
Ecole Nationale Supérieure des Télécommunications
Département Informatique et Réseaux

Student's Project

# OSPF Link-State Database in GNU Zebra

Gerhard Münz
muenz@enst.fr

Tutor: Jean-Louis Rougier

1 July 2002

**Abstract**

OSPF (Open Shortest Path First) is a widely used link-state routing protocol for intra-domain routing, a so-called Interior Gateway Protocol (IGP). As each OSPF router takes his routing decisions on the basis of its link-state database reflecting the network topology, timeliness of this database is essential for network stability and fast convergence in case of topology changes. Present development of OSPF aims at accelerating the exchange of link-state information, which will cause an increasing number of OSPF messages received and sent by the routers. A generator of Link State Advertisements (LSAs) is needed to examine the behavior of OSPF routers under such conditions.

Within this project, the code of the free routing software GNU Zebra was examined with respect to link-state related functionality, such as originating and refreshing LSAs. The results can be used to build a Zebra based LSA generator. Moreover, a patch is presented that inserts fake AS-external-LSAs into the link-state database.

# Contents

# 1 Introduction

OSPF (Open Shortest Path First) is a widely used link-state routing protocol for intra-domain routing, which has been developed by the IETF since 1988. OSPFv2, the current version for IPv4 routing, dates back to 1991, and RFC 2328 of 1998 represents the latest revision. An OSPF version for IPv6 (OSPFv3) is described in RFC 2740. These RFCs are written in an all-embracing and very comprehensible way, such that, in addition to being the standard reference, they can serve as OSPF tutorials. Within this report, only OSPFv2 is considered, i.e. the term OSPF refers to this version only.

In practice, OSPF proved to be robust and stable under most conditions, i.e. it copes well with changes in the network topology. Though, a proof of general stability cannot generally be given.

According to the link-state principle, OSPF routers exchange messages, so-called Link State Advertisements (LSAs), in order to keep up-to-date their view of the network topology that is stored in the link-state databases. Routing stability depends on how reliable and fast the routers process the exchanged messages. Some messages have to be acknowledged, others to be "flooded" out of the other interfaces. If a topology change is signaled, the database has to be updated and the routing tables have to be recalculated by applying a shortest path algorithm.

The present development aims at speeding up convergence of the link-state databases (e.g. fast hellos [1, 2]) and extending OSPF with regard to traffic engineering [3, 4]. This will significantly increase the number of OSPF messages exchanged between the routers, resulting in higher processor loads. LSA storms can already occur under normal conditions, as each router periodically floods Link State Update packets throughout the network, and the triggering time-outs might coincide. To test the routers' behavior under such "stress" conditions, an OSPF message generator is needed.

The purpose of this project was to understand the update and refresh mechanisms of the OSPF link-state database, and to examine how theses mechanisms are implemented in GNU Zebra. Finally, a patch was to be developed that would allow to manipulate the link-state database and send out fake OSPF messages.

This report is divided into a short introduction to GNU Zebra, a section about the implementation of OSPF with focus on the link-state database, the presentation of a patch originating arbitrary AS-external-LSAs, and a final conclusion.

# 2 GNU Zebra

## 2.1 General Description

GNU Zebra [5] is a free routing software (distributed under GNU Generic Public License). It supports TCP/IP based routing protocols like RIP, BGP and OSPF. At this time, the last release is the beta version 0.92a, 0.93 is available via the Zebra CVS repository. A commercial version of Zebra exists, that is distributed by IP Infusion Inc. [6] under the name of "ZebOS Advanced Routing Suite". Moreover, GNU Zebra is deployed in some Linux based routers [7, 8]. This shows the maturity the software project has reached, and a non-beta release can soon be expected.

## 2.2 Implementation Features

The Zebra code is characterized by high modularity; it consists of several daemons, one for each supported routing protocol, and one main daemon managing the transfer of routing information from one routing protocol to another. The main daemon also updates the kernel routing table to adapt the IP packet forwarding. Thanks to this implementation design, new routing protocols can be integrated very easily and existing ones can be developed separately.

## 2.3 Installation and Configuration

Installation notes describe how to build an executable version of Zebra. By default, the executable files are saved in /usr/local/sbin. There, you will find the main daemon `zebra` and the protocol daemons like `ripd`, `ospfd`, `bgpd` etc.

Before starting the daemons, the configuration files in /usr/local/etc must be adapted. Most important are the password settings for "view" and "enable" mode.

The main daemon zebra has to be started before any other protocol daemon. The command line parameter `-d` starts the executables as daemon background processes.

All running daemons are accessible through a virtual terminal interface called "vty" (Virtual TeletYpe interface), which provides Cisco-style commands to check and change current router settings. Therefore, Zebra can be configured without having user or super-user rights on the Zebra running machine. A connection to the vty is established via `telnet`, the default ports are 2601 for the main daemon `zebra` and 2604 for the OSPF daemon `ospfd`.

# 3 Zebra's OSPF daemon

**I wrote the following code description in accordance to my understanding; there is no warranty of correctness.**

## 3.1 Source Files

Zebra's RFC 2328 conform OSPF daemon is called OSPFD; its source code is situated in the subdirectory ./ospfd of Zebra's source directory. Table 1 gives a short description for each source file of OSPFD.

