Quagga architecture is shown in figure 2 (remarked, the OSPF6 daemon).

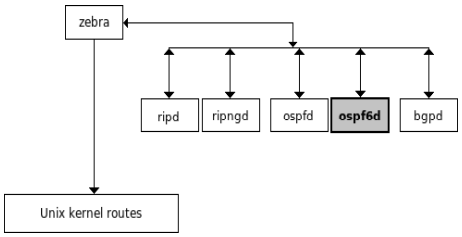

Figure 2: Quagga architecture diagram

The Quagga routing engine and the GTNetS simulation framework can interact each other in order to perform simulations involving any of the Quagga-supported routing protocols, in particular, experiments running networks based on OSPF6 or some OSPF MANET extension (e.g., MPR-OSPF). This interaction is schematically shown in figure 3. On one hand, the GTNetS layer take care of highest-level tasks (those nearest to the user), such as traffic patterns definition, scenario configuration or GTNetS-specific statistics collecting. It also manages the physical behavior of the network (propagation model), most of the

OSI stack protocols performance and, therefore, nodes lowest-level (link and network layer) communication.

On the other hand, OSPF packet processing tasks (transport-application layer) are transferred to the Quagga engine. GTNetS provides the classes `OSPF6Application` (inherited from the more general `Application` class) and `OSPF6Demux` (inherited from `L4Demux` class). OSPF6 packets are processed normally following the OSI stack protocol model and in layer 4 they are demultiplexed (`OSPF6Demux`) and transferred to the OSPF6 application (`OSPF6Application`). Objects of `OSPF6Application` class interact with the Quagga interface, in particular, with the *ospf6d* daemon.

This interaction takes part by means of the VTY (Virtual TeletYpe) interface, which allows GTNetS to configure and debug the different OSPF6 issues (interfaces, areas, global parameters) and is processed by the *ospf6d* implementation code.

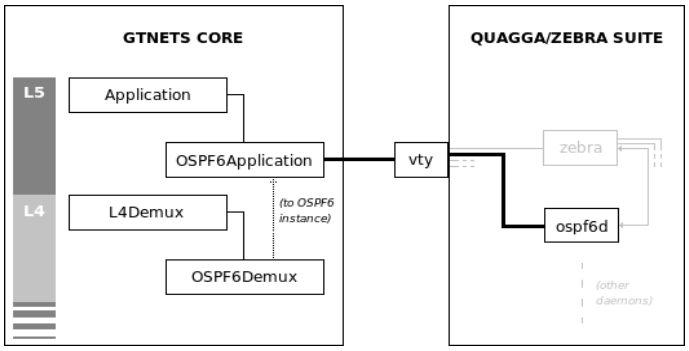

Figure 3: GTNetS-Quagga joint architecture diagram

According to this diagram, which is consistent with the standard (section A.1 [RFC2328]), OSPF-based received packets are processed by physical, link and network (IP) layers before multiplex/demultiplex and specific OSPF6 daemon processing – and reversely for transmitted packets.

2.2 OSPF6 daemon implementation

The implementation of the Quagga *ospf6d*'s implementation (OSPF6) consists of a set of C++ files that distribute elements and tasks in the way that the following list shows:

ospf6d OSPF6 daemon initialization.

ospf6_abr Support to Area Border Router (ABR) tasks, including ABR refresh for area enable/disables and summary-LSA origination.

ospf6_area Data structure of an area to which the router is attached, enable/disable router interfaces in the area and management.

ospf6_asbr Support to Autonomous System Border Router (ASBR) tasks, including LSA processing and AS-ext-LSA data structure handling and origination.

ospf6_flood Link State Advertisement (LSA) processing (as part of a LSUpdate packet[2]), flooding procedure functions (including installation in LSDB), acknowledge management.

ospf6_interface Interface data structure and management, interface states and events handling(finite machine state).

ospf6_intra LSA data structures (as parts of LSUpdate packets: Router-LSA, Network-LSA, Link-LSA, Intra-Area-Prefix-LSA), LSA generation.

ospf6_lsa LSA data structure (header, definition...) and internal management (as part of a LSDB).

ospf6_lsdb LSDB data structure and management.

ospf6_main Configuration parameters, attached VTY and finite state machine of the *ospf6d* Quagga daemon.

ospf6_message *Hello*, DBD, LSReq, LSUpdate and TLVs data structures, generic and specific (DBD, *Hello*, LSReq, LSUpdate, LSAck) send/receive functions, *Hello* packet generation and processing, LSUpdate processing.

ospf6_mdr Support to MDR/BMDR selection algorithm, including matrix calculation, tree data structure and management.

ospf6_mpr Calculation, update and management of relays and relays selectors, initialization of Path MPR / Path MPR selectors.

ospf6_neighbor Neighbor data structure[3] and management, neighbor states and events (finite machine state), adjacency management.

ospf6_network OSPF6 socket implementation, including message send/receive tasks.

ospf6_proto Inner protocol parameters (fixed and configurable constants, protocol number and version), IPv6 prefix data structure and management (see section A.4.1 [RFC2740]).

ospf6_route Data structure and management (lookup, adding and removing elements) of routes and paths in the network.

ospf6_sim_printing Notification of relays, neighborhoods, retx and push-back lists for user information proposals (in simulation scenarios).

ospf6_snmp Support for the Simple Network Management Protocol (SNMP), MIB configuration and handling.

ospf6_spf Vertex data structures for Shortest Path Tree (SPT) algorithm, calculation of the (area) SPT by implementing and adapting the Dijkstra algorithm, and construction of the routing tables.

[2]The OSPF6 implementation differentiates between the LSA as an element of a Link State Update packet (LSUpdate) for transmission/reception purposes, and units of the local Link State Database (LSDB), for interface topology information purposes.

[3]Neighbor of an interface whom the neighbor's data structure is attached to.

ospf6_top OSPF6 top data structure, implementation of *ospf6d* top router basic commands (section 8.1 [Quagga]), LSA retransmission and pushback timers management.

ospf6_zebra Implementation of *ospf6d* route redistribution tasks and interaction with the *zebra* daemon features.

The following subsections show in more detail the implementation of the main OSPF elements and procedures beyond their C++ files distribution.

2.3 Interface architecture

The interface (its data structure and the involving tasks implementation) is the main center of the OSPF6 activity, that is, the element taking care of the network state, transmitting, receiving and processing the routing flooding. In OSPF, and more in particular in MPR-OSPF, a node's interface is expected to collect information about:

- General data of the interface (OSPF interface state, router priority, flags, synch status in case of MPR-OSPF extension) and OSPF context information (area whom the interface belongs, DR/BDR references).
- Network configuration parameters, including general OSPF parameters (Hello, Dead and Rxmt intervals), MANET-specific parameters (acknowledgment interval) and MPR-OSPF-specific parameters (retransmission/pushback interval, MPR-OSPF options such as the MPR Topology Reduction mechanism).
- Complete network topology map, which is maintained by means of the interface Link State Database (LSDB).
- 1- and 2-hop neighbors status, both in a general approach and a MPR-OSPF perspective (which includes relay and relay selectors lists).

All this structure is implemented in the `ospf6_interface`[ospf6_interface.h] struct, which contains several other structs addressed to other data elements such as LSDB, relays, 2-hop neighbors and so on. From these, the most relevant is the `ospf6_neighbor`[ospf6_neighbor.h] struct, dedicated to information kept by the attached interface about the corresponding 1-hop neighbors.

There is one `ospf6_neighbor`-like variable for each 1-hop neighbor in the `ospf6_interface`-like variable attached to a certain interface. This contains the following elements:

- General data of the neighbor node (OSPF neighbor state, router priority, DR/BDR references) and status as a MPR-OSPF neighbor (synch node, Path MPR / Path MPR selector or MPR selector).
- Information about the neighbor's neighbors, that is, 2-hop neighbors of the attached interface.
- Lists showing the messages waiting to be (re)transmitted from the interface to the neighbor. This includes LSA being transmitted as part of the LSDB synchronization in the adjacency forming process

(`summary_ls`), LS requests from the interface to be answered by the neighbor (`request_list`), the messages being retransmitted to the neighbor until they are acknowledged (`retrans_list`) or the received messages requiring an acknowledge from the interface (`lsack_list`). There are also stored the acknowledgments received from the neighbor (`mack_list`).

2.4 Interface & neighbor state machines

Figures 4 and 5 depict the implementation of the finite state machines (FSM) corresponding to interfaces and neighbors evolution in OSPF. They are detailed in section 9 and 10 [RFC2328], respectively. The figures show the states alias and the transition functions.

Concerning the interface state machine, state alias are attached in `ospf6_interface.h` to numbers from 0 to 8. Transitions between states are handled in the `ospf6_interface_state_change`[ospf6_interface.c] function. The Loopback state is considered (code 2) and the function `loopind` is expected to deploy the LOOPIND event, although it is not yet fully implemented. The `OSPF6_INTERFACE_LOOPBACK` and two new state identifiers (`OSPF6_INTERFACE_MAX`, 8, and `OSPF6_INTERFACE_NONE`, 0) do not receive any of the foreseen transitions in the implemented interface state machine.

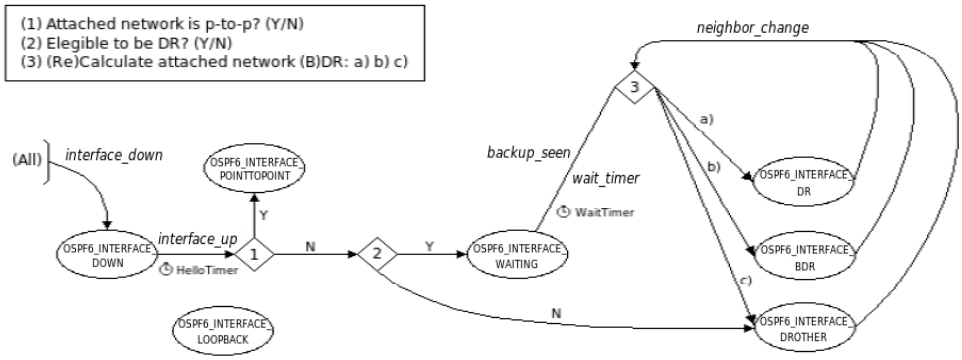

Figure 4: OSPF6 interface Finite State Machine (FSM) implementation.

