POLITECNICO DI TORINO

Faculty of Information Engineering

Final graduation project

Multi-area intra-domain routing

Simulation study of the OSPF protocol and its implications on the design of multi-area architecture

Supervisors:
prof. Mario Baldi and dr. Fulvio Risso,
prof. Olivier Bonaventure, dr. Steve Uhlig
and mr Bruno Quoitin

Candidate:
Stefano Iasi

November 2005

Abstract

In this work we study the OSPF protocol focussing our attention on the multi-area architecture. The target of the work is to model the multi-area OSPF and to analyze the complexity it introduces in the intra-domain routing.

To make this we propose a model of the OSPF protocol. The aims of the model is to discover the routing choices performed by the routers without reproducing the exchange of information between the routers. The model was implemented in the C-BGP simulator. The prototype is used to validate the model and to study a special situation that we call *forwarding deflection*. This consists in an inconsistency between the routing plane and the forwarding plane. The cause of the deflection are explained and the condition for it to arise are defined. An algorithm to detect deflection is proposed and implemented in C-BGP.

The result of this work consists in the OSPF model and in the deflection detection algorithm implemented in C-BGP. The first can be used in C-BGP to have most realistic models of real networks when these use a multi-area OSPF. The algorithm that detects deflection can be used to discover if deflection arises in a real network. Above all the work allows to show how the complexity introduced with the multi-area OSPF should be induced the operators to use it with care.

Acknowledgments

This work should not be what it is without the full support that all the professor Bonaventure's team, at the *Université catholique de Louvain*, provided to me.

I have no words to thank Dr. Steve Uhlig for his fundamental and patient help during the writing of this thesis. I would also to thank him to have encouraged and supported me to participate to *CoNEXT 2005*.

Thanks to Bruno Quoitin, for his precious and constant support during the coding work, and to Pierre François, for the numerous discussions about the forwarding deflection problem.

Finally, thanks to all the people of the team for the meaningful human experience they give me. I found a pleasant place where I worked well. Of course, thanks also to professors Baldi and Bonaventure to have believed in this final graduation project.

This works represents also the conclusion of five years of study spent in Turin. I would like to say thanks to my parents that have offered me the possibility to reach this goal and to have always sustained and encuraged me to give the best of myself. Thanks to my sister Roberta and to my brothers Francesco and Emanuele: thinking of them I always found back the smile, because they are joy reasons for me. And how to thank Giuliana? Maybe, it is impossible! She has sheered all the worries and the joys of these years and she has always supported me in each of my choices. All of them are the sure port where I have been able to find shelter.

I would also like thank to the uncles "of the north" that have always received me with affection and availability allowing to feel myself at home. Of course, thanks to the uncle "of the south" and to the prayers of my grandparents, always interested in my progresses in the university.

I have to thanks to Domenico, to which i am indebted, but also to Sebastian, with all his cups of coffee, Luigi for his competent advising, Dejanira for her precious friends, and Maria for her charge of positive energy.

Thanks to *Collegio Einaudi* that has supplied me logistic and technical support during the realization of this work. Special thanks to Emilio and Rossela for their kindness and professionality that helped me to feel myself less far from my home.

Thanks to Ornella and Enzo, that have opened to me the doors of their home

with rare simplicity.

I can not forget Rosario, that with him visits, has carried to me not only a bit of the “air” of my town, but also his great friendship; Simona and Carlo always near to me in the hard times; all the friends in Neviano, especially the ones of the AC group and the friends of the mailing list scattered over the entire Italy.

I would like to conclude saying that i’m happy to have arrived at this goal. It has been something arduous, conquered by hard work; by the way, i have only put the good will... the rest is a gift of the Sky!

Ringraziamenti

Questo lavoro di tesi non sarebbe ció che é, se non avessi avuto pieno supporto da parte di tutto il team del professor Bonaventure, presso la *Université catholique de Louvain*.

Non ho parole per ringraziare il Dr. Steve Uhlig per il suo fondamentale e paziente aiuto durante la scrittura di questa tesi. Vorrei anche ringraziarlo per avermi incoraggiato, ed aiutato, a partecipare alla conferenza *CoNEXT 2005*.

Ringrazio Bruno Quoitin, per il suo prezioso e costante supporto durante il lavoro di scrittura del codice, e Pierre François, per le numerose discussioni sul problema della *forwarding deflection*.

Grazie a tutte le persone del gruppo per la significativa esperienza umana che mi hanno regalato: ho trovato un luogo accogliente dove ho lavorato bene.

Grazie anche ai professori Baldi e Bonaventure per aver creduto in questo progetto di tesi.

Questo lavoro rappresenta anche la conclusione di cinque anni di studio trascorsi a Torino. Vorrei dire, prima di tutto, grazie ai miei genitori che mi hanno offerto la possibilitá di raggiungere questo traguardo e per avermi sempre sostenuto ed incoraggiato a dare sempre il meglio di me stesso. Grazie a mia sorella Roberta e ai miei fratelli Francesco ed Emanuele: pensando a loro ho ritrovato sempre il sorriso perché sono per me ragione di grande gioia. E come dire grazie a Giuliana? Forse, non é possibile! Lei ha condiviso con me tutte le preoccupazioni e le gioie di questi anni e mi ha sempre sostenuto in ognuna delle mie scelte. Tutti loro sono il porto sicuro dove ho potuto trovare riparo.

Vorrei anche dire grazie agli zii "del nord" che mi hanno sempre accolto con affetto e disponibilitá facendomi sentire a casa. Grazie agli zii "del sud" e alle preghiere dei miei nonni: gli uni e gli altri sempre interessati ai miei progressi universitari.

Devo ringraziare Domenico, con il quale sono molto in debito, ma anche Sebastian con tutte le sue tazzine di caffé, Luigi per la sua competente consulenza, Dejanira per le sue preziose conoscenze e Maria per tutta la sua carica di energia positiva.

Grazie al *Collegio Einaudi* che mi ha fornito pieno supporto logistico e tecnico

durante la realizzazione di questo lavoro. Specialmente, ringrazio Emilio e Rossella per la loro gentilezza e professionalitá, che mi ha aiutato a sentirmi meno lontano da casa.

Grazie a Ornella e ad Enzo, che mi hanno aperto le porte della loro casa con rara semplicitá.

Non posso dimenticare Rosario, che con le sue visite mi ha portato non solo un po' dell'aria del mio paese, ma anche la sua grande amicizia; Simona e Carlo sempre vicini nei momenti difficili; tutti gli amici di Neviano, specialmente quelli del gruppo di AC e gli amici della mailing list, sparsi in tutta l'Italia.

Vorrei concludere dicendo che sono felice di essere arrivato a questo traguardo. É stato qualcosa di arduo, conquistato con fatica; in ogni caso, io ho messo solo la buona volontá... il resto é un regalo del Cielo!

Table of contents

Chapter 1

Introduction

The Internet we use every day is made of an interconnection of networks around the Earth. These networks are named Autonomous Systems (AS). Broadly speaking, we can say that an AS is a network under a unique administrative authority. Each AS uses some routing protocols that permit to the routers of the AS to exchange information about reachability of the destinations. This information is used to perform the delivery of the packets from a given source to a given destination in the Internet.

In an AS, two types of routing protocols are used: the Interior Gateway Protocol (IGP) and the Exterior Gatway Protocol (EGP). The IGP allows each router of the AS to reach destinations located inside the considered AS. The EGP is used to announce the destinations reachable by the AS to the AS's neighbors. The EGP also serves at redistributing inside the AS the reachability information about destinations external to the AS.

Today, the standard de facto EGP is the Border Gateway Protocol (BGP) [1]. Several different IGPs are available. The most widely used today are link state protocols. With this technique, a router sends to all its neighbors a description of the state of its links. This information is flooded in the entire network. Having the state of each link of the network, a router can build a map of the whole network. Using this map, it can compute the best path towards each destination.

The link state approach requires that when a router receives the topological information from other routers (their links state), it must store that information. Since an AS can be very large and contain several hundred of routers, the storage of topological information can be cumbersome. For this reason, the idea of a multi-area architecture was introduced. It consists in splitting a network into areas in order to limit the amount of information exchanged among the routers, hence reducing the impact of the size of the network on the protocol. The operator can take advantage of the multi-area approach over time: when the network size increases it only has

to introduce new areas in the configuration. Today the link state with a multi-area architecture is supported by the Open Shortest Path First (OSPF) and by the Intermediate System-Intermediate System (ISIS) protocols.

In this work, we study the OSPF protocol, focusing on the routes computation process. When discussing the details of the OSPF protocol, we show that the important advantages of the multi-area architecture are obtained at a price in terms of the complexity added to the protocol. This complexity does not concern only the effort required for the implementation of the mechanisms that allow a multi-area configuration, but also the difficulty in using the protocol itself: before running multi-area OSPF on a given network, operators must understand many details about the multi-area architecture. When this is not taken into account, unexpected behaviors can be introduced, as we will show.

In the first chapter, the principles of the Link State approach are introduced. Then we show how OSPF introduces changes in the way routing information is exchanged in order to permit to split the domain into several areas. The details of the routes' computation are explained and the advanced features of the protocol are also presented and motivated.

In the second chapter, we propose a model of OSPF. The aims of the model is to capture the details of the routes computation, neglecting the flooding of the topological information. We discuss how it is possible to compute the routes without having to consider the exchange of the routing information among the routers. We model also optional features of OSPF showing how the computation of the routes must be changed if these options are activated. The model is designed to be implemented into C-BGP, an efficient BGP simulator [2]. The basic principles of the simulator are presented to explain our choices in the design of the model.

A prototype of the model was implemented in C-BGP. In the third chapter we discuss some details of the implementation. Some modifications to the topology model of the simulator were required to fully capture the IGP information of the domain. The details of our implementation are discussed and some improvements to the current implementation are also proposed. Finally, the prototype is used to validate our model discussing the output obtained on some scenarios. Our prototype contributes to a more realistic computation of the routes inside C-BGP, when multi-area OSPF is used inside an AS.

In the last chapter we discuss how some details of the routes computation process lead to unexpected behaviors. We show that, when some conditions are verified, the forwarding plane is inconsistent with the routing one. We have called this phenomenon *forwarding deflection*. We discuss why it can be interesting to investigate this behavior and propose a solution to solve it. Moreover, we discuss how deflection can be detected in different ways, showing the advantages and the limits of each approach. An algorithm to detect deflection was also implemented in C-BGP and tested on a sample scenario.

Note that the use of an IGP protocol like OSPF is not restricted to ASes, but to networks in general. We can define an IGP domain as a set of routers that use an IGP protocol to exchange routing information. Talking in terms of IGP domain, and not in terms of an AS, we are able to consider not only the case where the IGP domain coincides with the whole AS but also when more than one IGP protocol is used by different routers in the same AS. Our discussion will hence be more general and will treat also cases where an IGP is used without having an AS, like in a campus network for instance. In this work, we use the term *IGP domain*, or simply *domain* with the more general meaning given above.

Chapter 2

OSPF protocol

OSPF is a link-state routing protocol. Link-state is an approach to the distributed routing problem. It presents several advantages but, when it runs on a big network, like an entire AS, scalability problems may arise. For this reason, OSPF proposes the possibility to partition the network into areas, we then speak of multi-area OSPF.

In this chapter we present the OSPF architecture: the basic principles of the link-state approach are presented and the modifications introduced by the multi-area approach are discussed. Then, we present some features of OSPF and the the algorithm used to build the routes. Finally, advanced features of the protocol are discussed.

2.1 OSPF architecture

2.1.1 Definitions

The first component of a link-state routing protocol is the exchange of routing information beetwen the routers of the domain. This exchange of routing information makes possible for each router to have a complete map of the network topology (all links and routers).

A link is a physical connection beetwen two routers. For us a link allows a bidirectional connection beetwen the routers that are connected by the link, consistently with the notion of an IGP adjacency.

Three main types of links are available: *point-to-point* links, *broadcast networks* and *non-broadcast networks*. Each of them has specific features. OSPF can work with all of them but, for simplicity, we consider only point-to-point links and broadcast networks. *Point-to-point* links consist of a direct physical connection established beetwen two routers only. *Broadcast networks* on the other hand connect more than two routers. When a message is sent on a *broadcast network*, all routers receive

it. On such networks, a router discovers that it is the destination of a packet by checking its layer-2 information.

From a routing viewpoint it can be useful to classify broadcast networks into *transit* and *stub subnets*. This classification is based on the number of routers actually connected to the subnet. In the first case more than one router is connected on the subnet and traffic can transit on it. In the second case only one router is connected to the subnet.

2.1.2 Link State Advertisements

In a link-state protocol routers exchange information by sending Link State Advertisements (LSA). In a LSA, a router describes its links listing all its neighbors. An adjacency is built by a router with a neighbor when the link between them is up in the two directions.

LSAs sent by a router are received and stored by all the other routers of the domain. LSAs are propagated in all the domain with the selective flooding technique, a variation of the *flooding method*. When a node using the second technique receives a packet on one of its interfaces, it forwards the packet on all its other interfaces. With selective flooding, nodes store received packets and when a new packet arrives they check whether they have already received it or not. If it is the case then the packet is discarded, otherwise it is stored and forwarded.

With this approach a LSA injected in the domain reaches all its routers. A consequence of the LSAs flooding is that all the routers in the domain share the same LSA database. In this way, all the routers have the same knowledge of the network topology.

A router sends its LSA periodically in order to inform other routers about the state of its links. A router must also provide the age of the LSAs in its database: when a LSA expires the links it describes are considered down and the LSA is removed.

When a router builds or destroys adjacencies it informs its neighbors about the changes of its link state by sending a new LSA. A new LSA is sent when something changes in the router's links, for example a link is added (or removed), a link goes up (or down) or a neighbor router goes up (or down). In this case the router sends a new LSA describing the new state of its links. The new LSA is flooded and routers that receive it can verify whether the state of the links has changed [1]. If this is the case the description of the link is updated in the LSA database.

[1] To check whether the state of the links changed, OSPF uses some information contained in the LSA header. For more details see [3]

2.1.3 The Shortest Path First algorithm

From the LSAs database a router can build a model of the network topology. The model will be a graph: we call it the *graph of the adjacencies* because only links and routers for which an adjacency was built are represented in this graph.

In the graph of the adjacencies, routers are represented as the nodes of the graph. A point-to-point link is represented as a couple of directed edges, one for each direction beetwen the two routers connected by the link (figure 2.1 a). A transit network is modelled by a star-like topology: a node representing the subnet is connected to each other node (representing a router on the subnet) by a couple of directed edges (figure 2.1 b). Finally, a stub network is represented by a single node: the link with its unique router is modelled by a single directed edge from the router to the subnet (figure 2.1 c).

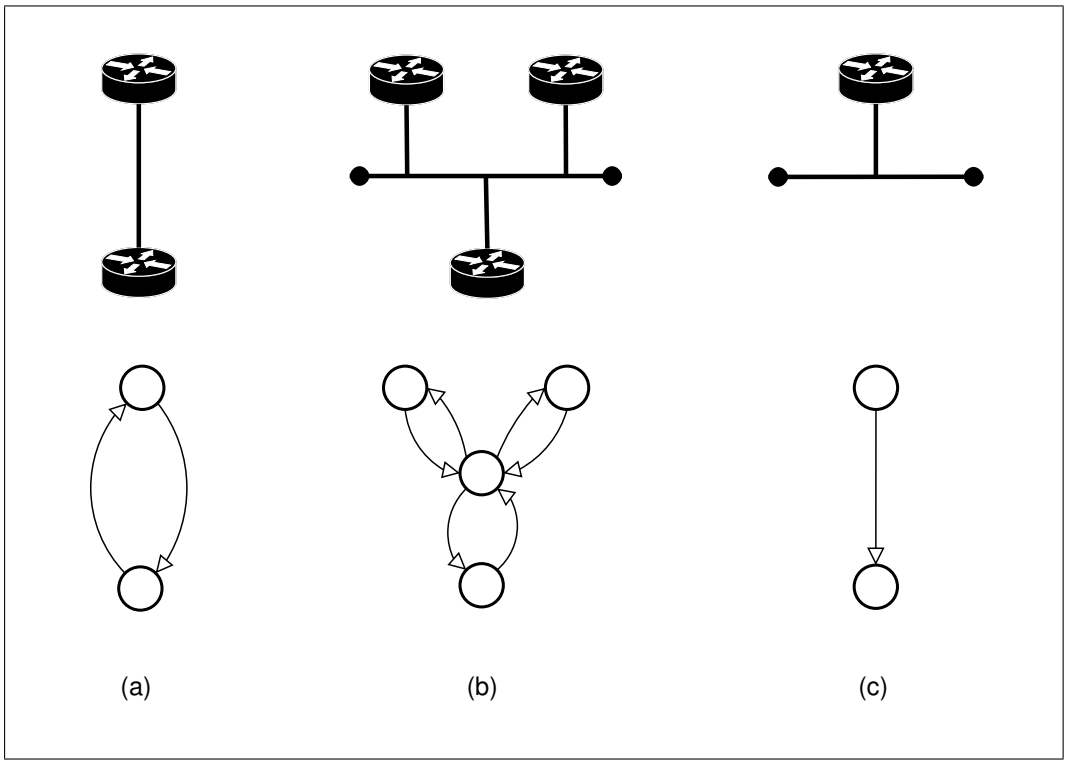

Figure 2.1. Model of the links in the graph of the adjacencies

Once a router has built this map of the network it can build a route towards every other node of this map. This is done in two steps.

The first step consists in running Dijkstra's algorithm [4]. This algorithm takes as input a graph and one of its nodes. It returns a tree rooted at the passed node. The characteristic of the tree is that the nodes that belong to it are reachable by

the root on the considered graph. Moreover, the path beetwen the root and each node in the tree is the shortest path available on the graph. For this reason, the resulting tree is named the Shortest Path Tree. The cost of the path is the sum of its arcs' costs. Because of the way this algorithm works, it is also called *Shortest Paths First* (SPT).

The second step consists in visiting the computed SPT in order to build the routes. The tree is visited and for each node (that can represent a router or a subnet) a route is built and the next hop for the route is calculated. The next hop is the router's neighbor to which a packet must be sent in order to arrive at the destination.

2.1.4 The multi-area architecture

The link-state protocol has several advantages compared to other solutions proposed in the literature[2]. Besides these advantages, some scalability problems can arise. By *scalability* we mean that the size of the network has a direct impact on the efficiency of the protocol. By efficiency we mean the amount of routing information exchanged and the resources required by routers in order to manage LSAs. When network size increases, the number of LSAs injected in the domain also increases. As a consequence, the volume of routing traffic increases and a significant amount of resources are required by the routers to store and manage LSAs.

Based on these observations, the idea of splitting the domain into areas was considered for OSPF [3]. An area is a subset of links of the domain. When a router has an interface in some area, then it belongs to the OSPF domain. If a router has interfaces on several areas, it is on the border of these areas and it is called a *border router*. Otherwise, if a router has all its interfaces inside a single area it is called a *internal router*.

With the multi-area architecture, routers inside the same area exchange detailed information about the topology of the area using the link-state methods discussed above. This LSA exchange is however confined to the considered area: border routers do not flood the LSA of a given area into another.

Moreover, for each area a border router belongs to, it builds summaries to announce destinations external to the area. In the summary the router announces its cost to reach the destination. Routers that receive this information can learn existence of destinations in other areas. The cost towards this destination is computed as the sum of the cost towards the border router that announces it plus the cost announced in the summary. If more than one border router is available in the area

[2]The other solution consist into the Distance Vector approach. For a detailed comparisong between the two mechanism we refer to [5].

[3]OSPF is not the unique link-state protocol that permits a multi-area architecture. Another IGP protocol is IS-IS.

only the one that provides the best cost towards the destination is chosen to exit the area: we call this router the *exit router* for the considered destination. Note that a border router announces the destinations learned in the area it belongs to or that it learned through summaries received from other border routers.

OSPF requires that the areas must be organized in a strict way. A central area called the "backbone area" (area 0) connects all other areas of the OSPF domain. All border routers must belong to the backbone area, as well as to at least another area. A possible configuration of the multi-area domain is shown in figure 2.2.

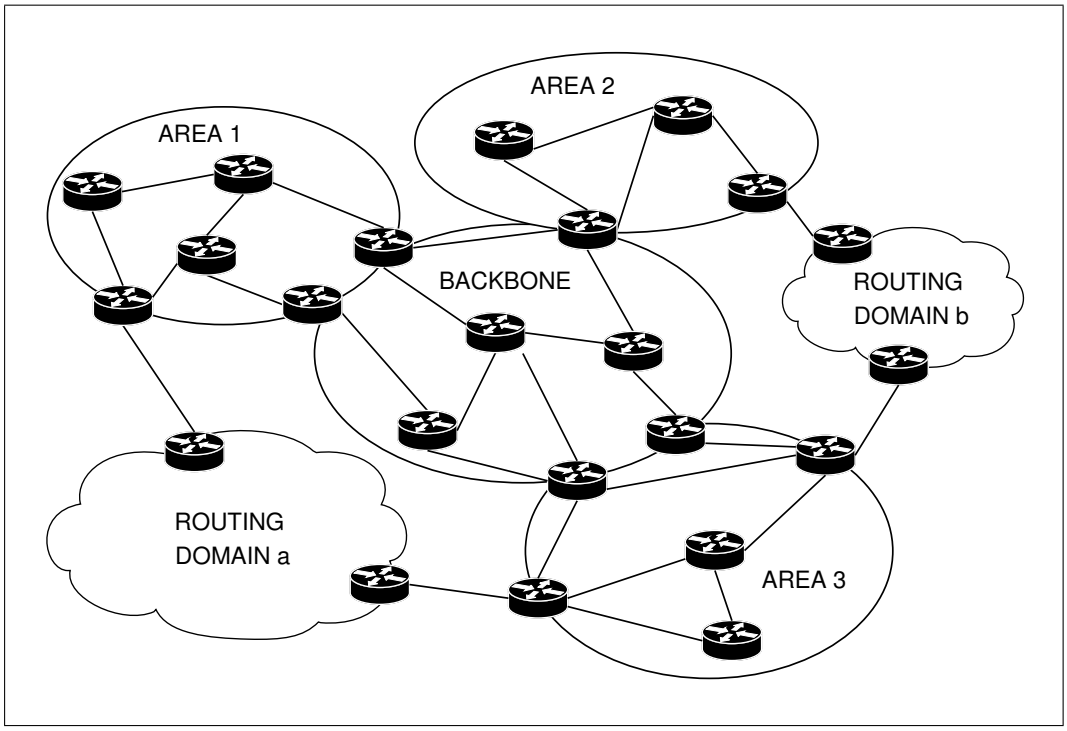

Figure 2.2. A possibile configuration of a domain.

This multi area approach has several consequences that we discuss in the next section.

2.1.5 Impact of the multi-area architecture

The first important consequence of the multi-area architecture is that a router has a detailed knowledge of the topology of the areas it belongs to, while it has only a partial knowledge of the other areas. It knows all the routers and the links (and their cost) that are directly reachable [4]. On the other hand, it does not know in

[4]Throughout this thesis, by *directly reachable* we mean that the router can reach the destination using links that all belong to the same area. Both the router and the destination must belong to

details the areas it is not directly connected to, but has only a simplified vision of them: it knows destinations present in these areas and the cost to reach them.

The second consequence is that unlike for the classical link state flooding, where all routers share the same LSA database, this is not true anymore in the multiple areas architecture. This means that routers in different areas have a different vision of the same domain.

Splitting domain into areas permits to have three main advantages at the cost of a major complexity of the algorithm.

First, the amount of routing information exchanged is reduced. This is because border routers build summaries that permit to hide the details of the topology that are not foundamental to compute the best route towards a destination. Of course this also reduces the amount of information that must be managed by routers.

Confining the LSA propagation inside the area has also the advantage of avoiding the propagation of LSAs in parts of the network where they are not needed. When a change in the topology occurs inside an area, routers in the area will be informed by LSA propagation; routers in other areas will be informed only by the summaries built by border routers. However, if summaries are not changed no change will be propagated in other areas. In this way useless LSA transmission and routes computation are avoided.

The last advantage consists of a reduced complexity in the Shortest Path Tree computation. The complexity of Dijkstra's algorithm is

$$O(L \ln(N))$$

where L is the number of links of the network and N the number of nodes. The literature contains many different implementations that can improve its computation down to a complexity of

$$O(L \ln \ln(N))$$

(see [6]). However, the complexity is reduced only under additional assumptions on links weight or on the number of the links in the network. This means that if the network is bigger and has a larger number of links the benefit of those special implementations is unclear. The multi-area approach, instead, permits to reduce the complexity of the SPT computation simply by reducing the size of the graph that is passed as input to the algorithm. In a link-state protocol the graph corresponds to the map of the entire network. With the multi-area architecture the graph corresponds to the map of the area the router belongs to.

These advantages allow a link-state protocol to work efficiently on a network, scaling to large network sizes. In fact, the number of LSAs transmitted and managed with a classical link-state approach increases with the network size. The multi-area architecture allows to reduce the impact of the network size on the link-state

this area.

protocol. However, the main operational interest of the multi-area approach is that when the network grows the operators have only to add new areas in order to limit the routing traffic and the route computation burden. However, this has a price, although not obvious, in terms of increased complexity of the protocol.

2.1.6 Announcing external domain destinations

To complete our discussion about the routing capabilities provided by OSPF we have to introduce another feature of this protocol. OSPF is designed to carry information about external destinations inside the domain, not only to build routes about destinations in it.

To know external destination OSPF define special routers, called *Autonomous System Boundary Routers* (ASBR). They are on the border of the OSPF domain and know destinations that do not belong to it. They can learn the existence of external destinations by another routing protocol. For example these routers can have an eBGP session or can belong to another IGP domain running a different IGP protocol at the same time. The ASBR reannounces using special LSAs, called *ASBR External LSA*, the external destinations in the domain. When receiving these advertisements, routers can build a route towards each external destination.

Although OSPF offers this possibility, often operators prefer to use a protocol like BGP [7] for this purpose. This was explicity designed to provide an efficient way to redistribute into a domain the routes learned by another routing domain. If the routing domain corresponds with the whole AS, the routing information about the external destination is, in the most of the cases, the routes towards the whole Internet. A motivation not to choose OSPF for this purpose is that a large number of LSAs about external destinations must be flooded inside the domain. This would amount to loosing the benefits of the multi-area architecture discussed above. Moreover, BGP was explicity designed in order to redistribuite external domain routes: it provides several attributes on which the best BGP route choice is based. If an ASBR learns external routes with BGP and announces them inside OSPF these attributes will be lost.

While BGP is the standard de facto protocol to redistribuite external domain routes, this feature of OSPF could be useful only in some special situations. For example if an AS contains more that one IGP domain, and one of them is an OSPF domain, it is possible to redistribute the routes of the non-OSPF domains inside the OSPF one.

2.2 OSPF components

In this section we discuss some of the main components defined in OSPF that allow to implement the multi-area architecture in a domain.

We present the *Hello protocol* that provides a mechanism to build adjacencies with neighboring routers. Different types of LSAs are also presented to announce different types of destinations. We discuss about the Equal Cost Multi Paths that OSPF takes into account during the computation of the routes. Finally the routing table is presented.

2.2.1 Hello protocol

OSPF needs a mechanism to discover neighbors and build adjacencies with them. This function is provided by the *Hello protocol*.

The Hello protocol implements the function of neighbor discovery: it allows to announce the router to its neighbors and, at the same time, to discover them. When a neighbor is discovered an adjacency is built. The Hello protocol uses *Hello packets* that are periodically sent on all router interfaces inside IP packets. This allows the use of multicast reserved addresses when a Hello packet is sent on a LAN: this avoids an end-system, or a router that does not run the OSPF protocol, to receive and process a packet it is not interested in.

Hello packets contain the sender's *router id*. This is an identifier for each node of the network. It is a 32-bit value and corresponds to the smallest IP address among the addresses of the interfaces (only those that participate to the OSPF process) of the router. If no interface is configured on the router (for example it has only point-to-point links with no IP address configured) the IP address of the loopback interface can be used. Hello packets also contain the list of router id of neighbors already discovered and the number of the area that the interface, on which sender sends the packet, belongs to.

Before building an adjacency two checks are performed. First, a router performs the *two way connectivity check*. The Hello protocol is designed to consider a link up only when it is up in the two directions. This happens when a router finds its router id in the list of neighbors contained in the Hello packet.

Second, an adjacency is built only when the two nodes on the link belong to the same area. The check can be performed because Hello packets contain the area id of the interface to which the sent packet belongs to. This last check has an important impact on the area configuration: links between routers that belong to different areas are not used by the OSPF protocol.

On broadcast networks the Hello protocol is also used to select the *designated router*. The purpose of this special router is to reduce the amount of traffic that is sent on the subnet and the amount of information a router must manage. The

designated router is elected on the subnet and when a router must sent a LSA it sends it only to the designated router. This prevents from having to send the information to all the other routers on the subnet.

The other purpose of the designated router is to allow to model a subnet as a star topology. A designated router corresponds to a node in the graph of the adjacencies and all the other nodes on the subnet are connected to it with point-to-point links.

The election process of the designated router is priority based. Priority can be set manually by the operator. Otherwise, the router with the highest router id is elected as the designated router. To permit a fast recovery in case of a failure of the designated router a backup router is also elected. The latter always stores a copy of the information sent to the designated router.

2.2.2 Types of LSA

Once a router has built its adjacecies with the Hello Protocol it can build the LSAs to announce the state of its links. OSPF defines five types of LSAs: *Router LSA*, *Network LSA*, *Summary LSA*, *ASBR summary LSA* and *AS external LSA*. Now we describe their function and the propagation rules for each of them.

Router LSA This is the LSA that permits to announce the configuration of the links of a router. It contains the list of the router's links: for each of them there are the *link id* and the associated cost. The *link id* is the router id of the router on which the adjacency is built. When a link points towards a transit subnet the *link id* is the router id of the designated router of the subnet. Finally, when a link is towards a stub subnet the link id is the IP prefix of the network.