**Table 1:** OSPFD source files

| | |
|---|---|
| ospf_abr.c | ABR related functions |
| ospf_asbr.c | some ASBR functions, database for external information |
| ospf_ase.c | algorithm for AS-external route calculation |
| ospf_dump.c | log file output |
| ospf_flood.c | LSA flooding |
| ospf_ia.c | inter-area routing |
| ospf_interface.c | interface related functions |
| ospf_ism.c | interface state machine |
| ospf_lsa.c | most LSA related functions: look up, generate, originate, refresh, age, and flush router, network, AS-external, and summary LSAs |
| ospf_lsdb.c | link-state database |
| ospf_main.c | daemon main routine, initialization |
| ospf_neighbor.c | neighbor related data, link-state retransmission and request lists |
| ospf_network.c | socket calls to join and quit multicast groups |
| ospf_nsm.c | neighbor state machine |
| ospf_packet.c | receiving, queuing, and sending OSPF packets, various timers |
| ospf_route.c | routing table |
| ospf_routemap.c | functions for exchange of routing information (route map) with other Zebra components |
| ospf_snmp.c | SNMP support |
| ospf_spf.c | shortest path algorithm |
| ospf_zebra.c | interface (API) between OSPFD and the Zebra main daemon |
| ospfd.c | several functions concerning areas, virtual links, vty commands |

## 3.2 Router-LSAs and Network-LSAs

OSPF routers originate a router-LSA for each area they belong to. Router-LSAs are flooded throughout a single area only. They contain information about the router and its interfaces, including the connected subnets.

Network-LSAs are originated by the Designated Router (DR) of a broadcast network and also flooded throughout a single area. The DR does not generate any network-LSA, if there is not at least one full adjacency to another router in the network. The network-LSA lists those routers that are fully adjacent to the DR.

### Router-LSAs in OSPFD

ospf_lsa.c:ospf_router_lsa_originate(...) originates new router-LSAs, triggered by the timer thread ospf_lsa.c:ospf_router_lsa_timer(...). This timer is activated by the function ospf_lsa.c:ospf_router_lsa_timer_add(struct ospf_area*), which is called by:

ospf_flood.c:ospf_process_self_originated_lsa(...)
: when a self-originated router-LSA has been received,

ospf_interface.c:ospf_if_recalculate_output_cost(...)
: when the cost associated with a router's interface has changed,

ospf_ism.c:ism_change_status(...)
: when the status of the interface state machine has changed,

ospf_lsa.c:ospf_router_lsa_update_timer(...)
: when router related information has changed,

ospf_nsm.c:nsm_change_status(...)
: when the status of the neighbor state machine has changed,

ospfd.c:[no_]area_shortcut_cmd
: when the shortcut propriety of an area has been changed,

ospfd.c:ospf_area_type_set(...)
: when the area type is set (default, stub, NSSA).

Figure 1 on page 9 shows the dependencies of the origination and refreshing of router-LSAs.

ospf_lsa.c:ospf_router_lsa_body_set(...) fills the generated router-LSA with information extracted from the corresponding ospf_area structure. This structure contains a list of all interfaces of the router that participate in this area. Thus, manipulating router-LSAs requires changes in area data, or more precisely in the associated interface data; they cannot be faked directly without substantial changes in the OSPFD code.

#### Network-LSAs in OSPFD

ospf_lsa.c:ospf_network_lsa_originate(...) originates new network-LSAs, triggered by the timer thread ospf_lsa.c:ospf_network_lsa_refresh_timer(...). This timer is activated by the function ospf_lsa.c:ospf_network_lsa_timer_add(struct ospf_interface*), which is called by:

ospf_flood.c:ospf_process_self_originated_lsa(...)
: when a self-originated network-LSA has been received,

ospf_ism.c:ism_change_status(...)
: when the status of the interface state machine has changed,

ospf_nsm.c:nsm_change_status(...)
: when the status of the neighbor state machine has changed.

Figure 2 on page 11 shows the dependencies of the origination and refreshing of network-LSAs.

ospf_lsa.c:ospf_network_lsa_body_set(...) generates a list of all routers in the network that are fully adjacent to the designated router. Therefore, it scans the neighbor list associated to the data structure of the interface that is connected to the network. Thus, manipulating network-LSAs requires changes in the neighbor list of the corresponding interface, or a fake interface has to be created. Furthermore, the router originating the network-LSA is always the Designated Router of the network that keeps up adjacencies to all other routers in the network. This has to be considered in any manipulation.

## 3.3 AS-external-LSAs

Autonomous System Boundary Routers (ASBRs) announce routes to destinations outside the Autonomous System (AS). There are several sources of information about external destinations: the ASBR might belong to another OSPF AS or it

uses information of other routing protocols. Routes to AS-external destinations can also be based on static routes. An ASBR does not have to be part of the backbone, and it does not have to be Area Border Router (ABR) either. Nevertheless, it must not be placed internal to "stub" areas.

Each AS-external destination is announced by an AS-external-LSA, which is flooded throughout the entire AS except for "stub" areas. The other routers then use the AS-external-LSA to calculate the shortest path to this AS-external destination. Intra-area and inter-area paths are always preferred over AS-external paths. Therefore, AS-external-LSAs announcing destinations within the AS would be ignored during the shortest path calculation.