Referring to the neighbor FSM implementation, state alias are attached in `ospf6_neighbor.h` to numbers from 1 to 8. The Attempt state is included, although there are not transitions provided from/to that state (it was exclusive to NBMA networks, according to the specification). Transitions between states are handled in the `ospf6_neighbor_state_change`[ospf6_neighbor.c] function. The implementation includes two additional events corresponding to the MPR-OSPF extension, in particular the MPR Adjacency Reduction mechanism (`OSPF6_MANET_MPR_ADJ_REDUC_OSPF_MPR` label enabled). These new events (not defined in classic OSPF specification as proper events and not corresponding to the prescribed behavior) allow every interface to begin adjacency forming processes with non-yet-symmetric neighbors, and are the following:

DBD received from Two-Way It is implemented in the transition function `two_way_dbdesc_recv`[ospf6_neighbor.c] and it is triggered when the neighbor state is TWO-WAY and a Database Description (DBD) packet is received from it. In that case, the neighbor passes to the EXSTART state (beginning of the adjacency forming process) and assumes the Master role in the database exchange procedure.

DBD received from Init It is implemented in the transition function `init_dbdesc_recv`[ospf6_neighbor.c] and it is triggered when the neighbor state is INIT and a DBD packet is received from it. As in the previous case, the state passes to the EXSTART state, being the neighbor the Master in the database exchange process.

It is worth to remind that [RFC2328] (section 10.6) states that DBD packets should be ignored when received from a TWO-WAY state neighbor and should trigger the TwoWayReceived event when coming from a INIT state neighbor.

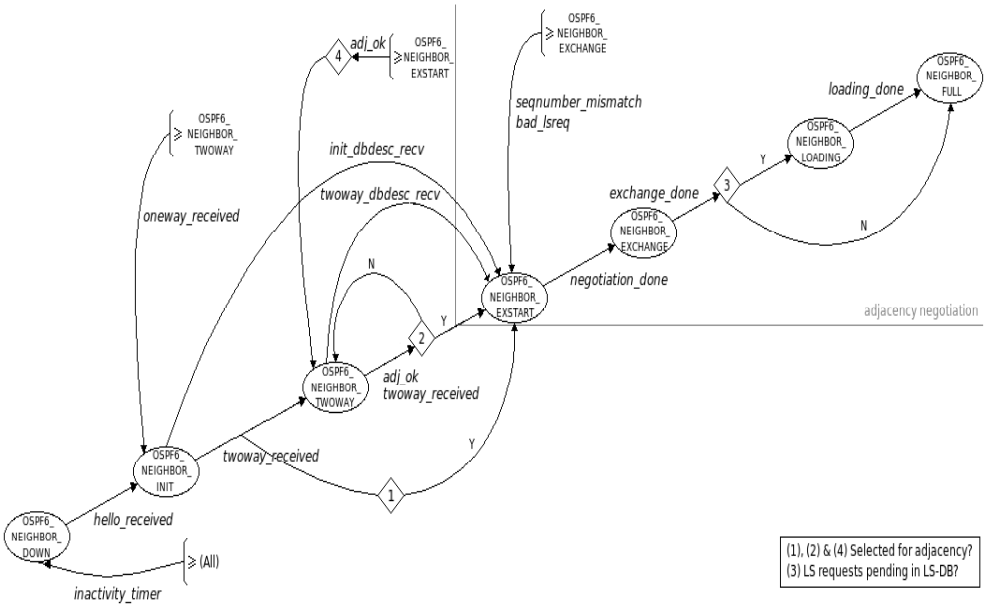

Figure 5: OSPF6 neighbor Finite State Machine (FSM) implementation (with MPR-OSPF-specific additions).

The AdjOK? event, even when it is implemented as the figure shows, is not used at all in the MPR-OSPF extension.

2.5 Message management

The message forming, transmission and processing issues involve many different functions in the OSPF6 implementation, which are placed in several code files. In fact, message transmission inner and attached features (data collection before

transmitting and data updates after processing) integrate the main core of the OSPF protocol and their MANET extensions, in particular MPR-OSPF. Figure 6 shows the basic blocks diagram of the deployed tasks in message handling.

Figure 6: Message processing blocks diagram in OSPF6.

Origination tasks (a) consist of collecting and updating information about to transfer, the 1-hop neighborhood in case of Hello packets (with different variations depending of the OSPF extension), the adjacent links or routes to external destinations in case of LSA flooding. The different LSA messages are originated in `ospf6_intra.c` (Router-LSA, Link-LSA, Network-LSA, Intra-Prefix-LSA), `ospf6_abr.c` (Inter-Area-Prefix-LSA) and `ospf6_asbr.c` (AS-ext-LSA) files.

Construction tasks (b) include functions dumping the collected data to the corresponding OSPF6 packet format so that it can be transmitted through the network. OSPF6 packet formats are detailed in Appendix A [RFC2328], modified by Appendix A [RFC2740]. Specific formats or modifications for MPR-OSPF are stated in section 7 of the specification (IETF-03). Although separation between message origination and packet construction is not always clear in the OSPF6 functions, there are specific functions in `ospf6_lsa`, which deploy part of the creation LSA tasks. In contrast, other parts of the creation procedure is developed in origination functions, both in the case of specific LSA formats (`ospf6_intra.c`, `ospf6_abr.c` and `ospf6_asbr.c` files) and the Hello packets (`ospf6_message.c` file).

Transmission (c) and reception (d) tasks are contained in `ospf6_message`. That includes the packet-specific functions for tx/rx (`ospf6_hello_send`$^{\texttt{ospf6\_message.c}}$ / `ospf6_hello_recv`$^{\texttt{ospf6\_message.c}}$, `ospf6_lsupdate_send`$^{\texttt{ospf6\_message.c}}$ / `ospf6_lsupdate_recv`$^{\texttt{ospf6\_message.c}}$ and so on) and the generic OSPF6 tx/rx functions (`ospf6_send`$^{\texttt{ospf6\_message.c}}$ and `ospf6_receive`$^{\texttt{ospf6\_message.c}}$).

Deconstruction tasks (e) are often deployed in the same function as packet reception is performed. Only in the case of individual deconstruction of LSA there is a specific function (`ospf6_receive_lsa`$^{\texttt{ospf6\_flood.c}}$) in addition to the function performing deconstruction (and reception) of LSUpdate, the LSA container packet.

Finally, packets and messages process involves many different functions. Hello processing implies update neighborhood data structures and eventually adjacencies (`ospf6_neighbor.c` file), since LSA processing includes local installation, but also new LSA or acknowledgements transmission (`ospf6_flood.c` file).

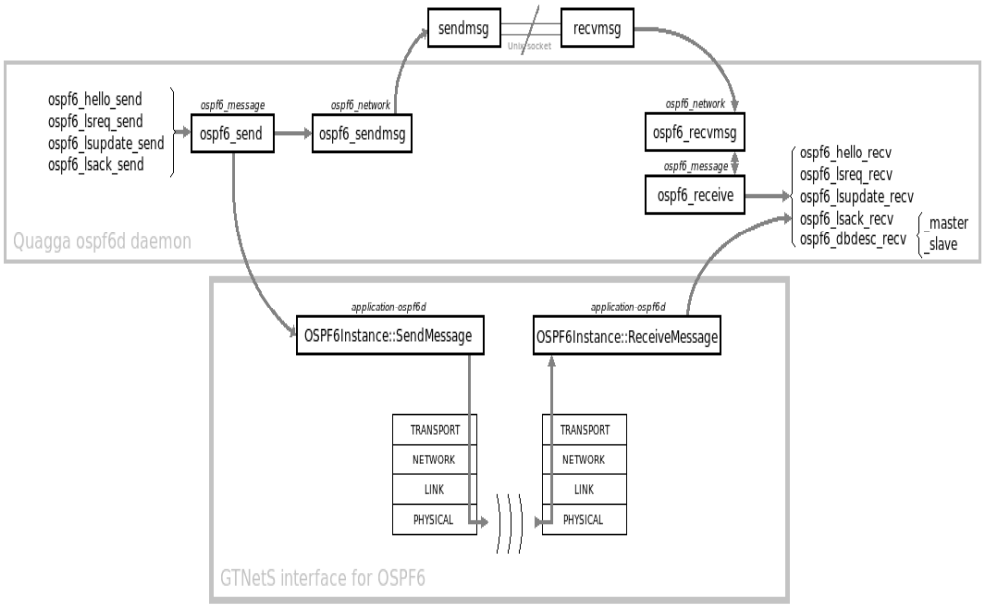

Figure 7: Message transmission diagram for GTNetS and Quagga OSPF6 implementation

Figure 7 shows more in detail the transmission/reception structure, not only the Quagga *ospf6d* daemon behavior, but also the relationship with the GTNetS simulation engine. In the simulation framework, packets are originated and created in the Quagga architecture, but are sent through the network by means of the GTNetS support for the OSI stack, not the Quagga/Zebra communication mechanism[4]. GTNetS interface to Quagga OSPF6 implementation is provided in the OSPF6Application family objects. In particular, network transmission simulation is deployed by means of the method `OSPF6Instance` :: `SendMessage`$^{\texttt{application-ospf6d.cc}}$. This method sends the packet from layer 5 of the transmitting node to the layer 3 processing protocol, where it is forwarded for encapsulation in lower OSI stack layers of the transmitting node and then sent to the receiving node(s). In each receiving node, the packet is treated in the corresponding OSI stack layers before layer 4 OSPF6 demultiplex (`OSPF6Demux` :: `Demux`$^{\texttt{ospf6demux.cc}}$) and specific OSPF6Application handling by means of `OSPF6Instance` :: `ReceiveMessage`$^{\texttt{application-ospf6d.cc}}$ method. This method transfers message management to *ospf6d* daemon, in concrete to the function corresponding to the received packet type.