When the link is point-to-point and there are IP addresses associated to the two interfaces of the link, this must be announced two times in the LSA: first it is announced like a regular point-to-point link, and then the router announces a stub network linked to it. The fictitious stub network has a prefix (/32) where the IP address is the IP of the interface on the other side of the link itself. If also a Local IP Subnet (LIS) is configured on the point-to-point link, the router announces two times the link: as a point-to-point link and as a stub network with the same IP prefix of the configured LIS.

Another important function of the *Router LSA* is to announce the special roles of a router: if it is a border router or an ASBR special bits in the packet are set to indicate it.

A *router LSA* describes all the links of a router that belong to a particular area. Router LSAs are only injected in this particular area. As an internal router can only belong to a single area, it describes in the router LSA all its links. An internal router sends its router LSA on all its interfaces. A border router on the other hand builds one *router LSA* per area it belongs to. This *router LSA* contains only the

links in the considered area. The border router sends the *router LSA* only on the interfaces located inside the considered area.

Router LSAs are confined to the area where they are announced. They are always propagated by an internal router. A border router, instead, does not forward a router LSA in an area different from the one it has received it.

Network LSA A *network LSA* is used to describe a broadcast network. Only the designated router for a subnet builds this LSA. In it, nodes linked on the transit network are enumerated. Each node is listed with its router id. A router that receives this packet can obtain the IP prefix of the LIS configured on the transit network using the netmask contained in the LSA.

As for *router LSA*, network LSAs are confined in the area in which they are injected. Router LSA and network LSA provide the topology description of an area.

Summary LSA *Summary LSAs* are built only by border routers. A *Summary LSA* contains destination prefixes and the cost to reach them for the border router. For each area a border router belongs to, it builds *Summary LSAs* to announce destinations that are in the area. It injects this summary in all the other area it is connected to. In this way routers that receive *summary LSAs* learn to reach destinations external to the area through the border router that sent the LSA. These LSAs are confined in the area in which they are injected.

ASBR Summary LSA An *ASBR Summary LSA* is built by a border router to announce reachability of an ASBR. A border router discovers the presence of ASBRs in each area it belongs to by parsing the Router LSAs the ASBRs send. The *ASBR Summary LSA* is injected by border routers in all the other areas it belongs to. In the *ASBR Summary LSA* the border router puts the cost it has to reach the announced ASBR.

In the areas the ASBR belongs to, internal routers learn about ASBR reachability directly by the Router LSA it sends. For this reason, no *ASBR Summary LSA* is injected in these areas.

A border router can learn about the reachability of ASBRs that are not in its areas, through *ASBR summary LSAs*. When this happens, the border router builds another *ASBR summary LSA* it injects in all the other areas. In the summary, it announces the cost towards the ASBR as the cost towards the border router that announces the summary, plus the cost announced in the LSA received.

These LSAs are flooded by internal routers, but the behavior of the border routers permit to contain them in the area where they are injected.

AS External LSA The *AS External LSAs* are injected in the domain by the ASBRs. With this LSA they announce reachability of destinations that are not in the domain. The LSA contains also the cost of the ASBR towards the announced destination. In this case the cost information has a special attribute that distinguishes beetwen two types of costs: *type 1 cost* indicates that the cost is comparable with the cost used in the domain; *type 2 cost* indicates that the cost is not comparable with the cost used inside the domain.

A *type 2 cost* can be used to tag routes learned through other routing protocols that use a link cost not comparable with the cost used inside the domain. Distinguishing beetwen to express a preference when path's cost towards external destinations are learned with a protocol different from OSPF. The *AS External LSAs* are the only LSAs flooded in the entire OSPF domain without restrictions.

All the LSA presented above are exchanged by OSPF with Link State Packets (LSP). LSPs are sent inside IP packets and are designed to carry more than one LSA. Using IP packets a LSP can be sent to a specified router (addressed to its router id) or to a reserved multicast address. LSPs are always exchanged beetween adjacent routers. For more details about LSP we refer to [3].

2.2.3 Equal Cost Multi Paths

Using the LSA database, a router computes the routes towards all the discovered destinations. Before illustrating how the routes are built we have first to discuss Equal Cost Multi Path that OSPF takes into account when building its routes.

When routes are built, only the best route is stored in the routing table. The best route corresponds to the best available path in the network. The path is chosen in terms of its cost defined as the sum of the link's cost that compose it. The route with the lowest cost is chosen in order to optimize network resource usage. However, it can happen that network topology offers several possible shortest paths having the same cost. These are called *Equal Cost Multi Paths* (ECMP).

A router can discover ECMPs in the area it belongs to but also in other areas. ECMP towards a destination in the directly connected areas can be discovered with the SPF procedure with simple modifications to the original version [3]. However, considering ECMP forces us to change the result of Dijkstra's algorithm into a directed acyclic graph, not a tree as originally.

ECMPs towards a destination in other areas can be discovered when a router has ECMPs towards the exit router for the destination. If more than one border router is available in the area, internal routers can discover ECMP if the costs announced for a given destination combined with the cost to reach the border routers result in equal costs.

It can be important to have the possibility to store more than one path towards a given destination. Having this permits to perform load balancing on the network or simply to have a backup path. In the first case the advantage is the possibility to equally distribute the load over the available paths. In the second case, a router can reply fast to a change in the connectivity of the network.

A routing table that supports ECMP must allow the possibility of storing more than one path for each route. OSPF distinguishes multiples paths having the same cost by considering the first link of the path, i.e. the interface on which a packet is sent. When a link points towards a transit network an ECMP can be identified by the router id of the next hop on the path. Thus more ECMP can be available passing by the same transit network. Finally, when a route describes an inter-area destination, potential ECMP can be distinguished with the router id of the exit router for each path. This could be useful to prevent invalidating the entire set of paths in the routing table when the connection with only one of the border routers is lost.

2.2.4 Routing table structure

The routing table of a router contains reachabilility information about destinations. It is populated by the routing process that stores an entry per destination. Since ECMP are supported in OSPF, its routing table can store a set of paths for each entry.

Several fields are associated with an entry. Let us to present only the basic fields and the possible values that they can take.

Destination type This field can take the values *router* or *network*. In the first case the destination is a border router or an ASBR. In all the other cases the field assumes the *network* value.

Destination id This field is used to identify uniquely the route destination. It contains an IP prefix. If the destination is an IP subnet its prefix is used. If the destination is a router this field contains its router id. This is considered as a /32 prefix.

Path type This field specifies the type of the set of paths associated with the destination. This depends on the source information in the LSA database used to build the route. The field can take four possible values:

- *intra-area*
- *inter-area*

- *type 1 external*
- *type 2 external*

If the route path type is *intra-area*, the destination belongs to one of the router's attached areas. These routes are built by examining only Router and Network LSAs.

The *inter-area* value is labelled with routes towards destinations that lie in areas different from the router's attached areas. These routes are built by examining Summary LSAs sent by a directly reachable border router.

The last two values concern routes towards destinations that do not belong to the domain. A router learns about them receiving ASBR Summary LSAs sent by an ASBR. If the ASBR announces a cost of *type 1*, routes built from the received LSA are labelled with a *type 1 external* value. Otherwise, i.e. the destination is announced with a *type 2* cost, the *type 2 external* value is used.

Area This field stores the area from which a router has learned the existence of this destination. In other words the field contains the id of the area from which the LSAs used to build the routes were used. This field has no meaning for routes with path type *external type 1* or *external type 2*. This field is used to distinguish beetwen multiple routes for the same border router as explained below.

Cost This field contains the OSPF cost of the route. When the route has a path type of *intra-area* or *inter-area*, this cost represents the entire cost to reach the destination. Otherwise if path type is *type 1 external* or *type 2 external*, this field contains the cost of the part of the path towards the destination contained in the domain.

Type 2 cost This fields is valid only for external domain routes. It contains the part of the cost's path external to the domain.

Next hop This field indicates the outgoing interface on which the router must send packets to reach the destination. When this interface is towards a subnet the field also contains an IP address. This is the interface of another router on the subnet towards which the packets must be sent.

Advertising router This field is valid only for routes with path type *inter -area*. It contains the *router id* of the border router or of the ASBR that is chosen to reach the route's destination. This happens when multiple border routers share an area.

Most of the fields presented in this section are used in the routing table process in order to perform some checks, while only a small subset of them is used by the forwarding process. Other fields are also needed for a full OSPF implementation, but here we did not consider them. For more details we refer to [3].

2.3 Building the routing table

Here we present the process that, starting from the LSA database, permits to build the routes. We assume that to add a route to the routing table two steps are required. During the first step the route is built, i.e. information from LSA database are converted into a routing table entry. During the second step the built route is installed if some requirements are met. Otherwise, the route is discarded.

Three types of routes can be built in function of the path they describe: intra-area, inter-area and external domain routes. Each time a change arises in the LSA database this can require a recomputation of the routes. Events that cause a recomputation are specified in [3].

For a given destination a router could discover more than one path with different type. For example this happens in figure 2.3 where two border routers are on the same area 1 as well as on the backbone. Each of them announces the destination in area 1 into the backbone. This means that each of them receives a summary for a directly reacheable destination. Each of them can reach the destination through the backbone and directly in area 1. For example R1 can reach R2 by an intra area path (the one dashed) and by an inter-area path (the one dotted).

OSPF enforces a strict order of preference among the possible routes towards a given destination. A route is installed in the routing table only if a route with a preferred path type is not already installed. The preference of the path types is expressend in the following order:

1. *intra-area*
2. *inter-area*
3. *type 1 external*
4. *type 2 external*

2.3.1 Intra-area routes

To build intra-area routes, a router considers only Router LSAs and Network LSAs. These provides the topology of a specific area.

The intra-area routes are built as for the classical link-state protocol, i.e. running the SPF algorithm. However, there are two main differences:

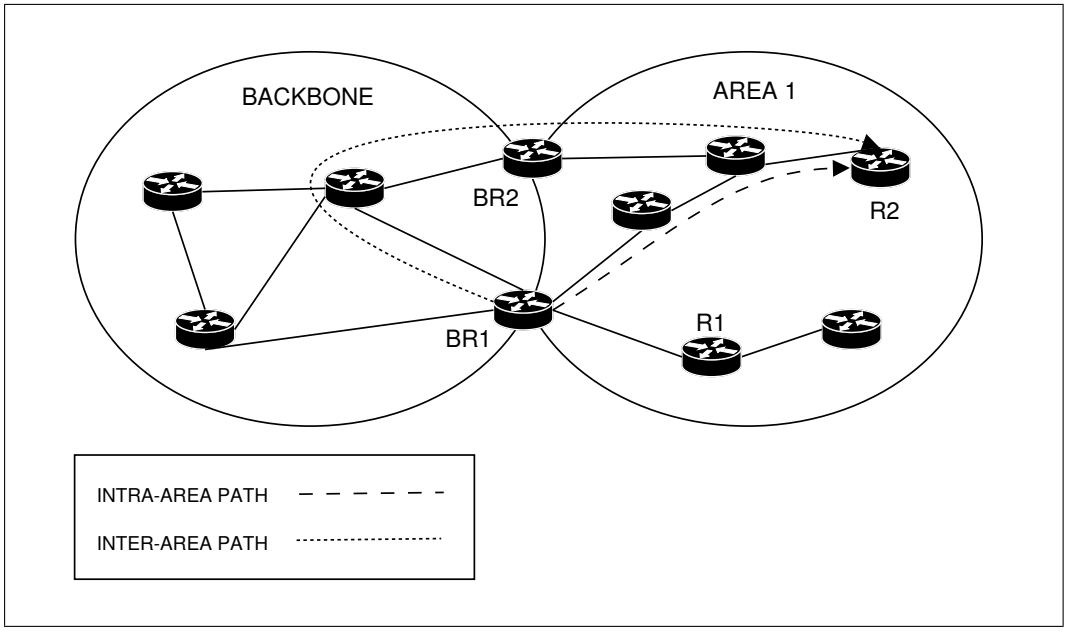

Figure 2.3. Different paths available for R1 towards R2

1. SPF is run on the topology of a single area and not on the entire domain topology;

2. the result of the SPF algorithm is a directed acyclic graph and not a tree due to ECMPs.

OSPF requires also some checks during the visit of the SPT in order to set the correct value for the field *destination id*. The field is filled with the value *router* only if the examined vertex is a border router or an ASBR. Moreover, if a router has at least a link towards a subnet no route towards it is installed: it can be reached by the routes that describe the path towards the subnet it belongs to. The router id will be part of one of the LISs the router belongs to. In this way the amount of routes installed is reduced.

The computation of intra-area routes is done separately for each area. Border routers have a separate process for building intra-area routes for each area they belong to. If two border routers share more than one area, each of them builds a route towards the other. One route is built for each area the border routers share. This is the only the case where in the routing table, more than one entry is stored for the same destination. The entries will be different for the values of the area field.

With the intra-area routes computation a router has routes toward all reachable destinations in all the areas it belongs to. Note that the reachable border routers and ASBRs present in these areas are also identified.

2.3.2 Inter-area routes

To compute inter-area routes, Summary LSA and ASBR Summary LSA are examined.

If the router is a border router it does not consider Summary LSAs injected in an area different from the backbone. This constraint prevents the creation of routing loops and has also a consequence illustrated in Chapter 5. All the other routers consider summaries only from the area they belong to.

Each received Summary LSA (or ASBR Summary LSA) is examined. The Summary is considered only if the router has a route towards the border router that has sent the Summary LSA (ASBR Summary LSA). As specified in [3] the route must be an *intra-area* route. It must also have the field area with a value that corresponds to the area from which summary is considered. Finally the destination type field must have a value of *router*.

If the LSA is considered, a new route is computed with the right fields. The *destination id* field value is set to *router* only if the examined LSA is an ASBR Summary LSA. Otherwise the value *network* is used. The cost of the route is the sum of the cost that router has toward the border router that sent the examined LSA, and the cost announced in the LSA itself.

The route is installed only if it satisfies the route path type preferences rules, explained above. If the path types are the same for the old and the new routes, the new one is installed only if it has a lower cost. If the new route's path has the same cost than the path in the older route, a new ECMP is added to the older route.

With the inter-area routes computation a router built routes toward destinations in areas it does not belong to. The existence of ASBR that are not in directly connected areas is also discovered.

2.3.3 External domain routes

The computation of the routes towards external destinations is performed by examining AS External LSAs. As for the previous case, a received LSA is considered only if the router has a valid route towards the ASBR that sent the LSA. A route is valid if it has a path type inter-area or intra-area and if it has a *destination id* field filled with the value *router*.

For each AS External LSA that passes the previous check a new route is built with the appropriate value for each field. Note that all the routes built here will have a *destination id* field of type *network*. The field area is devoid of meaning for external domain routes. The path type is set in function of the information carried in the LSA.

If the route has a path type *type 1 external*, the new route's cost is the sum of the cost to reach the ASBR and the cost announced for the destination in the LSA.

Otherwise, if the route has a path type with value *type 2 external*, it has a type 1 cost equal to the cost of the route towards the ASBR and a type 2 cost equal to the cost announced in the LSA.

If no route already exists, the new route is installed. Otherwise, the two routes are compared with respect to the path type preference. If the path type is the same, the cost is compared in the following way:

1. if the two routes are *type 1 external* the *type 1 cost* field is compared to choose the smallest cost or to add an ECMP if one is found.
2. If they are *type 2 external* path type the type 2 cost is compared.
3. In the last case, if the cost is the same, the *type 1 cost* is compared.
4. If, after this, the costs are still equal, a new ECMP is added to the older one.

With the external domain routes computation the routes towards the external destinations are built.

2.4 OSPF advanced features

In this section we present some of OSPF's features that can improve the scalability of the protocol or make it adapt faster to network changes.

We present each feature and motivate their existence. The *Synchronization of LSA Databases* is designed to reduce the time a router needs to syncronize its LSA database with the neighbors; *Address Summarization* and *Stub Areas* allow to reduce the amount of routing traffic exchanged. Finally, the most important feature is the possibility to establish *Virtual Links* to overcome the constraint on the area organization and to add redundant connectivity.

2.4.1 Virtual links

Virtual links are defined in order to prevent area partitioning. The partition of an area occurs when two systems in the same area cannot communicate using only links that belong to the area itself.

We already explained that destinations in the same area are reached exclusively using links belonging to this area, i.e. computing intra area routes and always prefer them. However, when a partion arises in an area different from the backbone, something different happens. If at least one border router is present in each partition communication is not compromised. Border routers inject Summary LSA about destinations of each partition in the backbone. From the backbone each border router learns the destination present in the other partition. This allows routers in

each partition to build inter area routes towards destination in the other one. In this way the two partitions are connected by the backbone and are treated as two areas. If a partition does not contain border routers it is isolated.

The partition of an area different from the backbone is illustred in figure 2.4. Here A link failure beetwen BR1 and R3 creates a partition in area 1. Communication beetwen the two partition is still possible and, for example, R1 can reach R2 passing across the backbone.

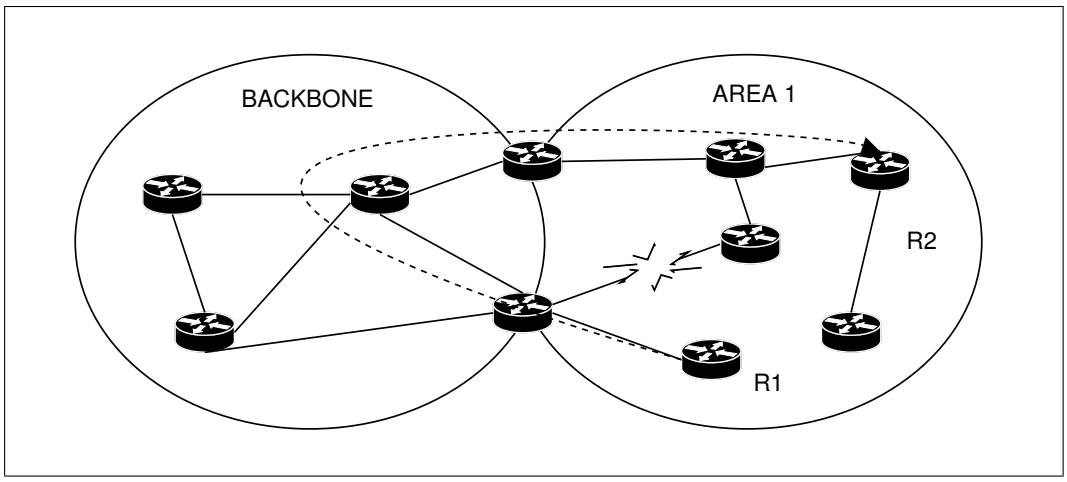

Figure 2.4. In spite of a partition in area 1, R1 can still communicate with R2.

If the partion arises inside the backbone, communication beetwen different areas is compromised, and there is no way to solve it. If a link exists beetwen destinations in different areas it is not used as explained in 2.2.1. So, it becomes important to design a backbone with high connectivity, in order to avoid partitions.

A virtual link is designed to add redundant connectivity to the backbone, using links that do not belong to it. Virtual links can be established beetwen border routers through physical links that belong to the same area (different from the backbone).

Once a virtual link is established beetwen two border routers (this can be done with some manual configuration) they become adjacent: they exchange Hello packets and LSPs in the area on which the virtual link is established. This is possible because LSPs are sent into IP packets and can reach directly the border router at the other side of the virtual link. When such backbone partitioning happens and isolates the two border routers on the backbone, they can continue to communicate using the virtual link.

A possible scenario is shown in figure 2.5. Here a partition is caused by a link failure beetwen R3 and R4. This prevents routers of the backbone from communicating. For example R5 can not communicate with R4. The partition also prevents communications beetwen area 2 and 3. Although a path beetwen BR1 and BR2

in area 1 makes the network physically connected, this path is not used because of the behavior of the border routers. Only the configuration of the virtual link (represented in the figure with a dashed line) in area 1 allows to use this alternative path as if it were inside the backbone.

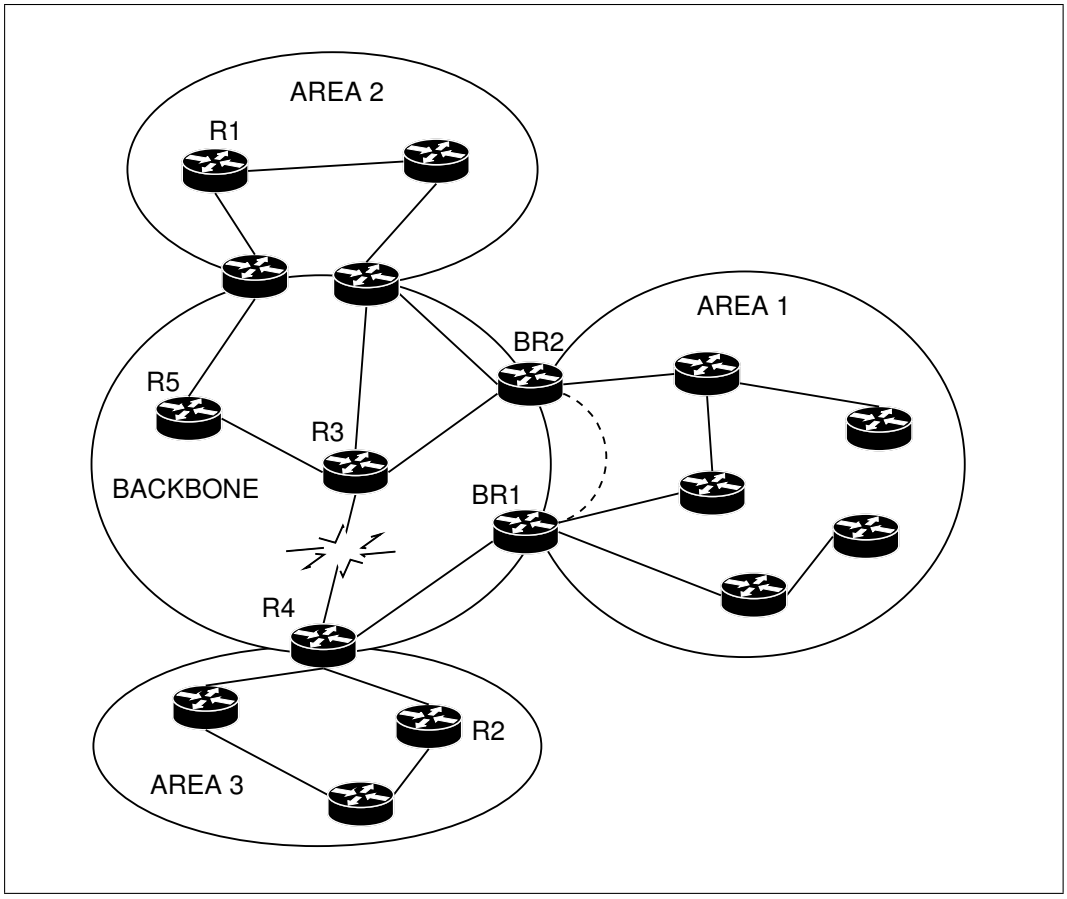

Figure 2.5. The virtual link in area 1 maintains the backbone connectivity.

Another possibility offered by virtual links is to relax the basic OSPF's rule for which all the border routers must have an interface inside the backbone. If a router that is not on the backbone has a virtual link configured with a border router on the backbone, it becomes a border router. This makes possible to add border routers and new areas without to have them directly on the backbone. In the same way it is also possibile to establish a virtual link with a router that has no interface on the backbone but has a virtual link configured.

This provides increased flexibility in network design. When a new area must be added to the domain (for example because the network size increases), it is not required that a router in the area must belong to the backbone in order to become a border router. A virtual link can be established beetween it and a border router

on the backbone passing through another area. Figure 2.6 shows such a situation. Here the virtual link beetwen BR1 and BR2 connects area 2 to the backbone.

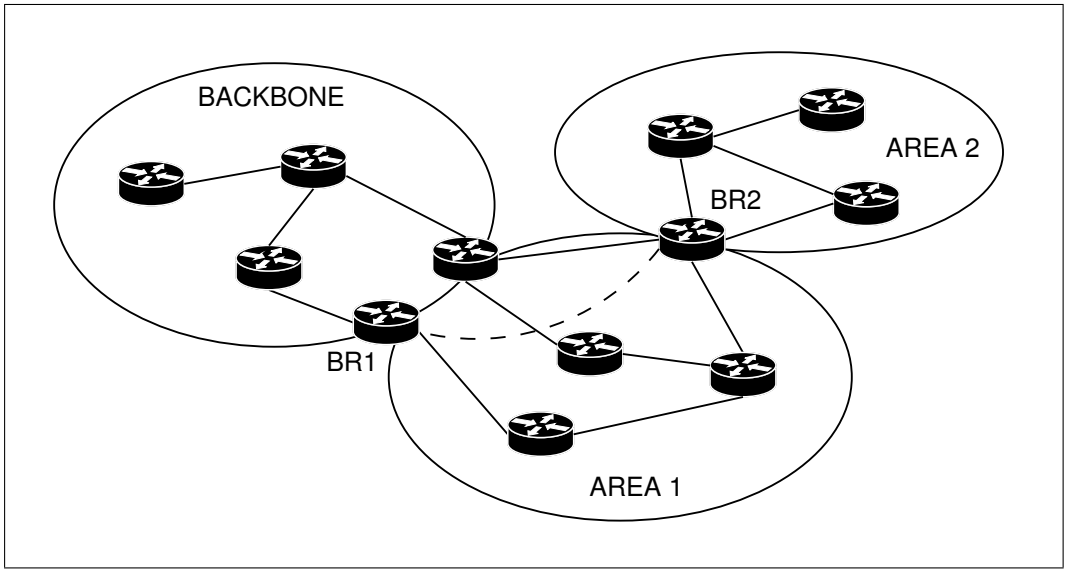

Figure 2.6. The virtual link in area 1 connects area 2 to the backbone.

Virtual links are a powerful tool that offers more flexibility in design and fault tolerance to routing. However, it requires a manual and correct configuration and this can be expensive for operators.

2.4.2 Synchronization of LSA Databases

Synchronization of LSA database is performed in order to permit to a new router discovered on the nework to share the LSA database of its neighbors in a short time. Natural mechanisms designed to send LSAs are not adapted to synchronize two different databases. The event that permits to receive neighbors' LSAs are a change in the topology or the expiration of the timer associated to LSA in neighbors' database. If a router must wait for receiving LSAs with this natural mechanism it might take a lot of time during which the newly added router has a limited reachability.

To avoid this, when a router discovers a new adjacency, it sends a *database description* LSP. In it, the router inserts some LSA headers from its database. The other router waits for receiving the message and replies, confirming its correct reception. The reply also contains some LSA headers of the second router. The process continues until the two routers have sent all their LSAs. If one router finishes before the other, it can send empty packets. At the end, if one of the routers discovers

that the other has some LSAs it does not know, it requests them with a *link state request*.

A priority based mechanism permits to define the two routers as master and slave in the synchronization process. First, *database description* packets sent are empty but contain the *OSPF id* that permits to select the master and the slave. After the master sends the first packet that contains its LSAs, the synchronization process begins.

Database synchronization is a feature that allows a router to share its LSA database in a short time, with its neighbors. This is very useful when a new router is inserted in the network and its database is empty.

2.4.3 Address aggregation

Another important feature of OSPF is address aggregation. It allows to configure a border router in order to summarize the routing information it sends. This is possible for the structure of IP prefixes that are announced. A border router injects a Summary LSA in an area where it announces all the destinations it can reach in the others, or it can reach through other border routers. For each destination to be announced, it must build a Summary LSA. The nature of the IP prefixes allows to aggregate this information: if the prefixes announced have a common part, they can be announced as a unique prefix. In this way, a border router announces only a prefix that groups several destinations sharing a common prefix. The cost announced is the worst of the costs among the set of destinations.

The consequence is that the amount of Summary LSA injected in the network by border routers is reduced. This is an important feature that allows OSPF to be scalable. In practice, the power of this feature depends on the IP address distribution. If they are badly distributed, the summarization of the addresses can be bad too.

Summarizing destinations has an impact on the routing tables, too. Firstly, the router that receives the LSA summary will build a more compact routing table: instead of having as many routes as the destinations considered in the summary, it will have only one route for all of them. Secondly, the path chosen to reach some destinations can be different in the presence of summarization. The information contained in each summary is reduced respect to the information included in each single LSA built for each single destination. Summarizing implies an inevitable loss of information and for this reason, routing could become less efficient from the viewpoint of the shortest paths. This is another source of suboptimal-paths.

Address aggregation has also an impact on where area partitions may arise. In this situation, some destinations in the area can be unreachable. This may happen when a router external to the partitioned area tries to reach a destination in the area that is in a different partition than the border router it uses. In this case, it

is better to disable address aggregation. Of course, care when designing areas with high connectivity is also very important.

Address aggregation is a powerful instrument that can reduce the amount of the control traffic exchanged. Its efficiency depends on the IP address distribution and it must be carefully used in order to prevent a badly designed routing. Another aspect to take into account is that it also requires manual configuration.

2.4.4 Stub areas

OSPF offers possibility to configure some *stub areas*. A stub area is an area where no traffic directed outside of the domain can be injected because no path to exit the domain is available. In other words, a stub area does not contain any ASBR. If this happens, border routers in the area can be configured to not inject in it the AS External LSAs (ans ASBR Summary LSAs) they receive. They only announce a default route (`0.0.0.0/0`). The internal routers in stub areas will use this default route for all the external destination.

Stub areas allow to reduce the amount of routing traffic flooded in the area. This can have a significative impact because for many ASs the majority of the link state database may consist of AS external LSAs, if no other protocol, like BGP, is used. However, this feature also requires manual configuration.

2.5 Conclusions

In this chapter we presented the basic link-state concepts on which OSPF is based. Link-state permits each router to announce the state of its links by sending LSAs. These are propagated into all the routing domain through selective flooding. This permits all routers to build the same LSA database. From the LSA database a map of the network is obtained and on this map Dikjkstra's algorithm is run in order to obtain a route for each destination in the domain.

To add scalability to the Link State approach the routing domain is split into areas. OSPF requires that a central area called the backbone be defined: all the other area must be adjacent to this area. Border routers are routers that belong to more than one area (and to the backbone too). A border router injects in each area it belongs to summaries about the reachability of destinations that are in the other areas of the domain. Thanks to this information, routers in the area can reach all the destinations of the domain. In the same way OSPF defines the Autonomous System Boundary Routers that inject in the domain summaries about destinations external to it.

