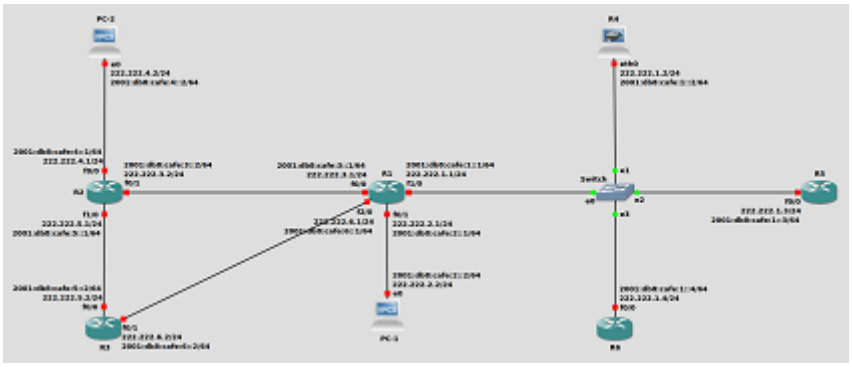
TÉCNICO LISBOA



OSPF extensions for the support of multi-area networks with arbitrary topologies

Miguel de Oliveira Dias Gonçalves

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Rui Jorge Morais Tomaz Valadas

Examination Committee

Chairperson: Prof. Maria Luísa Torres Ribeiro Marques da Silva Coheur
Supervisor: Prof. Rui Jorge Morais Tomaz Valadas
Member of the Committee: Prof. João Miguel Duarte Ascenso

January 2021

Acknowledgments

I would like to thank my dissertation supervisor, Prof. Rui Valadas, for his valuable guidance and support throughout the development of our implementation and the writing of the current dissertation.

I would also like to thank my family for all of their support, encouragement and guidance during this months and years.

Last but not the least, I would like to thank my friends for their friendship and support during this years.

Thank you very much.

Abstract

Routing protocols are a major part of the Internet, providing information about the network to IP routers in order to allow them to forward received packets. Link state routing protocols, such as OSPF and IS-IS, are the most used nowadays. They solve the count-to-infinity problem found in distance vector routing protocols by providing the complete network view to every router. Link state routing networks can be divided in areas to facilitate the scaling of the network. However, in current OSPF versions, inter-area routing uses a distance vector approach, bringing limitations to multi-area networks. In particular, they must have a two-level hierarchical structure, and optimal routing is not guaranteed for inter-area routing. This MSc Dissertation describes the development of two extensions of the OSPF protocol that overcome the mentioned limitations, one for OSPFv2 (IPv4) and the other for OSPFv3 (IPv6). They provide a link state approach to inter-area routing, allowing arbitrary multi-area topologies and optimal inter-area routing. An implementation in Python of the base OSPF protocol was also developed to support the development of the extensions, which were created on top of the base implementation. GNS3 networks were created and used to test the implementation, showing that both versions of the base implementation and the OSPFv2 extension are operational. This dissertation gives some background on OSPF and its multi-area extensions, describes the software architecture and the developed implementation, and presents the experiments performed and the respective results.
This MSc Dissertation was supported by the Instituto de Telecomunicações.

Keywords

OSPF; OSPFv2; OSPFv3; Routing; Link state routing; Routing protocols;

Resumo

Os protocolos de encaminhamento são uma parte importante da Internet, providenciando informação sobre a rede a routers IP de modo a permitir que encaminhem pacotes recebidos. Os protocolos do tipo link state, como o OSPF e o IS-IS, são os mais usados actualmente. Resolvem o problema da contagem-para-o-infinito encontrado em protocolos do tipo distance vector providenciando a visão completa da rede a cada router. Redes do tipo link state podem ser divididas em áreas para facilitar o crescimento da rede. Porém, nas versões actuais do OSPF, o encaminhamento inter-área usa uma abordagem distance vector, trazendo limitações às redes multi-área. Nomeadamente, devem ter uma estrutura hierárquica de dois níveis, e o encaminhamento óptimo não é garantido no encaminhamento inter-área. Esta dissertação descreve o desenvolvimento de duas extensões do protocolo OSPF que superam as referidas limitações, uma para OSPFv2 (IPv4) e a outra para OSPFv3 (IPv6). Providenciam uma abordagem link state ao encaminhamento inter-área, permitindo topologias multi-área arbitrárias e encaminhamento inter-área óptimo. Uma implementação em Python do protocolo OSPF base foi também desenvolvida para apoiar o desenvolvimento das extensões, que foram criadas por cima da implementação base. Redes GNS3 foram criadas e usadas para testar a implementação, mostrando que ambas as versões da implementação base e a extensão para OSPFv2 estão operacionais. Esta dissertação providencia algum contexto sobre o OSPF e as suas extensões multi-área, descreve a arquitectura do software e a implementação desenvolvida, e apresenta as experiências realizadas e os respectivos resultados.
Esta dissertação foi apoiada pelo Instituto de Telecomunicações.

Palavras Chave

OSPF; OSPFv2; OSPFv3; Encaminhamento; Protocolos do tipo link state; Protocolos de encaminhamento;

Contents

List of Figures

List of Tables

Acronyms

ABR	Area Border Router
AS	Autonomous System
ARPANET	Advanced Research Projects Agency Network
ASBR	Autonomous System Boundary Router
BDR	Backup Designated Router
BGP	Border Gateway Protocol
DB	Database
DD	Database Description
DR	Designated Router
DVR	Distance Vector Routing
IANA	Internet Assigned Numbers Authority
ID	Identifier
IETF	Internet Engineering Task Force
IP	Internet Protocol
IS-IS	Intermediate System to Intermediate System
LS	Link State
LSA	Link State Advertisement
LSDB	Link State Database
LSR	Link State Routing
MAC	Media Access Control
OSPF	Open Shortest Path First
RFC	Request For Comments

RIP Routing Information Protocol

UML Unified Modeling Language

VPCS Virtual PC Simulator

1

Introduction

Contents

1.1 Motivation

Information travels through the Internet contained in IP packets, having a source device and destined to another device. Routers are the pieces of equipment that allow IP packets to reach destinations not directly connected to their sources. Routers allow it by performing packet forwarding, i.e. selecting the interface through which a received packet will be sent in order to reach its destination. In order to forward packets, routers must possess information about the network, which is learnt and disseminated in the network using routing protocols. Routing protocols are therefore a vital mechanism of the Internet. They can be either inter-domain, when routing is performed across multiple routing domains, or intra-domain when routing occurs inside a single routing domain. There are two main types of intra-domain routing protocols: Distance Vector Routing (DVR) protocols, and Link State Routing (LSR) protocols.

DVR protocols, such as the Routing Information Protocol (RIP), operate by exchanging messages called distance vectors between directly connected routers in a network. Distance vectors bind a network destination to a path cost. In RIP, usually a hop receives a cost of 1. By receiving the distance vectors of all of the neighbor routers, a router is able to calculate the shortest path to reach any destination in the network, and the respective cost.

However, the approach brings problems, in particular a situation known as count-to-infinity. It occurs when a router becomes unreachable, for example by crashing. Its neighbor routers detect that they cannot reach it, however they keep receiving distance vectors from other neighbor routers in the network containing paths to the now unreachable router. Since routers running DVR protocols do not have a complete network view, the new paths are accepted and then disseminated with increased costs, leading to an infinite cycle where the failure of the router is never discovered and instead paths to it are successively disseminated with higher path costs. RIP solves the situation by declaring a path cost of 16 as infinity, which brings limitations to the size of RIP networks.

LSR protocols provide the complete network view to the routers, preventing the count-to-infinity problem. Open Shortest Path First (OSPF) and Intermediate System to Intermediate System (IS-IS) are two of such protocols. OSPF currently has 2 different versions, OSPFv2 and OSPFv3, respectively for IPv4 and IPv6. The main difference between the versions is the separation of topological and addressing information in OSPFv3 when disseminating routing information in the network.

OSPF networks (i.e. networks with routers running OSPF) can be divided in areas to facilitate scaling. Current OSPF versions use a distance vector approach for inter-area routing. Such approach brings two limitations for multi-area networks in OSPF: the networks are limited to a two-level hierarchical structure with a single area in the top level, and there are cases where it is not guaranteed that the selected path to a destination is the shortest (i.e. optimal routing is not guaranteed).

1.2 Objectives

The general objective of the current MSc Dissertation is to develop two OSPF extensions, one for OSPFv2 and another for OSPFv3, which can overcome the previously mentioned restrictions by applying a LSR approach to inter-area routing in OSPF, allowing arbitrary multi-area topologies and optimal inter-area routing. The developed extensions are described in [1].

The extensions have been designed to take advantage of the OSPF base mechanisms, in order to only require Area Border Routers (ABRs) to be updated so as to accommodate the extensions. The extensions are therefore transparent to area internal routers.

1.3 Achievements

Both OSPF versions have been implemented according to the current specifications, with the OSPF extensions being created on top of the base OSPF implementation. It is an objective of the current dissertation to create a proof of concept of the extensions, where readability is preferred over execution speed. For that reason, the base OSPF implementation and the extensions have been written in Python, and not in faster but less readable languages such as C++. Functional tests were performed on the base OSPF implementation and on the respective extensions using GNS3 and Wireshark. Another objective of the current dissertation is to create an implementation of the base OSPF and of the extensions which is compatible with Cisco routers, which therefore were used in functional and interoperability tests.

1.4 Structure of the report

Chapter 2 will provide information about routing protocols. In particular, Section 2.1 will provide an introduction to routing protocols, Section 2.2 will provide an overview of RIP, and Section 2.3 will provide background on OSPF.

Chapter 3 will describe the developed OSPF extensions. In particular, Section 3.1 will provide a general description of the extensions, Section 3.2 will describe the new Link State Advertisements (LSAs) created for the extensions, Section 3.3 will describe the differences between the base article and the implementation of the extensions, and Section 3.4 will provide an example of the functioning of the extensions.

Chapter 4 will describe the software architecture of the implementation of the base OSPF protocol and of the extensions. In particular, Section 4.1 will provide a general description of the software architecture, which will be detailed in the following sections. Section 4.2 will describe the communication between the different software layers, Section 4.3 will describe the support classes, Sections 4.4 to 4.7

will describe the different software layers of the implementation, and Section 4.8 will describe the architectural changes required in the base OSPF implementation in order to accommodate the extensions.

Chapter 5 will describe the implementation. In particular, Section 5.1 will provide an overview, Section 5.2 will provide detail about the language and the external libraries used, Section 5.3 will describe the source code, and Section 5.4 will describe the automated testing of our implementation.

Chapter 6 will describe the evaluation of the implementation. In particular, Section 6.1 will describe the different GNS3 networks used and how to execute the implementation, and Section 6.2 will describe the manual tests performed on the implementation.

Finally, Chapter 7 will describe the conclusions and the future work of the current dissertation.

2

Routing Protocols

Contents

2.1 Introduction to routing protocols

There are several types of physical devices connected to the Internet. From the point of view of computer networking, they can be classified in three categories: hosts, network devices, and communication media. Hosts, such as servers and mobile phones, are sources and destinations of the information that travels through the Internet. Network devices perform routing and forwarding of information, allowing it to reach destinations not directly connected to the information sources. Hosts and network devices are physically connected by communication media such as copper wires and Ethernet cables, forming networks that can be of several types.

In the network layer, information travels through the network in packets which are forwarded by IP routers. Forwarding is performed according to forwarding tables, which tell the outgoing interface of a packet according to its destination IP address. Each router maintains its own forwarding table, which can be created either statically by network administrators or maintained dynamically using routing protocols such as RIP or OSPF. Routing protocols have the advantage of being able to automatically update the forwarding tables of the routers in reaction to failures of links and routers in the network, or to the appearance of new ones.

Routing protocols can be inter-domain, when the routing is performed across multiple routing domains, even if inside the same Autonomous System (AS); or intra-domain, when routing occurs inside a single domain. The Border Gateway Protocol (BGP) is the currently used inter-domain routing protocol. Intra-domain routing protocols can be further subdivided; the most important categories are distance vector and link state. RIP is an example of the former, while OSPF and the IS-IS protocol are examples of the latter.

2.2 The Routing Information Protocol

DVR protocols such as RIP implement the Distributed Asynchronous Bellman-Ford algorithm, as described in Section 5.2 of [2]. A similar algorithm was used by ARPANET as early as 1969. RIP for IPv4 is currently defined in RFC 2453 [3], while RIP for IPv6 is currently defined in RFC 2080 [4].

In RIP, routers calculate the shortest path to a network destination based on distance vectors, which bind a destination to a path cost. Usually a cost of 1 is attributed to a hop, but higher costs can be assigned.

Routers disseminate the distance vectors they have to their neighbor routers. The cost of a path through a certain neighbor is calculated by adding the cost of the distance vector received from that neighbor and the cost from the router itself to the neighbor. The shortest path to a destination will be one or more of the available paths. By obtaining the shortest path for every destination, it is possible to insert in the forwarding table the next-hop routers and respective outgoing interfaces for all destinations.

Routers running RIP detect network topology changes (i.e. new routers and links, and existing routers and links becoming inoperative) also with distance vectors. All routers in the network must disseminate their distance vectors to all their neighbors every 30 seconds, or when their path cost to a network destination changes. If a router does not receive distance vectors from a neighbor router during 180 seconds, the neighbor is considered down, or at least unreachable through that link.

RIP is affected by a problem known as count-to-infinity, as illustrated in Figure 2.1. Consider a network of 3 routers in line, A, B, and C, where B is connected to A and C, and where each hop has a cost of 1. If A fails, B will detect the failure, but C will inform B that it can still reach A with a cost of 2, not knowing that A has crashed. Since RIP does not provide routers with the full map of the network, B will not know that the path from C to A actually goes through B, and will accept the new path with a cost of 3 and broadcast distance vectors accordingly. C, knowing that its route to A goes through B, will update its path cost to A to 4 after receiving the new distance vector, in turn broadcasting its updated path cost. The cost to A will therefore continue to increase in an infinite routing cycle.

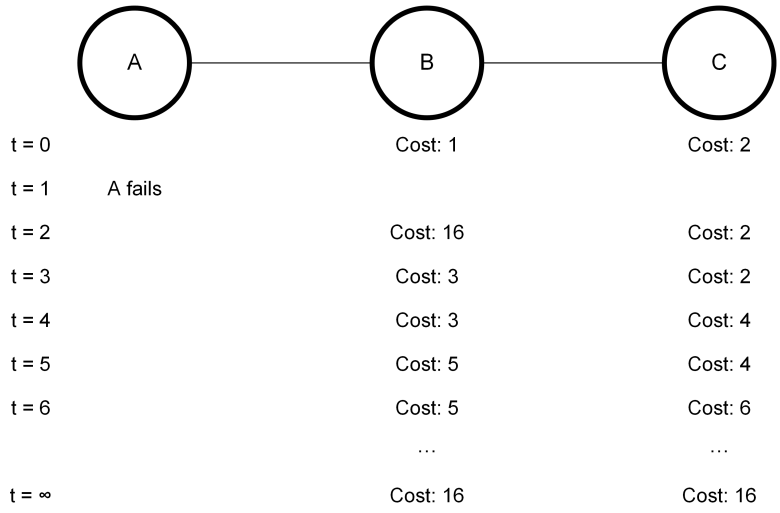


Figure 2.1: Count-to-infinity problem in RIP
Source: Adapted from [5]

RIP solves the count-to-infinity problem by considering infinity to be a finite number, actually 16. If the path cost between two routers of the network is set to 16, it is considered that one router cannot be reached from the other. Therefore, in the routing cycle described above, when the path cost to router A reaches 16, the router will be considered unreachable and its entry in the forwarding tables of routers B and C will be deleted. However, considering infinity as a finite number sets a limit on the network size, as RIP is unable to route traffic over networks with more than 15 hops (assuming each hop has cost 1).

Other techniques are used to mitigate the problem of count-to-infinity, although they do not completely solve it. One technique is called split-horizon, where router C from the previous example would not advertise back to B the route to A it learned from B.

2.3 The Open Shortest Path First protocol

2.3.1 Protocol overview

DVR protocols have been largely supplanted by LSR protocols such as OSPF, as stated in [3]. LSR protocols provide routers with a complete network view, preventing the count-to-infinity problem of DVR protocols and removing the limitation in the number of hops of a network path. OSPF for IPv4, known as OSPFv2, is currently defined in RFC 2328 [6], while OSPFv3, for IPv6, is currently defined in RFC 5340 [7].

Section 2 of [5] describes both types of routing information and the OSPF synchronization mechanisms. From the point of view of routing, 3 types of information must be known:

- Topological information: Provides information about the routers, the costs assigned to their interfaces, and the links between routers;
- Addressing information: Provides information about the routable address prefixes assigned to routers and links;
- Link information: Provides information about the addresses needed to carry packets to the next-hop router.

OSPF stores the network information in a Link State Database (LSDB), created in every router. LSAs contains specific parts of the routing information, and are the building blocks of a LSDB. Except for the link information in OSPFv3 and assuming the network only has one area, every router in the network must have the same LSAs in its LSDB.

The main differences between OSPFv2 and OSPFv3 lie in the way the routing information is stored: while OSPFv2 mixes topological and addressing information in its LSAs (with link information being implicit in their contents), OSPFv3 separates each type of information in different LSAs, creating for the effect new types of LSAs not present in OSPFv2. The main differences in how the routing information is stored will be further addressed in Sections 2.3.4 and 2.3.5.

OSPF uses 3 synchronization mechanisms to keep the network view consistent and updated across all routers. The mechanisms are the following:

- Hello protocol: Allows routers to discover active neighbor routers in a subnet they are attached to;
- Reliable flooding procedure: Allows the dissemination of network information across a network, ending when all routers have received it;
- Initial LSDB synchronization process: Allows synchronization of the network information known by routers that become neighbors, such as when a new router joins a network.

OSPF has 5 types of links between routers: point-to-point links, point-to-multipoint links, Non-Broadcast Multi-Access links, broadcast (shared) links, and virtual links. The most relevant are the point-to-point links and the broadcast links, which will be the ones focused on in this work.

Routers and links have identifiers. Transit shared links, which can connect 2 or more routers at the same time, are the most complex case. To identify them, a router known as the Designated Router (DR) must be elected using the Hello protocol to provide the identifier of the link. A Backup Designated Router (BDR) is also elected to replace the DR in case it fails.

To facilitate scaling, OSPF routing domains can be divided in areas. As described in Section 5.1.3 of [8], some restrictions apply. There must be a special area called backbone area directly connected to all others, and all inter-area traffic must pass through the backbone area even if it not the shortest path to the destination. The structure of an OSPF multi-area network is illustrated in Figure 2.2. A router interface always belongs to only one area; a router therefore always belongs to one or more areas.

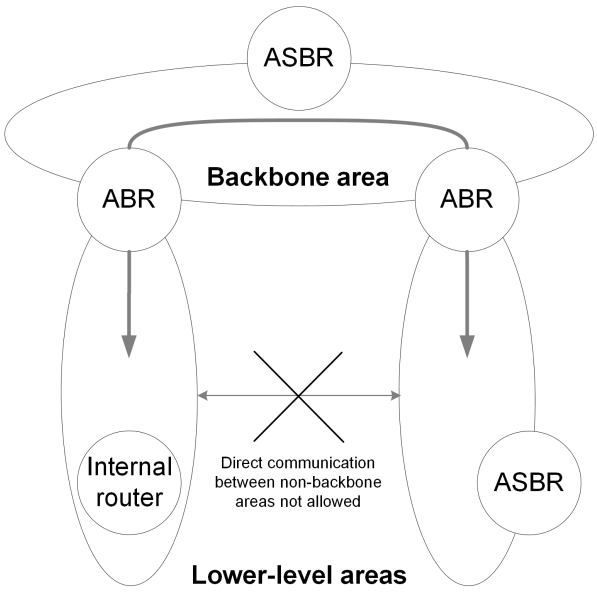


Figure 2.2: Topology of an OSPF multi-area network
Source: Adapted from [5]

Regarding the structure of an OSPF area, there are 3 types of routers. ABRs connect the backbone area with at least one other area: they have at least one interface assigned to the backbone area, and at least one interface assigned to every other area they are connected to. Direct connection between 2 non-backbone areas is not possible.

Autonomous System Boundary Routers (ASBRs) connect the OSPF routing domain with other routing domains or ASes. Routers that do not belong to the previous two types (i.e. routers whose interfaces belong all to the same routing domain and area) are called internal routers.

The next subsections will focus on single-area OSPF, in particular its base mechanisms and data

structures. Section 2.3.12 will describe the additional mechanisms and data structures needed to create multi-area OSPF networks.

2.3.2 Element identifiers

In OSPF, routers and links have identifiers. Routers are identified with a 32-bit number called the Router ID. The number is written as an IPv4 address, and is unique in the routing domain. In OSPFv2 it is not mandatory to manually set the Router ID; in that case the router will select one of the IPv4 addresses assigned to its interfaces to be its Router ID. Implementations of OSPF select the highest IP address.

Unlike router IDs, link identifiers are determined automatically through the Hello protocol. In OSPFv2, links are identified by concatenating a router ID to the IPv4 address of its interface attached to the link; in OSPFv3 the router ID is concatenated to an identifier generated by the router for that interface, called the Interface ID.

Point-to-point links connect 2 and only 2 routers. Each router in the link concatenates the neighbor router ID to the IPv4 address of the neighbor interface attached to the link, in OSPFv2, or to the respective Interface ID, in OSPFv3, in order to obtain the link identifier. The identifier is therefore different for each router in a point-to-point link.

Shared links can connect more than 2 routers. In OSPF, a special router called DR is elected in shared links to determine and disseminate the link identifier. In that case, for every router in the link, the identifier will be the DR ID concatenated with either the IPv4 address of the DR interface connected to the link, in OSPFv2, or the respective Interface ID, in OSPFv3. Also, another router known as the BDR is elected to replace the DR should it fails.

2.3.3 Link State Advertisements

As stated in Section 2.3.1, LSAs are the building blocks of a LSDB. Each LSA has a type, called Link State (LS) Type, determining the type of information it carries. Router-LSAs identify a router of the network and the links it is attached to. Network-LSAs identify transit shared links, i.e. shared links with 2 or more routers attached. AS-External-LSAs identify routing prefixes external to the routing domain. These types of LSAs are common to OSPFv2 and OSPFv3. Other LSA types are exclusive of OSPFv3 or relevant only for multi-area networks, and will be described in the next subsections.