### AS-external-LSAs in OSPFD

AS-external-LSAs are originated by ospf_lsa.c:ospf_external_lsa_originate(struct external_info*). The following list shows where this function is called:

ospf_lsa.c:ospf_external_lsa_originate_timer(...)
: when the router ID has changed, this timer is activated by ospfd.c:ospf_router_id_update() to adapt AS-external-LSAs (non-default routes only),

ospf_lsa.c:ospf_default_originate_timer(...)
: when the router ID has changed, this timer is activated by ospfd.c:ospf_router_id_update() to adapt AS-external-LSAs announcing default routes,
when the configuration concerning the redistribution of default information has changed, this timer is activated by ospf_zebra.c:ospf_redistribute_default_set(...),

ospf_lsa.c:ospf_external_lsa_refresh_default()
: when a non-existent default AS-external-LSA is to be refreshed,

ospf_lsa.c:ospf_external_lsa_refresh_type(...)
: when a non-existent AS-external-LSA of a given type (= source of external information) is to be refreshed,

ospf_zebra.c:ospf_zebra_read_ipv4(...)
: when the Zebra daemon provides external information,

ospf_zebra.c:ospf_distribute_list_update_timer(...)
: when a distribution list (= set of external information sources) for redistribution of external information has been set or unset,

ospfd.c:no_network_area_cmd

when a network has been removed (concerns AS-external-LSAs to "connected" destinations).

ospf_lsa.c:ospf_external_lsa_body_set(...) transfers the external information from a given external_info structure into AS-external-LSA data. No information of the OSPF database is used. Moreover, AS-external-LSAs only affect the link-state database (LSDB) and the routing table. This makes it easy to insert fake AS-external-LSAs in the LSDB without disturbing any OSPF mechanisms.

## 3.4 Summary-LSAs and ASBR-summary-LSAs

Summary-LSAs and ASBR-summary-LSAs are originated by Area Border Routers (ABR) and flooded through a single area only. Summary-LSAs describe routes to networks situated in other areas of the Autonomous System (AS), whereas ASBR-summary-LSAs indicate routes to AS Boundary Routers (ASBR).

The implementation of these LSAs in OSPFD is not followed up as they are originated on the basis of other LSAs.

## 3.5 LSA Refresh Mechanisms

Link states are soft states, i.e. they have to be refreshed periodically. When the age of a LSA reaches the value MaxAge without being refreshed, the LSA is removed from the link-state database and flushed from the routing domain.

It is evident that a LSA has to be refreshed by its originating router, which has to check before if the information is still valid. This refresh mechanism has to take place before the LSA's age reaches MaxAge.

### LSA refresh mechanisms in OSPFD

OSPFD refreshes LSAs using timers. However, the LSA refresh mechanisms are different for router-LSAs, network-LSAs and other LSAs.

### Refreshing router-LSAs

ospf_lsa.c:ospf_router_lsa_refresh(...) is the refresh function for router-LSAs which is triggered by the timer thread ospf_lsa.c:ospf_router_lsa_timer(...) only. This timer is attached to the data structure of the area the router-LSA is dedicated to, and it is activated by the following functions:

ospf_lsa.c:ospf_router_lsa_install(...)

after having originated or refreshed the LSA (ensures periodical refreshing),

ospf_lsa.c:ospf_router_lsa_timer_add(...)

which first stops an already running timer if existent. This function is called if router-LSA related information has changed.

Figure 1 shows the dependencies between all functions and timers involved in router-LSA origination and refreshing.

