Software Router: OSPF Performance Evaluation and Enhancement through a New Multi-path Incremental Algorithm

PhD candidate: Antonio Cianfrani

Tutor: Prof. Vincenzo Eramo

Dottorato in Ingegneria dell'Informazione e della Comunicazione (XX Ciclo)

Infocom Dept. - University of Roma "*Sapienza*"

SAPIENZA
UNIVERSITÀ DI ROMA

Contents

Introduction **7**

1 Software Routers **11**
1.1 Introduction 11
1.2 Software Router motivations 12
1.3 Quagga and Xorp 16
1.4 Open Shortest Path First (OSPF) protocol 19

2 OSPF performance evaluation **21**
2.1 OSPF test methodologies 21
2.1.1 OSPF performance indexes 23
2.2 SPF computation time 24
2.2.1 Test-bed description 25
2.2.2 Results 28
2.3 Optimization of the SPF Computation Time in Quagga 29
2.3.1 A Binary Heap to Implement the Candidate List . . . 31
2.3.2 The new Lookup Operation in the Candidate List . . . 33
2.3.3 Results for New Quagga Version 33
2.4 Switching time 35
2.4.1 Test-bed description 35
2.4.2 Results 37
2.5 Conclusions 40

3 A new multi-path incremental algorithm **42**
3.1 Incremental Algorithms 43
3.1.1 Motivations 44

3.1.2 Previous works 45
3.1.3 Incremental algorithms in a networking scenario 46
3.2 Our Multi-path Incremental Algorithm 47
3.2.1 Terminology 47
3.2.2 Algorithm description 48
3.2.3 Example of edge deletion 55
3.2.4 Algorithm extensions: weight and multiple edge modifications 58
3.2.5 Algorithmic complexity 59
3.2.5.1 Complexity of the "insertion of an edge" phase 60
3.2.5.2 Complexity of the "deletion of an edge" phase 60
3.2.5.3 Complexity of the "main" phase 61
3.3 Performance evaluation and results 62
3.3.1 Black box measurements 63
3.3.2 White box measurements 66
3.4 Conclusions 73

Conclusions **75**

List of Figures

1.1 Quagga architecture. 17
1.2 Xorp architecture. 18

2.1 Test-bed for the evaluation of the SPF computation time. . . . 25
2.2 Time measure T_{totSPF}. 26
2.3 Time measure T_{ov}. 27
2.4 Example of a fully meshed topology with 4 routers. 28
2.5 The SPF Computation Time in a Cisco 2801 router and a Software Router. 30
2.6 The SPF Computation Time in a Cisco 2801 router and a Software Router with new Quagga version. 34
2.7 Test-bed for the evaluation of the Switching time. 35
2.8 Network topology emulated to perform Switching time test. . 36
2.9 The Switching Time in a Cisco 2801 router and Software Routers. 39
2.10 The Switching Time in a Cisco 2801 router for different traffic rate F_p. 40

3.1 Example of edge deletion. a) Network graph: the solid thick arrows represents all the SP(G) edges. b) Set of affected nodes after edge deletion. c) and d) SP(G) during the initialization phase of the algorithm. e) Final SP(G) after that the operations of the algorithm have been performed. 57
3.2 SPF computation time in Cisco 2801 and Quagga in the case of link failure for Verio topology. 64

3.3 SPF computation time in Cisco 2801 and Quagga in the case of link failure for AT&T topology. 65
3.4 Performance evaluation of the uni-path and multi-path incremental algorithms in the case of link failure for Verio topology. 67
3.5 Performance comparison of the uni-path and multi-path incremental algorithms in the case of link failure for Verio topology. The SPF computation time of the first 50 links of Figure 9 are reported. 69
3.6 Performance evaluation of the uni-path and multi-path incremental algorithms in the case of link failure for AT&T topology. 71
3.7 Performance comparison of the uni-path and multi-path incremental algorithms in the case of link failure for AT&T topology. The SPF computation time of the first 50 links of Figure 9 are reported. 72

Introduction

During my Ph.D thesis I have studied new devices for the Internet network infrastructure: Software Routers. Software routers are achieving a great interest in the last years because they represent an even more realistic alternative to commercial routers. Software Routers are realized using the hardware of a Personal Computer, an open-source operating system and an open-source routing software. In my work I have always used Linux operating system because it is becoming an interesting competitor for Windows, with a number of users constantly growing. Quagga and Xorp represent the most common open-source routing software and in my work I have studied and tested both.

The interest in the Software Router employment has an economic motivation and a research motivation. From an economic point of view it is important to highlight that the router market, like all networking equipment one, is characterized by the development of proprietary architectures and by a reduced number of companies: these aspects has led to commercial practices not based on free competition and so the final cost of such devices is really high with respect to equipment complexity and performance. The introduction of Software Routers can lead to an open multi vendor market. In fact PCs hardware is available at low cost, their architectures are well documented and their performance evolution is guaranteed. Another important aspect of Software Routers is that software is free and documented while in the case of commercial devices software is not available. Of course it is important to evaluate Software Router performance; in this way such a device has to be conformed with protocols it implements, it has to communicate with different Software Routers and with commercial devices, and it should have performance at least comparable with ones of a commercial device. So

before introducing a Software Router in a real environment, a testing activity need to be executed. The performance evaluation of Software Routers represents the first activity of my work.

From a research point of view a Software Router is really useful to test new networking solutions on a real device. In particular if a new protocol or algorithm has been defined it is possible, thanks to open-source nature of Software Router, to implement it on such a device and to evaluate benefits and problems of the new solution in a real environment. Of course in my work this is the most interesting aspect of Software Routers: I have used this device as an instrument to evaluate a new solution I have realized. In particular I have proposed a new routing algorithm to be used inside Open Shortest Path First (OSPF) protocol instead of Dijkstra algorithm to speed up routing table calculation after a topological modification. In this way it is possible to reduce network convergence, which is the time for the network to react when a topological modification happens, and so to improve network performance. The requirement for a more responsive network is due to the proliferation of real-time services, which need high performance in terms of packets lost and delay. So to evaluate the performance of my algorithm I have implemented it in the OSPF code of a Software Router and I have evaluated OSPF performance of the Software Router in a real network topology when my algorithm is used to compute new routing table after topological modifications. It has been also possible to compare my algorithm performance with ones of a commercial device, in particular a Cisco router, and with ones of an algorithm proposed in literature.

My thesis is organized into three Chapters. In Chapter 1 I will introduce Software Routers explaining motivations for their introduction and describing Quagga and Xorp software; I will also briefly summarize most interesting OSPF features. In Chapter 2 I will describe OSPF test methodologies and in particular I'll present two performance indexes to characterize OSPF performance of a Software Router; after the evaluation of results obtained and a software code analysis I'll show a Quagga optimization which makes Software Router performance comparable with ones of a commercial router. In Chapter 3 I will discuss OSPF Fast Convergence and in particular I'll explain the advantages of using an incremental algorithm in real Internet; after

that I'll describe our multi-path incremental algorithm in detail and with an example. Our algorithm, after a complexity analysis, will be then compared with the incremental version proposed by Cisco routers, through Black Box measurements, and with an incremental algorithm proposed in literature, through White Box measurements.

Chapter 1

Software Routers

1.1 Introduction

Routers are the key components of Internet network infrastructure: they are interconnected through links or networks, forming a backbone network that guarantees the communication between Internet users. The essential operation performed by a router is packet forwarding: when a router receives an IP packet from one of its interfaces it has to evaluate packet IP header and select the correct output interface through which the packet has to be forwarded. The forwarding decision depends on router knowledge about network topology; if a router knows the exact topology of network it can evaluate a path to each destination and so it is able to forward every IP packet it receives. The knowledge of network topology is based on communication between routers, performed thanks to the implementation of various routing protocols.

From an hardware viewpoint, router essential components [1, 2] are:

- *interfaces*, that must implement the layers 1 and 2 OSI protocols on the link they control;
- a *central processor*, that manages the router and implements routing protocols to exchange routing information with other routers;
- a *routing table*, that collects all routing information so that IP destination addresses are associated to router output ports;

- an *internal switching fabric*, that allows the information transfer among previous router components.

Such an architecture is really similar to a Personal Computer (PC) architecture: a modern PC has several interfaces (Ethernet, fast-Ethernet, wifi, etc..), a central processor unit (CPU) and a bus, which is the same solution used by medium- and low-end router to implement the internal switching fabric. A PC also provides basic TCP/IP modules: in particular if Linux is used as operating system, a routing table is available and it is possible to provide routing protocols support installing an open-source routing software, free available in Internet.

The similarity among router and PC architecture and the availability of open source routing software have led to a great interest for Software Routers, devices composed by PC architecture, open-source operating system and open-source routing software.

In next Sections I will analyze motivations for the introduction of such devices and I will describe the most common open-source routing softwares, Quagga and Xorp. Finally I briefly describe OSPF protocol because in next Chapters I will compare OSPF performance of Software Routers and commercial routers.

1.2 Software Router motivations

In this section I will describe the reasons for using a Software Router instead of a commercial device, highlighting differences between router market and PC market and emphasizing the advantages of managing an open-source software that can be modified to support specific requirements.

The field of networking equipment in general, and of routers in particular, has always seen the development of proprietary architectures: there are de facto two companies (Cisco [3] and Juniper [4]) covering the entire world market. This means incompatible equipment and architectures, especially as regards configuration and management procedures. This situation has given rise to commercial practices which are not based on free competition, and often the final cost of the equipment is high with respect to the offered per-

formance and the equipment complexity. On the contrary for PC, standards were defined allowing the development of an open multi-vendor market, at least for the hardware components: in this way multivendor PC hardware is available at low cost, because of large scale production, PC hardware architectures are well documented, and their performance evolution is guaranteed by that of commercial PCs. Moreover in the case of commercial devices software is not available while for Software Routers the software is free, documented and changeable; in this way it is possible to understand Software Router behaviour, by a detailed code analysis, and to model its features to specific requirements. An other interesting aspect of Software Router is that code availability allows for the implementation of new networking solutions (protocols, algorithms, etc...) and for their testing in real network scenario: as explained later this aspect will be the most attractive for a researcher. Software Routers have also a weak point: their performance cannot be comparable to ones of an high performance router so they can be an alternative for low-end proprietary routers. Moreover, performance limitations can be compensated by the natural evolution of the PC architecture performance.

For all reasons described before Software Routers are appealing alternative for low-end routers; in fact in the last years many projects in the world propose free software and hardware implementations for routers based on PC architecture. Each project has specific goals in terms of flexibility and performance. Among all international projects I think the following are really interesting.

- *Click* [5].

  A pure software architecture based on Linux, and developed at the MIT, well documented, and freely distributed. A peculiarity of the Click architecture is the efficient dynamic definition of the router operations: through the interconnection of elementary blocks, named "elements", it is possible to control the behavior of the router in terms of packet processing algorithms, queueing, dropping policies and scheduling algorithms. The elements can be quite advanced and the definition of new elements is very simple and is based on C++ language. The results obtained allow to exploit the possibility of a software router

approach based on standard PC hardware, as shown by the large processing speed of packets.

- *Freesco* [6].

  A free and open distribution of Linux, with reduced functionalities and simpler configuration: all the software resides on a 1.44 diskette. Freesco allows to implement easily bridge, switch and router, but the maximum number of line interfaces is limited to 3 Ethernet cards and 2 modems. The router configurations allow only static assignments, and dynamic algorithms and routing protocols are not supported.

- *MikroTik* [7].

  Commercial product, relative cheap, with support to wireless access. It provides also a free (but not open) software router, but with reduced functionalities and without technical support.

- *POLLENS* [8].

  POLLENS (Platform for Open, Light, Legible, Efficient Network Services) is an ITEA project aimed at defining and realizing software solutions which can be useful for operators to design, develop and configure added-value services in IP networks. POLLENS provides a middleware platform able to program new algorithms for the traffic control and scheduling, to configure the network architecture and to add intelligence to IP routers for supporting innovative services.

- *Open Router* [9].

  European Community IST project, aimed at developing an high performance and user-friendly router/firewall with a wireless access point based on open hardware specifications and Open Source software. This project is targeted to meet the requirements of the Small Office/Home Office and Small and Medium-sized business market, with expected lower costs than commercial solutions.

- *Extensible Router*[10].

Developed at Princeton university, its goal is to build a prototype router that can be easily extended to support new network services (including overlay and peer-to-peer networks, firewalls, media gateways, proxies, and cluster-based servers), and exploits commercially available hardware components, including network processors. As a result, the project is distributing two Linux kernel modules: VERA, which is a device driver (plus microcode and development tools) for Intel's IXP1200 network processor, and SILK, which implements in-kernel forwarders in the Scout path architecture.

- *LIBEROUTER* [11].

  It is based on the development of a dual-stack (IPv6 and IPv4) router based on the standard PC architecture. The main objective of this project are threefold: performance, ease of configuration and control plane functionality. The project is based upon the concept of programmable hardware: it has developed a FPGA-based accelerator board named COMBO6, which is intended to take over the bandwidth- and CPU-demanding tasks from the main processor and PCI bus.

- *Quagga* [12].

  It is a free software that manages TCP/IP based routing protocols. It is released as part of the GNU Project, and it is distributed under the GNU General Public License. It supports the main routing protocols, such BGP, RIP and OSPF. Zebra follows a modular approach to manage the routing protocols: thanks to the multiprocess architecture, it is simple to modify or update the behavior of a routing protocol without touching the other routing protocols.

- *Xorp* [13].

  Developed at UC Berkeley, it is an open router software platform, aiming to easy extensions for supporting future services. Xorp supports different hardware platforms, from simple PC, to specialized network processors, to dedicated hardware architectures. It also supports a variety of routing protocols and control interfaces. Scheduling and buffer management algorithms for QoS support are available.

Quagga and Xorp are open-source routing software supported by Linux and easy to modify, so they represent the best solution to create a Router from a PC hardware with Linux operating system. In next Section I will give a detailed description of Quagga and Xorp, highlighting differences and strength points.

1.3 Quagga and Xorp

Quagga is a fork of GNU Zebra project [15] born in 1996 by an idea of Kunihiro Ishiguro. Quagga is distributed under GNU public license (GPL) [16], the same used by Linux kernel. GPL is a free software license that allows to change software source code but also requires that derivatives of the source code have to respect the original license: in this way open-source nature of software is preserved.

Quagga is a routing software suite for Unix platform, in particular Linux, FreeBSD and Solaris; it provides the implementation of the most common unicast routing protocols, such OSPF, RIP and BGP. Quagga architecture is reported in Figure 1.1. It is composed by some daemons (programs), each one implementing a different routing protocol: for example *ospfd* is OSPF daemon. There are also daemons with different scope, such *zebra* and *vty*. *Zebra* establishes communication between underlying Linux kernel and routing protocol daemons: for example when a routing update need to be installed in kernel routing table *zebra* sends a specific message (using API kernel method) to the kernel. *Vty* is an additional daemon allowing the configuration of various routing protocols through a network accessible Command Line Interface (CLI) that uses commands really similar to Cisco configuration ones; in this way it is possible to say that Quagga management is Cisco-like.

Quagga installation and configuration processes are quite simple for a Linux user: a file has to be downloaded from Quagga site and three simple commands have to be used to install software in a Linux PC. After that to configure routing protocols a simple text file, for each protocol, has to be properly written and, if further changes have to be made during Quagga execution, CLI has to be used.

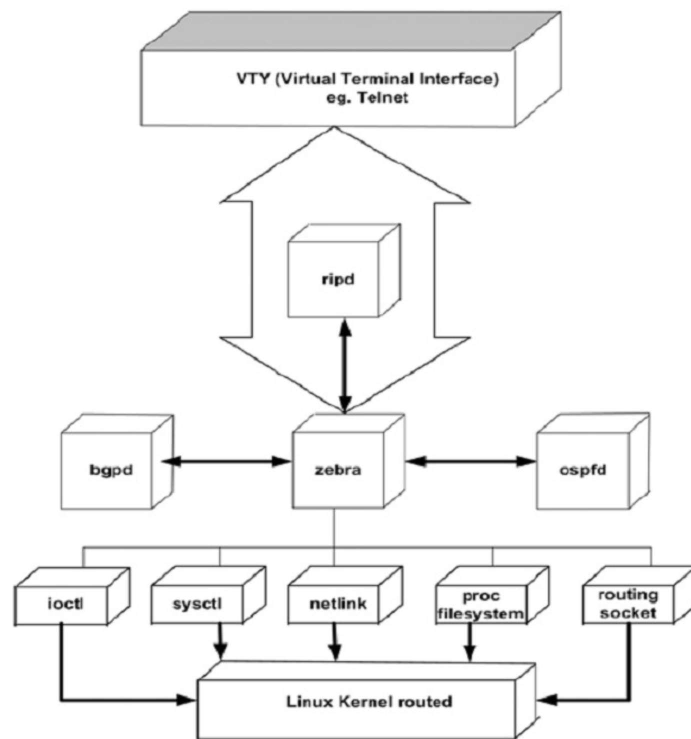

Figure 1.1: Quagga architecture.