All LSAs have a common 20-byte header, with minor differences between OSPFv2 and OSPFv3. There are two main parameters: the LSA identifier, which uniquely identifies a LSA; and the LSA freshness, which allows the freshness of two instances of the same LSA to be compared. Both are described by 3 fields each.

The LSA identifier includes the above mentioned LS Type, along with the ID of the Advertising Router

(the router responsible for that particular LSA) and the Link State ID (a field that allows identification of different LSAs of the same type coming from the same router). The LSA freshness includes the Sequence Number, the age, and the checksum of the LSA.

2.3.4 Router-LSAs and Network-LSAs

Router-LSAs and Network-LSAs provide the topological information about the network. In the case of OSPFv2, they also provide the addressing information.

A Router-LSA describes a router and its interfaces, including the respective costs and, for OSPFv2, IPv4 addresses. There is one Router-LSA for each router in the network, and each router is the responsible for its own LSA. The body of a Router-LSA contains one or more link descriptors, used to characterize interfaces.

Link descriptors in OSPFv2 contain topological and addressing information: identification of routers and links is mixed with the prefixes assigned to subnets and IPv4 addresses assigned to interfaces and routers (an address assigned to a router can be, for example, a loopback address). In OSPFv3 only topological information is in Router-LSAs: all address prefixes and IP addresses are replaced by Router IDs and Interface IDs. In both OSPF versions, cost information is contained in the link descriptors.

OSPFv2 recognizes 4 types of link descriptors: point-to-point links, link to transit networks, link to stub networks, and virtual links. Transit networks are the above mentioned transit shared links, while stub networks are shared links with a single router attached. Virtual links are not needed to understand OSPF. One interface is described by one link descriptor except if it is an interface to a point-to-point link, which requires a point-to-point link descriptor and a link to stub network descriptor. OSPFv3 removes both stub interfaces and router IP addresses from Routers-LSAs, allowing it to always describe a router interface with a single link descriptor.

A Network-LSA describes a transit shared link and its attached routers. The body of a Network-LSA contains the IDs of all attached routers, as well as the network prefix assigned to the network in the case of OSPFv2. The router responsible for a Network-LSA is the DR of the transit shared link being described. Unlike with Router-LSAs, a router can be responsible for more than one Network-LSA by being the DR of more than one transit shared link. The Link State ID contains the IPv4 address, in OSPFv2, or the identifier, in OSPFv3, of the DR interface attached to the link, in order to distinguish Network-LSAs from the same router.

2.3.5 Intra-Area-Prefix-LSAs and Link-LSAs

OSPFv3 places only topological information on Router-LSAs and Network-LSAs. In order to store addressing information and associate it to the topological information, a new type of LSA is used,

called Intra-Area-Prefix-LSA. Additionally, while link information in OSPFv2 is obtained implicitly from the Router-LSAs and Network-LSAs, in OSPFv3 the information is explicitly stored in another type of LSA called Link-LSA.

Intra-Area-Prefix-LSAs store pairs of network masks and address prefixes. They do not store IPv6 addresses associated with interfaces. Additionally, they are always associated with either a Router-LSA or a Network-LSA, whose LSA identifier fields are stored in the Intra-Area-Prefix-LSA. More than one pair of address prefixes and network masks can be stored in a single LSA, as long as the referenced Network-LSA or Router-LSA is the same. Router and stub interface address prefixes (loopback interfaces are considered stub interfaces) are also stored in Intra-Area-Prefix-LSAs, associated with the respective Router-LSA. Since a router can originate more than one Intra-Area-Prefix-LSA, a unique number is assigned to the Link StateID of each Intra-Area-Prefix-LSA.

Unlike all other types of LSAs, Link-LSAs are only disseminated across a link. Taking advantage of the fact that interfaces running IPv6 must always include a link-local address [9], OSPFv3 uses Link-LSAs to disseminate to all routers in a link the link-local addresses of the interfaces connected to the link, associating the addresses with the respective Interface ID. The address prefixes of the link (OSPFv3 links can be assigned more than one address prefix) are also disseminated, and OSPFv3 ensures that even if an address prefix is configured just at one interface, all others within the link will receive the address prefix as if it was configured in all of them. The Interface ID is used as the Link State ID to distinguish between Link-LSAs created by the same router.

2.3.6 AS-External-LSAs

Routing information from outside the routing domain is inserted at it via routers called ASBRs. Although the name implies they are at the border of Autonomous Systems, it is possible for them to be only at the border of routing domains inside the same Autonomous System.

AS-External-LSAs in both OSPF versions are used for disseminating domain-external address prefixes in the network, carrying both the address prefix and its network mask. Each LSA carries only a single prefix. When the network has a single ASBR, AS-External-LSAs can be used to inject in the network a default route to the ASBR instead of injecting individual address prefixes.

Different routing protocols may use different path metrics, such as the path cost in OSPF and the number of hops in RIP. Additionally, routing between Autonomous Systems takes in consideration other aspects such as financial cost and security, and performance may not be the most important aspect. The difficulties of combining different types of path metrics are discussed in Section 2.2.4 of [5]. In OSPF, external address prefixes are associated with a path cost, just like domain-internal address prefixes, which can be configured by network managers. For OSPF, the path cost represents the cost of the path from the ASBR to the external prefix.

Two types of metric are considered by OSPF when calculating the total path cost from an internal router to an external address prefix, type E1 and type E2. In the first, the cost of a path to the external address is the sum of the mentioned external cost and the domain-internal cost of the path to reach the ASBR from the internal router. In the second, only the external cost is considered. The metric type is carried in the AS-External-LSAs, and is configurable as well by network managers.

2.3.7 Control packets

OSPF has 3 synchronization mechanisms: the Hello protocol; the reliable flooding procedure; and the initial LSDB synchronization process. Their objective is to keep the LSDBs synchronized and updated across all routers in an OSPF routing domain. The mechanisms are the same in OSPFv2 and OSPFv3.

The 3 mechanisms require exchanging control packets between the routers, which are encapsulated in IPv4 (in OSPSv2) or IPv6 (in OSPFv3) packets using the IP protocol number 89. There are 5 types of control packets, based on their contents and functions: the Hello packet; the Database Description (DD) packet; the LS Request packet; the LS Update packet; and the LS Acknowledgement packet. Both OSPFv2 and OSPFv3 use the same control packet types, with slight differences between the two versions. The most significant differences happen in the packet header and in the Hello packet body.

All control packets of the same OSPF version carry the same OSPF header. In both versions, most of the header fields are kept intact. The common fields are the following: the OSPF version; the type; the packet length; the source Router ID; the area ID; and the packet checksum. OSPFv2 packet headers additionally carry authentication-related fields, while in OSPFv3 they are replaced with the Instance ID (OSPFv3 allows more than one independent instance of the protocol to run in a router at the same time).

2.3.8 Hello protocol

The Hello protocol allows routers to detect active neighbor routers on subnets, and the type of relationship that can be established with them. Two OSPF routers can become adjacent (there is bidirectional communication between them) if they have at least one interface connected to the same subnet and belonging to the same area, and also in OSPFv3 if those interfaces are running the same instance of the protocol. It is possible to establish up to as many neighbor relationships as links connecting two routers in the same area.

A router shows to its neighbor routers that it is alive by broadcasting Hello packets every 10 seconds. The IP multicast address AllSPFRouters is used: 224.0.0.5 for IPv4 (OSPFv2) and ff02::5 for IPv6 (OSPFv3). All OSPF routers accept packets with those destination addresses. Hello packets are only broadcast inside a link, i.e. they are not forwarded. Routers show that there is bidirectional communication with another router by including its ID in the Hello packets broadcasted. A router considers that

a neighbor is dead, or at least unreachable through the link, if it does not receive Hello packets from it during 40 seconds.

The Hello protocol exchanges only Hello packets. In both versions of OSPF, they include the Router ID of who the router believes is the DR and the BDR of the link if it is a transit shared link (or 0.0.0.0 if the router does not know the link DR/BDR), and the Router IDs of the neighbor routers that the router know. In OSPFv2 the subnet network mask is sent in the packet, while in OSPFv3 it is replaced with the source Interface ID.

Two types of adjacency between routers are possible: when the respective LSDBs are ready to be synchronized between the routers; and when the LSDBs are already synchronized. Section 6.2.3 of [5] discusses the terminology of both states. In particular, it names the first state as "adjacency" and the second state as "fully adjacency". Two routers become adjacent as soon as bidirectional communication is established between them, i.e. when both find their Router ID in the Hello packets coming from the other. A connection between two routers can become part of the network map when they become adjacent. Full adjacency between two routers can only be reached in point-to-point links or in transit shared links when one router is the DR of the link.

OSPF has a state machine associated with a neighbor relationship, described in Section 10 of [6] and illustrated in Figure 2.3. Every relationship runs its own instance of it. The initial state is Down, and the state machine moves to the Init state when a Hello packet from the neighbor is received. After bidirectional communication is established, one of two states are reached. If the routers can become fully adjacent, the next state is ExStart, otherwise the next state is 2-Way and the two routers are now considered adjacent. If the routers are in 2-Way state, certain events can lead to a new decision between 2-Way and ExStart states. There are further states after ExStart, which will be described in Section 2.3.10.

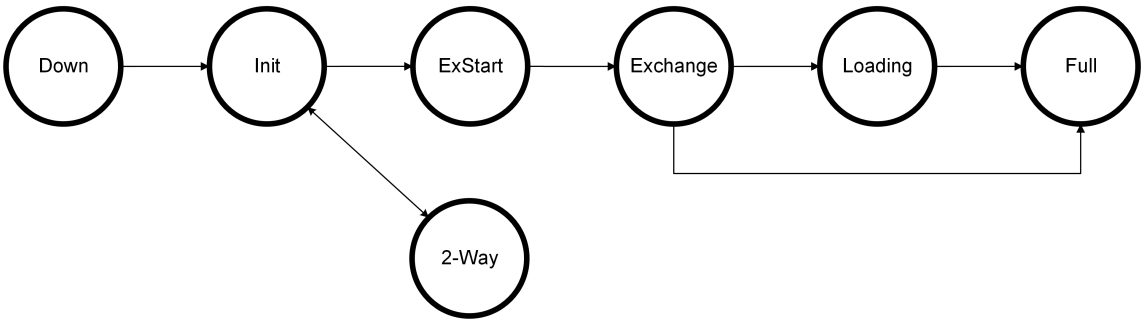


Figure 2.3: OSPF neighbor state machine in point-to-point and broadcast links
Source: Adapted from [5]

As described in Section 6.2.7 of [5], implementations of OSPF send Hello messages directly to a neighbor, instead of to the AllSPFRouters address, if it is detected and bidirectional communication isn't established yet with it, although is not part of the original specification.

2.3.9 Reliable flooding procedure

The reliable flooding procedure is used to disseminate new or updated LSAs across an OSPF area. The procedure is reliable since all LSAs disseminated by a router are acknowledged by the receiving routers, and it is also controlled (i.e. avoids infinite flooding) since received LSAs are only disseminated if they are new to the router or fresher (more recent) than the ones it already has.

The reliable flooding procedure exchanges LS Update and LS Acknowledgment packets. In both versions of OSPF, LS Update packets carry a variable number of LSAs and a field stating the number of LSAs in the packet. LS Acknowledgment packets carry the headers of the LSAs to acknowledge.

When routers create a LSA, they disseminate it to all their interfaces. A disseminated LSA is further flooded by the receiving routers if the LSA is either new to the routers or fresher than the one already in their LSDBs. The LSA is flooded to all their interfaces except the one that received it. The exception is the Link-LSA of OSPFv3, which is only disseminated inside the respective link. Accepted LSAs replace previous instances of the LSA contained in the LSDB, or are installed in it if they are new. Regardless of retransmitting or discarding the LSA in question, a router always acknowledges to the sending router the reception of disseminated LSAs using LS Acknowledgment packets. Routers resend the disseminated LSAs to routers that do not acknowledge their reception.

In transit shared links, a non-DR/BDR router sends the LSA to disseminate only to the DR and BDR routers, using the IP address AllDRouters, 224.0.0.6 in IPv4 (OSPFv2) and ff02::6 in IPv6 (OSPFv3). The DR will then disseminate the LSA through the link. If the DR does not, the BDR will disseminate it. The BDR does not need to send LSAs to be disseminated to the DR and it disseminates LSAs directly to the IP address AllSPFRouters.

As stated in Section 2.3.3, LSAs have an age field. LSAs that have an age of 3600 seconds (1 hour) are discarded. OSPF routers must disseminate all LSAs they are responsible for every 30 minutes. For that, routers disseminate a new instance of those LSAs, with the same LSA identifier and a higher Sequence Number, so they are replaced at the routers where they are received. New LSAs and LSA instances start with an age of 0, which is incremented in every receiving router before disseminating it again. Routers can also delete their LSAs by disseminating them with an age of 3600, which OSPF considers to be fresher than LSAs with the same identifier and Sequence Number. Once replaced, they are deleted in the routers as the maximum age was reached.

2.3.10 Initial LSDB synchronization

The initial LSDB synchronization process allows two routers to synchronize their LSDBs after they become adjacent using the Hello protocol, becoming fully adjacent at the end. An important use of the protocol is to allow a router just connected to a network to get to know more rapidly the respective

LSDB(s). The initial LSDB synchronization process is used by both routers in point-to-point links, and between pairs of routers where at least one is DR/BDR in transit shared links. Synchronization happens per neighbor interface, leading to multiple synchronization processes when a router is connected to a neighbor via several links, each one with its neighbor state machine.

All types of OSPF packets are used in the mechanism except the Hello packet. The LS Update and the LS Acknowledgment packets were described in the previous section. In both OSPF versions, LS Request packets carry one or more LSA identifiers, while the most important fields of Database (DB) Description packets are one or more LSA headers, a DD Sequence Number, and 3 bits respectively known as I-bit, M-bit, and MS-bit.

The neighbor state machine described in Section 2.3.8 is continued here, with both routers starting in ExStart state. Figure 2.3 illustrates the neighbor state machine. When in ExStart state, routers set the I-bit when sending DB Description packets. The synchronization of their LSDBs involves exchanging LSDB summaries, carried in DB Description packets, to let the routers know if their neighbor has new or fresher LSAs (the process is known as the database description process), and then the respective request and sending, respectively using LS Request and LS Update packets, if required. Reception of requested LSAs is acknowledged with LS Acknowledgment packets. A master-slave relationship must be established between the two routers in order to exchange the summary of their LSDBs.

Routers start by exchanging DB Description packets without LSA headers in order to determine the master, which will be the router with the highest Router ID. The MS-bit indicates whether the source router is the master or not. A router moves to the Exchange state when either it knows it is the slave (the other router has a higher Router ID) or when the other router confirms it is the slave (the MS-bit in its DB Description packet is not set). After that, the routers exchange DB Description packets with the headers of their LSAs. The respective DD Sequence Numbers are set by the master. The M-bit signals when the source router has more LSA headers to send. The process ends when both routers have sent packets to the neighbor with the M-bit clear, i.e. the router has no further LSA headers to send.

When the database description process is over, the routers know if the neighbor router has new or fresher LSAs in its LSDB. If it does, the router moves to the Loading state and requests them with at least one LS Request packet, carrying the identifiers of the desired LSAs. The LS Update packets not only carry the full requested LSAs but also serve as an acknowledgment of the request. Received LSAs are then installed in the LSDB or replace older instances of existing LSAs. The Full state is reached when either the router has no LSAs to receive after the database description process, or when it has received all of them. A router can be in the Full state while its neighbor router is in the Loading state.

2.3.11 Designated Router election

In order to provide the link identifier for transit shared links and manage the respective LSAs, a DR must be elected in every link of that type. A BDR must also be elected to replace the DR in case of failure. Two parameters are used to determine the DR/BDR routers: the most important is the Router Priority, included in Hello packets. The router with the highest priority becomes the DR, and the one with the second highest priority becomes the BDR. If there is a tie, the router with the highest ID is elected.

In OSPF, interfaces have their own state machine, just like neighbor relationships. The state machine is described in Figure 2.4. Each interface has its instance of the state machine. Turned off interfaces are in the Down state. When turned on, they move to the Point-to-point state in point-to-point links, and to the Waiting state in shared links. There are no further states in point-to-point links. In shared links, interfaces leave the Waiting state either after 40 seconds or after an adjacent router declares itself as DR or BDR.

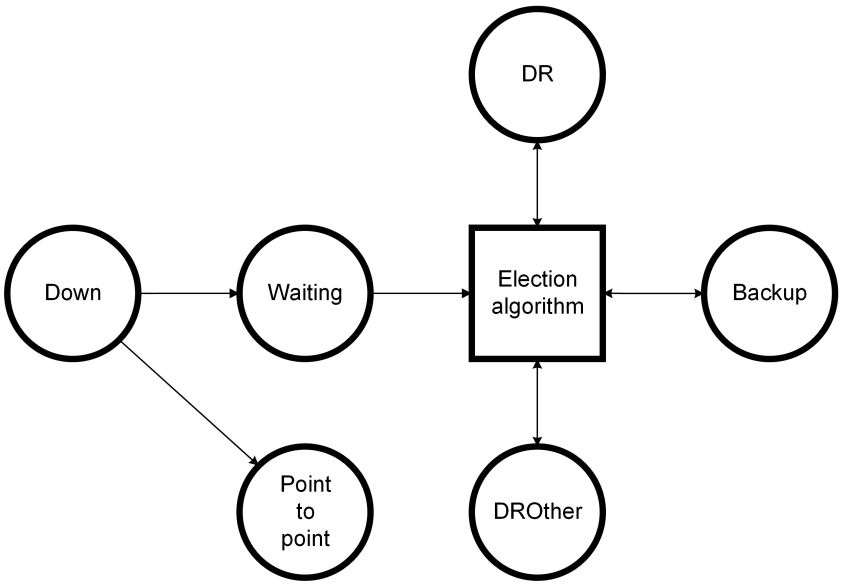


Figure 2.4: OSPF interface state machine in interfaces connected to point-to-point and broadcast links
Source: Adapted from [5]

The router then runs the election algorithm for the interface, taking in consideration the list of its adjacent neighbor routers, their Router IDs and priorities, and which of them (if any) declare themselves as DR or BDR. The information is found on the Hello packets exchanged previously with the neighbor routers. Routers that are already the DR or the BDR are favored in the election. The interface will then move to the DR, BDR, or DROther states, depending on whether the router declares itself as the DR, the BDR or non-DR/BDR in that transit shared link. Events such as new routers being connected to the link or connected ones being disconnected (for example by crashing) lead to the election algorithm being run again in every router in the link. In stub shared links, the sole router will be its DR.

2.3.12 Multi-area OSPF

OSPF networks can be divided in areas, each one with its own LSDB. Division of the network in areas allows to reduce the size of the LSDB stored by each router in the network. Areas are identified by a 32-bit number represented as an IPv4 address, unique in the routing domain. Each interface belongs to only one area. OSPF multi-area networks have a hierarchy of two levels. The top level contains only one area, the backbone area, which must always exist. Its ID is always 0.0.0.0, and it is also known as the area 0. Every other area in the routing domain belongs to the lower level, and must be directly connected to the backbone area through at least one ABR. Direct communication between non-backbone areas is not allowed, as well as interfaces of different areas being connected to the same link.

ABRs are routers who have interfaces in more than one area, with at least one interface in the backbone area. ABRs have a LSDB for every area they belong to. In each area, the respective Router-LSAs only describe their interfaces in the area. Router-LSAs also describe whether a router is an ABR, an ASBR, or an internal router.

Dissemination of routing information between areas is done with a DVR approach. In both OSPF versions, topological information is restricted to the area it belongs to, and only addressing information is disseminated across the routing domain. For every area an ABR is connected to, it disseminates to the other areas both the area address prefixes and the costs of the shortest paths from the ABR to them. The information is stored in distance vectors which bind an address prefix to a path cost.

The shortest path cost of an internal router to an area-external address prefix is calculated by, for every ABR in the area advertising the prefix, adding the path cost advertised by the ABR and the router's own shortest path cost to reach the ABR. The smallest sum indicates the shortest path cost to reach the address and what the respective ABR is. Address prefixes are also disseminated in non-backbone areas, either from the backbone area or from other non-backbone areas. If the disseminated prefix is in another non-backbone area, the ABR that injects it in the area updates the path cost in the distance vector using the described formula before disseminating it.

New types of LSAs must be introduced in order to allow multi-area routing. The distance vectors previously mentioned are disseminated using Network-Summary-LSAs in OSPFv2, and Inter-Area-Prefix-LSAs in OSPFv3. In both cases, a packet carries a single address prefix to be advertised and the shortest path cost from the sending ABR to the prefix. The Link State ID contains the address prefix in OSPFv2, and a local tag generated by the ABR for each advertised prefix in OSPFv3.

Other LSAs must be introduced to allow the dissemination of domain-external information in multi-area OSPF networks. AS-External-LSAs remain as described in Section 2.3.6. They are flooded to the entire routing domain, regardless of its areas. In single-area OSPF networks, all routers know how to reach the ASBRs. In multi-area networks only routers inside the same area will know how to reach them, since topological information does not leave its area. ASBR-Summary-LSAs in OSPFv2, and

Inter-Area-Router-LSAs in OSPFv3, solve the problem. They are disseminated by ABRs with interfaces in areas with ASBRs, and each packet contains the ID of an ASBR (carried in the Link State ID) and the shortest path cost from the advertising ABR to the ASBR. In that way, all routers get to know the ASBRs just as they get to know the area-external address prefixes.

2.3.13 Support for new types of LSAs

OSPF supports the creation of new types of LSAs in order to extend the protocol, which will carry information that can either be used by OSPF or by other protocols and applications. These new LSAs can be flooded and stored in the LSDBs just like regular LSAs. The structure of the headers of the new LSAs remains unchanged, with a small difference in OSPFv2. The packet bodies can have arbitrary structures to carry the desired information, as long as they are structured in 4-byte words.

