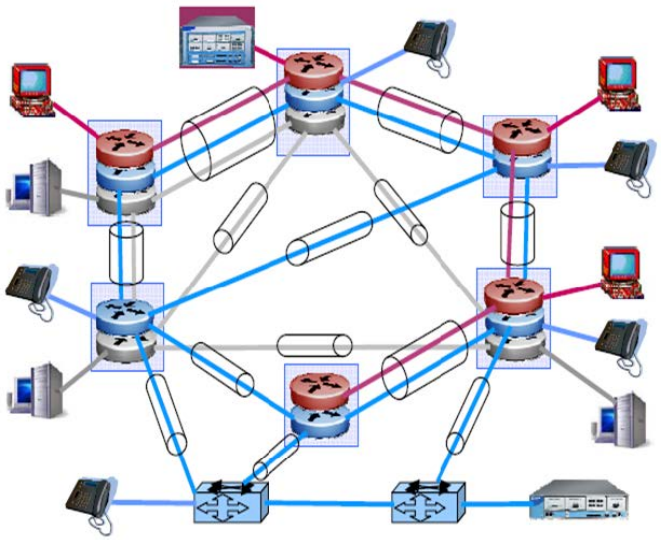
Autonomous management of quality of service in virtual networks

Thanh Son Pham

Par **Thanh Son PHAM**

Autonomous management of quality of service in virtual networks

Thèse présentée
pour l'obtention du grade
de Docteur de l'UTC



Soutenue le 26 novembre 2014
Spécialité : Technologies de l'Information et des Systèmes

D2147

A thesis submitted in partial fulfillment for the degree of Doctor
UNIVERSITY OF TECHNOLOGY OF COMPIEGNE

Autonomous management of quality of service in virtual networks

By **Thanh Son PHAM**

Thesis defence: November 26th, 2014

Jury Committee:

BUI Marc, Professor, EPHE Sorbonne, Laboratory LAISC, Paris
CARLIER Jacques, Professor, University of Technology of Compiègne, Laboratory Heudiasyc
JOUGLET Antoine, Maître de Conférences, University of Technology of Compiègne, Laboratory Heudiasyc
LUTTON Jean-Luc, Ingénieur de Recherche, Orange Lab, Issy-Les-Moulineaux
MYOUPO Jean-Frédéric, Professor, University of Picardie Jules Verne, Amiens
NACE Dritan, Professor, University of Technology of Compiègne, Laboratory Heudiasyc

Guest : Joël LATTMANN, Research Engineer, Orange Lab, Issy-Les-Moulineaux

Autonomous management of quality of service in virtual networks

University of Technology of Compiegne

PHAM Thanh Son

Abstract

This thesis presents a fully distributed resilient routing scheme for switch-based networks. A failure is treated locally, so other nodes in the network do not need to undertake special actions. In contrast to conventional IP routing schemes, each node routes the traffic on the basis of the entering arc and of the destination. The resulting constraint is that two flows to the same destination entering in a node by a common arc have to merge after this arc. We show that this is sufficient for dealing with all single link failure situations, assuming that the network is symmetric and two-link connected. We model the dimensioning problem with an Integer Linear Program which can be solved exactly for small networks. We also propose several heuristics for larger networks. Our method generalizes the methods of Xi et Chao and Li and of Nelakuditi et al. who have proposed similar schemes in the context of IP. Our methods are more efficient than previous ones. We have also studied the existence of a resilient routing scheme for single node failure situation in switch-based network. We study also the case of multi-link failure situations and show that requiring the network to be connected after any failure does not guarantee the existence of a resilent routing scheme as described above.

Acknowledgements

First I want to express my appreciation and my thanks to my two supervisors, Dritan and Jacques. It is for the knowledge that they have taught me as well as the tremendous help they have brought me over the years. I know I have a lot of shortcomings in the work, in behaviour. Still, they have taught me carefully. I have done a lot of mistakes when writing documents and reports in English and French, especially when writing this thesis. I know that it is hard to help me to correct my mistakes. I am grateful to them a lot but do not know how to express it in words. Maybe because I am so touched, may be my limit in expression language. But I still want to tell them that: "Thank you very much, my professors".

Next I would like to thank my enterprise supervisors who have guided me when I worked at Orange Lab in Issy-Les-Moulineaux. First of all I want to thank Joel, who shared the same office with me during the second half of the thesis. Joel spent a lot of time in guidance, exchange, and in answering my questions. There are many times when I was stopped by hard problems, Joel was with me to explore and to resolve these issues. Joel also gave me a lot of useful advice for work and life. I have just completed a small part of them, if I could achieve them all, maybe I would have better results. The second one I would like to thank is Jean-Luc. I'm grateful for several discussion and for the idea he has given when I created the mathematical model as well as the

assessment of my work. These assessments helped me to understand the issues better and to improve my Phd thesis. Next I would like to thank Laurent for his help when I came to the laboratory. His answers to informatics and programming questions help me to adapt myself more quickly.

Besides my supervisors, I would like to thank the rest of my thesis committee: Jean-Fréderic MYOUPO, Marc BUI and Antoine JOUGLET for their insightful comments and questions.

Also, I want to thank the other members of the group RIV for their warm welcome as well as their help to resolve administrative issues.

Next, I would like to thank my Vietnamese friends. Learning and staying away from home caused me much trouble. I want to thank you for having given me warm and fun moments after the periods of work stress, as well as the help when I get stuck. Thank you very much, my friends.

Last but not least, I want to thank my family, my parents as well as my sister. My family gives me more confidence, more power to accomplish this work. Although of the far distance, my parents frequently contact me in order to improve my motivation. Their advice has always been helpful and efficient. They help me in my daily work as well as in health issues. Thank you very much, my family.

Contents

Chapter 1

Introduction

The Internet has been hugely successful in spreading its services all over the world. But the more firmly it becomes implanted, the harder it becomes to introduce new developments. This is the phenomenon of Internet ossification. A number of works have attempted to rethink the architecture of the Internet. Network virtualization approaches [1] have proved their effectiveness in dealing with current limitations of the Internet and supporting new requirements. The principle of network virtualization is to implement multiple virtual node equipments on the same physical machine and to interconnect them through a physical network architecture. Establishing several virtual networks on a physical network infrastructure involves a superposition of different logical topologies with virtual node equipments. Each of the virtual networks behaves like a network in its own, on which it is possible to implement different routing protocols and services. We can suppose that we have two virtual networks on the same physical network infrastructure. One of these networks is used for a special service which requires quality guarantee, while the other one does not. These two networks are called respectively network with priority and "best-effort" network. Our work concerns the

quality of service on the network with priority, which means we have to guarantee a minimum loss of packets in case of failure for this network. Our work can be applied in the context of SDN and OpenFLow. Software Defined Networking (SDN) is a technology whose major application is network virtualization. OpenFlow, the part of SDN in data plane, permits to transfer a packet to its destination following a certain criterion. These criteria could be different, because whole networks support different types of services. This is the relation between OpenFlow and network virtualization. In the context of OpenFlow, switches are used to route the flow. But because the switches in the network do not have a direct exchange protocol between them, they could only forward the received packets following their routing table; so when there is a failed node or a failed link in the network, packets could be lost. Our aim is to propose restoration schemes for network based on switches ensuring that whenever a link or a node fails the traffic can always be rerouted to the destination by an alternative route. A route is a path of traffic between two nodes in the network. It could be the nominal routing path or a rerouting path in case of failure. The politics of the routing protocol in switches based network allows to determine the next hop of the traffic when the switch knows the incoming port/arc and the destination. Consequently, it is forbidden that two routes having the same incoming arc and destination have different outgoing arcs, which is called a conflict. So, our restoration scheme should be without conflict, that is whenever two routes have an arc in common for a same destination, they are identical after this arc.

After this introduction, chapter 2 presents the context of our working environment and related works. In fact, we will explain network virtualization, its interests and also its features. We present also the literature about SDN and OpenFlow which are our working environment. At the end of this chapter, we report the main lines of the works

done in both academic and industrial world in relation with our research. In chapter 3, we illustrate with an example how the restoration scheme works and demonstrate that there exists a free-conflict restoration scheme. Following this proof of existence, we present a mathematical model that allows to calculate a free-conflict restoration scheme and allows to optimize the dimensioning of the network at the same time. In chapter 4, we present two heuristics that permit to calculate a rerouting scheme without conflict while optimizing the dimensioning of the network. In chapter 5, we extend our study to the node failure problem and to the multi-link failure problem. In this chapter, we also demonstrate that a rerouting scheme without conflict for the node failure problem exists and we report a cons-example to show that constructing a rerouting scheme without conflict for multi-link failure problem is impossible. Finally, chapter 6 concludes the thesis.

Chapter 2

Working environment and related works

In this chapter, we present the definition of virtualization, its interest and its features. Next, we describe what are Software Defined Networking(SDN) and OpenFlow. In this context, we present the motivation of our work. At the end of the chapter, we present the main related works in the academic and industrial worlds.

2.1 Working environment

In this section, we will introduce the general context of our problem. This session includes a lot of technical knowledge about network and virtualisation which are mainly taken from [38] and [39].

2.1.1 Virtual machine and virtualization

In this section, we will describe the relation between virtual machine and virtualization. Before defining the notion of virtualization, we have to know exactly what a virtual machine is. Popek and Goldberg have considered it as an efficient, isolated duplicate of a real machine [40]. To have a better definition of a virtual machine, they have introduced the notion of a Virtual Machine Monitor (VMM) which is a piece of software having three essential characteristics. First one is equivalence. A VMM provides an environment for programming; this environment has to be essentially identical to the original machines. The second one is efficiency. Instructions of the virtual processors must be executed directly by the real processor without any intervention from the VMM. And last, the VMM must have a complete control of system resources. With these characteristics, we should can have a clear image of what a VMM is. Then, Popek and Goldberg define a virtual machine as the environment created by the virtual machine monitor.

Popek and Goldberg also show in [40] that a third generation computer has to satisfy some constraints to be able to construct a virtual machine monitor. To understand this work, we have to know what is a third generation computer. In fact, it is distinguished from a simple calculator (first and second generation) by the possibility of offering language instruction, management of hardware and software, support for multi

processing and for multi-users. The functionality of this type of machine is managed by a software layer called “operating system”. This operating system contains two sets of execution instructions: user set and privileged set. The constraint cited by Popek and Goldberg applies only to this type of machine.

To define the constraint for existence of a VMM, Popek and Goldberg classify the instructions of machines into three groups:

Privileged instructions: These instructions are executed only when the processor is on the privileged mode.

Control sensitive instructions: These instructions concern the modification of the configuration of resources in the system.

Behaviour sensitive instructions: The behaviour of these instructions depend on the context of execution.

With this, Popek and Goldberg express their theorem which gives sufficient condition for existence of a VMM.

Theorem 1: For any conventional third-generation computer, an effective VMM may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions.

This theorem shows that we will be able to construct a virtual machine monitor if all sensitive instructions that can affect the functioning of the VMM can always be executed.

To compare with the theories of Popek and Goldberg, we will show what happens and what is applied in reality. In fact, nowadays, creating a virtual machine is to simulate, by software, the operation of a physical machine in a real machine. Once the physical machine emulated, you can load an "operating system" (Windows, Linux, Solaris ...) as easy as in a real machine. Once the system is launched, the virtual machine

behaves like a machine in itself and allows to run software supported by the system. The "classic" use of virtualization is to juxtapose multiple virtual machines and systems on a single physical machine. We call this physical machine a host machine.

These machines see the host machine as an independent machine. They support different types of operating systems and communicate with each other and to the outside through an internal network more or less efficient (software, hardware ...). They are driven by the virtualization manager from the host machine (Xen, VMware, UML ...). There are different modes of virtualization adapted to different uses: models, simulation, operational processes (servers, routers, etc...).

2.1.2 Definition of network virtualization

In this section, we will give a complete definition of network virtualization. What is called "network virtualization" in this chapter is the extension of the above principles to network node equipments (switches or routers) implemented in the virtual computing architectures. This name comes from the academic terminology “network virtualization”. Its principle is to implement on the same physical machine multiple virtual node equipment which are interconnected through the same physical network architecture. In this case, the physical network, also called physical substrate or physical infrastructure, consists of nodes and links that are shared among multiple logical networks or virtual networks (Virtual Network: VN). The establishment of a virtual network on a physical network infrastructure includes a superposition of different logical topologies between virtual node equipments.

Each virtual network behaves like a network of its own:
- It is isolated from the others: the choice of engineering, addressing, configuration... done in a VN does not impact other VN. Events taking place in a VN do not impact

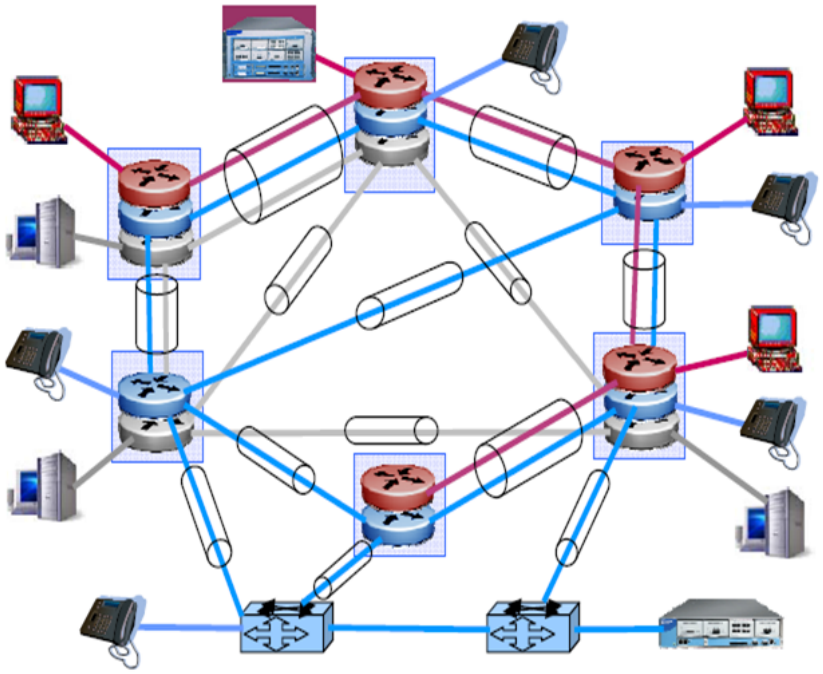


Figure 2.1: virtualized network with different type of services

other VN.

- It has all its own features: control plane, management plane and transfer plane.

- This is a full-fledged network. It is not differentiable to a conventional network by its customers and by its administrator.

Each virtual network behaves as a full network on which it is possible to implement routing protocols and services. Depending on the types of services which are supported, each virtual network must meet specific constraints of quality of service on transit time, availability etc... On such a structure, different overlay networks can be assigned to each service. Virtual networks realize the interconnection between sites that either provide services or are users of these services. The sites are connected via service accesses or clients accesses to the physical network nodes. Because of the functional equivalence between virtual node equipments and physical node equip-

ment, such a network service may have either virtual node equipments or physical node equipments.

The implementation of differentiated services supported by virtual networks over a physical infrastructure can be described by the simplified diagram above (Figure 2.1). Networks are either made of physical node equipments or of virtual node equipments. The links between the physical node equipment can be physical (e.g. fiber) or logical (for example MPLS tunnels). Flows to multiple virtual networks that borrow common links must obviously be multiplexed (for example VLAN 802.1Q Ethernet).

Moreover, the control plane of a VN, on a physical network can be centralized or distributed. A plane for centralized control allows a global view, optimizations and complex schedules with a plane of distributed control. The downside is the risk in return for the loss of connectivity between data planes and the control plane. A distributed control plane provides better resilience because each network element has visibility across all network elements and how to reach them, by implementation and configuration of demons.

The implementation of full network virtualization consists in creating virtual network devices interconnected with virtual links. Elements of a virtual network are dedicated to network functions in a broad sense: routing, switching, proxy, firewall, DPI, load balancing, access point, cache accelerator application, etc... A virtual network topology is a partial graph of the graph of the physical infrastructure which meets the needs of the services provided by the layer. The data exchanged between these virtual machines are multiplexed and sent on the links of the physical sub-strate.

We can find in this implementation the preoccupations which are induced from the works of Popek and Goldberg [40] on virtual processors. If we extend these concepts to network virtualization, we have to consider the virtualization manager as a group

of hypervisors that individually support the virtualization of network equipment. A prerequisite for network virtualization functions is that virtual machines are supported by IT equipment to comply with the rules and theorems for creating a virtual machine.

However, it is a necessary but not a sufficient condition; the function of a network based on switches or based on routers is not limited to data processing of the control plane but mainly concern the ability to commutate packets.

The criteria that define a virtualizable network are found not only in the ability to virtualize the control plane but also in communication capabilities of the equipments which support network virtualization.

Equivalence: It indicates the identity of network behavior whether it is real or virtual. A virtual network must behave like a full-fledged network, independently manageable.

Effectiveness: In performing physical facilities, the transfer plane is managed directly by the hardware. Packets are switched without going through the system that manages the control plane.

Control of network resources: A virtual network manager must have full control on the physical network resources. This implies that each virtualized machine in the network must have access to the physical network resources without explicitly taking into account the activities of other networks that share the same resource. The monitor of each machine is responsible of managing and scheduling accesses to physical resources. This principle characterizes the notion of isolation between networks.

In conclusion, a network is virtualizable if each of its equipment supports virtualization as introduced by Popek and Goldberg [40] in their theorem.

2.1.3 Interests of network virtualization

In this section, we will talk about the interest of network virtualization. What is the need for network virtualization? Firstly it is necessary to improve the network management, because it is currently insufficiently automated and it does not effectively meet the needs of new ecosystems as that of cloud computing (dynamicity, flexibility, insulation ...).

The concept of network virtualization can help the implementation of an operational model close to what was developed in the field of IP network. That is to say, an operational model for the network with effective and highly automated management, simplifying the use and offering new virtual network services on demand.

As in the field of Internet, the network virtualization provides the ability to instantiate multiple virtual networks on a shared physical infrastructure in order to optimize their use. Another advantage is that it becomes possible to instantiate networks, specific to the needs of the applications / services supported, in terms of performance, control, management, QoS , flexibility ...

Under the criterion of equivalence, each virtual network has its own control plane, management plane and transfer plane, its own QoS policy and is independent from those deployed in other virtual networks sharing the same physical infrastructure. With this approach, a configuration error in a virtual layer does not affect the other virtual layers. An important consequence: the introduction of a new service configuration is limited to the configuration of a virtual plane, which simplifies and accelerates the introduction of a new service. In addition, management of the physical network itself is simplified since it becomes limited to the management of different virtual planes independently of supported services.