It is worthy to remark that Quagga provides its own way for transmission/reception of OSPF6 routing messages. Functions `ospf6_sendmsg`$^{\texttt{ospf6\_network.c}}$ and `ospf6_recvmsg`$^{\texttt{ospf6\_network.c}}$ are able to communicate each other by using the Unix socket features (`sendmsg` and `recvmsg`)

[4]When handling routing messages transmission and reception in a computer network, Quagga uses the `ospf6_sendmsg`$^{\texttt{ospf6\_network.c}}$ and `ospf6_recvmsg`$^{\texttt{ospf6\_network.c}}$ functions, which rely on `sendmsg` / `recvmsg` Unix mechanisms. Connection between them in figure 7 does not illustrate the *loopback* internal connection, but the logical Unix socket between different routers.

tasks). But the functionalities provided by these functions are not used when the Quagga structure is addressed to simulation purposes. As it is shown in the figure, message transmission inside the simulated network flows exclusively through the GTNetS interface to Quagga and the OSI protocols stack, excluding the Unix socket processing performed by `ospf6_recvmsg`[ospf6_network.c] and `ospf6_receive`[ospf6_message.c]. Instead of this, packet distinction is performed in the `OSPF6Instance :: ReceiveMessage`[application−ospf6d.cc] method, from where the received data are directly forwarded to the packet-specific processing function.

3 Data structure and selection procedures

The MPR-OSPF data structure is detailed in section 5.1 of the specification, and consists of the following sets:

N List of the symmetric 1-hop neighbors of the interface. This list corresponds to the neighbors belonging to the `neighborlist` pList in `ospf6_interface`[ospf6_interface.h] struct whose state is higher than or equal to TWO-WAY[5]. Every member of this pList is a `ospf6_neighbor`-like variable (`ospf6_neighbor.h`), and contains a state variable, the router_id (corresponding to 1_HOP_SYM_id in the draft) and the timestamp (corresponding to 1_HOP_SYM_time), among many other parameters.

N2 List of the symmetric 1-hop neighbors of the nodes in *N*, that is, the 2-hop neighbors of the current interface, without considering the node itself and the members of *N*. The list is reachable from two ways: on one hand, in the `two_hop_list` pList in `ospf6_interface`[ospf6_interface.h] struct, with every member being an `ospf6_2hop_neighbor`-like variable (`ospf6_neighbor.h`); and in the other hand, in the `two_hop_neighbor_list` pList attached to every `ospf6_neighbor`-like variable, collecting the elements of N2 having a 1-hop distance to the corresponding neighbor.

Flooding MPR set The subset of 1-hop neighbors selected by the current interface as relays. The set is implemented in the `relay_list` pList in `ospf6_interface`[ospf6_interface.h] struct, with every member being an `ospf6_relay`-like variable (`ospf6_interface.h`). Each variable contains the router_id (corresponding to the Flooding_MPR_id in the draft) and the expire time (corresponding to the Flooding_MPR_time parameter in the draft).

Flooding MPR selector set The subset of 1-hop neighbors selecting the current interface as relay. This is listed in the `relay_sel_list` pList in `ospf6_interface`[ospf6_interface.h] struct, with every member being an `ospf6_relay_selector`-like variable (`ospf6_interface.h`). Each variable contains the router_id (corresponding to the Flooding_MPR_SELECTOR_id in the draft) and the expire time (corresponding to the Flooding_MPR_SELECTOR_time parameter in the draft). Neighbors selected as MPR selectors enable the boolean parameter `isMprSelector` in the `ospf6_neighbor` struct.

[5] That is, interface and neighbor have at least a symmetric relationship.

Path MPR set The subset of 1-hop neighbors providing shortest paths from each of the 2-hop neighbors to the origin router. Neighbors selected as *Path MPR* by a concrete interface enable the boolean parameter `adv_MPR` in their own `ospf6_neighbor`-like variable attached to the corresponding interface.

Path MPR selector set The subset of 1-hop neighbors selecting the current router as *Path MPR* relay. A neighbor belonging to the Path MPR selector set enables the boolean parameter `adv_MPRS` in its `ospf6_neighbor`-like variable attached to the current interface.

MPR set and selector set The MPR (selector) set is the joint of the Flooding MPR (selector) sets and the Path MPR (selector) set of the router.

Maintenance of *N* and *N2* is performed in the function reacting to the neighbor state change (`ospf6_neighbor_state_change`[ospf6_neighbor.c]), either explicitly for *N*, either by calling an auxiliary function (`ospf6_ospf_mpr_update_2hop_neighbor_list`[ospf6_neighbor.c]) for *N2*. Relay-related sets are based on the `ospf6_calculate_relays`[ospf6_mpr.c] function, which is called immediately before a Hello message is sent, then periodically each *HelloInterval* seconds.

The neighborhood lists (*N* and *N2*) and the Flooding MPR (selector) sets have an interface scope; there is one of them per interface. In contrast, there is only one Path MPR (selector) set per router. Since the implementation point of view, there is no difference since a single interface is provided per node, on one side, and the assumed hop metric makes equivalent the Path MPR and the Flooding MPR computations (see subsection 3.2), on the other side.

3.1 Flooding MPR selection procedure

A working heuristic for the Flooding MPR selection algorithm is detailed in Appendix A of the specification (IETF-03). The main steps are the following:

1. **Input:** *x*, *N*, *N2*.
   Let x be the router performing the Flooding MPR selection, N be the set of 1-hop neighbors of x and *N2* the set of 2-hop neighbors of x.

2. If there exist 2-hop neighbors only reachable from x certain 1-hop neighbors, these 1-hop neighbors are added to the Flooding MPR subset.

3. After that, the algorithm includes in the Flooding MPR set those 1-hop neighbors with best (greater) *reachability*[6], until every 2-hop neighbor in N2 is covered by the Flooding MPR set.

4. **Output:** $MPR(x, N, N2) \in N$, verifying that every 2-hop neighbor of x is covered by -at least- one member of $MPR(x, N, N2)$.

This is implemented in function `ospf6_calculate_relays`[ospf6_mpr.c].

[6]Reachability of a 1-hop neighbor n is the number of 2-hop neighbors non-yet-reached by the Flooding MPR set, which are reachable by n.

3.2 Path MPR selection procedure

Every node selects a Path MPR set among its 1-hop neighbors. Members of the Path MPR set provide shortest-paths to 2-hop neighborhood of the performing node. Selection procedure is detailed in Appendix B of MPR-OSPF specification (IETF-03), and its basic points are presented in the following:

1. **Input:** *x*, *N*, *N2*.
   Given A and B two 1-hop neigbhors, let *cost(A, B)* be the cost of the direct link between A and B. Given A and C be two 2-hops neighbors, let *dist(A, C)* be the cost of the shortest-path between A and C.

2. The router calculates the *router cost matrix* (RCM), which cointains link costs among neighbors (either 1- or 2-hop neighbors). Figure 8 shows the scheme of the RCM. It is worthy to remark that, since radio links are not necessarily bidirectional, the RCM matrix is not necessarily symmetric ($cost(i,j) \neq cost(j,i)$).

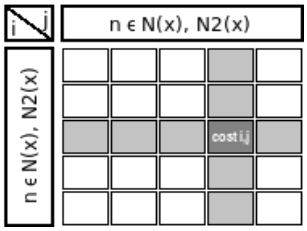

Figure 8: Router Cost Matrix for a router x, detailed in the specification (IETF-03)

3. The following subsets, $N' \in N$, $N2' \in N2$, are calculated:
   $N' = \{n \in N | cost(x,n) = dist(x,n)\}$
   $N2' = \{n \in N, N2 | n \notin N', \exists m \in N' : cost(n,m) + cost(m,x) = dist(n,x)\}$

4. The router runs the MPR selection procedure (see previous subsection) with arguments *x*, *N'* and *N2'*.

5. **Output:** $Path\_MPR(x,N,N2)$, providing shortest-paths from 2-hop neighbors to x (reverse shortest-paths).
   $Path\_MPR(x,N,N2) = MPR(x,N',N2')$

The Path MPR set of a node x is determined by running the MPR selection procedure over the sets *N'* and *N2'* and the result is a subset of *N*, not necessarily the same as (neither included in or including) the Flooding MPR selection set. Both sets (Path MPR and Flooding MPR relays) would match together in the case that all link costs had a unity (1) value. In that case, sets *N'* and *N2'* would be equivalent to *N* and *N2*, respectively.

$N' = \{n \in N | cost(x,n) = dist(x,n)\} = [cost(x,n) = 1, dist(x,n) = 1] = \{n \in N | 1 = cost(x,n) = dist(x,n) = 1\} = \{n \in N\} = N$

$N2' = \{n \in N, N2 | n \notin N', \exists m \in N' : cost(n,m) + cost(m,x) = dist(n,x)\} = [N = N'] = \{n \in N2 | \exists m \in N : cost(n,m) + cost(m,x) =$

$dist(n,x)\} = [dist(n,x) = 2, cost(n,m) = cost(m,x) = 1] = \{n \in N2 | \exists m \in N : 1 + 1 = 2\} = \{n \in N2\} = N2$

If link costs are not equal to 1, then Path MPR and Flooding MPR are different sets both included in *N*. Figure 9 shows an example of this situation.

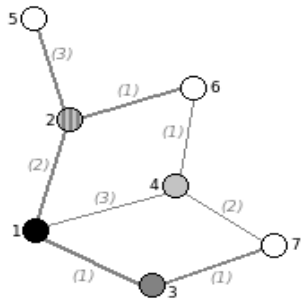

Figure 9: Network example: Flooding MPR set and Path MPR of node 1 are different. *MPR nodes are the dark grey ones (2 and 3), Path MPR nodes are the light grey ones (2 and 4). White nodes are 2-hop neighbors of node 1. Thick grey lines show shortest-path links to 2-hop neighbors.*

Node 1 (black) performs Flooding MPR and Path MPR calculation among 1-hop neighbors; the white nodes (5, 6 and 7) are 2-hop neighbors of node 1. Link costs are advertised with parenthesys. According to the algorithms shown in this subsection and the previous one, $MPR(1) = 2, 4$ and $Path\_MPR(1) = 2, 3$. That is, node 3 would be selected as Path MPR because it provides the shortest-path (indicated by a thick grey link line) to node 7.

