Interconnecting OLSRv2 and OSPF

Dang Quan Nguyen, Louise Lamont
Communications Research Centre Canada

Communications Research Centre Canada
Technical Note
CRC-TN-2009-01
June 2009

This Technical Note is an informal publication of
Communications Research Centre, Canada for internal use only.

Abstract

We design in this paper the mechanisms for interconnecting nodes running OLSRv2 and OSPF protocols, both are link state routing protocols. Nodes, however, are unable to route packets between OLSRv2 and OSPF domains without an OLSRv2-OSPF gateway. This paper makes use of internal mechanisms, well-defined in OLSRv2 and OSPF to design such a gateway.

Résumé

Ce rapport présente des mécanismes pour connecter des noeuds fonctionnant avec le protocole OLSRv2 et ceux avec OSPF. Tous deux sont des protocoles de routage basés sur l'état des liens. Cependant, les noeuds ne sont pas capable de router des données entre les domaines OLSRv2 et OSPF sans avoir une passerelle OLSRv2-OSPF. Ce rapport préconise l'utilisation des mécanismes internes, bien définis dans OLSRv2 et OSPF pour concevoir une telle passerelle.

This page intentionally left blank.

Table of contents

List of figures

1 Introduction

Open Shortest Path First (OSPF [3]) is a protocol used for routing in the Internet. OSPF is a *link-state* routing protocol, meaning that it keeps the routers in the Internet up-to-date with the states of the links between other routers. The availability of these links create a topology of the network. Thus, each router is able to compute the shortest route to any other router, using Dijkstra's algorithm [9].

Optimized Link State Routing (OLSR [1]) is also a link-state routing protocol. It is optimized for routing in wireless mobile ad hoc networks by greatly reducing the redundancies of network-wide information broadcasts. This is achieved by using a *Multipoint-Relay* set (MPR) which is the core element in the MPR-flooding mechanism. A secondary optimization is that OLSR employs partial link state information: each node maintains a subset of links to reach all destinations. Thus, it reduces the size of network-wide link state broadcasts. Like OSPF, OLSR also uses Dijkstra's algorithm to compute the shortest route between any two nodes.

OLSR version 2 (OLSRv2 [2]), while retaining the same basic mechanisms and algorithms of OLSR, provides a more flexible signaling framework and some simplification of the messages exchanged. Also, OLSRv2 accomodates either IPv4 and IPv6 in a compact manner.

OSPF, OLSR and OLSRv2 are proactive routing protocols which means that routing information is periodically exchanged and route availability is continuously updated.

In case a node has multiple interfaces, each of them can run a different routing protocol. Therefore, this node acts as a gateway and allows other nodes in the OSPF or OLSR network to connect to the foreign networks. In general, the setting of such gateways is done statically in the scenario's pre-configuration phase.

We investigate in this paper the necessary mechanisms to dynamically interconnect OSPF and OLSRv2 networks. Available routes in the OLSRv2 network are made known to the OSPF nodes and conversely.

Figure 1 illustrates an example of interconnecting two networks: OLSRv2 and OSPF. The interconnection is carried out by a gateway which has two network interfaces, one running OLSRv2 with address prefix 10.0.0.x and the other running OSPF with prefix 192.168.0.x. It is possible to have more than one gateway. The gateways ensure the routability of information from one network to the other. For example: any change of topology in the OLSRv2 network must be reported to the OSPF network and vice versa.

This paper is organized as follows. We briefly report two existing approaches for interconnecting OLSR and OSPF in section 2. The internal mechanisms of OLSRv2 and OSPF