Xorp (eXtensible Open Router Platform) is a younger project in respect to Quagga; in fact it was born in 2002 within a core team based at International Computer Science Institute in Berkeley with the support of many companies, such Intel, Google and Microsoft. Xorp is distributed under BSD license, a more permissive license than GPL because it allows proprietary commercial use. This aspect has been exploited to realize in 2007 the first commercial Software Router company, Vyatta [17], that use different x86 hardware devices with Debian Linux operating system and Xorp as routing software. The main attractive characteristics of Vyatta products are lower costs (50% saving) and better performance than similar Cisco devices. I think that Vyatta project is a really important trial to verify Software Router capabilities in a real environment.

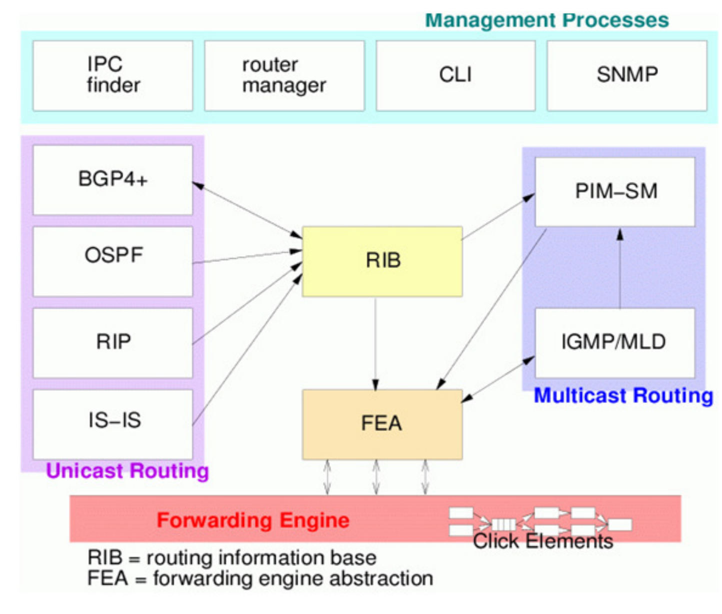

Figure 1.2: Xorp architecture.

Xorp is a routing software for Unix, Macintosh and Microsoft Windows operating systems; it provides the same unicast routing protocols as Quagga (OSPF, RIP, BGP) but also a multicast routing protocol, PIM-SM. Xorp architecture is reported in Figure 1.2 and it is really similar to Quagga one: there is a module for each routing protocol, a *FEA* module (analogous to

zebra daemon) to communicate with kernel and a CLI user interface for management. Differences regard the presence of *RIB* module, that manages Xorp routing table, *IPC finder* process, that manages intra-modules communication, and *router manager* process, that manages router configuration. Xorp also uses some Click elements, that has to be installed in underlying kernel. An interesting Xorp characteristic is that CLI is based on commands really similar to Juniper ones (Juniper-like management).

1.4 Open Shortest Path First (OSPF) protocol

OSPF is an intra-AS link-state routing protocol. As for all routing protocols, its purpose is to manage router's routing table so that the best path to each destination is used during packets forwarding. The intra-AS feature means that it works inside an Autonomous System (AS), that is a collection of routers and networks under the control of the same administrative entity. The link-state feature is instead related to OSPF functioning; each OSPF router describes its topological situation, in particular its active links, to all other AS routers so that every router knows exactly the AS topology (that is of course the same for each router). Router description is based on the generation of a Link State Advertisement (LSA), an OSPF message in which information about router are reported, in particular the list of router interfaces with its neighbors and the metric of each link. LSA dissemination is performed through flooding mechanism: when a router receives a new LSA, an LSA notifying a topological modification, it sends this LSA through all its interfaces, except the one it has received the new LSA from. In this way a new LSA can reach every AS router without sending all LSAs needed from the router generating the LSA.

As explained before every router knows exactly the AS topology thanks to the LSAs received, which are stored in the LSA database. To compute the shortest paths to all network destinations and so to construct routing table, the router performs Dijkstra algorithm [18], having itself as root node: in this way every router computes a different shortest path subgraph. The set

of shortest paths can be defined as a subgraph because a Dijkstra feature is that it computes all the shortest paths to a destination; so if for the same destination there are more shortest paths (of course at the same minimum cost), they all are inserted in the shortest paths subgraph. This OSPF aspect, called Equal Cost Multi-Path (ECMP), is really important because it allows load balancing support: traffic for a destination can be split, through a per-flow differentiation, among available shortest paths. As I will better describe in Chapter 3 OSPF multi-path support has been a relevant aspect of my work.

Chapter 2

OSPF performance evaluation

In this chapter I will introduce the indexes characterizing OSPF performance, highlighting test methodologies and required software to perform such a measurements. After that I will extensive describe test methodologies to measure two of the previous performance indexes, SPF computation time and switching time, showing the really interesting results obtained.

In particular I highlight the optimization performed on Quagga code that makes Software Router performance comparable with ones of a commercial device.

2.1 OSPF test methodologies

OSPF performance evaluation requires the execution of experimental tests. All experimental tests can be divided in two groups:

- Black Box tests
- White Box tests

In Black Box tests the Device Under Test (DUT) has to interact with an external system. The external system sends to the DUT specific signals and, as a consequence, the DUT is forced to execute specific internal functions and to send specific signals to the external system. These signals can be used to understand DUT functioning and eventually to measure DUT performance indexes. The advantage of Black Box tests is that they can be executed

on different devices, so that results comparison is possible; in my case this is a really important aspect because I can compare a Software Router with a commercial router, even if I don't know nothing about software code of the commercial device. The weak point is that measurements need to be processed to obtain results; in particular DUT internal functioning has to be modeled so that it is possible to understand each signal received and to associate each one to a specific internal function. In my case I'm sure that all devices are conformed to OSPF specifications and so this issue is overcome.

White Box tests are performed inside the DUT: in particular functions to be analyzed are started and monitored through internal sensors. In my case a White Box test is performed inserting timestamps in OSPF code of the DUT to evaluate starting time and finish time of a specific function. Of course it is important that timestamps do not influence functions monitored otherwise measurements will be incorrect. White Box tests are more easy to be realized in respect to Black Box ones because it is not required to model DUT functioning and to understand signal exchange with an external system. But White Box tests can be performed only on devices with open code and so, in my case, they could not be used to compare a Software Router with a commercial one; as explained in Chapter 3 they will be used when different algorithms on a Software Router has to be compared.

As a consequence of previous considerations I have to perform Black Box tests to evaluate OSPF performance of a Software Router and a commercial one. So I need an external system which is able to communicate with the DUT and to analyze packets received by the DUT so that specific performance indexes are evaluated. The external system can only be a PC, so called Testing PC, with Linux operating system, the most known and used open-source operating system; in fact with such a Testing PC I can choose among a lot of free programs available on Internet and I can easily realize custom-made programs to satisfy my own requirements. In particular to perform all test methodologies the Testing PC need to use the following softwares:

- *Quagga*: used to realize an OSPF module to communicate with the DUT.

- *OTEG (OSPF Topology Emulator and Generator)* [19]: a software realized by my research team able to generate a network topology and to create and send the OSPF packets describing this network topology. The topology has to be described with a specific text format which is used by a modified version of BRITE software [20, 21] for the network generation. The generation and the sending of OSPF packets describing the emulated topology is realized by a modified version of SPOOF [22] software.

- *Ethereal*: a Linux software used to capture and analyze IP packets (and so also OSPF packets).

A really simple test-bed composed by the DUT and a Testing PC, with previous software installed, can be used to evaluate different OSPF performance indexes.

2.1.1 OSPF performance indexes

The performance characterization of an OSPF router is based on the detection of its main functions and on the definition of procedures to measure them using OSPF control packets. This work has been completed by Benchmarking Methodology Working Group (BMWG) of IETF [23] which has defined OSPF performance indexes and methodologies for tests execution, described in [24, 25, 26]. The three performance indexes defined by BMWG , each one related to a specific function, are reported, with a brief description, in the following list:

- *LSA processing time.*

  When an OSPF router receives an LSA it has to control integrity, age, if LSA is a new one or a duplicated one (already present in its LSA Database), eventually it has to insert the new LSA in its Database and sends an Acknowledge LSA. Of course LSA processing time is influenced by LSA type (new or duplicate), LSA links number and Database extension.

- *LSA flooding time.*

When an OSPF router receives a new LSA it has to perform flooding and so it has to transmit the new LSA through all its interfaces, except the one it has received the LSA from. Also in this case LSA type, links number and Database extension influence this performance index.

- *Shortest Path First (SPF) computation time.*

  The execution of Dijkstra algorithm to compute all shortest paths is the most onerous operation for an OSPF router. The time needed to perform this function depends on network topology complexity and so, as will be better described in next Section, on LSA Database extension.

Besides these three performance indexes proposed by BMWG I have also defined a further one, which considers the interaction between control plane (OSPF) and data plane:

- *Switching time.*

  It represents the time for an OSPF router to update its routing table after a topological event and so to switch traffic from an old path to a new one.

In following Sections I will analyze in depth the last two indexes because they give a full characterization of impact of OSPF router functions on network performance and because results obtained represent the starting point for Software Router modification. In fact, as a consequence of SPF computation time results, I'll show how Software Routers need to be optimized to be compared with commercial ones.

2.2 SPF computation time

In this section I'll first describe test methodology to measure SPF computation time and then I'll analyze results obtained on Quagga, Xorp and Cisco 2801 routers.

2.2.1 Test-bed description

The test aims at determining how long it takes for a Device Under Test (DUT) to complete the SPF computation. The DUT can be either a market router or a PC based router. The test configuration used is reported in Figure 2.1. The network topology is made up of two real routers (a testing PC and the DUT) and a variable number of fictitious routers and networks, so that the DUT will have to find the shortest paths to all the vertexes of the emulated network, a vertex being either a network or a router. The testing PC is running *OTEG* [19] software allowing: i) any network topology to be generated; ii) Link State Advertisements (LSA) describing the network topology to be generated and sent to the DUT.

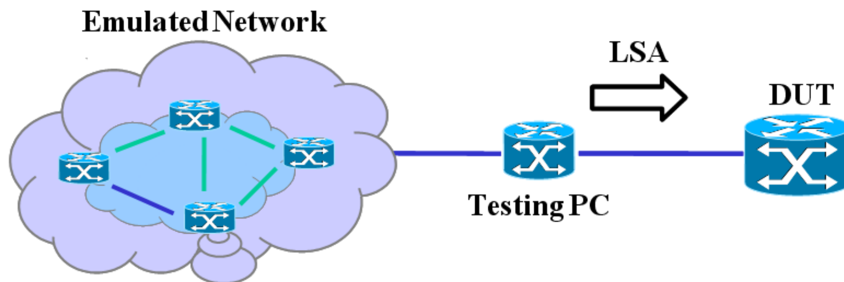

Figure 2.1: Test-bed for the evaluation of the SPF computation time.

I'm going to describe the procedure allowing the measuring of SPF computation time through Black Box measurements, performed according to IETF specifications. To understand the test methodology proposed, it is important to remember that OSPF routers use to schedule the instant in which SPF computation starts to avoid to perform the calculation too many times when receiving Update LSAs [14]. So, when an Update LSA arrives, notifying for example an edge deletion or insertion in an emulated network link, the SPF computation start time is scheduled with a fixed delay, a timer is set and the SPF calculation starts only when the timer expires. Moreover, another timer enforces a lag between two consecutive SPF computations. In particular the following two timers are defined in [14]:

- *spf_delay* : time between receiving an Update LSA and starting the

SPF computation;

- *spf_hold_time* : time between two consecutive SPF computations.

The SPF computation time measurement consists of two different steps, with different settings of these two timers. First, as shown in Figure 2.2, I set both the timers to 0, so forcing the DUT to immediately start the SPF computation when it receives an Update LSA.

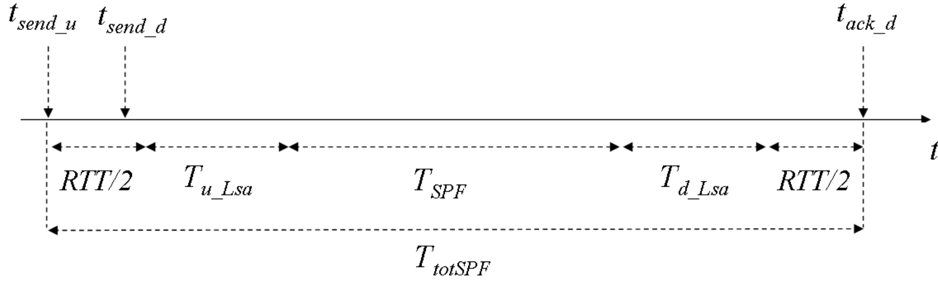

Figure 2.2: Time measure T_{totSPF}.

In Figure 2.2 I denote with RTT the Round Trip Time and further I assume that the propagation time is the same for the two directions: from Testing PC to DUT and from DUT to Testing PC. The first step of the test consists in loading the emulated network into the DUT and in sending an Update LSA at time t_{send_u} followed after a little delay by a Duplicate LSA at time t_{send_d}. The DUT processes the Update LSA in the time interval T_{u_Lsa} and starts to execute the SPF algorithm. Once begun, the SPF computation process cannot be interrupted, and goes on till its end. Then the DUT processes the Duplicate LSA in the time interval T_{d_Lsa} and sends back immediately, according to the OSPF protocol rules, its Acknowledge LSA. Thus I can use the Acknowledge LSA of the Duplicate LSA to understand when the SPF computation ends. In particular in this first test I measure the time T_{totSPF}, which represents the time difference between the sending of the Update LSA at time t_{send_u} and the receiving of the Acknowledge LSA of the Duplicate LSA at time t_{send_d}. As shown in Figure 2.2, the T_{totSPF} time can be expressed as follows:

$$T_{totSPF} = RTT + T_{u_Lsa} + T_{SPF} + T_{d_Lsa} = T_{ov} + T_{SPF} \qquad (2.1)$$

wherein:

- RTT is the Round Trip Time between the Testing PC and the DUT;
- T_{u_Lsa} is the Update LSA processing time;
- T_{d_Lsa} is the Duplicate LSA processing time;
- T_{SPF} is the SPF computation time;
- $T_{ov} = RTT + T_{u_Lsa} + T_{d_Lsa}$.

Hence in the performed measure I am able to evaluate T_{totSPF} but I am interested in evaluating T_{SPF}. In order to make this I evaluate T_{ov} and subtract it from T_{totSPF} obtaining T_{SPF}, that is:

$$T_{SPF} = T_{totSPF} - T_{ov} \tag{2.2}$$

So in order to obtain T_{SPF} I estimate T_{ov} by performing a second test where I set both the OSPF *spf_delay* and *spf_hold_time* timers to high values (60 sec). This time the DUT receives the Update LSA and schedules the SPF computation start time but does not execute it because the timers are too high. Instead the DUT goes on processing the Duplicate LSA and sends back the Acknowledge LSA of the Duplicate LSA as illustrated in Figure 2.3. The time difference between the sending of the Update LSA and the receiving of the Duplicate LSA Acknowledge is exactly T_{ov}.

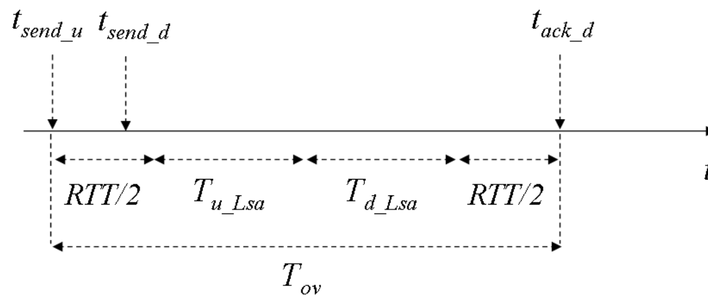

Figure 2.3: Time measure T_{ov}.

2.2.2 Results

All performed tests are based on fully meshed network topologies, with each router connected to each other through a different transit network. Figure 2.4 shows an example of fully meshed topology with 4 routers. It is important to remark that in representing the emulated network as a directed weighted graph [14], each router and each transit network becomes a vertex of the graph, and each network-router link becomes an edge. Each edge is labelled with a cost representing the interface cost of the link connecting a router to a network [14]. In the following the cost of all the edges will be chosen to be equal. When the Update LSA is sent as mentioned in Section II, a link cost is varied and the SPF computation procedure has to be executed because shortest paths could be changed.