The Path MPR selection procedure is not yet implemented in MPR-OSPF specification (IETF-03). Instead of it, every 1-hop neighbor selected as Flooding MPR of a certain node is marked also as Path MPR of that node, and then advertised in flooded LSA. This is done in the functions `ospf6_refresh_relay_list`[ospf6_mpr.c] (for relays) and `ospf6_refresh_relay_selector_list`[ospf6_mpr.c] (for selectors). In both cases the function enables the corresponding Path MPR variable in `ospf6_neighbor` struct (`adv_MPR` or `adv_MPRS`, depending on whether it is a relay or a selector). Therefore, the relays that are included in the originated LSA (which are explicitly selected in `ospf6_router_lsa_originate`[ospf6_intra.c]) are those with either `adv_MPR` or `adv_MPRS` enabled but, in fact, the function is taking those selected as Flooding MPR relays or selectors. This makes sense, or at least has no effect in terms of network performance, because OSPF6 and GTNetS simulation framework assume an unitary cost for every link in the network, and then Path MPR set can be identified with Flooding MPR set of the performing node.

4 *Hello* protocol

4.1 Specification (IETF-03)

The *Hello* protocol is detailed in section 5.2 of the MPR-OSPF draft.

The packet base format is specified at sections A.3.2 of RFCs 2328 and 2740. This classic OSPF structure is complemented by the addition of the following TLV[7] pieces:

TLV type 3 (fig. 1 section 7 draft [IETF-03]) This states the router's willingness, the number of symmetric neighbors and Flooding MPR relays listed in the *Hello* packet. According to subsection 5.2.3 of the draft (IETF-03), symmetric neighbors are prior to asymmetric ones in the list, and Flooding MPR relays should be placed before symmetric non-MPR neighbors.

TLV type 4 (fig. 2 section 7 draft [IETF-03]) This lists the metric cost of each of the links to the neighbors previously listed in the packet, according to subsection 5.2.4 of the draft (IETF-03).

TLV type 5 (fig. 3 section 7 draft [IETF-03]) The last TLV contains information concerning the Path MPRs selected by the sending interface. This includes a list of the neighbors elected as Path MPR relays, and the cost of the links connecting to each of them (see subsections 5.2.4 and 5.2.6 [IETF-03]).

Between the classic OSPF *Hello* format and the TLVs addition (LLS data), there is included a LLS header (32 bits) containing the checksum (16 bits) computed in the way that RFC 1071 states, and the added TLVs' length (16 bits). This follows the specification of RFC 4813 *OSPF Link-local Signaling*. The complete *Hello* packet format is shown in figure 10.

[7] Type-Length-Value data unit, to be attached to *Hello* packets format.

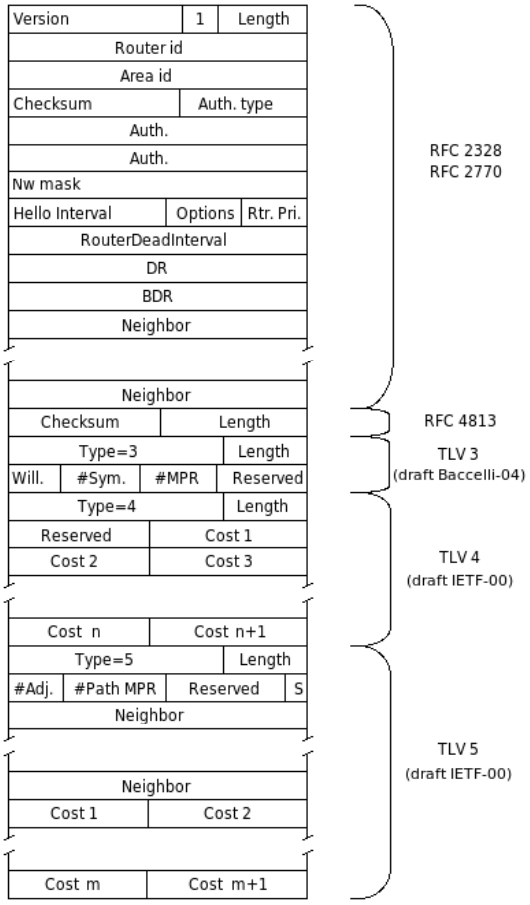

Figure 10: MPR-OSPF HELLO packet format detailed in the specification (IETF-03)

When a node receives a *Hello* packet, the N and N2 data structures should be updated in the receiving interface. If these structures change due to new *Hello* data, Flooding MPR and Path MPR sets have to be recomputed. Data changes in Flooding MPR or Path MPR lists from a 1-hop neighbor should make the receiving interface to update its own Flooding MPR and Path MPR selector sets.

Beyond the specific details in the draft, the expected behavior of an interface when receiving a *Hello* packet coming from a symmetric neighbor affects the different data structures in the following way:

N The transmitting neighbor is the only member of N that could be affected with the *Hello* packet reception. If it doesn't recognize the receiving interface as a neighbor, its state as neighbor of the receiving interface should fall to ONE-WAY.

N2 List of 2-hop neighbors can change due to changes in the 1-hop neighborhood of the *Hello*-transmitting neighbor, so N2 should be recalculated

if a received *Hello* shows changes in the interface's 2-hop neighborhood. However, this changes would be restricted to those 2-hop neighbors being also 1-hop neighbors of the *Hello*-transmitting node, so the recalculation should be constrainted to that scope, if possible.

Flooding MPR / Path MPR Since these sets depend on 2-hop neighbors, the same rule of the previous case can be applied. That is, changes in 2-hop neighborhood noticed by a 1-hop neighbor of the interface must cause Flooding MPR / Path MPR recalculation, but only restricted to decide whether the *Hello*-transmitting node should became or not an MPR / Path MPR relay.

Flooding MPR / Path MPR selectors *Hello* reception can only cause changes in the transmitting neighbor status as selector.

4.2 Implementation

Hello packets are generated, sent and received in the OSPF6 implementation for GTNetS following the specifications and format detailed in draft Baccelli-04, not [IETF-03] (figure 4 shows the implementated Hello packet structure).

4.2.1 Packet generation and transmission

Hello packet generation is deployed in the `ospf6_hello_send`[ospf6_message.c] function, which forwards to `ospf6_ospf_mpr_hello_send`[ospf6_message.c] in the case of MPR-OSPF extension. This function states the Hello packet header, creates and orders the neighbor list (MPR-Other symmetric-Asymmetric) by calling `ospf6_ospf_mpr_create_neighbor_list`[ospf6_message.c], adds the corresponding LLS header (function `ospf6_append_lls_header`[ospf6_message.c]), adds the Flooding MPR TLV type 3 (function `ospf6_append_flooding_mpr`[ospf6_message.c]) and adds the old Path MPR TLV type 4 corresponding to draft Baccelli-04, which contains the number of adjacent and Path MPR relays and a new list of adjacent and Path MPR neighbors, including direct and reverse cost to each of the links (in the `ospf6_append_path_mpr`[ospf6_message.c] function).

Before sending a *Hello* packet, the sending interface updates its relays (via `ospf6_calculate_relays`[ospf6_mpr.c] and `ospf6_refresh_relay_list`[ospf6_mpr.c] functions) and selectors (`ospf6_refresh_relay_selector_list`[ospf6_mpr.c]), both Flooding MPR and Path MPR. The `ospf6_refresh`-like functions update the relays (monitoring the timestamp, basically) and select automatically as Path MPR relays/selectors these active (non-expired) Flooding MPR relays/selectors; the specific Path MPR algorithm (Appendix B [IETF-03]) has not yet been developed.

4.2.2 Reception and processing

Reception of *Hello* packets is implemented in function `ospf6_hello_recv`[ospf6_message.c], redirected to `ospf6_ospf_mpr_hello_recv`[ospf6_message.c] for MPR-OSPF extension. There are performed some basic checks (consistency of *HelloInterval*, *RouterDeadIn-*

terval and E-bit[8]), then the TLVs are processed by calling a specific function (`ospf6_mpr_process_TLVs`[ospf6_message.c]) that details the kind of relationship between the neighbor and the interface (dumped into `twoway`, `seeMeAdj`, `isMprselector` and `isSynchNode` variables).

It is also updated the own neighbor status as a selector.

The different nodes' lists of the interfaces are also updated. If the received *Hello* packet shows changes in 1-hop neighbors of the transmitting neighbors (that is, 2-hop neighbors of the receiving interface), 2-hop neighbors linked to transmitting node are updated by calling the `ospf6_ospf_mpr_update_2hop_neighbor_list`[ospf6_neighbor.c] function. If this actualization implies a 2-hop neighbor addition or delete, variable `mpr_change` is enabled, and then the relays are recalculated by calling the `ospf6_calculate_relays`[ospf6_mpr.c] function.

In summary, *N*, *N2*, *Flooding MPR* and *Flooding MPR selectors* are correctly updated due to changes in the *Hello* packet lists. *Path MPR*-related sets (relays and selectors) are updated in the same way that those corresponding to the *Flooding MPR* sets (functions `ospf6_refresh_relay_list` and `ospf6_refresh_relay_selector_list`). The selector status is updated when receiving the *Hello* packet, the relays are updated before sending a new *Hello* packet.

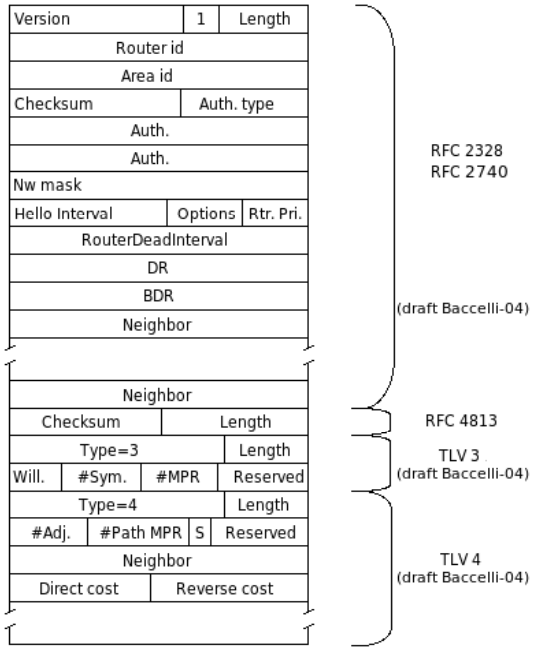

Figure 11: MPR-OSPF HELLO packet format implemented in GTNetS (corresponding to draft Baccelli-04)

[8]The E-bit (for *ExternalRoutingCapacity*) is enabled in interfaces belonging to areas able to become stub areas, and disabled in interfaces belonging to the backbone and non-stub areas (see section 4.5 [RFC2328]).