Virtualization has strong interests for both the operator of the physical network (de-

velopment of its infrastructure and lower operating costs) and the application providers (on-demand deployment and better adaptation to the demand from end customers).

Some examples are given below:

- the creation of a virtual network for telemedicine service with strict bandwidth requirements and network protection,
- cloud computing such as the implementation of a virtual network for the specific needs of multi- data center (eg real - time migration of VMs between data centers) communications,
- the creation of a virtual network for a game community with strong time constraints and important dynamicity needs to be related to the arrivals and departures of players,
- the sharing of physical infrastructure between multiple virtual operators in the mobile network. Each virtual operator must operate its virtual network independently (control level, management, QoS),
- ability to optimize the energy demand of a network of nodes by moving virtual nodes to concentrate virtual resources on a minimum of physical nodes,
- experimenting new technologies in an isolated virtual network of operating virtual networks. For example, the introduction and testing of new protocols (including the transfer plane, if the material is flexible enough to accept new functions),
- possibility for a virtual network to have different topologies according to a programmable schedule daily / weekly,

- the operation of multiple networks on a single physical infrastructure. That is to say that each network has its monitoring plane providing specific functions and has its own characteristics. For example:
    - 1 classic IPv4 network
    - 1 MPLS network to provide business services
    - 1 IPV6 network
    - 1 controlled OpenFlow network
    - 1 network dedicated to the multicast
    - 1 Network Information Centric Networking (ICN)
    - 1 CDN
    - ...

Although the concept of network virtualization is quite old and has not had much success in the past, it now attracts attention as a model for the achievements of the future data networks. The model of virtual network, beyond the response to environmental issues and sharing resources it offers, could impose itself as the preferred carrier service networks within or between ASs (Autonomous Systems), capable of providing a consistent view of services networks including access and mobility.

2.1.4 Properties of the network virtualization

In this section, we will describe the properties that a network virtualization must guarantee.

Isolation

The physical network must ensure isolation between virtual networks. The isolation between virtual networks is related to several aspects.

The first aspect is the secret. It allows any virtual network to make addressing and control independently of the other virtual networks and of the physical layer. The secret must be guaranteed within each network element. For this, a discriminator must be transported in each frame. This discriminator uniquely identifies each frame belonging to a virtual network. To ensure the complete isolation and without constraint between virtual networks, this discriminator must be explicit. Each network element supporting virtualization ensures that the frames to a virtual network will never be issued to another virtual network. In the network elements that do not support virtualization, virtual networks frames are transmitted in the physical plane and its VN identifier will not be used.

The second aspect concerns the monitoring of resources used by each virtual network (CPU, RAM, bandwidth ...). That is to say, in all circumstances a virtual network should not impact the other virtual networks with which it shares the physical infrastructure. This aspect is closely related to issues of QoS treated in the sub-section below. A virtual network must no longer need to worry about the other virtual networks in order to use the resources that have been allocated to it.

Simplifying network management

The virtualization of the network must guarantee the simplification and automation of network operations.

The introduction of a separation between the physical layer and the supporting services (itself divided into several virtual planes services) naturally brings a simplifi-

cation. Overall management is simplified because each virtual plane has a dedicated manager. Configuring a virtual plane is totally dedicated to a service. It cannot have any impact on other planes. Therefore this is an important simplification since the complexity of the configuration is reduced to a layer. It also reduces the risk of errors due to the simplification of configurations (the effects of a configuration error is limited to the layer on which the error occurred).

The physical plane can be simplified since it no longer has to implement the control layer in its totality.

Transparency

The virtualization of the network must guarantee transparency for the applications / services.

That is to say, the client applications that use a virtual network are not aware of this virtualization. Therefore, the network virtualization is immediately usable by existing applications (no change to be made).

In addition, the virtualization layer should not make any assumption about the format of client frames to have the ability to transport them, which are not necessarily based on a given protocol: IP for example.

Scalability

Full virtualization of the network must guarantee the scaling according to two dimensions:

- the number of elements in the physical network: that is to say, by increasing in proportion the number of elements in the physical network when the number of states to keep in each element of the physical layer increases,

- the number of virtual layers that is to say the implementation of a virtual layer affects only the physical network elements supporting it. Only those elements of the physical network that the virtual layer used have a state to maintain. The number of installable virtual networks depends only on available resources in the physical infrastructure. There is no particular complexity associated with a large number of virtual networks.

Control of resources and their allocation

The virtualization of the network must control the resource allocation.

A resource must be shared among multiple virtual networks dedicated to applications with specific network needs both in terms of transfer but also in the control plane. In order to provision for these virtual networks on demand, network virtualization has to offer means for performing an inventory of available resources in the physical layer (CPU, transfer table, memory, bandwidth ...). It should also allow allocating / deallocating resources to virtual networks. Finally, it must control its use.

Performance

The virtualization of the network must allow building virtual networks providing performance required by applications / services in terms of time, bandwidth ... to meet their needs perfectly.

QoS (inter and intra virtual networks)

The virtualization of the network must provide two levels of QoS management:

- A QoS management among virtual layers is provided, for instance: prioritization between the layers in the network elements and links. This QoS management

must ensure that the virtual network with a higher priority will have priority on virtual networks of lower priority in case of congestion on the physical network.

- A QoS management within each virtual layer can implement traditional mechanisms or be based on other criteria. With this QoS management, we have to implement a mechanism that assures the traffic can be transferred correctly in case of failure. Studying such mechanism , which is the main objective of the thesis, will be described in more details in chapter 3 and we can see it used throughout the thesis. Sharing policies between virtual layers are not addressed here.

Orchestration and adaptation to Cloud Computing

Orchestration describes the automated arrangement and management of complex computer system and services. The network virtualization must enable provisioning, change, delete, copy, inspect, move ... automatically and / or at the request of the elements of virtual network operators optionally supported by several physical networks. For this, the network must provide a virtualization layer orchestration. Mobility elements of a virtual network must be available without requiring modification to the virtual network level. This property is a consequence of insulating properties, transparency and simplification of management aspects.

Network virtualization has to permit to take into account rapidly and frequently the migration of a VM. These issues are developed in two documents "NaaS4Cloud Opportunity of a Network -as- a-Service Layer for Cloud services" [23] and "Network for Clouds, a key differentiator Telco" [24]. The virtual network must be adapted to the mobility needs of VM. If a VM is moved in a new area, the virtual network must be able to extend the orchestrator network in this area.

Security

The virtualization of the network must guarantee the safety inter-layers.

2.1.5 Emerging Solutions SDN

Although, there were many works in both academic [13] [14] [15] and industrial [17] [18] [19] [21] world about network virtualization, it has become a hot topic only recently. Network virtualization is currently attracting much attention because it is presented as a case of major use of SDN. This is a new notion that has to be well understood in our context.

Software Defined Networking (SDN) is a new paradigm of network architecture where the control plane is completely decoupled from the data plane. To recall, control plane is a part of network which permits to calculate the network topology or to exchange routing information while data plane or forwarding plane is a part of network where the packets are commutated. This decoupling allows to deploy a monitoring plane on standard servers with flexible computing capabilities compared to conventional switches. And it opens the opportunity to design an efficient centralized control plane. In addition, the creation of a standardized API (Application Programming Interface) between the control plane and the data plane allows developing network services. The control plane is capable of injecting states in the network elements.

According to what is described above, SDN has the ability to control or program the data plane from one or many independent control planes, these planes are possibly centralized. Because of the reduction of control part in the data plane, it doest not only simplify the data plane but also reduce the cost. SDN allows the control plane to have a global view.

As we have presented in the previous paragraph, the major use of SDN is network virtualization. We will try to analyze here in detail how SDN can help to achieve a complete solution for network virtualization. As seen in the previous paragraph, the principle of SDN is to control the data plane of a network from a centralized organ through an interface that allows the creation of flows based on more or less complex criteria which are then manageable by the supervisor. It is then possible to consider an "abstract" network virtualization dedicating routes and treatment to groups of flows characterized by certain criteria. These criteria can be the type of application or are based on a local or remote destination (inter-AS) etc... Thus, all determined paths in the network for a given set of criteria construct a topology, suited to the desired characteristics, extracted from the network graph. This is one of the idea which leads to the use of tree topology in our work.

The paths defined in the topology are not necessarily uniform in the assembly of their components: for example a path in a given topology may be comprised of a subset of edges defined by the criteria of destination address and another subset defined by an application test. However, the resulting topology generally associated with a service or a set of services can be like a "virtual plane" implemented on the network. The question that arises is whether the constraints defined by Popek and Goldberg [40] are applicable to this form of virtualization. For this, we must redefine, by analogy, the basic elements developed in [40] detailing the role of the different elements involved in virtualization.

The following diagram (Figure 2.2) reflects the paradigm SDN stated above:

In this diagram (Figure 2.2), physical substrate, virtual plane, network OS, applications are similar in functional structure to processor, hypervisor, system, user space encountered in the field of IT. We must transpose the three criteria that define a vir-

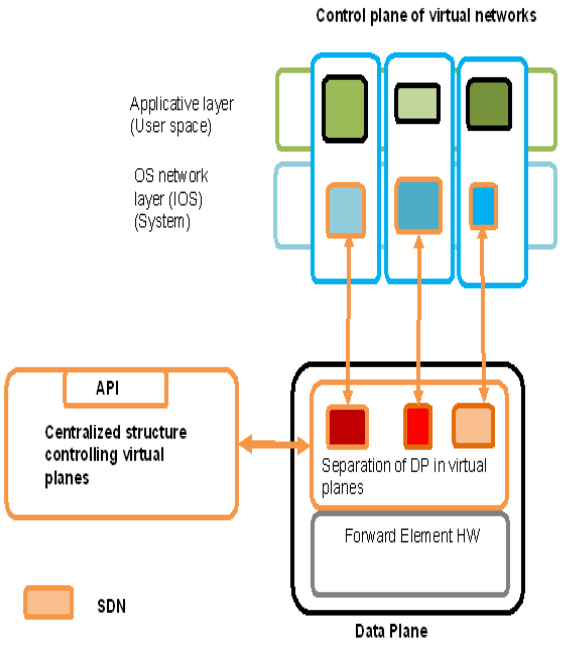


Figure 2.2: SDN paradigm

tualizable machine defined by Popek and Goldberg [40] in this virtualization driven networks.

Equivalence: A virtual plane piloted should behave as a set of fully-fledged network, regardless of other manageable planes, and especially be able to support the implementation of management protocols that provide a usable image recursively virtual network.

Efficiency: As for the field of IT , the measure of effectiveness involves determining the ability of virtual planes to allow direct execution of OS functions on the physical substrate network object.

Resources Control: The protocol stack supported by the virtual plane must have total control of the physical network resources. This implies that each network OS

needs access to physical network resources without explicitly taking into account the activities of other virtual planes that share the same resource. This is the battery access protocols that it is to manage and "schedule" the use of material resources. This principle is related to the principles defined above and characterizes the notion of isolation between the virtual planes.

2.1.6 OpenFlow

OpenFlow is the communication protocol between a logically centralized control plane and transfer plane, which is the most advanced in an SDN architecture. Indeed, it is implemented by many manufacturers such as NEC, HP, CISCO, Juniper...The section below briefly describes the principle, the advantages and disadvantages of the OpenFlow use in a network.

OpenFlow in its current version includes a protocol for controlling network devices and a toolbox for piloting Ethernet switches. The control of switches is done via a centralized controller which dialogues with each switch. The controller supervises and coordinates all switches in a network. Apart from the various services supported such as input and control counters, its main role is to manage the routing tables.

We call T_{ij} a table entry which defines a flow by:

- An input port i and an output port j
- A filter F_{ij} characterizing incoming packets on port i to be transmitted to the port j
- Actions to be performed on the packets (transfer, marking , destruction, transmission to the controller ...)

In a network node, the main role of the switch is to determine the flows which incoming packets belong by marking value P_i port that are in the filter F_{ij} and directing them to the output port P_j . Network equipment that meets the OpenFlow protocol

Ingress Port	Ether source	Ether dst	Ether type	VLAN id	VLAN priority	IP src	IP dst	IP proto	IP ToS bits	TCP/UDP Src port	TCP/UDP Dst port

Figure 2.3: Filter Field

must be able to filter and switch packets between Ethernet ports on criteria values of different fields up to level 4 (see figure 2.3). The filtering criteria and characterization of Ethernet frames are given in fields described in the table below:

Unrecognized packets are transmitted to the controller for analysis and definition of actions to be taken (e.g. characterize a new stream).

Interests

In this section, we will present the reason why we should use OpenFlow.

A first interesting point in the implementation of OpenFlow is the standardization and centralization of programming commands equipment that meets the standard.

A second point is related to the centralized management mode. The flow switching criteria can be selected and dynamically run on the equipments. This eliminates the implementation of complex "stacks" machines on the network and therefore problems of compatibility version between different manufacturers. Among other use cases, we can find:

- Implementation of differentiated treatment of packets input or during travel on the network depending on the type of flow they belong (i.e. split between several

virtual planes).

- The ability to manage data caches closer to the users in extracting, processing and storing individual flows. This type of treatment can be applied to content distribution, or maintaining databases within the "cloud computing".

- The marking of the packets for QoS management or for particular treatments.

Furthermore, the switching a priori is less expensive to implement than traditional routers.

Disadvantages

The obligation to centralize control makes the whole network vulnerable to failures and attacks. For large networks, the workload of the controller may become too large.

Motivation

The context of our study is the total or partial replacement of a network by switches. Networks based on switches controlled by an external controller may represent an interesting alternative to conventional router networks. Nevertheless, the absence of a direct exchange protocol between hardware nodes in our network might be problematic in cases of failure. Our aim is to propose a rerouting strategy for a network based on switches that ensures that whenever a link or node fails the traffic can always be rerouted to the destination by an alternative route. We now briefly explain this strategy in more detail.

We assume that each switch is programmed with filters that can determine the next hop for the incoming flow. We will recall here how a switch functions. For each embedded virtual network there is a specific filter. The controller sets the flow path

by programming the switches in the form of triples of type $(I, N, J)_F$, where I is the source port (node), N the current node, J the output port (node), and F the filter which indicates the destination. Clearly, for an incoming flow from a neighbour and a given destination, the scheme will give an output port. In the case of failure, for a given destination, only one of the two outermost nodes of the failed link needs to react by deviating the disturbed traffic towards one of its neighbours. The traffic is then routed according to the filters programmed in each node of the network. The proposed scheme therefore requires a local reaction only, making its implementation particularly easy in a distributed environment. This local reaction helps the network to operate normally and it can solve the problem of transient failures. We recall that a transient failure is a failure of short duration, while a persistent failure is longer. When it has been determined that a failure is persistent, the controller can recalculate the routing table for all the nodes in the network. In order to prevent the rerouted traffic (following a failure) causing disturbances in another part of the network, additional capacities are assigned to all the arcs in the network. We introduce a mathematical model that can not only calculate the rerouting paths in the network, but also optimize the total sum of additional capacity. Before explaining the model, we also present a proof of an existence theorem, ensuring that we can find a valid rerouting scheme for a network based on switches. Also, we have proposed approximate methods to solve this problem for network with larger size. We will present these elements in more detail in the following chapters.

When using Openflow, a virtual network can be defined by a filter. In fact, if we want to create virtual networks that implement different services, the filter can be the field type of service (Tos) in figure 2.3. This way, we will have different layers that have different services. In a virtual network, we will define another filter, which will be

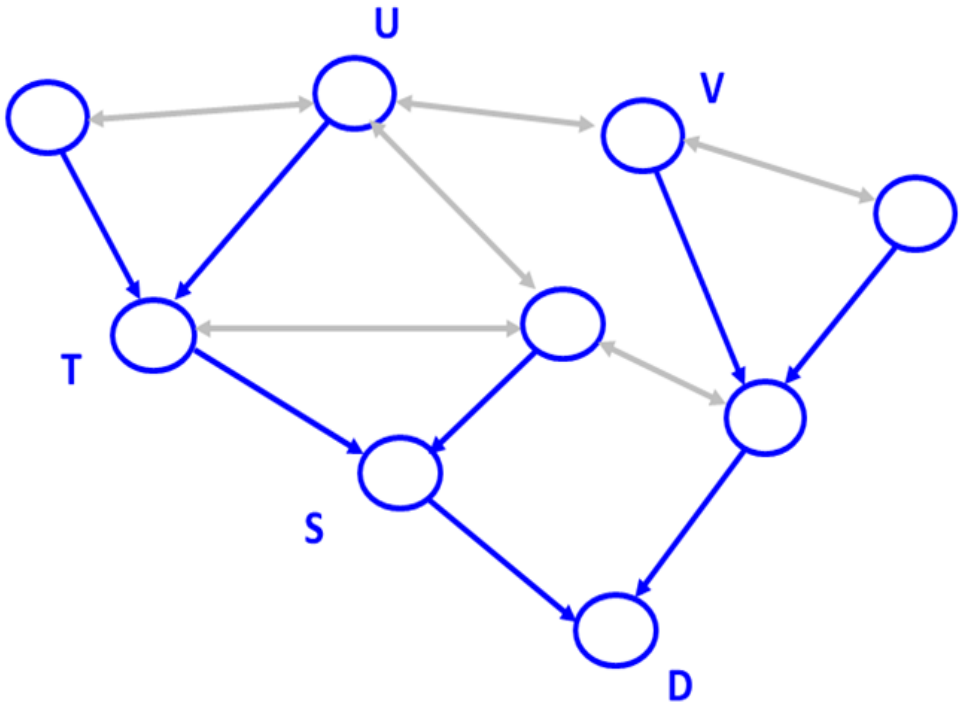


Figure 2.4: Tree topology in a virtual plane

used to route the traffic. The filter can be one of the fields mentioned in above sessions. The traffic that has this filter as destination will create a tree which has this filter as its root (Figure 10). We will use this tree as the base element in our theoretical studies in the later chapters. Our solution method will be applied for all the trees in this layer.

With the above definition, we have to apply the same method for all virtual networks.

Besides that we can also use one filter to define all virtual networks by using the notion of tree. As described before, one filter can define a tree. When two trees have their roots falling in the same physical node, they are then in different virtual layers. Thus, our resolution method, which is mentioned above and is described in more detail in next chapters, can be applied one time for all virtual networks.

In our study, for the moment, we suppose that there are two layers: one layer treated with priority and one layer best-effort. For the first layer, we will assure the quality of service in case of failure by applying our method. For the second one, the quality of service will not be treated.

2.2 Related works:

This study is about the treatment of failure for the network using Openflow in context of network virtualization. With that objective, we will study in this chapter the related works that concern network virtualization, Open-flow and failure resilience.