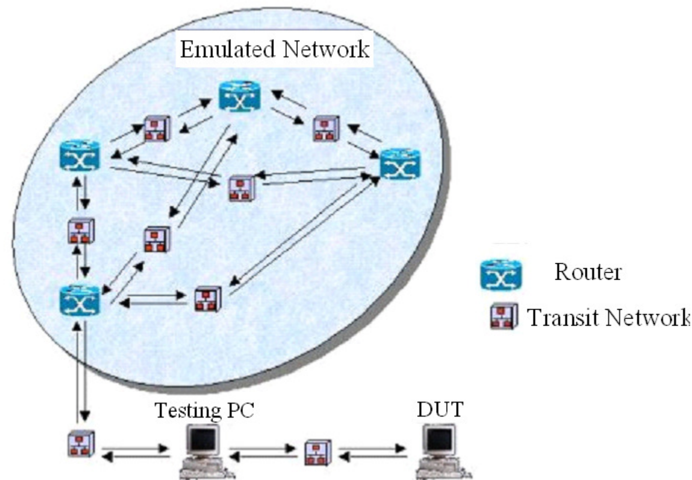

Figure 2.4: Example of a fully meshed topology with 4 routers.

The SPF computation complexity will depend on the number of vertexes and edges in the graph. Now let us denote with N the number of vertexes and with M the number of edges of the graph. If we consider an emulated network topology composed by R routers, we have that:

$$N = (R(R-1))/2 + R \tag{2.3}$$

$$M = 2R(R-1) + 2 \tag{2.4}$$

The number of edges M is proportional to the number of vertexes N, in particular from (3.1) and (3.2) we can assume that $M = O(N)$.

Experimental values taken on a Software Router and on a commercial router are reported in Figure 2.5. We report the SPF computation time as a function of the number of network topology vertexes. The Software Router is realized through a PC equipped with a 2.4Ghz processor, a 512MB memory and Quagga 0.98 or Xorp 1.2 routing software. The commercial router is a Cisco 2801, an access router with 128 MB of memory (no information available about processor). Notice as the experimental values taken on a Cisco 2801 router perfectly agree with the trend foreseen by the Dijsktra algorithm. In fact as shown in Figure 2.5 we notice as the experimental measure curve of the Cisco 2801 fits the curve $0.009NlogN$ very well. On the contrary the results obtained on a router based on the PC hardware and equipped with Quagga or Xorp routing software are quite different. Of the two Open Source Routing Software, Xorp performs much better than Quagga. For example when the number of vertexes is 5000, the SPF computation time in Xorp is about 6 s. On the contrary the SPF time in Quagga increases up to 16 s as the number of vertexes reaches 5000 and measured values fit on the $4 * 10^{-4}N^2$ interpolating curve, as shown in Figure 2.5.

On the basis of these results we retain that some changes are needed inside the Quagga code, to obtain performances comparable to commercial routers. Next section is dedicated to Quagga optimization. Xorp results also show a sub-optimal implementation of OSPF code: a more deep analysis has explained that the weak point of implementation is LSA database, but I have not performed a code optimization as for Quagga.

2.3 Optimization of the SPF Computation Time in Quagga

The SPF computation is based on the Dijkstra algorithm: the algorithm examines the directed weighted graph in order to find the shortest paths from a root vertex to each other vertex in the graph. All these paths give raise to a Shortest Paths subgraph (SP), because of multi-path support:in

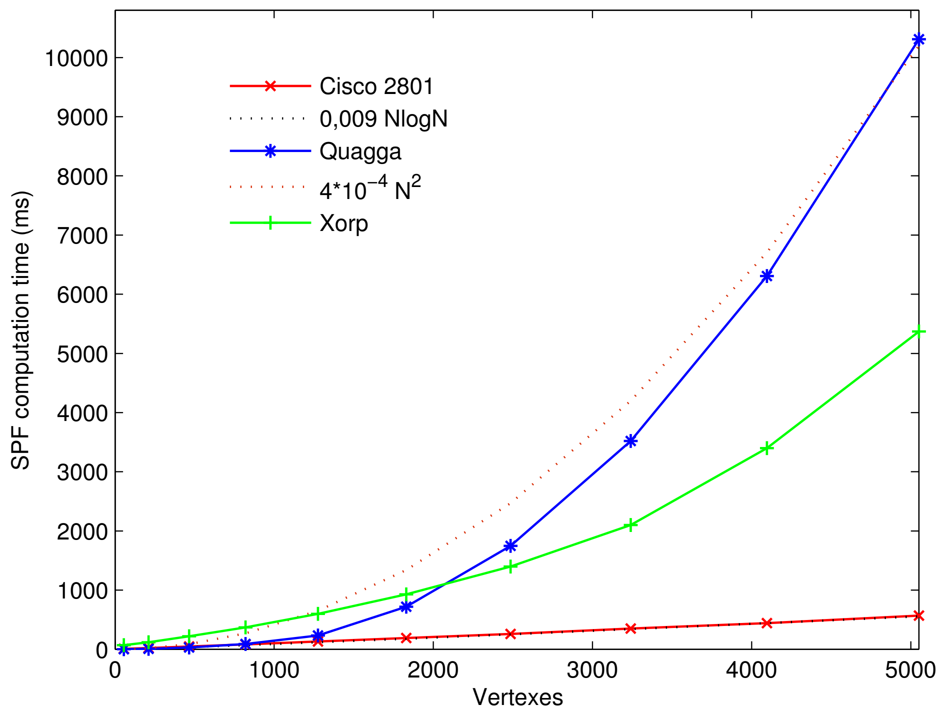

Figure 2.5: The SPF Computation Time in a Cisco 2801 router and a Software Router.

fact SP is composed by all paths of minimum cost to each destination and so it could happen that a destination is reachable through more disjointed paths.

In Quagga the directed graph is itself represented by the LSA set, stored in the LSA database. During the iterations of the algorithm all the vertexes must be extracted, one by one, from the graph and inserted into the SP. Moreover Quagga also uses the Candidate List, a structure that contains all the reachable vertexes that have not yet been inserted but can be reached from vertexes already inserted into SP. The Candidate List is used as a step in the middle during the migration of the vertexes from the graph to SP. Each of the reached vertexes is extracted from the graph, inserted into the Candidate List and provided with a key that represents the total cost needed to reach it starting from the root and crossing the minimum cost path composed by only the vertexes that have already been inserted into SP. According to the Dijkstras algorithm a vertex will be extracted from the Candidate List and inserted into SP only when it becomes the node with the lowest key in the Candidate List. The algorithm finishes when all of the vertexes have been inserted in SP and that occurs when the Candidate List becomes empty.

During the SPF computation the Candidate List is the most stressed structure. Its management is the key point of the resulting global performances, and it is performed by four different functions: the Extract-Min, the Insert, the Decrease-Key and the Lookup functions.

In Quagga, the Candidate List is implemented with a linked list, whose elements are stored in a key increasing order. It is possible to prove that in this case the Dijkstra algorithm complexity is $O(N^2 + NM)$. We have modified the Quagga original version with a patch available in [29]. In the new Quagga version we have chosen the binary heap data structure to replace the sorted list used in the original version.

2.3.1 A Binary Heap to Implement the Candidate List

A binary heap is a complete and balanced binary tree with a local sorting [27]. Leaves are always inserted starting from the left, and a new level is

actually created only when the previous one is complete. Thus the heap depth is always less than $logN$, where N is the number of nodes. Each node of the heap has a key, and the whole heap is locally ordered on these keys, so that each node has a key lower than the ones of both its children. This particular sorting ensures that the node with the minimum key is the root of the heap.

The management of the tree is based on two internal functions: the trickle-up and the trickle-down functions. The trickle-up function brings up a node with a low key toward the root node. The trickle-down function, on the other side, pushes down a node with an high key toward the leaves of the heap. These two procedures are exploited in order to perform the three main functions supported by the heap structure: the Insert, the Extract-Min and the Decrease-Key functions.

The Insert function takes the new node and inserts it at the end of the heap, i.e. makes it a leaf. Then it executes a trickle-up procedure on the inserted node, and brings it up to its correct position. Notice that because the Insert function needs at most $logN$ sift-up operations, equal to the maximum depth of the heap, its cost is $O(logN)$.

The Extract-Min function removes from the heap the node with the lowest key. This node is always at the root of the heap, and its extraction is almost costless. After this extraction we have two sub-tree, that are themselves ordered heap, and we need to fuse them into one single heap. To achieve this result, the Extract-Min function takes the last leaf of the heap and puts it at the root position, then executes the trickle-down procedure on it and pushes it down to its correct position. Notice that because the Extract-Min function needs the execution of at most $logN$ trickle-down operations, its complexity is $O(logN)$.

Finally the Decrease-Key function changes the key of a particular node to a lower value. Once the key value have been decreased, it executes the trickle-up procedure on the node, and takes it to its new position. Notice that to realize the Decrease-Key function, a number of trickle-up operations at most equal to the maximum depth of the heap is performed. For these reasons the Decrease-Key function cost is $O(logN)$.

The Insert function and the Extract-Min function will be performed ex-

actly N times. The number of times in which the Decrease-Key function is performed depends on the values of the costs of the links. However it will be always less than the number of edges, so we can assume that the Decrease-Key function will be performed $O(M)$ times. So the amortized costs are $O(NlogN)$ for the Insert function, $O(NlogN)$ for the Extract-Min function and $O(MlogN)$ for the Decrease-Key function. Finally, because in Section 2.3.2 I will show that by modifying the LSA database data structure the Lookup function is no more needed, the total cost of the new implementation of the Dijkstras algorithm in modified Quagga becomes as expected $O((M+N)logN)$. In particular when $M = O(N)$ the amortized cost reduces to $O(NlogN)$.

2.3.2 The new Lookup Operation in the Candidate List

Unfortunately the changes made to implement the Candidate List rise a new problem: the binary heap need to scan one by one all the nodes to perform the Lookup function, as the structure is only locally ordered, thus obtaining again a $O(N)$ cost. I have modified the LSA database data structure so that the Lookup function becomes no longer needed at all. In particular for each LSA, stored in the database, I have added an information denoting if or not the LSA is in the Candidate List. In positive case the information also denotes the position in the Candidate List where the vertex associated to the LSA is stored. That allows a vertex associated to an LSA to be immediately accessed during the execution of the Dijkstras algorithm. Further, because the trickle-up and the trickle-down operations may change the position of a vertex in the Candidate List, a pointer to the information of the associated LSA is added for each vertex.

2.3.3 Results for New Quagga Version

The test evaluating the SPF computation time on the New Quagga version produced experimental results that perfectly reflect the $NlogN$ trend. The measured values, varying the number of vertexes in the graph, are presented

in Figure 2.6, and compared with the same measure taken on the Cisco 2801. It is important to note that, comparing Figure 2.5 and 2.6, the SPF compu-

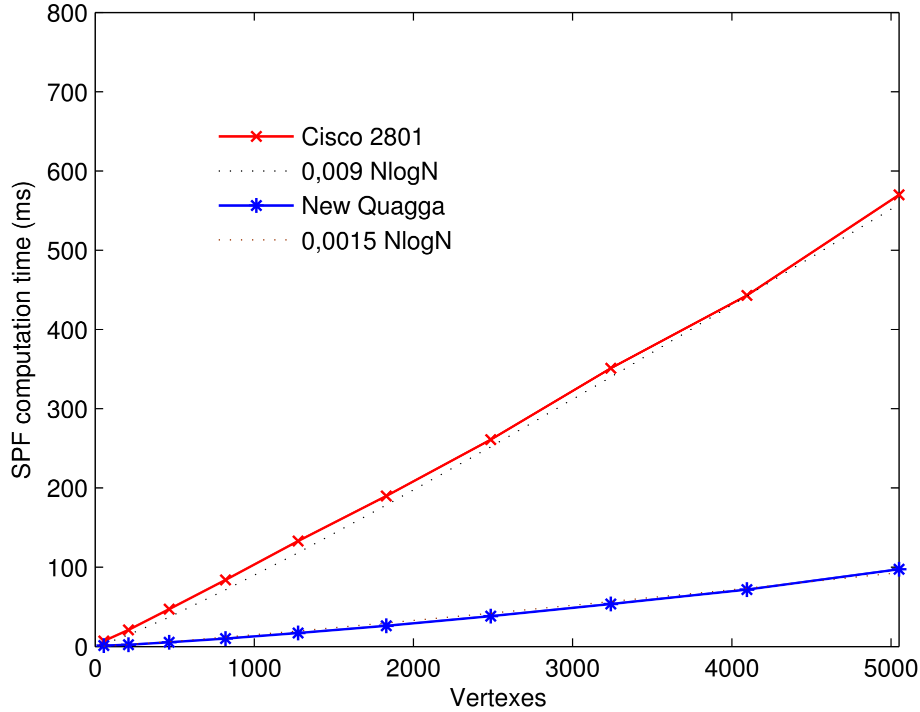

Figure 2.6: The SPF Computation Time in a Cisco 2801 router and a Software Router with new Quagga version.

tation time on the New Quagga version is always less than the time needed on the original version, proving that the optimization have been successfully done, and than the one in Xorp: in particular the curve of New Quagga version fits the curve $0.0015NlogN$ very well. The obtained results are also lower that the ones of Cisco 2801: this aspect can be explained highlighting that computational capacity of the PC used for Software Router implementation is of course higher than the one of an access commercial router, such Cisco 2801.

2.4 Switching time

In this section I'll describe test methodology to measure Switching time and I'll show results obtained on Quagga, Xorp and Cisco 2801 routers.

2.4.1 Test-bed description

This test determines the time for an OSPF router to reconverge the routing table and redirect data traffic when a best route to a destination is available. To determine the route reconvergence performance of a Device Under Test (DUT) we use the test configuration reported in Figure 2.7.

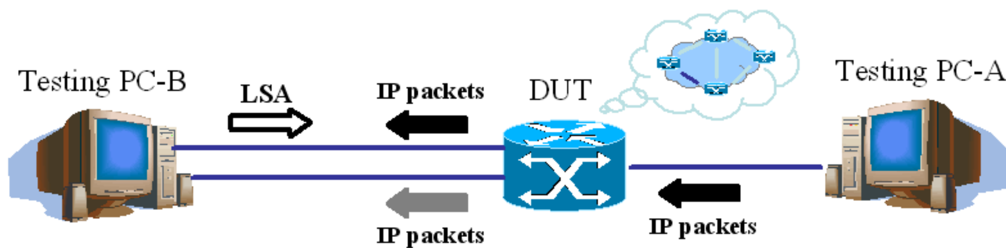

Figure 2.7: Test-bed for the evaluation of the Switching time.

The DUT is connected to two testing PCs, called PC-A and PC-B respectively . The PC-A is connected to the DUT with an only Fast Ethernet Network Interface. Its function is to generate the data traffic that the DUT will switch when a best route will be available. It generates UDP traffic by means of the RUDE traffic generator [28]. The PC-B is connected to the DUT with two Fast Ethernet Network Interfaces. Its function is to emulate a complex network topology and to generate some particular Link State Advertisements (LSA) notifying to the DUT the availability of a best route toward a destination network of the emulated topology. In particular the PC-B allows the topology reported in Figure 2.8 to be emulated.

This topology is made up of the two routers B_1, B_2 and a variable number of fictitious routers and networks, so that the DUT will have to find the shortest paths to all the vertexes of the emulated network, a vertex being either a network or a router. So that the DUT "sees" the emulated network

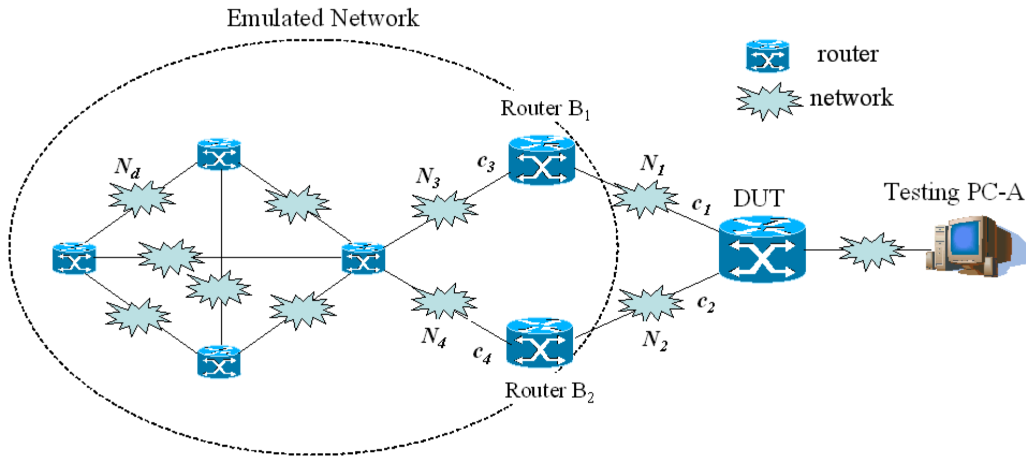

Figure 2.8: Network topology emulated to perform Switching time test.

topology it is needed that the testing PC-B sends to the DUT some appropriate Link State Advertisements (LSAs) describing the emulated network topology. For this reason the PC has been equipped with the *OTEG* software illustrated and available in [19].