Some checks are finally done to evaluate the modifications related to interface status, such as Router Priority, DR and BDR references. Depending on evolution in these parameters, changes in the interface state machine are planned.

5 Adjacencies

5.1 Specification (IETF-03)

Current MPR-OSPF specification (section 5.3) provides two main ways to become adjacent over MANET interfaces:

- A node becomes at least adjacent to its MPR set and its MPR selectors[9], which are part of its 1-hop neighborhood.
- In addition, certain routers, called *synch* routers, become adjacent to all their 1-hop neighbors. A router elects itself as a synch router if and only if its id is higher than every id of its 1-hop neighbors and the nodes advertised in the Link State Database (LSDB).

The *synch* router dessignation is a supplementary strategy that is added to the main one (inherited from OLSR) in order to assure connection in the adjacency set, according to the following results.

Lemma Let's assume that a network adjacency set is split into several (two or more) disconnected subsets. Then, the 1-hop neighborhood of every network node contains at least one member of each disconnected adjacent subset (*i.e.*, every disconnected adjacent subset is dense in the network).

Proof *Let's call N a node belonging to the adjacency set (and, in particular, to one of its disconnected subsets). It can be proved that, in fact, every node is connected (that is, there is a 1-hop distance) with the N's adjacency subset by induction on the distance* k *(in hops) from a generic node to N. The cases of k=1 and k=2 are trivial by the MPR definition. Assuming that every node* k*-distant from N is connected (1-hop) to the adjacent subset containing* N *(induction hypothesis), it is immediate to show that a* (k+1)*-distant node belongs to the 2-hop neighborhood of a certain node in the MPR connected set, and again by MPR definition, it is reachable through a 1-hop neighbor that belongs also to the adjacent connected subset. Therefore, its distance to the adjacent subset is 1 and it can be concluded that every connected adjacent subset is dense in the network.—Q.E.D.*

Corollary The election of an only *synch* router, that is, a router that becomes adjacent to all its 1-hop neighbors, guarantees the connection of the MPR adjacency set.

The formed adjacencies should be kept as long as possible although one neighbor falls from the FULL state. Only in the case that the relationship falls below the TWO-WAY state, the adjacency is allowed to break. When this occurs and the linked interface notices (*e.g.*, due to *Hello* packet reception), the

[9]MPR (selector) set includes Flooding MPR (selector) set and Path MPR (selector) set.

neighbor should take an EXSTART (if it was a MPR Selector) or TWO-WAY state (otherwise).

In addition to this, MANET interfaces are allowed to send routing packets (that is, *Hello* packets, LS Updates, LS Requests, LS Acknowledgements and DBD) when standing at TWO-WAY state. Nodes have to receive and process routing packets when coming from neighbors with state equal or higher than TWO-WAY.

Relaxing of the neighbor state reception condition (from EXCHANGE state in classic OSPF to TWO-WAY in MPR-OSPF) for routing protocol packets relies on the fact that MPR-OSPF cannot guarantee that routes are exclusively made up by adjacent links – at least in the first hop (section 5.3.1 of the specification [IETF-03]). Thus, processing routing packets coming from symmetric (and not necessarily adjacent) neighbors would assure that these packets would be successfully flooded over the adjacent set.

There are five packet formats in OSPF (section A.3.1 [RFC2328]): *Hello* messages, DBDesc packets, LS requests (LSReq), LS acknowledgments (LSAck) and Link State Update packets (LSUpdate). From these types, *Hello* packets are not involved in routing tasks and DBDesc, LSReq and LSAck are elements of a node-to-node synchronization process. Any of them are expected to be flooded beyond the first hop. Thus, the only packet format that could be processed when coming from a symmetric neighbor is the Link State Update container, whose LSA are expected to be flooded over the whole network.

5.2 Implementation

Adjacency creation conditions for MPR-OSPF are detailed in function `need_adjacency`[ospf6_neighbor.c]. This function, which is called when processing the event TwoWayReceived (deployed in `twoway_received`[ospf6_neighbor.c]) and the neighbor state is lower-or-equal than TWO-WAY, returns 1 in case that the adjacency is allowed and 0 otherwise. According to the specification, the adjacency is formed in the two following cases:

- One of the link endpoints is the MPR relay of the other. MPR Selector property of a neighbor (related to a certain node) is stored in the `isMprSelector` variable of the `ospf6_neighbor`[ospf6_neighbor.h] struct. MPR relays and selector assignments are deployed in the `ospf6_mpr.c` functions, in particular `ospf6_calculate_relays`[ospf6_mpr.c].
- One of the link endpoints is a *synch* router. In interfaces belonging to a *synch* router, the `synchNode` variable is enabled in the corresponding `ospf6_interface`[ospf6_interface.h] struct; and in *synch* neighbors of a given node, the `isSynchNode` variable of the corresponding `ospf6_neighbor`[ospf6_neighbor.h] struct is enabled. *Synch* router election is performed in function `ospf6_check_synch_node`[ospf6_interface.c].

For OSPF-MDR there is also a `keep_adjacency` function, which states the conditions for maintain an existing adjacency when the relationship falls below the FULL state. The special conditions in MPR-OSPF (see section 5.3.2) for adjacency conservation, in particular those preventing adjacency break in case of state higher or equal to TWO-WAY, are also implemented in function `twoway_received`[ospf6_neighbor.c]. There, neighbor state change (from FULL) is

rejected if the `seeMeAdj` variable is enabled or one of the `adv_MPR`/`adv_MPRS` variables is enabled.

The specification states that the adjacency break should be avoided in case of transitions from FULL to states higher or equal than TWO-WAY for former adjacent nodes or former MPR / MPR Selectors. Instead of this, `adv_MPR` and `adv_MPRS` variables are enabled for neighbors belonging to Path MPR or Path MPR Selector sets of the performing node. This works while Path MPRs and Flooding MPRs are equivalent, that is for uniform link cost networks (see section 3.2). In case that a metric other than the hop count is implemented, Path MPRs should be splitted from Flooding MPRs, in particular in this aspect.

Moreover, when a node receives a *Hello* packet showing that the transmitting neighbor is no longer adjacent (that is, it has lost complete LSDB synchronism with the receiving interface), it should change the neighbor state to EXSTART or TWO-WAY, depending on its previous particular Flooding MPR status. In this sense, the `ospf6_ospf_mpr_hello_recv`$^{\texttt{ospf6\_message.c}}$ function calls the TwoWayReceived event for the case of symmetric relationship between receiving interface and transmitting neighbor. In case that the previous state was FULL, but the received *Hello* indicates an adjacency breaking (the neighbor does not sees the node as adjacent), the `twoway_received`$^{\texttt{ospf6\_neighbor.c}}$ function checks the adjacency convenience (via `need_adjacency`$^{\texttt{ospf6\_neighbor.c}}$) and it changes the neighbor's state to TWO-WAY or EXSTART, depending on whether the neighbor is a MPR selector / part of a *synch* link or not.

Specification also states (section 5.3.1) that nodes should receive routing traffic when coming from neighbors in a state higher or equal than TWO-WAY. Nonetheless, the different routing packets (LSUpdate, LSReq, LSAck, without considering the *Hello* transmission) are not processed if they come from neighbors in state lower than EXCHANGE[10]. DBDesc packets are processed when coming from neighbors in state higher or equal than TWO-WAY if enabled the adjacency reduction mechanism from MPR-OSPF (in case they come from a INIT neighbor, its state upgrades to TWO-WAY and is then processed). Actually, only these packets and LSUpdate (as a LSA container) are processed from TWO-WAY neighbors: this permits to include first-hop non-necessarily-adjacent step in the flooding route.

6 LSA generation and flooding

6.1 Specification (IETF-03)

Link State Advertisements (LSA) are mainly generated in the MPR-OSPF extension in the way that OSPF general specifications (RFCs 2328 and 2740) detail. The only significant changes are due to the MPR Topology Reduction feature (section 4.2), which allows a router not to advertise adjacent neighbors, but only those selected as Path MPR (relays and selectors).

Flooding process of these LSA in MPR-OSPF becomes more different to the classic procedure. In this case, the changes mentioned in the draft (section 5.4.1) overlaps and replaces partially the mechanisms provided in the two reference RFCs. In the following, I will briefly expose the initial procedure stated in

[10]That means that the database exchange process has been at least started, after the master/slave status has been negotiated.

[RFC2328] and the modifications due to new OSPF standards or MPR-OSPF drafts releases.

6.1.1 Classic OSPF LSA flooding

Section 13 [RFC2328] states the main algorithm for reception of Link State Updates, processing and flooding of Link State Advertisements in a OSPF-based network. This is slightly modified in section 3.5 [RFC2740], in order to adapt the procedure to IPv6 conditions (in *italic* the steps affected by these changes).

1. When an interface receives a LSUpdate packet, some basic consistency checks are deployed in order to state the neighbor and area from where the LSU is coming. In case that the transmitting neighbor' relationship with receiving interface is lower than EXCHANGE, the packet is expected to be discarded, since there is no adjacency neither adjacency-forming link between them.

2. The LSUpdate packet can contain several Link State Advertisements (LSA) inside. After preliminary consistency checks, processing is done for each LSA belonging to the received LSUpdate.

   2.1 Checksum is verified, LSA discarded if verification fails.

   2.2 *LSA type is examined. If LSA type is unknown, the area is a stub area and the LSA flooding scope is the AS, or the U-bit = 1 (that is, the LSA should be flooded even with unknown type) then the LSA should be discarded.*

   2.3 *The LSA scope is examined. If it is reserved, then the LSA should be discarded.*

   2.4 Age is checked. If it is equal to *MaxAge*, there is no instance of the LSA in the local LSDB and the receiving interface have no *other*[11] neighbors in forming-adjacency state (EXCHANGE of LOADING), LSA is acknowledged to the sending neighbor and then discarded.