The routing table was presented and the way a the routes can be built too. Three types of routes are available: intra-area, inter-area and external domain. The first

describes paths that belong to a given area; the second describes paths that belong to the domain; the last describes paths to reach external destinations. During the routes computation OSPF permits to consider Equal Cost Multi Paths.

Special features are designed in OSPF in order to increase the flexibility of the design process of the domain, to further reduce the amount of control information exchanged, or to have fast mechanisms to reply to network changes.

The multi-area architecture, powered by the OSPF advanced features, permits to obtain a routing protocol that allows to reduce scalability problems that arise in large networks. The most interesting aspect of the multi-area approach seems to be possibility to make the efficiency of a link state protocol as independent as possible from the size of the network: when the network size increases the operator must only to add new areas. This has an obvious price in terms of the complexity introduced in the protocol.

Chapter 3

The OSPF model

In this chapter we present our OSPF model. Its design was guided by two main guidelines: obtaining a simple model, in order to simplify its implementation and maintenance, and the requirements to capture some aspects of the OSPF protocol useful for operators. The implementation environment has also influenced our choices, so we first present C-BGP [2], the simulator in which the model was implemented. Then, we illustrate the assumptions of the model, to finish with the model itself.

3.1 C-BGP: the environment of the model

C-BGP is a simulator that implements a model of the BGP protocol. It can be used to study how BGP routers choose the best route in scenarios with one or more ASs. A simulation with C-BGP allows to know whether BGP has converged and, if so, the best BGP routes chosen by each router in the simulated scenario. From this viewpoint we can say that C-BGP uses simulation as a way to build the AS's routing tables. In this meaning C-BGP is a routing solver. It represents an efficient implementation of an API that permits to compute the AS's routing tables. C-BGP, by design, does not capture the aspects of the network control plane that lie below the layer-3.

For the scope of this work we focus our discussion on scenarios where a single AS is modelled. From this viewpoint, the most interesting aspect of C-BGP seems to be the possibility to take into account the effect of the routing information injected by AS's neighbors routers. Other simulators like SSFNet [8] or J-SIM [9] typically assume that prefixes, internal to the model, will be originated by routers themselves. The approach of C-BGP, instead, consists in injecting into the model real routing information when it is available.

3.1.1 Autonomous System model of C-BGP

C-BGP assumes that the routing domain coincides with the AS. To correctly model the AS from a routing viewpoint, two main aspects must be considered: the diversity of the routes towards external destinations, and the configuration of the BGP process on the routers. The first is provided by the different routes that are injected in the AS from its neighbors. The second depends on the goals of the network operator and allows routers to choose beetwen several available routes to reach a given destination. Moreover, the last one influences not only the choice of the best route, and thus the final result of the computation, but also the way in which the routes are propagated by BGP inside the AS. The goal of C-BGP is to capture these two aspects of an AS's reality, in an efficient way. To obtain this, the simulator models the AS network topology, the BGP routers' configuration and the routing information.

The C-BGP's model of AS network topology does not include physical details, like physical links or hubs, as they are not significant from a routing viewpoint. Subnets and end-systems are also not modelled because they are not fundamental for the propagation of BGP routes. The topology is thus represented in C-BGP by a graph where nodes are routers and edges are the layer-3 links between routers. The links can be configured with asymmetric weights. A limit of the solver is that, actually, it cannot represent more than one link between a couple of router. This does not allow to model scenarios with layer-3 redundant connectivity. Moreover, in the network reality, more than one single IP address is often available on a router, at least one for each interface. C-BGP maps all the available IP addresses of a router onto a single IP address [10].

On the topology, C-BGP models the BGP routers. Each node of the network is fitted out with additional information that models the router's BGP configuration. The simulator is able to parse configuration files from different types of routers. Another possibility consists in using a Cisco like Command Line Interface (CLI).

C-BGP models the routing information, of two different types: about internal and external AS's destinations. The first is built with an IGP model implemented in C-BGP. The second is obtained by simulation. The routers with active eBGP sessions are filled with external AS routes. Then the simulation engine simulates the exchange of the routes beetwen the BGP routers. If BGP converges in reality [11], at the end of the C-BGP simulation, the routing tables of the AS's routers should contain the best BGP routes. If the real BGP does not converge, the C-BGP simulation does not stop [10].

Figure 3.1 provides a schematic view of C-BGP. As shown in figure 3.1, the IGP routes computation and the external routes computation are strictly connected: this relationship will be explained in the next subsection.

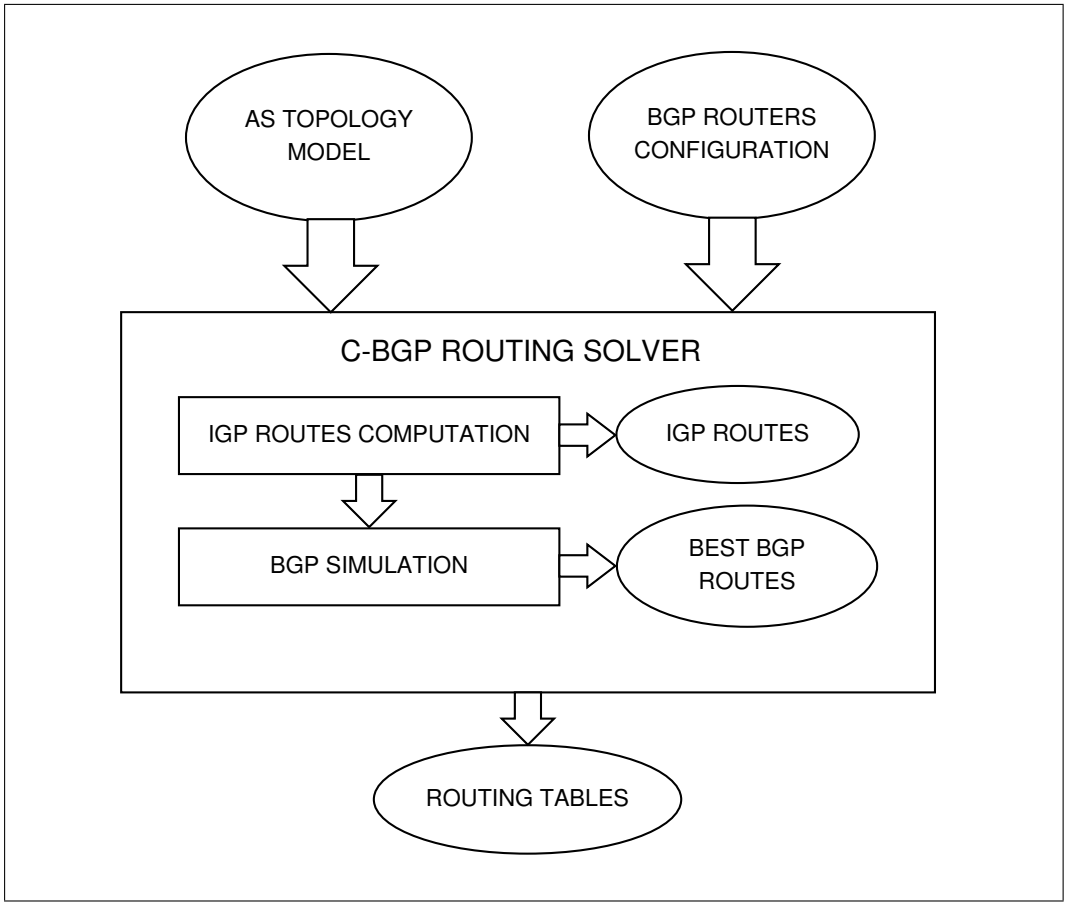

Figure 3.1. Model of the C-BGP routing solver.

3.1.2 The BGP model of C-BGP

The model of BGP that C-BGP implements contains many details about the BGP protocol. Here we discuss which aspects have not been considered in C-BGP. We refer to [10] for additional details.

Many aspects of BGP are very expensive from the viewpoint of the time and resources required during the simulations. To obtain an efficient solver two main choices have been been made. The first choice is to not incorporate any concept of time in C-BGP. This is in stark contrast with other simulators, which are discrete event simulators. There is thus no way in C-BGP to study the dynamics of the BGP messages exchange. As to now, it is only possible to know whether BGP converges or not. In this perspective, C-BGP can be used to study *what-if* scenarios to know if BGP converges or not, and to analyze the effect of changes in the C-BGP model on the outcome of the BGP routing.

The other important simplification consists in not modelling each TCP connection on which BGP sessions relies. The target of TCP is to preserve the order of the BGP messages exchanged among routers and to provide reliability. The same effect is obtained for all the router of the domain with a unique first-in-first-out queue on which each router pushes its BGP messages. The C-BGP's engine pops messages from the queue and realizes the delivery of the messages from the sender to the destination.

The delivery of the BGP messages is realized hop by hop. This means that a longest IP prefix match is performed on the message's destination IP address by consulting the routing table of the sender. The next hop, i.e. the next router in the path from the source to the destination, is found and the message is delivered to it. The process is reiterated until the packet arrives at the destination.

Delivering of messages hop by hop is required in order to avoid the delivery of a message between routers that are not reachable. For instance, two routers can not reach one another when a link failure arises in the network, partitioning the network and leaving each router in a different part of the network. In BGP, problems during BGP session establishment will be indicated by TCP connection establishment. Since C-BGP does not model the details of the TCP connections, the delivery of the message hop by hop is a simple way to perform the connectivity check without a full implementation of TCP. In this way BGP routers can be configured to have the desired BGP session, but only the working sessions will be used in the simulation. Moreover, exchanging the messages in this way allows to discover the actual path that each message has to follow.

The consequence of this way to perform the delivery of BGP messages is that IGP routes and the way to obtain external AS routes are strictly linked in C-BGP. In C-BGP the IGP routes must be computed before starting the simulation, in order to permit the correct delivery of the BGP messages.

3.1.3 The IGP model of C-BGP

The IGP routing model of C-BGP models a link state routing protocol. As we explained in chapter 2 the LSA exchange permits each router to obtain a map of the routing domain. Since in C-BGP the topology of the AS is available in the C-BGP topology model, no LSA exchange is required between the routers. Moreover, the aim of the IGP model is not to study the dynamics of the IGP protocol but only to provide the correct IGP routes. These are important for two raesons. Firstly, they provide to C-BGP the IGP cost that a router has towards a destination: this is used in the BGP decision process to choose the best route. Secondly, they can be used to model more realistically the exchange of the BGP messages.

By default, C-BGP performs the IGP routes computation by running Dijkstra's algorithm for each router of the AS. During this computation only links belonging to

the given AS are considered. This corresponds to modelling the behavior of a Link State protocol using a single area. In some situations, this default behavior is not enough. For instance, it is required for scenarios like the one presented in figure 3.2. Figure3.2 represents two ASs that exchange routes with an eBGP session configured between routers *R1* and *R2*. When router R2 receives BGP routes from router R1, it reannounces them into AS2. The BGP messages contain also the BGP next hop, which is supposed to be the exit router to leave the AS towards the destination. In this case the next hop corresponds to router R1 from which the routes were learned. A problem can arise if router R1 is not reachable in AS2. Routers that will receive the BGP routes will not be able to use them because the next hop is unreachable. In BGP, a route cannot be considered by the BGP decision process if the next hop of this route is not reachable inside the IGP.

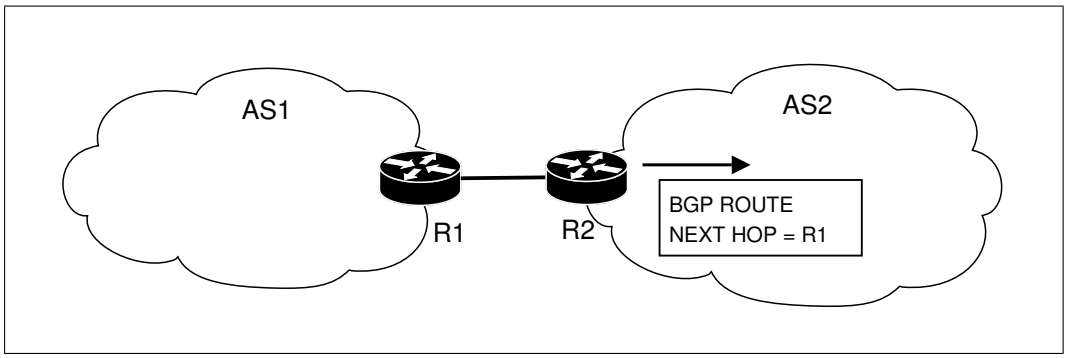

Figure 3.2. A scenario in which `IGP inter` option can be used.

Two solutions are possible to solve the problem: router R1 is known in AS2, because it is learned through another protocol (for example as an external destination in OSPF); otherwise the BGP option *next-hop-self* can be configured on R2. With the *next-hop-self* option R2 announces itself as the next hop for the announced routes. C-BGP models the first case by modifying the behavior of the IGP model presented above, by setting the option `IGP inter`. This permits to compute the SPT considering also the links outside the ASs, not only the links inside the given AS. Otherwise the *next-hop-self* BGP option is implemented in C-BGP and can be activated on the appropriate BGP sessions.

The computed routes are valid as long as no change in the topology occurs[1]. If this happens an inconsistency beetwen the network topology and the routing tables arises. In this case the computation of the IGP routes should be repeated.

The prefixes discovered with the IGP computation correspond to the routers' loopback interface IP only. This happens because there is currently no possibility to model any LIS (Local IP Subnet) or IP interface on the links in C-BGP.

[1]A change can consist in a change in the links weight or a link or a router failure.

The main limit of this simple model is its inability to capture multiple areas inside the IGP. Today several operators, particularly large tier-1 providers, use this type of configuration. Studying the behavior of the IGP with multiple areas is certainly relevant today. Given that other network simulators have a multi-area IGP model [8] [9], we designed such a model into C-BGP.

3.2 Assumptions

The challenge of the design of any model is to find the right trade-off between how well the model captures reality and the simplicity of the model itself. These two guidelines were central during our work towards building a model able to supply C-BGP with the correct IGP routes in a multi-area IGP scenario. Furthermore, our model had to be usable to investigate some aspects of OSPF with a multi-area configuration, as we will show in chapter 5.

Finding the right trade-off between these two objectives can be hard, particularly when considering the consequences on the model complexity. The inspection of multi-area OSPF behavior can be performed accurately only with a fine level of details about the IGP process. At the same time, computing routes inside C-BGP must be done in an efficient way.

3.2.1 A static approach

Following the C-BGP philosophy we built the OSPF model using a static approach. By static approach, we mean that we emulate the behavior of the protocol without reproducing how it actually exchanges messages. We prefer to operate with all the information already available in the model about the domain. This consists in the network topology, the routing tables and the routers configuration. C-BGP supplies all this information from a unique centralized viewpoint. It is possible in C-BGP to access each of the previous information in order to compute the routes.

For all the previous reasons, we do not implement LSAs exchange. Routers in C-BGP use the global map of the topology available in C-BGP. In our model, we limit this global visibility a router has, by the way routes are computed. This limited visibility is equivalent to the visibility routers have in the OSPF reality. The visibility of an OSPF router corresponds to the complete map of the areas it belongs to, distance information about inter-area destinations, and finally distance information about external domain destinations.

We said that the knowledge of the whole map of the topology in OSPF is used for intra-area routes only 2. It is not sufficient in order to build the inter-area and external domain routes without exchanging LSAs. In our static approach we inspect the border routers' routing tables to emulate this advertisement of inter-area routes.

In the same way, we inspect the ASBRs links to find information to build external domain routes. We thus aim at obtaining the same outcome of the IGP routes computation as in reality, without any exchange of LSAs. Ideally, doing this should also be efficient in terms of resource and time consumption. We take advantage of the global vision on the domain provided by C-BGP, to statically build the routes.

This static approach implies also to consider a route valid unless a link or a router failure arises. In such cases, we have to recompute all routes from scratch when any detail of the model changes. This behavior is similar to the default behavior of C-BGP for what concerns the IGP routes computation.

3.2.2 Enriching the topology

We need to configure the multi-area architecture in the domain. The configuration consists in assigning each interface of a router to a specified area. Moreover, it must be possible to identify border routers, internal routers and ASBRs and to discover whether a given router belongs to a given area.

We also want the model of the topology to represent LIS, although this is not supported in C-BGP yet. This requirement is motivated by the following considerations. While in inter-domain routing, prefixes are injected in the BGP border routers using real data, in the intra-domain routing prefixes are obtained mainly by AS's LIS discovered during the IGP routes computation. LIS are configured on subnets and, optionally, on point to point links. If we do not consider the LIS our model will not be able to capture a large fraction of the intra-domain routes in the AS. Considering only routers in the topology we would have had only routes towards /32 prefixes. This differs very much from reality.

3.2.3 Modelling external domain destinations

We provide minimal support for injecting external domain routes inside our model. This choice is motivated by the considerations discussed in section 2.3.3. C-BGP has already a mechanism (the BGP simulation) that allows to compute the best external domain routes for each router. Hence, full support for the advertisement of external routes inside the OSPF model is unnecessary.

Our support of these external domain routes is limited to provide a mechanism replacing the behavior of the `IGP inter` option in the former IGP model of C-BGP. This allows to make external peer AS routers reachable inside the IGP model. If we treat them as external destinations in OSPF, routers will then reach them using OSPF routes.

As shown in figure 3.2, the router that should be announced is connected to one or more AS's routers. The links that connect these routers will not belong to any area: they are external with respect to the OSPF domain. We can model this

situation considering as external domain destinations the routers connected to links that do not belong to the domain. These are treated as *type 1 external* destinations. We do not consider the possibility to advertize *type 2 external* destinations. In OSPF, *type 2 external* routes are considered differently from all the other OSPF routes. Their cost cannot be compared with the one of other OSPF routes [3]. Supporting them inside our OSPF model is thus independent from our assumptions related to the multi-area OSPF. We chose not to support *type 2 external* routes mainly because these routes are due to be originated from other routing protocols, hence are out-of-scope with respect to our OSPF model.

3.2.4 Mapping the Hello protocol

As explained in chapter 2, the OSPF *router id* corresponds to the smallest IP address among the IP interfaces of the router. We choose to map the *router id* to the loopback interface of the router. This choice is motivated by the fact that the use of the *router id* is to distinguish beetwen several routers in the domain. In C-BGP this is already done through the loopback interface. However, modelling the *router id* in this way has an impact on the routing table computation that will be analyzed in the next section.

We already said that no LSA exchange is performed since the topology of the network is already available in C-BGP. For the same reasons no Hello packets exchange is required in our model. However, the two checks performed by the Hello protocol, described in 2, must be considered in order to avoid building adjacencies that would not be built by the real OSPF. We map these two checks at two different locations: during the configuration of the area and during the SPT computation.

The first check realized by the Hello protocol consists to not build adjacencies between neighboring routers that do not belong to the same area. In our model we perform this check by avoiding the possibility to have such a situation. This can be done during the configuration of the domain when each link is assigned to an area of the domain. Before adding the link to the area we check if both routers on the link belong to the area itself. If this is not the case an error message is returned to the user and the OSPF model is not built inside C-BGP. We assume that OSPF routers' configuration must be correct for our model to build the OSPF routes inside C-BGP.

The second check performed by the Hello protocol is the *two way connectivity check*. This check considers up a link only when both directions are up. In our model we perform it during the computation of the SPT. When a link between two nodes is considered, we check if the link is up in the two directions. If this is the case the SPT computation continues considering this link. Otherwise the link is ignored.

3.3 The core of the OSPF model

Once we have clarified our assumptions we can discuss our model in details. In this section we present the core of the OSPF model, i.e. the way we compute OSPF routes with our static approach. We show how the routes are built and why our approach is coherent with the OSPF specifications.

The process of routes computation is split into three main steps that correspond to the computation of intra-area, inter-area and external domain routes.

3.3.1 Intra-area routes

The first step of the route computation algorithm consists in building intra-area routes. The computation is performed for each router of the domain.

A router computes its intra-area routes performing the SPT calculation separately for each area it belongs to. During Dijkstra's algorithm links are examined. Contrary to the classical Dijkstra algorithm we consider only links that belong to the given area. The links are not considered in the same manner. When the link is a point to point link, a check is performed in order to manage an optional LIS configured on it. In this case, a node N is added to the shortest path tree and the LIS's prefix is associated to this node. The node N is the son of the current router's node in the SPT that we name C(figure 3.3 a). If only the interfaces are configured on the link, and no prefix is available on it, N is associated to a prefix /32. This corresponds to the IP of the interface on the other side of the link, from the viewpoint of the router C (figure 3.3 b). The cost of the new edge between the node C the node N corresponds to the cost of the point to point link. This behavior emulates how a router announces a point to point link with a configured LIS (see chapter 2).

The result of the algorithm, as discussed in chapter 2, is a directed acyclic graph. The graph is visited and for each node (that can be a router or a subnet) a route is built and installed in the routing table. The fields of the routes are filled as requested in section **??**. Moreover, during the building of the SPT next hops are calculated for each node considering ECMP. This information is also stored in the routing table entries.

As already discussed in section **??**, OSPF specifics require that, if a router has an interface configured, no route towards it must be computed. However, we do not implement this behavior in our model because we do not map the *router id* in the same way. As explained in the previous section, we use as the *router id* the router loopback interface. If we do not store a route towards a router that has an interface configured, the router will be not reachable. There is thus no relationship between the router loopback interface and the LISs a router belongs to. The result will be that routers with configured interfaces will be unreachable and the exchange of BGP

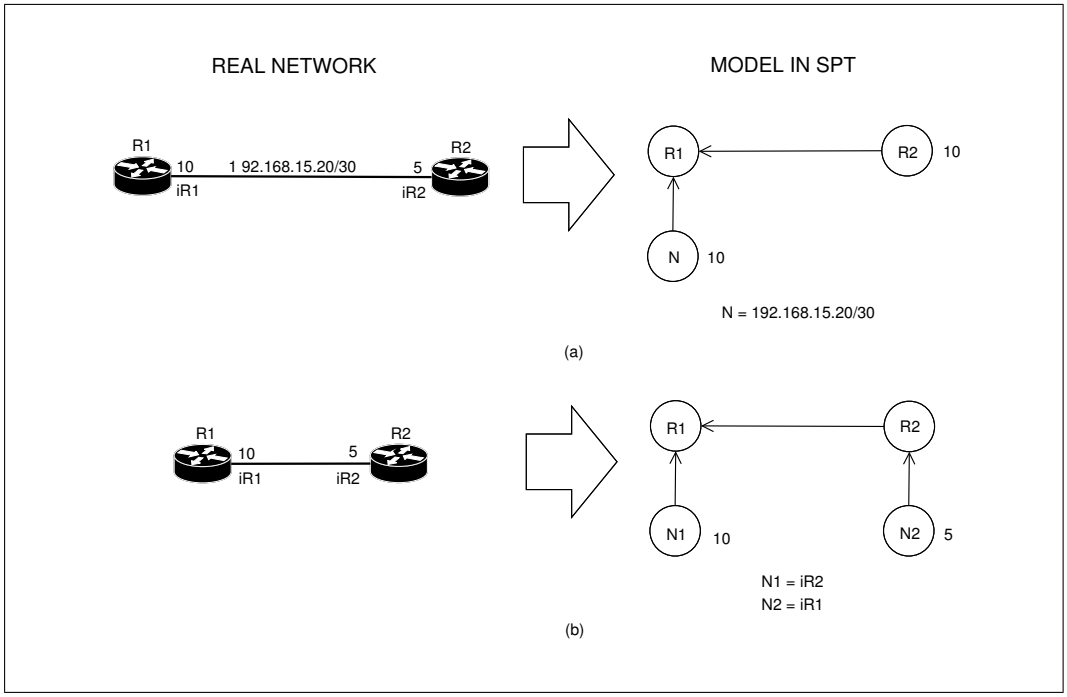

Figure 3.3. Model of prefixes and interfaces on a point to point link.

messages will be compromised. For this reason we have to store a /32 route for all the routers of the domain.

After this step each router has built the routes towards destinations located in areas it belongs to. Border routers and ASBRs in directly reachable areas are also discovered.

3.3.2 Inter-area routes

In the second step the inter-area routes are computed for all the routers in the domain. In figure 3.4 we provide the pseudocode for the inter-area routes computation.

A router R computes inter-area routes by inspecting the routing table of the reachable border routers. Only the border routers belonging to the considered area are inspected in this phase. We consider a border router BR reachable from R when in the R's routing table there is a route towards BR (line 2). Furthermore, the route must have a path type *intra-area*. The tag *area* must also have the value *area(router)*. The function *area()* returns the id of the area a router belongs to. For a border router, we assume that the returned value is the backbone area id. Finally, the destination id field must contain the value *router* 2. These checks respect the requirements described in [3].

The routing table of BR is inspected: only a subset of the routes are considered.

Only routes whose destination should be announced by BR to the router itself (line 4) are taken into account. If this check is passed a new route is computed by R (line 5). The new route's destination is the same as the current route considered in BR. The other fields are filled as explained in section 2.3.

The new route is installed in the routing table of R if no route for the same destination is present. Otherwise, the two routes are compared with respect to the rules explained in chapter 2 and only the best is kept (lines 6-7). If an ECMP is discovered, a new next hop is added to the existing route; the field *Advertising Router* is filled with the *router id* of the router BR.

```
01: procedure build_inter_area_routes(router) {
02:   for each border_router reachable by router in area(router) {
03:     for each route_in_br in routing_table(border_router) {
04:       if (route_is_announced_in_area(route_in_br, area(router)) == TRUE){
05:         new_route = build_inter_area_route(route_in_br)
06:         if (not exist_a_best_route(routing_table(router), new_route)){
07:           install_route(router, new_route)
08:         } //end if
09:       } //end if
10:     }//end for each on route_in_br
11:   }//end for each on border_router
12: }
```

Figure 3.4. Pseudocode to compute inter-area routes for a given router.

The assumption that function *area()* returns the backbone id value for border routers is not without implications. This means that a border router will build its routing table by inspecting only other border routers it can reach inside the backbone. If a border router is reachable only through an area different from the backbone it is not considered by other border routers. This is equivalent to the constraint in OSPF that prevents border routers from considering summaries injected in an area different from the backbone[2].

The procedure described above requires to find a way to select which routes must be considered during the border router's routing table inspection. This is equivalent to modeling the choice of which summary LSAs (or an ASBR summary LSAs) border routers must inject in a given area. We can observe that a border router makes this choice respecting the following rules:

[2]As explained in chapter 2 this behavior can be altered by using virtual links. The modelling of virtual links will be discussed later on.

1. if a destination is discovered by intra-area routes computation in some area, this destination is not announced in that area;

2. if a destination is discovered by receiving a Summary LSA (or ASBR Summary LSA) from an area, this destination is not reannounced in that area.

We have found a way to map these rules in our model. We perform some checks that permit to filter the routes considered in the inspected border router. We call this the *per area filtering* of the border router's routes. This filtering is illustrated in figure 3.5. To map the first rule, when we inspect the border router's routing table, we can simply discard intra-area routes for which the field *area* value corresponds to the *area()* returned value. This is equivalent to not consider routes for destination in the area the router belongs to (the backbone for a border router). We can observe that these destination are already discovered by the router in the first step of routes computation, so that no information is lost during this check. We can make this statement because we are sure that the router can reach the border router in the area (by the check discussed above). So the router can reach all the destinations that the border router can reach in the area.

The second rule is mapped to some additional checks during the inspection of the border router's routing table. However, the checks depend on whether the router that performs the routes computation belongs to the backbone or not. In the last case, the second rule is modelled considering the inter-area routes of the inspected border router. These destinations correspond to destinations discovered by Summary LSA (ASBR Summary LSA) by the border router. This choice has an important consequence: when in our model routers that do not belong to the backbone perform the inter-area routes computation, border routers' routing tables that will be inspected must contain all the available information in the domain. In other words, border routers must compute inter-area routes before internal routers in the area inspect them. This corresponds to the natural flooding of routing information that, due to the role of the backbone, is always flooded into the backbone and only after inside other areas. In our model we map this behavior by computing the inter-area routes for the backbone routers first, and then for all the others.

Now, we have to discuss how the second rule is mapped when the router that performs the computation is a router of the backbone. In that case, the information required to build inter-area routes consists simply in all the intra-area routes of the inspected border router. As a consequence, all the inter-area routes can be ignored. They represent a redundant information that is already considered thanks to the first rule, i.e. considering intra-area routes that do not describe destinations inside the backbone. This is equivalent to model the flooding of routing information about areas different from the backbone performed by border routers[3].

[3]Note that the inter-area routes could also be considered during the border router inspection:

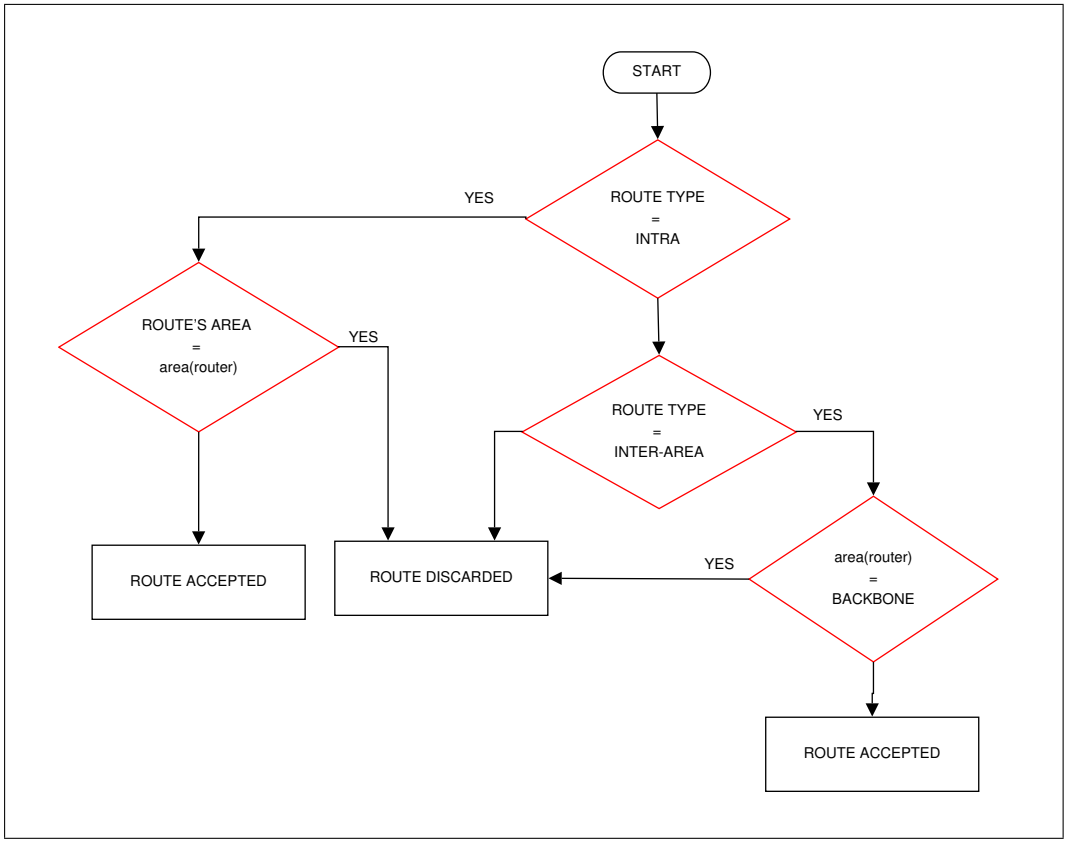

Figure 3.5. Algorithm to select routes during the border router inspection.