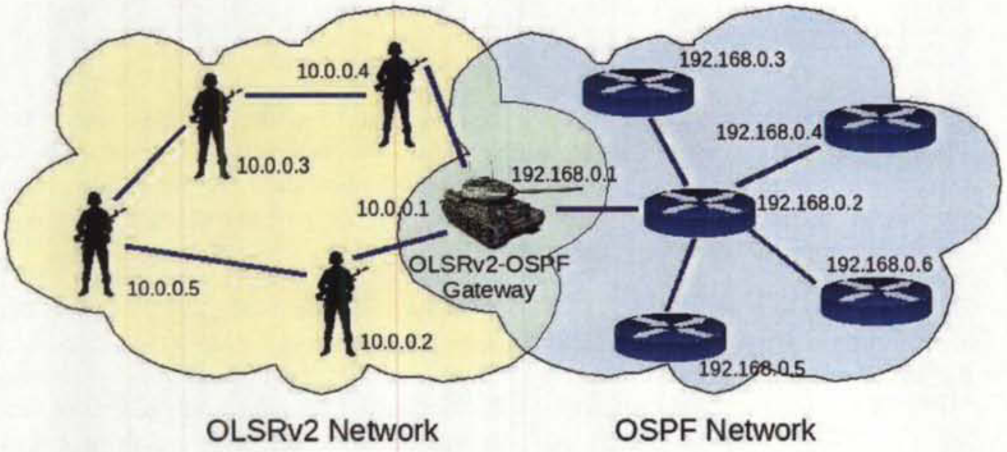

Figure 1: *An example of a network scenario connecting OLSRv2 and OSPF networks.*

related to the interconnection issue are described in section 3. The design of OLSRv2 and OSPF interconnection is given in section 4. Finally, we conclude in section 5.

2 Related work

In [6], the authors discuss two methods of interconnecting nodes running the OLSR protocol in one routing domain, and those running the OSPF protocol in another routing domain. Given the fact that both protocols (OLSR and OSPF) are Interior Gateway Protocols (IGP, specialized in routing inside a single domain), the first method considers the interconnection using a third-party routing protocol, such as the Border Gateway Protocol (BGP [8]), which is an Exterior Gateway Protocol (EGP). It turns out that this approach is inappropriate for the normal use of BGP, since most of its features are not needed in order to achieve the desired interconnection, e.g.: service level agreements, routing policies, etc...

The second method which is more simple and straightforward, makes use of the internal mechanisms, well-defined in both OLSR and OSPF, which are designed for the purpose of importing routes from external routing domains. Thus, using their respective mechanisms, OLSR and OSPF protocols are able to exchange their routes. More specifically, nodes running OLSR use the *Host and Network Association* (HNA) messages to populate routes coming from OSPF domain: by advertising they have reachability to OSPF hosts or networks. Similarily, OSPF nodes can use the *Link-State Advertisement* (LSA) messages to import OLSR routes into their OSPF network.

The specification and details on the implementation of this second method are given in [7].

The authors recommend using an open source implementation of OSPF version 3 [4], called *quagga*, since it is being actively developed. As for OLSR, the INRIA version, called *OOLSR*, is used. An extra program, called *Quagga OOLSR Exchange Daemon* (QOED), is developed and plays the role of the interface between OOLSR and Quagga. QOED is the central part of the interconnection implementation. It exchanges routes between OLSR and OSPF, and it triggers the sending of HNA messages in OLSR. It also triggers the sending of LSA messages in OSPF indirectly via Quagga route setting. We give more details on its main operations in section 4.

3 OLSRv2 and OSPF internal mechanisms

The second method described above appears to be a proper and customizable approach for interconnecting OLSR and OSPF. It will serve as a base for the design of the OLSRv2-OSPF interconnection. A few modifications are required due to the differences between OLSR and OLSRv2. This is due to the fact that in OLSRv2, foreign hosts and networks are advertized in *Topology Control* (TC) messages rather than with HNA messages that are no longer supported.

We only describe in this section the internal mechanisms of OLSRv2 and OSPF related to the interconnection issue. The reader may refer to the respective documents of OLSRv2 [2] and OSPF [3, 4] for other features such as MPR selection (OLSRv2), database synchronization (OSPF), route computation, etc...

3.1 OLSRv2 mechanisms

Each OLSRv2 node records in its *Local Attached Network Set* all local non-OLSRv2 interfaces that can act as gateways to other networks. The tuples in this set have the following format:

```
(AL_net_addr, AL_dist)
```

where:

- `AL_net_addr` is the network address of an attached network which can be reach via this node.
- `AL_dist` is the number of hops to the network with address `AL_net_addr` from this node.

In order to interconnect with OSPF, the gateway node's interconnection mechanims fills the *Local Attached Network Set* with tuples whose AL_net_addr is the address of an OSPF node and AL_dist is the distant in number of hops from the gateway to this OSPF node within the OSPF network.