   2.5 The interface looks up the LSA in its own LSDB. If LSA contains newer information than the LSDB (that is, there is no instance in the LSDB or the local information is older than the received LSA), the following steps should be performed.

      2.5.1 It is verified that the LSA has been received after the minimal time interval (*MinLSArrival*) required after the last LSA processing. If not, the LSA is discarded.

      2.5.2 *Flooding procedure is started up. For each neighbor of the processing interface, the steps exposed in subsection 13.3 [RFC2328] are performed, with the definition of* eligible interfaces *shown in subsection 3.5.2 [RFC2740].*

[11] The specification defines the condition as "*none of router's neighbors are in states Exchange of Loading*" (step 4 of section 13, [RFC2328]). However, the router's neighbor that transmits the LSA is necessarily in higher-than-EXCHANGE state.

2.5.2.1 For each neighbor of the processing interface:
(a) If state is lower than EXCHANGE, ignore.
(b) If the neighbor is not completely adjacent (FULL state), examine the request list associated to the neighbor, look up an instance of the flooding LSA, eventually updating it (removing the instance in case that it is equal or newer than the received LSA, ignoring the neighbor it the received LSA is not newer than the instance)) or adding the LSA to the retransmission list for the neighbor (in case it was not found an up-to-date LSA instance in the request list).

2.5.2.2 If the LSA has not been added to any retransmission list, any neighbor required the LSA flooding.

2.5.2.3 Otherwise, if the interface received the LSA and it came from a DR/BDR neighbor, it should not be further flooded, since DR/BDR transmissions have probably reached the whole network. Flooding is also stopped in case that the interface is BDR, since the flooding responsibility goes to the DR.

2.5.2.4 In other case, the LSA should be flooded out the interface and its age updated.

2.5.3 The current (old) database instance is removed from retransmission lists to every neighbor.

2.5.4 *The received LSA is installed in the own Link State Database (LSDB), and new LSAs corresponding to this instance will not be accepted during the* MinLSArrival *time interval. The LSA is stored in three different structures depending on its scope (global OSPF data structure for AS scope, the area data structure for LSAs with area scope or Ubit enabled, and the corresponding interface data structure for LSAs with link-local scope or U-bit disabled).*

2.5.5 Acknowledgment policy is developed (for ack discussion, see next section).

2.5.6 In case of self-originated LSA, it should be made up by a new updated LSA in case that it corresponds to a LSA flooded before the last interface restart, or flushed in case that it corresponds to an LSA that the interface does not want to longer flood. This is specificated in subsection 13.4 [RFC2328].

2.6 In case that LS database contains newer information than the received on the LSA, adjacency forming process has failed and it should be restarted by running the BadLSReq event.

2.7 In case that both LSA (the received one and the existing instance in the LSDB) are the same, the receiving router should acknowledge the LSA or treat it as an implicit ack (depending on whether the LSA was requested to the receiving interface or not).

2.8 In case that the instance in local LSDB is more recent than the received, then it is possible that the received packet has wrapped its SeqNumber (and then it should be discarded). Otherwise, the copy in

the local database should be sent back to the transmitting neighbor without acknowledging the older received LSA.

6.1.2 Flooding changes in MPR-OSPF (draft IETF-03)

Section 5.4.1 of draft (IETF-03) states modifications to the classic OSPF flooding procedure detailed in the RFCs. This can be summarized as follows:

- LSA processing and flooding is allowed for packets coming from neighbors in a TWO-WAY or higher state (instead of EXCHANGE or higher state).
- Substeps detailed in section 13.3 [RFC2328] (modified by section 3.5.3 [RFC2740]) are replaced by the following procedure:
  1. If the received LSA is older than the instance in LSDB, then it should be acknowledged but not further processed.
  2. In other case, it should be assigned an scope to the LSA. Depending on this assigned scope the following default flooding algorithm is deployed:
     (a) LSA installation.
     (b) Increase of the LSA age.
     (c) *(Reserved scope)* Discard.
     (d) *(Area scope)* Flooding over all router's interfaces belonging to the area (either MANET or non-MANET).
     (e) *(Non-area scope)*
     - Flooding over all non-MANET interfaces of the router.
     - If the LSA comes from a MPR Selector[12], flooding over all router's MANET interfaces within the corresponding LSA scope.

This substitution is not consistent with the expected behavior of MPR-OSPF extension and the OSPF flooding procedure structure.

Substeps of section 13.3 [RFC2328] manage the interfaces selection procedure for flooding. This procedure is only triggered in case that received LSA data is newer than the instance in LSDB.

In contrast, the steps expected to replace them in the MPR-OSPF specification (subsection 5.4.1, [IETF-03]) modify the behavior before deciding whether the received LSA is newer than the LSDB instance or not. They involve flooding procedure modifications (some of them, due to IPv6 address format), but also preliminary checks and look-up processing. So, they should be integrated in the global flooding procedure, not only in the part restricted to section 13.3 [RFC2328] (as the specification [IETF-03] states).

Thus, next section states the complete procedure, merging the basic algorithm (RFCs 2328 and 2740) with requirements of the MPR-OSPF specific structure.

[12]The restriction makes sense because MPR-OSPF allows LSA processing (not flooding) from TWO-WAY neighbors. But it also should include the case of LSA coming from/to *synch* neighbors. This could be done by replacing the MPR Selector condition with an adjacency-forming state (at least) requirement.

6.1.3 MPR-OSPF complete flooding procedure

In order to make easier the comprehension of the definitive specification of LSA flooding and processing, the complete process for MPR-OSPF extension over MANETs is exposed. As mentioned in previous section, some aspects, in particular the foreseen algorithm steps substitution of section 5.4.1 of the draft (IETF-03) has not been literally respected; and references to DR and BDR have been supressed, since they make no sense in the framework of the MPR-OSPF extension[13]. Modifications are signalized with *italic* text format.

1. When an interface receives a LSUpdate packet, some basic consistency checks are deployed in order to state the neighbor and area from where the LSU is coming. *In case that the transmitting neighbor's relationship with receiving interface is lower than TWO-WAY, the packet is discarded.*

2. The LSUpdate packet can contain several Link State Advertisements (LSA) inside. After preliminary consistency checks, processing is done for each LSA belonging to the received LSUpdate.

   2.1 Checksum is verified, LSA discarded if verification fails.

   2.2 LSA type is examined. If LSA type is unknown, the area is a stub area and the LSA flooding scope is the AS, or the U-bit = 1 (that is, the LSA should be flooded even with unknown type) then the LSA should be discarded.

   2.3 The LSA scope is examined. If it is reserved, then the LSA should be discarded (section 3.5.1 [RFC2740], steps 2.2 and 2.3 of section 5.4.1 of the draft [IETF-03]).

   2.4 Age is checked. If it is equal to *MaxAge*, there is no instance of the LSA in the local LSDB and the receiving interface have no other neighbors (differents to the transmitting neighbor) in forming-adjacency state (EXCHANGE of LOADING), LSA is acknowledged to sending neighbor and then discarded.

   2.5 The interface looks up the LSA in its own LSDB. If LSA contains newer information than the LSDB (that is, there is no instance in the LSDB or the local information is older than the received LSA), the following steps should be performed.

      2.5.1 It is verified that the LSA has been received after the minimal time interval (*MinLSArrival*) required after the last LSA processing. If not, the LSA is discarded.

      2.5.2 Flooding procedure is started up. *Flooding interfaces set is selected depending on LSA scope.*
      - (Area scope) The flooding interfaces set consists of all router's interfaces belonging to the area (either MANET or non-MANET).
      - (Non-area scope) The flooding interfaces set consists of (a) all the non-MANET router's interface, and (b) if the LSA comes

[13] DR and BDR fields in messages are kept for OSPF compatibility, but their only play a role in interfaces of wired OSPF networks and wired OSPF interfaces in hybrid networks. None of these interfaces would run the wireless MPR-OSPF extension.

from a MPR Selector[14]*, all the MANET router's interfaces within the corresponding LSA scope.*
For each flooding interface of the set, the following steps are performed:

2.5.2.1 For each neighbor of the flooding interface:
(a) If state is lower than EXCHANGE, ignore.
(b) If the neighbor is not completely adjacent (FULL state), examine the request list associated to the neighbor, look up an instance of the flooding LSA, eventually updating it (removing the instance in case that it is equal or newer than the received LSA, ignoring the neighbor it the received LSA is not newer than the instance) or adding the LSA to the retransmission list for the neighbor (in case it was not found an up-to-date LSA instance in the request list).

2.5.2.2 If the LSA has not been added to any retransmission list, flooding is not required by any neighbor.

2.5.2.3 *Otherwise, the LSA should be flooded out this interface and its age updated.*

2.5.3 The current (old) database instance is removed from retransmission lists to every neighbor.

2.5.4 The received LSA is installed in the own Link State Database (LSDB), and new LSAs corresponding to this instance will not be accepted during the *MinLSArrival* time interval. The LSA is stored in three different structures depending on its scope (global OSPF data structure for AS scope, the area data structure for LSAs with area scope or U-bit enabled, and the corresponding interface data structure for LSAs with link-local scope or U-bit disabled).

2.5.5 Acknowledgment policy is developed (for ack discussion, see next section).

2.5.6 In case of self-originated LSA, it should be made up by a new updated LSA in case that it corresponds to a LSA flooded before the last interface restart, or flushed in case that it corresponds to an LSA that the interface does not want to longer flood. This is specificated in subsection 13.4 [RFC2328].

2.6 *In case that the LSDB information is not older (so, it is equal or newer) than the received LSA, the LSA should be acknowledged and not further processed (step 2.1 of section 5.4.1 [IETF-03])*[15].

[14]The restriction makes sense because MPR-OSPF allows LSA processing (not flooding) from TWO-WAY neighbors. But it also should include the case of LSA coming from/to *synch* neighbors. This could be done by replacing the MPR Selector condition with an adjacency-forming state (at least) requirement.

[15]This is consistent with allowing that LSA coming from non-adjacency-forming neighbors (that is, in TWO-WAY state) are processed, according to the stated in section 5.3.1 of the draft (IETF-03).

6.2 Implementation

Implementation of Link State Advertisements (LSA) management includes generating the LSA, receiving the Link State Update (LSU) packets and processing the LSA that are aggregated to a single LSU. That includes flooding over other interfaces in the neighborhood.