In OSPFv2, Opaque-LSAs allow storage and dissemination of extra information. There are different types of Opaque-LSAs which determine the scope of their flooding: link-scope, area-scope, or AS-scope. The only difference in their headers compared to the headers of regular OSPFv2 LSAs lies in the Link State ID field, which is now replaced by two fields, Opaque Type and Opaque ID. Opaque-LSAs are not part of the original OSPFv2 specification, and are instead described in [10].

The original specification of OSPFv3 [7] already supports new types of LSAs. In particular, the first bit of the LS Type field in all OSPFv3 LSA headers indicates whether the LSA should be only flooded in the link, or if it should be stored and flooded as any other LSA. The next 2 bits identify the flooding scope of the LSA: link-scope, area-scope, or AS-scope. The remaining bits identify the actual LSA Type.

2.3.14 Building the forwarding table

The goal of a routing protocol is to create and keep updated the forwarding tables of the routers in the network. A router running OSPF creates its forwarding table with the information contained in its LSDBs.

First, a router creates one graph for every area it is connected to. Routers are represented as nodes, while stub shared links are not. Transit shared links and point-to-point links may or may not be represented as nodes. The address prefixes in the area are then assigned to the nodes. In the case of links not represented as nodes, the address prefixes are assigned to the routers connected to them.

With the graphs created, Dijkstra's algorithm is used to find the shortest path from the router node to every other node in the graphs. A shortest-path tree is calculated for every area the router is connected to, with the router itself as the tree root. A description of the process can be found on Section 2.3.2 of [5]. The address prefixes are then assigned to each shortest path. To obtain the intra-area shortest path cost to an address prefix not associated with a link node, for every router connected to the prefix the shortest intra-area path cost to reach the router and the cost of the interface connected to the prefix

are added. The smallest sum provides the shortest intra-area path cost to reach the prefix.

With the information stored in the area shortest-path trees, the router installs in a data structure representing the forwarding table the address prefixes, the shortest path costs to reach them, the interfaces through which packets should be sent to reach the prefixes with the smallest path cost, and the next-hop IP addresses if any. Then, the data is installed in the forwarding table of the router kernel. The shortest-path costs to area-external prefixes are calculated afterwards, followed by the shortest-path costs to domain-external prefixes. A default route can be installed if the area has a single ASBR.

2.3.15 Comparing OSPFv2 and OSPFv3

Sections 2 and 4 of RFC 5340 [7] describe the differences that exist between OSPFv2 and OSPFv3. The most relevant is the separation of the addressing and the topological information in OSPFv3: while OSPFv2 mixes them in the Router-LSAs and in the Network-LSAs, OSPFv3 uses those LSAs only for storing topological information, instead using Intra-Area-Prefix-LSAs to store addressing information and Link-LSAs to store link information. For that reason, the bodies of Router-LSAs and Network-LSAs have differences in OSPFv3 compared to OSPFv2, mainly the removal of address prefixes and IPv4 addresses. The LSA header also has small differences compared to the one used by OSPFv2.

OSPFv3 replaces the network mask in Hello packets with the source Interface ID, and the authentication-related fields in the packet headers with the Instance ID. There are also differences in the way unknown LSAs are handled, although the result is the same. In OSPFv2, different types of Opaque-LSAs allow the dissemination of new information with different scopes, while in OSPFv3 unknown types of LSAs are directly supported along with the indication of the desired flooding scope.

Unlike OSPFv2, multiple instances of OSPFv3 can run in the same router at the same time. OSPFv3 also allows an address prefix to be configured just at one of its connected interfaces, ensuring its dissemination to the other connected routers. Multiple subnets can also be configured in the same link.

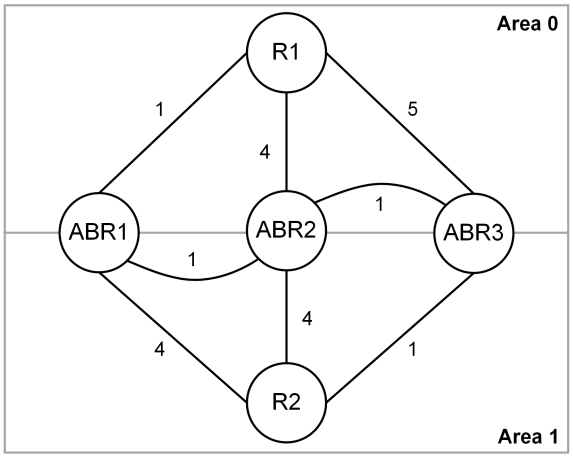
Section 4 of RFC 5340 also describes what remains unchanged in both OSPF versions. In particular, the synchronization mechanisms and the DR election remain unchanged, regardless of the differences found in the LSAs and exchanged packets when comparing the two OSPF versions. The concepts of area, interface and neighbor router, as well as the state machines associated to the interface and to the neighbor router, the structure of the LSDB, and the construction of the forwarding table from it, also are the same both in OSPFv2 and OSPFv3.

2.3.16 Limitations on shortest path calculation

As described in Section 2.3.12, multi-area OSPF networks have limitations in their topologies. The restrictions are motivated by the count-to-infinity problem of DVR, described in Section 2.2.

The DVR approach used by OSPF in inter-area routing also brings restrictions in the selection of the shortest path to a destination. In particular, an area is unable to receive information about paths to internal destinations that cross another area, and ABRs are also unable to provide information about paths to destinations outside the area that at some point cross that area. Additionally, when creating the forwarding table, intra-area paths are always chosen over inter-area paths, even if they have a larger path cost.

Figure 2.5 represents a multi-area network, and Figure 2.6 represents its ABRs overlay. An example of the first restriction is the shortest path from R1 to ABR2, which goes through ABR1 and crosses area 1 with a cost of 2. Since paths to area-internal destinations cannot cross other areas, R1 will instead choose the available intra-area paths with higher cost to reach ABR2.



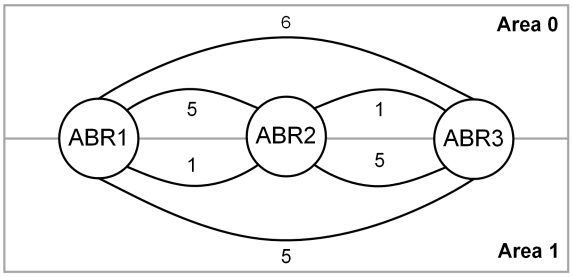


Figure 2.6: Corresponding ABR overlay

Figure 2.5: OSPF hierarchical network
Source: Adapted from [5]

An example of the second restriction is the shortest path from router R2 to router R1, which has a cost of 4 and goes through ABR3, then through ABR2 and finally through ABR1. Since ABRs are unable to advertise paths to area-external destinations that go through the area, the path is not advertised into area 1. Instead, R2 calculates the shortest path cost to reach R1 by, for all 3 ABRs, adding the advertised shortest intra-area path costs to reach R1 and the path cost from R2 to each of the ABRs. The discovered shortest path will have a cost of 5, going only through ABR1.

DVR does not allow routers to have a complete view of the network. In particular, ABRs deal with each area separately and are prevented from crossing information from multiple areas to learn the complete overlay created by all ABRs. For that reason, they are limited in the information that they can learn and disseminate, which prevents the routers in the network from computing the actual shortest paths to destinations in certain cases, as seen in the previous examples.

Chapter 3 will propose a solution to solve the topology restrictions and the limitations when selecting the shortest path to a destination in OSPF.

3

OSPF extensions

Contents

3.1 General description

The main goal of the current MSc Dissertation is to implement two OSPF extensions, one for each version, that will address and overcome the restrictions in shortest path calculation and selection of network topologies currently present in OSPF hierarchical networks. The extensions are described in [1], which will be referred to as the *base article*, and the implementation supporting the MSc Dissertation, present at [11], is based on it with a few changes detailed at Section 3.3.

As described in Section 2.3.16, the use of a DVR approach by OSPF in inter-area routing imposes restrictions in the selection of the shortest path to a network destination, and limits the network topologies to a two-level hierarchy with only one area at the top level. The OSPF extensions described in the base article, one for each OSPF version, propose a LSR approach to inter-area routing to overcome the restrictions. Such approach to inter-area routing is currently not implemented by any LSR technology, as stated in Section 3.2 of [5].

Given the role of ABRs as gateways between areas, they form a routing overlay over an OSPF hierarchical network, similar to having a single-area network running on top of the hierarchical network. In a graph representing the ABR overlay, the nodes are the network ABRs, while the arcs are the shortest intra-area paths between two ABRs connected to the same area. The arc costs correspond to the cost of the shortest path to the neighbor ABR. Internal routers do not form part of the graph, including as well ASBRs that are not ABRs, just like layer-2 equipment such as switches can be abstracted from graph representations of OSPF networks. Figure 3.1 shows a multi-area OSPF network with a topology not supported by the base protocol, while Figure 3.2 shows the equivalent ABR overlay. In the network, R1 is connected to the address prefix ap1, while R2 is connected to the address prefix ap2.

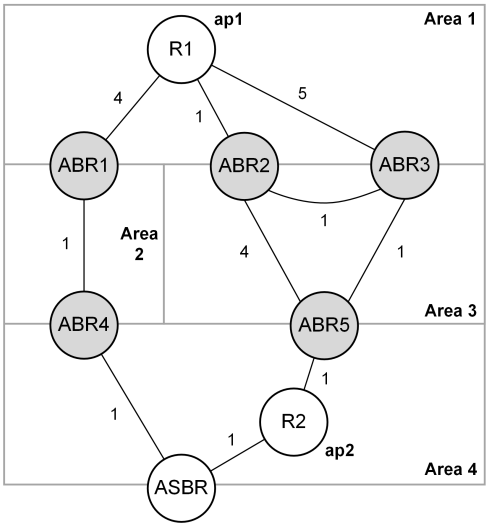


Figure 3.1: OSPF multi-area network

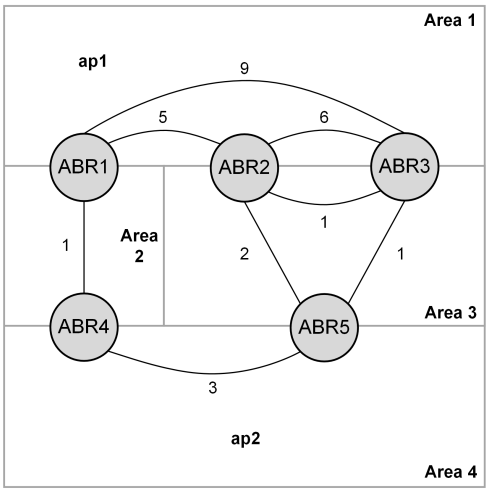


Figure 3.2: Corresponding ABR overlay

Source: Adapted from [1]

As ABRs are the nodes of the ABR overlay graph, the information on the shortest path between any two adjacent ABRs (i.e. router IDs and costs) must be disseminated to the entire overlay so that each ABR can obtain the complete view of the network. Additionally, it is necessary to disseminate the prefixes and the ASBRs associated with each ABR (i.e. in an area it is connected to) along with the shortest path costs from the router to them, so that each ABR can then compute the shortest paths to each prefix and ASBR in the network. This information is supplied to the area internal routers using Network-Summary-LSAs in OSPFv2 and Inter-Area-Prefix-LSAs in OSPFv3, for network prefixes, and ASBR-Summary-LSAs in OSPFv2 and Inter-Area-Router-LSAs in OSPFv3, for ASBRs, the same way it is currently done in hierarchical OSPF.

Each ABR computes its local view of the network using the information stored in the LSDBs of the areas it is connected to. Then, the information is stored and shared between ABRs using 3 new types of LSAs: ABR-LSAs, Prefix-LSAs, and ASBR-LSAs, called *overlay LSAs* in the base article and described in Table 3.1. An ABR will update and flood new overlay LSAs every time it detects a relevant change in one of its LSDBs.

Metric	3
Neighbor Router ID	4
...	
Metric	3
Neighbor Router ID	4

ABR-LSA

Metric	3
Subnet Mask	4
Subnet Address	4
...	
Metric	3
Subnet Mask	4
Subnet Address	4

Prefix-LSA (OSPFv2)

Prefix Number	4
Metric	3
Prefix Length	1
Prefix Options	1
Address Prefix	v
...	
Metric	3
Prefix Length	1
Prefix Options	1
Address Prefix	v

Prefix-LSA (OSPFv3)

Metric	3
Destination Router ID	4
...	
Metric	3
Destination Router ID	4

ASBR-LSA

Table 3.1: Overlay LSAs
Source: Adapted from [1]

Since area internal routers are unaware of the details of inter-area routing and rely solely on the information supplied by ABRs to know about prefixes and ASBRs outside the area, the OSPF extensions are transparent to them and no modification of the internal routers is required. As stated in Section 2.3.13, both versions of OSPF already support new types of LSAs. In particular, since overlay LSAs must reach the entire network regardless of the area, they will have AS-scope. Area internal routers will store and flood overlay LSAs as if their LS Type was known, even if they do not use the information stored in the LSAs. The synchronization mechanisms described in Sections 2.3.8 to 2.3.10 ensure the reliable distribution of the overlay LSAs and therefore the correctness of the network routing information.

Both extensions operate in the same way. The only design difference is in the Prefix-LSA, which has different content in each OSPF version.

3.2 Overlay LSAs

Table 3.1 shows the format of the proposed overlay LSAs. All of them advertise the shortest path cost from the Advertising Router, which is always an ABR, to another ABR, a prefix, or an ASBR, located in the same areas the source ABR is connected to. ABR-LSAs, Prefix-LSAs and ASBR-LSAs advertise costs to ABRs, network prefixes and ASBRs, respectively. The advertised path is always intra-area i.e. it will be calculated using only the Router-LSAs, Network-LSAs and Intra-Area-Prefix-LSAs (only in OSPFv3) present in the area LSDBs, even if the actual shortest path crosses other areas. It is possible for the path to cross an intermediate ABR, as long as the inbound and outbound interfaces belong to the same area as the Advertising Router and the destination.

The content of the overlay LSA bodies only differs between OSPF versions for the Prefix-LSA. The Advertising Router is always the source ABR. In OSPFv2, the overlay LSAs are type 11 Opaque-LSAs (also known as Opaque-AS LSAs), with the LS Type field always set to 11. Considering the IANA assignation of Opaque Types available at [12], which states that values up to 10 are registered (the registration of value 10 is temporary), in our implementation ABR-LSAs, Prefix-LSAs and ASBR-LSAs received a value of 11, 12 and 13, respectively.

OSPFv3 already supports new LSA types. The first byte of the LS Type (the U-bit) is set to True to indicate that the LSA should be stored and flooded just like a regular LSA, while the second and third bits (the S1 and S2 bits) are set to 10 to indicate a AS-scope. IANA has registered LSA Function Codes (which denote the LSA Type in OSPFv3) up to 16, with the registration of value 16 being temporary, as seen at [13]. Therefore, in our implementation, the ABR-LSA, Prefix-LSA and ASBR-LSA are given a value of 17, 18 and 19 as LSA Function Code. This information is summarized in Table 3.2.

	ABR-LSA	Prefix-LSA	ASBR-LSA
OSPFv2 LS Type	11	11	11
OSPFv2 Opaque Type	11	12	13
OSPFv3 LS Type	17	18	19

Table 3.2: LS Type and Opaque Type of overlay LSAs

An overlay LSA can store information on as many network elements of the same type as required, as long as all of them are associated with the same ABR. The set of parameters that describes a network element inside an overlay LSA is repeated for every element being described. For that reason, each ABR produces at most one overlay LSA of each type.

For each described network element, all overlay LSAs store a metric value which consists of the shortest intra-area path cost from the source ABR to the element, either another ABR, a prefix or an ASBR. Prefix-LSAs in OSPFv3 also store the number of prefixes in the LSA. Additionally, each overlay LSA contains the following fields for each element:

ABR-LSA The Router ID of the neighbor ABR

Prefix-LSA In OSPFv2, the subnet address and subnet mask. In OSPFv3, the prefix length and the address prefix, along with its options

ASBR-LSA The Router ID of the neighbor ASBR

3.3 Differences to base article

There are a number of differences between the the OSPF extensions as described in the base article and the implementation supporting the current MSc Dissertation.

In the base article, ABR-LSAs can store information on multiple ABRs, while Prefix-LSAs and ASBR-LSAs can store information on only one prefix or ASBR. For the implementation, it was decided to make all overlay LSAs consistent by allowing all 3 types to store information on multiple network elements on a single LSA. This behavior is seen in the current OSPFv3 specification, in the Intra-Area-Prefix-LSAs and Link-LSAs, which can store multiple prefixes in a single LSA. It was also decided to add a Prefix Number field to the Prefix-LSA in OSPFv3, present as well in the two LSA types of the current specification.

Analysing RFC 2328 [6] for OSPFv2 and RFC 5340 [7] for OSPFv3, it is possible to conclude that Router-LSAs and Intra-Area-Prefix-LSAs use 2 bytes to store the metrics. AS-External-LSAs, Summary-LSAs of both types, Inter-Area-Prefix-LSAs and Inter-Area-Router-LSAs use 3 bytes to store metrics. In the first case the metric is an interface cost, while in the second case the metric is the sum of interface costs. In overlay LSAs, the metric is a sum of interface costs as well. For that reason, it was decided to increase the length of the metric field from 1 to 3 bytes in all overlay LSAs for the implementation.

Section 3 of the base article proposes a mechanism similar to the OSPF initial LSDB synchronization process, with new messages, where an ABR that has just joined the network can request its overlay LSAs to the ABRs in the same areas. As described in Section 4.2.2 of [14], since the overlay LSAs are Opaque-AS-LSAs which are recognized by OSPF, they are already exchanged during the initial LSDB synchronization process between the new ABR and its neighbor routers even if they are area internal routers. Given that in OSPFv3 new LSAs can be stored and flooded like regular LSAs if their U-bit is set (which happens for the overlay LSAs), it can be concluded that there is no need to implement an additional mechanism to perform the initial LSDB synchronization process. However, OSPFv2 routers must signal their support of Opaque-LSAs by setting the O-bit in the Options field of DB Description packets. Otherwise, they will not receive Opaque-LSAs during the initial LSDB synchronization process.

3.4 Practical example

Table 3.3 shows the overlay LSAs for the network from Figures 3.1 and 3.2. It considers the address prefix ap1 connected to R1, in area 1, and the address prefix ap2 connected to R2, in area 4. All costs

shown in the overlay LSAs are costs of intra-area paths, including the costs of paths between ABRs.

ABR1
ABR2: 5
ABR3: 9
ABR4: 1

ABR2
ABR1: 5
ABR3: 1
ABR5: 2

ABR3
ABR1: 9
ABR2: 1
ABR5: 1

ABR4
ABR1: 1
ABR5: 3

ABR5
ABR2: 2
ABR3: 1
ABR4: 3

ABR-LSAs

ABR1
ap1: 4

ABR2
ap1: 1

ABR3
ap1: 5

ABR4
ap2: 2

ABR5
ap2: 1

Prefix-LSAs

ABR4
ASBR: 1

ABR5
ASBR: 2

ASBR-LSAs

Table 3.3: Overlay LSAs for the network from Figures 3.1 and 3.2
Source: Adapted from [1]

ABRs 1, 2 and 3 are connected to area 1, with all intra-area paths going through R1. ABR1 shares only area 1 with the other 2 ABRs, so the path costs to reach ABR2 and ABR3 that are introduced in its ABR-LSA are the ones that go through R1. On the other hand, ABR2 and ABR3 are both connected to area 1 and to area 3, with the shortest intra-area path between the 2 routers going through area 3 with cost 1. Therefore, for the ABR-LSAs of both routers, a cost of 1 to reach the other ABR is inserted. ABR3 does not insert in its ABR-LSA the shortest path cost of 5 to reach ABR1 by going through areas 3, 4 and 2, since it is an inter-area path, neither the path cost of 6 that can be obtained by going through areas 3 and 1. On the other hand, ABR2 puts in its ABR-LSA a shortest path cost of 2 to reach ABR5 despite being directly connected to it, since the path that goes through ABR3 is still an intra-area path even if it goes through an ABR.

In the network, ap1 and ap2 are being advertised inside area 1 by R1 and inside area 4 by R2, respectively. ABRs 1, 2 and 3 are connected to area 1, and therefore create and advertise Prefix-LSAs stating that ap1 is in a directly connected area, with the shortest intra-area path cost being the one that directly reaches R1. The same happens for ABR4 and ABR5 regarding ap2 and R2.

The network has a single ASBR, an area internal router in area 4. Given that only ABR4 and ABR5 are connected to the area, they are the only routers to create and flood an ASBR-LSA with the shortest intra-area path cost (which is also the overall shortest path cost) to reach the ASBR from them.

After the network stabilizes and all ABRs know all of the network overlay LSAs, they can compute the overall shortest path costs to reach the network prefixes and the ASBR. By using the network ABR-LSAs to build the overlay graph, each ABR will know the overall shortest path cost to reach all other ABRs in the network. For example, ABR1 will know that its overall shortest path costs to reach ABR2, ABR3, ABR4 and ABR5 are 5, 5, 1 and 4, respectively. Prefix-LSAs and ASBR-LSAs tell ABRs which prefixes and ASBRs are associated with each ABR, respectively, along with the intra-area shortest path

costs from the ABR to the network element. ABR1 learns that ABR4 can reach the ASBR with cost 1 and ap2 with cost 2, while ABR5 can reach the ASBR and ap2 with cost 2 and 1, respectively. By adding the overall shortest path cost to each ABR with the intra-area shortest path cost from the ABR to the network element, ABR1 can obtain the overall shortest path cost from itself to the network element. To reach ap2 and the ASBR from ABR1, the next hop router will be ABR4.

With the shortest path costs calculated, ABRs create Network-Summary-LSAs in OSPFv2 and Inter-Area-Prefix-LSAs in OSPFv3, for prefixes, and ASBR-Summary-LSAs in OSPFv2 and Inter-Area-Router-LSAs in OSPFv3, for ASBRs, containing the overall shortest path costs to reach them. ABR1, ABR2 and ABR3 will tell R1 that the shortest path cost from them to ap2 is 3, 3 and 2, respectively, and 2, 4 and 3 to reach the ASBR, respectively. By summing the costs advertised by the ABRs with the intra-area shortest path costs to reach each of the area ABRs, R1 can conclude that the shortest path cost to reach ap2 and the ASBR through ABR1 is 7 and 6, through ABR2 is 4 and 5, and through ABR3 is 7 and 8, respectively. With the information, R1 can now select ABR2 as the next hop router to reach ap2 and the ASBR, knowing that the overall shortest path cost to reach ap2 and the ASBR is 4 and 5, respectively.