According to the test configuration reported in Figure 2.8, the data traffic sent from the PC-A can reach any destination network N_d of the emulated topology through two different paths each involving one of the two network interfaces between the DUT and the PC-B. In particular if the costs c_1 and c_2 of the networks N_1 and N_2 are equal, either of the paths will be chosen by the DUT according to the cost values c_3 and c_4 of the networks N_3 and N_4. At the beginning of the test we set $c_3 < c_4$ and all data traffic is directed through the path including networks N_1, N_3 and router B_1. Then the PC-B will generate an update LSA with new cost c_3 and such that $c_3 > c_4$. So the DUT will process the update LSA, compute the new best paths by means of the Dijkstra algorithm and update its routing table; after that the data traffic will be switched on the best paths including networks N_2, N_4 and router B_2. The time this switching operation takes will be called Switching Time. To summarize the test aiming at evaluating the Switching Time is composed by the following steps:

1. The PC-B emulates a network topology and loads it on the DUT by

sending the LSAs describing the topology. The cost of the networks N_3 and N_4 are chosen so that $c_3 < c_4$.

2. The PC-A sends data traffic to a destination network N_d of the emulated network. According to the costs chosen in step 1, the DUT will route the data traffic on the best path passing through Network N_1.

3. PC-B sends an update LSA with new cost c_3 and such that $c_3 > c_4$. The PC-B also measures the time instant T_1 in which the update LSA is sent. After that the DUT has received and processed the update LSA, computed the Shortest Path First (SPF) and updated the routing table, the data traffic will be routed on the new path including network N_2.

4. PC-B measures the time instant T_2 in which the first packet of the data traffic is received from the network interface connected to the network N_2.

5. PC-B calculates for the network N_d the Switching Time $T_s = T_2 - T_1$

In particular the test can be performed varying the destination network N_d. Considering the OSPF protocol it is possible to say that Switching time is composed by Lsa processing time, SPF computation time and FIB (Forwarding Information Base) update time and so it is easy to predict that Switching time trend as a function of network vertexes will be really similar to SPF Computation time one.

2.4.2 Results

The results I will describe have been evaluated varying the following input parameters:

- F_p, the constant rate at which the packets are transmitted by PC-A;
- L_p, the length of packets sent from the PC-A;
- N, the number of vertexes of the directed graph representing the emulated network topology.

In all performed tests the routers B_1 and B_2 are connected to a fully meshed network topology having each router connected to each other through a different transit network. The Switching time will depend on the number of vertexes N in the graph representing the emulated network because one of the components of the Switching Time is the SPF computation time.

I have performed the measurements in two different cases. In the first case, called "near vertex case" the chosen destination network N_d is near the DUT, that is the vertex representing N_d in the directed graph is among the first to be inserted in the Shortest Path subgraph when the DUT executes the Dijkstras algorithm. In the second case, called "far vertex case" the chosen destination network N_d is among the latest vertexes to be inserted in the Shortest Path subgraph. In the performed measure we can choose either cases by modifying the cost of the edges connected to Network N_d.

Results obtained for the Software Routers (Quagga, New Quagga and Xorp) have been compared, as for SPF computation time case, with ones of Cisco 2801. The first conclusion I can make is that for Quagga there is no differences in Switching time between "near vertex case" and "far vertex case" while in the case of Cisco and Xorp switching time is lower in "near vertex case"; this means that Cisco router and Xorp Software Router update their routing tables every time a destination is inserted in SP during Dijkstra algorithm computation, while Quagga and of course New Quagga update their whole routing tables at the end of Dijkstra process. So to have a real comparison between a Software Router and a commercial one I have to evaluate the only "far vertex case". In Figure 2.9 I have reported switching time results for Quagga, Xorp and Cisco 2801 as a function of N for packet length L_p=100 bytes and rateF_p=5000 pack/s.

Results show that Switching time trend is really similar to SPF Computation time one; it is possible to note that Quagga optimization make the Switching process faster than Cisco one. I can conclude that Switching process is highly influenced by SPF computation phase while Lsa processing and FIB update phases don't affect its trend. Only for Xorp these two phases influences Switching time, making Xorp the one with higher time values (in the case of SPF Computation time Quagga has performance worse than Xorp).

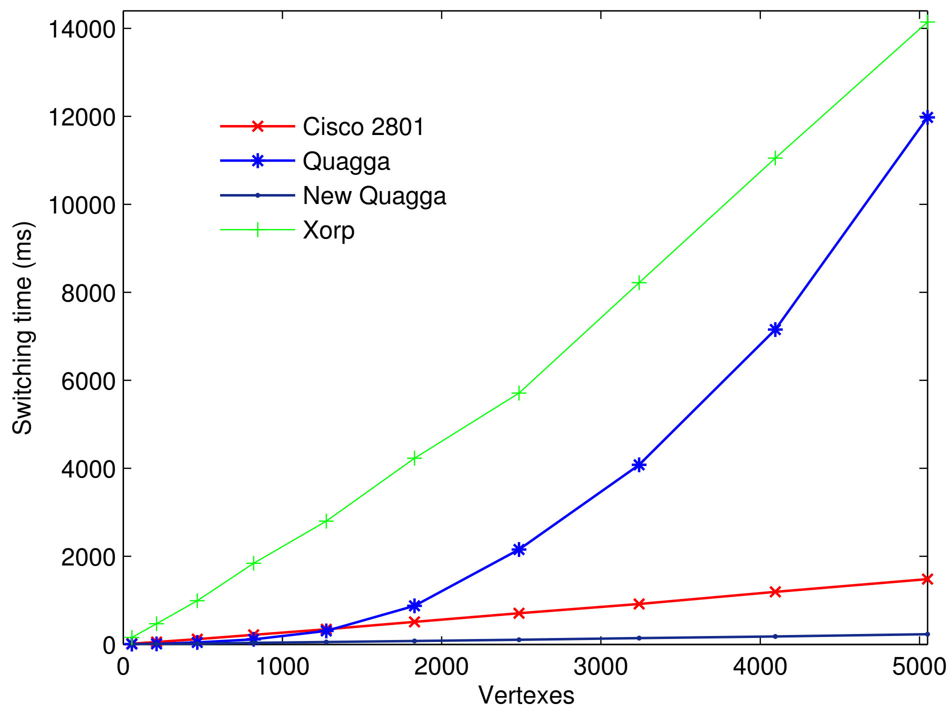

Figure 2.9: The Switching Time in a Cisco 2801 router and Software Routers.

The last thing to do is evaluating how Switching time is influenced by traffic rate. So I have performed the Switching time test varying F_p and I have observed that in all Software Routers (Quagga, New Quagga and Xorp) results are independent from F_p while in Cisco it doesn't happens, as reported in Figure 2.10. This aspect can be explained with high computational capacity of PC that make OSPF performance independent than traffic rate for the considered values of F_p.

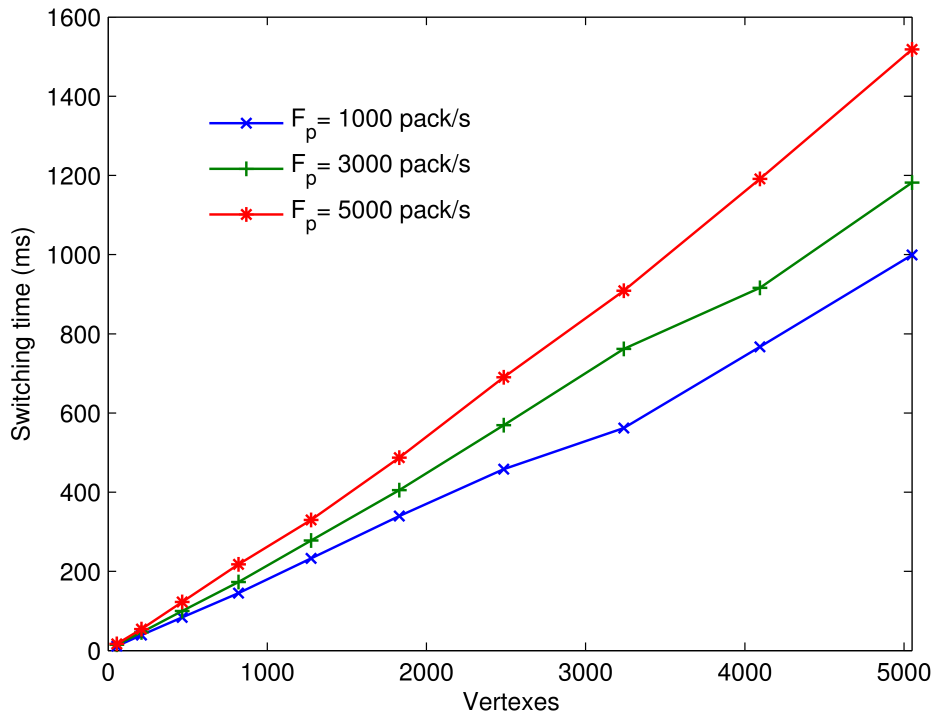

Figure 2.10: The Switching Time in a Cisco 2801 router for different traffic rate F_p.

2.5 Conclusions

In this first part of my PhD work I have proposed some performance indexes to characterize OSPF performance of a Router. In this way I have compared a Software Router with a commercial access router: the first results have

demonstrated that the most known open-source routing software, Quagga and Xorp, have performance really worse than commercial router. So Quagga and Xorp need a deep code analysis, to evaluate reasons of this results and to be submitted to an optimization process. In particular I have realized a new version of Quagga in which previous problems have been fixed so that the Software Router has performance comparable, even better, than the ones of a Cisco router. This work has also produced a patch [29] submitted to Quagga community and inserted in the last version of the software. The conclusion of this work is that a strong point for a Software Router is that it is possible for an user to modify software code, to improve its performance and to make available own work to all other open-source community members.

Chapter 3

A new multi-path incremental algorithm

In this chapter I will analyze one of the most discussed problem of actual routing protocols: the support of new real time services. Before introducing new services requirement let's summarize OSPF routing protocol features.

OSPF is a link-state protocols and use the well-know Dijkstra algorithm to construct the router routing table. Its network performance can be characterized with the convergence time, which represent the time for a network to reconfigure itself when a topological event happens. The convergence time is influenced by three different phases performed by routing protocol: detection, flooding and Shortest Path First (SPF) computation. The detection phase consists in identification of a topological change and, if no hardware detection mechanism is provided, depends on Hello messages exchange between routers; the propagation phase consists in exchange of Update messages from the router discovering the modification to all other network routers, through flooding mechanism; the SPF computation is the phase in which the shortest paths to all destinations are evaluated using Dijkstra algorithm.

The new real time services such VoIP and distributed gaming, which now represent a great part of Internet traffic, require high network performance in terms of delay and jitter. OSPF, such other intra-AS routing protocol, is inefficient in such a scenario [30] because it was designed to support best-effort traffic. In particular convergence time has to be hardly reduced from

the actual 40-50 seconds to 100-200 ms; this purpose can be achieved introducing some optimizations in the three phases of the convergence process [31, 32, 33]: the detection time can be reduced introducing the milliseconds Hello mechanism, the flooding time can be reduced making the flooding messages the ones with highest priority and the SPF computation can be reduced using an incremental algorithm instead of the Dijkstra one.

In this chapter I will introduce incremental algorithms for the single-source shortest path problem and I will evaluate the applicability of past algorithms in a networking scenario. After that I will propose a new incremental algorithm that overcome past algorithms limitations and in particular that supports multi-path, allowing the evaluation of all possible shortest paths to all destinations; in this way routers can perform load balancing for some destinations through paths of minimum equal cost (ECMP: Equal Cost Multi-Path). So I'll propose the implementation of the proposed incremental routing algorithm in the OSPF code of the open-source routing software Quagga: in this way I will show that Software Routers are a really important tool to test new networking solution in a real environment. The performance evaluation will be performed measuring the SPF computation time index of an OSPF router [34] through black box and white box measurements: in the first case our multi-path algorithm will be compared with Cisco incremental one, in the second case with an uni-path incremental algorithm presented in literature.

The chapter is organized into 6 sections. In section 2 incremental algorithms definition, motivation and previous works are described. In section 3 our multi-path incremental algorithm is described, an example in case of link deletion is reported and its complexity is evaluated. Test methodologies are reported in section 4 and the main numerical results of black box and white box measurements are shown in section 5 . Finally in section 6 conclusions and further research items are illustrated.

3.1 Incremental Algorithms

In this section I will introduce motivations and previous works about incremental algorithms. After that I will explain the advantages of the introduc-

tion of an incremental algorithm in an OSPF network to meet real networks requirements and I'll show essential features that such an algorithm need to have in a routing scenario.

3.1.1 Motivations

An algorithm to evaluate the shortest paths from a router to all AS destinations is the key element of a link-state routing protocol. Actual intra AS routing protocols, such as OSPF and IS-IS, make use of Dijkstra algorithm: when a topological change happens all the Shortest Paths (SP) are calculated from scratch, not using the old shortest paths information.

The SP computation, performed by Dijkstra algorithm, is the hardest operation for a router, because all CPU is used, but also the most problematic. In fact when a topological change occurs and SP computation is executing, the used routing table does not reflect the real network topology. In this way data packets can pass through sub-optimal paths or even through paths not more available and so they can be lost, degrading network performance. This last case can happen when there is a link failure, which represents the most dangerous event for a network.

In the last years some studies has been done to characterize the evolution of a network topology, analyzing routing protocols messages and SP successions. In particular the work presented in [35], where the ISP of Michigan State has been monitored for a year, demonstrates that consecutive SPs are really similar, they differ on average for less than five links; a really interesting result is that the 65% of SP computations produces the same SP that was used before the topological event.

These results highlight that Dijkstra algorithm is inefficient in a real scenario where only few operations have to be done to calculate new SP instead of a full computation. These motivations have lead to the introduction of a new class of algorithms for the computation of the shortest paths, the incremental ones [36, 37, 38, 39]. They make use of the previous SP to calculate the new one, computing the only SP influenced by the topological event. In this way incremental algorithms meet real network requirements: computing resources are saved, best paths are available first and so routing convergence

can be highly reduced.

3.1.2 Previous works

In the past years various incremental algorithms for the single-source shortest path problem in a directed and weighted graph have been presented. All this works have been conceived for a most generic problem than a networking scenario, so I analyze for each one its relevant proprieties.

The first interesting work has been presented by Ramalingam and Reps [36], which propose an algorithm reacting to edge deletion or insertion. The most interesting aspect of this work is the introduction of output complexity model, in which the cost of a dynamic algorithm is evaluated as a function of the number of output updates caused by each input modification. This model, with some variations, has been used by all other works. The weak point of the algorithm is that for each node it maintains only distance and outdegree information and not the useful information to calculate the sequence of nodes representing the shortest paths from root node to every topology node.

In [37] Frigioni et al. propose a dynamic algorithm reacting to edge deletion, insertion and modification of weight, which maintains for each node a list of children and a single parent; this last aspect means that the algorithm does not support multi-path equal cost feature. Moreover two further data structures, backward-level and forward-level lists, are maintained for each node allowing the scanning of a less number of edges when the algorithm is performed, with respect to [36].

In [38, 39] Narvaez et al. propose two different incremental algorithms to be used when a link increases or decreases its weight. The first one [38] is the dynamic version of three static algorithms, Dijkstra, Bellman-Ford and D´ Esopo-Pape, while the second one [39] is based on a Ball-and-String model, using an original search criteria. As for [37], these two algorithms maintain a completed SP structure, with parents and children list attributes for each node, but they do not support multi-path.

Other works [40, 41] have been proposed to solve the same problem but they do not introduce nothing new with respect to works described before

3.1.3 Incremental algorithms in a networking scenario

Incremental algorithms described before cannot be directly used in a routing protocol because in a networking scenario they have to satisfy specific requirements.