2.2.1 Works concerning network virtualization

Both in the academic and industrial world, studies are undertaken and lead to the proposal of making network virtualization. In particular, a number of studies and experiments that focus on architectures and methods of managing virtual networks are being conducted by university teams together more or less closely with industry.

Princeton University and the laboratories of AT & T (AT & T Labs), conduct academic studies on various topics:

- The reliability and resource allocation [13]
- Architectures virtual network service supported by network operators [14]
- Management and dynamic reconfiguration of virtual networks, based on ability to transfer virtual machines [15]

The University of Quebec at Montreal (UQAM) worked with the University Paris 6 (LIP6) on modelling solutions and allocation of resources for virtual networks [16].

The European AGAVE project brings together industry, universities and operators including France Telecom. It defines the notion of planes of virtual networks providing QoS that allow the interconnection of different service providers and operators to create networks of extended services.

Projects achieving infrastructure -based network virtualization has started, first in the U.S. then in Europe and Japan. It is a first step of experimental networks to operate large-scale networks virtual routers.

- Planetlab [17]

  This is the first major test networks with more than 1000 machines and 500 locations in over 30 countries. The machines support virtual routing nodes that are networked to form barrier layers of open routers. The goal is to make available to researchers a platform for realistic test in terms of users, traffic and measures, allowing them to deploy and evaluate theoretical models.

- GENI: Global Environment for Network Innovations [18]

  It is a set of initiatives to promote environmental testing large-scale network. The guiding concepts are inspired by "PlanetLab" which GENI incorporates the principles. It also advocates, functional enhancements and administration which offers, among other things, the possibility for a given change in time or to interface with the internet virtual network. The VINI project [19] implemented by Princeton University and built on the GENI model is based on the PlanetLab architecture.

- FIRE: Future Internet Research and Experimentation [20]:

  This is a European network scale intended for academic and industrial experimentation. It allows the interconnection of existing architectures such as Planet-

Lab tested with new architectures.

- CABO: Concurrent Architectures are Better than One.

  It started with "AT & T Labs-Research" and taken over by the "Georgia Institute of Technology" project and Princeton University. It offers a virtualized architecture solution from start to finish in a context of separation of roles between the operators responsible for management of network infrastructure and service providers.

- The Japanese AKARI project [21].

  Supposed to lay the foundation for overlays of new generation networks by providing new network architectures and services, this project aims to integrate the most advanced technologies such as optical switching or new concepts for network virtualization.

- NEC solution

  The system in this solution consists of a network of OpenFlow switches and a controller. It is suitable for Data Center internal network. Indeed, it allows you to configure and control a set of virtual networks in a multi-tenant environment.

- RouteFlow [28]

  This solution allows controlling an OpenFlow physical infrastructure by a classical control plane without having to rewrite the control part

- Juniper solution: JunOSV App Engine [29]

  This solution allows virtualizing networking functions currently supported by boxes arranged either at customer or in our network. .

- Cisco OnePK(Cisco Open Network Environment Platform Kit)

  This solution makes Cisco equipment programmable and allows its integration with orchestration tools.

- European FEDERICA project [30]

  FEDERICA is part of FIRE (Future Internet Research) pan- European projects. This solution seems to be a case of complete network virtualization.

- European project SAIL

  SAIL aims to define a network architecture capable of connecting rapidly and automatically users and applications while optimizing the use of available resources.

- European project OFELIA (OpenFlow in Europe: Linking Infrastructure and Applications) [31]

  Its purpose is to create the first European experimental network based on OpenFlow technology. It permits to create automatically virtual networks

- FlowVisor [32]

  It is a solution that only makes sense in the case of OpenFlow network. A large number of virtual networks seem difficult to maintain with this method. The definitions of implicit layers seem hard to manage.

- Virtual routers

  So far, two manufacturers (Cisco and Juniper) propose solutions “full virtualization” implementable in the core network:

  - Proposal to introduce virtual Cisco routers in the networks of operators dedicating routing planes for differentiated services (data, VoIP etc...) " Router Vir-

tualization in Service Providers " [33]

- After the offer of the Cisco Nexus 1000V virtual switch designed to facilitate the integration of internal communication services to Data Centers , offering CSR 1000V is designed to enable a customer to implement virtual routers in the Cloud, and the ” interface with its own network [34].

- Juniper Firefly is an alternative virtual routers which can be deployed in servers in the data center. These solutions are considered in the data center.

2.2.2 Works concerning failure resilience

The problem of failure resilience has been widely investigated in literature. Many works have been presented devoted to multi-protocol label switching (MPLS) [25] and real-time systems [27]. Nevertheless, when dealing with switch based networks the literature is less abundant. Our restoration scheme is pre-calculated, while other protocols in network virtualization [26] are pro-active. As a result of its pre-calculated scheme, our protocol does not need any communication time between network nodes, which guarantees the recovery time is mostly the failure detection time. We will present principal works studying similar problems for IP networks and switches based networks.

Rerouting in IP networks

Xi and Chao [11] propose a method for calculating back-up paths which permits to reroute the traffic in case of link failure. The proposed scheme, called IPFRR (IP Fast Reroute), uses two types of port, primary and back-up ports. Normally, the node uses the primary port to switch the sorting traffic. When there is a failure on the primary port, the node will use the back-up port to reroute the traffic. Also, when the traffic

comes from the primary port, the back-up port will be used to switch the traffic. This packet forwarding policy makes the traffic follow the reverse routing path starting from the upper node of the failure link until a node where it can jump to another node and use its primary port. We will see that this method is similar to the one we present in next chapter. They propose also an Integer Linear Programming (ILP) model to minimize the number of backup ports. However the model does not take into account the conflict problem, neither minimizes the capacity added to the network.

Another similar method is presented in Wang and Nelakuditi [8]. It determines the next hop for traffic when the destination and entering interface of traffic are provided. On the other hand, it calculates the rerouting path using shortest paths with the metrics stored in each node. Because the graph is symmetric, this method becomes identical to the method of Xi and Chao [11].

A number of methods have been proposed for IP fast rerouting, in order to solve the problem of transient failure. These methods nevertheless have certain limitations:

- With loop-free alternate mechanism based methods [2], there is no guarantee that traffic can be rerouted for all destinations. These methods can only help to reduce the number of lost packets in an IP network.

- Not-via addressing [3] and tunneling [4] mechanisms require the encapsulation and decapsulation of packets, while in multiple routing configurations mechanism [5], the packets need to carry configuration information. With the appearance of optical networks these methods that modify the packets are not recommended.

Finally, some works dealing with multicast trees can be useful in treating IP transient failures. At first, a method is proposed in [9]to create simultaneously two routing trees,

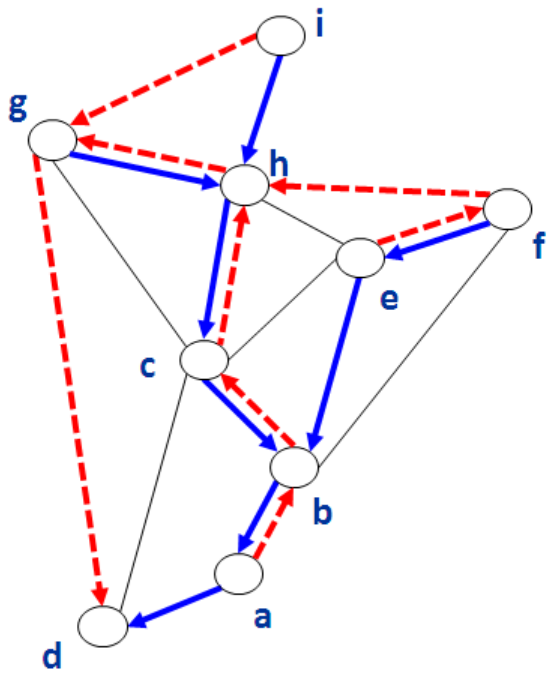


Figure 2.5: Two routing tree 1

which are named B (Blue) and R (Red). This proposed method is used for the case of multicast. In this method, after eliminating any vertex or any edge in the graph, the source s remains connected to all vertices through B and/or R. We notice that reversing the solution of [9] can provide a free-conflict routing scheme as defined above. We can see an example of this method in the figure 2.5. The two routing trees are described by the bold arcs and the bold dotted arcs. However, this doesn't work when the routing tree is fixed, which is the case for our problem. We can see clearly the reason in the figure 2.6. When the nominal routing tree covers all the directed arcs to destination d, we cannot construct the two routing trees by using [9].

Another work is presented in [10] that proposes fast reroute solution for multimedia IP backbone design. The proposed method creates two multicast trees which are a shared tree and a shortest path tree. This method uses the same principle as [9], but in

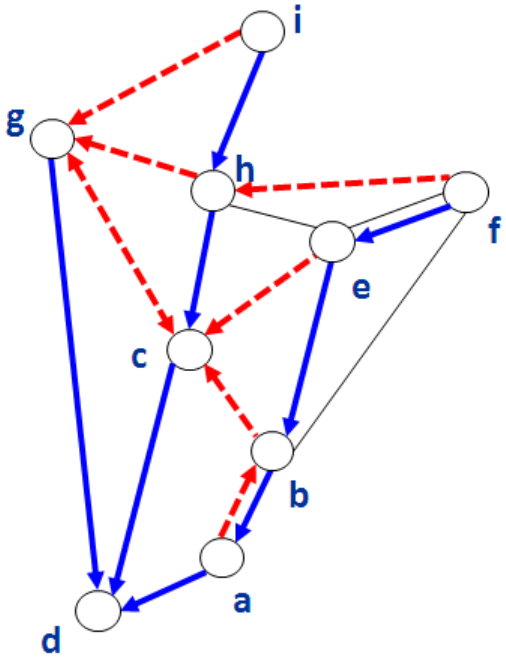


Figure 2.6: Two routing tree 2

addition it uses link's weights to create shortest path routing tree. Again, in this case, the method cannot be used for a given routing tree.

From above we can state that two main points need to be addressed: proposing a generic free-conflict routing scheme that allows dealing with any given routing scheme; ensuring more flexibility in the choice of alternative routing paths. We will answer to the first point in chapter 3, and in chapter 4 we address the second one.

Rerouting in switches based networks

In this section, we present several works about failure resilience that are applied in switches based networks. [35] proposes an OpenFlow-based segment protection (OSP) scheme which enables fast recovery in network composed of OpenFlow-based Ethernet switches. This scheme does not calculate back-up paths. Indeed, it supposes that we

have in our hand a rerouting scheme, it shows us how to use this scheme in context of OpenFlow network. This proposed method avoids the intervention of controller upon failure, thus assures a fast recovery time.

[36] demonstrates their proposed bootstrapping and failure recovery mechanisms for an in-band OpenFlow network. With this protection mechanism, failure recovery happens before the controller detects the failure. Indeed, the traffic is redirected from the working path to the restoration path without the need to inform controller.

[37] shows that OpenFlow can be deployed in carrier-grade networks in certain conditions. In fact, carrier-grade networks must have a strict requirement: they should recover from a failure within 50 ms interval. This paper only considers the link failures in the data plane.

[12] pre-calculates the back-up routes in case of failure using Openflow for the IP network, but it pleads for the modification of the header of package, which as said above are not recommended in practice.

In [41], Sharma proposes a restoration scheme for OpenFlow networks. When a failure appears, the node will contact the controller via packet-in message. We recall that packet-in messages are messages that are sent from a switch to controller while packet-out messages are messages that are sent from the controller to the switch. Controller will calculate the new path and then resends this new path to the node via packet-out message. This method is pro-active. Although, it may help to reroute the traffic before the time limit of the packet is expired but it is slower than our method when the rerouting paths are pre-calculated.

In order to have an autonomous restoration system in switches based network, we need to propose a method which is pre-calculated and which does not need to modify the packets. Our methods presented in next chapter satisfy the above conditions. It

helps to reroute the traffic without intervention from the controller. The methods [11] and [8] can also be applied for switches based networks. However they does not help to solve the dimensioning problem. We will compare the results obtained by their methods and our methods in chapter 4.

Chapter 3

Existence theorem and mathematical model

In this chapter, we illustrate the proposed restoration scheme through an example and prove formally the existence of a free-conflict restoration scheme. Next, we propose a mathematical model that permits to calculate a free-conflict restoration scheme that optimizes the dimensioning of the network for a given routing scheme.

3.1 A free-conflict restoration scheme for switch-based networks

In this section we detail our scheme for routing and rerouting in cases of link failure. The considered network is modelled using a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, directed and symmetric (i.e., each link includes the two opposite arcs), composed of a set of n nodes $\mathcal{V}$ and a set of m links $\mathcal{E}$. For any destination in the network the traffic is routed to the destination through a tree, called nominal routing tree. This tree is constructed using specific criteria; for instance, it might be the shortest path tree. In our study we assume that the routing is given. In the case of failure of an arc or link (both arcs composing the link are then concerned), the upstream extremity node will deviate the disturbed traffic to one of its neighbours. From this moment, any node traversed by the disturbed traffic will route it according to a free-conflict routing scheme. In figure 3.1, we report a free-conflict scheme. It can be noticed that all alternative paths to the same destination that are not in conflict (see above), can be embedded in a free-conflict scheme. Hence, there exist two free-conflict paths to destination G coming from A. Then, this situation can be represented at node A by two different inputs for destination G, as shown in the figure: one comes from input 1 and goes through output 2 and the other comes from 4 and continues through 3. Then, given a precomputed failure resilient free-conflict routing scheme, a rerouting procedure works as follows:

- In case of a (link) failure the upstream node initiates the rerouting procedure by deviating the traffic to some destinations to one of its neighbours. Such information relevant to failures of links is assumed known from each corresponding upstream node for any destination in the network.

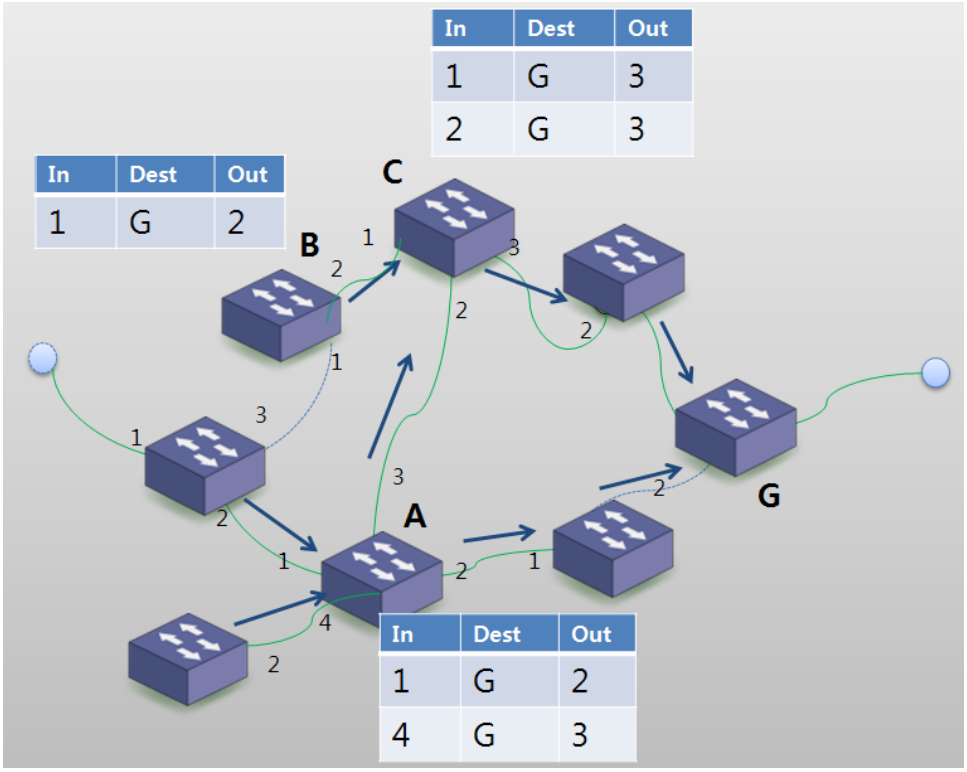
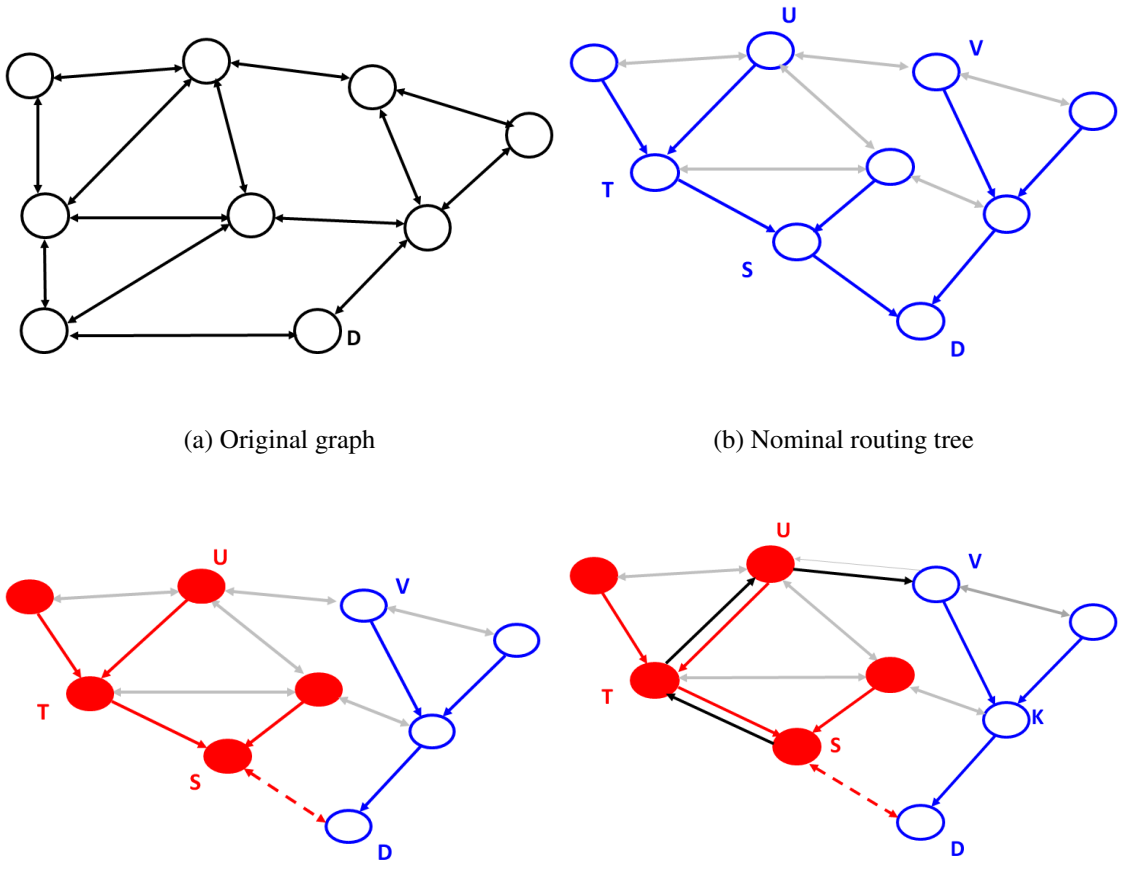


Figure 3.1: A free conflict routing scheme

- Other nodes in the network don't need to take any specific action. They will forward the traffic according to their routing tables. For any couple (input, destination) it finds the appropriate output and sends the traffic through it.