The OLSRv2 protocol does not modify its *Local Attached Network Set*, but it can respond to changes in the set, by sending the *Topology Control* (TC) messages to advertise these OSPF destinations to all the OLSRv2 nodes. Each OSPF address AL_net_addr declared in the TC message must be associated with a Time-Length-Value message (TLV [5]) with Type = Gateway and Value = AL_dist. The specifications of generating and processing of TC messages can be found in [2].

Let us consider the network scenario presented in figure 1. The following example details the semantics of a TC message, issued by the gateway into the OLSRv2 network to announce the route availability to an OSPF destination (192.168.0.6).

```
Originator address = 10.0.0.1
An address block containing:
  +  192.168.0.1  ;local address attached to OSPF network
  +  LOCAL_IF TLV with value UNSPEC_IF associated to this address
A second address block containing:
  +  192.168.0.6  ;OSPF destination address
  +  GATEWAY TLV with value 2 associated to this address  ;2 hops away
```

3.2 OSPF mechanisms

Routes external to OSPF routing domain are advertized by the gateway node throughout the OSPF network (refered to as *Autonomous System* or AS), by sending the AS-external-LSAs. The Link_State_ID field contains the address of an OLSR destination. The distance in number of hops from this gateway to the destination within the OLSR network is specified in the metric field.

OSPF networks, however, support two types of metric (specified by the bit_E field). Metrics of type 1 mean that the cost of an external route is comparable to that of an OSPF route. In the case of metric being the number of hops, type 1 simply means that distances in OLSR network are processed as if they were distances in OSPF network. On the other hand, the metric of an external route advertized as type 2 is not comparable to OSPF cost, whatever their values are. Type 2 metric is used when two networks use different kinds of cost such as number of hops and delay. More details can be found in [3].

In OSPFv3 for IPv6 [4], the algorithms are preserved from OSPFv2 (for IPv4) regarding importation of external routes. Some changes are, however, necessary to reflect the different address spaces between IPv6 and IPv4. For example, the Link_State_ID field of

AS-external-LSAs loses all of its addressing semantics in OSPF for IPv6. It simply serves to distinguish multiple AS-external-LSAs that are originated by the same gateway. The OLSR destination address is described by the `Address_Prefix` field embedded within the LSA body.

Let us consider the network scenario presented in figure 1. The following example details the semantic of an AS-external-LSA message, issued by the gateway into the OSPF network to announce the route availability to an OLSRv2 destination (10.0.0.5).

```
LS age = 0                         ;newly (re)originated
Options = (E-bit)
LS type = 5                        ;AS-external-LSA
Link State ID = 10.0.0.5
Advertising Router = 192.168.0.1
bit E = 0                          ;type 1 metric
Metric = 2                         ;two hops from the gateway
Forwarding address = 0.0.0.0       ;use 192.168.0.1 as gateway
```

4 Design of OLSRv2-OSPF interconnection

Our design of OLSRv2-OSPF interconnection strictly follows the architecture presented in [7]. Because this architecture preserves the specifications of the existing components such as OLSR and OSPF. Any extra part is developed as an add-on module and not as an extension to the already defined protocols. Moreover, the current implementation of OSPFv3 (Quagga) and that of OLSRv2 (CRC-OLSRv2) follow this line of philosophy by allowing the development of plug-ins rather than the modification of existing procedures and functions.

Figure 2 illustrates the architecture of the OLSRv2-OSPF interconnection modules. This architecture involves three main components: quagga, QCX (for *Quagga-Crc's olsrv2 eXchange deamon*) and OLSRv2. As stated above, **quagga** is an implementation of OSPF (among other protocols such as RIP, BGP, etc.) The architecture of quagga itself is composed of a main module (called **zebra**) and many protocol modules (OSPF, RIP, BGP,...) Only the OSPF protocol module is presented in the figure. The **zebra** module is in charge of the exchange and configuration of the network information at the system-lower level : route setting and reading, interface configuration, etc. The OSPF module which implements the OSPF protocol utilizes the APIs offered by zebra to perform these tasks. Those APIs are part of the **Zserv-Zclient** libraries. One main advantage of this separation in quagga is that it makes the protocol implementations independent of the operating system.