First at all an incremental algorithm has to be developed to react to the most common events in a network. As shown in [35] edge deletion and insertion are the topological changes that characterize an ISP routing protocol, while edge metric modifications are really rare. A more deep analysis, presented in [42], demonstrates that 70% of unplanned events involves single links. I can conclude that deletion and insertion of a single edge have to be events the algorithms react to. In this way algorithms proposed by Narvaez et al. cannot be used as they have been presented.

A really important aspect that a routing protocol algorithm needs to have is multi-path support. This feature, already supported by OSPF protocol, consists in calculating all shortest paths from a router to each destination. Multi-path additional information is then computed by IP protocol: when the router receives a packet directed to a specific destination, it can choose among different next-hop routers, in general one for each shortest path. In this way router can perform load balancing, choosing a specific next-hop for each packet, or flow of packets, through some functions [43]. Multi-path support, as well as allowing load balancing, also permits to improve TCP performance: in particular works in [44, 45, 46] demonstrate that with some TCP modifications to support path diversity, transport-layer performance can be increased. Multi-path is a feature of static Dijkstra algorithm while is not supported by Frigioni and Narvaez algorithms.

Another aspect to be considered is that the algorithm has to create a routing table: in this way it has to operate with appropriate data structures. In particular for each node some attributes has to be maintained: distance from root, list of parents, list of children and list of next-hops. These last elements represent the first routers in the paths from root to destination node and they are just the next-hop routers to be inserted in the routing table: they can be easily calculated from the list of parents attribute of each node. The algorithm of Ramalingam, unlike others, does not maintain such a data

structure.

Finally, an incremental algorithm to be used in a networking scenario has to react to single edge deletion and insertion, to support multi-path and to create specific data structures. In the next section I'll introduce our incremental algorithm supporting all previous aspects required in a networking scenario and I'll highlight its most important feature: multi-path information is used to reduce SPT computation time and so to quickly re-configure network, especially in a link failure scenario.

3.2 Our Multi-path Incremental Algorithm

In this section I introduce our multi-path incremental algorithm and I show its behaviour in a link failure scenario. Our algorithm is a dynamic version of Dijkstra static algorithm which react to single link deletion and insertion; its relevant characteristic is how it make use of multi-path information to speed up SP computation. Before describing our algorithm I have to introduce some notations used in network graphs.

3.2.1 Terminology

Let $G(V, E)$ denotes a weighted and directed graph where V is the set of nodes and E is the set of edges; let $r(G)$ denotes the root node of G. For each directed edge $e \in E$, let $S(e)$, $E(e)$ and $w(e)$ denote respectively its source node, its end node and its weight. A path from a node x to a node y is a sequence of edges connecting x with y; a cost equal to the sum of the paths edges weights is associated to each path in the graph. The Shortest Path $SP(G)$ is the set of all shortest paths from $r(G)$ to other nodes of the graph and it is unique. Because I take into account the multi-path, more than one minimum cost path from $r(G)$ to a node w in $SP(G)$ could exist. They all have the same cost referred to as w distance.

Each node $v \in V$ has different attributes: $P(v)$ is the set of v parents (a node p is a parent of v if $S(e) = p$, $E(e) = v$ and $e \in SP(G)$), $C(v)$ is the set of v children (a node c is a child of v if $S(e) = v$, $E(e) = c$ and $e \in SP(G)$), $D(v)$ is the set of v descendents (a node p is a descendent of v

if it exists a path in $SP(G)$ in which the node v precedes the node p), $d(v)$ is v distance and $NH(v)$ is the set of v next-hops (a node p is a next-hop for v if it exists a path in $SP(G)$ from $r(G)$ to p, for which p is the first node after $r(G)$). I need to introduce the attribute $NH(v)$ of a node v because in a routing protocol in Internet, a root node, to update its routing table, needs to know, for every path to v, the first node to reach v. $NH(v)$ can be easily calculated from next-hop attribute of all v parents, so if $P(v) = p_1, ..., p_n$ then $NH(v) = \cup_{p \in P(v)} NH(p) = NH(p_1) \cup ..... \cup NH(p_n)$. The $NH(v)$ attribute needs to be recalculated even if, after a topology change, v does not change any other attribute but a node of which v is a descendent has changed its parents attribute. To easily know if a node x belongs to the set of descendents of a node v, a nodes attribute $mar_v(x)$ is used: if $mar_v(x) = 1$ x is a v descendent, otherwise $mar_v(x) = 0$ and x is not among the v descendents. The algorithm uses two set of edges: $I(D(v))$, all edges incoming into the set of v descendents , and $O(D(v))$, all edges outgoing from the set of v descendents.

The algorithm maintains a data structure, the candidate list Q, that contains nodes whose attributes must be updated. An element in Q is the triple (v, P, d_{new}), where v is the node to be updated, P is the new set of parents and d_{new} is the new v distance. Two operations can be performed on Q: EXTRACT and ENQUEUE. The first one extracts from Q the element with the smallest d_{new} field. The second one adds one element to Q; if the node is already in Q the d_{new} fields are compared: if the new d_{new} field is smaller than the old one, the old element is replaced with the new one, if they are equal it is only added the new P field to the old one, else no operation is performed. The correctness of the incremental algorithm is reported in Appendix.

3.2.2 Algorithm description

The incremental algorithm is divided into two phases: the initialization phase, in which all nodes directly affected by deletion or insertion of edge are updated, and the main phase, in which candidate list nodes are inserted in $SP(G)$. The formal description of the algorithm is reported below.

procedure INSERTION OF EDGE e

Step-1
$v = E(e), p = S(e)$
if $d(p) + w(e) > d(v)$ **then**
  STOP
end if
if $d(p) + w(e) = d(v)$ **then**
  Go to Step-2
else
  Go to Step-3
end if

Step-2
$P(v) = P(v) \cup \{p\}$
$C(p) = C(p) \cup \{v\}$
$NH(v) = NH(v) \cup NH(p)$
for all $n \in D(v)$ **do**
  $NH(n) = NH(n) \cup NH(v)$
end for
STOP

Step-3
$\Delta = d(v) - [d(p) + w(e)]$
$d(v) = d(v) - \Delta$
for all $x \in P(v)$ **do**
  $C(x) = C(x) - \{v\}$
end for
$P(v) = \{p\}$
$NH(v) = NH(p)$
$C(p) = C(p) + \{v\}$
$mar_v(v) = 1$
for all $n \in D(v)$ **do**
  $d(n) = d(n) - \Delta$
  $NH(n) = NH(v)$
  $mar_v(n) = 1$

end for
for all $n \in D(v)$ **do**
  for all $y \in P(n)$ **do**
    if $mar_v(y) = 0$ **then**
      $P(n) = P(n) - \{y\}$
      $C(y) = C(y) - \{n\}$
    end if
  end for
end for

Step-4
for all $e \in O(\{v\} \cup D(v))$ **do**
  $\alpha = S(e), \beta = E(e)$
  if $d(\alpha) + w(e) \leq d(\beta)$ **then**
    ENQUEUE $(\beta, \alpha, d(\alpha) + w(e))$
  end if
end for
end procedure

procedure DELETION OF EDGE e
Step-1
$v = E(e), p = S(e)$
if $p \in P(v)$ **then**
  $P(v) = P(v) - \{p\}$
  $C(p) = C(p) - \{v\}$
else
  STOP
end if
if $P(v) \neq \{\emptyset\}$ **then**
  Go to Step-2
else
  Go to Step-3
end if

Step-2

$NH(v) = \cup_{p \in P(v)} NH(p)$
for all $n \in D(v)$ **do**
  $NH(n) = NH(n) \cup NH(v))$
end for
STOP

Step-3
$d(v) = \infty$
$P(v) = \{\emptyset\}$
$C(v) = \{\emptyset\}$
$NH(v) = \{\emptyset\}$
$mar_v(v) = 1$
for all $n \in D(v)$ **do**
  $mar_v(n) = 1$
end for
for all $n \in D(v)$ **do**
  $P_{ext} = \{\emptyset\}$
  for all $x \in P(n)$ **do**
    if $mar_v(x) = 0$ **then**
      $P_{ext} = P_{ext} + \{x\}$
    end if
    if $P_{ext} \neq \{\emptyset\}$ **then**
      $P(n) = P_{ext}$
      $mar_v(n) = 0$
      $NH(n) = \cup_{p \in P(n)} NH(p)$
    else
      $d(n) = \infty$
      $C(n) = \{\emptyset\}$
      $NH(n) = \{\emptyset\}$
      for all $x \in P(n)$ **do**
        $C(x) = C(x) + \{n\}$
      end for
    end if
  end for
end for

Step-4
for all $e \in I(\{v\} \cup D(v))$ **do**
  $\alpha = S(e), \beta = E(e)$
  if $d(\alpha) + w(e) \leq d(\beta)$ **then**
    ENQUEUE $(\beta, \alpha, d(\alpha) + w(e))$
  end if
end for
end procedure

procedure Main
while $Q \neq \{\emptyset\}$ **do**
  $(v, P, d_{new}) = \text{EXTRACT}(Q)$
  $\Delta = d(v) - d_{new}$
  if $\Delta = 0$ **then**
    Go to Step-1
  else if $\Delta > 0$ **then**
    Go to Step-2
  end if

  Step-1
  $P(v) = P(v) \cup P$
  for all $x \in P$ **do**
    $C(x) = C(x) + \{v\}$
  end for
  $NH(v) = NH(v) \cup \{\cup_{p \in P} NH(p)\}$
  for all $n \in D(v)$ **do**
    $NH(n) = \cup_{p \in P(n)} NH(p)$
  end for
  STOP

  Step-2
  $d(v) = d_{new}$
  for all $x \in P(v)$ **do**
    $C(x) = C(x) - \{v\}$
  end for
  $P(v) = P$

$NH(v) = \cup_{p \in P} NH(p)$
for all $x \in P(v)$ **do**
  $C(x) = C(x) + \{v\}$
end for
$mar_v(v) = 1$
for all $n \in D(v)$ **do**
  $mar_v(n) = 1$
end for
for all $n \in D(v)$ **do**
  $d(n) = d(n) - \Delta$
  for all $x \in P(n)$ **do**
    if $mar_v(y) = 0$ **then**
      $P(n) = P(n) - \{x\}$
      $C(x) = C(x) - \{n\}$
      $NH(n) = \cup_{p \in P(n)} NH(p)$
    end if
  end for
end for

Step-3
for all $e \in O(\{v\} \cup D(v))$ **do**
  $\alpha = S(e), \beta = E(e)$
  if $d(\alpha) + w(e) \leq d(\beta)$ **then**
    ENQUEUE $(\beta, \alpha, d(\alpha) + w(e))$
  end if
end for
end while
end procedure

The initialization phase is different for edge deletion and insertion. Let us start describing initialization phase when an edge e insertion happens, denoting $E(e) = v$ and $S(e) = p$. First it is checked, step-1, if the path with e as last edge is a shortest one: if the path has a cost higher than $d(v)$ the algorithm stops, because the insertion of the edge e does not allow any shorter path to be found. If the cost of the new path is equal to $d(v)$, step-2,

the new path has to be added to previous shortest paths of node v so it is only changed parent-child relationship between p and v, adding p to v set of parents and v to p set of children. Then for all v descendents it is recalculated the next-hop attribute: it is important to highlight that this last operation has to be done in an ordered way, so if for a node is recalculated next-hop attribute then the same operation has to be done first for all its parents. To do that, the set $D(v)$ has to be ordered: a node always follows all its parents. If new path cost is smaller than $d(v)$, step-3, the new path is the only shortest path of node v so all v attributes are changed: $d(v)$ is now equal to new path cost, p is the only v parent and $NH(v)$ is equal to $NH(p)$. Moreover v is added to p set of children and it is deleted from its old parents sets of children. Then the distances of the v descendents are updated and they are marked using $mar_v()$ attribute; so parent-child relationships between v descendents and node external to $D(v)$ are modified. In this way, every v descendent has in its set of parents only v descendents nodes, in other words nodes belonging to $D(v)$ are reachable only through v and so their next-hop attribute is the same of the v one. Finally all edges outgoing to v and $D(v)$ are scanned to find new possible shortest paths for node external to $D(v)$.

Now I describe the initialization phase of our algorithm when an edge deletion happens. The first thing to do is to check if edge e belongs to $SP(G)$. If e is not an $SP(G)$ edge, the algorithm stops because the edge deletion does not change $SP(G)$. If e is an $SP(G)$ edge, some preliminary steps are performed: p is deleted from v set of parents and v from p set of children. So it is checked if other paths of minimum cost exist, evaluating $P(v)$. If other paths of minimum cost exist, step-2, the algorithm uses this multi-path information to easily re-compute $SP(G)$: it is only changed the next-hop attribute for v and all its descendents. If there are not other paths of minimum cost, step-3, the algorithm try to reduce the number of affected nodes using the multi-path information. First v passes in an unreachable state: its distance is set to infinite, sets of parents, children and next-hop are set to empty and the $mar_v()$ attribute of its descendents is set to 1. For every v descendents is evaluated the set of parents to find possible multi-path not involving edge e. If at least one external path is found, the descendent is deleted from the set of v descendents, its $mar_v()$ attribute is set to 0,

its set of parents is reduced with the only external parents and next-hop attribute is updated. If there are not external paths, the descendent is put in an unreachable state without deleting the set of parents and it continues to belong to the set of v descendents, adding it to its parents set of children composed by only internal nodes. In this way the set of v descendents, for which the old parent-child relationships are maintained, is reduced to a subset of nodes that has to increase their distance, while nodes with external multi-paths are quickly updated. Finally all edges incoming into v and $D(v)$ are scanned to find new shortest paths for node internal to $D(v)$. Notice that the distance of these node has been set to infinite.

The last part of the algorithm is Q elements extraction and handling. If Q is not empty its best element (v, P, d_{new}) is extracted and it is checked if d_{new} is equal to v distance or d_{new} is smaller than v distance. In the first case, step-1, new paths have to be considered so the algorithm only adds P to $P(v)$, v to new parents sets of children and it recalculates next-hop attribute for v and its descendents. In the other case, step-2, new paths are the better ones so v distance and relationship between v and its parents are changed: P is new v set of parents, v is deleted from old parents sets of children and it is added to new parents sets of children, next-hop attribute is recalculated. The same operations are performed for all v descendents, deleting parent-child relationship between them and nodes external to $D(v)$. Finally, step-3, all possible new shortest paths for external nodes are evaluated.

3.2.3 Example of edge deletion

I can explain with an example on a simple network how our multi-path incremental algorithm works. Figure 3.1(a) shows a network graph, where each link is bidirectional and weighted: for simplicity the two edges of the same bidirectional link have the same cost. The solid thick arrows are all the $SP(G)$ edges while the thin dashed ones does not belong to $SP(G)$.

Let us suppose that edge from node p to node v fails. In the initialization phase of the incremental algorithm it is first checked if edge deleted belongs to $SP(G)$ and then all nodes affected by failure are checked: this set of nodes is represented in Figure 3.1(b) with the dashed curve.

The algorithm has to evaluate v set of parents to find possible multi-paths: the only parent of v is p so v is unreachable and its attributes are changed $(d(v) = \infty, P(v) = \{\phi\}, NH(v) = \{\phi\})$. The search of external multi-path is performed for all v descendents, scanned in an ordered way. For node l there are not external multi-paths while for node i there is an external multi-path, with e as parent: so i is deleted from $D(v)$, its set of parents now contains the only node e, its distance remains 40 and its nexthop attribute is recomputed. The $SP(G)$ at this time is represented in Figure 3.1(c).

Scanning v descendents, the algorithm find an external equal cost path for node t too. It has two parents: l, a v descendent, and i, just removed from set of v descendents. So t is in turn removed from $D(v)$, its distance is not changed and its set of parents contains the only node i. The last v descendents is s and for it there are not external multi-path so it is set to an unreachable state. The unreachable nodes maintain their parent-child relationships ($C(v) = l$, $P(l) = v$, $C(l) = s$, $P(s) = l$). After initialization phase the algorithm produces the $SP(G)$ represented in Figure 3.1(d).

The last thing to do in the initialization phase is to find new shortest paths for affected nodes, through nodes external to $D(v)$. In particular all external incoming edges have to be evaluated. In this case for node v the best path has a as parent and a cost equal to 35, so the element $(v, a, 35)$ is enqueued in Q, for node l it has t as parent and a cost equal to 70, $(l, t, 70)$ is enqueued in Q, and for node s it has q as parent and a cost equal to 65, $(s, q, 65)$ is enqueued in Q. Notice as during the initialization phase the set of affected nodes has been reduced from five to three elements, only using multi-path information.