3.1.1 An example

In this section we present a complete example of the proposed restoration scheme (Figure 3.2). The original graph is shown in Figure 3.2(a). If we apply to all the nodes of the graph a filter with respect to a destination D, the traffic flow that corresponds to this filter uses a directed tree which converges to destination D (Figure 3.2(b)). When the link (S, D) fails, the routing tree will be divided into two parts: the red part (the filled nodes), and the blue part, whose nodes are unfilled (Figure 3.2(c)). All the



(a) Original graph

(b) Nominal routing tree

(c) Failure situation

(d) Rerouting scheme

Figure 3.2: Small example of the restoration scheme.

traffic to destination D that transits via node S can no longer use the link (S, D), and will instead be rerouted by the path (S, T, U, V, K, D) that was pre-calculated by our restoration scheme (Figure 3.2(d)). This alternative path connects the red part to the blue part and it will not interfere with the nominal routing. In this scheme, in case of failure of the link (S, D), node S is advertised of the failure and it will take the necessary action to restore the traffic through an alternative rerouting path while the other nodes will operate normally as programmed. We can see that the traffic towards

destination D over the arc (T, S) is rerouted over the arc (S, T), but this will not cause any looping problem because of the configuration of filters. When the node T receives traffic coming from U and knows that the destination is D, T will transfer the traffic to node S. In the case of (S, D) link failure, node S will send back the traffic through the reverse routing path (the network is assumed directed with arcs in both directions for each link) to T. Knowing that traffic is coming from S and is heading towards D, T will transfer this traffic to U, which in its turn will transfer it to V. Then, the traffic will be transferred from V to K and on to destination D following the *nominal* routing tree for destination D. Therefore, there is no looping problem and the traffic is rerouted without causing any conflict. Nevertheless, the problem of existence of a valid routing/rerouting scheme, i.e. without conflict, for all possible non-simultaneous link failures is not straightforward.

3.1.2 Existence of a restoration scheme

In this section we study the question of whether there exists a rerouting solution that gives rise to no conflicts. We shall make the following assumptions:

- the graph is assumed to be directed with arcs in both directions for each link of the network;
- there exist at least two disjoint-arc paths between any two nodes of the graph;
- for each destination the nominal routing scheme follows a fixed tree;
- only one link failure can occur at a time.

Our goal is to achieve a *full* free-conflict scheme handling both nominal and failure situations. This full routing scheme is composed of a nominal routing scheme realized

by a routing tree for each destination (which is given for all destination) merged with the restoration scheme in such a way that there is no conflict with respect to filters. One can notice that routing for both nominal and failure situations is done with respect to a given destination l and there is no interaction between routing schemes for distinct destinations. This suggests restricting the proof of existence of free-conflict scheme to the case of a fixed destination without loss of generality. The same can be reproduced for any other destination and merging them leads to a free-conflict scheme as well.

Theorem 1. *For single link failures perturbing the nominal routing tree there exists a rerouting plan without conflict for any destination l.*

Proof. Let A be the routing tree to the destination l. l is therefore the sink of A. Let (p_1, q_1) be an arc of A subject to failure. We assume that this arc fails and we must find a rerouting scheme without conflict. We remark that (q_1, p_1) does not belong to A. This is clear as for a given destination only one of arcs (directions) can be part of the routing tree. Hence, this means that both link and arc failures have the same impact on traffic lost for a given destination and we can restrict ourselves to arc failures. Without loss of generality, we consider in this proof the problem of the failure of the arc (p_1, q_1). p_1 is the sink of a sub-tree A_1 whose nodes are colored in red. The other vertices in the tree are colored in blue. Without loss of generality we assume that all vertices are part of the tree A ($|A| = n$) and this is true for any destination l.

We know that there are at least two disjoint paths in the initial graph going from p_1 to l. Given the assumption of two-link connectivity the upper part that includes p_1, but does not include the vertex l, must contain at least two outgoing arcs. Therefore there are at least two arcs going out of A_1 (Figure 3.3). Since one of these arcs is (p_1, q_1), there exists a path μ from p_1 that visits the vertices of A_1, and connects a vertex of A_1, which is red, to a blue vertex. So there is at least one arc (i, j) (called also a bridge) of

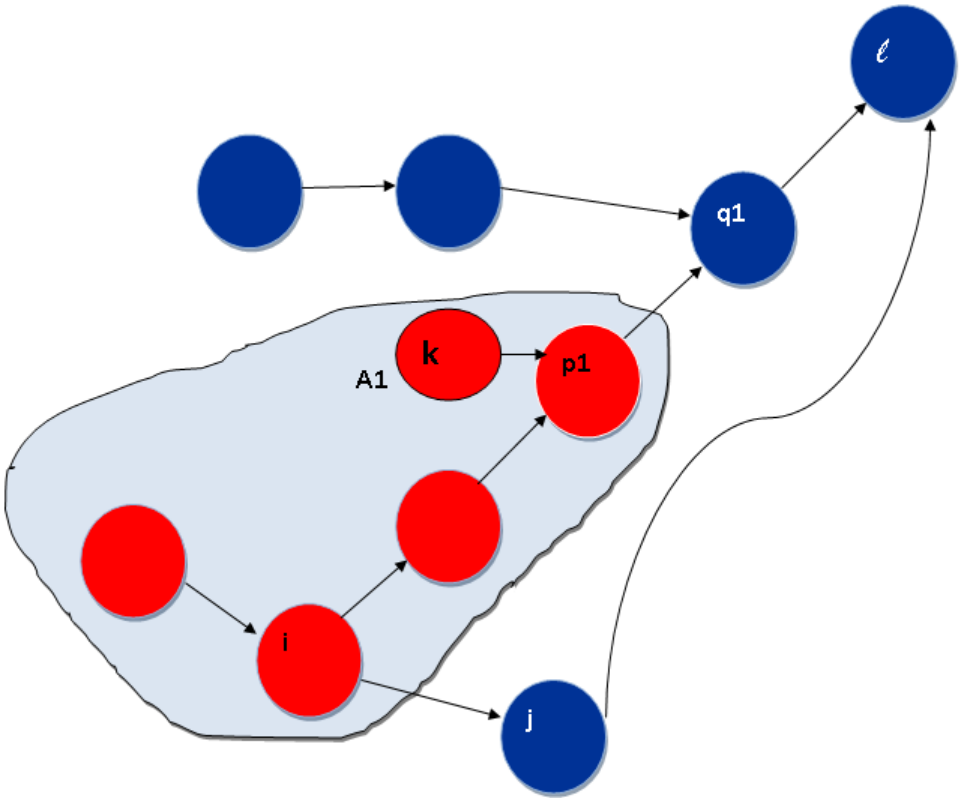


Figure 3.3: Proof

μ connecting the red vertices to the blue vertices (Figure 3.3). We associate with this arc a rerouting path for the failed arc. Let k be a red vertex of A_1 which is affected by the failure. Traffic to destination l and coming from k goes first to p_1, then it follows the the reverse path on the nominal routing tree (it is assumed above that arcs in oposite directions are present for each link) from p_1 to i. It uses bridge (i, j), then from j to destination l follows the nominal routing tree. According to the above choice of the rerouting path, traffic coming from different sources in A to destination l associated with the breakdown of (p_1, q_1) necessarily follows rerouting paths without conflict.

We now consider the $n - 1$ failures of arcs of the tree and choose the different arcs (i, j). We number the arcs of the tree by decreasing order as we approach the sink l,

and consequently choose the arcs subject to failure in successive order of increasing numbers. Let (p_r, q_r) be one of the arcs subject to failure under consideration, and let us suppose that we have chosen arcs $(i_1, j_1), (i_2, j_2) \dots (i_{r-1}, j_{r-1})$ as bridges for constructing the rerouting paths with respect to previously examined failures. p_r is the root of tree A_r. We consider two cases. The first case is when we have chosen for an arc (p_s, q_s), with s strictly smaller than r, a bridge (i_s, j_s) whose extremity i_s is in the tree A_r and the other extremity j_s outside the tree A_s. Clearly at this case the tree A_r is contained within A_s (Figure 3.4). Please note that or any $s < r$ we have either $A_r \subset A_s$ (the path from p_r to destination l goes through p_s) either $A_r \cap A_s = \emptyset$. We therefore choose arc (i_s, j_s) as bridge for the tree A_r. Note that there exists at most one bridge with this property. This can be deduced from the inclusion property of trees. Indeed, having two such arcs (i_s, j_s) and (i_t, j_t), with respect to failures (p_s, q_s) and (p_t, q_t) with $s < t < r$ (case $t < s < r$ can be solved in a similar way), means that (i_s, j_s) is also a bridge for A_t and it should have been chosen instead of (i_t, j_t). In the case when $A_s \cap A_r = \emptyset$, there is no rerouting arc with the above property. Then, we choose any arc (i_r, j_r) (Figure 3.5) that connects A_r to its complement.

We need to show that the rerouting has no conflict. We demonstrate this by recurrence on the number of rerouted arcs. We consider that we have already rerouted $r - 1$ arcs in the tree. By the recurrence hypothesis, we assume that there is no conflict for the first $r - 1$ reroutings. We verify that the r-th rerouting built as above also has no conflict with the first $r - 1$ reroutings. Regarding the rerouting in the outside part of the tree A_r, that is to say the part that is in common with the nominal routing, there is no conflict by construction. Even if it uses the same arc with some previous rerouting path, in this part it will follow the same rerouting path as far as destination l, and so it is without conflict. We must also check that there is no conflict for the part where it

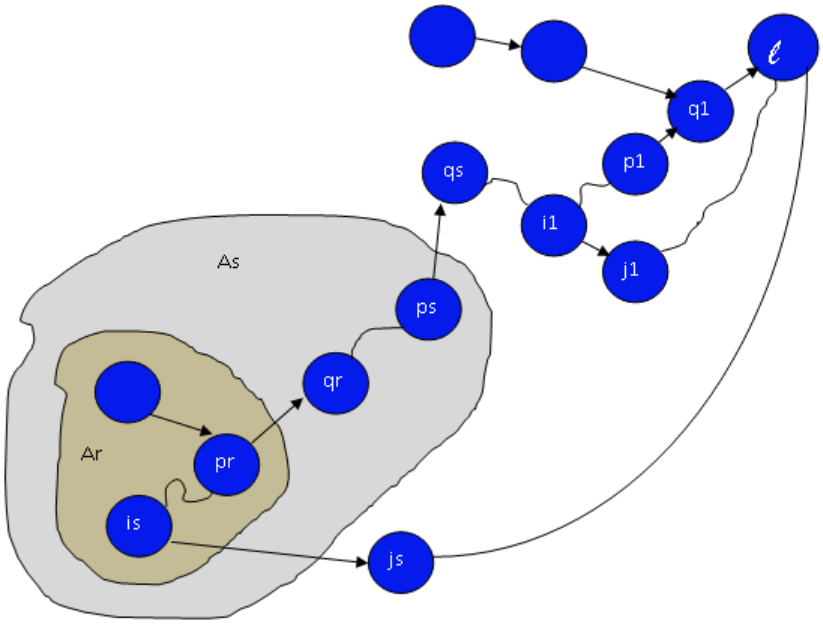


Figure 3.4: Recurrence Hypothesis

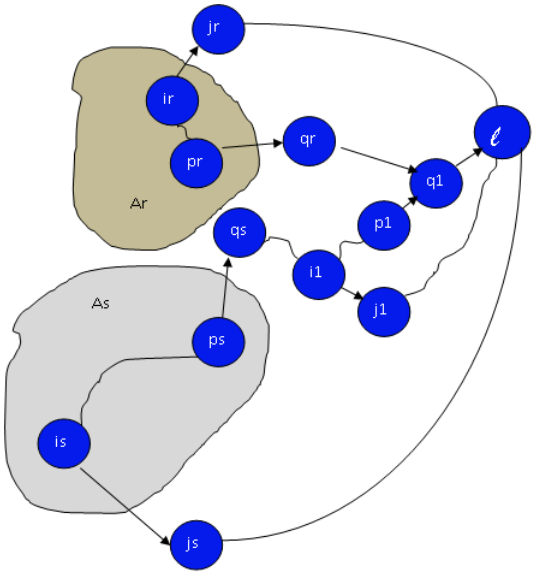


Figure 3.5: Recurrence Hypothesis 2

goes in the opposite direction in the routing tree, which means verifying that there is no conflict in the two cases considered above. In the first case, where an arc (i_s, j_s) has been chosen in the tree A_r, clearly no conflict is present in that part of the tree because

A_r will use the same arcs until i_s and continue with the same bridge (i_s, j_s) between its red part and its blue part. In the second case, the part that ascends the tree can have nothing in common with the other rerouting arcs, since this would imply the existence of (i_s, j_s)! Therefore, there is no conflict in this case either. We can conclude that the property remains true to the order r, and we have therefore demonstrated by recurrence the absence of conflict. □

3.2 Mathematical model

The routing/rerouting scheme proposed in this study requires the single path routing of traffic demands both in the nominal situation and in situations of failure. An additional difficulty for this strategy comes from the conflict avoidance constraint. All this makes modelling and solving using an arc-path flow formulation extremely hard. We have therefore opted for an adapted variant of the arc-node formulation that is better suited to expressing these type of constraints. At this stage, the mathematical formulation concerns only the spare-capacity assignment problem, (that is computing the optimal capacities to be added for recovery needs, the routing being given). Furthermore, we assume that in cases of failure all disturbed traffic demands will be rerouted along the same single path. The model given below deals with link failure situations, but node failure situations can be handled in a similar way. We first list the notation used in the mathematical model.

Parameters:

- Arc: all arcs of the graph.
- E: set of links.

- V: set of nodes.
- Triple: all triples (i, k, j), where i, k, j are nodes of the graph and (i, k) and (k, j) are two adjacent arcs, i being different from j.
- A^l: set of arcs of the routing tree to the destination l.
- red_v^l: sub-tree of tree A^l with sink v. Recall that in the case of failure the tree is divided into two parts: the isolated part, i.e. the red part, and the blue part. The alternative path will reroute traffic from the red part to the blue part.
- $blue_v^l$: $A^l - red_v^l$.
- 0: a fictitious node used to divert traffic in case of failure. We introduce the fictitious node 0 that will be used for all failures. For a given failure ($v,\ l$), the traffic to l will be rerouted along a single path from 0 to l and starting with the arc (0, v).
- T_v^l: total traffic for l that passes through the node v, v being the node that detects the failure. In fact, the failure is characterized by a source v and a destination l. This is because the nominal routing is done through a tree, and we need to reroute only the nominal traffic passing via this tree. Hence, for a destination l, each node in the tree is concerned with only one (failure) arc in the tree, and this node is necessarily the origin of the failed arc.
- α_{ab}^{lv}: a binary coefficient equal to 1 if the arc (a, b) belongs to the nominal routing path from v to l excluding the failed arc.

Decision variables:

- y_{ikj}^{lv}: this binary variable indicates whether the alternative path to destination l and for a given failure contains arcs (i,k) and (k,j), the node v being the node that detects the failure.

- x_{ikj}^{l}: this binary variable indicates the rerouting scheme to the destination l. It takes the value 1 if there exists a failure whose alternative path to destination l contains arcs (i,k) and (k,j). In other words, this variable takes the value 1 if there exists v that y_{ikj}^{lv} is equal to 1.

- r_{ab}: additional capacity assigned to the arc (a,b).

We will explain the mathematical model in detail in the below paragraphs.

Firstly, we have to introduce the objective function

$$\min \sum_{(a,b)\in Arc} r_{ab} \tag{3.1}$$

Recall that the objective is to minimize the sum of additional capacity allocated to each arc. This objective function (3.1) will allow us to evaluate the ratio between the additional capacity and the installed capacity.

Flow constraints and rerouting. The constraints in this paragraph are the constraints of rerouting. They will be able to construct a rerouting path from the node fictive 0 to destination l (Figure 3.6).

$$\sum_{vj\notin A^l,\ j\in neighbor\ of\ v} y_{0vj}^{lv} = 1,\ v \in V,\ l \in V \tag{3.2}$$

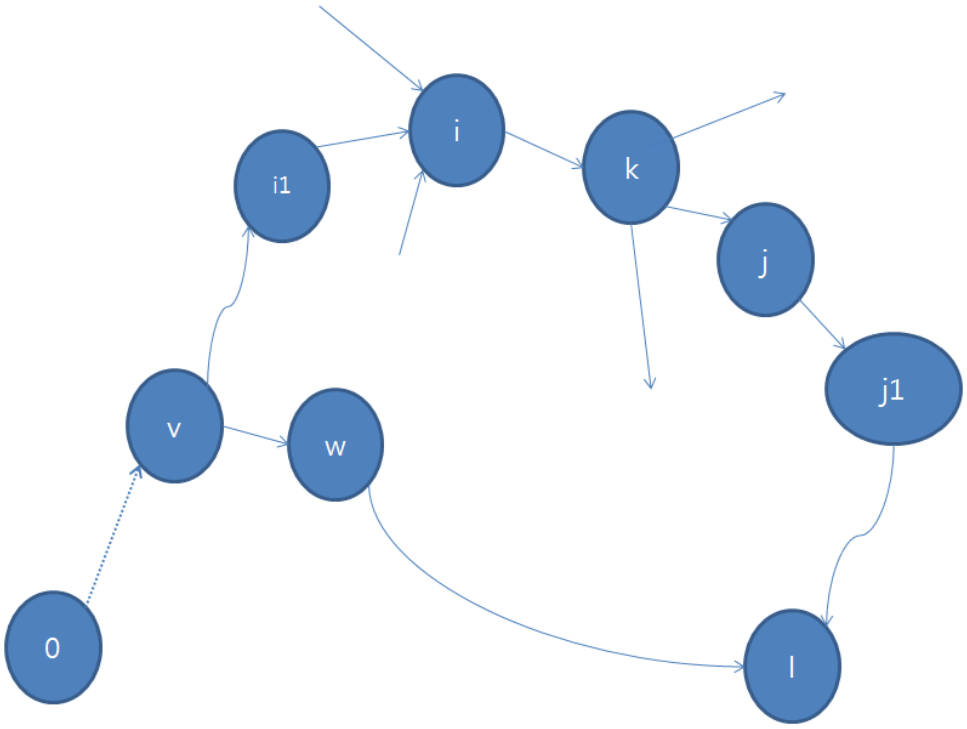


Figure 3.6: Model illustration

Constraint (3.2) implies that there exists exactly one arc coming out of v for the disturbed traffic from v to l. As we describe above, all the traffic will be rerouted by only one path. This path begins by the node fictive 0, then it uses the node v which is the node that detects the failure and has to reroute the traffic(Figure 3.6). This constraint allows to control the number of outgoing arc from the node v. Because the rerouting path must not pass by the failed arc,(v, j) must not be the failed arc.