To complete our discussion, we would like to show that when a border router computes its inter-area routes all the information it needs is provided exclusively by other border routers. First, all the information about destinations in the domain is injected inside the backbone. As all border routers must belong to the backbone, they only need to inspect other border routers' routing tables to find the information concerning all reachable destinations of the domain. In our model, since the LSA exchange is not implemented, its equivalent is represented by the border routers routing table.

We have to be cautious about the treatment of routes that describe border routers and ASBRs in this step. While the last ones are announced by border routers inside ASBR Summary LSAs, this is not true for border routers whose identity is not announced by a summary. We have to consider this during the inter-area routes computation. When a router finds a route towards a border router this route must

at the end of the computation only the best route will be installed in the routing table. This way to filter the routes is also a simple optimization that allows to improve the routes computation, limiting the amount of routes that have to be considered.

be ignored. When a route towards an ASBR is found on the other hand, it must be considered and the new built route must be tagged with the value *router* 2.

However, we have already discussed the necessity to store in our model the routes towards all the routers of the domain. Even though in the real OSPF routes do not have /32 routes towards all routers of the domain, in our model we need these specific routes as the prefixes configured on subnets (see subsection 3.3.1) do not always contain the loopback of the routers attached to this subnet. Without these more specific routes, some routers would be unreachable by all routers of the domain. This is due only to our mapping of the OSPF router id in the model (see section 3.2.4). For these reasons, routes that describe border routers must be considered and not ignored. However, the computed routes associated to these routers will not be tagged with the *router* value but with the *network* one 2.

Note that to implement this behavior, we need to distinguish route towards border router and routes towards ASBR. Thus the two values *router* and *network* for the route's field *destination type* proposed in [3] are not sufficient in our model. We introduce two different values *BR* and *ASBR* to distinguish the two cases.

At the end of the second main step of the routes building process each router knows all the reachable destinations of the domain. Moreover, all routers have a route towards each reachable border router that lies in their respective areas (already discovered in the first step). They also have routes towards reachable ASBRs that lie in the whole domain.

3.3.3 External domain routes

After intra-area and inter-area routes computation, all the routers have discovered the existence of reachable ASBRs in all areas of the OSPF domain. So now it is possible to compute the routes towards external destinations announced by ASBRs. A possible way to do this is shown in figure 3.6.

All ASBRs of the domain are considered (line 2), but the procedure ignores ASBRs towards which the router has no route (line 3). This route must have a path type *intra area* or *inter area* as requested in [3]. Moreover, the destination id field must have the value *ASBR*. The ASBR's OSPF configuration is checked in order to find links that do not belong to any area (lines 4-5). For each of them a new route is built by the router with the appropriate value for each field, as explained in subsection 2.3.3. We consider them as *type 1 external* routes (line 6).

If no other route is available towards the announced destination, the new route is installed in the routing table. Otherwise, the two routes are compared and the routing table is updated as explained in subsection 2.3.3 (line 7-8).

Since we only consider external destination of *external type 1*, the rules to manage external domain routes explained in 2.3.3 will be simplified.

```
01: procedure build_external_domain_routes(internal_router) {
02:   for each ASBR in domain {
03:     if (ASBR is reachable by router)
04:       for each link in ASBR {
05:         if link does not belong to area {
06:           new_route = build_external_area_route(link);
07:           if (not exist_a_best_route(routing_table(router), new_route))
08:             install_route(router, new_route);
09:         } //end if
10:       } //end for each
11:     } //end if
12:   }//end for each
13: }
```

Figure 3.6. Pseudocode to compute external domain routes for a given router.

3.3.4 Virtual links

Here we discuss the modelling of the advanced features presented in section 2.4. We already discussed about our static approach to design the model. A consequence of this choice is that it is not required to implement a feature like the Synchronization of LSAs Databases. Moreover no LSA database exists in our model: it is modelled by the union of the available domain topology and the OSPF router configuration.

We can think of virtual links as a possibility provided by OSPF to extend the backbone using links that belong to other areas. In our model, virtual links are introduced in the topology through a special type of directed edge. A couple of these special edges, one for each direction, is installed between two routers that have a virtual link configured. These edges can be installed only between two border routers. Moreover the two edges of a virtual link must belong to the same area, different from the backbone. The special edges cannot be used by the forwarding process, but are considered during the SPT computation.

Modifications to the procedure that builds intra-area routes are required in order to assign a correct cost to the virtual link. Especially a cost for each one of the two edges that define the virtual link, must be assigned. The edge from router A (B) to router B (A) must be assigned the cost of the route towards B (A) contained in A's (B's) routing table. If A (B) has no route for B (A) the edge cannot be considered during Dijkstra's algorithm. The computation of intra-area routes thus requires a specified order to be enforced. We cannot compute intra-area routes for routers in the backbone without having computed before all the intra-area routes about the

other areas.

The intra-area routes computation can be summarized as follows: first intra-area routes are computed for all the routers but only for areas different from the backbone. After that, the cost of all the special edges that model virtual links must be set as explained above. Then the computation of intra-area routes can be performed in the backbone. The routes built through virtual links will be treated in the same way as the other ones built starting by physical links. The only difference lies in the next hop information. For routes obtained using physical links the next hop contains the description of a link that belongs to the backbone. For routes built using virtual links the next hop contains a description of a link that does not belong to the backbone. Of course the procedure that computes the next hops for each node must take care of this: the information about the next hops associated to virtual links can be found in the same intra-area route used to set the virtual link's cost.

Once the intra-area routes computation is changed as explained above, there is no other impact on the model. The inter-area routes computation will work as explained in subsection 3.3.2. A border router in the backbone area explores another border router's routing table only if it has a route in the backbone area towards the considered border router. The difference, if virtual link is configured, is that now this route can describe a path that does not belong to the backbone.

Note that the setting of the virtual link cost can be performed during the SPT each time we consider a virtual link. Moreover there is no requirement to install a priori special edge in the topology of the model. Virtual links can be modelled by adding information to border routers. For instance, special edges can be installed in the topology. These edges would be installed right after the intra-area routes computation. The cost for the special edges can also be set at this time. After the intra-area routes computation has finished inside the backbone (considering the virtual links), the special edges can be removed from the model as they are not used further. This approach would also have the advantage to permit the special edges to be considered in the model only when they are actually used during the routes computation.

3.3.5 Address summarization

Address summarization can be modelled by adding to border routers information about prefixes that are summarized. For each area a router belongs to, a list of prefixes can be added. If this happens summarization is activated for the border router in the given area.

To model this feature we have to introduce an additional step during the inter-area routes computation. Additional checks to select a route in the inspected border router's routing table must be added. We call this new filtering a *summarization*

filtering. It follows the *per-area filtering*: if the route is considered, i.e. the destination should be announced in a given area, we have to check if address summarizing is configured for the area. If this is the case we have to check also if the route's prefix is part of one of the summarized prefixes associated to the area. If this is verified the routes should be ignored. However its cost is associated to the summarizing prefix. When a new route will pass the same check for the same prefix the cost associated to it is updated only if it is worse than the previous one. Finally, at the end of the inter-area routes computation, the routes for each summarized prefix must be added. The cost for each of these routes results from the previous computation: it is the worst cost among all the routes comprised in the considered prefix.

If for a given summarized prefix no route is announced then no route will be built for it.

Note that if the route in the inspected border router's routing table describes an ASBR, an inter-area route to the ASBR must be also built if the ASBR is part of the summarizing prefixes.

3.3.6 Stub areas

To model stub areas we need to add information inside border routers, about the fact that an area is configured as a stub. A modification to the inter-area routes computation algorithm is also required in order to model the fact that ASBR Summary LSA and ASBR External LSA are not injected in stub areas. When a route is considered in the inspected border router's routing table, we have to check if it describes an ASBR. If so, we check if the current area is configured as a stub area. If it is the case, the route is ignored. We call this additional filtering the *stub area filtering* to distinguish it from the previous ones. With the *stub area filtering*, no ASBR will be reachable from stub areas. This obviously impacts the external domain routes computation.

As already mentioned in chapter 2, when an area is configured as stub, a default route is injected by border routers. We thus need to model the injection of this default route inside stub areas. We introduce a new substep at the end of the external domain routes computation. After the routes are computed, we consider each border router: if it has an area configured as stub, we inspect the internal routers of the given area. In their routing tables we install the default route. The cost (next hop) of the default route is the cost (next hop) of the considered route towards the border router.

3.3.7 Putting it all toghether

We try to summarize all the steps discussed above to show that solving each step of the computation is consistent with the way OSPF builds its routes. We illustrate

the entire process in figure 3.7.

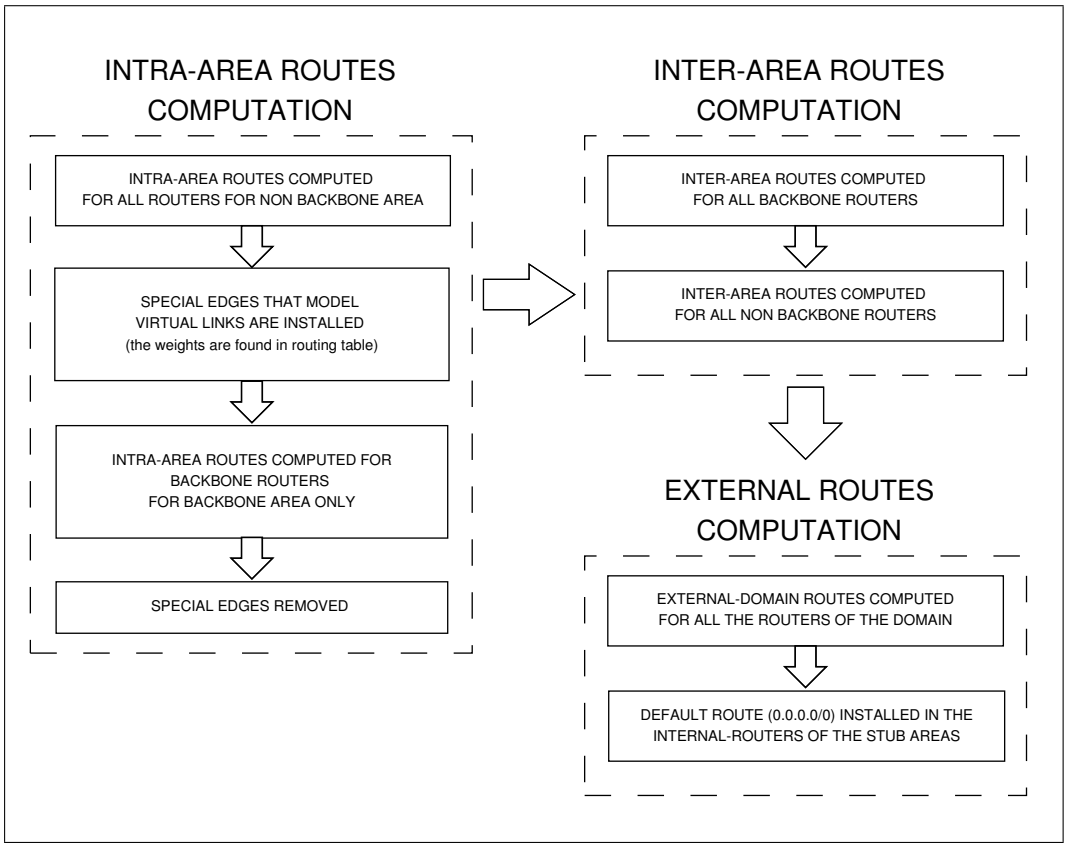

Figure 3.7. A complete representation of the model.

First, intra-area routes are computed for all the routers. Each router computes the SPT and builds the routes for each area it belongs to. The backbone area is skipped. This permits to discover the cost associated to each virtual link. Then special edges that emulate the virtual links are installed in the topology and the virtual link cost is associated with the. The cost of the virtual link from a router BR1 to a router BR2 in an area i is found in BR1's route towards BR2 with area field filled with value i. If the route is not available the virtual link is removed. The computation of intra-area routes for the backbone area is then performed. During this step virtual links are considered. Then the special edges for virtual links are removed. After this step all routers know directly reachable destinations.

Inter-area routes are computed for backbone routers first. During this step the routing tables of the border routers are filled with the available routes they can discover. After, inter-area routes are computed for all the non-backbone routers that inherited the information discovered by border routers in the previous steps. In every case, inter-area routes are computed from a router inspecting the routing

table of all the reachable border routers. The border routers are discovered in the previous step and routes that describe them are tagged in a special way. From the routing table of the border router only a subset of the available routes are considered. The selection process of the routes is illustrated in figure 3.8.

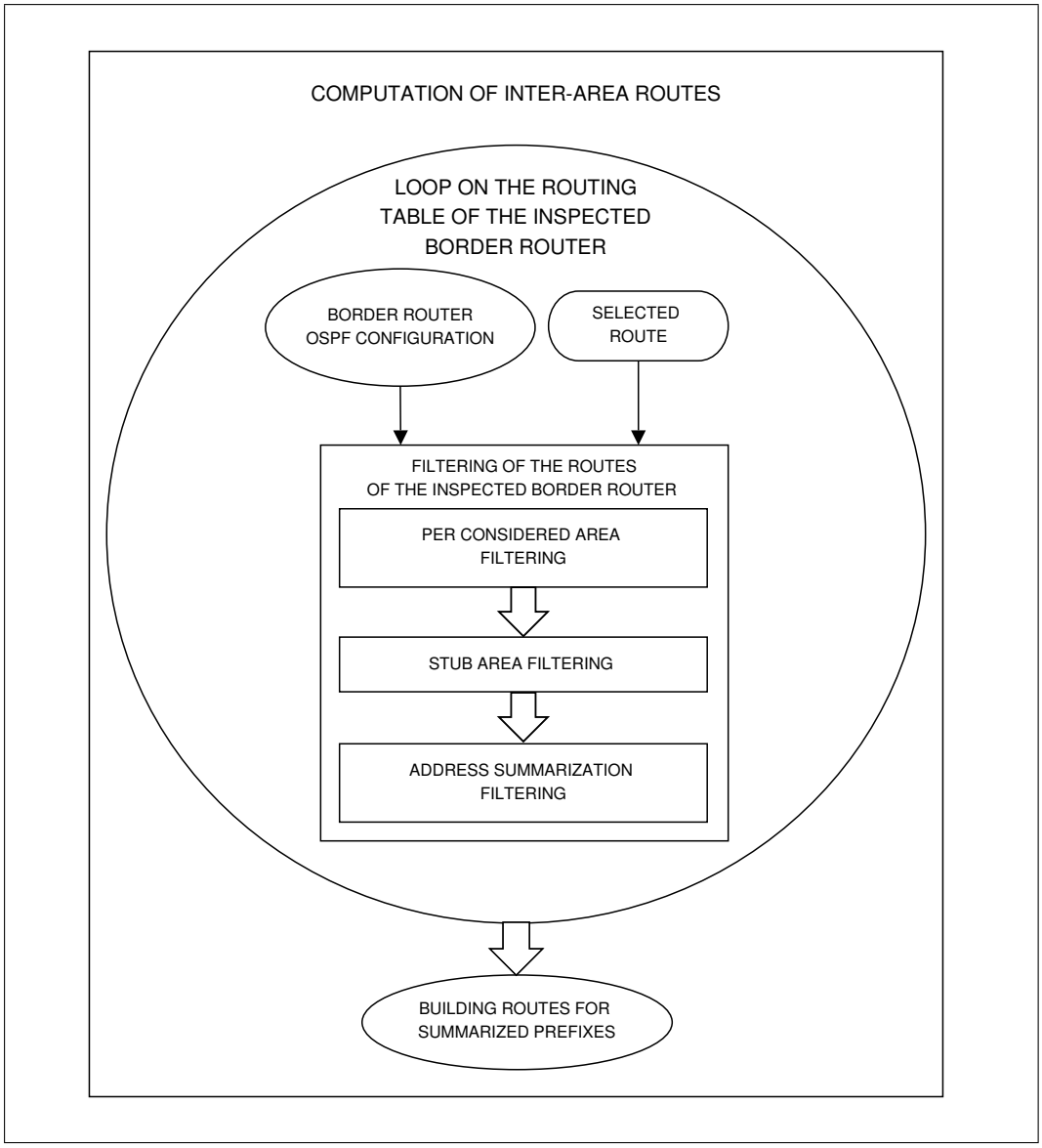

Figure 3.8. Inspection of the reachable border router's routing table.

Given a route in the routing table of the inspected border router, we apply the *per area filtering*. Routers discard routes that describe directly reachable destinations. Moreover, if the router is a border router it ignores also the inter-area routes.

Then the *stub area filtering* is applied on the routes that pass the previous check. We check if the inspected border router is configured to treat the given area as stub. If this is the case and if the considered route is an ASBR External LSA or an ASBR Summary LSA, it is ignored.

If the route passes the previous checks, we apply the *address summarization filtering*. We check if the considered border router is configured to perform address summarization. If so, we check if the considered route's destination address belongs to one of the configured summarized prefixes. If this is the case, the route is ignored but its cost is used to compute the cost of the summarized prefixes.

Finally, we build a new route for the given router. It has a cost that is the sum of the cost of the path towards the considered border router and the cost of the selected route. At the end of the inter-area routes computation, we check if there are some summarized prefixes that are announced. If this is the case, we build a route towards each of them associating a cost calculated during the *address summarization filtering*.

Finally, routes for external destinations are built. A router builds these routes examining the links of the reachable ASBRs, discovered during the previous steps. For each link, the router builds a route for the destination of the link. The cost is the sum of the cost of the link plus the cost the router has to reach the ASBR. However, if an area is configured as stub area, no external domain route is installed because no ASBR will be reachable in the area. To fully emulate the OSPF stub area behavior, a default route (`0.0.0.0/0`) is installed in each router of the stub areas. The route points towards the border router of the area, and the next hop is the same as the one for the border router.

3.4 Conclusions

In this chapter we have discussed our model for the OSPF protocol. First we have presented the C-BGP simulator. Understanding the way C-BGP works was foundamental in order to design a model able to provide the correct IGP routing tables. Moreover, these are important to provide information (the cost of the IGP routes) to the BGP decision process.

Then we have illustrated the assumptions we made to design our model. The router id is always mapped to the loopback interface of the C-BGP router. Contrary to the behavior of OSPF, we have to store in C-BGP a route for each router of the domain. We assumed we can model LISs in the domain. We provide only a limited support for annoucing external routes in the domain: only destinations immediately adjacent to the ASBR are announced like *external type 1*. Finally, the Hello protocol was mapped in the model by assuming a correct configuration of OSPF and performing the two way connectivity check during the building of the

SPT for a given area.

To perform the routes computation in an efficient way we have emulated the LSA exchange. We replaced Router LSAs and Summary LSAs with the topology model of C-BGP. We found a way to limit the visibility of routers to the areas they belong to. Summary LSAs and ASBR Summary LSAs were mapped to the inspection of the border routers' routing tables. The ASBR External LSAs were mapped to the inspection of the ASBR links. Finally, we have discussed in details the algorithm to build the routes, showing its correctness. We have summarized our design choices with their justifications in table 3.4.

OSPF	Model	Justifications
Hello protocol	Checks during OSPF configuration and during the SPT computation	Check during configuration guarantees that only valid adjacencies are built. Checks during the SPT computation correspond to the two way connectivity check.
Router LSA and Network LSA	C-BGP topology model and limitation of router's view to the areas they belong to.	The information provided by these LSAs correspond to a map of the area where they are exchanged. This is already available in C-BGP's AS topology model.
Summary and ASBR Summary LSA	Inspection of border routers' routing tables	The information announced by border routers in Summary LSA and ASBR Summary LSA can be found in the border router's routing tables. It consists in the cost of the border router towards the destination. Filtering is required to select the right routes inside border routers.
ASBR External LSA	Inspection of ASBRs' links	We model only *type 1 external* destinations, to allow reachability of BGP next hops in C-BGP.
Virtual links	Special edges in the topology and computation of intra area routes into two steps.	Virtual links allow to use non backbone links as belonging to the backbone. Special edges are used only during the backbone SPT computation.
Address Summarization	Address summarization filtering	The summarization consists in not announcing some destination. This is done with appropriate checks during the inspection of border routers in order to select the right routes.
Stub area	Stub area filtering and installation of default routes	The stub area is reproduced preventing the discovery of ASBRs. This is done with a check during the inspection of the border routers' routes. Moreover the default routes are installed in internal routers of stub areas.

Table 3.1. Mapping between the OSPF specifics and our model.

Chapter 4

Implementation of the OSPF model

We have implemented a prototype of the model presented in the previous chapter. It is available at [2]. We chose to implement the basic functions of the model (intra-area and inter-area routes computation) in order to have a working implementation as early as possible and to use it to study some aspects of multi-area OSPF. A complete implementation of the model will be available soon.

Before implementing the model, we made some modifications to the topology representation in C-BGP. This was necessary to support LISs (see section 3.2.2). This modification is illustrated before reporting some details about the implementation choices. We focus our attention on two critical aspects of the implementation (from a performance viewpoint), and propose improvements to the current way C-BGP works.

Finally a validation of the model is proposed. Two scenarios are considered and the routing table of some routers in the considered networks are examined to show that our computation of the routes is consistent with the OSPF specification [3].

4.1 Implementing the model

Before implementing the OSPF model in C-BGP, we have introduced some modifications to the topology model in order to represent LISs. Here we discuss how these modifications have been made. Then we show how multi-area OSPF is introduced in C-BGP. Finally, the integration of the OSPF model inside C-BGP is discussed.

4.1.1 Modelling Local IP Subnets

In our model, we have assumed that Local IP Subnets are supported in the model. However, C-BGP models only routers, not LISs. Besides the /32 prefixes corresponding to the loopback of the routers, the only IP prefixes injected in the domain are those injected at the border routers (typically collected from a real network).

We model the LISs with two modifications in C-BGP's topology model. First a new class of objects was introduced: the *subnet* object. This models the subnet of a network, without taking into account the physical details of the subnet below the IP layer. The information associated to this object consists in the prefix of the LIS configured on the subnet. We distinguish beetwen transit networks (subnets on which there are more than one router) and stub networks (subnets on which there is a single router). First transit networks are implemented in the graph topology as special nodes. Then stub networks are modelled with a special node. The connections between the special nodes representing the networks (transit and stub) are made as already explained in section 2.1.3.

To correctly compute the SPT, the edge from a subnet to a router has always a fixed cost of zero [3]. Table 4.1.1 contains in rows 1 and 2, the commands introduced into the CLI in order to manage subnets.

The second modification we have introduced is on the point to point links, in order to model situations in which LIS are configured on them. On a point to point link, we can only have IP addresses for the interfaces that are on the link, but also a prefix (see chapter 2). To model the first situation we have added the possibility to configure interfaces on the link. We do that with the CLI command reported in row 3 of table 4.1.1. We invoke this command on an already existing link by specifying a `prefix`. If this is a /32 prefix, only the IP addresses for the interfaces are configured while no LIS is set on the link. To configure a LIS on the link we have to pass a /30 prefix as parameter of the command. In this way a flag is set on the link and we consider that a LIS is configured on it. Note that, in each case, this command modifies a link only in one direction.

We want to conclude our discussion on the implementation of the LISs by insisting on the fact that to model LISs it was sufficient to introduce only the interfaces and the prefixes on the point-to-point link. Subnets can be modelled using virtual links only: a subnet is represented as a full mesh among the routers and the subnet object itself. However, doing this has a complexity of N^2 in terms of the required edges, where N is the number of nodes on the subnet. The introduction of the subnet class permits to reduce this complexity to N. The complexity is not only in terms of the amount of memory required but also in terms of the Dijkstra algorithm computation since the number of links to process is reduced. Moreover, this simplification in the topology also simplifies the building of the network since less commands are required to design a model of a given network.

CLI command	Description
`net add subnet <prefix> <transit|stub>`	Adds a subnet in the topology model. Assigns the given `prefix` to the subnet. Declares the subnet as `transit` or `stub`.
`net add link <ip-address> <ip-prefix> <weight>`	Adds a link between the router identified by `ip-address` and the subnet identified by `prefix` with a weight specified by `<weight>`. The prefix must specify a 32 bit IP address length that is used as IP interface on the link.
`net link <ip-address-src> <ip-address-dest> ipprefix <ip-prefix>`	Adds interfaces on the link between the routers identified by the IP addresses `<ip-address-src>` `<ip-address-dest>`. If the given `<ip-prefix>` is a `/32` prefix only an interface is added on the link. Otherwise the `<ip-prefix>` can be only a /30 prefix and, in this case, also a LIS is configured on the link. In each case modifications are performed only on the interface of `<ip-address-src>`. The IP address `<ip-interface-src>` is assigned to the interface of node `<ip-address-src>` while the `<ip-interface-dst>` is assigned to the interface of node `<ip-address-dest>`.

Table 4.1. CLI command to manage subnets.

4.1.2 OSPF configuration

We discuss the way we require the user to configure the multi-area OSPF scenario.

Firstly we assign each router to a set of areas. We do not require to declare IDs of the areas. Assigning a router to some areas makes possible to implement one of the checks performed by the Hello protocol as explained in 3.2.4. We use this information (the fact that a router belongs to some area) when a link is assigned to a specified area. For this reason the user **must** assign the router before and then the links the areas. From this viewpoint subnets are treated as a router: they must be assigned to a single area to perform the Hello protocol's check and must be assigned before the links assignation. Rows 1 and 2 of table 4.1.2 reports the commands used to assign a router or a subnet to a given area.

We have to be able to assign each link to a given area. To obtain this we have added a field to the link class that contains the ID of the area a link belongs to. If a link does not belong to any area, a special value `OSPF_NO_AREA` is used. By default, when a new link is introduced in the model, it does not belong to any area. It is

possible to set the area a link belongs to with the command in table 4.1.2, row 3.

Finally we require that each node that participates in the OSPF domain belongs to a specified IGP domain in the network. We added the class of objects IGP domain and the command on row 4 in table 4.1.2 to add a router to a given domain.

Starting from the information provided by the OSPF configuration command, we are able to identify an internal router (a router that belongs at most to a single area), a border router (a router that belongs at least to two areas), and an ASBR (a router that has at least one interface that does not belong to any area). For simplicity and efficiency we require to consider an ASBR as active, i.e. it sends ASBR External LSAs, the user must explicitly declare it as an ASBR that redistributes external routes.

The aim of C-BGP is also to model in an efficient way large topologies, with more than one AS. For this reason, we have made the effort to provide the entire OSPF model as an option that can be selected at compile-time. This means that if the user does not need OSPF he can choose to compile C-BGP without the OSPF support. In this way the topology object like the nodes and the links will be smaller in terms of memory and useless fields for the OSPF information will not be allocated. If the OSPF model is not compiled, the OSPF commands will not be available in the CLI. The OSPF support can be obtained compiling C-BGP with the `ospf-support` option.

4.1.3 IGP model and routes computation

Here we discuss the integration of our model with the current C-BGP. We have chosen not to replace the simple IGP model already present in C-BGP. Although the same simple model can be obtained by configuring our OSPF model to work with a single area, doing this can be inefficient to model a single-area domain. The OSPF model requires some additional information on link and routers that are not useful in the computation with the single area architecture. In the same way, most checks performed during the routes computation are useless. Another good reason to keep the simple model is that it is already part of C-BGP and works well. So, we have considered that it will be more useful to have the possibility to choose which model to use based on the required level of details. The simple IGP model can be utilized to efficiently model scenarios where it is not interesting to model the multi-area aspects.

For what concerns the data structure required by the OSPF routes computation, we needed a more complex routing table than the one already available in C-BGP. As illustrated in section 2.2.4, more fields are required for each routing table entry in order to perform several checks during the OSPF routes computation. Moreover, the C-BGP routing table does not support ECMPs. C-BGP's routing table is simple in order to limit the amount of memory space required to store all the BGP routes

CLI command	Description
`net node <ip-address> ospf area <area-id>`	Assigns the router identified by the IP address `<ip-address>` to the area `<area-id>`. A router may belong to multiple areas.
`net subnet <ip-prefix> ospf area <area-id>`	Assigns the subnet identified by the IP address `<ip-prefix>` to the area `<area-id>`.
`net node <ip-address-src> link <ip-prefix-dst> ospf area <area-id>`	Assigns the link from the source router (identified by the IP address `<ip-address-src>`) to the destination node (identified by the prefix `<ip-prefix-dest>`) to the area `<area-id>`. The destination can be a router or a subnet. In the first case the prefix is a /32 prefix.
`net add domain <igp-domain-id> <igp|ospf>`	Declares a new IGP domain in the network. The domain is identified by the `<IGP-domain-id>`. The type of the domain can be `igp` ot `ospf`. In the first case a model with a single area is used. In the second case the ospf model is used.
`net node <ip-address> ospf domain <igp-domain-id>`	Assigns the node identified by the address `<ip-address>` to a given IGP domain identified by `<igp-domain-id>`. The domain must be declared in the network. If the domain is of type `igp`, the previous model of C-BGP for the domain is used. Otherwise if the value `ospf` is specified our model is used.