In the main phase of the algorithm, the first element extracted from Q is v: its new possible distance (35) is obviously better than the present one (infinite) so all its attributes, except set of children, are changed ($d(v) = 35$, $P(v) = a$, $NH(v) = \{p\}$); node v will certainly not be modified during the last part of the algorithm, as explained in the correctness analysis. All v descendents have to be updated, so $d(l) = 45$, $NH(l) = p$, $d(s) = 65$ and $NH(s) = p$. The second element extracted is s: its new possible distance is equal to its distance, updated in the first step, so the algorithm simply stores

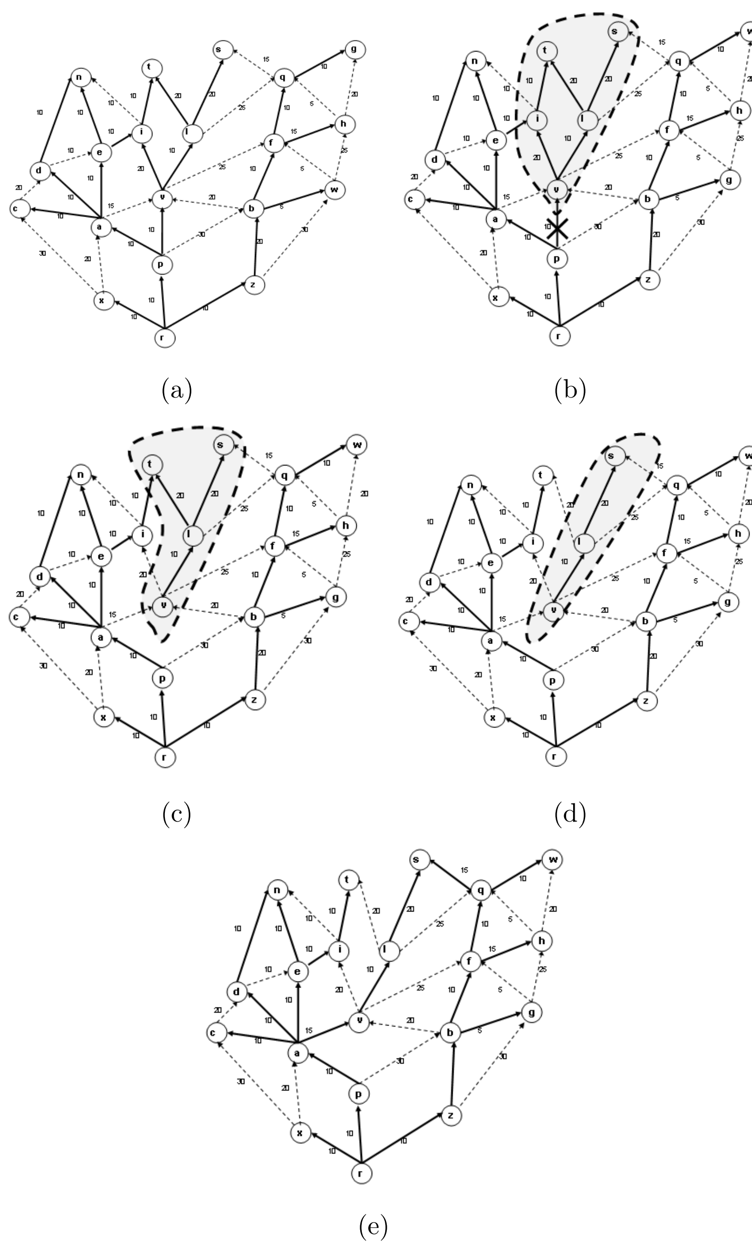

Figure 3.1: Example of edge deletion. a) Network graph: the solid thick arrows represents all the SP(G) edges. b) Set of affected nodes after edge deletion. c) and d) SP(G) during the initialization phase of the algorithm. e) Final SP(G) after that the operations of the algorithm have been performed.

this new equal cost path adding q to $P(s)$ and z to $NH(s)$. Considering equal cost multi-path the router r can reach node s through two different next-hop routers and so it can balance traffic. The last element extracted from Q is l but its new possible distance is bigger than the present one, so nothing has to be done. The candidate heap is empty so the algorithm stops. Final $SP(G)$ is represented in Figure 3.1(e).

3.2.4 Algorithm extensions: weight and multiple edge modifications

As explained in Section 3.1.3 I have proposed an incremental algorithm reacting to single edge deletion and insertion because these are the most common events in a real network. However our algorithm can be modified to react to edge weights modifications. In reality if an edge weight decrease happens the same edge insertion procedure can be executed, while if an edge weight increase happens a modified edge deletion procedure has to be performed: if edge belongs to $SP(G)$ and there are other paths of minimum cost, line 3-4 and Step-2 are performed; if not, line 2-3 are not executed, v distance is updated and Step-3 is performed without deleting paths for node not having multi-path but increasing their distance. The main procedure, in both increasing and decreasing cases, is the same proposed in our algorithm.

Our multi-path algorithm can also be easily modified to handle multiple edge modifications. In particular if both edge deletions and insertions happens, the algorithm need to execute first an an initialization phase in which each delete edge is examined and then an initialization phase regarding inserted edges. In this way both initialization phases need to be extended; let's start describing modifications introduced for edge deletions, which are really similar to ones regarding edge insertions. The first seven lines of Step-1 have to be performed for each deleted edge. Before performing Step-2 and Step-3 the algorithm has to detect all nodes involved, the descendents of all deleted edges, and to shift them in two subset: D_1, that contains all nodes still having at least one path of minimum cost, and D_2, that contains all nodes not having paths of minimum cost. It is relevant to highlight that D_1 and D_2 need to be ordered, as happens for $D(v)$ in the case of single edge

modification. After that Step-2 is performed on D_1 nodes and then Step-3 is performed on D_2 nodes. Modifications regarding initialization phase in case of multiple edge insertions are really similar to ones of multiple edge deletions, essentially based on detection of D_1 and D_2 subset of nodes.

Multiple link modifications, as explained in Section 3.1.3, are not common events in a real network, and they will become more and more rare when a millisecond Hello mechanism will be used.

3.2.5 Algorithmic complexity

In this section I'll discuss the complexity of the incremental algorithm presented in the first section. The algorithm complexity depends on two following topological parameters:

- N: number of nodes of the network topology
- I_d : number of nodes changing distance attribute
- I_p : number of nodes changing parents attribute
- I_{dp} : number of nodes changing both distance and parents attributes ($I_{dp} = I_d \cup I_p$)
- D_{MAX} : maximum number of edges outgoing (or incoming) from a node

The other elements influencing algorithmic complexity are the data structures used to implement the candidate list, the parents and children lists of a node. The candidate list has been implemented with a binary heap, a structure that in case of Dijkstra algorithm allows to reach its theoretical complexity (as explained in Section 2.3.1); if the heap contains n elements, a node extraction and a node insertion have a cost of $O(1)$ and $O(log_2 n)$ respectively. Set of parents and set of children of a node has been implemented with simply lists. The insertion and the deletion of an element from a list of n elements have a cost of $O(1)$ and $O(n)$ respectively. In this section is reported a deep analysis of the incremental algorithm to evaluate its complexity. In each of the three phases of the algorithm, the complexity is influenced by two operations: i)

management of children and parents lists; ii) searching of external shortest paths. In the following I report a complexity analysis for each of the phase of the incremental algorithm.

3.2.5.1 Complexity of the "insertion of an edge" phase

The complexity of the initialization phase in case of insertion is bounded by operations of Step-3 and Step-4, when a better path is found: operations performed when a new path of same cost is found (Step-2) have a lower complexity. The cost of parents and children lists management is influenced by element deletion. In particular at Step-3 each descendent of v is searched and deleted from the children list of nodes external to the set $D(v)$. Because the set $D(v)$ is composed from at most I_d elements, each node in $D(v)$ has at most D_{MAX} parents, each external node has at most D_{MAX} and the search cost of an element in the children list is linear with the number of stored node, the complexity of the step-3 is $O(I_p D_{MAX}^2)$. At step-4, the complexity is dependent on the searching of external shortest paths: all of the edges outgoing from node v and from nodes in $D(v)$ are scanned to find neighbours not belonging to $D(v)$ for which there are new shortest paths. These found neighbours are stored in the candidate list Q. Because the number of edges scanned is at most $I_d D_{MAX}$, the nodes stored in Q are the ones changing parents in number equal to I_p, the cost of insertion of an element in Q is $log(I_p)$, the complexity of the step-4 is $O(I_d D_{MAX} log(I_p))$. Hence the complexity of the "insertion of a node" phase is $O(I_p D_{MAX}^2) + O(I_d D_{MAX} log(I_p))$.

3.2.5.2 Complexity of the "deletion of an edge" phase

The "deletion of an edge" phase has a complexity depending on the Step-3 and Step-4. In particular at Step-3, when no external paths are found, each node in $D(v)$ must be inserted in the children list of its own parents. Because $D(v)$ contains at most I_d elements, each node has at most D_{MAX} parents and the cost of insertion of an node in the children list is $O(1)$, the complexity of the operations performed in Step-3 is $O(I_d D_{MAX})$. At step-4, the complexity is dependent on the searching of external shortest paths: all of the edges incoming to node v and nodes in $D(v)$ are scanned to find

new shortest paths to node v and to nodes in $D(v)$. These found nodes are stored in the candidate list Q. Notice that the number of edges scanned is at most $I_d D_{MAX}$, the nodes stored in Q are the ones changing parents and distance attributes in number equal to I_{dp}. In fact nodes for which are found new shortest paths are v descendents not having any external parent in their parents lists, so their new external shortest paths certainly have a distance greater than old one and parents not belonging to the parents list of the node. According to these considerations and because the cost of insertion of an element in Q is $log(I_{dp})$, the complexity of the step-4 is $O(I_d D_{MAX} log(I_{dp}))$. Hence the complexity of the "insertion of a node" phase is $O(I_d D_{MAX}) + O(I_d D_{MAX} log(I_{dp}))$

3.2.5.3 Complexity of the "main" phase

In the main phase the complexity depends on extraction of a node with a better distance (Step-2 and Step-3). Management of node attributes and searching of new external paths have the same complexity of the operations performed in insertion initialization phase but in this case the operation has to be performed for each better node extracted (at most I_{dp} times) so the complexity of the "main" phase is $O(I_{dp} I_p D_{MAX}^2) + O(I_{dp} I_d D_{MAX} log(I_p))$.

Notice that according to the complexities evaluated in Section 3.2.5.1, 3.2.5.2 and 3.2.5.3, I can conclude that the complexity of the incremental algorithm is $O(I_{dp} I_p D_{MAX}^2) + O(I_{dp} I_d D_{MAX} log(I_p))$.

Algorithmic complexity depends on the number of nodes affected by edge deletion or insertion. In the worst case, when all topology nodes change distance and parents attributes ($I_{dp} = I_d = N$) and the topology is fully meshed ($D_{MAX} = N - 1$), complexity is $O(N^4)$), strongly worse than static Dijkstra one ($O(Nlog(N))$). The worst case analysis is not the best way to characterize an incremental algorithm, as discussed in [36, 37] where the output complexity model is presented. In addition it is highly unlikely that a single edge deletion or insertion affects all topology nodes, in fact, as discussed in Section 3.1, consecutive SP are really similar.

3.3 Performance evaluation and results

As discussed in the last part of the previous section, complexity analysis cannot be a full characterization of our incremental algorithm. So I have decided to evaluate algorithm behaviour in a real environment, implementing it in a routing protocol, OSPF, and measuring protocol performance indexes in different topology scenario. To implement incremental algorithm I have used a routing software with an open code (Open-source routing software), Quagga. Quagga is designed for Unix operating systems (Linux, BSD and Solaris) and it provides TCP/IP based routing protocols, including OSPF, RIP and BGP. The most interesting aspect of an open-source routing software is its flexibility that allows evaluation of new routing feature, such as our algorithm. I have implemented incremental algorithm in OSPF code of Quagga software so that every time an edge deletion or insertion happened, incremental algorithm, instead of Dijkstra one, is performed.

As discussed in Chapter 2 there are different indexes, defined by IETF RFCs, to evaluate OSPF router performance indexes but certainly the most interesting performance index to characterize our algorithm is SPF computation time [47, 48], time needed for a router to complete SPF computation. So I have performed the same test described in previous Chapter and I have also confirmed the obtained Black Box measurements, performing White Box measurements. These have been accomplished by inserting timestamps in OSPF code at the beginning and at the end of the procedure in which the Shortest Paths are evaluated.

The performance evaluation has been carried out by emulating, by means of OTEG software, on the DUT real network topologies measured within the Rocketfuel project [49]. In particular I have considered the topology of two USA Internet Service Providers: Verio, whose network is composed by 893 routers and 4150 links, and AT&T, whose network is composed by 729 routers and 4366 links. All of the link costs have been set to 10. I have decided to characterize algorithms performance in a link failure scenario because it can cause data lost and so network performance degradation. Moreover the SPF computation time in an incremental algorithm depends on link position and on type of change occurring, so I have chosen to measure this time when the

deletion of each single link of the Verios network occurs; after each deletion I have re-inserted the link just deleted and then I have performed the successive link insertion measure.

3.3.1 Black box measurements

Our multi-path incremental algorithm implemented in Quagga has been compared to the one implemented in Cisco 2801, which is the only router company offering an incremental algorithm inside its devices. In case of Quagga a PC with 2,4 GHz processor and 512 MB RAM has been used, while Cisco 2801 is an access router with 128 MB RAM. The SPF computation time in the case of link deletion for Verio topology is reported in Figure 3.2 as a function of the link interested. In the figure I also report the time that the DUT takes to run the Dijkstras algorithm. Obviously this time is constant and independent of the link position in which failure occurs.

Observing Figure 3.2 you can notice that the incremental algorithm is more efficient than the Dijkstra one, both in Cisco 2801 and Quagga . In fact the average SPF computation time for incremental algorithm is 9,5 ms and 0,9 ms in Cisco 2801 and Quagga respectively while for Dijkstra algorithm is 34 ms and 9 ms respectively. Hence the incremental algorithm in Quagga allows for a 8 ms reduction in SPF computation time. That means to avoid to loose 80 millions of bit in the case of a 10 Gbit/s link, if a link failure happens.

The difference in terms of absolute values between Quagga and Cisco 2801 depends on different hardware involved: the PC has a high computational capacity with respect to Cisco 2801. However a deeper analysis shows that the incremental algorithm implemented in Cisco 2801 allows "only" a saving of about 72% in processing time with respect the case in which the shortest paths would be evaluated by using the Dijkstras algorithm, while Quagga saves about 90% in processing time.

The measure performed on Cisco 2801 shows that there are some links whose failure causes an SPF computation time higher than the one obtained when the Dijkstras algorithm is applied. For example in Figure 3.2, that occurs for the link 1 whose failure determines an SPF computation time equal

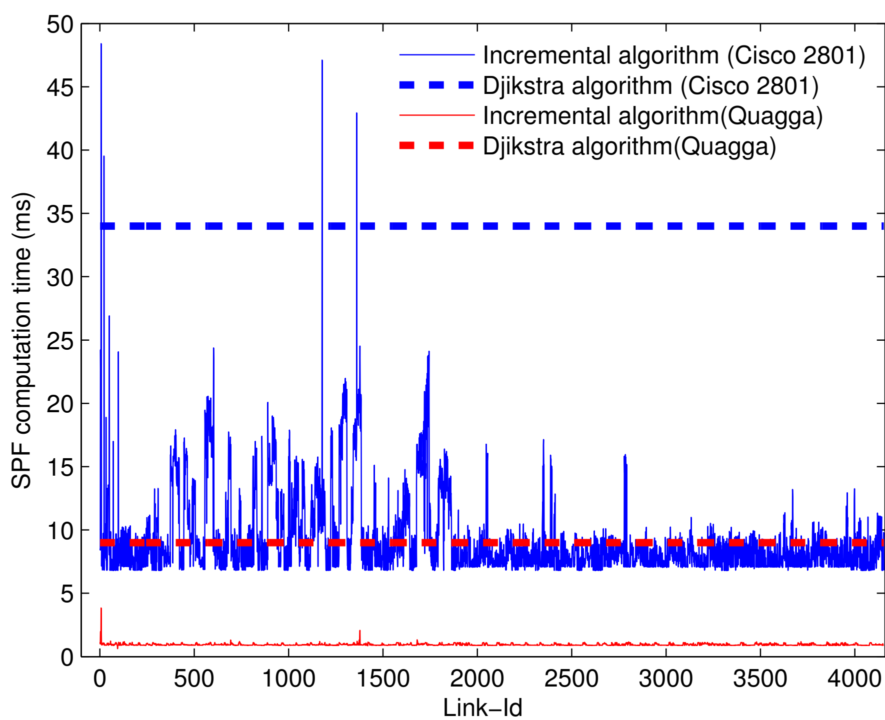

Figure 3.2: SPF computation time in Cisco 2801 and Quagga in the case of link failure for Verio topology.