6.2.1 LSA generation and advertised routers

The Link State Advertisements (LSA) generation and the advertised nodes election in MPR-OSPF mainly follow the general OSPF6 policy. The only modification stated in the draft (IETF-03) relies on advertising only Path MPR and Path MPR selectors in Router-LSA (those selected as `adv_MPR` or `adv_MPRS`). That is performed in the MPR-OSPF-specific excerpt of the Router LSA generation function (`ospf6_router_lsa_originate`[ospf6_intra.c]).

6.2.2 LSU processing

In contrast, flooding and processing implementation in MPR-OSPF has many differences to the classic OSPF procedure. Figure 12 shows the main diagram of LSU and LSA processing.

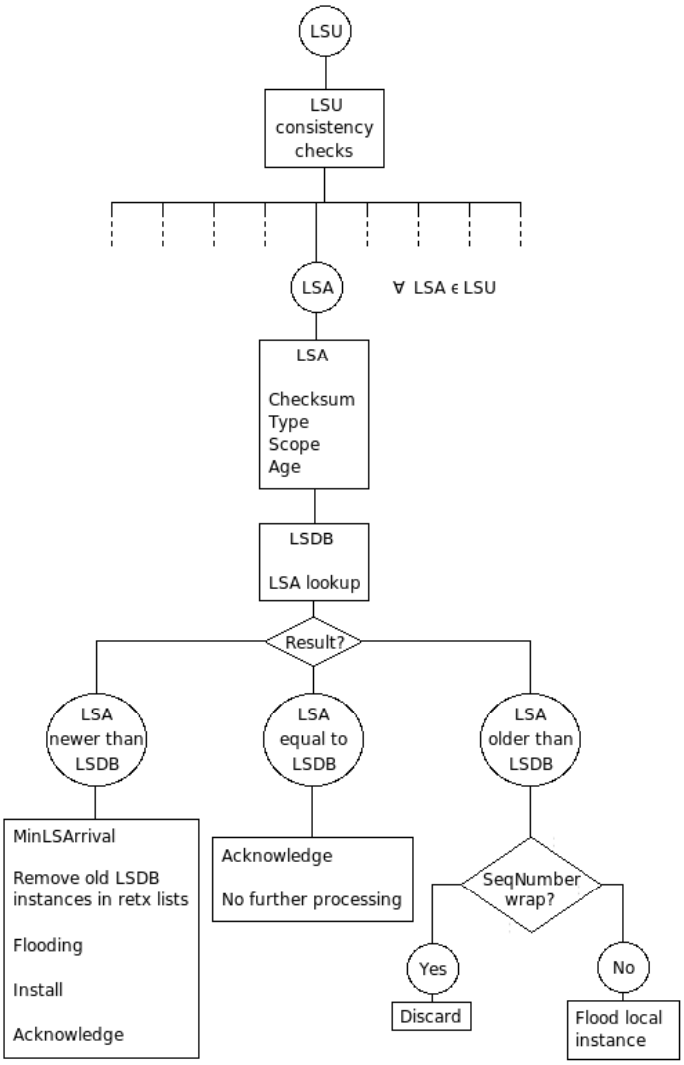

Figure 12: LSU/LSA processing diagram in the GTNetS MPR-OSPF implementation

Beyond the general reception function (function `ospf6_receive`), processing and flooding are implemented in first term in the `ospf6_lsupdate_recv`[ospf6_message.c] function. This provides the general mechanisms and verifications to be done over the Link State Update (LSUpdate) packet, such as neighbor lookup and consistency checks involving the following ones (step 1, mostly implemented in function `ospf6_header_examin`[ospf6_message.c]):

- OSPF version,
- area correspondence and

- OSPF6 instance id[16].

6.2.3 LSA processing & flooding

Each LSA belonging to the received LSU is processed independently by calling the `ospf6_receive_lsa`[ospf6_flood.c] function in the MPR-OSPF version (only enabled if the label `OSPF6_MANET_MPR_RETRANS_OSPF_MPR` is on). In first term, new verifications are performed, corresponding to checksum check (step 2.1, by calling the `ospf6_lsa_checksum`[ospf6_lsa.c]), type verification[17] (step 2.2, AS-scope LSA discarded in stub areas), scope analysis (step 2.3, LSA is discarded if its scope is reserved), age update and evaluation (via `ospf6_is_maxage_lsa_drop`[ospf6_neighbor.c], which lets the LSA, in case it is up to date or any neighbor belonging to the area is in adjacency-forming state, to be after processed in the way that step 2.4 details).

Second part of LSA processing includes look-up in the local link state database (LSDB). The LSA is stored in the acknowledge-pending list for the transmitting neighbor (function `ospf6_store_mack`[ospf6_neighbor.c]), and three scenarios are considered depending on the search results. In each of them, the list of requests from the neighbor to the main interface is examined, the received LSA is looked up. If the request is equal or older than the received LSA, the corresponding request is removed from the list.

1st scenario The received LSA contains new information not stored in the local LSDB (that is, there is no instance in the database or the instance is older). If the time interval between the two last receptions is less that *MinLSArrival*, then the new LSA is flooded (if non self-originated[18] and sent by a MPR selector), installed and acknowledged.

Flooding procedure is initiated in `ospf6_flood`[ospf6_flood.c] function and follows the diagram in figure 13.

[16] The OSPF6 instance has only a link-local meaning, so that different protocol instances can be running over the same link (defined in section A.3.1 [RFC2740]).

[17] According to [RFC2740], section 3.5.1, step (2), the LSA type evaluation should discard the LSA in case of unknown type, stub area and AS scope or U-bit on, which is not implemented.

[18] Self-originated LSA are those whose transmitting router is the same that the one processing it, according to new definition of section 3.6 [RFC2740].

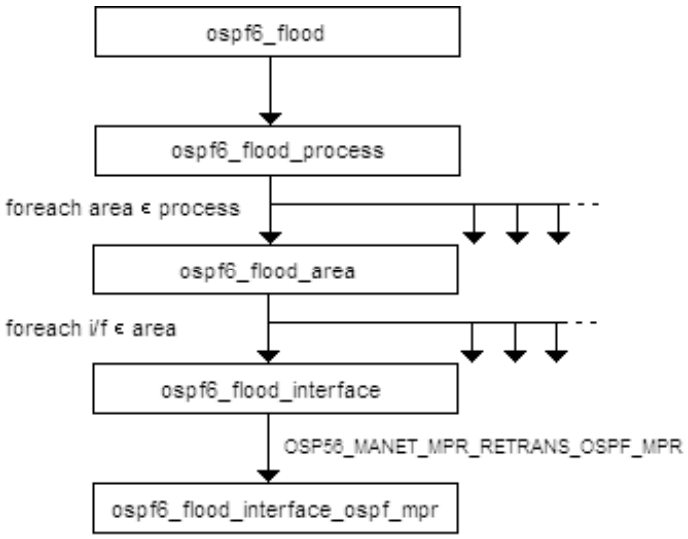

Figure 13: Flooding process functions in the GTNetS MPR-OSPF implementation

The process involves the OSPF6 instance that is running over the router of the receiving interface. The flooding process in extended to those areas connected to router's interfaces, and, in each area, the attached router interfaces manage its own flooding policy, detailed in `ospf6_flood_interface_ospf_mpr`[ospf6_flood.c] for MPR-OSPF extension.

This function discards for flooding the receiving router interfaces having received the LSA in state BDR or from an DR/BDR transmitting interface[19].

LSA is decided to be flooded over the receiving router interfaces' neighbors. For each of them, some checks are developed and the request list for the neighbor is examined in order to avoid flooding in the following cases:

- There are requests for the LSA that are newer than or equal to the LSA in the request list for the neighbor (if the request is older or equal, it is removed).
- The neighbor relationship is lower than EXCHANGE (not adjacency forming), substep 2.5.2.1 (a).
- The neighbor has generated or already transmitted the LSA.
- The LSA has been already acknowledged.

Excluding these cases, the neighbor is added to the LSA retransmission list for the neighbor (`ospf6_retrans_lsa_add`[ospf6_top.c]) and the LSA is signalized to be acknowledged (`neighborShouldAck`). Finally, the LSA

[19] It is not clear that these decisions make sense in MPR-OSPF extension, in which DR/BDR status has no place in the whole design. Moreover, the same steps are repeated at the end of the function, so either the first, either the last, either both are unneeded and could be removed.

is scheduled for retransmission (if the list is non-empty) and flooding out the interface (`ospf6_lsdb_add`[ospf6_lsdb.c]).

Install process is deployed in the `ospf6_install_lsa`[ospf6_lsa.c] function, which also includes removal of the current database instance in the retransmission lists (steps 2.5.3 and 2.5.4). Acknowledgement policy (step 2.5.5) will be detailed in next section. And self-originated LSAs, which have been prevented from flooding, generate a new instance of the LSA to be flooded[20] (step 2.5.6).

2nd scenario LSA and LSDB instance have the same age. LSA does not contain useful information to be flooded, so it is acknowledged and discarded without further processing.

3rd scenario Database instance is more recent than the received LSA. The received LSA is likely to be in *SeqNumber wrapping*[21], in this case the LSA is discarded. Otherwise, the local copy is more recent than the received LSA, and it is consequently flooded over the transmitting neighbor, if it is within an adjacency-forming LSDB exchange process (EXCHANGE or LOADING) state[22].

7 Link State Acknowledgments

7.1 Specification (IETF-03)

MPR-OSPF acknowledge policy relies on the principle that nodes belonging to the adjacency set have to acknowledge every Link State Advertisement (LSA), either explicitly or implicitly. According to this main rule, the protocol is expected to behave in the way that the flux diagram in figure 14 indicates.

[20] Flooding a new instance is one of the possible alternatives (following section 13.4 [RFC2328]), the other one is to flush the packet; in any of these cases the self-originated LSA should be flooded.

[21] The SeqNumber parameter is higher than the maximum allowed value and it has been reinitialized.

[22] 2nd and 3rd scenarios seem to be collapsed in the specification; both of them are expected to imply an acknowledgment without further processing (step 2.6). Does not MPR-OSPF needs to manage SeqNumber wrapping events, for instance? In that case, this should be removed from `ospf6_receive_lsa`[ospf6_flood.c] for MPR-OSPF. Otherwise, this should be included or mentioned in the draft.