While the OLSRv2 module simply implements the OLSRv2 protocol specifications, an extra module is needed to exchange routing information with OSPF (e.g.: notification of

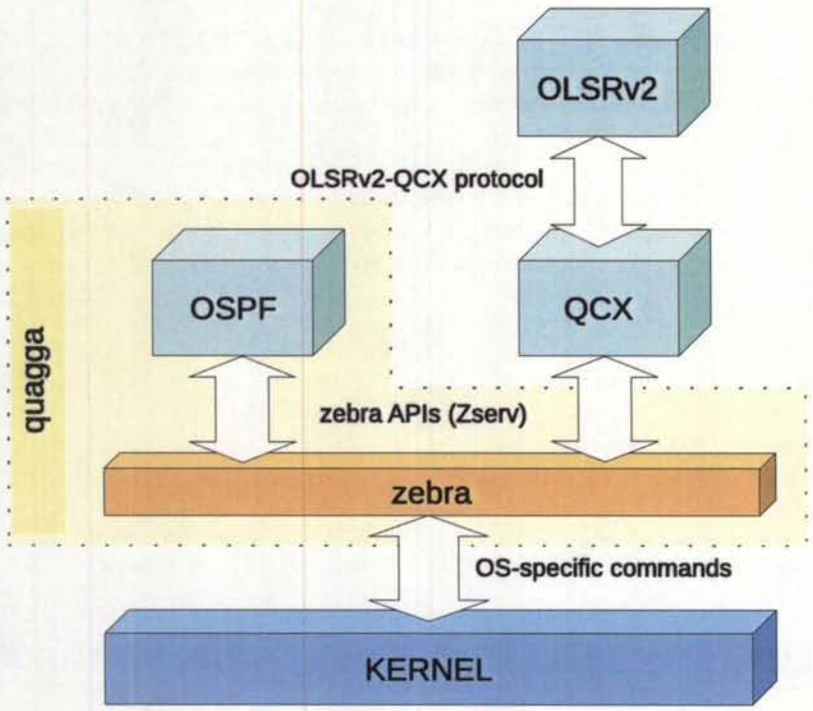

Figure 2: *Architecture of the OLSRv2-OSPF interconnection modules.*

changes in their respective routing tables.) This is the main purpose of the QCX module. On the one hand, QCX communicates with OLSRv2 by the OLSRv2-QCX protocol that will be described shortly. On the other hand, QCX communicates with OSPF by a quagga-standard way, i.e. using zebra.

It is possible to consider removing the QCX module from this scheme and to make OLSRv2 communicate directly with zebra. However, doing so would bring more modifications to the existing CRC-OLSRv2 and zebra code. There are also risks of altering the OLSRv2 current specifications since some procedures of QCX (such as checking for OSPF-imported routing table's version) are not specified in the draft.

We now describe two main sequences of actions in this architecture. They consist of importing routes from OLSRv2 to the OSPF network and vice versa. These actions are defined in [7] for OLSR and OSPF. We slightly adapt them to the OLSRv2 protocol.

4.1 Importing routes from OLSRv2 to OSPF

The following actions are taken at the gateway node when importing a new route from OLSRv2 to the OSPF network:

1. OLSRv2 detects a route change in the OLSRv2 network.
2. OLSRv2 transmits the routes to QCX.
3. QCX verifies that all parts of the routing table are received (in case of large routing tables and the messages being fragmented).
4. QCX transmits the routes to zebra.
5. zebra updates the kernel routing table.
6. zebra sends the route update to OSPF.
7. OSPF sends an AS-external-LSA message into the OSPF network to declare these new destinations.

The communications between these modules are done via locally opened sockets.

4.2 Importing routes from OSPF to OLSRv2

This can be done at the gateway node by performing the following sequence of actions:

1. OSPF detects a route change in the OSPF network.
2. OSPF transmits the route update to zebra.
3. zebra updates the kernel routing table.
4. zebra sends the route update to QCX.
5. If needed, QCX splits the OSPF routing table into several messages.
6. QCX sends the whole OSPF routing table to OLSRv2.
7. OLSRv2 sends a TC message with one or many associated gateway TLVs into the OLSRv2 network to declare these destinations.