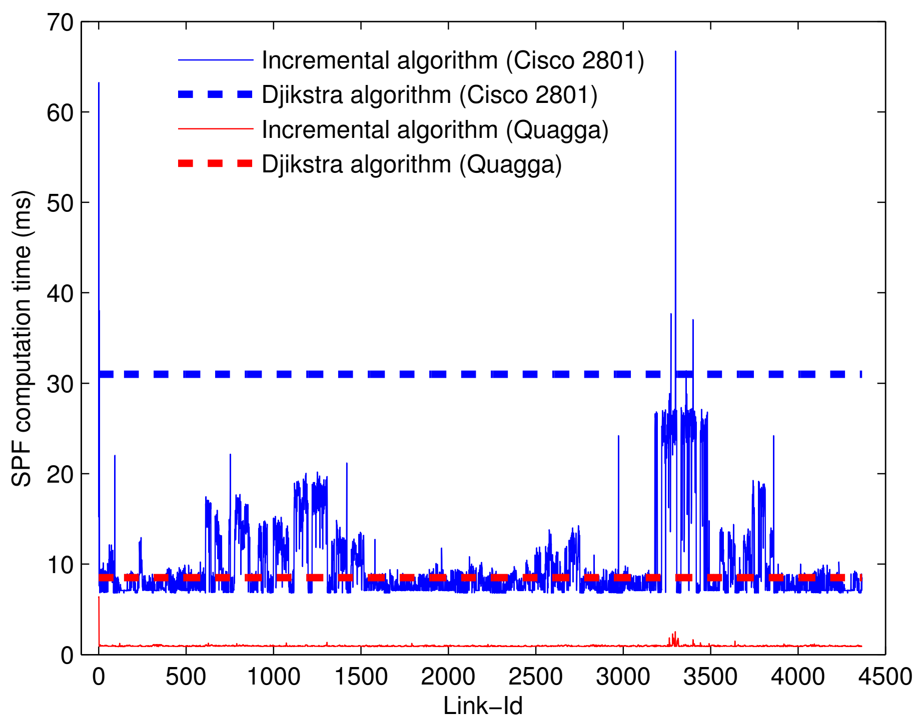

Figure 3.3: SPF computation time in Cisco 2801 and Quagga in the case of link failure for AT&T topology.

to 48 ms against the 34 ms needed when the Dijkstras algorithm is applied. This behaviour never happens in the incremental algorithm implemented in Quagga. To better understand these results I have analyzed the topological situation when link 1 is deleted. I have seen that in this case 571 nodes are directed involved by link deletion (they represents $D(v)$) but 441 of them have external multi-path: I know that in our multi-path algorithm these nodes are updated during initialization phase while in Cisco I can suppose, because software code is not available, that this information is not exploited.

Results obtained for AT&T topology are really similar to Verio's ones and are reported in Figure 3.3.

3.3.2 White box measurements

Our multi-path incremental algorithm has also been compared to the uni-path algorithm proposed by Narvaez [38]. To do that I have implemented the uni-path algorithm, with some modifications regarding mainly deletion support and data structures used, in OSPF Quagga code. I have used the same PC described in Black-Box measurements. The SPF computation time for Verio topology is reported in Figure 3.4 in the case of multi-path and uni-path incremental algorithm respectively, as a function of the link interested; I have decided to order links in x-axis in decreasing order of SPF computation time in multi-path algorithm. In Figure 3.4 I also report the time that the DUT takes to run the static Dijkstras algorithm, which is different in the two cases of uni-path and multi-path: in the first case Dijkstra compute a single path of minimum cost for each destination, in the second case all shortest paths to each destination and so I have two different values in the two cases. Obviously Dijkstra uni-path and multi-path times are independent of the link position in which failure occurs.

Observing Figure 3.4 you can notice that SPF computation time is always less than static algorithm one. Furthermore the average SPF computation time is 0,35 ms and 0,349 ms in multi-path and uni-path algorithm respectively, while the static SPF computation time is 8,116 ms and 7,407 ms in multi-path and uni-path cases; this means that the incremental algorithms allow a saving of about 95% in processing time with respect the case in which

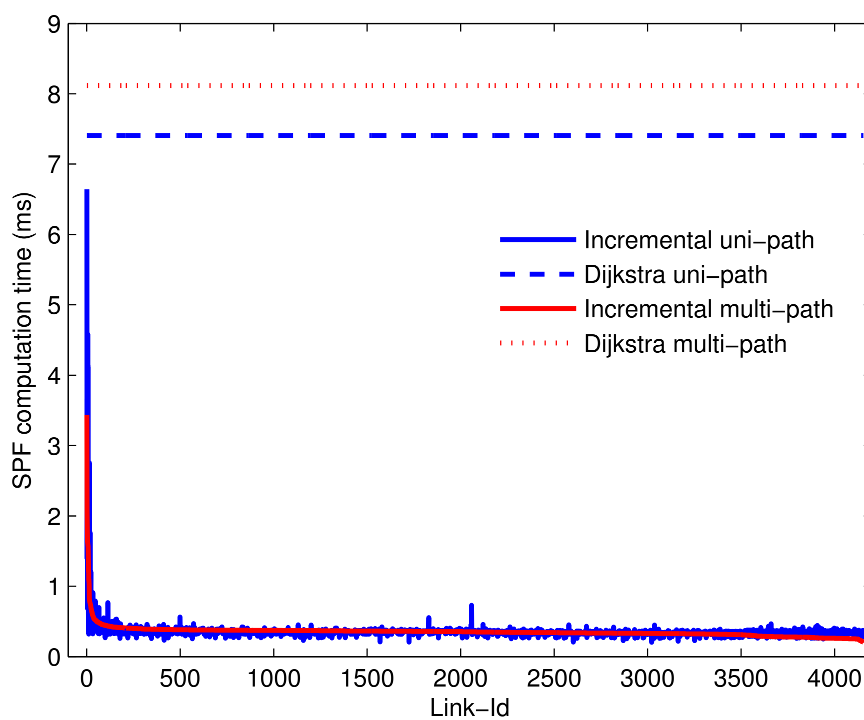

Figure 3.4: Performance evaluation of the uni-path and multi-path incremental algorithms in the case of link failure for Verio topology.

the shortest paths would be evaluated by using the Dijkstras algorithm. A really interesting consideration is that SPF computation time trend is really similar for the two algorithms despite multi-path algorithm allows to compute a more complex structure and so more information. In fact in the case of a uni-path algorithm $SP(G)$, referred to as $USP(G)$, is a tree and so it has exactly $(N-1)$ links, where N is number of nodes; in the case of a multi-path algorithm $SP(G)$, referred to as $MSP(G)$, is a graph, because all paths of minimum cost have to be considered, and so it can have more than N links. In Verio topology $USP(G)$ has 892 links while $MSP(G)$ has 1429 links, so in this last case 500 links more belongs to $SP(G)$ causing a higher number of operations to be performed. The $USP(G)$ and $MSP(G)$ are referred to the situation before each link deletion, because after that $SP(G)$ can change its structure.

Let us consider a subset of links to better understand the results: I only consider the first 50 links because they cause the highest SPF computation times in both algorithms. Results of this subset of links are presented in Figure 3.5. Moreover I have reported in Table 3.1 the most significant links with some information. In the case of uni-path algorithm I report for each link, identified by its id, information about its affiliation to $USP(G)$, that can be Yes or No, number of descendents in $USP(G)$ and SP computation time. In the case of multi-path algorithm I report for each link information about its affiliation to $MSP(G)$, that can be Yes, No or YesMP if link end-node has other paths of minimum cost, number of descendents in $MSP(G)$, number of descendents with other paths of minimum cost (MP Desc) and SP computation time.

Analyzing results I can immediately see that multi-path algorithm results for the first six links of Table 3.1 are really better than uni-path algorithm ones. These points reflect a common situation. In both $MSP(G)$ and $USP(G)$ the deleted link has many descendents but multi-path algorithm allows for a quicker SPF computation, two or three time faster than uni-path algorithm, because it exploits multi-path information. For example the first link regards the one with 448 descendents in $USP(G)$ and 571 descendents in $MSP(G)$, but in this last case 441 descendents have at least another path of minimum cost so the multi-path algorithm makes use of these information to

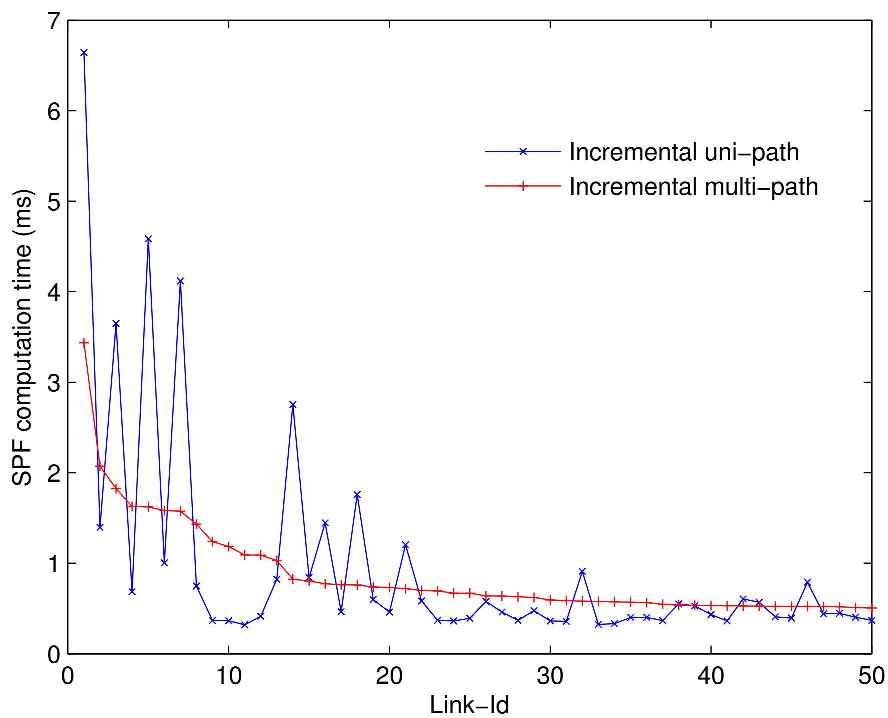

Figure 3.5: Performance comparison of the uni-path and multi-path incremental algorithms in the case of link failure for Verio topology. The SPF computation time of the first 50 links of Figure 9 are reported.

Table 3.1: Most relevant Verio links statistics

	Uni-path			Multi-path			
Link-id	USP	Desc.	Time(ms)	MSP	Desc.	MP Desc.	Time(ms)
1	Yes	448	$6,643$	Yes	571	441	$3,435$
3	Yes	261	$3,648$	Yes	409	357	$1,826$
5	Yes	331	$4,584$	Yes	466	435	$1,622$
7	Yes	325	$4,199$	Yes	369	328	$1,576$
16	Yes	91	$1,446$	$YesMP$	91	/	$0,774$
21	Yes	83	$1,204$	$YesMP$	83	/	$0,719$
2	Yes	55	$1,397$	Yes	187	146	$2,072$
4	Yes	28	$0,684$	Yes	202	188	$1,628$
6	Yes	35	$1,004$	Yes	103	32	$1,581$
8	Yes	26	$0,749$	Yes	198	189	$1,431$
9	No	/	$0,366$	$YesMP$	219	/	$1,238$
10	No	/	$0,365$	$YesMP$	207	/	$1,185$

stabilize these descendents in the initialization phase without inserting them into Candidate List. This result is the most important for our multi-path algorithm; in fact in a link failure scenario, which is the most dangerous for a network because it can cause packet loss, when a lot of nodes are involved, our algorithm allows a quick reconfiguration with respect to an uni-path algorithm.

Another situation in which our algorithm performs better is when the end node of deleted link has itself other paths of minimum cost: this happens for links 16 and 21 in which the number of descendents is 91 and 83 respectively and it is the same in $USP(G)$ and $MSP(G)$, but multi-path algorithm has better reconfiguration times, an half of uni-path algorithm ones.

Obviously there are also situations in which uni-path algorithm performs better. Links 2, 4, 6, and 8 have a lot of descendents in $MSP(G)$ and a few in $USP(G)$ so nodes and links involved in SPF computation during uni-path algorithm are much less and reconfiguration times are better. For example link 4 has 28 descendents in $USP(G)$ while it has 202 descendents

in $MSP(G)$. Most propitious situations for uni-path algorithm are the ones of links 9 and 10: in this cases the deleted link does not belong to $USP(G)$ while it belongs to $MSP(G)$ and it has also a lot of descendents. For example link 9 does not belong to $USP(G)$ but it has 219 descendents in $MSP(G)$; the difference in terms of SPF computation time is limited because the end node of the deleted link has other paths of minimum cost and so multi-path algorithm performs a limited number of operations.

In Figure 3.6 and 3.7 I report the results obtained for AT&T topology, which are really similar to Verio's ones.

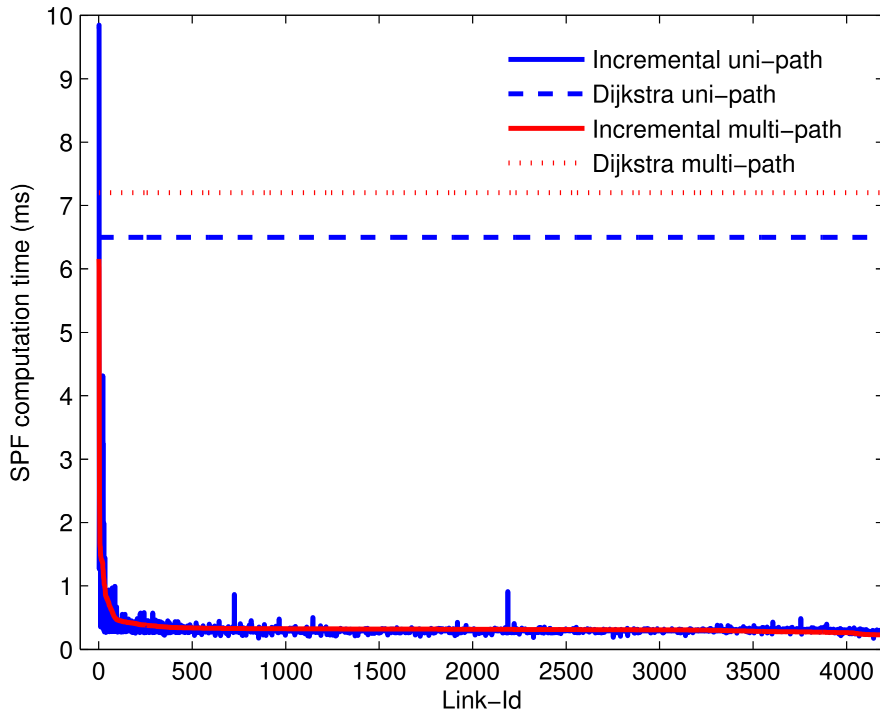

Figure 3.6: Performance evaluation of the uni-path and multi-path incremental algorithms in the case of link failure for AT&T topology.

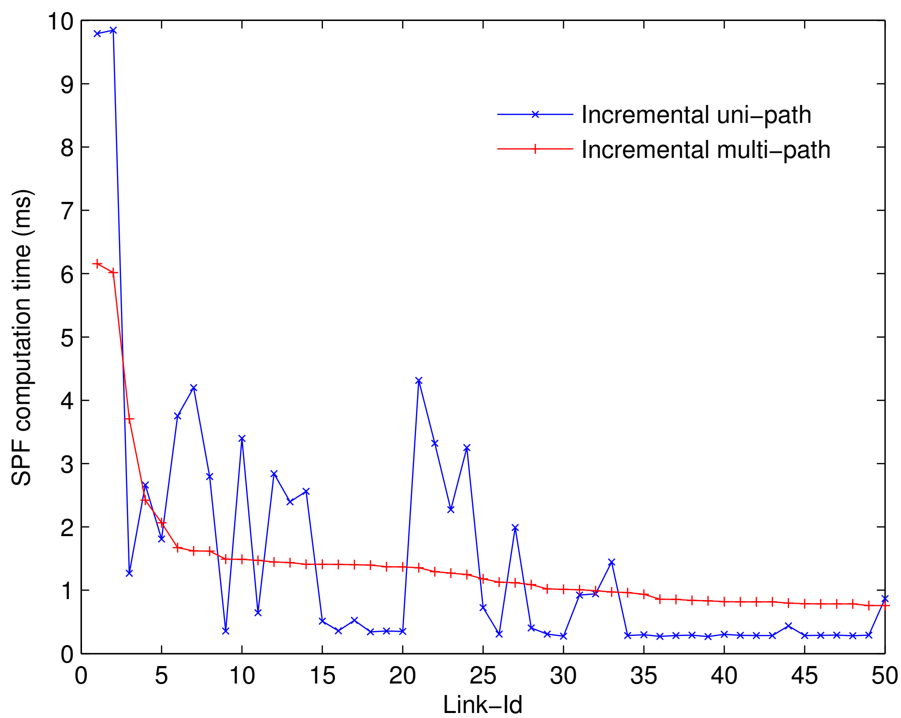

Figure 3.7: Performance comparison of the uni-path and multi-path incremental algorithms in the case of link failure for AT&T topology. The SPF computation time of the first 50 links of Figure 9 are reported.