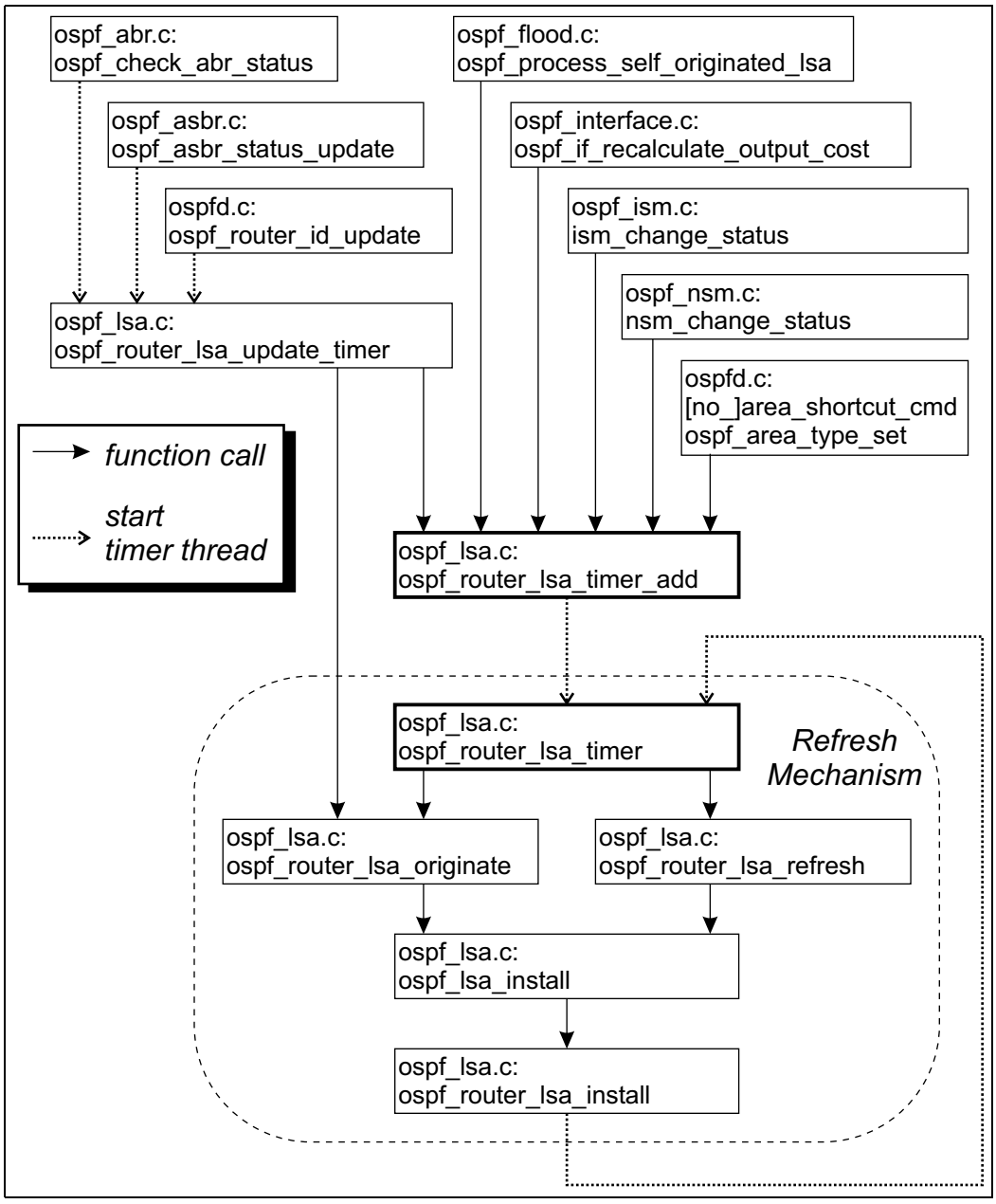


**Figure 1:** Origination and Refreshing of Router-LSAs

### Refreshing network-LSAs

ospf_lsa.c:ospf_network_lsa_refresh(...) is the refresh function for network-LSAs which is triggered by the timer thread ospf_lsa.c:ospf_network_lsa_refresh_timer(...) only. This timer is attached to the data structure of the interface the network is connected to, and it is activated by the following functions:

ospf_lsa.c:ospf_network_lsa_install(...)
: after having originated or refreshed the LSA (ensures periodical refreshing),

ospf_lsa.c:ospf_network_lsa_timer_add(...)
: which first stops an already running timer if existent. This function is called if network-LSA related information has changed.

Figure 2 shows the dependencies between all functions and timers involved in network-LSA origination and refreshing.

### Refreshing AS-external-LSAs, summary-LSAs, ASBR-summary-LSAs

A common refresh mechanism called ospf_lsa.c:ospf_lsa_refresh_walker(...) is responsible for all these LSA types. This timer thread is started during the initialization of OSPFD. For each LSA that has to be refreshed, it calls ospf_lsa.c:ospf_lsa_refresh(...) which passes on to ospf_lsa.c:ospf_external_lsa_refresh(...), ospf_lsa.c:ospf_summary_lsa_refresh(...) or ospf_lsa.c:ospf_summary_asbr_refresh(...), according to LSA type.

A LSA has to be registered by ospf_lsa.c:ospf_refresher_register_lsa(...) to be taken into account by the "refresh walker". The registration takes place when ospf_lsa.c:ospf_external_lsa_install(...), ospf_lsa.c:ospf_summary_lsa_install(...), or ospf_lsa.c:ospf_summary_asbr_lsa_install(...) respectively is called in the originate or refresh procedure.

ospf_lsa.c:ospf_refresher_unregister_lsa(...) unregisters a LSA that is discarded from the link-state data base by ospf_lsa.c:ospf_discard_from_db(...) (concerns summary-LSAs and ASBR-summary-LSAs) or flushed by ospf_lsa.c:ospf_external_lsa_flush(...) (AS-external-LSAs only).

In addition and in my opinion without any necessity, ospf_lsa.c:ospf_external_lsa_refresh(...) unregisters AS-external-LSAs before calling ospf_lsa.c:ospf_lsa_refresh(...), and registers them again afterwards.

Apart from the refresh walker, the refresh mechanism for AS-external-LSAs ospf_lsa.c:ospf_external_lsa_refresh(...) is also triggered by the following functions:

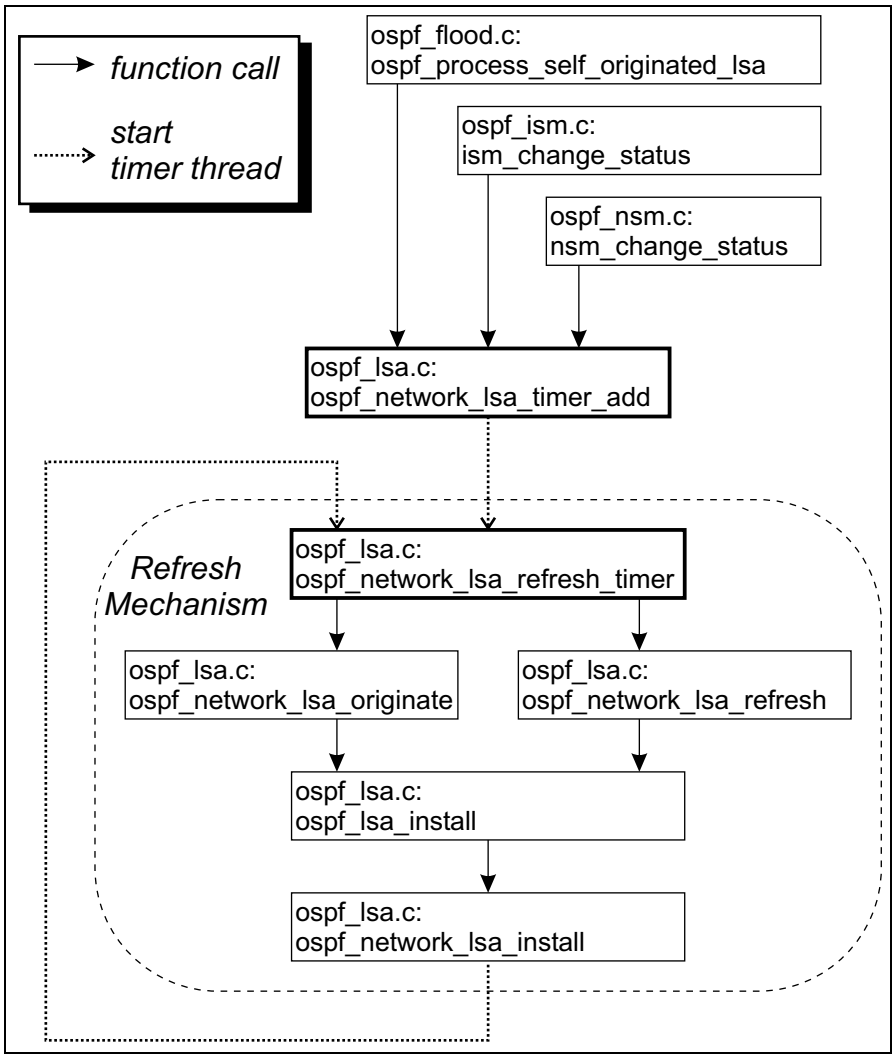


**Figure 2:** Origination and Refreshing of Network-LSAs

ospf_lsa.c:ospf_external_lsa_refresh_type()
: to refresh all AS-external-LSAs of a certain type (= source of external information),

ospf_lsa.c:ospf_external_lsa_refresh_default(...)
: to refresh an AS-external-LSA announcing an external default route,

ospf_flood.c:ospf_process_self_originated_lsa(...)
: after having received the same LSA,

ospf_zebra.c:ospf_zebra_read_ipv4(...)
: if Zebra updates an already existing external route,

ospf_zebra.c:ospf_distribute_list_update_timer(...)

when other settings concerning the redistribution of external information have changed.

Figure 3 shows the dependencies between all functions and timers involved in AS-external-LSA origination and refreshing.

Before refreshing an AS-external-LSA, ospf_flood.c:ospf_process_self_originated_lsa(...) and ospf_lsa.c:ospf_lsa_refresh_walker(...) check with ospf_flood.c:ospf_external_info_check(...) if the underlying external information still exists (external information is stored separately from the link-state database) and if external information from the issuing source is still to be redistributed. Otherwise, the AS-external-LSA will be flushed from the AS instead of being refreshed.

The refresh mechanism for summary-LSAs is also triggered by ospf_abr.c:ospf_abr_announce_network_to_area(...), those for ASBR-summary-LSAs by ospf_abr.c:ospf_abr_announce_rtr_to_area(...).