4

Software architecture

Contents

4.1 General description

The program that supports the current MSc Dissertation consists of an OSPF router, able to be interconnected with other machines running either the same router program or current OSPF implementations. The program runs both OSPF versions as defined in the current specifications and supports Opaque-LSAs as well, with the OSPF extensions being created on top of the base implementation. ASBRs and domain-external address prefixes are not supported.

The software architecture of the program is based on the OSPF specifications as described in RFC 2328 [6] and RFC 5340 [7], and also on the OSPF implementation described in [15]. The architecture is object-oriented, with an hierarchy of four software layers which from top to bottom are: the router, the area, the interface, and the neighbor. The architecture is illustrated in Figure 4.1. Grey lines represent the interactions between layers, and arrows indicate the direction of the interaction.

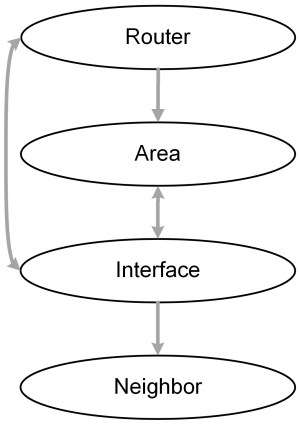


Figure 4.1: Software architecture of the supporting program

Router neighbors are directly associated with router interfaces, to the point where a router can have different neighbor relationships with the same neighbor router through different interfaces. For that reason, the neighbor layer can be placed under the interface layer.

Organizing the router, area and interface layers is less direct. Routers have at least one interface, which will belong to one and only one area. Areas also have at least one interface, regardless of the routers those interfaces belong to. Therefore, a router can belong to a single area while having several interfaces, or at most belong to as many areas as the interfaces it has. Given that, the router layer can be placed at the top of the hierarchy, with the area layer immediately below and the interface layer below it, leaving the neighbor layer as the bottom layer of the hierarchy.

Despite the differences between OSPFv2 and OSPFv3, the software architecture is suitable for both OSPF versions. The differences and similarities of OSPFv2 and OSPFv3 are summarized in Section 2.3.15. In particular, state machines, mechanisms, and the OSPF base concepts such as the area,

the interface and the neighbor router remain unchanged. The main consequences of the separation of topological and addressing information in OSPFv3 are the creation of new types of LSAs, and changes in the structure and content of existing LSA types. These changes are accommodated by the software architecture and only affect the implementation.

Figure 4.2 shows the UML class diagram of the router program, containing its relevant classes organized by layers. The top layer contains four classes: the router, the OSPF routing table, the interface to the routing table stored in the kernel of the machine running the program, and the part of the LSDB that stores the overlay LSAs. The area layer contains the area class and the remaining LSDB, where the regular LSAs are stored. One router instance is associated with at least one area instance. Both the interface and the neighbor layers contain a single class, which respectively represents an OSPF interface and an OSPF neighbor router from the point of view of an interface. An area instance is associated with at least one interface instance, while the latter may have no neighbor instances associated.

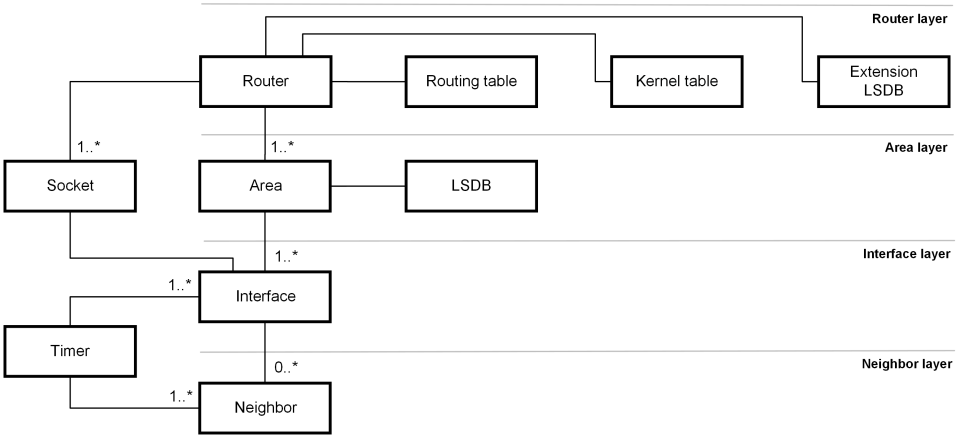


Figure 4.2: UML class diagram of the supporting program

A few classes support the program operation without belonging to a specific layer, being instantiated by classes in more than one layer. Both router and interface instances are associated with at least one instance of the socket class, while both interface and neighbor instances are associated with at least one instance of the timer class.

4.2 Communication between layers

Figure 4.1 shows the communication that exists between the different software layers of the program. Six possible communications between different software layers can occur if bidirectional communications are counted twice, although one layer can communicate with another for more than one reason. The router layer has bidirectional communication with the interface layer, and unidirectional communication with the area layer. The interface layer is the only one to communicate with all other layers, by additionally having

bidirectional communication with the area layer and unidirectional communication with the neighbor layer. The bottom software layer is the only one not to communicate with any other.

In the program, synchronous communication between layers occurs when a thread running in one layer calls a method of a class in another layer, while asynchronous communication between layers occurs when a thread running in one layer sends a message to a thread running in another layer.

Figure 4.3 shows UML sequence diagrams for three different situations, which can occur in the same program execution, where all possible communications between different software layers occur. It is possible for the same software layers to communicate for other reasons not present in the scenarios described by the diagrams.

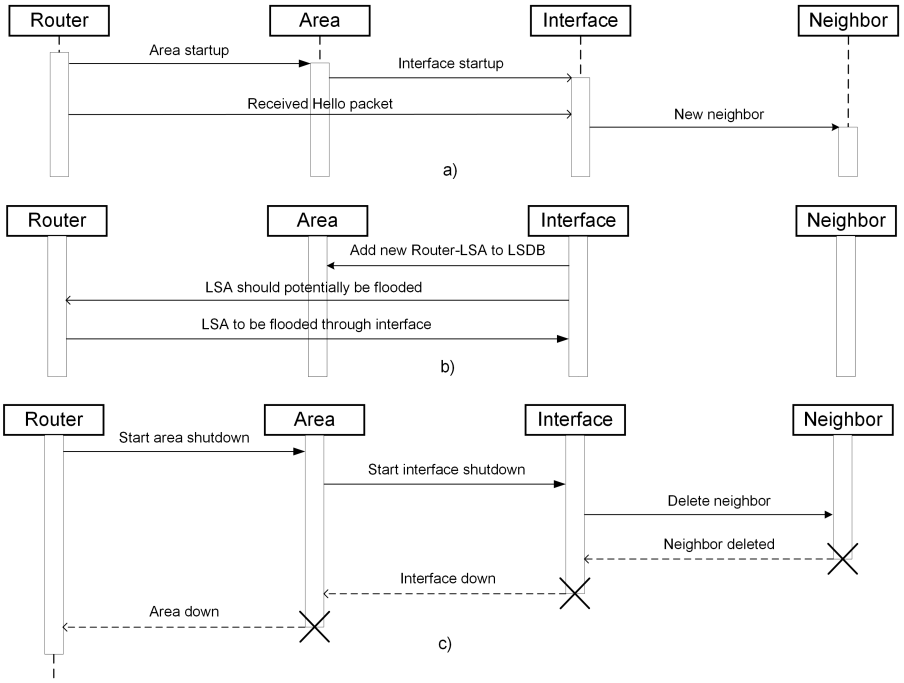


Figure 4.3: UML sequence diagrams of the supporting program for three different scenarios; a) Router startup, b) Interface creates new instance of own Router-LSA, c) Router shutdown

Diagram a) shows the message exchange between layers during the router startup. Immediately after the start, the router layer sends a synchronous message to the area layer to start it. As part of the area startup, the layer sends an asynchronous message to the interface layer in order to start it. The message exchange between layers related to the router startup is now complete. Later, the router layer receives an Hello packet from the network and forwards it in an asynchronous message to the interface layer. It is an Hello packet from a new neighbor, therefore the interface layer sends a synchronous message to the neighbor layer in order to create a new neighbor.

Diagram b) shows the message exchange between layers when an interface creates a new LSA or a new instance of an already existing one. First, the interface layer sends the LSA to the area layer to

be stored in the area LSDB, in a synchronous message. Immediately after, the interface layer sends the same LSA to the router layer, which will decide which interfaces should flood it, if any. As the LSA was created by the router, every interface should flood it and therefore the router layer sends the LSA back to the interface layer to be flooded.

Diagram c) shows the message exchange between layers during the router shutdown. First, the router layer sends a synchronous message to the area layer to start its shutdown process. As part of the process, the area sends another synchronous message to the interface layer to start its shutdown process, which triggers a third synchronous message from the interface layer to the neighbor layer in order to delete the existent neighbor. The neighbor is deleted and the interface layer shuts down after receiving the confirmation that the neighbor has been deleted. The area layer also shuts down after receiving the confirmation that the interface layer is down. Finally, the router layer shuts down after the confirmation that the area layer, and its network elements, are down as well.

4.3 Support classes

Two support classes are included in the software architecture: timers and sockets. They are not included in any software layer, and instead classes in more than one layer can instantiate them as needed.

Timers are described in Section 4.4 of the RFC 2328 [6] as a basic requirement for any implementation of OSPF, with a required granularity of one second. Two types of timers must be created, one-shot timers and interval timers. One-shot timers fire only once, at the end of a specified number of seconds, while interval timers fire every time a certain number of seconds has elapsed. As an example, the first timer type is suitable for telling an interface in Waiting state that 40 seconds have elapsed, allowing it to proceed to the next state. The second timer type is suitable for timing the sending of Hello packets, since they must be sent out of an interface every 10 seconds.

Sockets interact with the network, receiving and sending OSPF packets. The same section of the RFC says that it is necessary to support the receiving and sending of IP multicast packets, along with IP unicast packets. The multicast IP addresses to be used are 224.0.0.5 and 224.0.0.6 in OSPFv2, and ff02::5 and ff02::6 in OSPFv3. Sockets instantiated by the router layer receive OSPF packets from the network and forward them to the router layer, while sockets instantiated by the interface layer receive OSPF packets from it to be sent to the network.

4.4 Router layer

The router layer contains the OSPF top-level data structures, as described in Section 5 of [6], such as the Router ID, the list of area data structures for the areas the router belongs to, and the OSPF forwarding

table also mentioned in Section 2.3.14.

The router class controls directly or indirectly all layers of the program. It creates an instance of the area class for each area the router is connected to, according to the starting configuration. The router class then communicates directly with the interface layer after the router startup. The router class has sockets, one for each interface, that listen to the network and receive any OSPF packets that reach the machine running the program. The packets are then forwarded directly to the corresponding interface instance in order to be processed.

The interface layer also sends LSAs to the router class to be potentially flooded, which the router class processes. If any LSA should be flooded, according to the content of the LSDBs and of the LSA, the router class then sends the LSA to the right interfaces in order to flood it.

Another task of the router class is to update the forwarding table of the kernel of the machine running the program. The router layer has a class for storing the OSPF routing table, modeled after the routing table described in Section 11 of the RFC 2328 [6]. It is built from the area LSDBs every time a relevant change is detected. Then, its information is transferred to the kernel routing table, using an interface that allows to create, read and delete routes in the kernel routing table. The interface is also a class of the router layer.

The router class is responsible to start all other layers and signal their shutdown. The startup of the router layer triggers the startup of the area and the interface layers (the neighbor layer is only started after the first neighbor router is found), and the shutdown of the router layer triggers the shutdown of the remaining software layers, one after the other. The router layer is always the last to complete its shutdown process, as it waits for the shutdown of the other layers to complete.

4.5 Area layer

The area layer contains the OSPF area data structures, as described in Section 6 of [6]. Each area data structure contains elements such as the Area ID, the interfaces connected to the area, and most of the LSAs defined in the base OSPF specifications in use by the supporting implementation. In particular, Router-LSAs and Network-LSAs in both OSPF versions, Network-Summary-LSAs in OSPFv2, and Intra-Area-Prefix-LSAs and Inter-Area-Prefix-LSAs in OSPFv3, are stored in the area layer.

During the router startup, the area instances start the interface layer, reading the starting configuration and creating one or more interface instances each. It is also a task of the area layer to initialize the router LSDBs with their initial content. For both OSPF versions, in our implementation each area LSDB will always start with a Router-LSA. Initially, in OSPFv2 for every interface there will be a link to a stub network in the Router-LSA, while in OSPFv3 the Router-LSA will have no links. Also in OSPFv3, each LSDB instance will start not only with a Router-LSA but also with an Intra-Area-Prefix-LSA with the

prefixes of the area interfaces.

In our implementation, the area-scope LSAs of the area LSDB are stored in a LSDB class instance. The LSDB class is a data object that stores LSAs, where LSAs can be added to an instance, and deleted and retrieved from it. Both the router and the interface layers access and manipulate the content of the LSDB class instances. The router layer can use the LSDBs content to obtain the directed graphs of the areas. By running the Dijkstra's algorithm on each one, it is possible to obtain the area shortest path trees, which the router layer can use to create the OSPF routing table.

4.6 Interface layer

The interface layer contains the OSPF interface data structures, as described in Section 9 of [6]. Each interface data structure contains elements such as the IP address, the network DR and BDR, the current interface state, the interface cost, the list of OSPF neighbors associated with the interface, and the list of the interface Link-LSAs for OSPFv3. Unlike the upper software layers, the interface layer only has one class, the interface class.

The interface layer uses both support classes available in our implementation. Each instance creates a socket to send OSPF packets. Two timers are used by each instance, an one-shot timer to signal when the interface has been in the Waiting state for 40 seconds, and an interval timer for timing the sending of Hello packets to the network.

During the router startup, in OSPFv3, each interface instance generates a Link-LSA, which is stored in the instance. Link-LSAs are the only LSAs defined by the current OSPF specifications in use in our implementation that are not stored in the area layer.

Each interface instance has its instance of the interface state machine, as described in Section 2.3.11. The state machine may change its state as a result of the processing of incoming packets, the lack of reception of Hello packets from the neighbors during 40 seconds, and the reception of a shutdown signal from the area layer.

Each interface instance performs a number of tasks inside the router program:

- Sending an Hello packet to the network every 10 seconds
- Sending new LSAs or LSA instances to the router layer, for the router class to analyse if they should be flooded. The LSAs can be either self-originated or from another router
- Flooding all LSAs received from the router class to the network. The LSAs are supplied either by the interface itself or by another interface. LSAs are flooded inside LS Update packets
- Processing incoming OSPF packets, as they arrive to the router layer and are forwarded to the corresponding interface instance. The packet processing may lead to the creation, removal or

update of self-originated LSAs.

- Updating the area LSDB, both with self-originated LSAs or from other routers. LSAs can be added, updated of deleted
- Managing the interface neighbors, by creating or deleting them, by changing their state, or by resetting their inactivity timer. New neighbors are created when Hello packets from neighbors are received. Neighbors are deleted if no Hello packets are received from them during 40 seconds, or as part of the interface shutdown process
- Retransmitting the last DB Description, LS Request or LS Update packets to any neighbor routers that do not reply to them in 5 seconds
- Electing the DR and the BDR of the subnet

Some of the tasks can be performed in sequence. For example, a LS Update packet with new LSAs may be received by the router layer, which forwards it to the corresponding interface instance. The instance processes the packet, updates the area LSDB, and sends the new LSAs to the router layer for potential flooding. The new LSAs should be flooded through all interfaces except the interface that has received them, therefore the router layer sends the LSAs to all other interface instances, which flood them to the network inside LS Update packets. The packets are resent to any neighbor that does not acknowledge them in 5 seconds.

4.7 Neighbor layer

The neighbor layer contains the OSPF neighbor data structures, as described in Section 10 of [6]. Each neighbor data structure contains elements such as the current neighbor state, the Router ID and the IP address of the neighbor, and the neighbor inactivity timer. Like the interface layer, the neighbor layer has only one class, the neighbor class.

Each neighbor instance has a one-shot timer, which fires if no Hello packets are received from the neighbor router during 40 seconds. The timer is reset after an Hello packet is received. In our implementation, other one-shot timers are used to measure the time since the last DB Description, LS Request and LS Update packets were sent to the neighbor router. If 5 seconds elapse without a reply, the interface associated with the neighbor resends the packet to the neighbor. The timers are stopped after a reply is received.

The neighbor layer is the only software layer that is not started during the router startup. Instead, neighbor instances are created by the interface layer only after the reception of Hello packets from the

neighbor routers. If the router is not connected to any other router, it is possible for the router to operate without starting the neighbor layer.

Similar to the interface instances, each neighbor instance has its instance of the neighbor state machine, as described in Sections 2.3.8 and 2.3.10. Every change to the current state of the neighbor state machine of one neighbor is performed by the corresponding interface.

The neighbor class has no other tasks beyond storing timers and information about the neighbor router, such as the list of LSAs to request when an adjacency is being formed with the neighbor router. The neighbor layer is the only layer in the program that does not communicate with any other.

4.8 OSPF extensions

Changes in the router and the interface layers were required to accommodate the OSPF extensions implemented for the current MSc Dissertation. Most changes were performed in the router layer.

A fourth class was added to the router layer, a data object for storing the overlay LSAs. The class is similar to the LSDB class of the area layer, storing overlay LSAs and allowing the addition, update and removal of elements from their instances. The router class creates a single instance of the overlay LSDB class. It was decided to store the overlay LSAs in the router layer since the current OSPFv3 specification, described in Section 4.1 of [7], assigns unknown LSAs with their U-bit set and AS-scope to the top-level data structure, which matches the characteristics of the overlay LSAs.

The process of building the OSPF routing table has been updated, compared to the process of the current OSPF specifications as described in Section 2.3.14. After the intra-area OSPF routing table is generated, the self-originated overlay LSAs are updated using the routing table content, and are flooded to the network. Then, the OSPF routing table is updated using only the information contained in the overlay LSAs. Unlike in current OSPF specifications, it is possible for the intra-area shortest paths to be replaced by inter-area shortest paths to the same destination if the inter-area paths are shorter.

The information of the updated OSPF routing table is then transferred to the kernel routing table. Network-Summary-LSAs and Inter-Area-Prefix-LSAs are created, installed in the respective area LSDB and flooded with the information of the updated OSPF routing table.

The interface layer required less changes. As the overlay LSAs in both OSPF versions are stored and flooded like regular LSAs, the processing of incoming OSPF packets could remain intact. Only the access to the LSDB was updated. In order to add, remove or update an overlay LSA, an interface instance will directly access the extension LSDB in the router layer. Regular LSAs continue to be added to, and fetched and deleted from the area LSDB.

5

Implementation

Contents

5.1 Overview

The implementation supporting the current MSc Dissertation is available at [11], and was developed using Python 3.6 [16] in machines running Ubuntu 18.04 [17]. The structure of the program follows the software architecture defined in Chapter 4. In particular, the root directory of the project contains a Python package for each software layer defined previously. Each class defined in the previous Chapter is stored in a different module named after the class, and stored inside the corresponding package.

The implementation has a total of eight packages, which are described in Table 5.1. The area, the interface, the neighbor and the router packages contain the classes of the respective layers as defined in Chapter 4. The general package contains the socket and the timer classes, along with a third class which stores utility functions that are used throughout the program. The initial configuration is stored in the configuration package as a list of parameters. All classes representing the LSA and the packet content are stored in the LSA and the packet packages, respectively.

Area	**Conf**	**General**	**Interface**	**LSA**	**Neighbor**	**Packet**	**Router**
Area	Conf	Sock	Interface	Body	Neighbor	Body	Extension LSDB
LSDB		Timer		Extension ABR		DB Description	Kernel Table
		Utils		Extension Prefix		Header	Router
				Header		Hello	Routing Table
				Inter-Area-Prefix		LS Acknowledgment	
				Intra-Area-Prefix		LS Request	
				Link		LS Update	
				LSA		Packet	
				Network			
				Router			
				Summary			

Table 5.1: Modules of our implementation organized by packages

Each class contains the code and the parameters for both OSPF versions, when required. The program can run both OSPF versions at the same time, each version in a separate process with different instances of the same classes. No state is shared between the processes. A parameter containing the OSPF version to be run is passed around the entire program, determining its behavior.

The interaction with the user is provided by a command-line interface, which is available at every moment after the router program is started. The different commands available allow to access the content of the kernel routing table, the content of the router LSDBs, and the information about the router neighbors, among other functionalities. The commands are the same for both OSPF versions, and fetch the content from the two router processes if both OSPF versions are running at the same time.

Automated testing is provided with the source code. More than 200 automated tests are stored in a ninth package in the root folder of the project. Each test focuses mainly on one source code module, and tests focusing on the same module are grouped in the same test module. Test modules are grouped in test packages with the same name of the corresponding packages of the source code.

5.2 Language and external packages

The language used in our implementation is Python 3.6 [16]. While Python is slower than C++, the language used in the implementation described at [15], Python is an easier and more readable language. The main objective of the supporting implementation is to provide a proof of concept of the OSPF extensions described at [1], where obtaining the fastest possible execution speed is not a priority. Python 3 is used since it is the most recent version of Python.

The program uses the CPython implementation of Python. CPython features a global lock known as the Global Interpreter Lock [18], which prevents parallelism between threads in the same process even in machines with multi-core processors. For that reason, processes are used in the program when parallelism is required to maintain an acceptable execution speed, such as when running both OSPF versions at the same time.

Two external Python packages must be installed in order to run the program. The *netifaces* package, available at [19], provides a simple way to obtain the addresses of the interfaces of the machine running the program. The *timeout-decorator* package, available at [20], allows automated tests to be stopped and failed after a certain number of seconds, if they do not complete or fail until then. It prevents tests from running indefinitely in case of an undetected failure.