Table 4.2. CLI commands to configure the multi-area OSPF domain.

that can contain between 150.000 and 500.000 prefixes. Due to this constraint, a new and independent routing table was added and the computation of the OSPF routes is completely distinct from the BGP routes' computation. The OSPF routing table is used to compute the OSPF routes only.

Once the topology model is built and the OSPF multi-area architecture is defined, the OSPF routes' calculation can be performed. The user can do it by invoking the command `net domain <domain-id> compute`. If the given `<domain-id>` identifies an IGP domain that uses the old IGP domain model the old computation of the routes is performed. Otherwise, if the `<domain-id>` identifies an OSPF domain, our

computation of the routes, described in chapter 3, is performed. Then, the OSPF routing table of all the routers of the domain will contain the OSPF routes for the given domain.

If the OSPF routes must be used to exchange the BGP message they must be installed in the forwarding table of the router. Some information used during the OSPF routes computation will be lost and only a single path will be stored in the table in the case ECMPs were discovered during the OSPF computation. This installation of the computed routes will be available soon. It will be done by the command `net domain <igp-domain-id> install-routes <destroy|no-destroy>`. The option `destroy` permits to remove the OSPF routing tables from the memory. The `no-destroy` option permits to maintain them if the IGP routing aspect must be studied in details.

4.2 Details of the implementation and possible improvements

Here we discuss some details of the SPT computation. Our model uses an implementation of Dijkstra's algorithm that is different from the one used in the older IGP model of C-BGP. Introducing the subnets, we have also put some effort to have a harmonious integration with the router object in the SPT computation. We then focus on some important points in the routes computation and some possible alternatives to improve the current implementation are provided.

4.2.1 SPT computation details

The addition of the subnets to the topology model was performed by giving particular attention to a good integration with other network objects like routers and links. A new implementation of the Dijkstra algorithm was made in order to manage the different classes of objects (routers and subnets) in a homogeneous way. For this purpose, we have introduced a superclass *spt_vertex* that generalizes the *node* class (that represent a router) and the *subnet* class that represents the subnets. The vertex class generalizes the topological properties of the two classes: routers and subnets can be viewed as vertices of a graph connected by edges that represent a link. Methods to manage the edges are provided in the *spt_vertex* class. Thus this class represents only the topological details that router and subnet classes have in common. Information required in the SPT computation, like the current cost computed for each vertex is also stored in the new class.

The *router* class extends the *spt_vertex* class by adding information that concern all the router's features, like the capability to manage a routing table or to model the OSPF configuration parameters (for instance the areas a router belongs to or the

fact that it is an ASBR). An IP address is also stored and used as id of the router. The subnet class, instead, extends the vertex class by adding the prefix associated to the subnet and also the information about the OSPF configuration (the area a subnet belongs to).

During the SPT computation router and subnet objects are managed in the code as a *spt_vertex* object. The method that performs the SPT computation sees the SPT as composed of *spt_vertex* objects. The computation of the intra-area routes, as explained in chapter 2, is done by visiting the SPT. Here some information about the OSPF configuration is required: for instance we need to know if a router is a border router or not. Here a polymorphic behavior is realized and the correct methods are invoked on each *spt_vertex* object instance.

4.2.2 Possible improvements

The current implementation of the SPT computation is straightforward. The only optimization we made is to treat the stub subnets in a different way. When we discover a stub subnet we keep a reference about it but we do not examine it in order to discover other vertices in the graph. Since it has a stub subnet the unique router on it was already discovered before the subnet itself. This avoids a useless elaboration of the stub subnet.

Several optimized implementations of the algorithm are proposed in the literature [6]. The multi-area topology should allow to assume that the size of the graph on which the SPT is computed is smaller than the size of the whole network. Thus this part of the routes' computation should not be critical and the contribution of a very optimized implementation should not be too significant. However, if the model is used in scenarios where the size of the area becomes large the current implementation could be replaced. Specifically, the critical aspect of the SPT computation is represented by the selection of the next node to be added to the SPT. This is the node with the smallest cost among those already examined. An implementation with priority queues or Fibonacci heaps might significantly reduce the complexity of this part.

For what concerns the inter-area computation a potential improvement would be the selection of the inspected border router. When inter-area routes are computed for a router, the reachable border router must be considered as explained in chapter 3. To implement this we scan the routing table of the router to find the routes that describe a border router. However, this can be expensive since a full scan of the router's routing table must be executed, while the routes that describe border routers are few compared to the whole amount of routes. This represents a critical point of the implementation that can be improved.

A possible way to obtain a faster selection of the reachable border router can be as follows. We assume that we know the border routers of a given area. Starting

from this set, for each router of the area we check that the considered border router is reachable with an appropriate route. In the actual implementation the selection of the border router has a cost of $O(N)$ where N is the number of routes in the routing table. In the worst case this corresponds to all the destinations of the modelled network. In the second case, the cost of the selection of the border router is $O(1)$ since we assume that given the address of the destination the access to the routing table is constant[1]. Having the set of border routers of a given area strongly improves the computation of the inter-area routes.

The set of border routers of an area can be provided in two ways, using either a soft-state or a hard-state approach. In the first case, all the routers and only those belonging to a given area are considered and added to a special list. With the second approach, we assume to have a class of objects that describe a given area of the OSPF domain. This class contains also a list of references to the border routers of this area. The list of references can be populated on the fly during the configuration phase of the OSPF domain or before starting the computation of the routes with the soft-state mechanism described above.

Finally, we can modify our implementation in order to obtain a flexible behavior of the model. If the model must be used to study the OSPF behavior only (and not to provide IGP routes to the BGP simulation in C-BGP) the user might set an option that allows our model to fully emulate the OSPF protocol. We call this option of the model `full-ospf`. Enabling the full OSPF model has two consequences. Firstly the OSPF router id is mapped as specified in [3]. Thus it corresponds to the smallest IP interface configured on the router, otherwise, if no interface are configured, the ospf-id corresponds to the loopback of the C-BGP routers. Secondly, we do not store a route for each router in the domain as required actually. A route for a router is stored only if it has no interfaces configured on it. Note that in this way we have no guarantee that the forwarding will work well in C-BGP. This will impact the exchange of all routing messages inside C-BGP during the BGP simulations. However we have assumed that when the `full-ospf` option is activated we are not interested in the BGP simulation.

4.3 Validation of the model - Scenario 1

We are not aware of the existence of an OSPF domain for which the configuration and topological information are publicly available. Thus we are not able to perform a validation on a real network comparing the output of our model with the real routers' routing tables. The developed prototype was used to compute the OSPF routing tables of the routers of a sample network. The output provided by this

[1]This happens in our implementation since we have used a radix tree provided in the *libgds* [2] library.

computation is discussed in order to show that the behavior of the model matches the OSPF RFC [3].

4.3.1 Outline of the validation

The network on figure 4.1 is used as a case-study to check the behavior of our model. The domain is split in four areas. For each router we plot the IP of the loopback interface, used by C-BGP to identify a router. In the same way, near each subnet we write the IP prefix assigned to it. For each link the weight is reported. Moreover, if an IP address is configured on an interface it is reported near the link in the form $.X$ where X is the host part of the IP address, while the network part corresponds to the prefix assigned to the link.

In the given scenario no virtual link is configured on area 2 on border router `192.168.32.11`. With this situation we want to show that our model reproduces the behavior of OSPF described in [3]. In a real scenario, when router `192.168.0.11` realizes that it belongs to two areas, it knows itself to be a border router. As a consequence, it announces destinations of area 2 inside area 3 and viceversa. However, the border routers on area 2 (`192.168.0.7` and `192.168.0.10`) do not consider summaries injected by `192.168.0.11` about destinations in area 3. Destinations in area 3 are unreachable by other routers in area 1 and in the backbone. In the same way, border router `192.168.0.11` does not consider summaries injected in area 2 and destinations in area 3 are not able to reach the backbone nor area 1. In the next subsections we show that our model is consistent with this behavior[2].

The tests are performed with some scripts written for the Command Line Interface (CLI) of C-BGP. Appendix A contains the main scripts used to perform this validation and the information to run it. For more details about the CLI of C-BGP we refer to [2].

We assume that all the links are up in the given scenario. The output of the script consists in all the routing tables of the routers of the studied network. We consider only some of them due to space limitations. We examine a few routing tables for each area. By re-running the script the reader can check that all the routers of each area have routing tables consistent with the ones considered here.

4.3.2 Routes' computation inside area 1

The routing table of router `192.168.16.1` is shown in figure 4.3.2. The field `T` corresponds to the field *destination type*; the field `DESTINATION ID` contains the IP prefix that identifies the destination; the field `PATH` assume the valule `INTRA` for

[2]Situations similar to this one are used by operators as discussed in [12]. [12] propose changes in the inter-area routes computation useful for this situation that we have not considered in our model.

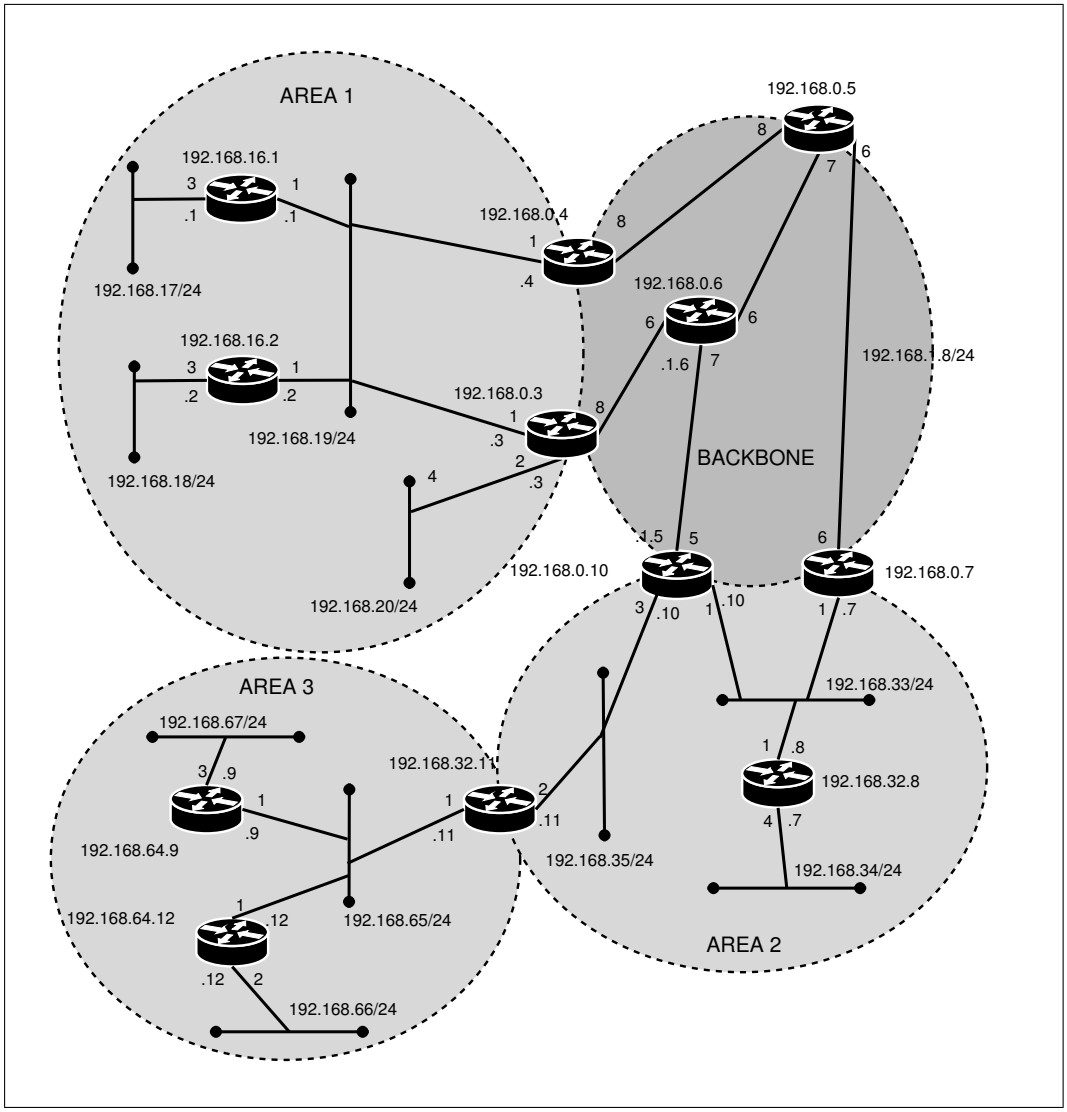

Figure 4.1. Scenario 1

intra-area routes or `INTER` for inter-area routes; the field `A` store the area from which the route was learned; the field `C` contains the cost of the route. After the vertical bar `|` we have the information about the set of paths, i.e. the next hops associated to the network. The filed `INTERFACE` contains the IP of the interface on which a packet for the route's destination will be forwarded; the field `NEXT HOP` contains the IP of the router to which the packet will be sent (it is significative only if the link is towards a subnet); finally the field `ADV` contains the router id of the Advertising Router and it has has no meaning for intra-area routes.

We can verify that the routing table of router `192.168.16.1` has intra-area routes

only for the routers and the subnets that belong to area 1. Among these, only the routes for routers `192.168.0.3` and `192.168.0.4` are tagged with the value *router* (R). This is correct since they are border routers. The route for router `192.168.16.2` should not be present in the routing table since it has an interface configured. The route is however computed due to the requirement in C-BGP to store routes for all the routers of the domain. Note that it is correctly tagged with a value `network` (`N`) for the field *destination type* (`T`).

The reader can verify that the cost for each intra-area route is correctly computed and corresponds to the shortest path each router has to reach the intra-area destination. The field *interface* shows the link used to forward packets for the given destination. The next hop field shows the node to which a packet is delivered on a given network. To reach network `192.168.18.0/24`, for instance, the packet must be forwarded to the node `192.168.16.2` using the interface `192.168.18.2` as next hop.

The examined router can reach external area destinations through border routers `192.168.0.3` and `192.168.0.4`. We can observe that all the routers and the network external to the area, have an entry in their routing table except for the destinations in area 3. This is consistent with the behavior of the border router `192.168.32.11` discussed above.

We can see that, an inter-area route for each router that is outside the considered area is stored. Note that the routes for border routers external to the area are not tagged with the value *router*. This is correct since the routers inside area 1 do not know about the existence of border routers outside this area.

The costs computed for the inter-area routes are also correct. For instance, router `192.168.0.5` is reached by the router using `192.168.0.4` as exit router. The cost is the sum of the cost to reach the border router plus the cost the border router has towards the destination. The former can be found in the routing table of `192.168.0.1`. The latter can be found in the routing table of the border router `192.168.0.4` shown in figure 4.3.2. Note that the best path towards `192.168.0.4` is chosen by the router: the path through the other border router `192.168.0.3` has a larger cost.

Finally, we can observe that the router has discovered two ECMPs for destination `192.168.32.11`. The first is through border router `192.168.32.3`, the other is through `192.168.32.4`. In the same way, two ECMPs are discovered towards the LIS `192.168.35.0/24`. By looking at the routing tables of the border routers `192.168.0.3` (figure 4.3.3), `192.168.0.4` (figure 4.3.3) and `192.168.16.1` (figure 4.3.2), the reader can verify that the ECMPs are valid.

```
*******************************************************************************
** 192.168.16.1 OSPF routing table
*******************************************************************************
-------------------------------------------------------------------------------
 T  DESTINATION ID   PATH    A   C  | INTERFACE     NEXT HOP      ADV
-------------------------------------------------------------------------------
 R  192.168.0.3/32   INTRA   1   1  | 192.168.19.1  192.168.19.3  --
 R  192.168.0.4/32   INTRA   1   1  | 192.168.19.1  192.168.19.4  --
 N  192.168.16.2/32  INTRA   1   1  | 192.168.19.1  192.168.19.2  --
 N  192.168.17.0/24  INTRA   1   3  | 192.168.17.1  --            --
 N  192.168.18.0/24  INTRA   1   4  | 192.168.19.1  192.168.19.2  --
 N  192.168.19.0/24  INTRA   1   1  | 192.168.19.1  --            --
 N  192.168.20.0/24  INTRA   1   3  | 192.168.19.1  192.168.19.3  --
-------------------------------------------------------------------------------
 N  192.168.0.5/32   INTER   1   9  | 192.168.19.1  192.168.19.4  192.168.0.4
 N  192.168.0.6/32   INTER   1   9  | 192.168.19.1  192.168.19.3  192.168.0.3
 N  192.168.0.7/32   INTER   1  15  | 192.168.19.1  192.168.19.4  192.168.0.4
 N  192.168.0.10/32  INTER   1  16  | 192.168.19.1  192.168.19.3  192.168.0.3
 N  192.168.1.5/32   INTER   1  16  | 192.168.19.1  192.168.19.3  192.168.0.3
 N  192.168.1.6/32   INTER   1  21  | 192.168.19.1  192.168.19.3  192.168.0.3
 N  192.168.1.8/30   INTER   1  15  | 192.168.19.1  192.168.19.4  192.168.0.4
 N  192.168.32.8/32  INTER   1  16  | 192.168.19.1  192.168.19.4  192.168.0.4
 N  192.168.32.11/32 INTER   1  19  | 192.168.19.1  192.168.19.3  192.168.0.3
                                    | 192.168.19.1  192.168.19.4  192.168.0.4
 N  192.168.33.0/24  INTER   1  16  | 192.168.19.1  192.168.19.4  192.168.0.4
 N  192.168.34.0/24  INTER   1  20  | 192.168.19.1  192.168.19.4  192.168.0.4
 N  192.168.35.0/24  INTER   1  19  | 192.168.19.1  192.168.19.3  192.168.0.3
                                    | 192.168.19.1  192.168.19.4  192.168.0.4
-------------------------------------------------------------------------------
```

Figure 4.2. Routing table of the router `192.168.16.1`

4.3.3 Routes' computation inside the backbone

Figure 4.3.3 contains the routing table of router `192.168.0.6`. As required, intra-area routes describe only destinations belonging to the backbone, while other destinations are reached by non intra-area routes. The reader can verify that the router computes the intra-area routes chosing the best path towards each backbone destination. The only router that is not a border router is `192.168.0.5`: it is reached by a route correctly tagged with a value *network* for the field *destination type* (the other routers in the backbone are all border routers, hence are tagged with the *router* value).

Among the intra-area routes we can find `192.168.1.5/32` and `192.168.1.6/32` that correspond to the interfaces configured on the link between router `192.168.0.6/32` and `192.168.0.10/32`. The costs are consistent with the way OSPF models the presence of only interfaces configured on a point to point link. The route towards the interface `192.168.1.5/32` has a cost of 7 since it is represented in the graph of the adjacencies as a stub network linked to the router itself and the cost corresponds to the cost of the considered point to point link (see section 2.2.2). The route for `192.168.1.6/32`, instead, has a cost of 12: the interface is modeled as a stub subnet linked to the node `192.168.0.10/32` (see section 2.2.2) thus the cost to reach it is

```
*******************************************************************************
** 192.168.0.6 OSPF routing table
*******************************************************************************
-------------------------------------------------------------------------------
 T  DESTINATION ID    PATH   A   C   | INTERFACE    NEXT HOP   ADV
-------------------------------------------------------------------------------
 R  192.168.0.3/32    INTRA  B   6   | 192.168.0.3  --         --
 R  192.168.0.4/32    INTRA  B  14   | 192.168.0.5  --         --
 N  192.168.0.5/32    INTRA  B   6   | 192.168.0.5  --         --
 R  192.168.0.7/32    INTRA  B  12   | 192.168.0.5  --         --
 R  192.168.0.10/32   INTRA  B   7   | 192.168.1.6  --         --
 N  192.168.1.5/32    INTRA  B   7   | 192.168.1.6  --         --
 N  192.168.1.6/32    INTRA  B  12   | 192.168.1.6  --         --
 N  192.168.1.8/30    INTRA  B  12   | 192.168.0.5  --         --
-------------------------------------------------------------------------------
 N  192.168.16.1/32   INTER  B   7   | 192.168.0.3  --         192.168.0.3
 N  192.168.16.2/32   INTER  B   7   | 192.168.0.3  --         192.168.0.3
 N  192.168.17.0/24   INTER  B  10   | 192.168.0.3  --         192.168.0.3
 N  192.168.18.0/24   INTER  B  10   | 192.168.0.3  --         192.168.0.3
 N  192.168.19.0/24   INTER  B   7   | 192.168.0.3  --         192.168.0.3
 N  192.168.20.0/24   INTER  B   8   | 192.168.0.3  --         192.168.0.3
 N  192.168.32.8/32   INTER  B   8   | 192.168.1.6  --         192.168.0.10
 N  192.168.32.11/32  INTER  B  10   | 192.168.1.6  --         192.168.0.10
 N  192.168.33.0/24   INTER  B   8   | 192.168.1.6  --         192.168.0.10
 N  192.168.34.0/24   INTER  B  12   | 192.168.1.6  --         192.168.0.10
 N  192.168.35.0/24   INTER  B  10   | 192.168.1.6  --         192.168.0.10
-------------------------------------------------------------------------------
```

Figure 4.3. Routing table of the router `192.168.0.6`

the cost to reach `192.168.0.10/32` plus the cost it has to the considered point to point link.

A route towards prefix `192.168.0.8/30` is also present in the routing table: it corresponds to the prefix configured on the link between routers `192.168.0.8/30` and `192.168.0.7/30`. Also in this case, the cost is consistent with the way OSPF models this situation.

Inter-area routes describe destinations in areas 1 and 2. Destinations in area 3 are not reachable due to the absence of a virtual link in area 2 as explained above. We can observe that the best path for destinations in area 1 is always provided by border router `192.168.0.3`, while border router `192.168.0.10` is always chosen to reach destinations in area 2. The reader can verify that the cost is correctly computed for each inter-area destination. For instance, to reach router `192.168.16.1/32` router has a cost of 6 (the cost towards `192.168.0.3` that can be found among the intra-area routes in the same routing table) and 1 (the cost the border router has to reach the destination that can be verified in figure 4.3.3).

In figure 4.3.3 we can verify that router `192.168.0.4` has intra-area routes for destinations of the two areas it belongs to. This matches the description of the routing table computation we have made in chapter 2. Moreover, the router has two routing table entries for border router `192.168.0.3`. The field *area* (`A`) is used to distinguish the two routes since one belongs to the backbone, and the other belongs

```
*******************************************************************************
** 192.168.0.4 OSPF routing table
*******************************************************************************
-------------------------------------------------------------------------------
 T  DESTINATION ID   PATH   A   C  | INTERFACE     NEXT HOP      ADV
-------------------------------------------------------------------------------
 R  192.168.0.3/32   INTRA  B  21  | 192.168.0.5   --            --
 N  192.168.0.5/32   INTRA  B   8  | 192.168.0.5   --            --
 N  192.168.0.6/32   INTRA  B  15  | 192.168.0.5   --            --
 R  192.168.0.7/32   INTRA  B  14  | 192.168.0.5   --            --
 R  192.168.0.10/32  INTRA  B  22  | 192.168.0.5   --            --
 N  192.168.1.5/32   INTRA  B  22  | 192.168.0.5   --            --
 N  192.168.1.6/32   INTRA  B  27  | 192.168.0.5   --            --
 N  192.168.1.8/30   INTRA  B  14  | 192.168.0.5   --            --
-------------------------------------------------------------------------------
 R  192.168.0.3/32   INTRA  1   1  | 192.168.19.4  192.168.19.3  --
 N  192.168.16.1/32  INTRA  1   1  | 192.168.19.4  192.168.19.1  --
 N  192.168.16.2/32  INTRA  1   1  | 192.168.19.4  192.168.19.2  --
 N  192.168.17.0/24  INTRA  1   4  | 192.168.19.4  192.168.19.1  --
 N  192.168.18.0/24  INTRA  1   4  | 192.168.19.4  192.168.19.2  --
 N  192.168.19.0/24  INTRA  1   1  | 192.168.19.4  --            --
 N  192.168.20.0/24  INTRA  1   3  | 192.168.19.4  192.168.19.3  --
-------------------------------------------------------------------------------
 N  192.168.32.8/32  INTER  B  15  | 192.168.0.5   --            192.168.0.7
 N  192.168.32.11/32 INTER  B  18  | 192.168.0.5   --            192.168.0.7
 N  192.168.33.0/24  INTER  B  15  | 192.168.0.5   --            192.168.0.7
 N  192.168.34.0/24  INTER  B  19  | 192.168.0.5   --            192.168.0.7
 N  192.168.35.0/24  INTER  B  18  | 192.168.0.5   --            192.168.0.7
-------------------------------------------------------------------------------
```

Figure 4.4. Routing table of the router `192.168.0.4`

to area 1. Border router `192.168.0.3` stores two routes towards `192.168.0.4` (see figure 4.3.3). The same happens for routers `192.168.0.7` and `192.168.0.10` that share area 2 and the backbone.

The inter-area routes are correctly tagged with a value *backbone* for the field area. Indeed, a border router considers only summaries injected in the backbone to compute inter-area routes. The destinations reachable by an inter-area route are those in area 2 only. Destinations in area 3 are unreachable by router `192.168.0.4`, as they should be.

4.3.4 Routes' computation inside areas 2 and 3

In area 2 we consider the routing table of router `192.168.32.8`. We can verify that intra-area routes concern only destinations in area 2. Inter-area routes on the other hand concern destinations in all other areas. Note that routers in this area can reach destinations in area 3 since border router `192.168.32.11` announces them. Thus our model is consistent with the OSPF specification also from this viewpoint.

We examine border router `192.168.32.11` to validate the routes computation in area 3 (figure 4.3.4). As explained before, this router knows all destinations of the areas it belongs to (area 2 and 3). However, since it is not directly connected to the

```
*******************************************************************************
** 192.168.0.3 OSPF routing table
*******************************************************************************
-------------------------------------------------------------------------------
 T  DESTINATION ID   PATH     A   C  | INTERFACE     NEXT HOP      ADV
-------------------------------------------------------------------------------
 R  192.168.0.4/32   INTRA    B  22  | 192.168.0.6   --            --
 N  192.168.0.5/32   INTRA    B  14  | 192.168.0.6   --            --
 N  192.168.0.6/32   INTRA    B   8  | 192.168.0.6   --            --
 R  192.168.0.7/32   INTRA    B  20  | 192.168.0.6   --            --
 R  192.168.0.10/32  INTRA    B  15  | 192.168.0.6   --            --
 N  192.168.1.5/32   INTRA    B  15  | 192.168.0.6   --            --
 N  192.168.1.6/32   INTRA    B  20  | 192.168.0.6   --            --
 N  192.168.1.8/30   INTRA    B  20  | 192.168.0.6   --            --
-------------------------------------------------------------------------------
 R  192.168.0.4/32   INTRA    1   1  | 192.168.19.3  192.168.19.4  --
 N  192.168.16.1/32  INTRA    1   1  | 192.168.19.3  192.168.19.1  --
 N  192.168.16.2/32  INTRA    1   1  | 192.168.19.3  192.168.19.2  --
 N  192.168.17.0/24  INTRA    1   4  | 192.168.19.3  192.168.19.1  --
 N  192.168.18.0/24  INTRA    1   4  | 192.168.19.3  192.168.19.2  --
 N  192.168.19.0/24  INTRA    1   1  | 192.168.19.3  --            --
 N  192.168.20.0/24  INTRA    1   2  | 192.168.20.3  --            --
-------------------------------------------------------------------------------
 N  192.168.32.8/32  INTER    B  16  | 192.168.0.6   --            192.168.0.10
 N  192.168.32.11/32 INTER    B  18  | 192.168.0.6   --            192.168.0.10
 N  192.168.33.0/24  INTER    B  16  | 192.168.0.6   --            192.168.0.10
 N  192.168.34.0/24  INTER    B  20  | 192.168.0.6   --            192.168.0.10
 N  192.168.35.0/24  INTER    B  18  | 192.168.0.6   --            192.168.0.10
-------------------------------------------------------------------------------
```

Figure 4.5. Routing table of the router `192.168.0.3`

backbone and no virtual link is configured in area 2, it does not consider summaries injected in area 2. Thus it does not know destinations in the backbone nor in area 1. In figure 4.3.4 the reader can verify that no route is computed for this destination.

Finally we examine router `192.168.64.12` whose routing table is presented in figure 4.3.4. We can verify that this router computes intra-area routes only for destinations in area 3 while the inter-area routes are only about destinations in area 2. Only destinations in area 2 are announced by border router `192.168.32.11` since it has no connectivity (physically or by a virtual link) with the backbone. The exit router is always border router `192.168.32.11` as it is the sole border router of the area.