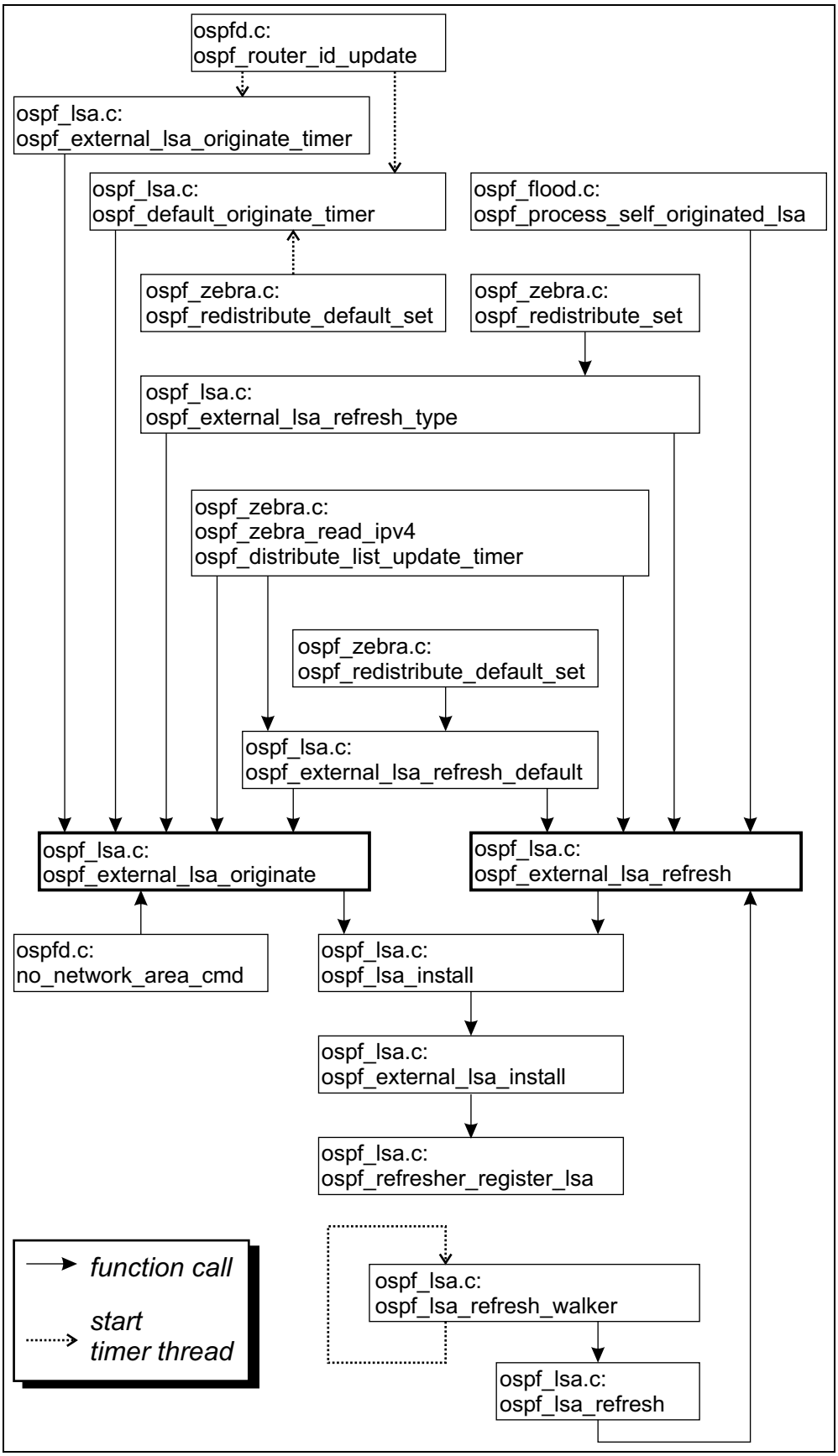


**Figure 3:** Origination and Refreshing of AS-external-LSAs

# 4 Faking AS-external-LSAs

Section 3.3 has shown that manipulating AS-external entries in the link-state database is feasible without making substantial changes in the original source code of OSPFD. That is due to the fact that AS-external-LSAs do not depend on areas, interface states, neighbor adjacencies etc. In the following, a patch for OSPFD is presented that allows inserting arbitrary external information in order to generate fake AS-external-LSAs.

## 4.1 Installation of the Patch

The patch has been conceived for GNU Zebra 0.92a. However, it should work with new versions as well, since only small changes are supposed to be made in the existing code. The patch consists of the following files:

**ospf_lsa_patch.c:** the source file of the patch.

**ospf_lsa_patch.h:** the corresponding header file.

**ospf_main.c:** the modified main source file of OSPFD.

**ospf_zebra.h:** a modified header file of OSPFD.

**Makefile.am:** the modified Makefile of OSPFD.

For use with GNU Zebra 0.92a, just copy all these files to the OSPFD directory ospfd/ by overwriting the existing ones. Then follow the regular Zebra install instructions (`configure`, `make`, `make install`).

If the patch is to be used with another version of Zebra, try out the modifications described in the README file of the patch.

## 4.2 Command Description

The patch adds two vty commands to the "router ospf" configuration mode. Starting from enable mode, you access this mode with:

```
# configure terminal
# router ospf
```

The first patch command is:

```
lsa-patch external-info add (kernel|connected|static|rip|bgp) \
          A.B.C.D/M [forward A.B.C.D]
```

It adds an external information to the external information database. The information is defined by the type of the issuing source (kernel, connected, static, RIP or BGP) and the IP destination prefix. In addition, an optional forward address can be given to indicate another non-OSPF router as gateway (see RFC 2328).

On the basis of the external information, an AS-external-LSA is originated, even if redistribution is currently not enabled for the given source type. However, when reaching MaxAge for the first time, the AS-external-LSA will not be refreshed but flushed instead, unless redistribution is enabled.

The redistribute settings can be changed with:

```
[no] redistribute (kernel|connected|static|rip|bgp)
```

The second patch command

```
lsa-patch external-info delete (kernel|connected|static|rip|bgp) \
          A.B.C.D/M [forward A.B.C.D]
```

deletes the given external information from the database and flushes the corresponding AS-external-LSA immediately.

Remember that only destinations outside the AS are considered during the shortest path calculation.

## 4.3 Implementation

The patch does quite the same as the API function ospf_zebra.c:ospf_zebra_read_ipv4(...). The principal difference is that the external information is not read from a stream, but entered through the new vty commands. The source code of ospf_lsa_patch.c can be found in appendix A.

# 5 Conclusion

With the presented Zebra patch, arbitrary AS-external-LSAs can be originated. These LSAs are flooded throughout the whole AS (except for stub areas) and stored in the link-state databases of all routers. However, only packet routing to destinations outside the AS is affected. Provided that the announced route is convenient, packets sent within the AS to the external destinations will be forwarded to the router that has originated the fake AS-external-LSA, or to the given next hop address respectively. Since the external route does not exist, the packets will not reach their destination.

Routing inside the AS is mostly based on router-LSAs, since they inform about IP subnets connected to the routers, the principal information needed for routing decisions. Thus, subnet changes could be simulated by originating fake router-LSAs. In contrast, manipulating network-LSAs is less interesting, because they only contain information about OSPF routers connected to a common broadcast network.

A desirable experiment might be the false advertisement of a subnet connected to an existing or non-existing router interface by originating a fake router-LSA. However, as shown in section 3.2, the necessary changes in the OSPFD code are substantial and tricky, because the router-LSA information is gathered directly from the current interface configuration.