To avoid the problems of looping and conflict, the alternative path should not contain any arc of nominal routing in the red part of the network.

$$y_{ikj}^{lv} = 0,\ l \in V,\ v \in V,\ i \in red_v^l, (i,\ k) \in A^l,\ (i,\ k,\ j) \in Triple \qquad (3.3)$$

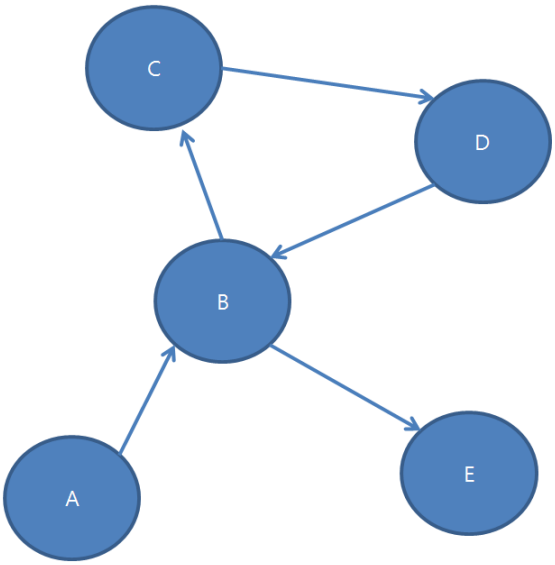


Figure 3.7: Cycle in rerouting path

(3.3) ensures that this condition holds. Indeed, when the rerouting path uses one arc of the nominal routing tree, it has to continue to use the arcs of the nominal routing tree until it reaches the destination l. If this arc is in the red part of the network, the rerouting path will follow the nominal routing tree to the failed arc. This will cause the looping problem and the traffic will not be rerouted to the destination.

While testing our mathematical model, we find that there could be a case when there is a circle in our rerouting path (Figure 3.7). Therefore, we add the following constraint to avoid this problem:

$$\sum_{i \in V, j \in V | (i,k,j) \in Triple} y_{ikj}^{lv} \leq 1, l \in V,\ v \in V,\ k \in V \tag{3.4}$$

Constraint (3.4) ensures that there will be no looping in the network, since the alternative path can transit once at the most via any given node. This constraint also ensures that the alternative path from v to l is elementary.

We will introduce then the flow constraints that ensure the continuity of the alternative path.

$$\sum_{i_1 \in neighbour\ of\ i} y^{lv}_{i_1 ik} = \sum_{j \in neighbor\ of\ k} y^{lv}_{ikj},\ l \in V, \\ (i,k) \in Arc, i \neq 0\ , i \neq v, k \in V \backslash \{l\} \tag{3.5}$$

$$\sum_{j \in neighbor\ of\ k} y^{lv}_{vkj} = y^{lv}_{0vk},\ l \in V,\ v \in V,\ (v,k) \in Arc \tag{3.6}$$

$$y^{lv}_{kjj_1} - y^{lv}_{ikj} \geq 0,\ j \in blue^l_v \backslash \{l\}, \\ (j,j_1) \in A^l,\ (k,j) \in A^l, \forall l \in V, \forall (i,k,j) \in Triple \tag{3.7}$$

(3.5) is the constraint of flow conservation. As illustrated by Figure 3.6, the total amount of traffic entering i equals the total traffic leaving k. Because there exists only one alternative path for a failure and a destination, there is only one incoming stream and one outgoing stream. This type of flow conservation does not hold for the node v that detects the failure; we therefore have a constraint of flow (3.6) for this case. In the blue part, if the path uses an arc of the initial routing, it must continue to destination l, and we thus have (3.7).

Next, it will be the constraint when the rerouting path reaches destination l.

$$\sum_{(i,\ k,\ l) \in Triple} y^{lv}_{ikl} = 1,\ l \in V,\ v \in V \tag{3.8}$$

Constraint (3.8) ensures that the alternative path will reach the destination l and it will be the only one. This constraint consequently ensures the single path routing requirement.

Given the complexity of the above LP formulation, we think that it will be useful to

briefly discuss the validity of the model. We recall first that the traffic to be rerouted is characterized by the couple $(l,\ v)$. We list below all the requirements that a rerouting path from v to l must satisfy:

1. The path does not use the nominal routing when it goes through the red sub-tree of the network.

2. If the path uses an arc of the nominal routing when it is in the blue sub-tree of the network, it must continue to use the nominal routing until it reaches the destination l.

3. It is an elementary path.

Clearly, constraint (3.3) ensures the first requirement and constraint (3.7) ensures the second. The third requirement comes first from constraints (3.2) and (3.8) specifying constraints for the origin (v) and the destination (l), secondly from constraint (3.4) that ensures the elementarity of the path, and finally from the Kirchoff constraints.

Avoiding conflict constraints. After the construction of the rerouting paths, we have to assure that theses rerouting path have no conflict between them.

$$\sum_{v \in V} y_{ikj}^{lv} \geq x_{ikj}^{l} \geq \frac{\sum_{v \in V} y_{ikj}^{lv}}{cardinal(V)},\ (i,k,j) \in Triple,\ l \in V \tag{3.9}$$

$$\sum_{j \in neighbor\ of\ k} x_{ikj}^{l} \leq 1,\ (i,\ k) \in Arc,\ l \in V \tag{3.10}$$

$$x_{ikj}^{l} \in \{0,\ 1\}, \forall (i,k,j) \in Triple, \forall l \in V \tag{3.11}$$

$$y_{ikj}^{lv} \in \{0,\ 1\}, \forall (i,k,j) \in Triple, \forall l \in V, \forall v \in V \tag{3.12}$$

(3.9), (3.10), (3.11), (3.12) ensure the absence of conflict. We have at first the constraints (3.11),(3.12) express the value of the variables x^l_{ikj} and y^{lv}_{ikj} is 0 or 1. Then, constraint (3.9) expresses the fact that $x^l_{ikj} = 1 \Leftrightarrow \exists v : y^{lv}_{ikj} = 1$. Indeed, if $x^l_{ikj} = 1$, it is obvious that there exists a v such that $y_{ikj} = 1$, because $\sum_v y^{lv}_{ikj} \geq x^l_{ikj}$. And vice-versa, if $\exists v : y^{lv}_{ikj} = 1$, then $\sum_v y^{lv}_{ikj} \geq 1$ and consequently $\frac{\sum_v y^{lv}_{ikj}}{cardinal(V)} \geq \frac{1}{cardinal(V)}$. We can deduce that $x^l_{ikj} \geq \frac{1}{cardinal(V)}$. Because $x^l_{ikj} \in \{0, 1\}$, we have $x^l_{ikj} = 1$.

Therefore, the constraint of absence of conflict can be expressed by (3.10): $\sum_{j \in neighbour\ of\ k} x^l_{ikj} \leq 1$, because if we use arc $(i,\ k)$ for the alternative path, we need only use at most one arc $(k,\ j)$ (Figure 3.6). Indeed, when two rerouting paths use the same entering arc $(i,\ k)$, the constraint ensures that it could use at most one outgoing arc $(k,\ j)$, which means there is no conflict.

Capacity constraints. In this paragraph, we present the constraints that concern the capacity added to the arc.

$$\sum_{l \in V | (v,w) \in A^l} \sum_{i \in neighbor\ of\ k,\ i \neq j} y^{lv}_{ikj}.T^l_v + \sum_{l \in V | (w,v) \in A^l} \sum_{i \in neighbor\ of\ k,\ i \neq j} y^{lw}_{ikj}.T^l_w \leq r_{kj} + \sum_{l \in V | (v,w) \in A^l} \alpha^{lv}_{kj}.T^l_v + \sum_{l \in V | (w,v) \in A^l} \alpha^{lw}_{kj}.T^l_w, (k,j) \in Arc,\ k \neq v,\ k \neq w,\ (v,w) \in E \tag{3.13}$$

$$\sum_{l \in V | (v,w) \in A^l} y^{lv}_{0vj}.T^l_v \leq r_{vj} + \sum_{l \in V | (v,w) \in A^l} \alpha^{lv}_{vj}.T^l_v, \quad (v,\ j) \in Arc,\ j \neq\ w,\ (v,w) \in E \tag{3.14}$$

$$\sum_{l \in V | (w,v) \in A^l} y^{lw}_{0wj}.T^l_w \leq r_{wj} + \sum_{l \in V | (w,v) \in A^l} \alpha^{lw}_{wj}.T^l_w, \quad (w,j) \in Arc,\ j \neq\ v,\ (v,w) \in E \tag{3.15}$$

For each failure of link $(v,\ w)$ (Figure 3.6), constraint (3.13) consider rerouted paths for both arcs $(v,\ w)$ and $(w,\ v)$, and only trees that contain the arc failure are involved. They also take into account the released bandwidth on the initial routing paths. Indeed, the first term of the left part of the constraint (13) signifies the capacity of traffic rerouted by the failure of arc $(v,\ w)$ that passes by arc $(k,\ j)$ and the second one signifies the capacity of traffic rerouted by the failure of arc $(w,\ v)$. In addition, the first term of the right part of the constraint (3.13) signifies the capacity added to arc $(k,\ j)$ while the second one and third one signifies the capacity released for arc $(k,\ j)$ if this arc is on the nominal routing paths for the failure of arc $(v,\ w)$ and $(w,\ v)$ correspondingly. In brief, this constraint gives us the bound of the capacity added to arc $(k,\ j)$. (3.14) and (3.15) are special cases of (13) for the nodes that detect failures v and w.

3.3 Numerical results

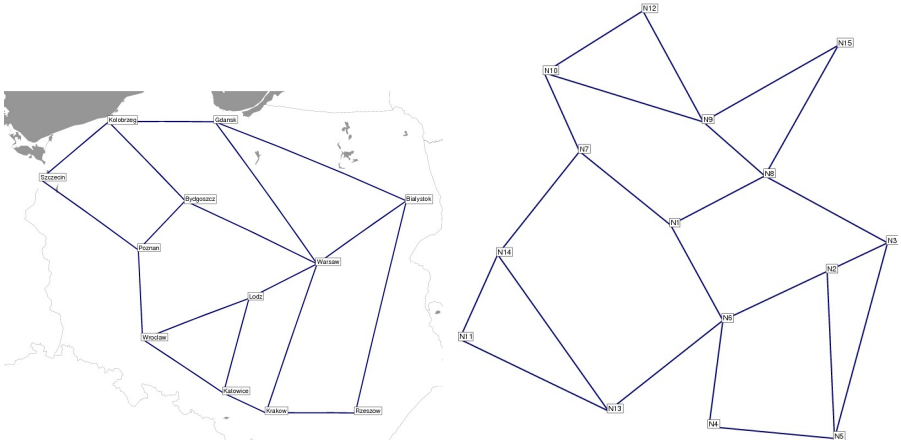
We ran our program on IBM ILOG OPL IDEA using IBM ILOG CPLEX 12.1.0. The calculations were performed on a virtual computer with the following configuration: Quad Core 1.8GHz, 8.00 Go RAM, 12Mb cache.

We applied our model to four networks, namely one test network (7 nodes, 9 links) and three real networks: Polska (12 nodes, 15 links: Figure 3.8(a)), Atlanta (15 nodes, 22 links: Figure 3.8(b)), Nobel-Germany (17 nodes, 26 links: Figure 3.8(c)). A description of these networks is given in Table 3.1. There are 5 columns: the network instances, number of nodes, number of links, number of constraints and finally the number of variables in the model. We notice that the number of constraints and variables becomes very large, even for medium size meshed networks.

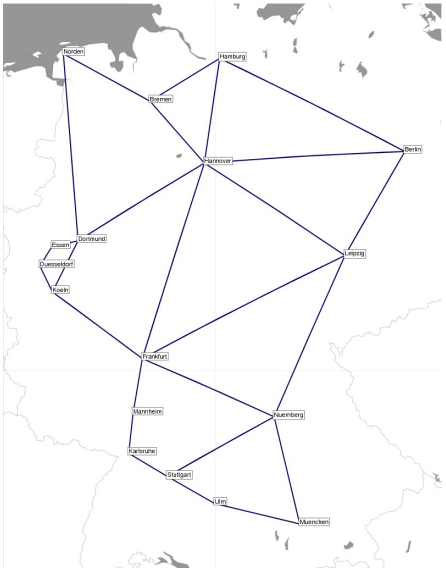
Table 3.2 has 4 columns: the network instance, the ratio capacity of exact method

without "free-conflict constraints", the ratio capacity of our mathematical model and the ratio capacity calculated for the strategy of end-to-end rerouting with stub release (WR). The ratio capacity column describes the ratio between the added capacity and the installed routing capacity. The tests were run on the above networks and the routing was considered to be fixed except for WR. Notice that for WR, the calculation is done not for a fixed routing tree. The program computes both routing and rerouting, which partially explains why WR performs so good. We used the shortest paths to fix nominal routing for our method. Before discussing the numerical results let us recall briefly what is meant by the strategy of end-to-end rerouting with stub release. In case of a failure state, the failing nominal flows are rerouted through possibly several paths, while the unaffected nominal flows are maintained. The failing flows release the capacity on the links unaffected by a failure and the released capacity is used for rerouting needs: this is known as stub release. WR is known to be very efficient in terms of cost-effectiveness compared to other rerouting strategies. Comparing WR to our strategy, we notice that ours applies stub release as well. However there are huge differences between the two strategies. The biggest difference is that our strategy reroutes over a single path from the node near the failure to the destination, while WR allows multi-path rerouting from the origin to destination of any affected traffic demand. Another significant difference is in the requirement to merge the (re)routing paths for different demands with the same destination when they share an arc. All this explains the significant difference to be seen in Table 3.2. Nevertheless, our proposed strategy leads to a very simplified rerouting scheme in terms of management cost, which is a primary objective when dealing with virtual networks. We have also compared the result of our mathematical model to a mathematical model without free-conflict constraints in order to evaluate the cost of the free-conflict constraints. We can see from the results that the

ratio capacity of this model is close to our mathematical model. This suggests that the cost of the free-conflict constraints is not expensive to our network dimensioning.



(a) Polska (b) Atlanta



(c) Nobel-germany

Figure 3.8: Network instances.

Network	Nodes	Links	Constraints #	Variables #
Test	7	9	1951	2419
Polska	12	18	10976	16567
Atlanta	15	22	20371	30781
Nobel-Germany	17	26	57587	49828

Table 3.1: Network instances

Network Instance	Exact-method without "free-conflict constraints"	Exact method	WR
Test	0.83	0.96	0.56
Polska	0.87	0.9	0.49
Atlanta	1.06	1.08	0.86
Nobel-Germany	1.08	1.16	0.73

Table 3.2: Network Cost

Chapter 4

Approximate methods

In this chapter, we present two heuristics that permit to calculate a rerouting scheme without conflict while optimizing the dimensioning of the network. Before that, we also present two algorithms which are in the origin of our heuristics.

4.1 Rerouting schemes

In this section, we present two algorithms that calculate a free-conflict rerouting scheme for a single link failure problem. Their validity is proved theoretically; the first one uses Theorem 1 presented in the previous chapter and the presentation of the second one and its proof follow in section 4.1.2 below.

4.1.1 Algorithm 1

As said above, this algorithm formalizes the method used to build rerouting paths as shown in Theorem 1 presented in the previous chapter. Let us recall first this result.

Theorem 1 *For single link failure situations there exists for any destination l a free-conflict rerouting scheme.*

Let recall briefly the hypotheses and the principle of the method. The main hypotheses are the following: the network is bidirected, symmetric and two-link connected; the routing tree (n nodes) is given for any destination l. We assume that all nodes are involved in the routing tree to destination l. Then, the free-conflict routing scheme is built by adding sequentially the rerouting paths. Each rerouting path is composed of three parts: it follows the inverse path from the upstream end node of the failed arc, borrows a bridge, and follows the nominal routing path to the destination. Embedding all these paths in the routing scheme yields free-conflict routing as shown in the previous chapter. This idea is extended to single node failures presented in chapter 5. Finally, we have shown that extending the existence of a such free-conflict scheme for a set of predefined failure situations is not always possible and a counterexample is given. This algorithm is depicted below:

Algorithm 1:

Input: a directed, symmetric two-link connected network; a routing tree A, $|A| = n$, destination l which is the sink of the reverse tree A;

Output: a free-conflict routing scheme;

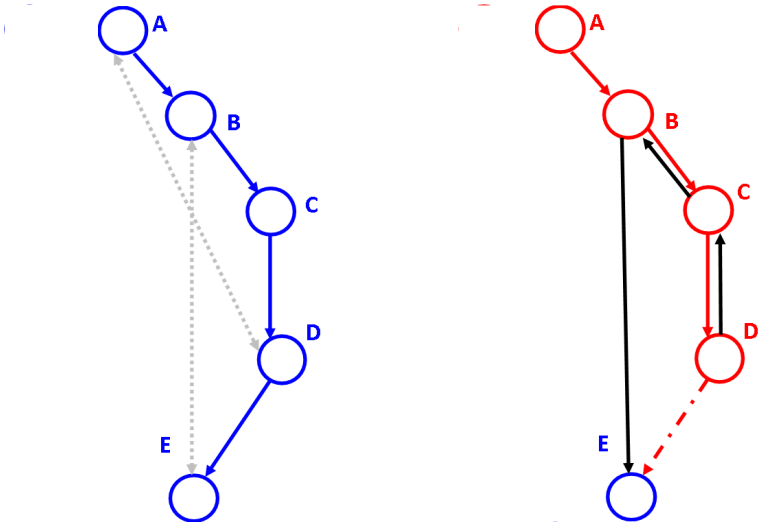
Number the arcs of the routing tree such that their number decreases in value as we approach the sink l: $r \in \{1...n-1\}$. Failures are considered in this order;

for *($r = 1$* **to** $n-1$*)* **do**

- Let $(p_r,\ q_r)$ be an arc of the tree and A_r the subtree of sink p_r;
- Compute a bridge $(i,\ j)$ of A_r where A_r is the subtree of sink p_r. This bridge connects A_r to $A - A_r$. This bridge is chosen so that there is no conflict with $\mu_1, \mu_2, \ldots, \mu_{r-1}$;
- Compute the path μ_k^1 from p_r to i, following the inverse path of the nominal routing tree;
- Set the rerouting path of $(p_r,\ q_r)$ as : $\mu_r = \mu_r^1 + (i,\ j) + \mu_r^3$ where μ_r^3 is the path of the routing tree going from j to l;

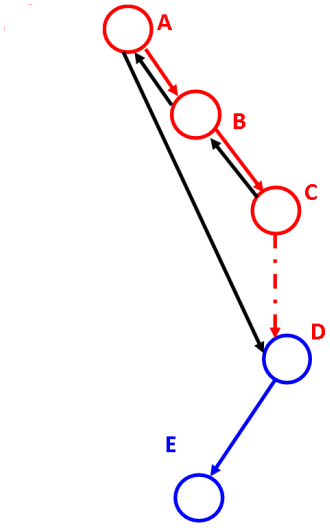
We have to choose the bridge appropriately because if the bridge is chosen according a fixed criteria, it can give us problem.