5.3 Source code

5.3.1 User interface, startup and shutdown

The user interface of the program is a custom command-line interface, created with the *cmd* Python library. With a terminal, the user must go to the *src* folder of the project, and then run the command *python3 main.py*. As the program manipulates sockets, it is necessary to run the program as root. The output of the program during the startup, and a list of available commands, can be seen in Figure 5.1.

The starting script is stored in the file *main.py*, located in the root directory of the project. Once started, the script will activate the IP packet forwarding for IPv4 and IPv6, allowing the machine running the program to act as a router. Then, the program will disable the IPv6 address autoconfiguration and delete the autoconfigured IPv6 addresses if any exists, as the addresses will cause errors in the program. A command prompt will then appear, asking the user if both OSPF versions, just OSPFv2, or just OSPFv3, are to be run. The user must press 1, 2 or 3, respectively, and then press ENTER. If the prompt does not match a valid value, the prompt will appear again.

Each router process receives two atomic events and one atomic pipeline during the startup. The pipeline allows the main process to send commands to the router processes. One of the atomic events is used by the router processes to tell the main process that the command has been complied with. Each

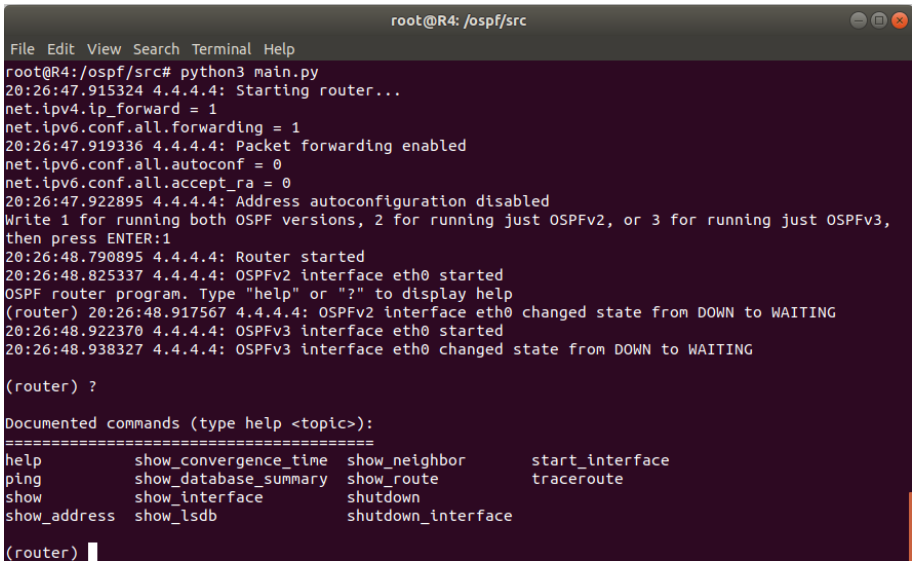


Figure 5.1: Program output during the startup and available commands in the command-line interface

router process processes one command at a time, to avoid mixing the output of separate commands. The second atomic event is set by the main process after the shutdown command is received, signalling the router processes to stop as well. The program exits after the router processes have stopped.

Once the program starts, the command-line interface becomes available. The code of the command-line interface is located in the file *main.py*, where the *cmd* library automatically reads and processes the commands. If both OSPF versions are running, the command will perform the operation or print the information for both router processes. The available commands are presented in the following list. Unless otherwise stated, the commands do not receive arguments.

help Receives one argument, which can be any of the available commands. The command shows a description of the inputted command and an example.

ping Receives one argument, the IP address to ping. The command sends 5 pings to the destination address with no interval between them.

show Equivalent to the commands *show ip ospf* and *show ipv6 ospf* of Cisco routers. The command shows general information about the router processes, such as the interfaces in each area.

show_address Equivalent to the commands *ip address* and *ip -6 address* of the Linux Bash. The command shows the MAC and the IP addresses of the interfaces of the machine.

show_convergence_time Shows the router start time, and the time since the router startup until the last update to the kernel routing table.

show_database_summary Equivalent to the commands *show ip ospf database* and *show ipv6 ospf database* of Cisco routers. The command shows a summary of all of the LSAs stored in the router.

show_interface Equivalent to the commands *show ip interface* and *show ipv6 interface* of Cisco

routers. The command shows information about the OSPF interfaces, such as the neighbor routers and the time until the next Hello packet is sent by the interface.

show_lsdb The closest equivalents are the commands *show ip ospf database* and *show ipv6 ospf database* of Cisco routers. The command prints to the screen the full content of all of the LSAs stored in the router, in text format.

show_neighbor Equivalent to the commands *show ip ospf neighbor* and *show ipv6 ospf neighbor* of Cisco routers. The command shows the neighbor routers known to the program and their information, such as the Router ID and the time until the neighbor is declared dead.

show_route Equivalent to the commands *ip route* and *ip -6 route* of the Linux Bash. The command prints the entire content of the kernel routing table in text format, including the prefixes known to the machine and the respective next hop addresses.

shutdown Performs the shutdown of the program.

shutdown_interface Receives one argument, the ID of the OSPF interface to shut down. The command shuts down the specified interface, if the interface is up.

start_interface Receives one argument, the ID of the OSPF interface to start. The command starts the specified interface, if the interface is down.

traceroute Receives one argument, the destination IP address. Prints the packet route to the provided destination address.

The program command-line interface has an autocomplete functionality, like the Linux Bash. By pressing TAB with a part of a command written, the terminal either prints all of the available possibilities, or the command is autocompleted if there is only one possibility.

5.3.2 Configuration and constants

In our implementation, most constants are stored in the *conf.py* file, in the configuration package. The file does not contain any classes, consisting in a list of static parameters. The static parameters defined by the current OSPF specifications, such as the time to wait after sending each Hello packet, the time to wait until a neighbor is declared dead if no Hello packets are received from the neighbor, and the types of the LSAs and the packets, are stored in the file.

The configuration file is accessed by most of the classes of the program. The file provides the starting configuration for the program: the Router ID, the interface costs, the physical interface IDs (distinct from the OSPFv3 interface IDs), and the interface areas. The configured interface IDs must match the physical (or virtual) interface IDs of the machine running the program. The starting configuration can be changed as required, as long as the changes are performed before the program is started. The program does not support changes to the static parameters during runtime.

5.3.3 General package

The general package contains classes that do not belong to a specific software layer of the program, and are used by classes in more than one layer. The socket and the timer classes are stored in the general package, along with a third class containing utility functions used throughout the code.

The socket class performs two main tasks, sending and receiving OSPF packets. Each task is performed by two methods, one for each OSPF version. Sockets are instantiated by the router and the interface classes, which use sockets to receive and to send packets, respectively.

In our implementation, the packet reception is handled by the router class. During the startup, the router class instance creates one socket class instance for each physical interface in the machine. The instances then listen for OSPF packets in the network, in separate processes. Each arriving packet is forwarded by the router instance to the corresponding interface instance, which processes the packet. The sending of the OSPF packets is handled by the interface class. Each interface instance creates a socket class instance, distinct from the socket instances created by the router class, which is used to send packets to the network through the corresponding physical interface.

In order to send OSPF packets to the network, each sending method receives the packet content converted to a byte stream, along with the destination IP address and the ID of the machine interface to bind to. The method will then create a socket using the Python internal libraries, which will bind to the specified physical interface, and send the OSPF packet encapsulated in an IP packet to the network. As the method returns immediately after sending the packet, there is no need to create new threads or processes in order to send OSPF packets.

In order to receive OSPF packets from the network, each receiving method receives an atomic pipeline, two atomic events, and the ID of the machine interface to bind to. The method will create as well a Python socket bound to the specified physical interface, and enter in a loop controlled by one of the received atomic events, the shutdown event. The socket will continually listen for OSPF packets from the network, which are converted from byte streams to Python data objects, and put in the atomic pipeline to be fetched by the router class. Each method is run in a new process. The second atomic event tells the socket if the router is DR/BDR in the network the interface is connected to. If it is, then the socket will listen to packets sent to the multicast IP address AllDRouters. Packets sent to the address AllSPFRouters are always received. When the shutdown event is set by the router class, the process leaves the receiving loop and terminates.

The timer class provided two main methods, one for each timer type available. Both OSPF versions use the same timers, therefore no differences need to be implemented. Timers run in separate threads. Both timer methods receive a shutdown event and a timeout event, while the single-shot timer method receives a reset event as well. Each timer thread runs until either the shutdown event is set or, for one-shot timers, if they reach a timeout. One-shot timers can be reset by setting the reset event. The timers

inform that they have reached the timeout by setting the timeout event.

The utility class contains only static methods, which are used throughout the program. Examples include the conversion of IP addresses to decimal numbers and vice versa, obtaining the IP address of a physical interface given its physical ID, and calculating the checksum of the packets and the LSAs.

5.3.4 LSAs and packets

In our implementation, LSAs and OSPF packets are represented using data objects. The LSA and the packet headers are represented as different classes. Each LSA body type and each packet body type are represented as different classes as well. Each data object can be packed into a byte stream in order to be sent to the network, or unpacked from a byte stream received from the network. Although the LSA and the packet classes are used by more than one software layer, given their particular role and their number, it was decided to place the classes in separate packages and not in the general package. The classes of the LSA and the packet packages can be seen at Figure 5.2 and at Figure 5.3, respectively.

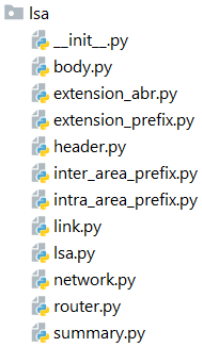


Figure 5.2: Classes of the LSA package

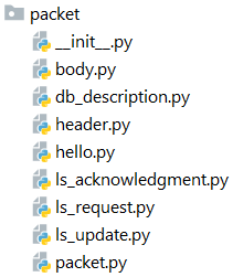


Figure 5.3: Classes of the packet package

In the LSA and the packet packages, a header class contains the respective header fields. Most of the fields are common to both OSPF versions. Fields unique to a certain OSPF version have a comment stating the OSPF version they belong to. A method to pack the header into a byte stream, and a static method to create an header instance from a byte stream, are present in both header classes.

The LSAs and the packets have common headers in each OSPF version, with the differences between different LSA and packet types being present in the respective bodies. In our implementation, each LSA and each packet type is represented as a different class which stores the elements present in the respective bodies for both OSPF versions, similar to the header classes. All body classes inherit from the same abstract class and must implement the methods that pack and unpack the bodies.

The LSA and the packet packages includes a class that stores the complete LSA or packet, respectively, storing the respective header and body as fields. The classes are also an interface to the LSA and the package manipulation, respectively, with methods that allow an easier creation of the LSA or the

packet, and call the available methods in the header and the body classes, without the need to directly access the header or the body classes. After a call to a method that changes the LSA or the packet content, the respective length and checksum are recalculated.

5.3.5 Router package

The router package contains the classes of the router layer defined in Chapter 4. The class containing the overlay LSAs will be described in Section 5.3.9.

The router class controls directly or indirectly all layers of the program. It is one of the largest classes of the program, with more than 1.000 lines of code. During the program startup, a router main process is started, which controls the operation of the corresponding OSPF version in the program. It creates the area class instances according to the initial configuration, leading to the creation and startup of the interface instances. The router socket instances are created, along with the processes that will listen for OSPF packets from the network. A second thread for listening for commands from the command-line interface and complying with them is also created.

After the router startup, the main thread of the router main process enters a main loop, until the command-line interface signals the process to shut down by setting the shutdown event, received by the router process on startup.

During the program startup, two pipelines are created for every interface instance created, in order to connect them with the router main loop. The pipelines passed to the socket processes, distinct from the pipelines connected to the interfaces, are continually accessed by the router main loop, and any OSPF packets inside are forwarded to the correspondent interface instance for processing, using one of the connecting pipelines. The other connecting pipeline is used by the interfaces to send to the router main loop LSAs to be potentially flooded. The router main loop then processes them. If the LSAs are to be flooded, the loop accesses the sending sockets of the appropriate interfaces in order to flood the LSAs, encapsulated in LS Update packets. The router main loop also performs the LSA aging, as the loop has access to all of the LSAs stored by the router process.

The kernel routing table is also set by the router main loop. After the router neighbors are considered stable (i.e. all of them are either in the 2-Way or in the Full state), if any area LSDB was changed since the last update of the kernel routing table, then the router creates a new thread. The thread fetches a copy of the content of the area LSDBs, then obtains the shortest path trees of all areas the router is connected to. Obtaining a copy of the LSDBs is required to ensure the atomicity of the operation without acquiring the LSDBs locks for the entire duration of the update process.

With the information of the shortest path trees, the thread creates an OSPF routing table. Setting the kernel routing table is a slow operation, therefore a new process is created in order to set the kernel routing table using the information of the new OSPF routing table. The router main loop does not start

a new thread to update the routing table if there is already one operating. Both the new thread and the new process exit after the update is complete.

When the router main process receives the shutdown signal and exits the main loop, the process sets the shutdown events of all of the receiving socket processes, waiting for the processes to join. Then, every route that the program inserted in the kernel routing table is deleted, for the corresponding OSPF version. The method that performs the area shutdown is called for each area the router is connected to. The router main process terminates after waiting for the thread listening to the commands from the command-line interface to terminate as well.

The routing table module contains three classes, all of them being data objects with methods to add, get and delete information stored in the classes. The OSPF routing table is modeled as one class. The entry of an OSPF routing table is modeled as a second class, while the path for an entry of the routing table is modeled as a third class. A routing table entry contains a reachable prefix, and a routing table entry path provides a cost, a next hop address, and an outgoing interface, to reach the prefix stored in the entry. In our implementation, it was decided to store a single path to the same address prefix. The OSPF routing table class contains a single field, a list of routing table entries.

The kernel routing table class provides an interface to create, delete and get routes in the routing table of the kernel of the machine running the program. Each method is protected with a lock to prevent two router processes from accessing the kernel routing table at the same time. A prefix is not added to the kernel routing table if the machine is directly connected to it, as direct routes are always preferred over routes learned with protocols such as OSPF.

5.3.6 Area package

No loops are run inside the area layer classes, which act as data objects to store information. In particular, the area class is used to start all underlying interfaces during the router startup, and provide the initial content of the respective LSDB. During the program shutdown, the area class triggers the shutdown of all of its interfaces, only returning after all of the interfaces are down.

The LSDB class is a data object, providing methods for adding, getting and deleting stored LSAs. It stores Router-LSAs and Network-LSAs for both OSPF versions, Network-Summary-LSAs for OSPFv2, and Intra-Area-Prefix-LSAs and Inter-Area-Prefix-LSAs for OSPFv3. LSAs of the same type are stored in the same list. Each LSDB can be accessed by multiple threads at the same time, therefore each LSA list is protected by a different lock, acquired when a method accesses the content of the list. In order to access and manipulate the content of the entire LSDB (for example when clearing the database content), it is necessary to acquire all locks. Using multiple locks instead of a single lock increases the execution speed of the program, as two threads can access different lists of the same LSDB concurrently.

The LSDB class additionally has a method that calculates and returns the shortest path tree of the

corresponding area. The tree is calculated by obtaining the directed graph of the area, and then running the Dijkstra's algorithm on top of it. In our implementation, the shortest path tree is a dictionary, where the keys are the node IDs (either routers or transit networks), and the values are lists storing the shortest path cost to reach the node and the ID of the parent node.

5.3.7 Interface package

The interface class, the only class of the interface package, is the largest class of the code with more than 1.500 lines of code. The class contains the code of all events defined in the OSPF specifications that change the states of the interface and the neighbor state machines. Each event is represented as a different method. The code of the DR election is also contained in the interface class, with each step of the election represented as a different method. In our implementation, OSPFv3 interface instances can store Link-LSAs, supporting the addition, deletion and getting of stored LSAs, the same way it is done in the LSDB class. The interface class also contains a loop, similar to the router main loop.

During the program startup, a thread is started for each interface instance to be created. Each instance receives an atomic pipeline in order to receive OSPF packets from the router class, a shutdown event, and a reference to the area LSDB. The pipeline used to send LSAs to the router class in order to be flooded is created by the interface instance, and read by the router main loop. Each interface instance creates as well an instance of the socket class, an one-shot timer and two interval timers, and threads to run each of the timers. The first interval timer times the sending of the Hello packets, firing every 10 seconds. The second interval timer times the sending of the LS Acknowledgement packets. The one-shot timer fires after 40 seconds, telling the interface instance that it has stayed 40 seconds in the Waiting state.

After the interface startup, the interface main thread enters the interface loop, until the shutdown event is set by the corresponding area class instance.

Most of the tasks described in Section 4.6 are performed by the interface loop. The only exception is the flooding of new LSAs, which is performed by the router main thread even if it uses the sending sockets of the interface instances. The interface loop additionally deletes neighbors if no Hello packets are received from the corresponding neighbor routers during 40 seconds. The time is measured by the inactivity timer created by each neighbor instance, which is accessed by the interface thread. Also, after 40 seconds in the Waiting state, the interface loop calls the event that starts the DR election. The interface loop deletes LSAs in the area LSDB with an LS Age of 3600, and creates new instances of self-originated LSAs when the existent instances reach a LS Age of 1800.

The program implements the delayed acknowledgement feature defined in the OSPF specifications. Instead of sending a LS Acknowledgement packet as soon as a new LSA is received, the interface instance creates a second interval timer. Every time the interval timer fires, the headers of the LSAs

received after the previous firing of the timer are placed in a new LS Acknowledgement packet, which is sent to the network. In our implementation, the acknowledgment timer fires every 2 seconds.

When the interface main thread is signalled to shut down, each neighbor instance is deleted. The shutdown events of the timers are set, and the interface main thread waits for the timer threads to terminate. After every timer thread has terminated, the interface main thread resets the values of the interface instance and terminates as well.

5.3.8 Neighbor package

The neighbor class is the only class of the neighbor package. It is instantiated by the interface class when an Hello packet from a new neighbor router is received. Similar to the area class, no loops are run inside the neighbor class.

During the neighbor startup, an one-shot timer is started in a new thread. The timer fires after 40 seconds to indicate that no Hello packets have been received from the corresponding neighbor router, setting the corresponding timeout event. The interface class sets the reset event of the timer every time an Hello packet has been received from the neighbor router.

The interface class must retransmit any DB Description, LS Request and LS Update packets that do not receive a reply from the neighbor routers in the network during 5 seconds. Each neighbor instance has an one-shot timer for each of the three packet types. The corresponding interface instance starts the timer of the corresponding packet type after the packet has been sent to the network, and stops the timer when a reply is received from the neighbor. If the timer fires, the interface retransmits the packet.

When the shutdown method of the neighbor class is called by the interface class, each timer of the neighbor instance is stopped, and the corresponding threads are terminated.

5.3.9 OSPF extensions

Most of the code written for the OSPF extensions was placed in the router layer. The interface layer suffered minor changes in order to accommodate the OSPF extensions.

A new class was created in the router layer, the extension LSDB class. The class is very similar to the LSDB class of the area layer, storing the overlay LSAs of the router. Overlay LSAs of the same type are stored in the same list, and each LSA list is protected by a thread-safe lock. The class contains methods to get, add and delete LSAs stored in the class instance. The class contains as well a method that returns the shortest path tree of the ABR overlay, in the same format as the area shortest path tree returned by the base LSDB class.

The procedure of updating the kernel routing table has been updated in the router class. The new steps are only executed if the router is an ABR. The router main process counts the number of areas

that the router is connected to, according to the initial configuration. If the router is connected to more than 1 area, then the router is an ABR.

For ABRs, a copy of the overlay LSAs is obtained together with the copies of the area LSDBs. An OSPF routing table is created, and the table receives the paths to the prefixes inside the directly connected areas, which allows the router to update its overlay LSAs. The updates are written to the extension LSDB class and to the copy created by the update procedure, to avoid fetching a new copy of the overlay LSAs.

The router then updates the OSPF routing table using the information of the overlay LSAs, which will always provide the shortest path costs to each prefix in the network. If a prefix already has an entry in the OSPF routing table but with an higher shortest path cost, then the entry is updated. With the OSPF routing table updated, the router updates the kernel routing table. Finally, the router creates Network-Summary-LSAs in OSPFv2, and Inter-Area-Prefix-LSAs in OSPFv3, using the information of the updated OSPF routing table. The new LSAs are stored in the corresponding area LSDBs and flooded to the corresponding area.

The overlay LSAs are represented in the program by new body classes in the LSA package. The new classes implement the body abstract class, and can be packed into byte streams and unpacked from byte streams. The LSA interface class contains methods that call the methods of the overlay LSA body classes, and the class supports the creation and manipulation of overlay LSAs. For that reason, overlay LSAs can be handled by the program like regular LSAs.

During the program startup, each interface instance receives a reference to the extension LSDB class, along with the reference to the area LSDB, in order to allow the interface instances to access and manipulate the overlay LSAs.

5.4 Automated testing

5.4.1 Unit testing

Using the *unittest* Python library, more than 200 automated unit tests have been created to test our implementation. The test package is stored in the source code folder, and contains one test sub-package for each package of the source code. Each test sub-package contains a module with tests for a different module of the corresponding source code package.

An *unittest* test contains asserts, where conditions are evaluated. For example, it is possible to assert that a method return value matches the expected return value, assert that a certain exception is raised during the execution of the test, or assert that a parameter of a class instance is not None.

The simpler tests call a static method of a certain class with different parameters, and assert that the results are what is expected. For more complex tests, it may be required to create class instances with

certain parameters before each test. It is possible to perform actions after each test, such as stopping threads started during the test. For example, most of the tests testing the interface class require an interface instance to be created before each test.

The tests stop at the first failed assert. For example, a test fails if a method returns True when it is expected to return False. If a set of tests is being run in sequence and one test fails, the remaining tests are still run. The *timeout-decorator* library is used to stop and fail tests after a certain time if they do not complete or fail until the timeout.

The test execution times are stated in comments before each test. At the beginning of each test module there is a comment stating the time required to run all of the module tests. The test execution times range from instant to 2-3 minutes, with the entire test suit taking around 10 minutes to complete.

Four commands are relevant to run one or more tests of the test suit. All of the commands must be run in a terminal in the *src* folder of the project.

python3 -m unittest discover Runs the entire test suit.

python3 -m unittest discover test.sub_package_name Runs all of the tests inside a test sub-package, where *sub_package_name* is the name of the sub-package.