Beyond originating LSAs, a LSA generator should also control LSA flooding, i.e. when and where LSAs are sent out. Usually, OSPF routers flood LSAs that have changed or that have to be refreshed because of their age. Flooding is throttled by various timers in order to avoid high routing traffic, e.g. two instances of the same LSA may not be originated within the time period MinLSInterval (see RFC 2428). These protection mechanisms have to be bypassed if high LSA traffic is needed.

# A Source Code of the Patch

ospf_lsa_patch.c:

```
#include <zebra.h>

#include "command.h"
#include "vty.h"
#include "prefix.h"
#include "log.h"

#include "ospfd/ospfd.h"
#include "ospfd/ospf_asbr.h"
#include "ospfd/ospf_lsa.h"
#include "ospfd/ospf_zebra.h"

#define LSA_PATCH_STR "Patch for LSA manipulation\n"


/* Add AS-external-LSA to LSDB */
int
ospf_lsa_patch_external_add (u_char type, struct prefix_ipv4 p,
    unsigned int ifindex, struct in_addr nexthop)
{
  struct external_info *ei;

  /* Create external information and
     store it in external info list ospf_top->external_info[type] */
  ei = ospf_external_info_add (type, p, ifindex, nexthop);

  if (ospf_top->router_id.s_addr == 0)
    /* no router ID assigned yet,
       set flags to generate AS-external-LSA originate event
       for each redistributed protocols later. */
    ospf_top->external_origin |= (1 << type);
  else
  {
    if (ei)
    /* external information is valid */
    {
      if (is_prefix_default (&p))
      {
        /* generate AS-external-LSA for default route */
        zlog_info ("LSA patch: adding/refreshing external default route");
          ospf_external_lsa_refresh_default ();
      }
      else
      {
        struct ospf_lsa *current;

        /* look for old LSA to same destination */
        current = ospf_external_info_find_lsa (&ei->p);
        if (!current)
```

```
        {
          /* no old LSA found, originate new LSA */
          zlog_info ("LSA patch: adding external route %s/%d",
              inet_ntoa (p.prefix), p.prefixlen);
          ospf_external_lsa_originate (ei);
        }
        else /* if (IS_LSA_MAXAGE (current)) */
        {
          /* old LSA found, refresh it */
          zlog_info ("LSA patch: refreshing external route %s/%d",
              inet_ntoa (p.prefix), p.prefixlen);
          ospf_external_lsa_refresh (current, ei, LSA_REFRESH_FORCE);
        }
      }
    }
  }
  return 0;
}


/* Delete AS-external-LSA from LSDB */
int
ospf_lsa_patch_external_delete (u_char type, struct prefix_ipv4 p,
    unsigned int ifindex, struct in_addr nexthop)
{
  zlog_info ("LSA patch: deleting external route %s/%d",
      inet_ntoa (p.prefix), p.prefixlen);

  /* Delete external info from ospf_top->external_info[type] */
  ospf_external_info_delete (type, p);

  /* Flush LSA */
  if (!is_prefix_default (&p))
    ospf_external_lsa_flush (type, &p, ifindex, nexthop);
  else
    ospf_external_lsa_refresh_default ();
  return 0;
}


/* Manipulate external info */
int
ospf_lsa_patch_external_route (struct vty *vty, int add_cmd,
    char *dest_str, char *mask_str, char *nexthop_str, char *type_str)
{
  int ret;
  struct prefix_ipv4 p;
  struct in_addr mask;
  struct in_addr nexthop;
  unsigned long ifindex;
  int type;

  char buf[BUFSIZ];
```

```
  /* Get destination address/mask */
  ret = str2prefix_ipv4 ( dest_str , &p);
  if (ret <= 0)
  {
    vty_out (vty, "%% Malformed address%s", VTY_NEWLINE);
    return CMD_WARNING;
  }

  /* Cisco like mask notation. */
  if (mask_str)
  {
    ret = inet_aton (mask_str, &mask);
    if (ret == 0)
    {
      vty_out (vty, "%% Malformed address%s", VTY_NEWLINE);
      return CMD_WARNING;
    }
    p.prefixlen = ip_masklen (mask);
  }

  /* Apply mask for given prefix . */
  apply_mask_ipv4 (&p);

  /* If nexthop is not specified , nexthop is 0.0.0.0 */
  if (nexthop_str)
  {
    if (! inet_aton (nexthop_str, &nexthop)) return CMD_WARNING;
  }
  else inet_aton ("0.0.0.0", &nexthop);

  /* Take first interface as default */
  ifindex = 0;

  /* Get route type (=source of external info) */
  if (! str2distribute_source ( type_str , &type))
    return CMD_WARNING;

  /* Trigger action : add or delete external info */
  if (add_cmd)
     ospf_lsa_patch_external_add (type, p, ifindex , nexthop);
  else
     ospf_lsa_patch_external_delete (type, p, ifindex , nexthop);

  vty_out (vty, "%s %s/%d", LOOKUP (ospf_redistributed_proto, type),
                                        inet_ntoa (p.prefix ), p.prefixlen );
  vty_out (vty, "  %s %s", inet_ntoa (nexthop), VTY_NEWLINE);

  return CMD_SUCCESS;
}


/* VTY Commands */

/* lsa-patch command without next hop */
```

```
DEFUN (ospf_lsa_patch_add_external_route,
   ospf_lsa_patch_add_external_route_cmd ,
  "lsa-patch external-info add (kernel|connected|static|rip|bgp) A.B.C.D/M",
  LSA_PATCH_STR
  "Manipulate external info\n"
  "Add external info, originate AS-external-LSA\n"
  "Kernel routes\n"
  "Connected\n"
  "Static routes\n"
  "Routing Information Protocol (RIP)\n"
  "Border Gateway Protocol (BGP)\n"
  "IP destination prefix (e.g. 10.0.0.0/8)\n")
{
  char *nexthop_str = NULL;

  /* Get next hop string if available */
  if (argc >= 3) nexthop_str = argv[2];

  return ospf_lsa_patch_external_route (vty, 1, argv[1], NULL, nexthop_str, argv[0]);
}

/* lsa-patch command with next hop */
ALIAS ( ospf_lsa_patch_add_external_route ,
   ospf_lsa_patch_add_external_route_forward_cmd ,
  "lsa-patch external-info add (kernel|connected|static|rip|bgp) A.B.C.D/M forward A.B.C.D",
  LSA_PATCH_STR
  "Manipulate external info\n"
  "Add external info, originate AS-external-LSA\n"
  "Kernel routes\n"
  "Connected\n"
  "Static routes\n"
  "Routing Information Protocol (RIP)\n"
  "Border Gateway Protocol (BGP)\n"
  "IP destination prefix (e.g. 10.0.0.0/8)\n"
  "Forward to another OSPF router\n"
  "IP forward address\n")


/* lsa-patch command without next hop */
DEFUN ( ospf_lsa_patch_delete_external_route ,
   ospf_lsa_patch_delete_external_route_cmd ,
  "lsa-patch external-info delete (kernel|connected|static|rip|bgp) A.B.C.D/M",
  LSA_PATCH_STR
  "Manipulate external info\n"
  "Delete external info, flush AS-external-LSA\n"
  "Kernel routes\n"
  "Connected\n"
  "Static routes\n"
  "Routing Information Protocol (RIP)\n"
  "Border Gateway Protocol (BGP)\n"
  "IP destination prefix (e.g. 10.0.0.0/8)\n")
{
  char *nexthop_str = NULL;
```

```
  /* Get next hop string if available */
  if (argc >= 3) nexthop_str = argv[2];

  return ospf_lsa_patch_external_route (vty, 0, argv[1], NULL, nexthop_str, argv[0]);
}


/* lsa-patch command with next hop */
ALIAS ( ospf_lsa_patch_delete_external_route ,
    ospf_lsa_patch_delete_external_route_forward_cmd ,
  "lsa-patch external-info delete (kernel|connected|static|rip|bgp) A.B.C.D/M forward A.B.C.D",
  LSA_PATCH_STR
  "Manipulate external info\n"
  "Delete external info, flush AS-external-LSA\n"
  "Kernel routes\n"
  "Connected\n"
  "Static routes\n"
  "Routing Information Protocol (RIP)\n"
  "Border Gateway Protocol (BGP)\n"
  "IP destination prefix (e.g. 10.0.0.0/8)\n"
  "Forward to another OSPF router\n"
  "IP forward address\n")


/* Initialization */
void
 ospf_lsa_patch_init ()
{
   install_element (OSPF_NODE, &ospf_lsa_patch_add_external_route_cmd);
   install_element (OSPF_NODE, &ospf_lsa_patch_add_external_route_forward_cmd);
   install_element (OSPF_NODE, &ospf_lsa_patch_delete_external_route_cmd);
   install_element (OSPF_NODE, &ospf_lsa_patch_delete_external_route_forward_cmd);
}
```