We prove in this example (Figure 4.1) that if the rerouting paths take only the bridge with the largest index node to reroute traffic, it can cause a problem of conflict. Indeed, we have a routing tree $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E$ to destination E (Figure 4.1(a)). The node is indexed so that the number decreases when the node approaches the node E. For failure link $(D,\ E)$ (Figure 4.1(b)), the largest index is B therefore the rerouting path for this failure is $(D,\ C,\ B,\ E)$. For failure link $(C,\ D)$, A has the largest index therefore the rerouting path is (C, B, A, D, E). We see that these two rerouting paths has a conflict at node B. As we describe in our theorem presented in the previous chapter, the bridge is chosen appropriately with the bridge chosen before, which means in order to avoid the conflict between rerouting paths. When there is no possible conflict between the rerouting path and other rerouting paths calculated previously, the bridge can be chosen randomly. In this example (Figure 4.1), we have



(a) Nominal routing tree

(b) Failure link DE



(c) Failure link CD

Figure 4.1: Selection of bridges with biggest index

to choose the bridge for failure of link $(C,\ D)$ to avoid the conflict with the rerouting path for failure of link $(D,\ E)$. The bridge should be $(B,\ E)$ and the rerouting path is $(C,\ B,\ E)$ in this case.

4.1.2 Algorithm 2

The main drawback of the above algorithm is its lack of flexibility. Given the rerouting path, the only degree of freedom that it can offer is the choice of the bridge. Furthermore, the free-conflict constraint doesn't permit any alternative for the third part of the path. Hence, the only possible variation is in its first part. We show that under some assumption it is always possible to build the first part of the path in a more general way (not only following the inverse path of the nominal routing path). Nevertheless, the free-conflict constraint reduces at a certain point this choice as shown in the algorithm 2 and the proof of validity reported in Theorem 2. In algorithm 2, we initialise a rerouting path which traverses only the nodes in the red part of the routing tree and does not use any arc in the red part of the routing tree. Then, this path follows the nominal routing tree in the blue part. Next, we look if this rerouting path has any conflict with other rerouting ones. In case of conflict, we modify this rerouting path by using the conflicted one. At the end of this algorithm, we remove some directed cycle in the rerouting path to ensure the elementarity of this path.

Theorem

We have stated and demonstrated a theorem to validate this algorithm. We shall make the following assumptions: the graph is assumed to be oriented symmetric; there are at least two disjoint-arc paths between any two nodes of the graph; only one link failure can occur at a time.

Theorem 2. *The rerouting paths computed sequentially by algorithm 2 are elementary and without conflict.*

Proof. Let A be the routing tree to the destination l. l is therefore the sink of A. Let

Algorithm 2:

Input: a bidirected, symmetric two-link connected network; a routing tree A, $|A| = n$, destination l which is the sink of the reverse tree A;

Output: a free-conflict routing scheme;

Number the arcs of the routing tree such that their number decreases in value as we approach the sink l: $r \in \{1...n-1\}$. Failures are considered in this order;

for *($r = 1$* **to** $n-1$*)* **do**

- Let $(p_r,\ q_r)$ an arc of the tree and A_r the subtree of sink p_r;
- Compute a bridge $(i,\ j)$ of A_r where A_r the subtree of sink p_r;
- Compute a path μ_k^1 from p_r to i, traversing only nodes of A_r and not borrowing arcs of A_r;
- Set the rerouting path of $(p_r,\ q_r)$ as : $\mu_r = \mu_r^1 + (i,\ j) + \mu_r^3$ where μ_r^3 is the path of the routing tree going from j to l;
- *Conflict avoidance:*
- **if** *μ_r is in conflict with one of $\mu_1, \mu_2, \ldots, \mu_{r-1}$* **then**
  - Denote μ_k the first rerouting path in conflict with μ_r;
  - Let (a, b) be the first arc in common;
  - Set μ_r^4 as the sub-path of μ_r from p_r to b and μ_k^5 the sub-path of μ_k from b to l;
  - Set the rerouting path $\mu_r = \mu_r^4 + \mu_k^5$;
- *Ensuring path elementarity:*
- **if** *(μ_r is not elementary)* **then**
  - Remove the cycle included in μ_r;

(p_k, q_k) be an arc of A. At first, we will explain how to build a rerouting path for the failure of arc (p_k, q_k). Next, we will modify this rerouting path appropriately in an iterative scheme. We assume that the link corresponding to this arc fails and we must find a rerouting scheme without conflict. p_k is the sink of a sub-tree A_k whose nodes are colored in red. The other nodes in the tree are colored in blue. Without loss of generality we assume that all nodes belong to the tree A and this is true for any destination l. We will also show in this proof that the constructed rerouting path is elementary.

We know that there are at least two disjoint paths in the initial graph going from p_k

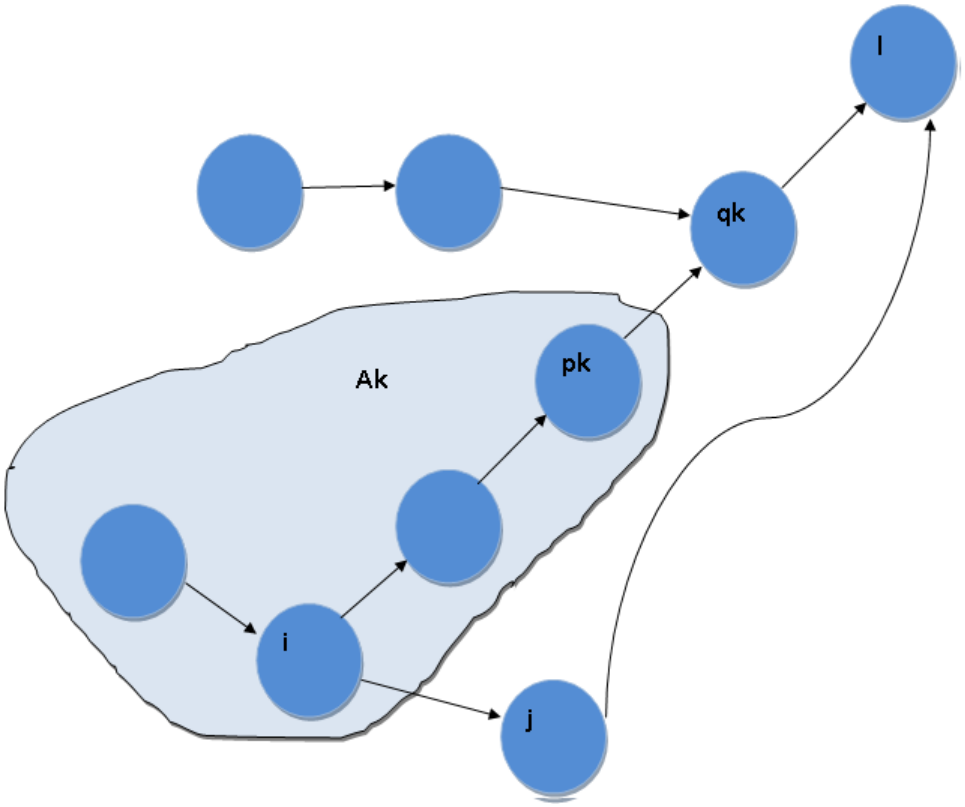


Figure 4.2: Path initialization

to l. Given the assumption of two-link connectivity, the upper part that includes p_k, and does not include the vertex l must contain at least two outgoing arcs. Therefore there are at least two arcs going out of A_k (Figure 4.2). Since one of these arcs is (p_k, q_k), there exists a path μ_k from p_k that visits the vertices of A_k, and connects a vertex of A_k, which is red, to a blue vertex. So there is at least one arc (i, j) of μ connecting the Red vertices to the Blue vertices (Figure 4.2). We call this arc a bridge of A_k. We associate with this arc an elementary rerouting path for the failed arc. Let v be a red vertex of A_k which is affected by the failure. Traffic to destination l and coming from v goes first to p_k, then it follows a rerouting path to destination l. This rerouting path is composed by 3 parts: a path μ_k^1 from p_k to vertex i, a path μ_k^2 that contains only

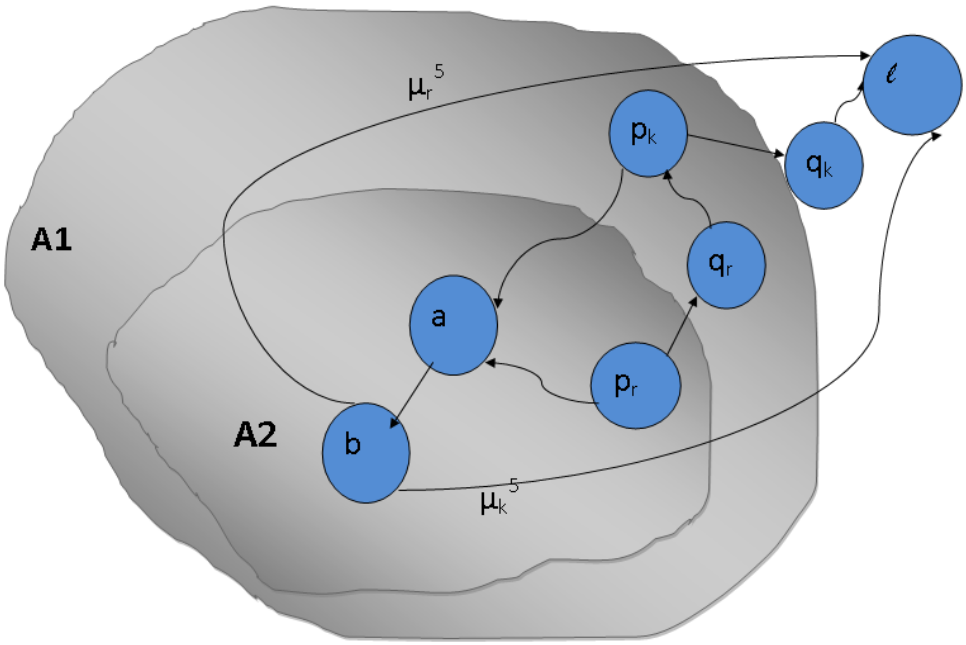


Figure 4.3: Conflict avoidance

the bridge (i, j) and a path μ_k^3 from j to destination l follows the original routing tree. There exists a path from p_k to vertex i that uses no arc of the original routing tree; it suffices to backtrack in the routing tree and join node i. We will describe a general rule of construction of these paths in the following paragraph.

We will explain now how to build the rerouting paths. We number the arcs of the tree so that their numbers decrease in value as we approach the sink l, and consequently choose the rerouting paths in successive order of increasing numbers. For the first failed arc (p_1, q_1), we construct the rerouting path μ_1 like above, so μ_1 is elementary by construction. We consider that we have already computed the rerouting paths for the first $r-1$ arcs in the tree $(r \geq 2)$. By the recurrence hypothesis, there is no conflict for the first $r-1$ reroutings and these paths are elementary. We construct now the r-th rerouting. We verify that the r^{th} rerouting also has no conflict with the first $r-1$ reroutings and that it is elementary. Let (p_r, q_r) be the arc under consideration. p_r is

the root of tree A_r. Now we initialize μ_r that starts with p_r then goes through other vertices of A_r (μ_r^1) and connects a vertex of A_r, which is red, to a blue vertex (μ_r^2: this is a bridge), then starting from this blue vertex follows the original routing tree to destination l (μ_r^3). There are two cases to be considered with this new path. In the first case, μ_r has no conflict with the paths $\mu_1, \mu_2 \ldots \mu_{r-1}$ chosen before. We have nothing to do with μ_r in this case. In the second case, there is a conflict, so there is an existing rerouting path that has an arc in common with μ_r. Let $(a,\ b)$ be the first arc in common between μ_r and the other rerouting paths and $\mu_k, (k < r)$ be the first rerouting path among them having this property.

Notice first that there are only two possibilities for subtrees A_k and A_r; either $A_r \subset A_k$ (the arc (p_k, q_k) is included in the nominal routing path from p_r to l), either A_r and A_k are disjoint. In the last case (a, b) is included in μ_k^3 and μ_r^3 and the rerouting paths are necessarily without conflict. In fact, the two rerouting paths will follow the original routing tree to destination l so there will be no conflict between them and the path is elementary. Consider now the case $A_r \subset A_k$. If (a, b) doesn't belong to μ_k^1, then the same reasoning as above applies and the rerouting path is elementary and without conflict.

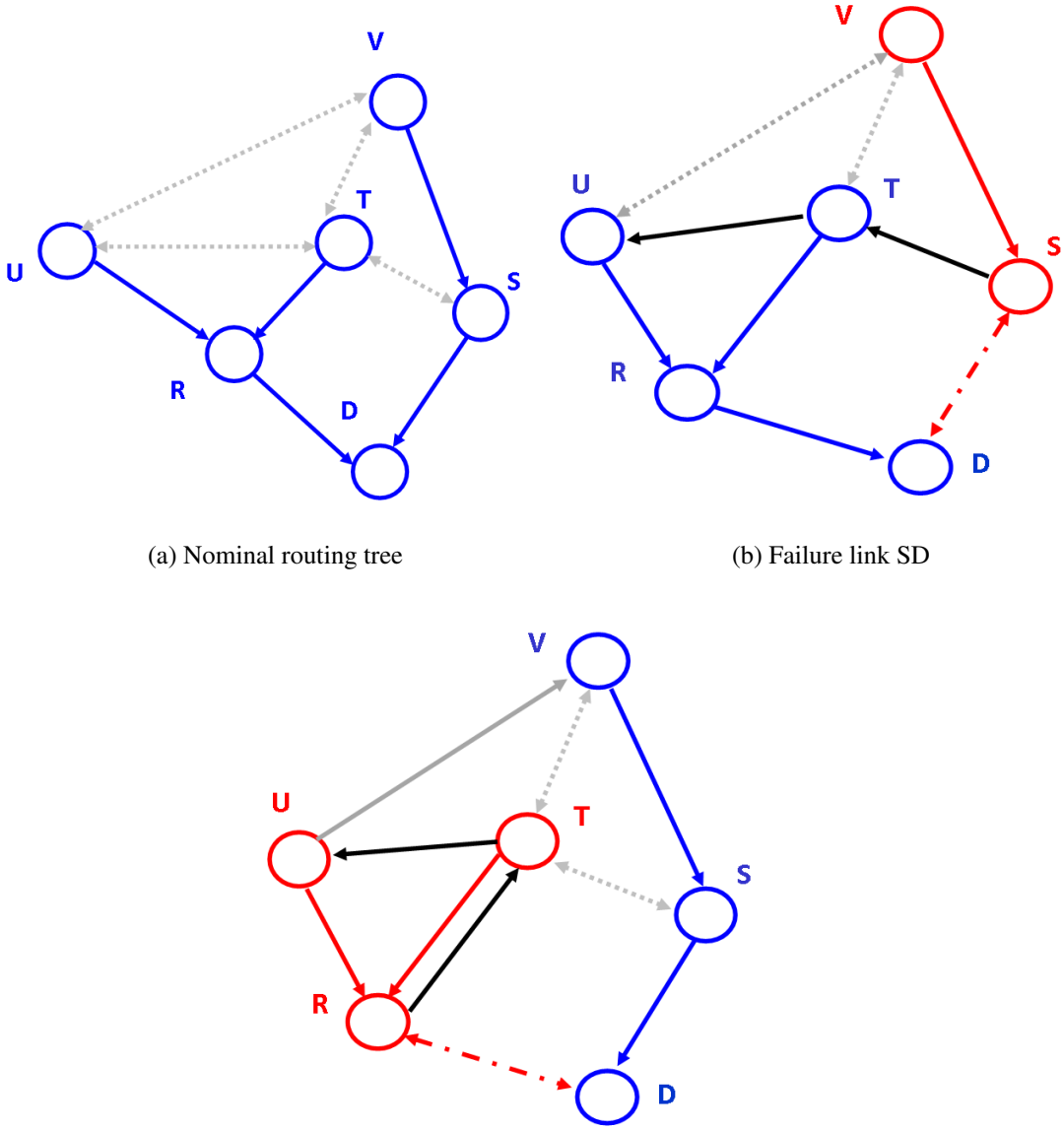
Let consider the case when (a, b) belongs to μ_k^1.This implies that (a, b) doesn't belong to μ_r^3. We define then μ_r^4 the part of μ_r from p_r to arc (a, b) and μ_r^5 the rest of μ_r from b to destination l. In the same way, we define μ_k^4 and μ_k^5. If there is no vertex in common between μ_r^4 and μ_k^5, we reconstruct then the new path μ_r' by concatenating μ_r^4 and μ_k^5. We use this new path to reroute the traffic for the failure $(p_r,\ q_r)$ (Figure 4.3). This path is without conflict from definition of (a, b) as the first arc in common with other rerouting paths. This path does not borrow the failed arc (p_r, q_r) because $A_r \in A_k$ and the path u_k supposed not to borrow any arc of A_k. In case when there

is a vertex in common in μ_r^4 and μ_k^5, let m be the first one for example, we construct a new path μ_r' that uses μ_r^4 from p_r to m and uses μ_k^5 starting from the vertex m to destination l. Clearly, there is no vertex in common between the part of μ_r^4 and μ_k^5 used by the new path μ_r'. So, μ_r' is elementary. We notice that this rerouting path uses the bridge of rerouting path μ_k. We can show also that this contraction will yield no conflict. In fact, we only need to consider if there is a conflict for the arc of this new path μ_r which has m as the end vertex, we suppose this arc is (n, m). Let suppose, by absurdity, that μ_r has conflict with another rerouting path at this arc (n, m), which means (n, m) is the arc in common between μ_r and another rerouting path. This is in contradiction with the hypothesis that (a, b) is the first arc in common with other rerouting paths. Then, we can conclude that this contraction of μ_r is without conflict. We have therefore demonstrated by recurrence the absence of conflict and the property of elementary of the rerouting paths. Finally, the path does not borrow the failed arc $(q_r,\ p_r)$ because it is elementary.