Example: *python3 -m unittest discover test.packet*

python3 -m unittest test.sub_package_name.module_name Runs all of the tests inside a test module, where *module_name* is the name of the module.

Example: *python3 -m unittest test.packet.test_header*

python3 -m unittest test.sub_package_name.module_name.test_class.test_name Runs the specified test, where *test_class* is the name of the class inside the test module, and *test_name* is the name of the test.

Example: *python3 -m unittest test.packet.test_header.TestHeader.test_pack_header*

5.4.2 Integration testing

One of the *unittest* tests is an integration test, integrating two router instances in a virtual network. The integration test tests the successful set up of a full adjacency between two neighbor OSPF routers.

During the program startup, a parameter is passed to the router processes stating if the router process is being run inside an integration test. If the parameter is True, then the router processes will not create Python sockets, but instead will put OSPF packets to be sent to the virtual network in exit pipelines, and read entry pipelines to obtain packets received from the virtual network. A virtual hub class, running in a separate thread, was created to connect each router in a virtual network. The class fetches OSPF packets in exit pipelines, and puts them in the entry pipelines of the destination routers.

No other automated integration tests have been created, as the priority was given to manual testing.

6

Evaluation

Contents

6.1 Test environments

6.1.1 Software

In order to test the implementation supporting the current MSc Dissertation, a total of three virtual networks have been created. GNS3 [21] was used to create the virtual networks, since it supports Cisco [22] virtual routers.

It is an objective of the current dissertation to perform interoperability tests on our implementation, using Cisco routers. Two versions of the Cisco IOS were used in the virtual networks: the version 12.2(15)ZJ (the Cisco 3725 router was used), and the version 15.7(3)M.

In order to virtualize the machines running our implementation, Docker containers [23] were used. Not only GNS3 supports Docker containers, but also containers are much faster to start and stop than virtual machines, and use less memory. While VMware [24] was used during the initial months of the development, it became impractical once the number of virtual machines in the network started to grow, leading to the virtual machines being replaced by Docker containers in the networks.

6.1.2 GNS3 networks

The test networks are divided in two groups. The first group contains two single-area OSPF networks, and the second group contains a multi-area OSPF network with a topology not possible with the current OSPF specifications. The networks inside the first group have the same topology and addresses. The only difference between the networks lies in the routers: in the first network a router can be represented by a Cisco router, while in the second network the same router is replaced by a Docker container running our implementation.

The appendices contain the diagrams of the three GNS3 test networks. The symbols used can be seen in Figure 6.1. Aside from the Cisco routers and the Docker containers running our implementation, GNS3 VPCSs and network switches are also used.

Figure 6.1: Icons used in the network diagrams of the appendices. From left to right: Cisco router, Docker container, GNS3 VPCS, and network switch

All of the networks are configured for both OSPFv2 and OSPFv3. The IP addresses for the networks always have the form 222.222.X.Y/24, in OSPFv2, and 2001:db8:cafe:X::Y/64, in OSPFv3, where

X identifies the subnet inside the network and Y identifies the interface inside the subnet. For every network interface, the values for X and Y are the same for both OSPF versions. For example, if an interface has the IPv4 address 222.222.1.3/24, then the interface will also have the IPv6 address 2001:db8:cafe:1::3/64.

The first group of networks contains the networks 1-1 and 1-2. The diagram of the network 1-1 can be seen in the Appendix A, while the diagram of the network 1-2 can be seen in the Appendix B. Both networks are single-area, and have 6 routers and 6 network prefixes. Four of the routers, R1, R4, R5, and R6, are connected to the same subnet. R1 connects the subnet with the rest of the network, and forms a loop with R2 and R3. A VPCS is connected to R1, and another VPCS is connected to R2. The network 1-1 contains one Docker container, the R4, while the remainder of the routers are Cisco routers. The network 1-2 replaces every Cisco router with a Docker container running our implementation. The networks use the Cisco IOS release 12.2(15)ZJ.

The second group of networks contains a single network, the network 2-1. The diagram of the network 2-1 can be seen in the Appendix C. The network contains 6 routers and 5 network prefixes, and is divided in 4 areas: the area 1 at the top, the area 2 in the center on the left, the area 3 in the center on the right, and the area 4 at the bottom. All of the network routers, except for R6, are ABRs. R1, R2 and R3 are connected to the area 1 through a switch, which also connects R6 to the network. R1 is also connected to the area 2, while R2 and R3 are also connected to the area 3. The area 2 contains a single link, connecting R1 to R4. The area 3 contains 2 links, connecting R5 to R2 and R3, respectively. R4 and R5 are connected through the area 4, using two switches. R6 is a Cisco router, while the remaining routers are Docker containers. The network uses the Cisco IOS release 15.7(3)M.

6.1.3 Installing and running the program

Our implementation has been developed in Ubuntu 18.04. In order to set up the test networks and run the program, it is necessary to install GNS3 and Docker Engine. It is also required to install the Python libraries *netifaces* and *timeout-decorator*, described in Section 5.2.

After GNS3 and Docker Engine are installed, it is necessary to obtain the *Dockerfile* file, located in the root folder of the project repository, and place the file in an empty directory of the host machine. With a terminal, it is necessary to go to the directory containing the *Dockerfile*, and then run the command "*docker build -t ospf .*" (including the final "."). The command creates a Docker image called *ospf*, from which Docker containers can be created by GNS3, and takes some minutes to complete. The Docker image only needs to be created once, before starting the networks and running the program for the first time. It is not necessary to import the Docker image or the Cisco IOS releases into GNS3.

The Cisco IOS release 15.7(3)M can use hardware acceleration, but by default it requires GNS3 to be run as root, which is not recommended. A solution is to disable hardware acceleration for the Cisco

IOS release. In the GNS3 main screen, clicking in *Edit*, then *Preferences*, then *QEMU*, and disabling all of the options referring to hardware acceleration, will disable hardware acceleration for the Cisco IOS release and allow the respective routers to be started without running GNS3 as root, at the cost of adding a few seconds to the routers startup time.

The networks can now be opened in GNS3. The directories containing the test networks of our implementation are stored in the *gns3* folder of the root directory of the project repository, with each network being stored in a different directory.

The next step is to place the implementation source code inside each Docker container. The operation can be performed at any moment, even with the Docker containers operating. The source code is stored in the *src* folder of the root directory of the project repository. Inside the directory of a GNS3 network, by entering the *project-files* directory and then the *docker* directory, one or more folders with hexadecimal IDs will appear.

Each folder belongs to a different Docker container of the network. In order to know which folder corresponds to a specific container, it is necessary to go to the GNS3 network. By clicking with the right button on the desired Docker container and then clicking on *Show node information*, the field *server ID* can be seen, with the ID matching one of the folders of the *docker* directory. Inside each container folder, there will be an *ospf* directory, which is the working directory of the container. It is possible that the user does not have permission to modify the content of the folder. The solution is to change the folder permissions using a terminal. The *src* folder can then be placed in the working directory, and accessed from the corresponding Docker container. It is not necessary to restart the Docker container in order to access the new folder.

The OSPF configurable parameters are stored in the *conf.py* file of the configuration package of the source code, at the beginning of the file. Figure 6.2 shows the configurable parameters. Additionally, two environment variables, NETWORK and ROUTER, are directly configured in GNS3 in order to be accessed by the Docker container. They tell each Docker container about the network group where they are, and their ID inside the network, respectively. For example, R3 of the network 1-2 has the NETWORK variable set to 1, and the ROUTER variable set to 3. The variables allow the Docker containers to fetch the right initial configuration.

The first parameter is the Router ID, stored in the ROUTER_IDS variable, containing a list with 6 elements. If the Docker container is the R1 of the network, it will fetch the first element of the list (currently 1.1.1.1), while if the container is the R6 of the network, it will fetch the last element of the list (currently 6.6.6.6).

The following set of parameters contains the physical IDs of the interfaces to be considered by OSPF. The parameters must match the physical interface IDs of the container. Each variable stores the interface IDs for one router. For example, INTERFACES_R1 stores the interface IDs for R1. The variable stores a

```
#  OSPF configurable parameters - Default

ROUTER_ID = '4.4.4.4'
ROUTER_PRIORITY = 1
INTERFACE_COSTS = [10]
#  1st element in interface names tuple must match 1st element in interface areas tuple, and so on
#  Ex: Interface "ens32" belongs to area '0.0.0.0'
INTERFACE_NAMES = ["eth0"]  # Must match interface names in the machine
INTERFACE_AREAS = ['0.0.0.0']
KERNEL_UPDATE_INTERVAL = 0  # Implementation-specific - Minimum time between updates of kernel routing table

#  Only applicable if program is running inside provided GNS3 networks - Replaces default parameters

ROUTER_IDS = ['1.1.1.1', '2.2.2.2', '3.3.3.3', '4.4.4.4', '5.5.5.5', '6.6.6.6']
#  First sublist for first GNS3 network group, second sublist for second GNS3 network group
INTERFACES_R1 = [['eth0', 'eth1', 'eth2', 'eth3'], ['eth0', 'eth1']]
INTERFACES_R2 = [['eth0', 'eth1', 'eth2'], ['eth0', 'eth1']]
INTERFACES_R3 = [['eth0', 'eth1'], ['eth0', 'eth1']]
INTERFACES_R4 = [['eth0'], ['eth0', 'eth1']]
INTERFACES_R5 = [['eth0'], ['eth0', 'eth1', 'eth2']]
INTERFACES_R6 = [['eth0'], []]
AREAS_R1 = [['0.0.0.0', '0.0.0.0', '0.0.0.0', '0.0.0.0'], ['0.0.0.1', '0.0.0.2']]
AREAS_R2 = [['0.0.0.0', '0.0.0.0', '0.0.0.0'], ['0.0.0.1', '0.0.0.3']]
AREAS_R3 = [['0.0.0.0', '0.0.0.0'], ['0.0.0.1', '0.0.0.3']]
AREAS_R4 = [['0.0.0.0'], ['0.0.0.2', '0.0.0.4']]
AREAS_R5 = [['0.0.0.0'], ['0.0.0.3', '0.0.0.3', '0.0.0.4']]
AREAS_R6 = [['0.0.0.0'], []]
INTERFACE_COSTS_R1 = [[10, 10, 10, 10], [10, 10]]
INTERFACE_COSTS_R2 = [[10, 10, 10], [10, 10]]
INTERFACE_COSTS_R3 = [[10, 10], [10, 10]]
INTERFACE_COSTS_R4 = [[10], [10, 10]]
INTERFACE_COSTS_R5 = [[10], [10, 10, 10]]
INTERFACE_COSTS_R6 = [[10], []]
```

Figure 6.2: OSPF configurable parameters in the *conf.py* file of the source code

list with two lists inside, one for each network group. Each internal list contains the IDs of the interfaces to be considered by OSPF in a certain network group. For example, in the first network group, R1 has four interfaces: *eth0*, *eth1*, *eth2*, and *eth3*, while R6 has one interface, *eth0*. In the second network group, R1 has two interfaces, *eth0* and *eth1*, while R6 does not have any interfaces since it is a Cisco router in the network 2-1. The physical interface IDs are included in the diagrams of the appendices.

The following set of parameters contains the area ID of each router interface, and follows the same structure as the interface IDs set of parameters. Each area ID corresponds to an interface ID. For example, for R2, in the network group 1 all of the three router interfaces belong to the backbone area, while in the network group 2, the interface *eth0* belongs to the area 0.0.0.1 and the interface *eth1* belongs to the area 0.0.0.3.

The last set of parameters contains the cost of each router interface, and follows as well the same structure as the interface IDs set of parameters. Each cost corresponds to an interface ID. All of the interface costs are set to 10.

The changes in the configuration become available to the Docker container as soon as the file is saved. However, our implementation does not support the change of OSPF configuration parameters during runtime, and instead the program must be restarted for the changes to be read.

The automated tests described in Section 5.4 are designed to run in R4 of the network 1-1, with R1, R2 and R3 started and R4 and R5 down, after the Cisco routers have converged. The network VPCSs do not need to be started. Before running the tests, it is necessary to start the router program and wait until the prompt for the OSPF version appears, in order to remove any automatically configured IPv6 addresses. At that point, the program can be stopped by pressing Ctrl+C, and the tests can be run.

6.2 Manual testing and results

6.2.1 Single-area OSPF

6.2.1.A Interoperability with Cisco routers

In order to test the correct implementation of the single-area OSPF, two experiments have been performed. The command-line interfaces of the Cisco routers and of our implementation were used to observe the internal state of the routers, while Wireshark [25] was used to capture the packets being exchanged in the network.

The first experiment was performed on the network 1-1, and tested the interoperability of our implementation with Cisco routers. The network 1-1 contains one Docker container, R4, while the rest of the routers of the network are Cisco routers. The routers of the network were started at the same time, running both OSPF versions.

Figure 6.3 shows a Wireshark packet capture during the initial LSDB synchronization process among the routers R1, R4, R5 and R6. The link-local IPv6 address fe80::ecad:90ff:febb:6e13 belongs to R4, while the link-local addresses fe80::c001:18ff:fe34:10, fe80::c004:43ff:febc:0, and fe80::c005:66ff:fe84:0, belong to R1, R5, and R6, respectively. In the image, we can observe the usual exchange of OSPF packets during the initial LSDB synchronization process.

No.	Time	Source	Destination	Protocol	Length	Info
171	142.740274	222.222.1.3	222.222.1.2	OSPF	66	DB Description
172	142.793841	fe80::c004:43ff:feb…	fe80::ecad:90ff:feb…	OSPF	82	DB Description
173	142.807341	fe80::c004:43ff:feb…	ff02::5	OSPF	102	Hello Packet
174	142.816418	222.222.1.1	224.0.0.6	OSPF	98	LS Update
179	142.860334	fe80::c005:66ff:fe8…	fe80::ecad:90ff:feb…	OSPF	82	DB Description
180	142.870354	222.222.1.4	224.0.0.5	OSPF	134	LS Update
181	142.879992	222.222.1.2	222.222.1.4	OSPF	86	DB Description
182	142.902443	fe80::c005:66ff:fe8…	ff02::5	OSPF	98	LS Update
183	142.903488	222.222.1.4	224.0.0.5	OSPF	234	LS Update
184	142.914240	222.222.1.4	224.0.0.5	OSPF	98	LS Update
185	142.927039	222.222.1.4	222.222.1.2	OSPF	206	DB Description
186	142.937318	fe80::c001:18ff:fe3…	ff02::6	OSPF	98	LS Update
187	142.960903	222.222.1.2	222.222.1.3	OSPF	66	DB Description
188	142.991280	fe80::c004:43ff:feb…	ff02::5	OSPF	98	LS Update
189	143.016609	fe80::c001:18ff:fe3…	ff02::6	OSPF	174	LS Update
190	143.035387	fe80::c004:43ff:feb…	ff02::5	OSPF	522	LS Update
191	143.066684	fe80::c004:43ff:feb…	ff02::5	OSPF	174	LS Update
192	143.116217	fe80::ecad:90ff:feb…	fe80::c004:43ff:feb…	OSPF	142	DB Description
193	143.119168	fe80::c004:43ff:feb…	fe80::ecad:90ff:feb…	OSPF	402	DB Description
194	143.119253	fe80::ecad:90ff:feb…	ff02::5	OSPF	102	Hello Packet
195	143.183876	222.222.1.2	224.0.0.5	OSPF	90	Hello Packet
196	143.199704	222.222.1.4	224.0.0.5	OSPF	98	LS Update
197	143.205505	fe80::ecad:90ff:feb…	fe80::c005:66ff:fe8…	OSPF	82	DB Description
198	143.217019	222.222.1.4	224.0.0.5	OSPF	98	LS Update
199	143.227776	222.222.1.2	222.222.1.3	OSPF	86	DB Description
200	143.237555	222.222.1.3	222.222.1.2	OSPF	226	DB Description
201	143.300719	fe80::c004:43ff:feb…	ff02::5	OSPF	226	LS Update
202	143.310254	222.222.1.3	224.0.0.5	OSPF	98	LS Update
203	143.320869	222.222.1.1	224.0.0.6	OSPF	98	LS Update
205	143.405538	222.222.1.2	224.0.0.6	OSPF	134	LS Update
206	143.414404	222.222.1.3	222.222.1.2	OSPF	78	LS Acknowledge
207	143.417225	222.222.1.4	222.222.1.2	OSPF	78	LS Acknowledge
208	143.497717	fe80::c005:66ff:fe8…	ff02::5	OSPF	106	LS Update
209	143.532360	fe80::c001:18ff:fe3…	ff02::6	OSPF	106	LS Update
210	143.550645	222.222.1.2	224.0.0.6	OSPF	122	LS Update
211	143.552762	222.222.1.4	222.222.1.2	OSPF	78	LS Acknowledge
212	143.560191	222.222.1.3	222.222.1.2	OSPF	78	LS Acknowledge
213	143.570033	fe80::ecad:90ff:feb…	fe80::c005:66ff:fe8…	OSPF	142	DB Description
215	143.583422	fe80::c005:66ff:fe8…	fe80::ecad:90ff:feb…	OSPF	402	DB Description
216	143.636909	fe80::ecad:90ff:feb…	ff02::6	OSPF	98	LS Update
217	143.643064	fe80::c005:66ff:fe8…	fe80::ecad:90ff:feb…	OSPF	90	LS Acknowledge
218	143.661344	fe80::c004:43ff:feb…	fe80::ecad:90ff:feb…	OSPF	90	LS Acknowledge
219	143.696646	fe80::ecad:90ff:feb…	fe80::c004:43ff:feb…	OSPF	90	LS Acknowledge

Figure 6.3: Wireshark capture of some of the packets exchanged in the subnet 222.222.1.0/24 / 2001:db8:cafe:1::/64 during the initial LSDB synchronization process, during the first experiment

After the network converged, it was observed that R4 successfully recognized R6 and R5 as the network DR and the network BDR, respectively, establishing with each of the routers a full adjacency.

Since R1 was neither a DR nor a BDR, R4 did not try to establish a full adjacency with R1, and the neighbor relationship with R1 remained in the 2-Way state.

It was also observed that the LSDB of R4 contained all of the area-scope LSAs of the network for both OSPF versions, and the link-scope LSAs of the link for OSPFv3. Apart from the prefix of the subnet that R4 was connected to, which was already in the respective kernel routing table, our implementation was able to add all of the prefixes of the network to the kernel routing table of R4, along with the indication that the next hop address was the address of the interface of R1 connected to the subnet. R4 was also able to update its LSAs and send them to the subnet DR in order to be flooded to the network. R4 could ping routers outside the subnet, and it was possible to ping R4 from outside the subnet.

Figure 6.4 shows the console of R4 after the network converged, with the content of the respective kernel routing table showing all of the network prefixes. The three neighbor routers of R4 are also shown in the figure, along with their information. Figure 6.5 shows the LSDB of R1, where the Router-LSA of R4 can be seen. Figure 6.6 shows R4 listed as an attached router in the Network-LSA of the link.

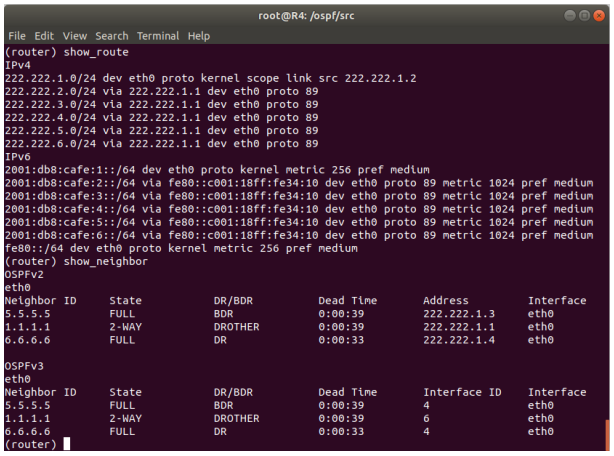


Figure 6.4: Command-line interface of R4 showing the prefixes stored in the respective kernel routing table and the information about the router OSPF neighbors, during the first experiment

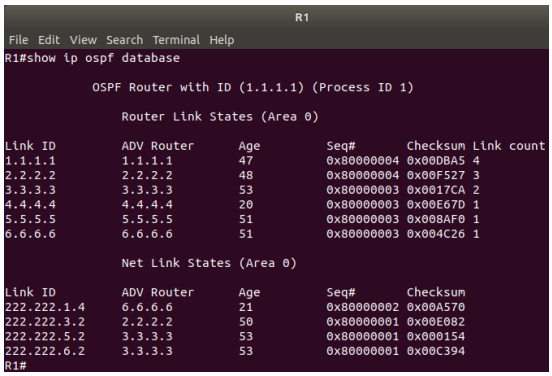


Figure 6.5: Router-LSA of R4 in the LSDB of R1

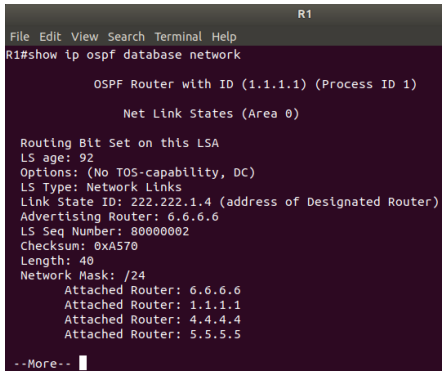


Figure 6.6: R4 in the Network-LSA of the subnet 222.222.1.0/24 / 2001:db8:cafe:1::/64

A variant of the experiment was also performed, where all Cisco routers of the network were started at the same time and allowed to converge. Then, R4 was started and allowed to converge as well. The experiment was performed three times, in order to measure the average convergence times of the routers of the subnet 222.222.1.0/24 / 2001:db8:cafe:1::/64. For each OSPF version, the convergence time of each Cisco router was considered to be the time interval between the sending of the first OSPFv2 Hello packet by the router (the first OSPFv3 Hello packet was always sent 10 seconds later) and the sending of the last LS Acknowledgment packet of the respective OSPF version by the router.

It was observed that the Cisco routers took on average 52 seconds to converge for OSPFv2 and 59 seconds for OSPFv3, while our implementation took 23 seconds to converge for OSPFv2 and 20 seconds to converge for OSPFv3. The convergence times of the Cisco routers include the 40 seconds that each OSPF router remains in the Waiting state after the respective startup.