# Abbreviations

| | |
|---|---|
| ABR | Area Border Router |
| AS | Autonomous System |
| ASBR | Autonomous System Boundary Router |
| BGP | Border Gateway Protocol |
| DR | Designated Router |
| IETF | Internet Engineering Task Force |
| IGP | Interior Gateway Protocol |
| LSA | Link State Advertisement |
| LSDB | Link-State Database |
| OSPF | Open Shortest Path First |
| RFC | Request For Comment |
| RIP | Routing Information Protocol |
| SNMP | Simple Network Management Protocol |

# References

[1] Gagan L. Choudhury, Anurag S. Maunder, Vera D. Sapozhnikova, "Faster Link-State IGP Convergence and Improved Network Scalability and Stability", *IEEE Conference on Local Computer Networks (LCN) 2001*

[2] Gagan L. Choudhury, Anurag S. Maunder, Vera D. Sapozhnikova, IETF `draft-ietf-ospf-scalability-01.txt`

[3] Dave Katz, Derek Yeung, Kireeti Kompella, IETF `draft-katz-yeung-ospf-traffic-06.txt`

[4] Anindya Basu, Jon G. Riecke, "Stability Issues in OSPF Routing", *ACM SIGCOMM 2001*

[5] GNU Zebra Homepage: *www.zebra.org*
Zebra 0.91a Manual: *dc.qut.edu.au/usr/share/doc/zebra-0.91a/zebra.html*
Zebra for Dummies: *wwwstud.informatik.uni-karlsruhe.de/∼s_uriart/zhh.txt*
API for opaque LSA origination: *www.tik.ee.ethz.ch/∼keller/ospfapi/*

[6] IP Infusion Inc., *www.ipinfusion.com*

[7] GCT-AllWell, *www.gctglobal.com*

[8] Tisya Microsystems, *www.tisya.co.in*