3.4 Conclusions

The aim of this work was to propose and evaluate the performance of a multi-path incremental shortest path algorithm implemented in the Quagga routing software. First a new incremental algorithm has been realized so that requirements of a networking scenario could be satisfied: in particular the algorithm supports multi-path, reacts to the most common topological events and allows for the creation of a routing table. After that a test-bed has been realized and SPF computation time of the new version of Quagga, with our multi-path algorithm, has been measured. The realized implementation performs well and allows for an SPF computation time lower than in Cisco 2801. Moreover our multi-path incremental algorithm has been compared with an uni-path incremental algorithm and it has been proved that our algorithm performs better than the uni-path one in some topological situations, especially in a link failure scenario when a lot of nodes are involved. The most interesting results has been that multi-path support of incremental algorithm can be used to speed-up network convergence mechanism.

Conclusions

The work I have realized during my PhD thesis can be divided into two phases: a first phase in which an OSPF performance evaluation of Software Routers has been carried out, and a second phase in which a new multi-path incremental algorithm for OSPF protocol has been defined and tested thanks to Software Router device.

In the first part of my work I have described a Software Router, analyzing the most common open-source routing software: Quagga and Xorp. After that I have introduced OSPF protocol and its main features, in particular Dijkstra algorithm and multi-path support, because OSPF is the protocol I have studied in the rest of my work. In particular I have reported the performance indexes to characterize an OSPF router. So I have directed my attention to two performance indexes: SPF computation time, which represents the time for a router to compute Dijkstra algorithm, and Switching time, which represents an index evaluating interaction between control plane and data plane. So to have a performance comparison between Software Routers and a commercial router I have realized a test-bed to evaluate these indexes on both devices. After a results analysis I have concluded that the two Software Routers considered, Quagga and Xorp, have performance worse than a Cisco device. So I have performed a software code investigation and I have detected the weak points of the Software Routers. Thanks to this evaluation I have been able to realize a modified version of Quagga software, accepted and included in official Quagga version, having performance better than ones of Cisco access router. In this way I have proved that a Software Router is a realistic competitor for a commercial device, as demonstrated by Vyatta project, even if a performance evaluation and a further code modification need to be executed before employment.

The second phase has regarded the study of new requirements that routing protocols, in particular OSPF protocol, has to satisfy in a new Internet scenario. I have shown that new real time services, such as VoIP, need a more responsive network so that their requirements in terms of packet lost and delay are fullfilled. So I have described OSPF Fast Convergence which proposes modifications to OSPF protocol so that network convergence can be reduced. I have focused my attention on the introduction of an incremental algorithm instead of Dijkstra one: an algorithm that, after a topological modification, modifies the only network part really involved by modification instead of calcuting all shortest paths from scratch. After a description of past incremental algorithms proposed in literature and the observation that these algorithms cannot be used in a networking scenario, I have proposed a new solution: an incremental algorithm that reacts to link failure or insertion, which are the most common events in a real network, that is able to construct a routing table and that supports multi-path. This last aspect, multi-path support, is the distinguish feature of my algorithm. The algorithm has been described, its correctness has been proved and its complexity has been evaluated. To have a full characterization of the algorithm I have used the Software Router "instrument": I have implemented the algorithm in the OSPF code of Quagga and I have evaluated its performance measuring SPF computation time. The performance evaluation has been realized comparing my multi-path incremental algorithm with Cisco incremental algorithm, through Black Box measurements, and with an uni-path incremental algorithm proposed in literature, through White Box measurements. Results have shown that my algorithm performs always better than Cisco incremental algorithm and that its performance are really similar to uni-path algorithm ones, even if the uni-path algorithm has to compute less information. In particular results have highlighted that my algorithm performs better than uni-path one in a link failure scenario when a lot of network routers are involved by topological modification. I also have demonstrated that this behaviour is a consequence of multi-path support, in fact my algorithm is able to exploit multi-path information, to speed-up shortest paths computation and so routing table update, especially in really critical topological situations.

In conclusion the multi-path incremental algorithm is the innovative as-

pect of my work, but also Software Router characterization and its employment to evaluate algorithm performance represent interesting points of my PhD thesis.

Appendix

Correctness of the Incremental Algorithm

In Lemmas 1-6 we provide some intermediate results. The correctness of the Incremental Algorithm is proved in Lemma 7.

Lemma 1. *In each stage of the algorithm if $d \neq \infty$ is v distance, there is at least one path of length d from $r(G)$ to v for each node in v parents list.*

Proof. During step 4 of the insertion phase a new path is found for node v and its new set of parents contains the only new parent p; for all v descendents, each path external to the set of v descendents is cancelled deleting the external parent of the path from v descendents set of parents. During step 2 of the deletion phase the old path through edge e is cancelled deleting node p from $P(v)$; during step 4 for all v descendents, paths internal to $D(v)$ are deleted. In the insertion and deletion phases when a new path (better or equal than the old one) is discovered, a node in the candidate set of parents is always added. In the main when a candidate node is extracted from Q the parents list is replaced (better path) or expanded (equal paths) using candidate set of parents. Therefore a parent in $P(n)$ is associated with at least one path from $r(G)$ to n because there can be multiple paths from $r(G)$ to the parent. $\square$

Lemma 2. *During the execution of the algorithm, $SP(G)$ does not contain any cycle.*

Proof. In the insertion and main phases of the algorithm, when a new path is found the new parent always has a distance smaller than the one of its new child so there cannot be cyclic paths in $SP(G)$. $\square$

Lemma 3. *The algorithm will terminate.*

Proof. A node v can enter Q if its candidate set of parents is different from the one in $SP(G)$ (this attribute is different when the new distance is smaller than or equal to v distance). Using Lemma 2, we know that exist a finite number of paths and also, using Lemma 1, we know that each node can have a finite number of sets of parents; so a node can enter Q a finite number of time. Since one node is selected and removed from Q in each iteration, the algorithm will terminate. □

Definition A node v is consolidated at some step if its attributes will not change during the rest of the algorithm.

Lemma 4. *If node v is enqueued in Q with its distance attribute d_{min} equal to its shortest distance from $r(G)$, after a finite number of steps, node v will be consolidated with its distance attribute d_{min}.*

Proof. A node v enters Q only if its distance can be reduced or if its parents list can be expanded through others shortest distance paths. When a node v enters Q with its distance d_{min} the only attribute that can change (a finite number of time) is its candidate parents list; when v will be extracted from Q (it happens because of Lemma 3) it will not re-enter Q because all other paths evaluated from this time have a distance bigger than d_{min} so v will be consolidated. □

Lemma 5. *If node v is consolidated with its shortest distance, all descendents of v in $SP(G)$ will be consolidated with their shortest distances.*

Proof. We prove the statement for all v children so we can extend the result to all its descendents. Let c be a v child in $SP(G)$ and let v be consolidate with its shortest distance d_{min}. The child c will either have an update distance $d_{min} + w(e)$ (step 3b), where e is edge connecting v to c, or be examined in step 5. In the first case, c cannot improve its distance furthermore and so it can enter Q only if there are other paths of shortest distance; in the latter case, c will enter Q with distance $d_{min} + w(e)$ and it will extend its candidate parents list attribute if there are multiple paths of shortest distance. In all the cases c enter Q with its shortest distance and so, according to Lemma 4, it will be consolidated. □

Definition After an edge deletion or insertion there is a set of nodes H not having the shortest distance or all shortest paths from $r(G)$. A germinal set is composed by nodes whose descendents include all nodes in H.

To re-compute the correct $SP(G)$ all germinal set nodes have to be detected; so initialization purpose is to discover the germinal set.

Lemma 6. *The initialization enqueues a germinal set of nodes with their shortest distances in the list Q.*

Proof. Let we start evaluating insertion phase. In the first steps all nodes belonging to $D(v)$ are updated with new shortest distances and parents attributes; at this point the germinal set is the set of external nodes for which exists shortest paths with parents belonging to $D(v)$. This set of nodes is discovered in the initialization phase because all edges outgoing the $D(v)$ are scanned and so they are enqueued in Q. In the deletion phase are evaluated all affected $D(v)$ nodes having external parents: the parents lists of these nodes are changed and they are cancelled from $D(v)$ with their descendents. At this point nodes belonging to $D(v)$ have a distance equal to infinite and they have to be update, if possible. The germinal set is composed by nodes belonging to $D(v)$ for which exist shortest paths with external parents; these nodes are detected scanning all edges incoming into $D(v)$ and so they are enqueued in Q with their shortest distances. Finally all germinal set nodes will be consolidated. □

Lemma 7. *At the end of the algorithm, $SP(G)$ contains the shortest paths from $r(G)$ to other nodes of the graph G.*

Proof. From Lemma 3 we know that the algorithm will terminate. From Lemma 6 we know that a germinal set of node is consolidated in a finite number of steps, so from Lemma 5 all nodes not having the shortest distance or all shortest paths from $r(G)$ will be consolidated. □

Bibliography

[1] Bux W., Denzel W.E., Engbersen T., Herkersdorf A., Luijten R.P., *Technologies and building blocks for fast packet forwarding*, IEEE Communication Magazine, January 2001, pp.70-77.

[2] Keshav S., Sharma R., *Issues and trends in router design*, IEEE Communication Magazine, vol.36, n.5, May 1998, pp.144-151

[3] Cisco System Inc., *http://www.cisco.com* .

[4] Juniper Networks Inc., *http://www.juniper.net* .

[5] Click Modular Router, *http://www.pdos.lcs.mit.edu/click* , MIT, Cambridge, USA.

[6] FREE ciSCO, *http://www.freesco.org* .

[7] Mikrotik, *http://www.mikrotik.com* .

[8] ITEA POLLENS, *http://www.itea-pollens.org* .

[9] OpenRouter, *http://www.inaccessnetworks.com/projects/openrouter* .

[10] Extensible router, *http://www.cs.princeton.edu/nsg/router.html* .

[11] Liberouter, *http://www.liberouter.org* .

[12] GNU Quagga, *http://www.quagga.net* .

[13] Xorp, *http://www.xorp.org* , Berkeley University, USA.

[14] J. Moy. *OSPF Version 2* , Request for Comments 2328, April 1998.

[15] GNU Zebra, *http://www.zebra.org* .

[16] GNU public license, *http://www.gnu.org/copyleft/gpl.html* .

[17] Vyatta Inc., *http://www.vyatta.com* .

[18] E. Dijkstra, *A note two problems in connection with graphs*, Numerical Mathemat., vol. 1, pp. 269271, 1959

[19] OTEG (OSPF Topology Emulator and Generator), available at .

[20] Network Emulation Software Brite, *http://www.cs.bu.edu/brite/download.html* .

[21] A. Medina, A. Lakhina, I. Matta, and J. Byers, *BRITE: An Approach to Universal Topology Generation*, in Proc. Ninth International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'01), IEEE Computer Society, Cincinnati, August 15-18 2001.

[22] LSA Generator Software SPOOF, *http://www.cs.ucsb.edu/ rsg/Routing/download.html* .

[23] Benchmarking Methodology Working Group (IETF) *http://www.ietf.org/html.charters/bmwg-charter.html* .

[24] V. Manral, R. White, A. Shaikh. *Benchmarking Basic OSPF Single Router Control Plane Convergence*, Request for Comments 4061, April 2005.

[25] V. Manral, R. White, A. Shaikh. *Benchmarking Terminology and Concepts*, Request for Comments 4062, April 2005.

[26] V. Manral, R. White, A. Shaikh. *Consideration When Using Basic OSPF Convergence Benchmarks*, Request for Comments 4063, April 2005.

[27] A. V. Goldberg and R. E. Tarjan, *Expected performance of Dijkstras Shortest Path algorithm*, Technical Report 96-062, NEC Research Institute, Princeton, NJ, June 1996.

[28] RUDE/CRUDE Traffic Generator, *http://rude.sourceforge.net* .

[29] Routing Software Patch GNU Quagga 0.98, available at *http://net.infocom.uniroma1.it/projects/progetti_dip/zebra/index.htm* .

[30] C. Boutremans, G. Iannaccone and C. Diot. *Impact of link failures on VoIP performance* , in Proceedings of ACM NOSSDAV, May 2002.

[31] C. Alaettinoglu, V. Jacobson, H. Yu, *Towards Milli-Second IGP Convergence* , IETF Internet Draft, November 2000, draft-alaettinoglu-ISIS-convergence-00.

[32] A. Basu and J. G. Riecke, *Stability Issues in OSPF Routing* , in Proc. ACM Sigcomm, August 2001.

[33] P. Francois, C. Filsfils, J. Evans, O. Bonaventure, *Achieving Sub-second IGP Convergence in Large IP Networks* , SIGCOMM Comput. Commun. Rev., 35(3): pp 3544, July 2005.

[34] V. Eramo, M. Listanti, A. Cianfrani, *OSPF Performance and Optimization of Open Source Routing Software* , International Journal of Computer Science and Applications, Vol. IV Issue 1, 2007 .

[35] D. Watson, F. Jahanian, C. Labovitz. *Experiences With Monitoring OSPF on a Regional Service Provider Network* , In Proceedings of the 23rd International Conference on Distributed Computing Systems, page 204, IEEE Computer Society, 2003.

[36] G. Ramalingam and T. Reps. *On the computational complexity of dynamic graph problems* , Theoretical Computer Science, vol. 158, pp. 233277, 1996.

[37] D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni. *Fully dynamic algorithms for maintaining shortest paths trees* , Journal of Algorithms, vol. 34, pp 251-281, February 2000.

[38] P. Narvaez, K.-Y. Siu, and H.-Y. Tzeng. *New dynamic algorithms for shortest path tree computation* , IEEE Transaction on Networking, vol. 8, pp. 734-746, 2000.

[39] P. Narvaez, K.-Y. Siu, and H.-Y. Tzeng. *New dynamic SPT algorithm based on a Ball-and-String model* , IEEE Transaction on Networking, vol. 9, pp. 706-718, 2001.

[40] E. Chan, Y. Yang. *Shortest Path Trees Computation in Dynamic Graphs* , University of Waterloo, Technical Report, March 2005.

[41] B. Xiao, J. Cao, Q. Zhuge, Z. Shao, E. Sha. *Dynamic Upadate of Shortest Path Tree in OSPF* , 7th International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN), Hong Kong (China), 10-12 May 2004.

[42] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.N. Chuah, and C. Diot. *Characterization of failures in an IP backbone* , in IEEE Infocom2004.

[43] D. Thaler, C. Hopps. *Multipath Issues in Unicast and Multicast Next-Hop Selection* , RFC 2991, November 2000.

[44] H.Han, S. Shakkottai, C.V. Hollot, R. Srikant, D. Towsley. *Multi-Path TCP: A Joint Congestion Control and Routing Scheme to Exploit Path Diversity in the Internet* , IEEE/ACM Transactions on Networking , vol. 14, Issue 6, pp. 1260-1271, December 2006.

[45] Y. Lee, I. Park, and Y. Choi, *Improving TCP performance in multipath packet forwarding networks* , Journal of Communications and Networks, vol. 4, no. 2, pp. 148-157, June 2002.

[46] P. Key, L. Massoulie, D. Towsley. *Combining Multipath Routing and Congestion Control for Robustness* , 40th Annual Conference on Information Sciences and Systems, pp. 345-350, 2006.

[47] V. Eramo, M. Listanti, A. Cianfrani, E. Cipollone. *Performance and flexibility of open source routing software* , in Proceedings of NTMS 2007, Paris (France), May 2007.

[48] V. Eramo, M. Listanti, A. Cianfrani. *Switching time measurement and optimization issues in Gnu Quagga routing software* , in Proceedings of IEEE Globecom 2005, St. Louis (USA), November 2005.

[49] Rocketfuel Project, *http://www.cs.washington.edu/research/networking/rocketfuel*
.