6.2.1.B Interoperability with other Docker containers

The second experiment was performed on the network 1-2, and tested the interoperability of different Docker containers running our implementation and connected to the same network. The network 1-2 replaces all of the Cisco routers of the network 1-1 with Docker containers. For the experiment, R1, R2 and R3 were started at the same time. After the routers converged, R4 was started. As in the first experiment, the routers ran both OSPF versions.

Figure 6.7 shows a Wireshark packet capture during the initial LSDB synchronization process between the routers R1 and R4. The link-local IPv6 address fe80::7412:3aff:fe70:5f8a belongs to R4, while the link-local IPv6 address fe80::887d:a4ff:fe3f:9df3 belongs to R1.

No.	Time	Source	Destination	Protocol	Length	Info
42	145.979855	fe80::7412:3aff:fe7…	ff02::5	OSPF	94	Hello Packet
43	146.910043	222.222.1.1	222.222.1.2	OSPF	66	DB Description
44	147.054641	fe80::887d:a4ff:fe3…	fe80::7412:3aff:fe7…	OSPF	82	DB Description
45	149.940161	222.222.1.1	224.0.0.5	OSPF	82	Hello Packet
46	150.005489	fe80::887d:a4ff:fe3…	ff02::5	OSPF	94	Hello Packet
47	150.107415	222.222.1.2	222.222.1.1	OSPF	66	DB Description
48	150.214871	fe80::7412:3aff:fe7…	fe80::887d:a4ff:fe3…	OSPF	82	DB Description
49	150.616939	222.222.1.1	222.222.1.2	OSPF	186	DB Description
50	150.670477	222.222.1.2	222.222.1.1	OSPF	86	DB Description
51	151.830339	fe80::887d:a4ff:fe3…	fe80::7412:3aff:fe7…	OSPF	342	DB Description
52	151.955271	fe80::7412:3aff:fe7…	fe80::887d:a4ff:fe3…	OSPF	142	DB Description
57	152.804907	fe80::887d:a4ff:fe3…	fe80::7412:3aff:fe7…	OSPF	82	DB Description
58	152.857014	fe80::7412:3aff:fe7…	fe80::887d:a4ff:fe3…	OSPF	82	DB Description
59	153.107484	222.222.1.1	222.222.1.2	OSPF	66	DB Description
60	153.326181	222.222.1.2	222.222.1.1	OSPF	66	DB Description
61	153.453770	222.222.1.1	222.222.1.2	OSPF	66	DB Description
62	153.507856	222.222.1.1	222.222.1.2	OSPF	70	LS Request
63	153.552574	fe80::887d:a4ff:fe3…	fe80::7412:3aff:fe7…	OSPF	82	DB Description
64	153.701713	222.222.1.2	222.222.1.1	OSPF	130	LS Request
65	153.702774	fe80::887d:a4ff:fe3…	fe80::7412:3aff:fe7…	OSPF	106	LS Request
66	153.778883	fe80::7412:3aff:fe7…	fe80::887d:a4ff:fe3…	OSPF	226	LS Request
67	153.874254	222.222.1.2	222.222.1.1	OSPF	98	LS Update
68	153.991335	fe80::7412:3aff:fe7…	fe80::887d:a4ff:fe3…	OSPF	198	LS Update
73	155.433099	222.222.1.1	222.222.1.2	OSPF	338	LS Update
74	156.131283	fe80::7412:3aff:fe7…	ff02::5	OSPF	94	Hello Packet
75	156.482852	fe80::887d:a4ff:fe3…	fe80::7412:3aff:fe7…	OSPF	670	LS Update
76	156.548859	222.222.1.2	224.0.0.5	OSPF	98	LS Update
77	156.637197	222.222.1.2	224.0.0.5	OSPF	82	Hello Packet
78	156.716693	222.222.1.2	224.0.0.5	OSPF	178	LS Acknowledge
79	157.155234	fe80::7412:3aff:fe7…	ff02::5	OSPF	130	LS Acknowledge
80	158.699897	222.222.1.1	224.0.0.5	OSPF	78	LS Acknowledge
81	158.878111	fe80::887d:a4ff:fe3…	ff02::5	OSPF	110	LS Acknowledge
82	158.996511	fe80::7412:3aff:fe7…	ff02::5	OSPF	114	LS Update
83	158.999025	fe80::7412:3aff:fe7…	fe80::887d:a4ff:fe3…	OSPF	170	LS Update
84	159.247917	fe80::7412:3aff:fe7…	ff02::5	OSPF	270	LS Acknowledge
85	159.415833	222.222.1.1	224.0.0.5	OSPF	134	LS Update
86	159.605603	fe80::7412:3aff:fe7…	ff02::5	OSPF	118	LS Update
87	160.914739	fe80::887d:a4ff:fe3…	ff02::5	OSPF	90	LS Acknowledge
88	161.566607	222.222.1.1	224.0.0.5	OSPF	82	Hello Packet
89	162.884935	fe80::887d:a4ff:fe3…	ff02::5	OSPF	146	LS Update
90	162.909167	fe80::887d:a4ff:fe3…	ff02::5	OSPF	94	Hello Packet
91	162.978422	222.222.1.1	224.0.0.5	OSPF	78	LS Acknowledge
92	163.367672	fe80::7412:3aff:fe7…	ff02::5	OSPF	90	LS Acknowledge

Figure 6.7: Wireshark capture of some of the packets exchanged in the subnet 222.222.1.0/24 / 2001:db8:cafe:1::/64 during the initial LSDB synchronization process, during the second experiment

After R1, R2 and R3 converged, it was observed that, in each subnet, the connected routers successfully determined the correct DR and the correct BDR. Each of the three routers generated the correct LSAs with the correct content, which were flooded to the other routers. The routers were able to successfully stop the flooding procedure after the LSAs were flooded, allowing the network to converge. Additionally, each of the three routers was able to add all of the prefixes of the network to the respective kernel routing table. When two shortest paths with the same cost to the same destination were available, the routers selected one of the available paths.

After R4 was started and the network converged again, it was observed that R4 successfully recognized R1 as the subnet DR and declared itself as BDR, establishing a full adjacency with R1.

It was also observed that the LSDB of R4 contained all of the area-scope LSAs of the network for both OSPF versions, and the link-scope LSAs of the link for OSPFv3. Similar to the first experiment, our implementation was able to add all of the prefixes of the network to the kernel routing table of R4, along with the indication that the next hop address was the address of the interface of R1 connected to the subnet. R4 was also able to successfully update its LSAs and send them to R1 in order to be flooded to R2 and R3. R4 could ping routers outside the subnet it was connected to, and it was possible to ping R4 from outside the subnet.

Figure 6.8 shows the console of R4 after the network has converged with the four routers connected to it, with the content of the respective kernel routing table showing all of the prefixes in the network. The figure also shows the information of the neighbor relationship with R1. Figure 6.9 shows the content of the LSDB of R4 in the same situation, where all of the current LSAs of the network can be seen.

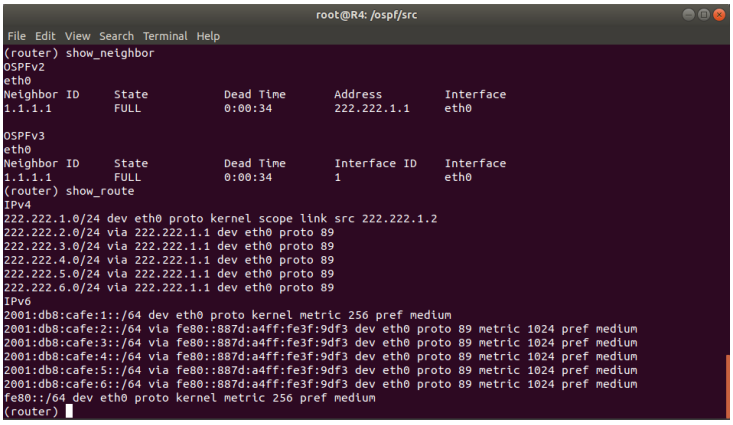


Figure 6.8: Command-line interface of R4 showing the prefixes stored in the respective kernel routing table and the information about the neighbor relationship with R1, during the second experiment

The experiment was performed a total of three times in order to measure the network convergence times, as in the first experiment. It was observed that R1, R2 and R3 took on average 74 seconds to converge for OSPFv2 and 115 seconds to converge for OSPFv3. When R4 was started, the router took 28 seconds to converge for OSPFv2 and 45 seconds to converge for OSPFv3. The convergence times

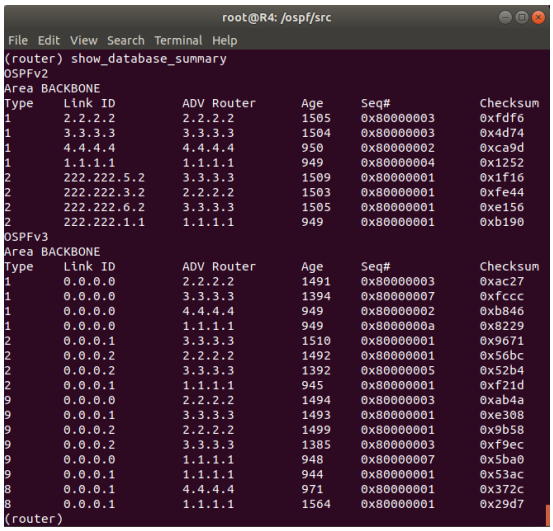


Figure 6.9: All of the current LSAs of the network listed in the LSDB of R4, during the second experiment

for R1, R2 and R3 include the 40 seconds that each OSPF router waits in the Waiting state after startup.

All of the measured convergence times in the current experiment are longer than the convergence times measured in the first experiment in the network 1-1, with increments ranging from 22% to 125%. The time increments were higher in OSPFv3, with the network formed by R1, R2 and R3 taking 95% more time to converge than the Cisco routers R1, R5 and R6 of the network 1-1, while R4 took 125% more time to converge in the network 1-2 than in the network 1-1. In OSPFv2, the time increments were 42% and 22%, respectively.

It can be considered that the use of Python in the implementation contributed to the increase of the convergence times. However, it can also be considered that our implementation of the base OSPF protocol can be improved and optimized, specially the implementation of the base OSPFv3 protocol.

6.2.2 OSPF extensions

6.2.2.A OSPFv3 bugs in Cisco routers

During the implementation of the OSPF extensions, it was noticed that the Cisco 3725 routers did not correctly flood stored OSPFv3 LSAs with unknown LS Type during the initial LSDB synchronization process. The LSAs were flooded with the LS Function Code set to 0, preventing the identification of the respective LS Type by the receiving routers. An attempt was made to overcome the problem by replacing the Cisco routers of the network 2-1 with other Cisco routers running the Cisco IOS release 15.7(3)M, more recent than the Cisco IOS release executed in the Cisco 3725 routers.

While the problem disappeared, a new apparent bug was found: the Cisco routers being used seemed to treat all OSPFv3 LSAs with unknown LS Type as if all of them had the same LS Type. If

a Cisco router received a LSA with unknown LS Type, and later it received another LSA with a different unknown LS Type and a lower sequence number, then the Cisco router would discard the second incoming LSA and send the first LSA back to the sending router, as if it was a more recent instance of the second LSA. To the best of our knowledge, the problem was not caused by a bug in our implementation or by a configuration error.

Considering the apparent bugs in the Cisco routers regarding LSAs with unknown LS Type in OSPFv3, it was decided to perform experiments only with the OSPFv2 extension.

6.2.2.B Multi-area network - Multiple shortest paths

In order to test the correct implementation of the OSPFv2 extension, two experiments have been performed in the network 2-1. The network has a multi-area topology that is not supported by the current OSPF specification, being divided in 4 non-hierarchical areas.

The first experiment tested the correct operation of the OSPFv2 extension, including the generation and flooding of the overlay LSAs and the update of the kernel routing table based on the respective information. A second objective was to test the correct creation and flooding of Network-Summary-LSAs, in order to allow the area internal routers to know the network prefixes outside the area and how to reach them. The network 2-1 has one area internal router, a Cisco router, placed in the area 1.

For the experiment, the five network ABRs were started at the same time. The network was started and allowed to converge a total of three times, in order to measure the network convergence times. Table 6.1 shows the measured convergence times for each router in each execution, in seconds. The time was measured since the router startup until the last update of the respective kernel routing table, therefore including the 40 seconds that each router remains in the Waiting state after startup.

Execution	R1	R2	R3	R4	R5	Average
1	189,2	196,2	193,4	187,1	191,7	191,5
2	190,5	201,2	200,2	187,8	194,6	194,9
3	171,9	175,3	173,1	169,3	165,5	171,0
Average	183,9	190,9	188,9	181,4	183,9	185,8

Table 6.1: Convergence time for each ABR of the network 2-1 during 3 executions, in seconds

On average the routers took 186 seconds to converge, a value 258% higher than the average convergence times of the Cisco routers R1, R5 and R6 in the network 1-1, and 151% higher than the average convergence times of the routers R1, R2 and R3 running our implementation in the network 1-2.

The increase in the number of routers converging at the same time, the additional processing required by the OSPFv2 extension, and the exchange of overlay LSAs along with regular OSPF LSAs, can be considered causes of the increment of the convergence times. However, it can also be considered that further work can be performed on the OSPFv2 extension to improve and optimize it.

Figure 6.10 shows the overlay LSAs generated by the network ABRs. Each router in the network

generates one Prefix-LSA with the different prefixes of the areas that the ABR is connected to, and one ABR-LSA with the different ABRs connected to the same areas, along with the costs to reach the prefixes and the ABRs, respectively. Every interface cost was set to 10 for the experiment.

After the network converged and the convergence times were measured, R6 was started. When the router stabilized, the kernel routing table of the router contained paths to the 5 subnets of the network. In particular, the subnet with the address 222.222.3.0 could be reached through three different shortest paths, each path having a cost of 30 and going through a different ABR of the area 1. The routes known to R6 can be seen at Figure 6.11 along with the paths from R6 to the IP address 222.222.3.4 obtained with traceroute, while the different Network-Summary-LSAs flooded to the subnet of R6 and the overlay LSAs created in the network can be seen in the LSDB of R6 in Figure 6.12.

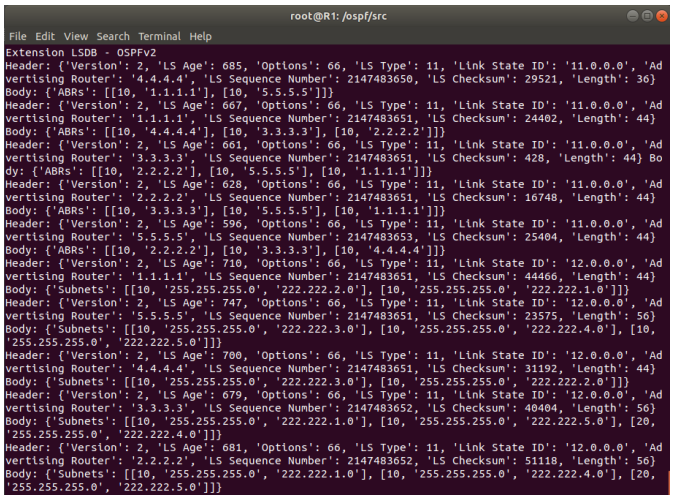


Figure 6.10: Overlay LSAs of the network 2-1 as seen in the command-line interface of R1

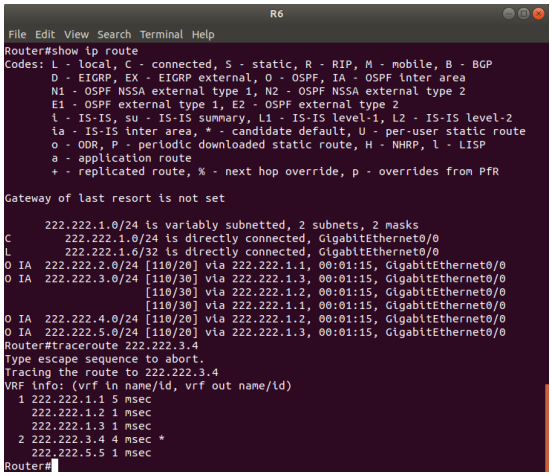


Figure 6.11: Routes of R6 after convergence, during the first experiment

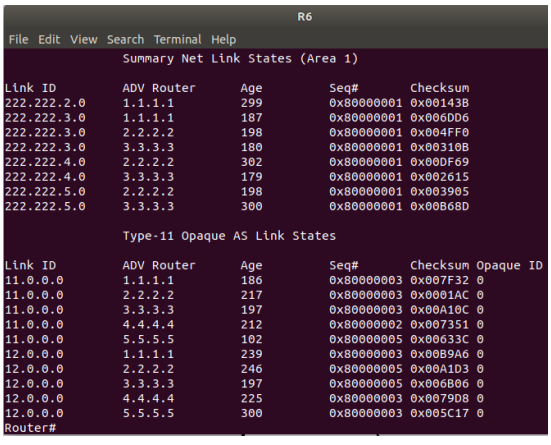


Figure 6.12: Network-Summary-LSAs and overlay LSAs in the LSDB of R6

Figure 6.13 shows a Wireshark packet capture during the initial LSDB synchronization process between the routers R1, R2, R3 and R6. The IPv4 addresses 222.222.1.1, 222.222.1.2, 222.222.1.3 and 222.222.1.6 belong to R1, R2, R3 and R6, respectively.

No.	Time	Source	Destination	Protocol	Length	Info
60	70.983136	222.222.1.2	224.0.0.5	OSPF	90	Hello Packet
61	70.996012	222.222.1.6	222.222.1.3	OSPF	78	DB Description
62	71.204884	222.222.1.3	222.222.1.6	OSPF	66	DB Description
63	71.683136	222.222.1.3	222.222.1.6	OSPF	526	DB Description
64	71.892436	222.222.1.6	222.222.1.3	OSPF	78	DB Description
65	72.397377	222.222.1.3	222.222.1.6	OSPF	66	DB Description
66	72.403709	222.222.1.3	224.0.0.5	OSPF	98	LS Update
67	72.602330	222.222.1.6	222.222.1.3	OSPF	334	LS Request
68	73.052528	222.222.1.2	224.0.0.5	OSPF	98	LS Update
69	73.097805	222.222.1.3	224.0.0.5	OSPF	90	Hello Packet
70	73.386943	222.222.1.1	224.0.0.6	OSPF	98	LS Update
71	73.861212	222.222.1.1	224.0.0.6	OSPF	98	LS Update
72	74.040301	222.222.1.3	224.0.0.5	OSPF	102	LS Update
73	74.724916	222.222.1.3	222.222.1.6	OSPF	938	LS Update
74	74.820408	222.222.1.2	224.0.0.5	OSPF	98	LS Update
75	74.831510	222.222.1.6	222.222.1.3	OSPF	478	LS Acknowledge
76	74.831719	222.222.1.6	222.222.1.3	OSPF	98	LS Update
77	75.265541	222.222.1.1	224.0.0.6	OSPF	102	LS Update
78	75.495895	222.222.1.6	224.0.0.6	OSPF	538	LS Acknowledge
79	75.890309	222.222.1.3	224.0.0.5	OSPF	98	LS Update
83	76.941120	222.222.1.2	224.0.0.5	OSPF	102	LS Update
84	77.321805	222.222.1.6	224.0.0.6	OSPF	98	LS Update
85	80.153530	222.222.1.1	224.0.0.5	OSPF	90	Hello Packet
86	80.752612	222.222.1.2	224.0.0.5	OSPF	78	LS Acknowledge
87	81.094886	222.222.1.2	224.0.0.5	OSPF	90	Hello Packet
88	81.378819	222.222.1.3	222.222.1.1	OSPF	174	LS Update
90	81.588178	222.222.1.6	224.0.0.5	OSPF	102	Hello Packet
91	81.671097	222.222.1.1	222.222.1.3	OSPF	78	LS Acknowledge
92	81.671762	222.222.1.1	222.222.1.3	OSPF	78	LS Acknowledge
93	81.712070	222.222.1.1	222.222.1.3	OSPF	78	LS Acknowledge
94	81.952571	222.222.1.3	222.222.1.2	OSPF	98	LS Update
95	82.264085	222.222.1.2	222.222.1.3	OSPF	138	LS Update
96	82.502067	222.222.1.3	222.222.1.6	OSPF	98	LS Update
97	82.637852	222.222.1.6	222.222.1.3	OSPF	98	LS Update
98	82.639228	222.222.1.6	222.222.1.3	OSPF	98	LS Update
99	83.327930	222.222.1.3	224.0.0.5	OSPF	90	Hello Packet
100	84.186820	222.222.1.3	222.222.1.2	OSPF	78	LS Acknowledge
101	84.490489	222.222.1.3	222.222.1.2	OSPF	78	LS Acknowledge
102	84.642849	222.222.1.3	222.222.1.6	OSPF	78	LS Acknowledge
103	84.774507	222.222.1.3	224.0.0.5	OSPF	98	LS Update
104	84.863034	222.222.1.6	222.222.1.3	OSPF	78	LS Acknowledge
105	85.127563	222.222.1.2	222.222.1.3	OSPF	78	LS Acknowledge
106	85.174801	222.222.1.1	224.0.0.6	OSPF	98	LS Update
107	85.936669	222.222.1.2	222.222.1.3	OSPF	78	LS Acknowledge
108	85.988791	222.222.1.2	224.0.0.5	OSPF	98	LS Update
115	87.119679	222.222.1.3	222.222.1.2	OSPF	78	LS Acknowledge
118	88.990955	222.222.1.3	222.222.1.1	OSPF	98	LS Update
119	89.140913	222.222.1.1	222.222.1.3	OSPF	78	LS Acknowledge
120	89.934473	222.222.1.1	224.0.0.5	OSPF	90	Hello Packet

Figure 6.13: Wireshark capture of some of the packets exchanged between R1, R2, R3 and R6 during the initial LSDB synchronization process of R6, during the first experiment

```
R6
File  Edit  View  Search  Terminal  Help
Router#show ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       a - application route
       + - replicated route, % - next hop override, p - overrides from PfR

Gateway of last resort is not set

      222.222.1.0/24 is variably subnetted, 2 subnets, 2 masks
C        222.222.1.0/24 is directly connected, GigabitEthernet0/0
L        222.222.1.6/32 is directly connected, GigabitEthernet0/0
O IA  222.222.2.0/24 [110/11] via 222.222.1.1, 00:02:52, GigabitEthernet0/0
O IA  222.222.3.0/24 [110/21] via 222.222.1.1, 00:02:52, GigabitEthernet0/0
O IA  222.222.4.0/24 [110/20] via 222.222.1.2, 00:02:52, GigabitEthernet0/0
O IA  222.222.5.0/24 [110/20] via 222.222.1.3, 00:02:52, GigabitEthernet0/0
Router#traceroute 222.222.3.4
Type escape sequence to abort.
Tracing the route to 222.222.3.4
VRF info: (vrf in name/id, vrf out name/id)
  1 222.222.1.1 1 msec 1 msec 1 msec
  2 222.222.3.4 1 msec 0 msec 0 msec
Router#
```

Figure 6.14: Routes of R6 after convergence, during the second experiment

6.2.2.C Multi-area network - One shortest path

The second experiment was similar to the first experiment, with the difference of changing the costs of the interfaces of R1 from 10 to 1. The remaining interface costs were not changed. The goal of

the experiment was to test the correct operation of the OSPFv2 extension in a network with different interface costs, and the correct transmission of the respective routing information to the area internal routers.