4.1.3 Remark

If someone wants to give more freedom degree by modifying the rerouting path in the blue part of the rerouting tree, we can show that it is impossible. Indeed, if the rerouting path does not use the arcs of the nominal routing tree in the blue part of the routing tree, it could give us an unsolvable conflict in the rerouting scheme. A solvable conflict is a conflict that we can avoid by fusioning the rerouting paths as we did in the above algorithm. So, an unsolvable conflict is a conflict that cannot be avoided by fusioning the rerouting paths. We will present in the example below (Figure 4.4) this type of conflict.

We have in this example a rerouting tree to destination D (Figure 4.4(a)). In this



(a) Nominal routing tree

(b) Failure link SD

(c) Failure link RD

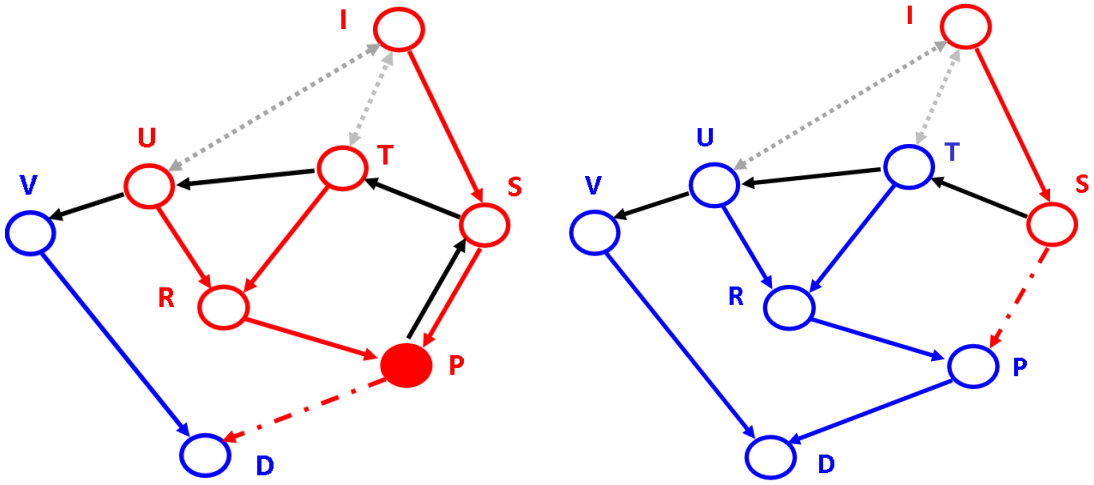
Figure 4.4: Unsolvable Conflict

figure, the arcs in the nominal routing tree are the straight arcs while the dotted arcs are the arcs in the graph that could be used by the rerouting path. When link RD fails (Figure 4.4(b)), the path (R, T, U, V, S, D) is chosen as the rerouting path. When link

SD fails (Figure 4.4(c)), the path (S, T, U, R, D) is chosen as the rerouting path for this failure. We can see that this path uses the arc (T, U) which is not an arc of the nominal routing in the blue part of the routing tree. When we compare these two rerouting paths, we can see that they have the arc (U, T) in common but they separate after this arc. So there is a conflict between these two paths. Now, we try to solve the conflict like in the above algorithm by fusioning two rerouting paths. If the failure of link (R, D) is treated before the failure of link (S, D), we keep the rerouting path of link (R, D) and the rerouting path of link (S, D) becomes (S, T, U, V, S, D). This is impossible because the link (S, D) fails. So, it will make a loop (S, T, U, V, S) and the packets can not be rerouted to destination. This conflict is unsolvable. For another case, when the failure of link (S, D) is treated before the failure of link (R, D); we can see the same result. The rerouting path of link (R, D) becomes (R, T, U, R, D). Because link (S, D) fails, it will make a loop (R, T, U, R) and the traffic cannot be routed to destination.

In the above example (Figure 4.4), we see that not using the arcs of nominal routing tree in the blue part of the routing tree could give us an unsolvable conflict. However, it is true only for the step of initialization of the rerouting path. We can see that after the step conflict suppression, the rerouting path could use the arcs which are not the arcs of the nominal routing tree in the blue part of the routing tree and it will not give us any problem. We can see it in the example below (Figure 4.5).

We have a rerouting tree to the destination D. In this example, because of the numeration of links, the failure of link (P, D) is treated before the failure of link (S, P). When link (P, D) fails (Figure 4.5(a)), the path (P, S, T, U, V, D) is chosen as the rerouting path. When link (S, P) fails (Figure 4.5(b)), after the step conflict suppression, the rerouting path of this failure fusions with the rerouting path of link (P, D) and it becomes (S, T, U, V, D). We can see that this rerouting path uses the arc (T,U) and (U,V)



(a) Rerouting path for failure link PD

(b) Rerouting path for failure link SP

Figure 4.5: Rerouting path after modification

which are not arcs of the nominal routing tree in the blue part of the routing tree. And this rerouting path will not give us problem.

In brief, in the step of initialization of the rerouting path, the rerouting path can go with more liberty in the red part of the rerouting tree but it must use only the arc of the nominal routing tree in the blue part of the routing tree.

4.2 Heuristics

Although an exact compact mathematical model is proposed in the previous chapter, it produces a large-scale ILP not scalable and often intractable even for moderate size networks. This justifies the need for heuristic solution methods. Before detailing the heuristics, let recall the main assumptions. The graph is assumed to be directed, symmetric, and two arc-connected. There is only one link failure at a time and the routing

tree is given. The proposed approaches, like the mathematical model given in previous chapter, are intended to solve the corresponding dimensioning problem, that is minimizing the amount of additional capacities put in the network while ensuring full restoration through free-conflict routing for all single link failures. We present below two polynomial heuristic approaches which make use of the two algorithms given in the previous section.

4.2.1 Heuristic 1

The main idea behind Heuristic 1 is using the rerouting path constructed by Algorithm 1. The heuristic goes through two loops. The external loop iterates over destinations and the internal loop over link failures. For a given destination l and failure p, the heuristic constructs the rerouting path as proposed in Algorithm 1. The choice of the rerouting path is exclusively determined by the choice of the bridge. In practice, several bridges may exist. Then, the choice is done with respect to the cost in terms of the amount of added capacities necessary to realize the traffic rerouting through the corresponding path. Hence, we need to take into account added capacities in each arc and for each failure situation. The final added capacity is given by the maximum of added capacities over the set of failures.

We will describe in detail here the data structures and functions used in our heuristic. Let $G = (\mathcal{V}, \mathcal{E}, C)$ be the symmetric graph, drawn from the network. We want to determine the capacities added to the arcs to ensure the rerouting in case of single failure. We suppose that the nominal routing is given and the graph has the minimum necessary capacity to route the traffic in the absence of failure. For each destination node arcs are numbered such that arcs in the same path to the destination have decreasing numbers as approaching to destination.

We use the matrix $M[p][a]$ to keep the additional capacity needed on the arc a if the failure of arc p occurs. The matrix M will be determined and updated during the execution of the heuristic. The array of constants $T[d][p]$ contains the amount of traffic to reroute to destination d for the failure of arc p. We use the table R, where $R(a)$ indicates the additional capacity on the arc a. $R(a)$ must be sufficient to ensure rerouting associated with each failure p, that is $R(a) = \max_p M[p][a]$. We define the function $\delta(a, b)$ which is equal to $b - a$ if $b \geq a$ and to 0 otherwise. We define $A[d]$ the set of rerouting paths chosen during algorithm for destination d. $A[d][p]$ indicates the rerouting path to the destination d in the algorithm for failure p. A situation can be identified by the pair $(p,\ d)$ where p is the failure of the arc p and the destination d.

When an arc fails, the capacity of arcs between this failure and the destination is assumed as released which is taken in account in the computation. Indeed, we take this in consideration in the function $CalculRecuperation(p)$.

Heuristic 1 follows the same procedure as algorithm 1, but it takes into account also the capacity element. In order to optimize the additional capacity, when we compute the bridge $(i,\ j)$, we search a path that minimizes the cost S below: $S = \sum_{a \in \mu} \delta(R[a],\ M[p][a] + T[d][p] + M[p'][a])$, with p ' the symmetric arc of p. In fact, we have that $M[p][a] + M[p'][a]$ is equal to the additional capacity needed on the arc a if the link corresponding to arc p and p' fails. When we add $T[d][p]$ to this sum, we will have the new additional capacity needed on the arc a if the rerouting path uses this arc. Then, we compute the function δ to have the difference between this new value with $R[a]$. So, this term will calculate the capacity added to arc a when the rerouting path uses this arc. This value indicates then the sum of capacities added to each arc of the rerouting path. Indeed, in order to minimize the additional capacity of the network, we try to minimize the additional capacity each time we create a rerouting path.

We minimize the cost S and compute the bridge in the function $CalculChemin(p, d)$ which calculates the path rerouting without conflict for a given failure.

Based on the above functions, our heuristic can be formulated as follows:

Algorithm 3: Heuristic 1

Input: a directed, symmetric two-link connected network; a routing tree A, a destination list and a set of failures;
Output: a fully free-conflict routing scheme;

```
1 Initialize M[p][a] to 0 for all p and a.
2 Initialize R[a] to 0 for all a in E.
3 Initialize A[d][p] to empty.
4 forall the failure p in the failure list P do
5     CalculRecuperation(p)
6 forall the destination d in destination list D do
7     forall the failure p in the failure list P do
8         CalculChemin(p, d)
```

The above algorithm is time-polynomial. This is clear as the $CalculRecuperation(p)$ and $CalculChemin(p, d)$ are of polynomial time.

4.2.2 Heuristic 2

Heuristic 2 uses the idea behind the Algorithm 2, allowing thus for more flexibility. The main difference with Heuristic 1 lies in the way the first part of the rerouting path is constructed. We use here the shortest path (Dijkstra Algorithm) with respect to an appropriate metric. This metric characterizes the added capacity for each arc. This metric also assures that the rerouting path respects the imposed constraints. For example, to make sure that it will not use any nominal arc in the red part of the routing tree, we initialize their metrics as infinite.

We present below the main lines of Heuristic 2. The above algorithm is clearly

Algorithm 4: Heuristic 2

Input: a directed, symmetric two-link connected network; a routing tree A, a destination list and a set of failures;

Output: a fully free-conflict routing scheme;

```
1  forall the destination d in destination list D do
2      forall the failure p in the failure list P do
3          Initialize metrics to infinite for nominal arcs in the red tree;
4          Initialize metrics to infinite for non nominal arcs in the blue tree;
5          Initialize metrics to infinite for all inverse arcs in alternative bridges (i.e.
           arcs from Blue part to the Red one on the bridge);
6          Initialize metrics for arc a to M[p][a] + M[p'][a] + T[d][p] − R(a) if this
           term is positive and initialize metrics to 0 if it is not;
7          Search the shortest path using Dijkstra for this failure p;
8          if (there is a conflict between this path and the paths calculated
           previously) then
9              Fusion the two paths in conflict as described in the Algorithm 2 to
               have an elementary path without conflict;
10         Using this path as the rerouting path for failure p and destination d;
11         Update R(a), M[p][a];
```

of polynomial complexity since each iteration uses Dijkstra algorithm to compute the rerouting path and this is done for $|D| * |P|$ iterations. In addition, it can be easily established that the fusion procedure is also polynomial while the metric initialization for each arc is costless.

4.2.3 Amelioration of heuristic 2

The heuristic 2 presented above is still simple. We want to enhance this heuristic to achieve better results. We have proposed two alternatives to heuristic 2 that we will present in the following sections.

Decreasing order of capacities

The first alternative is based on the decreasing order of lost traffic. Indeed, as we can see from heuristic 2, the additional capacity for a failure depends on the additional capacity that we have added for the failures treated before it. So, what happens if we reroute the failure with larger amount of lost traffic before the failure with small amount of lost traffic? Maybe we do not need to add much capacity for the second one. This intuition makes us to come to alternative 1. At first, we need to sequence in decreasing order the rerouted traffic. We consider the pair (destination, failure) as a situation of failure. Each failure situation has a traffic that needs to be rerouted. Then, we organize in decreasing order the rerouted traffic of these failure situations. We modify the heuristic 2 using this order to have alternative 1.

Algorithm 5: Alternative 1

Input: a directed, symmetric two-link connected network; a routing tree A, a destination list and a set of failures;
Output: a fully free-conflict routing scheme;

1 Create a list of situation (destination, failure) in decreasing order of rerouted capacity. **forall the** *situation (d, p) in this list of situation* **do**
2 Initialize metrics to infinite for nominal arcs in the red tree;
3 Initialize metrics to infinite for non nominal arcs in the blue tree;
4 Initialize metrics to infinite for all inverse arcs in alternative bridges (i.e. arcs from Blue part to the Red one on the bridge);
5 Initialize metrics for arc a to $M[p][a] + M[p'][a] + T[d][p] - R(a)$ if this term is positive and initialize metrics to 0 if it is not;
6 Search the shortest path using Dijkstra for this failure p;
7 **if** *(there is a conflict between this path and the paths calculated previously)* **then**
8 Fusion the two paths in conflict as described in the Algorithm 2 to have an elementary path without conflict;
9 Using this path as the rerouting path for failure p and destination d;
10 Update $R(a), M[p][a]$;

Using multi-start with random choice

We want to do a multi-start to enhance heuristic 2. A multi-start means to launch the heuristic many times and choose the best result. In order to use this method, we need to add random choice to heuristic 2. We modify heuristic 2 by choosing the destination randomly to have alternative 2. We present here the main lines of this alternative:

Algorithm 6: Alternative 2

Input: a directed, symmetric two-link connected network; a routing tree A, a destination list and a set of failures;
Output: a fully free-conflict routing scheme;

```
Create a list of destinations.
Shuffle this list of destinations to have a random order.
forall the destination d in this shuffled list do
    forall the failure p in the failure list P do
        Initialize metrics to infinite for nominal arcs in the red tree;
        Initialize metrics to infinite for non nominal arcs in the blue tree;
        Initialize metrics to infinite for all inverse arcs in alternative bridges (i.e.
        arcs from Blue part to the Red one on the bridge);
        Initialize metrics for arc a to M[p][a] + M[p'][a] + T[d][p] − R(a) if this
        term is positive and initialize metrics to 0 if it is not;
        Search the shortest path using Dijkstra for this failure p;
        if (there is a conflict between this path and the paths calculated
        previously) then
            Fusion the two paths in conflict as described in the Algorithm 2 to
            have an elementary path without conflict;
        Using this path as the rerouting path for failure p and destination d;
        Update R(a), M[p][a];
```

4.2.4 Numerical results

In order to evaluate the effectiveness of the above heuristics, we tested them on 8 network instances presented in Table 4.1. Table 4.2 gives the numerical results obtained with heuristics, the exact model, the mathematical model without "free-conflict" con-

Network	Nodes	Links	Demands
Test	7	9	42
Polska	12	18	66
Atlanta	15	22	210
Nobel-Germany	17	26	121
France	25	45	300
India35	35	80	595
Pioro40	40	89	780
Germany50	50	88	662

Table 4.1: Network instances

Network Instance	Exact method	Heuristic 1	Heuristic 2	Xi and Chao method
Test	62	66	68	69
Polska	19110	21449	21949	25093
Atlanta	308171	333480	343969	330745
Nobel-Germany	1862	1980	1940	2744
France	OM	261529	260451	416670
India35	OM	7874	7784	11689
Pioro40	OM	289168	279046	431332
Germany50	OM	7339	7453	9847

Table 4.2: Network cost

Network Instance	Alternative1	Alternative2
Test	66	66
Polska	21583	21136
Atlanta	323820	322384
Nobel-Germany	1936	1932
France	267829	251869
India35	8261	7923
Pioro40	277823	284293
Germany50	7285	7311

Table 4.3: Network cost 2

straints and the method of Xi and Chao [11]. Notation OM (Out of Memory) stands for cases when the program stops without reaching a solution because of memory problems. We can remark from this table that the results from both heuristics are close to those of the MIP model given in chapter 3. Indeed, the gap is between 6% and 12%.

Furthermore, no heuristic is clearly better than the other. We notice that Heuristic 2 performs generally better than Heuristic 1 for large instances (except for the last one). Heuristic 1 finally performs quite well given its simplicity. Borrowing arcs of the inverse path results to be a good strategy as such arcs will be taken by all rerouting paths used for failures in the same routing path. We have talked about the similarity of our method and the method of Xi and Chao [11] in previous chapter, so now we compare our method with this method. We recall that the rerouting path always chooses the first bridge in [11] and when two rerouting paths have the same node in common, they will fusion until they reach the destination. Their method is more restrictive than our methods and it did not take into account capacity optimization. We can see from the numerical results that our methods are better than [11] except for Atlanta network, which is a very simple network.

We notice that the CPU times for both heuristics are very low for the application in hand. The CPU time for Heuristic 2 solving the largest instance (Germany 50) is less than 15 seconds. We believe that the proposed heuristics achieve a good trade-off between the calculation time and the performance in terms of quality of the provided solutions.

We also evaluate the effectiveness of the above alternatives and we obtain then numerical results in table 4.3. When we compare with the results in table 4.2, we can see that the two alternatives achieve better results than the two heuristics. Alternative 2 seems to perform better than alternative 1. The result of these alternatives is close to the mathematical model presented in the above chapter. The gap is between 4% and 10%.

In order to evaluate the two heuristics and two alternatives when the node degree of the network increases, we have used the modified version of the network $Ta2$ taken

Network	Nodes	Links	Demands
$Ta2$	64	107	1869
$Ta2_1$	64	112	1869
$Ta2_2$	64	121	1869
$Ta2_3$	64	141	1869
$Ta2_4$	64	161	1869
$Ta2_5$	64	176	1869
$Ta2_6$	64	269	1869

Table 4.4: Network instances

Network Instance	Initial Capacity	Heuristic 1	Heuristic 2	Alternative 1	Alternative 2
$Ta2$	37740	47191	50189	42612	42887
$Ta2_1$	37664	46131	4623	40195	44779
$Ta2_2$	37087	44889	44239	38099	43486
$Ta2_3$	34513	37118	38807	35671	39145
$Ta2_4$	31808	30951	30761	30001	29329
$Ta2_5$	30842	29269	29464	29772	29317
$Ta2_6$	29675	28094	34787	27962	30758

Table 4.5: Network cost

from the library sndlib as network instance and we have increased its node degree. These networks are described in table 4.4. We apply the two heuristics and two alternatives to these networks. We can see from this result that the two alternatives perform better than the two heuristics.