4.3 Information validity timer

OLSRv2 and OSPF recommend different timer values to maintain updated information in their local databases. Since OSPF networks are more static than OLSRv2 networks and timers in OLSRv2 are usually shorter than in OSPF to maintain the validity of updated information. Thus, OLSRv2 nodes may remove OSPF routes (imported to their routing tables with TC messages) before receiving an update from the OSPF network.

We suggest two solutions to handle this issue. The first solution consists of the gateway node periodically checking the validity of OSPF routes and sending a TC message with associated gateway TLVs into the OLSRv2 network to update the OSPF destinations. This period must be shorter than the validity timer of this information being held at the other OLSRv2 nodes. In practice, it consists of having the QCX module at the gateway node periodically send the OSPF routing table to OLSRv2. Once the *Local Attached Network Set* in OLSRv2 is alimented, this node automatically generates the TC messages at an appropriate period according to the specifications of OLSRv2.

The second solution consists in extending the validity time in OLSRv2 for external routes imported from OSPF. In practice, when an OLSRv2 node receives a TC message with associated gateway TLV, it stores this information in its *Attached Network Set*. Each tuple of this set has a field `AN_time` indicating the time at which the tuple must be removed. Therefore, this value must be extended to be equal to the route holding time in the OSPF network.

Each of these solutions has its own tradeoffs. For example, the first solution preserves the reactivity of the OLSRv2 nodes in the case of frequent changes in OSPF routes, while the second solution generates less overhead. However, we recommend the first solution because it does not require any modification in OLSRv2. Another argument advocating for the first solution is when there are several OSPF routing domains, each has a different holding time value, interconnecting with an OLSRv2 network. The second solution becomes impracticable in this case.

5 Conclusion

We have presented in this paper the architecture for interconnecting OLSRv2 and OSPF networks. We use the internal mechanisms of OLSRv2 and OSPF to exchange routes. The design of this interconnection is modular to allow a flexible implementation and to preserve the specification of the existing protocols. We also describe the main actions performed by each module to achieve the interconnection and discuss the harmonisation of the validity hold times of information recorded in OSPF and OLSRv2.

6 Acknowledgement

The authors thank Dr. M. Salmanian and Mr. R. Paxton for their valuable reviews and comments.

References

[1] C. Adjih, T. Clausen, P. Jacquet, A. Laouiti, P. Minet, P. Muhlethaler, A. Qayyum, L. Viennot: *Optimized Link State Routing Protocol*, IETF RFC 3626, http://www.ietf.org/rfc/rfc3626.txt, October 2003.

[2] T. Clausen, C. Dearlove, P. Jacquet: *The Optimized Link State Routing Protocol version 2*, IETF Internet Draft, work in progress, http://www.ietf.org/internet-drafts/draft-ietf-manet-olsrv2-06.txt, 2008.

[3] J. Moy: *OSPF version 2*, IETF RFC 2328, http://www.ietf.org/rfc/rfc2328.txt, 1998.

[4] R. Coltun, D. Ferguson, J. Moy: *OSPF for IPv6*, IETF RFC 2740, http://www.ietf.org/rfc/rfc2740.txt, 1999.

[5] T. Clausen, J. Dean, C. Dearlove, C. Adjih: *Generalized MANET Packet/Message Format*, IETF Internet Draft, work in progress, http://www.ietf.org/internet-drafts/draft-ietf-manet-packetbb-12.txt, 2008.

[6] INRIA: *Interconnecting OLSR and OSPF*, CELAR Report, September 2005.

[7] INRIA: *Interconnecting OLSR and OSPF: Specification and Implementation of QOED (Quagga OOLSR Exchange Daemon)*, CELAR Report, June 2006.

[8] Y. Rekhter, T. Li (Ed.): *A Border Gateway Protocol 4 (BGP-4)*, IETF RFC 1771, http://www.ietf.org/rfc/rfc1771.txt, 1995.

[9] E. W. Dijkstra: *A note on two problems in connexion with graphs*, Numerische Mathematik 1, pp. 269-271, 1959.