4.4 Validation of the model - Scenario 2

Here we perform a simple modification to the scenario of the previous section. We allow border router `192.168.32.11` to belong to the backbone in order to verify that the computation of the routes has changed. This is obtained by adding a new physical link between routers `192.168.0.3` and `192.168.32.11`. The new link is assigned to the backbone area. The new scenario is illustrated in figure 4.9.

```
*******************************************************************************
** 192.168.32.8 OSPF routing table
*******************************************************************************
-------------------------------------------------------------------------------
 T  DESTINATION ID    PATH     A   C   | INTERFACE     NEXT HOP       ADV
-------------------------------------------------------------------------------
 R  192.168.0.7/32    INTRA    2   1   | 192.168.33.8  192.168.33.7   --
 R  192.168.0.10/32   INTRA    2   1   | 192.168.33.8  192.168.33.10  --
 R  192.168.32.11/32  INTRA    2   4   | 192.168.33.8  192.168.33.10  --
 N  192.168.33.0/24   INTRA    2   1   | 192.168.33.8  --             --
 N  192.168.34.0/24   INTRA    2   4   | 192.168.34.8  --             --
 N  192.168.35.0/24   INTRA    2   4   | 192.168.33.8  192.168.33.10  --
-------------------------------------------------------------------------------
 N  192.168.0.3/32    INTER    2  12   | 192.168.33.8  192.168.33.10  192.168.0.10
 N  192.168.0.4/32    INTER    2  15   | 192.168.33.8  192.168.33.7   192.168.0.7
 N  192.168.0.5/32    INTER    2   7   | 192.168.33.8  192.168.33.7   192.168.0.7
 N  192.168.0.6/32    INTER    2   6   | 192.168.33.8  192.168.33.10  192.168.0.10
 N  192.168.1.5/32    INTER    2  13   | 192.168.33.8  192.168.33.10  192.168.0.10
 N  192.168.1.6/32    INTER    2   6   | 192.168.33.8  192.168.33.10  192.168.0.10
 N  192.168.1.8/30    INTER    2   7   | 192.168.33.8  192.168.33.7   192.168.0.7
 N  192.168.16.1/32   INTER    2  13   | 192.168.33.8  192.168.33.10  192.168.0.10
 N  192.168.16.2/32   INTER    2  13   | 192.168.33.8  192.168.33.10  192.168.0.10
 N  192.168.17.0/24   INTER    2  16   | 192.168.33.8  192.168.33.10  192.168.0.10
 N  192.168.18.0/24   INTER    2  16   | 192.168.33.8  192.168.33.10  192.168.0.10
 N  192.168.19.0/24   INTER    2  13   | 192.168.33.8  192.168.33.10  192.168.0.10
 N  192.168.20.0/24   INTER    2  14   | 192.168.33.8  192.168.33.10  192.168.0.10
 N  192.168.64.9/32   INTER    2   5   | 192.168.33.8  192.168.33.10  192.168.32.11
 N  192.168.64.12/32  INTER    2   5   | 192.168.33.8  192.168.33.10  192.168.32.11
 N  192.168.65.0/24   INTER    2   5   | 192.168.33.8  192.168.33.10  192.168.32.11
 N  192.168.66.0/24   INTER    2   7   | 192.168.33.8  192.168.33.10  192.168.32.11
 N  192.168.67.0/24   INTER    2   8   | 192.168.33.8  192.168.33.10  192.168.32.11
-------------------------------------------------------------------------------
```

Figure 4.6. Routing table of the router `192.168.32.8`

```
*******************************************************************************
** 192.168.32.11 OSPF routing table
*******************************************************************************
-------------------------------------------------------------------------
 T  DESTINATION ID   PATH   A   C  | INTERFACE      NEXT HOP     ADV
-------------------------------------------------------------------------
 R  192.168.0.7/32   INTRA  2   3  | 192.168.35.11  192.168.35.10  --
 R  192.168.0.10/32  INTRA  2   2  | 192.168.35.11  192.168.35.10  --
 N  192.168.32.8/32  INTRA  2   3  | 192.168.35.11  192.168.35.10  --
 N  192.168.33.0/24  INTRA  2   3  | 192.168.35.11  192.168.35.10  --
 N  192.168.34.0/24  INTRA  2   7  | 192.168.35.11  192.168.35.10  --
 N  192.168.35.0/24  INTRA  2   2  | 192.168.35.11  --             --
-------------------------------------------------------------------------
 N  192.168.64.9/32  INTRA  3   1  | 192.168.65.11  192.168.65.9   --
 N  192.168.64.12/32 INTRA  3   1  | 192.168.65.11  192.168.65.12  --
 N  192.168.65.0/24  INTRA  3   1  | 192.168.65.11  --             --
 N  192.168.66.0/24  INTRA  3   3  | 192.168.65.11  192.168.65.12  --
 N  192.168.67.0/24  INTRA  3   4  | 192.168.65.11  192.168.65.9   --
-------------------------------------------------------------------------
```

Figure 4.7. Routing table of the router `192.168.32.11`

4.4.1 Extending the backbone

To verify that something has changed in the computation of the routes we analyze the new routing table of router `192.168.16.1`, reported in figure 4.4.1. We see that

```
*******************************************************************************
** 192.168.64.12 OSPF routing table
*******************************************************************************
-------------------------------------------------------------------------------
 T  DESTINATION ID    PATH    A   C   | INTERFACE      NEXT HOP       ADV
-------------------------------------------------------------------------------
 R  192.168.32.11/32  INTRA   3   1   | 192.168.65.12  192.168.65.11  --
 N  192.168.64.9/32   INTRA   3   1   | 192.168.65.12  192.168.65.9   --
 N  192.168.65.0/24   INTRA   3   1   | 192.168.65.12  --              --
 N  192.168.66.0/24   INTRA   3   2   | 192.168.66.12  --              --
 N  192.168.67.0/24   INTRA   3   4   | 192.168.65.12  192.168.65.9   --
-------------------------------------------------------------------------------
 N  192.168.0.7/32    INTER   3   4   | 192.168.65.12 192.168.65.11  192.168.32.11
 N  192.168.0.10/32   INTER   3   3   | 192.168.65.12 192.168.65.11  192.168.32.11
 N  192.168.32.8/32   INTER   3   4   | 192.168.65.12 192.168.65.11  192.168.32.11
 N  192.168.33.0/24   INTER   3   4   | 192.168.65.12 192.168.65.11  192.168.32.11
 N  192.168.34.0/24   INTER   3   8   | 192.168.65.12 192.168.65.11  192.168.32.11
 N  192.168.35.0/24   INTER   3   3   | 192.168.65.12 192.168.65.11  192.168.32.11
-------------------------------------------------------------------------------
```

Figure 4.8. Routing table of the router `192.168.64.12`

destinations in area 3 are reachable while in the previous scenario it was not the case. The router chose border router `192.168.0.3` to reach destinations in area 3. Note that with the new link introduced, destinations in area 2 become closer to area 1: the other border router is chosen only to reach two destinations in the backbone.

This discussion about the routing table of router `192.168.16.1` should convince the reader that routing tables of the backbone and of area 2 will also change accordingly to the changes in area 1. To verify that routing tables in area 3 are correctly changed we discuss the routing table of router `192.168.67.4` shown in figure 4.4.1.

4.5 Performance test

We have tested the performance of our implementation on a topology with a large number of routers. The topolology was generated by IGen [13]. IGen is a network topology generator that does not rely on probabilistic methods. It implements various network design heuristics for the purpose of building realistic network topologies.

The test was performed as follows: nine topologies were generated by using IGen. The standard parameters in IGen were used to build a topology fitting the Northern American continent. The topologies have a number of nodes ranging from 100 to 900. From one topology to the next, the number of router is increased by 100. IGen is able to generate the OSPF configuration too. In this case, all the topologies have a number of areas equals to five, considering the backbone. The result is a script for the CLI of C-BGP.

Each synthetic topology is loaded in the simulator and we compute the routes, and we record the computation time. The measurement of the computation time was performed using the standard C function `gettimeofday()` that provides the

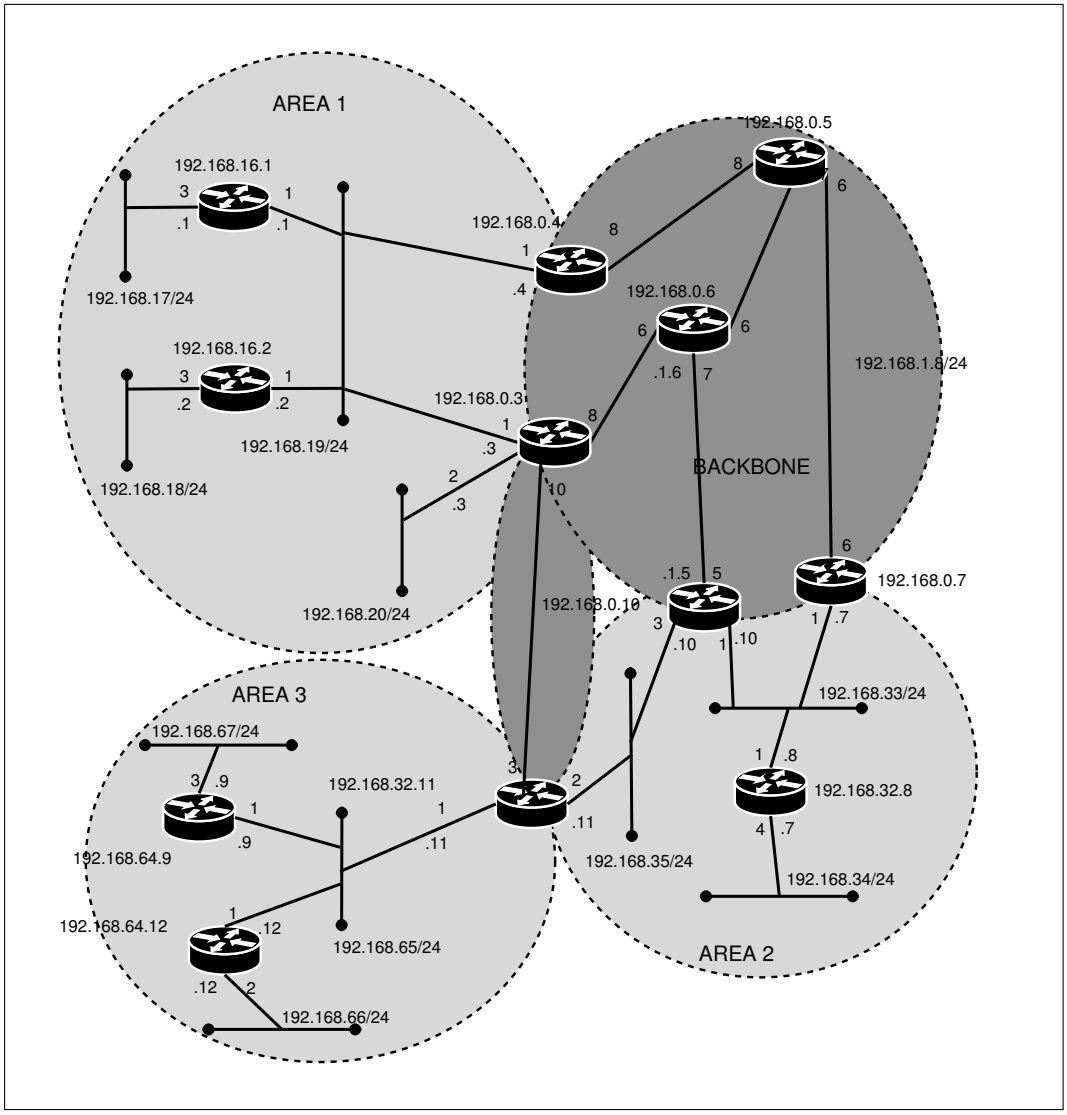

Figure 4.9. Scenario 2

time in seconds and microseconds.

The experiment was performed on a *Toshiba ®Satellite, Intel ®Pentium ®4, 1.7 GHz, Front Side Bus 400 MHz, L2 cache of 256 KB*. The machine runs a Linux Gentoo operating system.

To obtain a more precise measurement, the routes were computed 10 times for each topology, to include in the measurements the variance of the computation time. The average for each series of data was computed and plotted in the chart in figure 4.12. Here the number of routers of the area are showed on the X axis, while on the Y axis the time (in seconds) required for the computation of the routes is reported.

```
*****************************************************************************
** 192.168.16.1 OSPF routing table
*****************************************************************************
-----------------------------------------------------------------------------
 T  DESTINATION ID   PATH   A   C  | INTERFACE     NEXT HOP      ADV
-----------------------------------------------------------------------------
 R  192.168.0.3/32   INTRA  1   1  | 192.168.19.1  192.168.19.3  --
 R  192.168.0.4/32   INTRA  1   1  | 192.168.19.1  192.168.19.4  --
 N  192.168.16.2/32  INTRA  1   1  | 192.168.19.1  192.168.19.2  --
 N  192.168.17.0/24  INTRA  1   3  | 192.168.17.1  --            --
 N  192.168.18.0/24  INTRA  1   4  | 192.168.19.1  192.168.19.2  --
 N  192.168.19.0/24  INTRA  1   1  | 192.168.19.1  --            --
 N  192.168.20.0/24  INTRA  1   3  | 192.168.19.1  192.168.19.3  --
-----------------------------------------------------------------------------
 N  192.168.0.5/32   INTER  1   9  | 192.168.19.1  192.168.19.4  192.168.0.4
 N  192.168.0.6/32   INTER  1   9  | 192.168.19.1  192.168.19.3  192.168.0.3
 N  192.168.0.7/32   INTER  1  15  | 192.168.19.1  192.168.19.4  192.168.0.4
 N  192.168.0.10/32  INTER  1  16  | 192.168.19.1  192.168.19.3  192.168.0.3
 N  192.168.32.8/32  INTER  1  14  | 192.168.19.1  192.168.19.3  192.168.0.3
 N  192.168.32.11/32 INTER  1  11  | 192.168.19.1  192.168.19.3  192.168.0.3
 N  192.168.33.0/24  INTER  1  14  | 192.168.19.1  192.168.19.3  192.168.0.3
 N  192.168.34.0/24  INTER  1  18  | 192.168.19.1  192.168.19.3  192.168.0.3
 N  192.168.35.0/24  INTER  1  13  | 192.168.19.1  192.168.19.3  192.168.0.3
 N  192.168.64.9/32  INTER  1  12  | 192.168.19.1  192.168.19.3  192.168.0.3
 N  192.168.64.12/32 INTER  1  12  | 192.168.19.1  192.168.19.3  192.168.0.3
 N  192.168.65.0/24  INTER  1  12  | 192.168.19.1  192.168.19.3  192.168.0.3
 N  192.168.66.0/24  INTER  1  14  | 192.168.19.1  192.168.19.3  192.168.0.3
 N  192.168.67.0/24  INTER  1  15  | 192.168.19.1  192.168.19.3  192.168.0.3
-----------------------------------------------------------------------------
```

Figure 4.10. Routing table of the router `192.168.16.1` for the scenario 2

We can observe a quadratic trend in the chart. The computation time depends from the way the selection of the border routers to inspect is performed during the computation of the inter-area routes. As explained in section 4.2.2, this has a $O(N^2)$ complexity. Improving the selection of the border routers, as we proposed in the same section, should reduce drastically the total time required to the computation of all the routes.

4.6 Conclusions

In this chapter we have discussed the prototype implementation of our OSPF model. The prototype implements the computation of the intra-area and of the inter-area routes. Before discussing some details of the prototype we have illustrated the modification made to the topology model of C-BGP (introducing the subnets). This was necessary in order to fully model LISs.

We have then discussed some details of our implementation. We require that the OSPF configuration of the domain be performed into two steps. First the router and the subnets are assigned to the areas. In the second step the links are assigned to the areas. This guarantees a correct configuration and also permits to implement

```
*******************************************************************************
** 192.168.64.12 OSPF routing table
*******************************************************************************
-------------------------------------------------------------------------------
 T  DESTINATION ID    PATH   A   C  | INTERFACE      NEXT HOP       ADV
-------------------------------------------------------------------------------
 R  192.168.32.11/32  INTRA  3   1  | 192.168.65.12  192.168.65.11  --
 N  192.168.64.9/32   INTRA  3   1  | 192.168.65.12  192.168.65.9   --
 N  192.168.65.0/24   INTRA  3   1  | 192.168.65.12  --              --
 N  192.168.66.0/24   INTRA  3   2  | 192.168.66.12  --              --
 N  192.168.67.0/24   INTRA  3   4  | 192.168.65.12  192.168.65.9   --
-------------------------------------------------------------------------------
 N  192.168.0.3/32    INTER  3   4  | 192.168.65.12  192.168.65.11  192.168.32.11
 N  192.168.0.4/32    INTER  3  26  | 192.168.65.12  192.168.65.11  192.168.32.11
 N  192.168.0.5/32    INTER  3  18  | 192.168.65.12  192.168.65.11  192.168.32.11
 N  192.168.0.6/32    INTER  3  12  | 192.168.65.12  192.168.65.11  192.168.32.11
 N  192.168.0.7/32    INTER  3   4  | 192.168.65.12  192.168.65.11  192.168.32.11
 N  192.168.0.10/32   INTER  3   3  | 192.168.65.12  192.168.65.11  192.168.32.11
 N  192.168.16.1/32   INTER  3   5  | 192.168.65.12  192.168.65.11  192.168.32.11
 N  192.168.16.2/32   INTER  3   5  | 192.168.65.12  192.168.65.11  192.168.32.11
 N  192.168.17.0/24   INTER  3   8  | 192.168.65.12  192.168.65.11  192.168.32.11
 N  192.168.18.0/24   INTER  3   8  | 192.168.65.12  192.168.65.11  192.168.32.11
 N  192.168.19.0/24   INTER  3   5  | 192.168.65.12  192.168.65.11  192.168.32.11
 N  192.168.20.0/24   INTER  3   6  | 192.168.65.12  192.168.65.11  192.168.32.11
 N  192.168.32.8/32   INTER  3   4  | 192.168.65.12  192.168.65.11  192.168.32.11
 N  192.168.33.0/24   INTER  3   4  | 192.168.65.12  192.168.65.11  192.168.32.11
 N  192.168.34.0/24   INTER  3   8  | 192.168.65.12  192.168.65.11  192.168.32.11
 N  192.168.35.0/24   INTER  3   3  | 192.168.65.12  192.168.65.11  192.168.32.11
-------------------------------------------------------------------------------
```

Figure 4.11. Routing table of the router `192.168.64.12` for the scenario 2

the checks performed by the Hello protocol.

We have chosen to keep a separate routing table for the OSPF protocol, as it is the case in practice. The OSPF routing table contains some fields that are used during the computation of the routes. Hence, it is natural to keep BGP routing tables and OSPF ones separate to minimize the interactions between the code of the OSPF model and other parts of the C-BGP routing solver. Since the BGP routing table is also used for the forwarding process inside C-BGP, the OSPF routes must be installed inside the BGP routing table before running the BGP simulation.

Then some details of the implementation are discussed. We provide a simple Dijkstra's algorithm implementation. The only optimization consists in considering the routers and the transit networks in a first step and the stub networks in a second step. With a multi-area architecture, we expect that the size of the graph on which the SPT is computed will be far smaller than the size of the whole IGP domain. Thus no special optimization of the SPT computation is expected to bring significant computational advantage. However, if improving the SPT computation becomes necessary, priority queues or Fibonacci heaps could be used. We also proposed an improvement to the selection of which border routers to inspect during the inter-area routes computation.

Finally a validation of the model was proposed. We have tested our model on

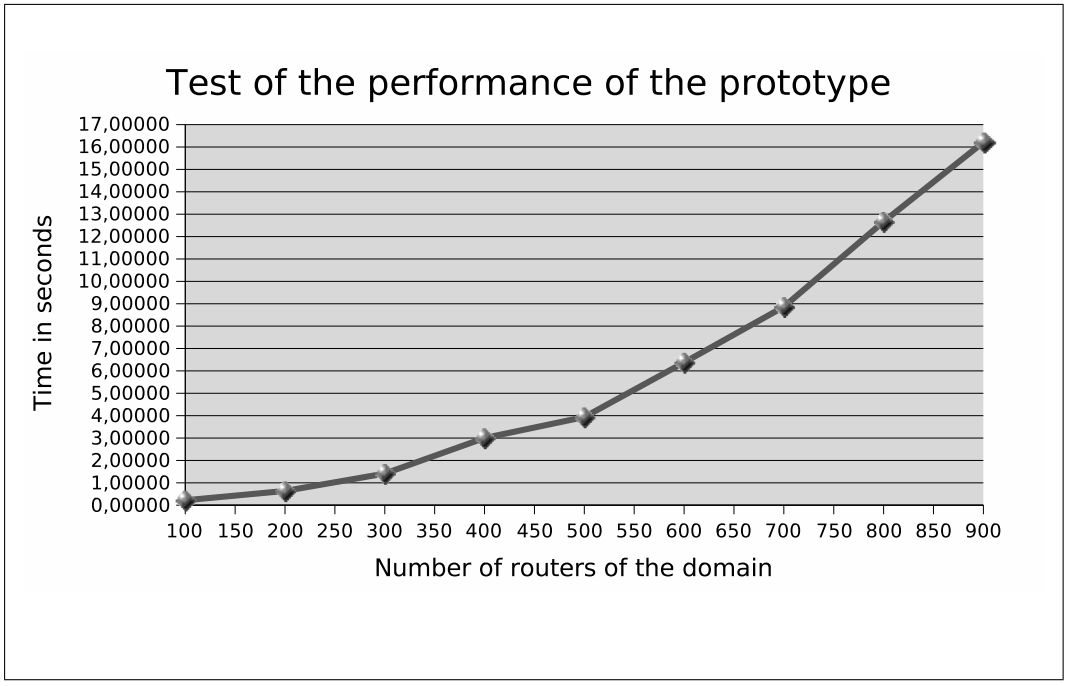

Figure 4.12. Test of the performance of the prototype.

two representative scenarios, to show that our model is able to reproduce the OSPF behavior in terms of the routes' computation.

Chapter 5

Forwarding deflection in OSPF

In this chapter we analyze how the way OSPF routers compute their routes makes it possible that the forwarding path to differ from the routing path. The forwarding path is the one followed by IP packet while the latter is the one computed by the routing protocol. The two paths should be the same but this does not happen in some situations.

We discuss the interest of these special situations and how they are linked to the presence of suboptimal paths in the network. We propose methods to detect the two problem separately (suboptimal paths and deflection). Then we propose a unified approach to these problems.

5.1 Problem statement

We define *forwarding deflection* as the inconsistency of the forwarding path compared to the routing path for a given destination. The forwarding path results from the choices that each router makes when it receives a packet towards a destination. The routing path instead is calculated in OSPF with the routing process explained in chapter 2. Usually, these two should be consistent but the existence of deflection captures this inconsistency.

In this section we present a possible scenario where deflection arises and explain why it happens.

5.1.1 An example of deflection

Figure 5.1 illustrates an example of forwarding deflection. In this figure three different areas are shown. The source of the traffic (S) is located in area 1, the destination (D) is located in area 2, and all border routers (denoted by BRx) belong to area 0 (backbone), as well as to other areas. BR1 and BR2 belong to area 1 and 0, and

BR3 belongs to area 2 and 0. Arrows on figure 5.1 represent routing paths, not physical links.

We assume that all routers have computed their shortest paths towards all destinations, and that OSPF has converged inside the network. The routing table of each router contains all the routes for intra-area and inter-area destinations.

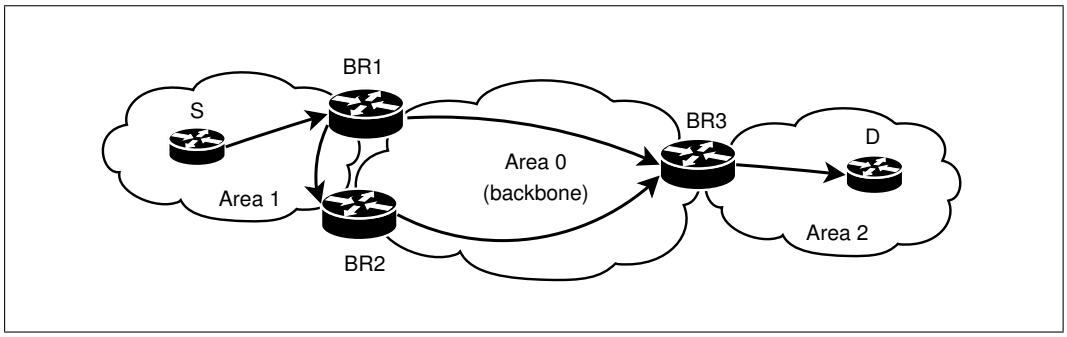

Figure 5.1. Example of deflection

Suppose that S wants to reach router D. The route that S uses towards D has BR2 as its exit router. The intra-area path of S towards BR2 crosses border router BR1. When a packet from S towards D arrives at BR1, the latter does not forward the packet to BR2 on the path inside area 1 as thought by S. Rather, BR1 sends the traffic towards D on one of its links belonging to area 0.

Deflection arises in this example because S thinks that packets towards D will follow links within area 1 to reach BR2, while BR1 will never use links belonging to area 1 to reach D. For BR1 to reach D using links within area 1, D would have to lie inside area 1.

5.1.2 Causes of deflection

Two properties of OSPF inter-area routing (independently) lead to deflection.

Firstly, a border router $BR1$ only considers summaries injected inside the backbone. If a destination D is not in one of the areas $BR1$ belongs to it will thus not be able to choose a path via an area $i \neq 0$ for the destination. Due to this behavior of the border routers all the traffic from an area towards a different one will pass across the backbone. However, $BR1$ could be on the shortest path from a router X in area i to the border router $BR2$ that X selected for destination D. When a packet with destination D, entering the network in X will reach $BR1$, it will be forwarded into the backbone, instead of being forwarded to BR_2 and only then be forwarded into the backbone. Packets can thus be forwarded out of a **non-backbone area** via a border router that is different from the border router that was selected by upstream

routers. In figure 5.1 this happens for BR1 that is on the shortest path S uses to reach D exiting the area by BR2.

Secondly, a router will always prefer an intra-area path over an inter-area path as explained in 2. This means that if a border router BR is connected to two areas i and j, it will forward a packet towards a destination D in i along its shortest path to D within area i. However, $BR1$ could be on the shortest path from a router X in area j to the border router $BR2$ that X selected for destination D. When a packet with destination D reaches $BR1$, it will be forwarded to BR_1, and be deflected into area i, instead of being forwarded to BR_2 using links in area j. Figure 5.2 reports an example of this situation.

In figure 5.2 area j is the backbone. Here the border router BR has two possible paths towards destination D: the intra-area one (represented by a dotted line) and the inter-area one (represented by a dashed line). The intra-area path is preferred. Router S selects ER to reach the inter-area destination D. When S forwards the packet for inter-area destination D to BR, the latter uses the inter-area path towards D: this results in a different path from the one computed by S.

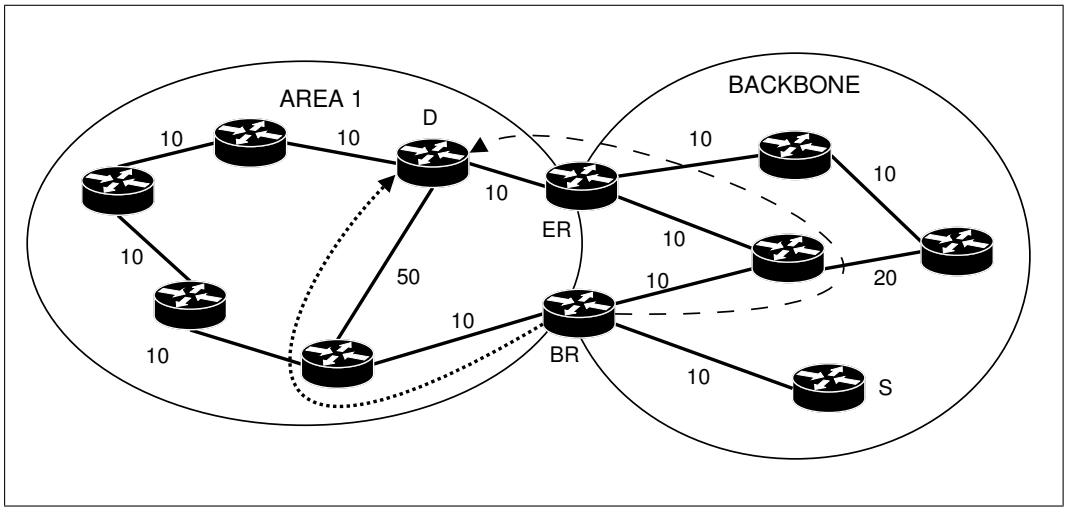

Figure 5.2. Example of deflection due to preference of intra-area path.

We can observe that first property leads deflection in an area when at least one among the source and destination are not in the backbone. The second property, instead, leads to deflection when the source's area and the destination's one are adjacent.

5.1.3 Why studying deflection

Deflection introduces a discordance beetwen the routing plane and the forwading plane in the network. What seems interesting is that there are no symptoms of

this situation: network does not send any perceptible signal about this. All seems working correctly: packets are sent towards the destination and reach it without problem. While deflection seems to not have any impact on the users, can be important to detect it from the operators' viewpoint. In fact, they can perform choices in terms of network design and traffic engineering that can be invalidated by this behavior.

At the design time, for example, network designer can organize multi-area architecture and assigning links weight in order to distrubuite the traffic flow [14]. Usually this is done by using a traffic matrix that defines the required amount of traffic exchanged between each couple of nodes in the network. The link weights are optimized in order to obtain the required loads. If during this process we do not take into account that deflection can be happend and useles optimization can be done. We will choose the link weights considering the routing viewpoint that could will be different from the forwarding one cause to deflection. Thus the actual traffic load could not correspond to the required one.

Moreover, the operator can organize the traffic flows deciding a certain path in the domain, or putting some constraint on the usage of the links in terms of delay or bandwith. This consist into features added to OSPF to permit traffic engineering. However, the features do not take into account multi-area scenarios [15].

Deflection can also have consequences on the management of a network. For instance, the operator sets BGP sessions to exchange the external AS routes between the AS's routers. Among all the available routes each router has, it chooses a single best one for each destination prefix using the "BGP decision process" [16]. This choice is based on the attributes of the BGP routes. One of these attributes is the BGP next hop, which is used in the decision process to choose the best route. When several BGP routes having the same quality for the decision process arrive at the next hop rule, BGP chooses the route having the next hop towards which the IGP cost is smallest. If the actual IGP cost of the path followed by the packets is different from the one towards the next hop of the BGP route, then the decision on which BGP relies will be incorrect. In this way the inconsistency between the routing plane and the forwarding plane in OSPF is transmitted to the BGP plane.

In the same way, when a BGP router advertizes a BGP route to a neighboring AS for which the forwarding path differs from the routing one, the information on which the BGP decision process of the neighboring ASs will rely will also be incorrect. Generalizing the example of BGP we can affirm that all the network protocols that assume a coherence beetwen the routing path and the forwarding path could have trouble due to deflection.

5.2 Deflection and suboptimal path

In the previous section we have shown that each time deflection arises this is caused by a choice performed by a border router. Here we show that this choice always corresponds to a preference for a suboptimal routing path. However, there are some situations in which a border router performs a suboptimal choice but this does not lead to deflection.