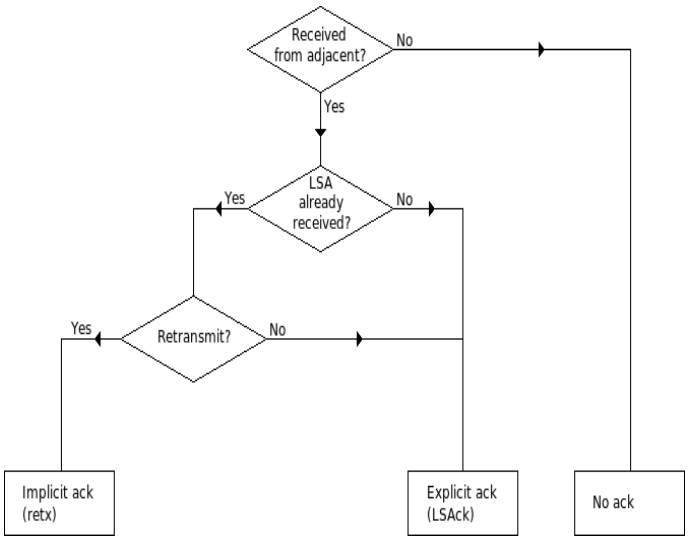

Figure 14: Acknowledge diagram in MPR-OSPF (draft IETF-03)

If an LSA has not been received from an adjacent neighbor, it should not be acknowledged. If the transmitting neighbor is adjacent, then the LSA should be acknowledged. The way it is acknowledged depends on whether the LSA is received for the first time. If not (that is, if other neighbor has sent the same LSA before), the receiving neighbor is not going to perform retransmission of the LSA through the neighborhood, so no implicit acknowledges (due to retransmissions) are going to be deployed. In that case, the receiving neighbor should acknowledge explicitly the LSA by means of an Link State Acknowledgment (LSAck) packet.

In contrast, if the LSA has been received for the first time, the neighbor will sent an LSAck in the case that no retransmissions are foreseen according to the flooding policy. If the node retransmits the LSA, then it works as an implicit acknowledgment for the transmitting neighbor.

LSAck packets may be delayed so that they can be aggregated in multiple-acks transmissions. Jitter features can also be implemented in order to reduce collision likelihood.

7.2 Implementation

Acknowledge policy is centralized in the `ospf6_acknowledge_lsa`[ospf6_flood.c] function, which derives to two different subfunctions (`ospf6_acknowledge_lsa_bdrouter`[ospf6_flood.c] and `ospf6_acknowledge_lsa_allother`[ospf6_flood.c]) depending on the OSPF6-like state of the acknowledging interface. Since MPR-OSPF extensions does not take care of DR/BDR/DROther interface states, their specific acknowledging mechanisms are only deployed in the function referring to *allother* (neither DR nor BDR) interfaces, `ospf6_acknowledge_lsa_allother`[ospf6_flood.c]).

This function mainly implements the decisions detailed in the "All-other states" column on table 19 (section 13.5) [RFC2328], with some minor modifications due to MPR-OSPF and OSPF-MDR particularities.

- If LSA has been flooded back out the receiving interface, it should not be acknowledged, according to [RFC2328]. In fact, the draft does not provide special behavior for flooded-back LSAs, though `OSPF6_LSA_FLOODBACK` flag is enabled for packets in which the processing interface is the same to the LSA generating interface. To be consistent with the specifications, both flag activation and acknowledgement floodback processing should be supressed, except that this is expected to be added to the specific policy of MPR-OSPF.

- If LSA is more recent than the instance in the LSDB, but it was not flooded back to the receiving interface (so previous step has been avoided), the interface should send a delayed acknowledgement. It is consistent with the theorical behavior (see previous subsection) on MPR-OSPF extension: in the case that the retransmission (equivalent to floodback in a wireless medium) is avoided, then the packet needs to be explicitly acknowledged, as stated in step 2.5.5 of subsection 5.1.3 of this report.

- No implied acknowledgement flags are enabled in MPR-OSPF-based routers when processing an LSA (function `ospf6_receive_lsa`[ospf6_flood.c]). In case of duplicate LSA (flag `OSPF6_LSA_DUPLICATE` enabled), that is, the received LSA is the same as the LSDB instance, the packet should be acknowledged, according to the draft (IETF-03).

- If LSA age reaches *MaxAge*, no instance is found in the LSDB and any of the router's neighbors is in forming-adjacency state (EXCHANGE or LOADING), then the LSA should be acknowledged. This is implemented directly in `ospf6_receive_lsa`[ospf6_flood.c] (and not in specific acknowledgement processing function), after age check.

With all these mechanisms, the main algorithm detailed in previous subsection is basically respected. In addition, it is worth reminding that acknowledge is restricted (in the acknowledgment handling part of function `ospf6_receive_lsa`[ospf6_flood.c]) to LSAs coming from neighbors in state higher-or-equal than EXCHANGE, following the specification (section 5.4.2, point 1).

8 Routing table and SPT calculation

Each network node is expected to maintain a unique routing table. This table maps packet destination with next hop identity, and permits each node to forward correctly a packet when it is received and addressed to a further destination.

8.1 Specification (IETF-03)

There are not explicit references to routing table construction in the current specification. According to classic OSPF policies[23], the routing table is computed taking into account the information collected by the node from the different received LSA messages[24]. The Dijkstra algorithm is run to determine the shortest path tree of every area attached to the computing node[25].

After the SPT calculation, the routing table is filled by routing entries, one per destination. Each entry stores the shortest path(s) to the corresponding destination, their (minimum) cost and the next hop(s) to reach it.

In MPR-OSPF, Router-LSAs of MANET routers contain the set of Path MPR and Path MPR selectors of the originating node. Consequently, the SPT is calculated over the Path MPR sets of each node in the network. Since Path MPR peers are selected to be intermediate nodes in the min-cost links from the 1- (when possible) and 2-hop neighbors of the computing node, the outgoing routing entries from the SPT algorithm are expected to cover at least every destination 2-hops away or further from the computing node. They are excluded, however, these 1-hop neighbors whose shortest path to the computing node is the direct link. Thus, routing entries to these 1-hop nodes are required (at least) to complete the routing table with every possible destination in the network.

Actually, MPR-OSPF nodes include neighbors belonging to *N* and *N2* to the Shortest Path Tree calculation: this should be included in the specification, since it is a specific feature from MPR-OSPF not considered (not required) in the classic OSPF documentation.

8.2 Implementation

The SPF and routing tables of a router attached to a certain area are stored inside an `ospf6_area`[ospf6_area.h] structure, in the `ospf6_route_table`-like variables `spf_table` and `route_table`. The routing table struct consists of a set of entries, each of them stored in a `ospf6_route`[ospf6_route.h]-like variable. Auxiliary data structures for the routing entries handling and the SPF algorithm are detailed in `ospf6_route.h` and `ospf6_spf.h`, and the procedures for SPT calculation and routes management are deployed in the corresponding C files.

In particular, the Shortest Path Tree calculation procedure, which adapts the Dijkstra algorithm, is implemented in `ospf6_spf_calculation`[ospf6_spf.c]. The calculation is done is two steps. In the first one, the tree root (computing router), the 1- and 2-hop neighbors of the root's interfaces in the area are added as candidates (vertices) to the tree, calculating the cost to the root[26], the number of hops and the next hop from the root. The second step iterates over the candidate list: for each candidate node, it is installed (if there is no better route) in the route table (in `ospf6_spf_install`[ospf6_spf.c]), removed from the candidate list and they are explored the link-state descriptions of its corresponding LSA. If these links are bidirectional, the involved neighbors are also added to the candidate list. When the iteration is finished (that is, no more candidates

[23] Sections 11 and 16 [RFC2328] and sections 3.3 and 3.8 [RFC2740].

[24] Router-LSAs for intra-area routes, summary-LSAs (inter-area-prefix-LSAs and inter-area-router-LSA, in OSPFv3 terminology) for inter-area routes, AS-external-routes for routes to external destinations and intra-area-prefix-LSA for prefix information.

[25] Section 16, step (2) [RFC2328].

[26] To the root or from the root?

exist), the route table of the calculating router associated to the area contains the shortest paths to every destination in the area.

9 References

[RFC2328] J. Moy: RFC 2328, *OSPF Version 2*. Internet Society (ISOC). April 1998.

[RFC2740] R. Coltun, D. Ferguson, J. Moy: RFC 2740, *OSPF for IPv6*. Internet Society (ISOC). December 1999.

[RFC4813] B. Friedman, L. Nguyen, A. Roy, D. Young: RFC 4813, *OSPF Link-local Signaling*. Internet Society (ISOC). February 2007.

[IETF-03] E. Baccelli, P. Jacquet, D. Nguyen: Internet-Draft, *OSPF MPR Extension for Ad Hoc Networks*. OSPF WG of the Internet Engineering Task Force (IETF). `draft-ietf-ospf-mpr-ext-03.txt`. November 2008. *(work in progress)*

[Baccelli-04] E. Baccelli, P. Jacquet, D. Nguyen: Internet-Draft, *OSPF MPR Extension for Ad Hoc Networks*. OSPF WG of the Internet Engineering Task Force (IETF). `draft-baccelli-ospf-mpr-ext-04.txt`. October 2007. *(expired)*

[Quagga] K. Ishiguro *et al.*: *Quagga. A routing software for TCP/IP networks. Quagga v0.98.6*. `http://www.quagga.net/docs/quagga.pdf`. June 2005.

[Evaluation] J.A. Cordero: *Evaluation of OSPF extensions in MANET routing*. Master Thesis. Équipe HIPERCOM, Laboratoire d'Informatique (LIX) - École Polytechnique. September 2007.

[GTNetS] G. F. Riley: *The Georgia Tech Network Simulator*. Proceedings of the ACM SIGCOMM 2003 Workshops. August 2003.

[SNMP] R. J. Millán: "SNMPv3", in *BIT* #139. June-July 2003.

Centre de recherche INRIA Paris – Rocquencourt
Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399