Similar to the first experiment, first the network ABRs were started and allowed to converge, and then R6 was started. Figure 6.14 shows the routes of the kernel routing table of R6 after converging, during the second experiment, along with the path from R6 to the IP address 222.222.3.4 obtained with traceroute. With R1 having lower interface costs than R2 and R3, R6 inserted in the respective kernel routing table a single shortest path to the subnet with the prefix 222.222.3.0, with a cost of 21 and going through R1. The cost of the shortest path to the prefix 222.222.2.0, which already passed through R1 in the previous experiment, was reduced from 20 to 11. The shortest paths to the other prefixes were not changed.

7

Conclusion

Contents

7.1 Conclusion

LSR protocols are superior to DVR protocols, since they allow routers to have the complete network view, preventing the problem known as count-to-infinity that affects DVR protocols such as RIP. However, the current versions of OSPF use a distance vector approach to inter-area routing, bringing two important restrictions to OSPF multi-area networks: the networks must have a two-level hierarchical structure with a single area in the top level, and optimal routing between areas is not guaranteed.

For the current MSc Dissertation, two extensions to the base OSPF protocol were implemented, one for each version, which apply a link state approach to inter-area routing, providing the complete network view to all routers in the network even across areas. The extensions allow to create an ABR overlay, a logical network formed by the network ABRs over the OSPF network, where all ABRs know the shortest paths to reach each other, and advertise as well the shortest path costs to reach the address prefixes located in their areas. New types of LSAs were created in order to carry the information between ABRs. As the current OSPF specification already supports the creation of new types of LSAs, the extensions are transparent to area internal routers.

An implementation of the current OSPF specifications was first created, in order to be later extended with the OSPF extensions. The software architecture contains four different layers: from the top to the bottom, the router, the area, the interface, and the neighbor, each layer corresponding to a data structure or a set of data structures defined in the OSPF RFCs. The router layer contains the top-level OSPF data structures, and is responsible for listening to OSPF packets in the network, coordinating the flooding of LSAs, and setting the kernel routing table. The area layer stores the area-scope LSAs. The interface layer manages the state machines of the interface and of the neighbor, elects the DR and the BDR of the subnet, and processes the incoming packets. The neighbor layer contains the neighbor data structure. The OSPF extensions required modifications in the router and in the interface layers, mainly a reformulation of the process of updating the kernel routing table of the machine running the program.

In order to test the implementation, three GNS3 networks were created, interconnecting Cisco routers and Docker containers running the implementation. Experiments were conducted in single-area OSPF networks, and in a multi-area network with a topology not possible with the current OSPF specifications. The single-area experiments were performed for both OSPF versions, and proved that the implementation can work correctly in single-area OSPF networks, being capable of interoperating with Cisco routers, and successfully forming networks only with machines running the program. Bugs were found in the Cisco routers that prevented the correct manipulation of LSAs with unknown LS Type in OSPFv3. For that reason, the experiments in the multi-area network were conducted only for the OSPFv2 extension. The experiments proved that the extension is operating, being capable of generating and flooding overlay LSAs across a multi-area network, and flooding the routing information to the area internal routers.

7.2 Future work

As a future work, our implementation of the base OSPF specifications should be improved and optimized, with the objective of fixing bugs in the implementation and decreasing the convergence times of networks with Docker containers running our implementation.

Additionally, experiments should be performed on our implementation of the OSPFv3 extension, using networks with topologies not possible with the current OSPF specifications. The implementation of both OSPF extensions can then be improved and optimized like the base implementation, in order to fix bugs and reduce the respective convergence times.

Another possibility of future work is the addition of support for ASBRs and domain-external prefixes, both in the base OSPF implementation and in the implementation of the OSPF extensions.

Finally, the OSPF extensions and the respective implementation can be described in an Internet-Draft, to be submitted to the IETF.

Bibliography

[1] R. Valadas, “OSPF extension for the support of multi-area networks with arbitrary topologies,” 2017. [Online]. Available: http://arxiv.org/abs/1704.08916

[2] D. P. Bertsekas and R. G. Gallager, *Data Networks*, 2nd ed. Prentice Hall, 1992. [Online]. Available: https://web.mit.edu/dimitrib/www/datanets.html

[3] G. Malkin, “RFC 2453 - RIP Version 2,” 1998. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2453.txt

[4] G. Malkin and R. Minnear, “RFC 2080 - RIPng for IPv6,” 1997. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2080.txt

[5] R. Valadas, *OSPF and IS-IS From Link State Routing Principles to Technologies*, 1st ed. CRC Press, 2019. [Online]. Available: https://www.crcpress.com/OSPF-and-IS-IS-From-Link-State-Routing-Principles-to-Technologies/Valadas/p/book/9781138504554

[6] J. T. Moy, “RFC 2328 - OSPF Version 2,” 1998. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2328.txt

[7] R. Coltun, D. Ferguson, J. Moy, and E. A. Lindem, “RFC 5340 - OSPF for IPv6,” 2008. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5340.txt

[8] O. Bonaventure, *Computer Networking : Principles, Protocols and Practice*, 1st ed. Université Catholique de Louvain, 2010. [Online]. Available: https://www.computer-networking.info/

[9] R. Hinden and S. Deering, “RFC 4291 - IP Version 6 Addressing Architecture,” 2006. [Online]. Available: https://tools.ietf.org/html/rfc4291.html

[10] L. Berger, I. Bryskin, A. Zinin, and R. Coltun, “RFC 5250 - The OSPF Opaque LSA Option,” 2008. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5250.txt

[11] M. Gonçalves, “ospf-multiarea-arbitrary-topology,” 2020. [Online]. Available: https://github.com/migueldgoncalves/ospf-multiarea-arbitrary-topology

[12] “Opaque-LSA Option Types,” 2020. [Online]. Available: https://www.iana.org/assignments/ospf-opaque-types/ospf-opaque-types.xhtml

[13] “OSPFv3 parameters,” 2020. [Online]. Available: https://www.iana.org/assignments/ospfv3-parameters/ospfv3-parameters.xhtml

[14] A. P. Pires, “OSPF extension to support multi-area networks with arbitrary topologies,” Instituto Superior Técnico, Tech. Rep., 2018. [Online]. Available: https://fenix.tecnico.ulisboa.pt/departamentos/dei/dissertacao/1972678479054624

[15] J. T. Moy, *OSPF Complete Implementation*, 1st ed. New York: Addison-Wesley, 2001.

[16] “Welcome to Python.org.” [Online]. Available: https://www.python.org/

[17] “Ubuntu.” [Online]. Available: https://ubuntu.com/

[18] A. Ajitsaria, “What Is the Python Global Interpreter Lock (GIL)?” [Online]. Available: https://realpython.com/python-gil/

[19] A. Houghton, “netifaces,” 2019. [Online]. Available: https://pypi.org/project/netifaces/

[20] P. Ng, “timeout-decorator,” 2020. [Online]. Available: https://pypi.org/project/timeout-decorator/

[21] “GNS3 — The software that empowers network professionals.” [Online]. Available: https://www.gns3.com/

[22] “Cisco - Global Home Page.” [Online]. Available: https://www.cisco.com/

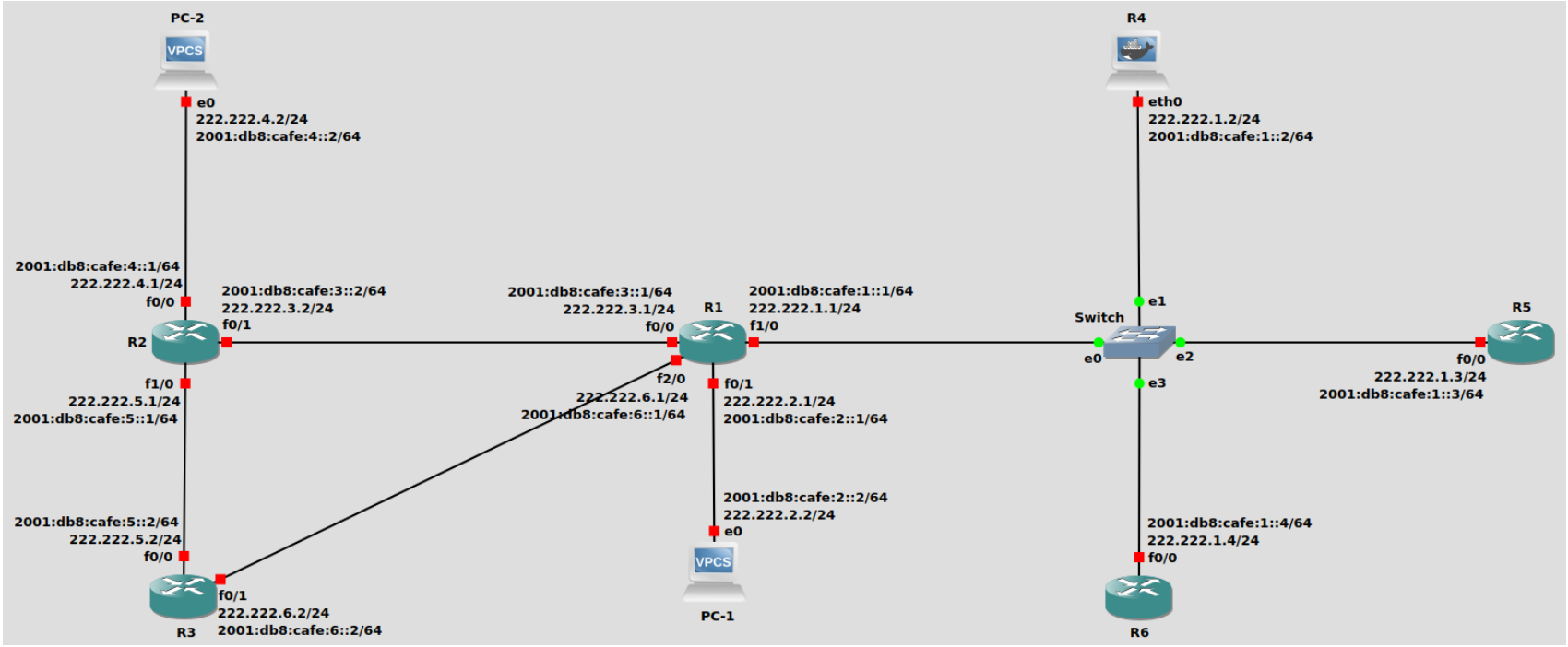
[23] “Docker - Empowering App Development for Developers.” [Online]. Available: https://www.docker.com/

[24] “VMware – Delivering a Digital Foundation for Businesses.” [Online]. Available: https://www.vmware.com/

[25] “Wireshark · Go Deep.” [Online]. Available: https://www.wireshark.org/

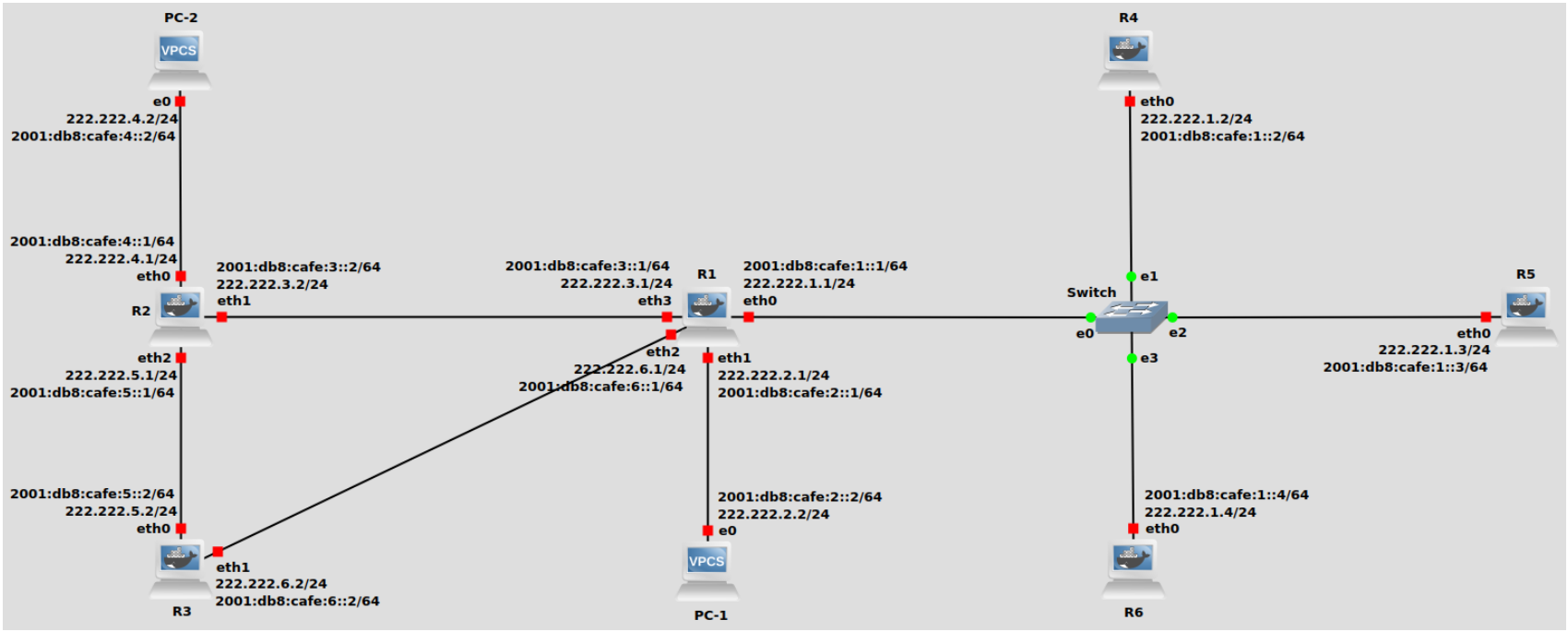
A

Diagram of the network 1-1


PC-2
VPCS
e0
222.222.4.2/24
2001:db8:cafe:4::2/64
R4
eth0
222.222.1.2/24
2001:db8:cafe:1::2/64
2001:db8:cafe:4::1/64
222.222.4.1/24
f0/0
2001:db8:cafe:3::2/64
222.222.3.2/24
f0/1
2001:db8:cafe:3::1/64
222.222.3.1/24
f0/0
R1
2001:db8:cafe:1::1/64
222.222.1.1/24
f1/0
Switch
e1
e0
e2
e3
R5
f0/0
222.222.1.3/24
2001:db8:cafe:1::3/64
R2
f1/0
222.222.5.1/24
2001:db8:cafe:5::1/64
f2/0
222.222.6.1/24
2001:db8:cafe:6::1/64
f0/1
222.222.2.1/24
2001:db8:cafe:2::1/64
2001:db8:cafe:2::2/64
222.222.2.2/24
e0
PC-1
VPCS
2001:db8:cafe:5::2/64
222.222.5.2/24
f0/0
2001:db8:cafe:1::4/64
222.222.1.4/24
f0/0
f0/1
222.222.6.2/24
2001:db8:cafe:6::2/64
R3
R6

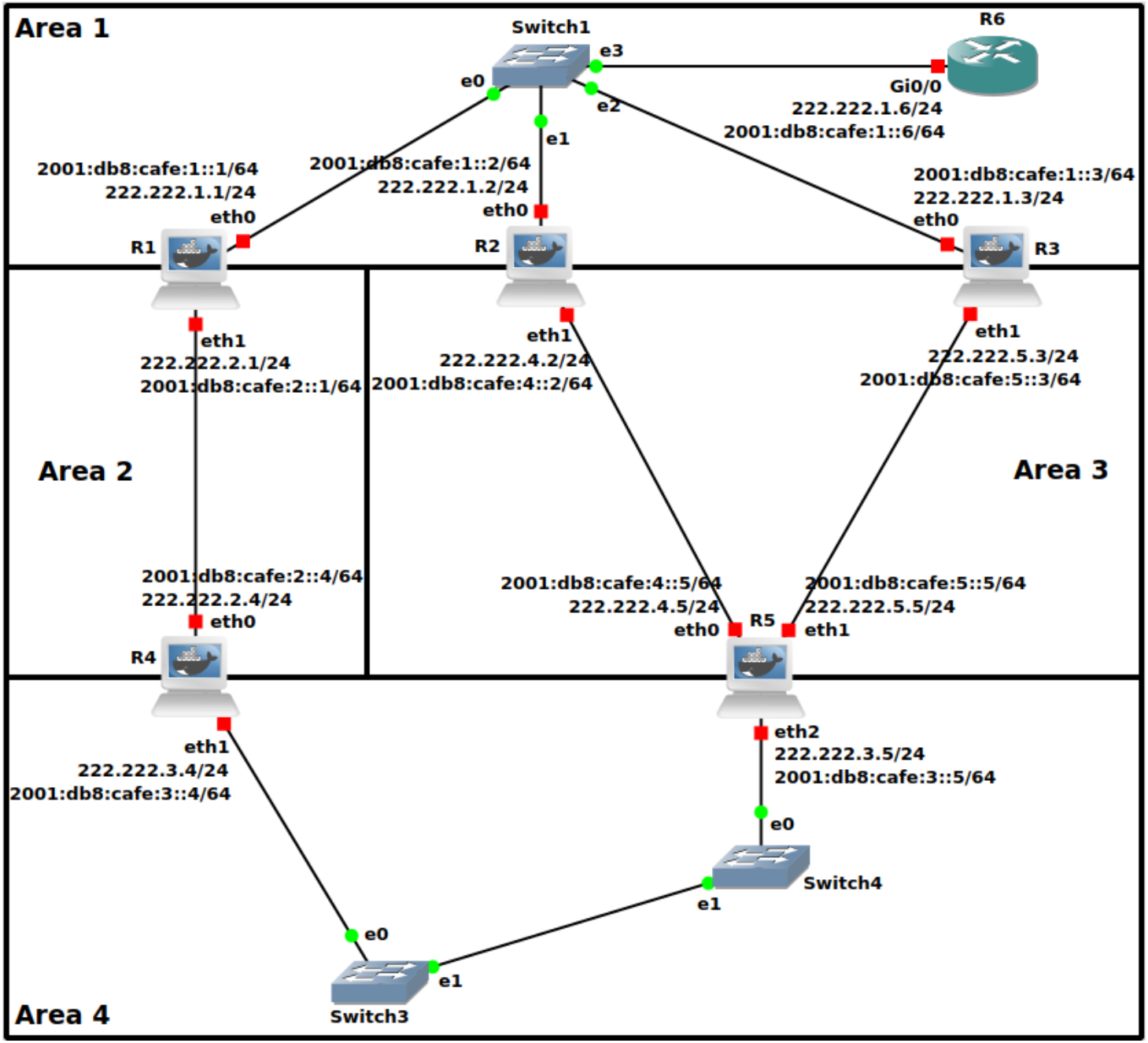
B

Diagram of the network 1-2


PC-2
VPCS
e0
222.222.4.2/24
2001:db8:cafe:4::2/64
2001:db8:cafe:4::1/64
222.222.4.1/24
eth0
R2
2001:db8:cafe:3::2/64
222.222.3.2/24
eth1
eth2
222.222.5.1/24
2001:db8:cafe:5::1/64
2001:db8:cafe:5::2/64
222.222.5.2/24
eth0
R3
eth1
222.222.6.2/24
2001:db8:cafe:6::2/64
2001:db8:cafe:3::1/64
222.222.3.1/24
eth3
R1
2001:db8:cafe:1::1/64
222.222.1.1/24
eth0
eth2
222.222.6.1/24
2001:db8:cafe:6::1/64
eth1
222.222.2.1/24
2001:db8:cafe:2::1/64
2001:db8:cafe:2::2/64
222.222.2.2/24
e0
VPCS
PC-1
R4
eth0
222.222.1.2/24
2001:db8:cafe:1::2/64
Switch
e1
e0
e2
e3
2001:db8:cafe:1::4/64
222.222.1.4/24
eth0
R6
R5
eth0
222.222.1.3/24
2001:db8:cafe:1::3/64

C

Diagram of the network 2-1


Area 1
Switch1
e3
e0
e2
e1
R6
Gi0/0
222.222.1.6/24
2001:db8:cafe:1::6/64
2001:db8:cafe:1::1/64
222.222.1.1/24
eth0
R1
2001:db8:cafe:1::2/64
222.222.1.2/24
eth0
R2
2001:db8:cafe:1::3/64
222.222.1.3/24
eth0
R3
eth1
222.222.2.1/24
2001:db8:cafe:2::1/64
eth1
222.222.4.2/24
2001:db8:cafe:4::2/64
eth1
222.222.5.3/24
2001:db8:cafe:5::3/64
Area 2
Area 3
2001:db8:cafe:2::4/64
222.222.2.4/24
eth0
R4
2001:db8:cafe:4::5/64
222.222.4.5/24
eth0
R5
2001:db8:cafe:5::5/64
222.222.5.5/24
eth1
eth1
222.222.3.4/24
2001:db8:cafe:3::4/64
eth2
222.222.3.5/24
2001:db8:cafe:3::5/64
e0
Switch4
e1
e0
e1
Switch3
Area 4