Since we have shown that the suboptimal routing path can cause deflection, we propose a partial solution to the suboptimal routing problem and thus deflection. We also show that there are situations in which this solution does not work.

5.2.1 Relationship between deflection and suboptimal paths

The objective of each routing protocol is to find the best path from a source towards a destination in the network. For IGP routing the best path is often defined in terms of link weights. When multi-area architecture is introduced it can happen that the path computed is not the best path. We disregard here the suboptimal path introduced by the use of stub areas and address summarization. We focus our discussion on suboptimal path introduced by the rules that an OSPF router follows to builds routes. It prefers always an intra-area path over an inter-area path even if the former has a better cost 2.3.

Firstly, we would like to show that deflection implies the presence of a suboptimal path. Let us to comment figure 5.3. To have deflection we need that a border router is on the path between a source S and the selected exit router ER for a given inter-area destination D. This is equivalent to verify the following relationship.

$$B + C < D \tag{5.1}$$

This corresponds to affirm that the path chosen by S is better than the one provided by BR. This path consists into the one followed to the packet due to the choice of BR. Thus deflection always implies a suboptimal routing path choice perfomed by the border router BR. It knows the path that S would use, but it chooses the worst one due to the rules of routes computation. Note that this discussion does not depends by the considered area, thus it does not depend in the fact that the area is the backbone or not.

Second question that we ask to ourself consists in asking whether there is a way for suboptimal path to arise having not deflection. This can happen only when S, the source of the traffic, coincides with the first BR on the path towards D. To discuss this we refer to figure 5.4. Deflection does not arise in the backbone. The only possible suboptimal path is for the border router BR. BR chooses to reach D through an intra-area path although it has a better inter-area path. However,

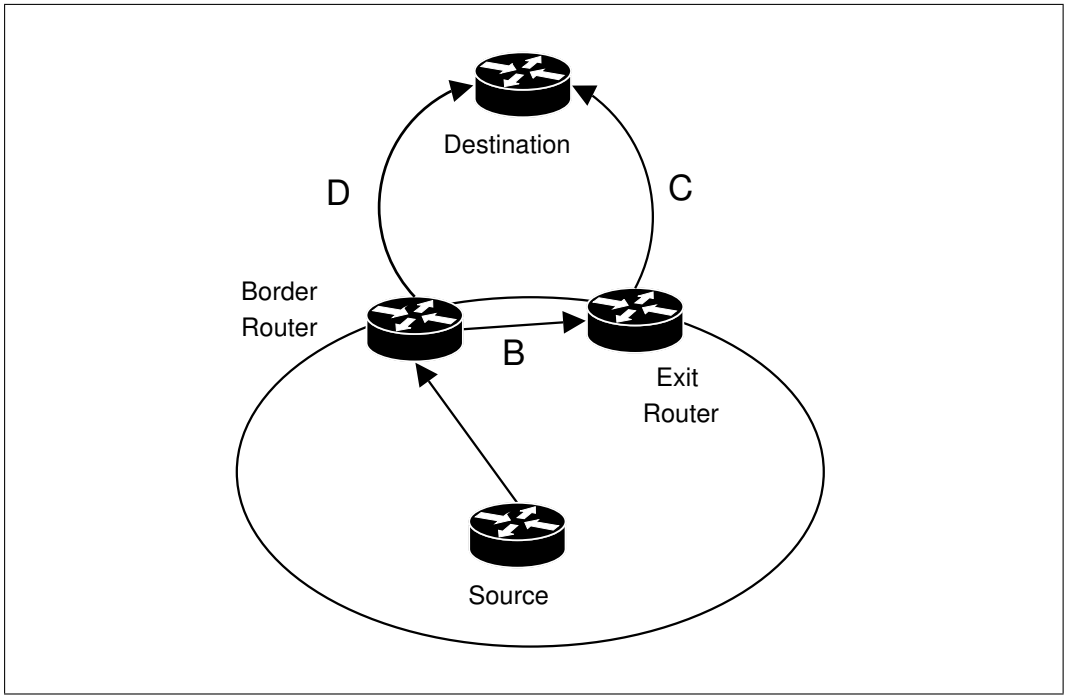

Figure 5.3. Condition to have deflection

since S has visibility on all the available paths, its routing choice coincides with its forwarding choice. No deflection arises because any router in area 1 reach ER across BR and since BR has a full visibility on the two area it belongs to. However, we have a suboptimal path chosen by BR.

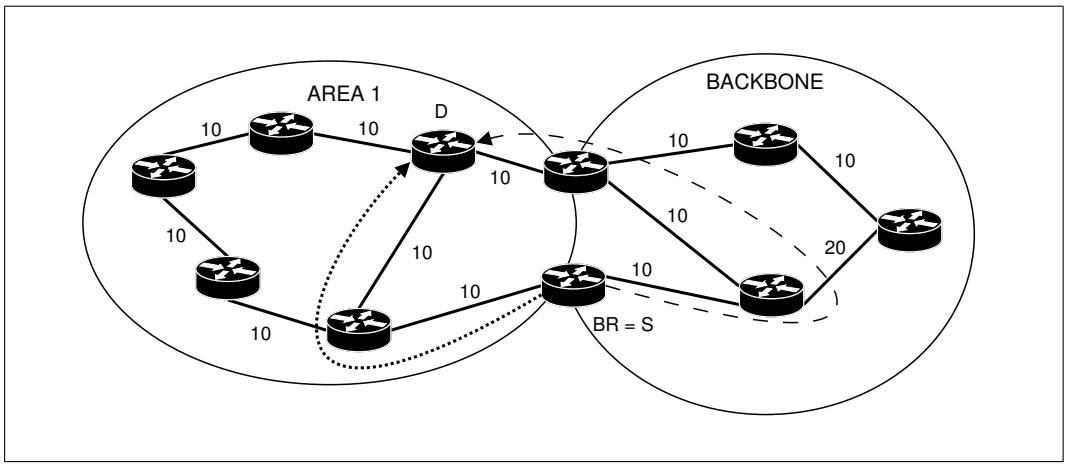

Figure 5.4. A case where suboptimal path are present but deflection does not arise.

Concluding we can affirm that deflection implies a suboptimal path but we can not say that the opposite is true. Equation 5.2 exprimes this affirmation.

$$deflection => suboptimal\ paths \tag{5.2}$$

5.2.2 Fixing the problem

Once we have shown that deflection and suboptimal path are in the relationship expressed by equation 5.2, we discuss a possible way to fix them at the same time, since fixing suboptimal routing we can fix deflection too.

A simple solution proposed to solve this kind of suboptimal routing problem consists in establishing appropiate virtual links. As explained in 2 a virtual link can be configured between border routers only and can belong to an area different from the backbone. Two parameters are required in order to configure a virtual link on a given border router: the router id of the router on the other side of the virtual link and the area on which the virtual link must belong to.

The configuration of a virtual link allows the border router to use links in areas different from the backbone as they belong to it. As a consequence, the border router will compute intra-area routes for destinations located inside the backbone, using paths that do not belong exclusively to the backbone. In the example of figure 5.1 this means that S can reach D using links in area 1 that provide a best path. This is allowed by virtual links that make BR1 and BR2 reachable in the backbone using links in area 1. Without the virtual links, that path is not used. In this way suboptimal path is solved and deflection too.

However, this solution does not solve all the situations of deflection, and thus of suboptimal routing. In figure 5.2, for example, is not possibile to solve deflection with a virtual link. Here we need of a mechanism that permits to use backbone links as they belong to another area. But virtual links permit only the opposite, i.e. using links in other areas (different from the backbone).

To solve deflection in the backbone one could propose a short term solution that consists in configuring static routes. A static route can be setted on the border routers that cause deflection. The static route describes destinations of the deflected path and enforces the border router to use a link in the backbone to forward packets for the given destination. The main limitation of this solution is that it is not self-adaptative to the changes in the network. When the routing change static routes configured in order to avoid deflection can become the cause of the deflection. In the worst case it can happen that deflection is introduced by this routes. Moreover, an approach like this one can be very hard from an operational viewpoint since the operator must know detailed information about the deflected path. This make very hard and inefficient solving deflection with static routes.

Another approach could consist in configuring IP tunnels. Given a deflected path an IP tunnel must be configured between the source and the exit router. The endpoint of the tunnel must be an interface of the exit router and the interface must belong to the source's area. This guarantees that the routing path coincides with the forwarding path. However, also this solution has difficult to be realized since suppose a detailed information about the deflection. Moreover, like static routes solution, it requires manual configuration and this is a big limit from an operational viewpoint. Finally, an overhead due to the double packet incapsulation is introduced.

A long term solution could be provided by a modification in the OSPF protocol. The concept of virtual link can be generalized in order to permit to extend an area different from the backbone, not only the backbone as it is the case in the current version of OSPF. However, the implications and the consequences of this choice must be considered carefully. Discussing the details of this solution is out of the scope of this work.

5.3 Detecting deflection and suboptimal paths separately

In the previous sections we have discussed the forwarding deflection problem and its relationship with the presence of suboptimal paths caused by the multi-area architecture. Once we have shown that by solving suboptimal paths we can solve also deflection we made an effort to find a way to check deflection in a given OSPF domain. Here we discuss how deflection and suboptimal paths can be found separately and in independent ways. In the next section we discuss an algorithm that finds deflection and suboptimal path at the same time.

5.3.1 Detecting deflection

Given an area of an OSPF domain, a simple way to detect deflection could consist in performing a traceroute. The algorithm is shown in figure 5.5. It is thought to be implemented in C-BGP so it takes advantages from the centralized viewpoint provided by C-BGP.

The procedure detects deflection inside one area. To detect deflection in the domain we have only to invoke it for all areas of the domain. The idea consists in checking if, given an inter-area route, the corresponding path is deflected. The path is deflected if the actual exit router id is different from the one selected by the routing process. The latter is contained in the route, while the former can be discovered by the traceroute. We assume to have a procedure *traceroute()* that performs a traceroute bounded to the given area. When *traceroute()* is invoked for a given source, destination and area, it performs the delivery of a message (typically

an ICMP echo request packet) hop by hop. However, when a router chooses a link that does not belong to the given area the procedure stops. It returns the portion of the forwarding path contained in the given area.

To detect deflection, all the routers of the area are considered (line 2) and for each node the routing table is inspected. For each inter-area route in the routing table (line 3) we invoke the *traceroute()* procedure (line 4): the result is the list of routers on the forwarding path in the area. If the last router does not coincide with the selected exit router this means that the path is deflected (lines 5-6).

```
01: procedure check_deflection(area) {
02:     for each router in area {
03:         for each route inter_area in node {
04:             t = traceroute(node, dest(route), area)
05:             if (tail(t) != ER){
06:                 deflection found!
07:             } //end if
08:         } //end for each on router
09:     }//end for each on route inter_area
10: }
```

Figure 5.5. Detecting deflection with traceroute.

This simple way to find deflection has several features. First it does not require to make any assumption on the cause of deflection. The algorithm can be implemented to check not only the deflection introduced by OSPF but every type of deflection. For instance with this approach we can check deflection that can be introduced by other routing protocols or by the configuration of static routes. Moreover, although we have assumed to have a centralized viewpoint on the network, as the one supplied by the C-BGP simulator, the idea of a traceroute bounded to a given area can be reused in a distributed context. This approach can be used in a real network in order to find actual deflection considering a given router. In this perspective this approach could be used in order to obtain a check that operators can use to detect deflection for troubleshooting purposes.

The main limitation of this approach is that it is not able to provide all the couples of border routers between which we should configure a virtual link to fix the deflection problem. This happens when on the path from a router towards the exit router for a given destination, there is more than one border router. A possible scenario is represented in figure 5.6. Here S reaches ER, the selected exit router for D, through BR1 and BR2. If we run our traceroute algorithm for router S, we discover that there is a deflected path beetwen S and D. However, if we wish to fix the problem, the procedure tells only to fix the problem beetwen BR1 and BR2 by

establishing a virtual link between BR1 and ER. This is not sufficent because when BR1 will forward packets towards BR2 inside area i, thanks to the virtual link, BR2 that is on the path will forward them in the backbone. Two virtual links are needed to fix the problem, one between BR1 and BR2 and the other between BR2 and ER. We have no guarantee that this check will detect the couple <BR1, BR2>(or <BR2,BR3>). This is due to the fact that we cannot be sure that a deflected path exists for a source in area 1 that uses BR2 (or ER) as exit router passing through BR1 (or BR2). For instance, this is the case for the scenario of figure 5.6.

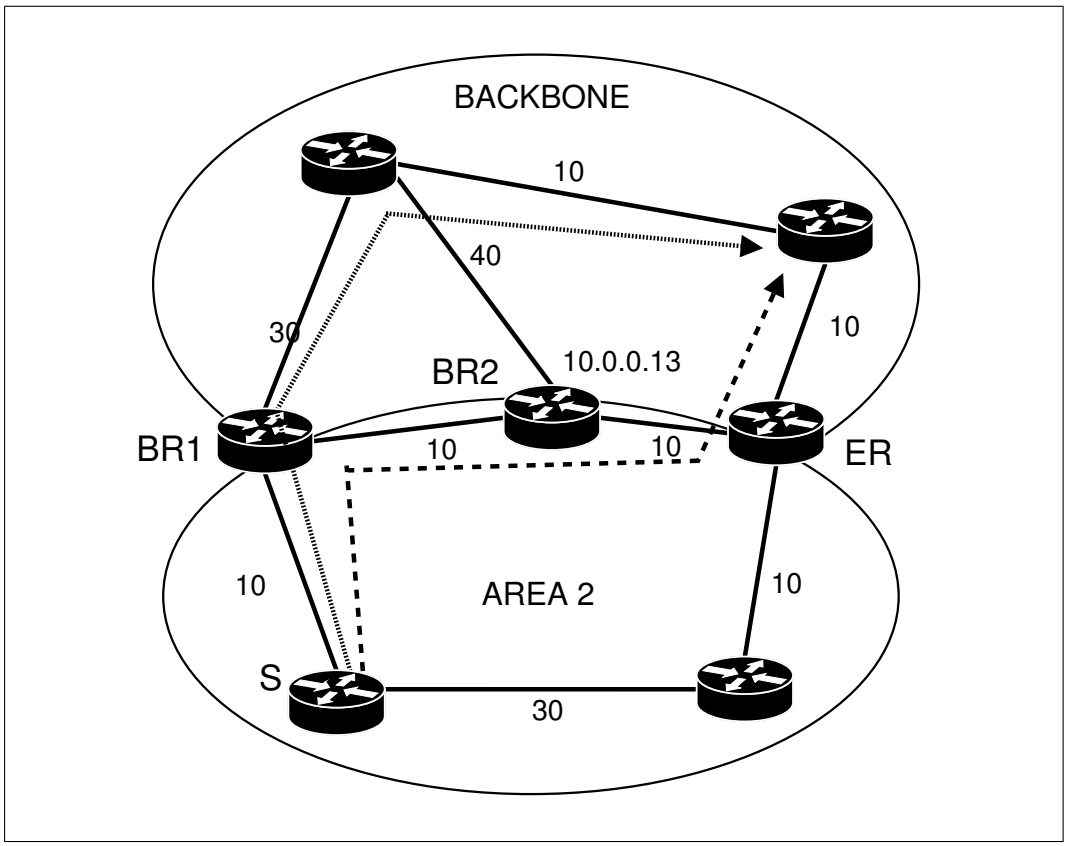

Figure 5.6. The virtual link between BR1 and ER avoid suboptimal path and deflection between S and D.

5.3.2 Detecting suboptimal paths

As in the previous section, we consider in this section the presence of suboptimal paths in a given area. To detect suboptimal paths in the domain, all the areas must be checked and the result will be the list of couples of border routers that cause suboptimal paths.

To detect suboptimal paths, we have to observe that suboptimal routing only occurs between two border routers that lie in the same two areas. The reason is that a suboptimal path can be found only when a border router has an alternative inter-area path to reach a destination. The alternative path can be provided only by another border router.

The idea to detect suboptimal routing consists in considering all couples of border routers of a given area. Given a couple BR1 and BR2 of border routers, in the routing table of BR1 there will be at least as many routes as BR1 and BR2 have areas in common (see section 2.3.1).

The procedure shown on figure 5.7 illustrates a possible way to find suboptimal paths beetwen two areas. It returns the couple of border routers that cause suboptimal paths. We check the presence of suboptimal paths from *area1* towards *area2*. All the possible couples *BR1, BR2* of border routers are considered (line 2). If BR1 and BR2 do not share *area2*, no inter-area routing path can be found in area 2 for BR1 through BR2 (line 3). If the cost of the path between BR1 and BR2 is smaller than the one in area 2, the couple of routers can be the source of suboptimal routing paths and a virtual link should be established between them (lines 4-5).

```
1: procedure check_so_path(area1, area2) {
2:   for each couple of border_router BR1 BR2 in area1 {
3:     if (cost(route(BR1, BR2, area2) != infinite)) {
4:       if (cost(route(BR1, BR2, area1)) < cost(route(BR1, BR2, area2))) {
5:         suboptimal path BR1 beetwen BR2!
6:       } //end if
7:     } //end if
8:   } //end for each on couple
9: }
```

Figure 5.7. Discovering border routers cause of suboptimal paths from area1 to area2.

With the previous check we detect suboptimal routing beetwen two border routers. We have no guarantee that they cause suboptimal routing paths betwen a source different from BR1 and an inter-area destination. For this to happen, deflection must be also verified. Moreover, [3] does not specify inside which area BR1 will reach BR2. Thus we are not able to say if a suboptimal path exists between BR1 and BR2. The answer depends on implementation choices. Of course, in all the other cases (BR1 is the source and the destination is not BR2) suboptimal routing paths are used by BR1 if the check returns a positive answer.

We can observe that with this check we treat a problem that covers only the routing aspect: we check if OSPF has chosen a path that is not the best one for

some destination. This is the reason why we prefer to separate the two problems of deflection and suboptimal paths. This check is not independent from the assumption that we rely on OSPF: we are able to check only the suboptimal paths introduced by the multi-area OSPF architecture. Moreover, the check assumes we have a map of the network and the OSPF configuration, i.e. it takes a centralized point of view of the network. Thus a distributed implementation of this check would be hard.

While the traceroute check is more general than the current one, it is able to provide useful information from an operational viewpoint: the couples of border routers we have to connect with a virtual link. Since a suboptimal path inside the backbone cannot be fixed with a virtual link, in that particular case the check gives the couple of border routers that are a possible source of problems, i.e. that can originate deflection.

5.4 Detecting deflection and suboptimal paths together

The proposed approaches to detect deflection and suboptimal paths contain limitations and advantages as already discussed. From an operational viewpoint we would like to solve both suboptimal paths and deflection. However, since we are actually unable to solve the suboptimal paths problem when it occurs in the backbone, the detailed information provided by the traceroute check could be useful in this case. For this reasons, we have put additional effort in the design of an algorithm that provides the same information as the two checks above at once. We would like to discover the critical couples of border routers, as well as the source and destination of each deflected path.

To obtain that, we have defined below a general condition to detect deflection. This condition was especially designed to be implement inside C-BGP. A detailed presentation of the algorithm is provided, and the output obtained on a case study is discussed.

5.4.1 Condition to check deflection

Let us present the condition we have found for deflection to arise in a given area.

Given

- an area **a** of the OSPF domain,
- a router **S** source of traffic in area **a**,
- a destination **D** that **S** reaches with an inter-area route,

- the exit router **ER** selected by **S** to reach **D**,

deflection can arise for **S** towards **D** when

1. at least another border router **BR** is on the path between **S** and **ER** and
2. the link on which **BR** forwards packets towards **D** must not belong to area **a**.

We can justify and prove this condition simply by observing that it maps directly to the traceroute approach. However defining in this way the condition for deflection to arise we have a way to make an implementation in C-BGP taking advantages by the domain model of C-BGP.

Once the couple of border routers that introduce deflection is discovered by verifying the above condition, we need to find the couple of border routers that do not introduce deflection but that perform suboptimal choices in terms of routing. To do that another check is performed in order to examine the cost each router has towards the other. To avoid adding to much complexity to the algorithm we do not check the presence of suboptimal paths in the backbone (unsolvable anyway) that do not introduce deflection.

5.4.2 Implementing the deflection detection

To check deflection in a domain we check it separately in each of its areas. Given an area, we have to check the conditions of deflection. For this the SPT of all the nodes should be analyzed in order to check the presence of a border router on the path towards the exit router. This check can be done by visiting the tree to find the border router that respects relationship 1 on the path. However, this can result in a complex method. A simpler alternative uses the reverse approach, the Reverse Shortest Path Tree (RSPT).

The input of the algorithm that computes the RSPT is a graph and a node. The result is a tree where the root is the given node. The tree has the following property: the set of links beetwen a node in the tree and its root is the shortest path between the node and the root. In other words, with the computation of the RSPT we find the shortest path of each node of the network towards the root of the RSPT.

To compute the RSPT Dijkstra's algorithm is used with a simple modification. Given a network topology and a node the result of the RSPT and of the SPT computation are the same if the links have symmetric weights, i.e. the weight of the link is the same in the two directions. If we allow the use of asymmetric links weights, the two algorithms may provide different results beacause the SPT uses the weight of the link in the direction from the root to the considered node. The RSPT, instead uses the weight of the link in the direction from the discovered node towards the root.

Given a border router, we compute the RSPT on a border router in order to discover the path that the other nodes use to reach it. Once we have this information we have only to find the border routers in the tree: if they have some child they satisfy condition 1, i.e. on the path they use to reach a border router there is another border router.

Of course this is not sufficient for deflection to arise. Assuming we have found a border router, that we call BR, in the RSPT and that it has a son, which we call S, we must be sure that S uses the root of the RSPT, that we call ER, as exit router for some inter-area destination. To verify this we can examine the routing table of S in search of inter-area routes in which the field *Advertising Router* is filled with a reference to ER. Once this is verified we have to check whether condition 2 is also respected by inspecting BR's routing table.

The previous checks only verify the occurence of deflection. We would like to introduce also a check that finds suboptimal paths. This is easy when a border router is considered in the RSPT. We can check the cost it has towards the exit router ER. Computing the RSPT for each border router and finding each other border router in it permits to consider all the possible couples of border routers in the given area.

In figure 5.8 we show the details of the algorithm. Given an area we consider all the border routers among those belonging to the area (line 2). We compute the RSPT of the border router considering only links in the area (line 3). For each border router BR in the SPT we consider its children (S). We inspect S's routing table to verify that S exits the area using ER for a given destination. If we find a route for which this happens we check that BR uses a link in the area to forward the packet. If this is not the case deflection happens and the path from S to destination(route) is deflected (line 9-10). We assume that function *destination()* returns the destination of the route. If no deflection is found for the considered router we have to check if a suboptimal path is present beetwen BR and ER. We have to compare the cost of the path beetwen BR and ER in the backbone and in the considered area. If the latter is smaller than the former we have to communicate that a virtual link should be installed beetwen the two routers in the given area (line 16). This check is performed only to consider the suboptimal path that are not the cause of deflection. We perform this check only if the examined area is not the backbone (line 15) because this is the only situation we can solve in practice.

Note that the implementation of the procedure must take into account that the check at line 14 must be performed for all the border routers on the path between the source and the destination, not only for the first border router on the path BR. Not checking all border routers on the path might miss some suboptimal paths.

Note that in some situations the previous code can detect deflection that does not really arise in the network. This happens when there is a border router on the path in the considered area, between the border router that causes deflection and

```
01: procedure check_deflected_path(area) {
02:   for each border router ER in area {
03:     rspt = rspt(ER, area);
04:     for each border router BR != ER in rspt {
05:       for each son of BR in rspt {
06:         for each inter-area-route route in son {
07:           if exit_router(route) == ER {
08:             route_in_br = route(BR,destination(route));
09:             if (link(route_in_br) does not belong to area) {
10:               deflection!
11:           }//end if exit_router()
12:         }//end for each inter-area-route
13:       }//end for each son
14:
15:       if (deflection was not found) {
16:         if (area != backbone) {
17:           if (cost(BR,ER,backbone) > cost(BR,ER,area)) {
18:             fix virtual link on BR ER!
19:           }//end if cost
20:         }//end if area
21:       }//end if deflection
22:     }//end for each border router BR
23:   }//end for each border router ER
24: }
```

Figure 5.8. Pseudo code to dectect deflection and suboptimal paths in a given area.

the exit router. This happens, for instance, in figure 5.9 where router `10.0.0.3` (S) has a deflected path towards `10.0.0.15` (D). Border router `10.0.0.1` causes deflection and `10.0.0.2` is the exit router. However the check will detect also a deflected path between S and D where `10.0.0.13` is the border router that causes deflection. To avoid this anomalous detection we need an enforced order of visit of the RSPT. We need to consider first the border router that is deeper within the tree (we call it the *start BR*). When its children are visited we check if the border router causes deflection. If this does not happen we have also to go back towards the root of the tree performing the same check for the border router on the path. When we found it we stop the visit toward the root and register the deflected path. Finally we mark the start BR as visited. When a BR is considered in order to visit its children we visit only not visited node.

The algorithm illustred above requires to have a specific order to visit border routers. The order consists simply in the same order as nodes are added to the RSPT. This assumption is motivated by the fact that considering two border routers that

are on the path beetwen the exit router (for which we compute the RSPT) and the source (for which we check if there is some deflected path) the border router located deeper in the RSPT will have a higher cost compared to the other: thus the router deeper in the RSPT will be added after the others.

A way to obtain the order of visiting of the border routers in the area can be the following: during the RSPT computation, when a border router is added to the RSPT a reference to it can be added in queue into a list. At the end of the computation the list will contain the reverse order in which the router must belong to. Simply visiting the list from the end to the beginning we have a border routers' visit order that allows to solve the problem of anomalous detection of deflected path.

Note that in this way we can visit successively border routers that are not on the same path from the root of the RSPT and the examined destination. This is not a problem. The objective of this approach is only to avoid anomalous detection enforcing a special order in the check of the border router.

5.4.3 A case study

We implemented the algorithm to detect deflection and suboptimal paths in C-BGP. We used this implementation to check the sample network in figure 5.9. The output given by our prototype is illustrated in figure 5.4.3.

The result of the check in figure 5.4.3 shows that deflection arises in area 1. At line 9 we can see that the border routers `10.0.0.5` and `10.0.0.6` cause deflection. The former is selected as exit router by router `10.0.0.11` to reach destinations `10.0.0.4` (line 12). In figure 5.9 we have reported the routing path computed by `10.0.0.11` to reach `10.0.0.4` with a dashed line. This passes through `10.0.0.6`. The deflected path is caused by `10.0.0.5` and it is plotted as a dotted line. The reader can verify on the weight of the links that the deflected path is more costly than the routing path computed by `10.0.0.11`. The same is true for destinations `10.0.0.2` and `10.0.0.15`.

In the same way, all other rows of the output describe the source of the deflection. For instance, line 4 describes how the same border routers `10.0.0.5` and `10.0.0.6` cause deflection when `10.0.0.5` is selected as exit router by `10.0.0.9`. The output of the check drives us to ask for a virtual link to be established in area 2.

No deflection is detected by the prototype in area 2. In fact, the conditions for deflection to arise are not verified. The reader can check that no router in area 2 reaches a border router with a path on which another border router is present.

In the backbone on the other hand, deflection arises. Here the couple of routers `10.0.0.1` and `10.0.0.2` cause deflection when the latter is selected as exit router (line 27). The deflected path is beetwen `10.0.0.3` (but also `10.0.0.3`) and `10.0.0.15`. The routing path is shown in figure 5.9 as a dashed line, while the forwarding path is plotted as a dotted line.

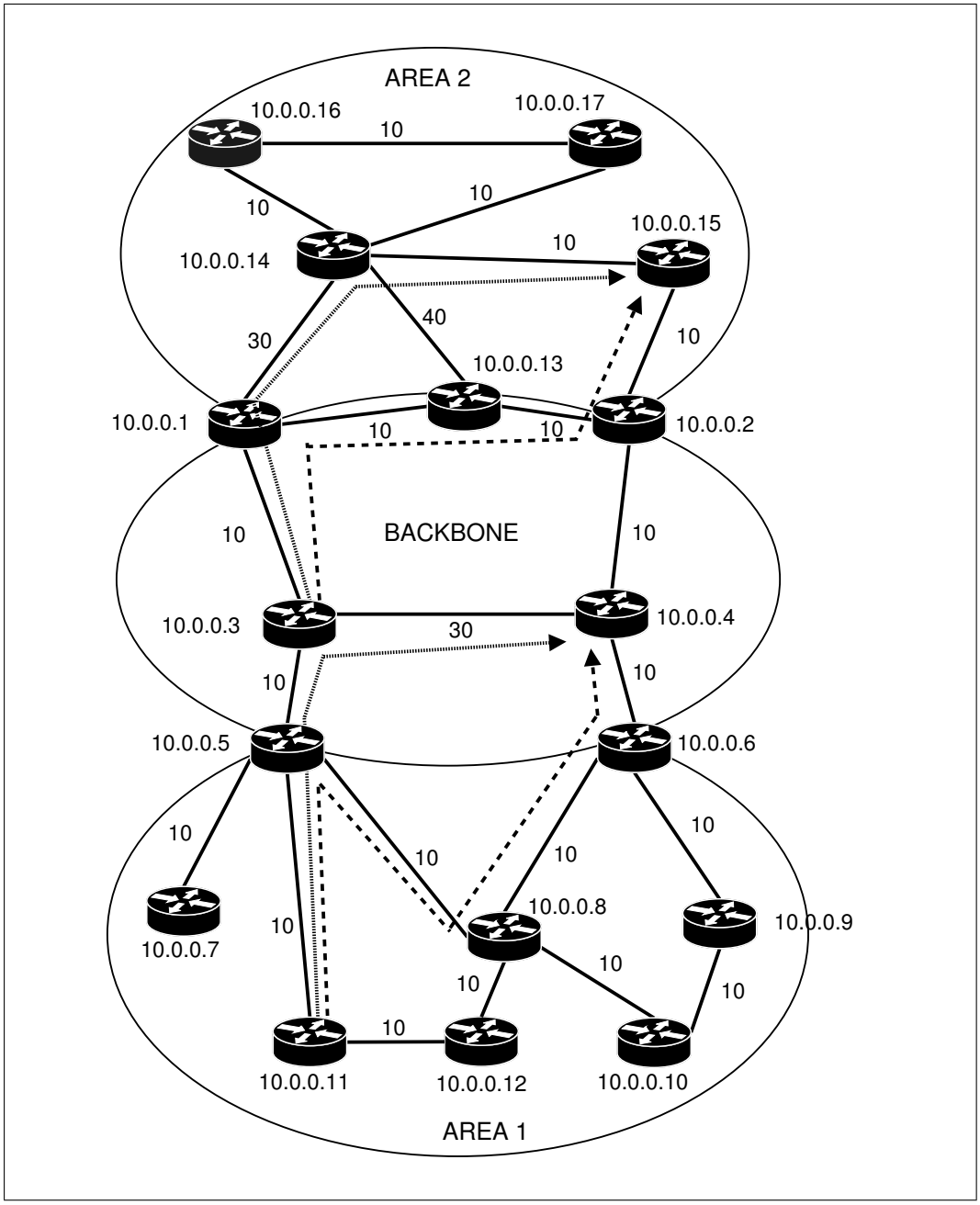

Figure 5.9. The case study network on which the prototype was tested.

5.4.4 Another approach

The proposed algorithm tries to be general and to provide all the available information about deflected paths: the source, the destination and the border routers

```
01: ==================================
02: # DEFLECTION IN AREA 1
03: #
04: #> ER: 10.0.0.5 - BR:10.0.0.6
05: #  * R: 10.0.0.9
06: #       - D:  10.0.0.1/32
07: #       - D:  10.0.0.3/32
08: #
09: #> ER: 10.0.0.6 - BR:10.0.0.5
10: #  * R: 10.0.0.11
11: #       - D:  10.0.0.2/32
12: #       - D:  10.0.0.4/32
13: #       - D:  10.0.0.15/32
14: #  * R: 10.0.0.7
15: #       - D:  10.0.0.2/32
16: #       - D:  10.0.0.4/32
17: #       - D:  10.0.0.15/32
18: ==================================
19:
20: ==================================
21: # NO DEFLECTION IN AREA 2
22: ==================================
23:
24: ==================================
25: # DEFLECTION IN THE BACKBONE
26: #
27: #> ER: 10.0.0.2 - BR:10.0.0.1
28: #  * R: 10.0.0.3
29: #       - D:  10.0.0.15/32
30: #  * R: 10.0.0.5
31: #       - D:  10.0.0.15/32
32: #
33:==================================
```

Figure 5.10. The output of the prototype on the case study.

that cause deflection. However it can be simpler to follow the approach illustrated in figure 5.11 that combines the two seperate checks for suboptimal paths and deflection. Starting from the OSPF configuration of a given network (block 1) the suboptimal paths are detected by the check discussed in 5.3.2 (block 2). Then the output of this check permits to establish the virtual links useful to solve some of the suboptimal paths (block 3). Then the deflection check based on the traceroute is performed (block 4): this permits to detect the remaining deflected paths caused by the unsolvable suboptimal paths.

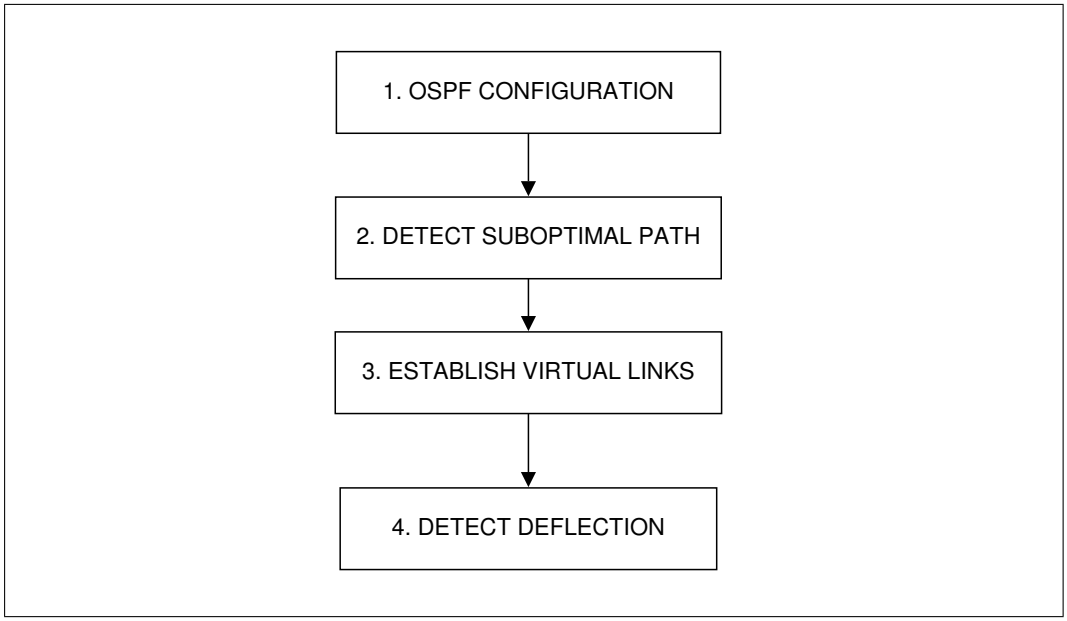

Figure 5.11. Alternative approach to the detection of deflection.

5.5 Conclusions

In this chapter we analyzed the way OSPF routers compute their routes, having as consequence forwarding deflection in some cases. Deflection is an inconsistency between the routing path and the forwarding path beetwen two nodes. This inconsistency beetwen the routing plane and the forwarding plane could have some repercussions on other layers of the network stack.

Deflection is strictly linked with another problem: the presence of suboptimal routing paths, i.e. paths chosen even though there is a better path available in the network. We discussed how the causes of the two problems are actually the same: the preference of intra-area paths over inter-area paths and the rule that prevents a border router from considering Summary LSAs and ASBR Summary LSAs from areas different from the backbone.

We also illustrated how to solve some situations with suboptimal paths, and thus deflection, can be solved by establishing virtual links. Scenarios where a suboptimal path is found in the backbone area are unsolvable with virtual links as they exist today. Today, virtual links cannot be installed in the backbone.

We have proposed two simple ways to check either for deflection or for suboptimal paths, separately. We illustrated the advantages of each approach showing that none of them gives all the available information about deflection and suboptimal paths at once. However, each of them has properties that make it more adapted to some implementation requirements.

We have given a necessary and sufficient condition for *deflection* to arise in a domain, and have described how to detect it. The goal was to provide a tool that, using our OSPF model, permits to decide where to establish OSPF virtual links to prevent deflection from arising. At the same time, we want to obtain detailed information (the source, the destination of the deflected path and the border routers causing deflection) about deflection when it cannot be solved using virtual links. The detection of deflection was implemented in C-BGP and was used on a particular network example. The results were also illustrated. Finally, an alternative approach to the problem was proposed.

Chapter 6

Conclusions and further works

In this work, we have studied the multi-area architecture implemented in the OSPF protocol. Firstly, we have discussed the link state approach finding the motivation for which multi-area was introduced. The main reason for multi-area is to bring scalability to the link-state approach. Then OSPF was analyzed discussing the rules it uses to compute the routes. Advanced features of the protocol, like virtual links, stub areas and address summarization were also illustrated.

We have proposed a model for the OSPF protocol. Given a network and its OSPF multi-area configuration the model is able to supply the IGP routing table for all the routers of the network. We found a way to reproduce the behavior of the protocol performing routes' computation without having to flood the Link State Advertisements. We discussed our choices of design and have motivated each of them.

A prototype of the model was implemented in C-BGP, an efficient BGP simulator. We have discussed our implementation and shown that some modifications to the topology model of C-BGP were required. We have also presented the way we have integrated our model with the rest of the C-BGP environment.

Finally, we have discussed the problem of the forwarding deflection in OSPF. Deflection is an inconsistency between the routing plane and the forwarding plane. Deflection arises due to how OSPF computes its routes. We have shown how deflection is linked with the presence of suboptimal routing paths in the network. We have also discussed several approaches to detect deflection and suboptimal paths on a given network: one was implemented and it is actually available in C-BGP. Finally we have proposed a possible solution for the deflection that consists in introducing a more general concept of virtual link in OSPF. The discussion of the details of this solution are delayed to future works.

The model we have proposed will allow to this simulator to model scenarios where the multi-area OSPF is used in an AS. This should allow us to obtain results more coherent with the routes of the real network. The deflection check can help

operators to detect possible deflection situations in a network providing minimal information needed to solve it. At design time it can be used to be sure that the configuration of the network does not allow deflection to arise. In future works the model could be useful from an operational viewpoint in order to find inconsistencies in the configuration of a multi-area OSPF domain.

However, we think that the main contribution of this work consists in showing a part of the complexity of the multi-area architecture. This complexity leads to unexpected behaviors like, for instance, forwarding deflection. Further works are required to show the impact of deflection in real situations. We need to understand if it represents an extreme possibility or a frequent situation. The algorithm for deflection we have implemented in C-BGP could be used for this purpose.

By the way we can state that choosing to use the multi-area is a question of trade-off between the advantages it supplies and the complexity it introduces. This direction could be interesting to evaluate the real advantages introduced in terms of scalability from the viewpoint of the link state protocol. Today, several ASs use link state IGP with a single area configured. Since some of them are among the biggest in the world, we can suppose that the real motivation to use the multi-area is not the scalability requirements in terms of IGP.

Talking with operators of ASs that use the multi-area OSPF, we could discover that the value of the multi-area are in the *divide and conqueror* approach but mainly in terms of network management. Splitting the network in smaller portions (the areas) an operator can decentralize the control and the management of the network. Working on a single area he does not have to know the details of the rest of the domain in terms, for instance, of routers' configuration. Thus the real scalability that today could be a value for the operators seems to be the one linked to the management while the one that works on the link state limitations might be less important.

In conclusion we can say that there seems to be a paradox in the use of the multi-area: it should decentralize the network management, as happens today in some AS, but it can make complex and unforeseable the behavior of the routing, like we showed with deflection. Thus there is a trade-off in choosing to use or not the multi-area. We think it is very hard to have an a priori way to decide when the multi-area can increase or reduce complexity. The choice should be performed carefully by the operator considering the complexity of its network. If this does not present issues in terms of management and there are no other strong reasons to use multi-area, maybe it is better to not use the multi-area mantaining simple the configuration of the network.

Appendix A

Validation script for the model

A.1 Test for scenario 1

```
# Test of ospf model of C-BGP
#
# Author: Stefano Iasi
# October 2005
#
# To execute this script run
# ./cbgp -c test-ospf.cli

#Area 0 with common prefix 192.168.0/24
net add node 192.168.0.3
net add node 192.168.0.4
net add node 192.168.0.5
net add node 192.168.0.6
net add node 192.168.0.7
net add node 192.168.0.10

#Area 1 with common prefix 192.168.16/24
net add node 192.168.16.1
net add node 192.168.16.2

net add subnet 192.168.17/24 stub
net add subnet 192.168.18/24 stub
net add subnet 192.168.19/24 transit
net add subnet 192.168.20/24 stub

#Area 2 with common prefix 192.168.32/24
net add node 192.168.32.8
net add node 192.168.32.11

net add subnet 192.168.33/24 transit
net add subnet 192.168.34/24 stub
```

```
net add subnet 192.168.35/24 transit

#Area 3 with common prefix 192.168.64/24
net add node 192.168.64.9
net add node 192.168.64.12

net add subnet 192.168.65/24 transit
net add subnet 192.168.66/24 stub
net add subnet 192.168.67/24 stub

#Area 0 links
net add link 192.168.0.4 192.168.0.5 8
net add link 192.168.0.3 192.168.0.6 8
net link 192.168.0.6 192.168.0.3 igp-weight 6
net add link 192.168.0.5 192.168.0.6 7
net link 192.168.0.6 192.168.0.5 igp-weight 6
net add link 192.168.0.5 192.168.0.7 6
net add link 192.168.0.6 192.168.0.10 7
net link 192.168.0.10 192.168.0.6 igp-weight 5

#configure interface on point-to-point link
net link 192.168.0.10 192.168.0.6 ipprefix 192.168.1.5/32
net link 192.168.0.6 192.168.0.10 ipprefix 192.168.1.6/32
#configure LIS on point-to-point link
net link 192.168.0.7 192.168.0.5 ipprefix 192.168.1.9/30
net link 192.168.0.5 192.168.0.7 ipprefix 192.168.1.10/30

#Area 1 links
net add link 192.168.16.1 192.168.17.1/24 3
net add link 192.168.16.1 192.168.19.1/24 1
net add link 192.168.16.2 192.168.18.2/24 3
net add link 192.168.16.2 192.168.19.2/24 1
net add link 192.168.0.3 192.168.20.3/24 2
net add link 192.168.0.3 192.168.19.3/24 1
net add link 192.168.0.4 192.168.19.4/24 1

#Area 2 links
net add link 192.168.0.7 192.168.33.7/24 1
net add link 192.168.0.10 192.168.33.10/24 1
net add link 192.168.0.10 192.168.35.10/24 3
net add link 192.168.32.8 192.168.33.8/24 1
net add link 192.168.32.8 192.168.34.8/24 4
net add link 192.168.32.11 192.168.35.11/24 2

#Area 3 links
net add link 192.168.32.11 192.168.65.11/24 1
net add link 192.168.64.9 192.168.65.9/24 1
net add link 192.168.64.9 192.168.67.9/24 3
net add link 192.168.64.12 192.168.65.12/24 1
```

```
net add link 192.168.64.12 192.168.66.12/24 2

#OSPF configuration - Area 0 nodes
net node 192.168.0.3 ospf area 0
net node 192.168.0.4 ospf area 0
net node 192.168.0.5 ospf area 0
net node 192.168.0.6 ospf area 0
net node 192.168.0.7 ospf area 0
net node 192.168.0.10 ospf area 0

#OSPF configuration - Area 1 nodes
net node 192.168.16.1 ospf area 1
net node 192.168.16.2 ospf area 1
net node 192.168.0.3 ospf area 1
net node 192.168.0.4 ospf area 1

net subnet 192.168.17/24 ospf area 1
net subnet 192.168.18/24 ospf area 1
net subnet 192.168.19/24 ospf area 1
net subnet 192.168.20/24 ospf area 1

#OSPF configuration - Area 2 nodes
net node 192.168.32.8 ospf area 2
net node 192.168.32.11 ospf area 2
net node 192.168.0.7 ospf area 2
net node 192.168.0.10 ospf area 2

net subnet 192.168.33/24 ospf area 2
net subnet 192.168.34/24 ospf area 2
net subnet 192.168.35/24 ospf area 2

#OSPF configuration - Area 3 nodes
net node 192.168.64.9 ospf area 3
net node 192.168.64.12 ospf area 3
net node 192.168.32.11 ospf area 3

net subnet 192.168.65/24 ospf area 3
net subnet 192.168.66/24 ospf area 3
net subnet 192.168.67/24 ospf area 3

#OSPF configuration - Area 1 links
net node 192.168.16.1 link 192.168.17.1/24 ospf area 1
net node 192.168.16.1 link 192.168.19.1/24 ospf area 1
net node 192.168.16.2 link 192.168.18.2/24 ospf area 1
net node 192.168.16.2 link 192.168.19.2/24 ospf area 1
net node 192.168.0.3  link 192.168.20.3/24 ospf area 1
net node 192.168.0.3  link 192.168.19.3/24 ospf area 1

#OSPF configuration - Area 0 links
```

```
net node 192.168.0.4 link 192.168.0.5 ospf area 0
net node 192.168.0.3 link 192.168.0.6 ospf area 0
net node 192.168.0.5 link 192.168.0.6 ospf area 0
net node 192.168.0.5 link 192.168.0.7 ospf area 0
net node 192.168.0.6 link 192.168.0.10 ospf area 0

#OSPF configuration - Area 2 links
net node 192.168.0.7 link 192.168.33.7/24 ospf area 2
net node 192.168.0.10 link 192.168.33.10/24 ospf area 2
net node 192.168.0.10 link 192.168.35.10/24 ospf area 2
net node 192.168.32.8 link 192.168.33.8/24 ospf area 2
net node 192.168.32.8 link 192.168.34.8/24 ospf area 2
net node 192.168.32.11 link 192.168.35.11/24 ospf area 2

#OSPF configuration - Area 3 links
net node 192.168.32.11 link 192.168.65.11/24 ospf area 3
net node 192.168.64.9 link 192.168.65.9/24 ospf area 3
net node 192.168.64.9 link 192.168.67.9/24 ospf area 3
net node 192.168.64.12 link 192.168.65.12/24 ospf area 3
net node 192.168.64.12 link 192.168.66.12/24 ospf area 3

#OSPF configuration - OSPF domain
net add domain 1 ospf

net node 192.168.0.3 ospf domain 1
net node 192.168.0.4 ospf domain 1
net node 192.168.0.5 ospf domain 1
net node 192.168.0.6 ospf domain 1
net node 192.168.0.7 ospf domain 1
net node 192.168.0.10 ospf domain 1
net node 192.168.16.1 ospf domain 1
net node 192.168.16.2 ospf domain 1
net node 192.168.32.8 ospf domain 1
net node 192.168.32.11 ospf domain 1
net node 192.168.64.9 ospf domain 1
net node 192.168.64.12 ospf domain 1

#Computation od routinig tables
net domain 1 compute

#Show routing tables
net node 192.168.0.3 show rt *
```

A.2 Test for scenario 2

```
# Test of ospf model of C-BGP
#
```

```
# Author: Stefano Iasi
# October 2005
#
# To execute this script run
# ./cbgp -c test-ospf.cli

#Area 0 with common prefix 192.168.0/24
net add node 192.168.0.3
net add node 192.168.0.4
net add node 192.168.0.5
net add node 192.168.0.6
net add node 192.168.0.7
net add node 192.168.0.10

#Area 1 with common prefix 192.168.16/24
net add node 192.168.16.1
net add node 192.168.16.2

net add subnet 192.168.17/24 stub
net add subnet 192.168.18/24 stub
net add subnet 192.168.19/24 transit
net add subnet 192.168.20/24 stub

#Area 2 with common prefix 192.168.32/24
net add node 192.168.32.8
net add node 192.168.32.11

net add subnet 192.168.33/24 transit
net add subnet 192.168.34/24 stub
net add subnet 192.168.35/24 transit

#Area 3 with common prefix 192.168.64/24
net add node 192.168.64.9
net add node 192.168.64.12

net add subnet 192.168.65/24 transit
net add subnet 192.168.66/24 stub
net add subnet 192.168.67/24 stub

#Area 0 links
net add link 192.168.0.4 192.168.0.5 8
net add link 192.168.0.3 192.168.0.6 8
net link 192.168.0.6 192.168.0.3 igp-weight 6
net add link 192.168.0.5 192.168.0.6 7
net link 192.168.0.6 192.168.0.5 igp-weight 6
net add link 192.168.0.5 192.168.0.7 6
net add link 192.168.0.6 192.168.0.10 7
net link 192.168.0.10 192.168.0.6 igp-weight 5
net add link 192.168.0.3 192.168.32.11 10
```

```
net link 192.168.32.11 192.168.0.3 igp-weight 3

#configure interface on point-to-point link
#net link 192.168.0.10 192.168.0.6 ipprefix 192.168.1.5/32
#net link 192.168.0.6 192.168.0.10 ipprefix 192.168.1.6/32
#configure LIS on point-to-point link
#net link 192.168.0.7 192.168.0.5 ipprefix 192.168.1.9/30
#net link 192.168.0.5 192.168.0.7 ipprefix 192.168.1.10/30

#Area 1 links
net add link 192.168.16.1 192.168.17.1/24 3
net add link 192.168.16.1 192.168.19.1/24 1
net add link 192.168.16.2 192.168.18.2/24 3
net add link 192.168.16.2 192.168.19.2/24 1
net add link 192.168.0.3 192.168.20.3/24 2
net add link 192.168.0.3 192.168.19.3/24 1
net add link 192.168.0.4 192.168.19.4/24 1

#Area 2 links
net add link 192.168.0.7 192.168.33.7/24 1
net add link 192.168.0.10 192.168.33.10/24 1
net add link 192.168.0.10 192.168.35.10/24 3
net add link 192.168.32.8 192.168.33.8/24 1
net add link 192.168.32.8 192.168.34.8/24 4
net add link 192.168.32.11 192.168.35.11/24 2

#Area 3 links
net add link 192.168.32.11 192.168.65.11/24 1
net add link 192.168.64.9 192.168.65.9/24 1
net add link 192.168.64.9 192.168.67.9/24 3
net add link 192.168.64.12 192.168.65.12/24 1
net add link 192.168.64.12 192.168.66.12/24 2

#OSPF configuration - Area 0
net node 192.168.0.3 ospf area 0
net node 192.168.0.4 ospf area 0
net node 192.168.0.5 ospf area 0
net node 192.168.0.6 ospf area 0
net node 192.168.0.7 ospf area 0
net node 192.168.0.10 ospf area 0
net node 192.168.32.11 ospf area 0

#OSPF configuration - Area 1
net node 192.168.16.1 ospf area 1
net node 192.168.16.2 ospf area 1
net node 192.168.0.3 ospf area 1
net node 192.168.0.4 ospf area 1

net subnet 192.168.17/24 ospf area 1
```

```
net subnet 192.168.18/24 ospf area 1
net subnet 192.168.19/24 ospf area 1
net subnet 192.168.20/24 ospf area 1

#OSPF configuration - Area 2
net node 192.168.32.8 ospf area 2
net node 192.168.32.11 ospf area 2
net node 192.168.0.7 ospf area 2
net node 192.168.0.10 ospf area 2

net subnet 192.168.33/24 ospf area 2
net subnet 192.168.34/24 ospf area 2
net subnet 192.168.35/24 ospf area 2

#OSPF configuration - Area 3
net node 192.168.64.9 ospf area 3
net node 192.168.64.12 ospf area 3
net node 192.168.32.11 ospf area 3

net subnet 192.168.65/24 ospf area 3
net subnet 192.168.66/24 ospf area 3
net subnet 192.168.67/24 ospf area 3

#OSPF configuration - Area 1 links
net node 192.168.16.1 link 192.168.17.1/24 ospf area 1
net node 192.168.16.1 link 192.168.19.1/24 ospf area 1
net node 192.168.16.2 link 192.168.18.2/24 ospf area 1
net node 192.168.16.2 link 192.168.19.2/24 ospf area 1
net node 192.168.0.3  link 192.168.20.3/24 ospf area 1
net node 192.168.0.3  link 192.168.19.3/24 ospf area 1

#OSPF configuration - Area 0 links
net node 192.168.0.4 link 192.168.0.5 ospf area 0
net node 192.168.0.3 link 192.168.0.6 ospf area 0
net node 192.168.0.5 link 192.168.0.6 ospf area 0
net node 192.168.0.5 link 192.168.0.7 ospf area 0
net node 192.168.0.6 link 192.168.0.10 ospf area 0
net node 192.168.0.3 link 192.168.32.11 ospf area 0

#OSPF configuration - Area 2 links
net node 192.168.0.7 link 192.168.33.7/24 ospf area 2
net node 192.168.0.10 link 192.168.33.10/24 ospf area 2
net node 192.168.0.10 link 192.168.35.10/24 ospf area 2
net node 192.168.32.8 link 192.168.33.8/24 ospf area 2
net node 192.168.32.8 link 192.168.34.8/24 ospf area 2
net node 192.168.32.11 link 192.168.35.11/24 ospf area 2

#OSPF configuration - Area 3 links
net node 192.168.32.11 link 192.168.65.11/24 ospf area 3
```

```
net node 192.168.64.9 link 192.168.65.9/24 ospf area 3
net node 192.168.64.9 link 192.168.67.9/24 ospf area 3
net node 192.168.64.12 link 192.168.65.12/24 ospf area 3
net node 192.168.64.12 link 192.168.66.12/24 ospf area 3

#OSPF configuration - OSPF domain
net add domain 1 ospf

net node 192.168.0.3 ospf domain 1
net node 192.168.0.4 ospf domain 1
net node 192.168.0.5 ospf domain 1
net node 192.168.0.6 ospf domain 1
net node 192.168.0.7 ospf domain 1
net node 192.168.0.10 ospf domain 1
net node 192.168.16.1 ospf domain 1
net node 192.168.16.2 ospf domain 1
net node 192.168.32.8 ospf domain 1
net node 192.168.32.11 ospf domain 1
net node 192.168.64.9 ospf domain 1
net node 192.168.64.12 ospf domain 1

#Computation od routinig tables
net domain 1 compute

net node 192.168.0.3 show rt *
```

Appendix B

CLI commands

This appendix contains a description of all the new commands introduced in the CLI of C-BGP during the development of the prototype of the OSPF model.

B.1 Commands for the topology model

CLI command	Description
`net add subnet <prefix> <transit|stub>`	Adds a subnet in the topology model. Assigns the given `prefix` to the subnet. Declares the subnet as `transit` or `stub`.
`net add link <ip-address> <ip-prefix> <weight>`	Adds a link between the router identified by `ip-address` and the subnet identified by `prefix` with a weight specified by `<weight>`. The prefix must specify a 32 bit IP address length that is used as IP interface on the link.
`net link <ip-address-src> <ip-address-dest> ipprefix <ip-prefix>`	Adds interfaces on the link between the routers identified by the IP addresses `<ip-address-src>` `<ip-address-dest>`. If the given `<ip-prefix>` is a `/32` prefix only an interface is added on the link. Otherwise the `<ip-prefix>` can be only a /30 prefix and, in this case, also a LIS is configured on the link. In each case modifications are performed only on the interface of `<ip-address-src>`. The IP address `<ip-interface-src>` is assigned to the interface of node `<ip-address-src>` while the `<ip-interface-dst>` is assigned to the interface of node `<ip-address-dest>`.

B.2 Commands for the multi-area OSPF configuration

CLI command	Description
`net node <ip-address> ospf area <area-id>`	Assigns the router identified by the IP address `<ip-address>` to the area `<area-id>`. A router may belong to multiple areas.
`net subnet <ip-prefix> ospf area <area-id>`	Assigns the subnet identified by the IP address `<ip-prefix>` to the area `<area-id>`.
`net node <ip-address-src> link <ip-prefix-dst> ospf area <area-id>`	Assigns the link from the source router (identified by the IP address `<ip-address-src>`) to the destination node (identified by the prefix `<ip-prefix-dest>`) to the area `<area-id>`. The destination can be a router or a subnet. In the first case the prefix is a /32 prefix.
`net add domain <igp-domain-id> <igp|ospf>`	Declares a new IGP domain in the network. The domain is identified by the `<IGP-domain-id>`. The type of the domain can be `igp` ot `ospf`. In the first case a model with a single area is used. In the second case the ospf model is used.
`net node <ip-address> ospf domain <igp-domain-id>`	Assigns the node identified by the address `<ip-address>` to a given IGP domain identified by `<igp-domain-id>`. The domain must be declared in the network. If the domain is of type `igp`, the previous model of C-BGP for the domain is used. Otherwise if the value `ospf` is specified our model is used.
`net domain <domain-id> compute`	Computes the OSPF routes for the domain specified by the given `<domain-id>`.

Bibliography

[1] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). Internet draft, draft-ietf-idr-bgp4-22.txt, work in progress, October 2003.

[2] B. Quoitin. C-BGP, an efficient BGP simulator. `http://cbgp.info.ucl.ac.be/`, September 2003.

[3] J. Moy. OSPF Version 2. Internet Engineering Task Force, RFC2328, April 1998.

[4] Edsger. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[5] M. Baldi and P. Nicoletti. *Internetworking*. McGraw-Hill, 1999.

[6] Mikkel Thorup. On ram priority queues. In *SODA*, pages 59–67, 1996.

[7] J. Stewart. *BGP4 : interdomain routing in the Internet*. Addison Wesley, 1999.

[8] B. J. Premore. SSF Implementations of BGP-4. available from `http://www.cs.dartmouth.edu/~beej/bgp/`, 2001.

[9] H. Tyan. *Design, realization and evaluation of a component-based compositional software architecture for network simulation*. PhD thesis, Ohio State University, 2002.

[10] B. Quoitin and S. Uhlig. Modeling the routing of an Autonomous System with C-BGP. *IEEE Network Magazine*, November 2005.

[11] T. Griffin and G. Wilfong. An analysis of BGP convergence properties. In *Proc. of ACM SIGCOMM'99*, September 1999.

[12] D. Yeung A. Zinin, A. Lindem. Alternative implementations of ospf area border routers. Internet Engineering Task Force, RFC3509, April 2003.

[13] B. Quoitin. IGen, topology generation through network design heuristics. `http://www.info.ucl.ac.be/~bqu/igen/`, November 2005.

[14] B. Fortz, J. Rexford, and M. Thorup. Traffic engineering with traditional IP routing protocols. *IEEE Communications Magazine*, October 2002.

[15] D. Katz, K. Kompella, and D. Yeung. Traffic Engineering (TE) Extensions to OSPF Version 2. Internet Engineering Task Force, RFC3630, September 2003.

[16] Cisco. BGP Best Path Selection Algorithm. http://www.cisco.com/warp/public/459/25.shtml, 2002.

[17] A. Retana R. White. *IS-IS: Deployment in IP Networks*. Addison-Wesley Professional, 2002.

[18] S. Iasi P. Francois and Steve Uhlig. Forwarding deflection in multi-area OSPF. Abstract for CoNEXT 2005, October 2005.

[19] S. Shamim. OSPF: Frequently asked questions. `http://www.cisco.com/warp/public/104/9.html#q13`, August 2005.

[20] Cisco Systems Inc. OSPF on line documentation. `http://www.cisco.com/en/US/tech/tk365/tk480/tsd_technology_support_sub-%protocol_home.html`.

[21] J. Doyle. *Routing TCP/IP*. Cisco Press, Indianapolis, IN, 1998.

[22] TOTEM: TOolbox for Traffic Engineering Methods. `http://totem.info.ucl.ac.be/index.html`, March 2005.