4.3 Application to network virtualization

As we suppose at the beginning of the thesis, there are two virtual networks in our case. One virtual network, which will be secured with our method, can react automatically and assure no loss of traffic in case of single link failure. Another one is best-effort and there is no guarantee on the quality of service in this network. However, in network virtualization, there could be more than one virtual network (virtual layer) that should

be treated with priority. In that case, we have to modify our method in order to ensure the quality of service of these virtual networks.

The mathematical model could be modified to be adapted with several virtual networks. Indeed, we have to add only a layer index to our variable and modify a little our capacity constraints for this objective. However, as said before, the exact method using MILP formulation is not tractable for this case too.

We will present in this paragraph how we can apply our method to network virtualization. Even if there are several virtual layers in our network, there is no conflict between these layers. Therefore, we can apply our two algorithms to each layer to find out the rerouting paths for each of them separately. Another way of speaking, we can apply directly our two heuristics to each layer to find the rerouting path and the dimensioning for this layer. However, the dimensioning is not optimized in this case and there is a waste of additional capacity. As we can see, several networks can use the same additional capacity if this is not for the same failure. For example, we have two virtual network with priority $v_1,\ v_2$, we add an amount of x capacity to arc a. This amount x will be used by v_1 in case of failure of link p_1 and it will be used by v_2 in case of failure of link p_2. In this example, arc a is used only by v_1 in case of failure of link p_1 and is used only by v_2 in case of failure of link p_2. In our heuristics presented in previous section, for the same failure, we sum the additional capacity on an arc for each destination. Now, for the same failure of physical link, we have to take into account the sum of additional capacity on an arc for each destination and each layer. With this small modification, our method can be applied to several virtual networks in order to find the rerouting paths and to optimize the dimensioning of these networks.

Chapter 5

Enhancing the theoretical study

In this chapter, we extend our study to the node failure problem and multi-link failure problem. Indeed, we have proven that there exists a rerouting scheme without conflict for the node failure problem and we have found a cons-example to show that we cannot construct a rerouting scheme for the multi-link failure problem.

5.1 Node failure

Another problem that we consider is resilience to node failure. This type of failure is rarer than link failure. Nevertheless, we need to check that a valid rerouting scheme can be computed for this case too. We first note that in the case of a node failure any traffic demand having this node as origin or destination cannot be recovered; only the transiting traffic needs to be rerouted. In practice, a node failure situation is like several simultaneous link failures. Figure 5.2 provides a small example of a node failure situation. In this example, all the traffic is assumed to have the same destination D. When the node R fails, all the traffic coming from R_1, R_2 and R_3 needs to be rerouted. In practice this is like the situation where links $(R_1,\ R)$, $(R_2,\ R)$ and $(R_3,\ R)$ fail at the same time. We have to reroute the traffic so that there is no conflict in the network. In the next section we provide a formal proof that under some more restrictive conditions there always exists a rerouting scheme without conflict to recover single node failure situations.

At the beginning of our study, we wanted to see if the solution we used for the link failure problem could be applied to the node failure problem. We found out that it could not be applied. We have an example (Figure 5.1) that shows this fact. In this example (Figure 5.1), when the node S fails, we can consider that the two links $(A,\ S)$ and $(F,\ S)$ fail at the same time. If we compute the rerouting paths for these both link failures as described in the previous chapters, we could find out that there is a problem. Indeed, for the failure of link $(A,\ S)$, the only possible rerouting path is $(A,\ C,\ B,\ E,\ D)$; $(F,\ B,\ E,\ D)$ is the rerouting path for the failure of link $(F,\ S)$. We could see that the path $(A,\ C,\ B,\ E,\ D)$ has a conflict with the nominal routing path at the node B. When B receives the traffic coming from C, it can not determine

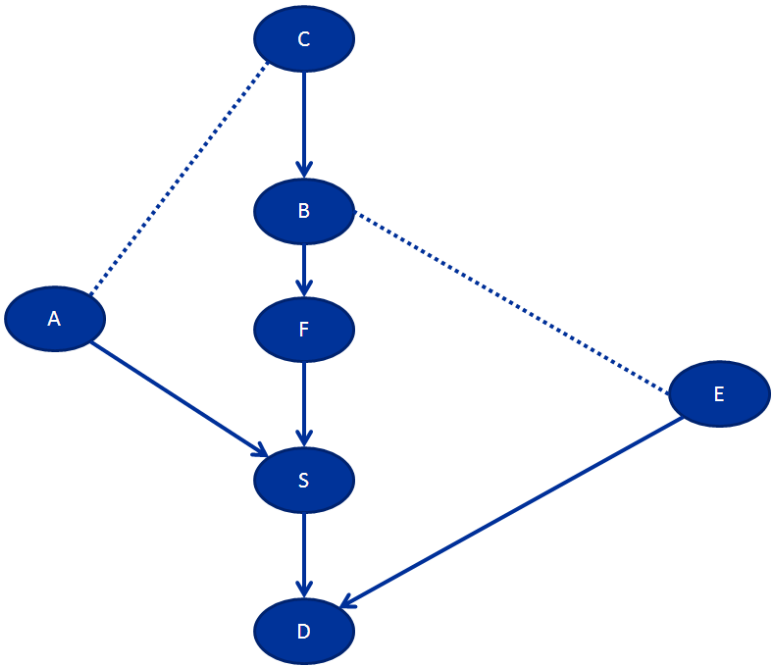


Figure 5.1: Node failure example

if it should transfer the traffic to F or to E. From this, we can see that we can not apply what we have done for the link failure problem to the node failure problem. To solve this issue, we should program the filter of node B so that when it receives the traffic coming from C, it would transfer it to F. As we know, F is the node that detects the failure of link $(F,\ S)$ and all the traffic coming to F will be rerouted following the rerouting path of link $(F,\ S)$ described above. With this, the traffic of the failure of link $(A,\ S)$ is rerouted to the destination D. It is not rerouted directly but intermediately via the rerouting path of link $(F,\ S)$. We can use this idea and extend it to have the solution for the node failure problem.

5.1.1 Theorem of the existence of a restoration scheme in cases of node failure

In this section we study the question of existence of a rerouting scheme without conflict in case of node failure. Let us start by listing the assumptions: the graph is assumed to have bidirected links; there are at least two disjoint-node paths between any two nodes of the graph; only one node failure occurs at a time.

Theorem 3. *For single node failure situations there exists for any destination l a rerouting plan without conflict.*

Proof. Let A be the nominal routing tree to the destination l. l is therefore the sink of A. Let R be a node of A. We shall suppose that this node has failed and that we need to find a rerouting scheme without conflict. R is the sink of a sub-tree R_{sub} whose nodes are colored in red. The other vertices in the tree are colored in blue. Without loss of generality we assume that all vertices are part of the tree A and this is true for any destination l. We denote as R_{set} the set of all red nodes that have a direct connection with R, $R_{set} = \{R_i, 1 \leq i \leq m\}$. These nodes are also the sinks of the nominal routing sub-trees R_{sub_i} with destination l. Under our hypothesis R is in a state of failure, and so we shall consider only the sub-trees R_{sub_i} with $i \in \{1..m\}$. Finding a rerouting scheme for a failed node R is similar to finding the rerouting paths for the failed links $(R_i,\ R)$. The only difference is that these rerouting paths cannot pass through node R.

We know that there are at least two node-disjoint paths in the initial graph for any pair of nodes. There is therefore at least one arc going from $R_{sub} \backslash R$ to the set of blue vertices. Let I denote the set of all red vertices having a direct connection (that is, an arc) with the blue vertices. Within this set I we need to determine the sub-tree R_{sub_i} to which these different vertices belong. All sub-trees that contain at least one node from

I are called R_{sub}^{direct}. For each of these sub-trees we choose a node a_i from I and its corresponding b_i among the blue vertices, and use these nodes to construct a rerouting path for the corresponding failed link (R_i, R). The rerouted traffic to destination l then follows the reverse path of the nominal routing tree from R_i to a_i. It uses the arc (a_i, b_i) and then from b_i all the way to the destination l follows the nominal routing tree. For any sub-tree (i.e. R_{sub_i}) that does not contain any vertices in I, there is necessarily a path μ from vertex R_i to l that does not pass through R. This path μ will first visit only red nodes contained in different sub-trees before it reaches a blue vertex. As there are no vertices of R_{sub_i} in I, two or more sub-trees will necessarily be traversed by this path. The sub-trees having a direct connection with the R_{sub}^{direct} sub-trees are denoted $R_{sub}^{indirect-first}$. For each sub-tree in $R_{sub}^{indirect-first}$, say R_{sub_j}, we choose one sub-tree in R_{sub}^{direct} and fix an arc, let us say (a_j, b_j), connecting them. Similarly, we define the set $R_{sub}^{indirect-second}$ and choose for each sub-tree in this set a corresponding sub-tree in the set $R_{sub}^{indirect-first}$, together with an arc connecting them, and so on. Then, the traffic to destination l for one of these sub-trees, let us say $R_{sub_j} \in R_{sub}^{indirect-first}$, goes from sink R_j through the reverse path to the corresponding vertex $a_j \in R_{sub_j}$, arc (a_j, b_j), continues along the the nominal routing path from b_j to the sink of the corresponding sub-tree, and ends with the rerouting path to l. The latter part of the constructed path is explained above. Each path is thus composed of alternate sub-paths with nominal routing parts from some node down to the sink on the one hand, and rerouting sub-paths from the sink to the next sub-tree or Blue part on the other.

We now show that the rerouting scheme remains without conflict, whichever single node failure is considered. Without loss of generality we consider in the following that only R_{sub}^{direct} and $R_{sub}^{indirect-first}$ are concerned for each node failure situation. We consider successively the failures of $n-1$ nodes in the nominal routing tree to the

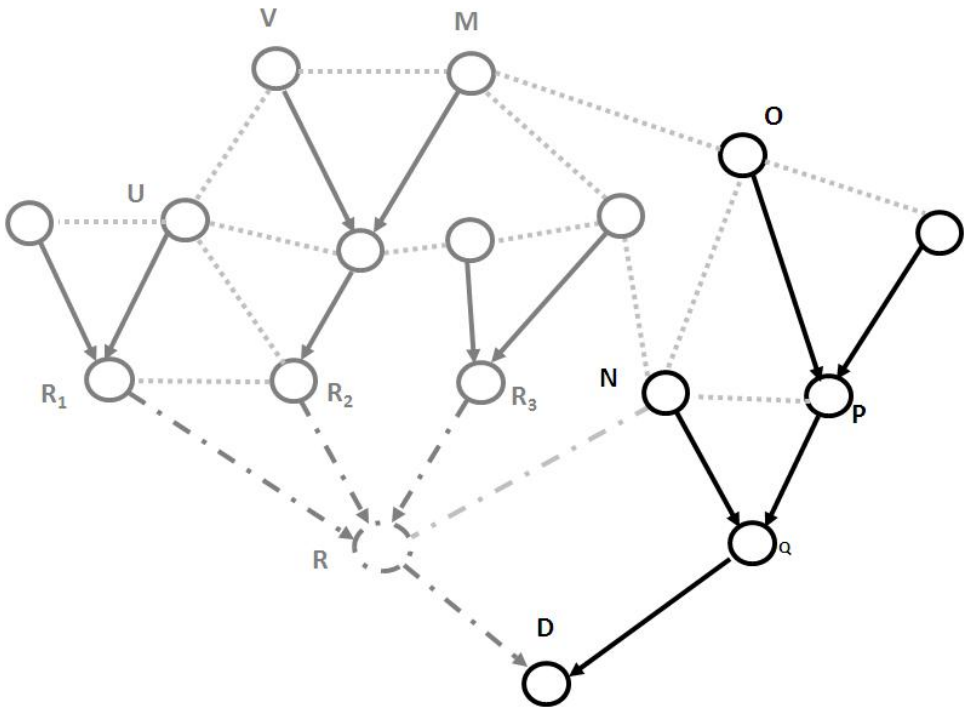


Figure 5.2: Node failure

destination l and choose the different arcs $(a_i,\ b_i)$ for each situation. We number the nodes of the tree so that their numbers decrease in value as we approach the sink l. We then choose the arcs (a_i, b_i) in increasing order of number of the corresponding node failure and we place them in a set named $Connection - arcs$. Let $p_r(= R)$ be the node currently under consideration, and let us assume that we have already examined the first $r - 1$ node failures and $Connection - arcs = \{(a_1, b_1), (a_2, b_2) \ldots (a_k, b_k)\}$. R is the root of sub-tree R_{sub}. As above, we consider sub-trees according to whether they belong to R_{sub}^{direct} or to $R_{sub}^{indirect-first}$. We first consider a sub-tree in R_{sub}^{direct}, say R_{sub_i}. We find all candidate connection arcs for this sub-tree and check whether any of them is already in the set $Connection - arcs$, that is to say whether there exists p_s with $s < r$ and a connecting arc (a_s, b_s) whose extremity a_s is in the tree R_{sub_i}. Two cases can be distinguished here. If answer is yes, then the other extremity b_s is

outside the tree R_{sub_i}, and *a fortiori* outside the sub-tree R_{sub} which is included in the tree with sink p_r (this is similar to A_s in Figure 3.4 that was used to illustrate the previous demonstration). In this case we choose arc (a_s, b_s) as the connecting arc for the sub-tree R_{sub_i}. If answer is no, we choose one of the connection arcs at random. We proceed in a similar way for the sub-trees in $R_{sub}^{indirect-first}$.

We need to show that the rerouting has no conflict, and this can be demonstrated by recurrence on the number of failed nodes. Let us suppose that we have already rerouted the traffic for $r-1$ nodes in the tree. By the recurrence hypothesis, there is no conflict for the first $r-1$ reroutings. We verify that the r-th rerouting also has no conflict with the first $r-1$ ones. Regarding the rerouting of the outside part of the tree r_{sub}, that is to say the part in common with the nominal routing, there is no conflict by construction. Even if it uses the same arc, in this part it will follow the same rerouting path all the way to the destination l, and so it is without conflict. We also verify that there is no conflict for the part where it goes in the opposite direction of the tree, which means verifying that there is no conflict in the two cases above. In the first case, where an arc $(a_s, b_s) \in Connection - arcs$ has been chosen from the tree R_{sub}, there is clearly no conflict in that part of the tree, because R_{sub} will use the same arc (a_s, b_s) as a bridge between its red and Blue part. In the second case, the part that ascends the tree can have nothing in common with the other rerouting arcs, since this would imply the existence of at least one (a_s, b_s). There is therefore no conflict in this case either. We conclude that the property remains true to the order r. This demonstrates by recurrence the absence of conflict. □

It will be remarked that switching to an alternative path is local to the node that detects the failure. It is therefore not possible at the level of this node to distinguish between a link failure and a node failure. Note that the method applied to node failures

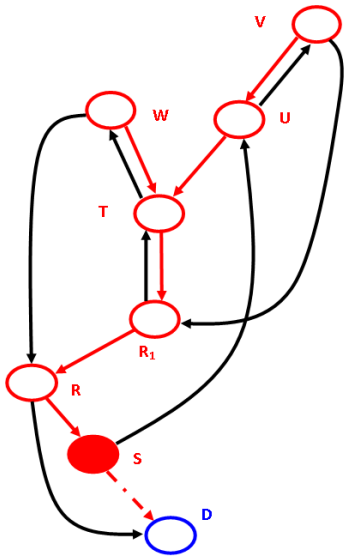
protects the network against link failures as well as node failures.

5.1.2 Remark about single node failure problem

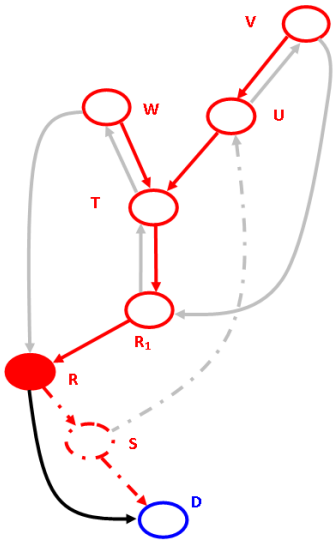
As we present at the beginning of the thesis and the previous chapters, only the nodes at extremity of the failures can detect the failure and reroute the traffic to another path. In fact, only one node in case of single link failure situation and several nodes in case of single node failure situation could detect the failure, while other nodes transfer the traffic normally as described in their routing table. Even so, these nodes cannot identify if it is the link failure or node failure problem. This is the reason why we have to decide at the beginning if our network secures the link failure problem only or it secures also the node failure problem. As we see from the theorem above, this solution secures both link failure and node failure problem, while the solution presented in chapter 4 secures only link failure problem.

We present in the previous chapter a theorem which gives more liberty degree for the rerouting path in the Red part of the network in case of link failure problem. We wanted to know if this idea could be applied for the node failure problem. However, we found a con-example to show that this is impossible.

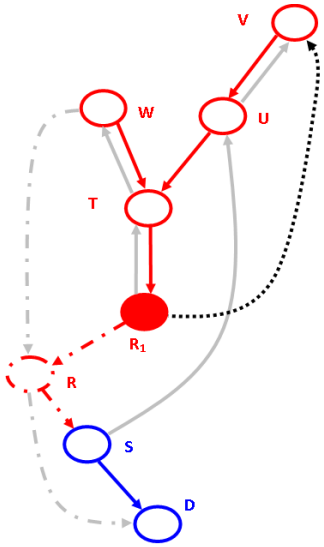
In this example (Figure 5.3), we have a tree to destination D. The arcs in the tree can be identified by the straight downward arrow in Figure 5.3(a). Because the solution of node failure problem have to solves also the link failure problem, we consider here the failure of link $(S,\ D)$ (Figure 5.3(a)). In fact, we can see that the failure of link $(S,\ D)$ is equivalent to the failure of a fictive node between S and D, so we have to take this failure into consideration. When link $(S,\ D)$ fails, the rerouting path could be $(S,\ U,\ V,\ R_1,\ T,\ W,\ R,\ D)$. This rerouting path has no conflict with the nominal routing tree and it has more liberty degree in the red part of the routing tree. Next, we



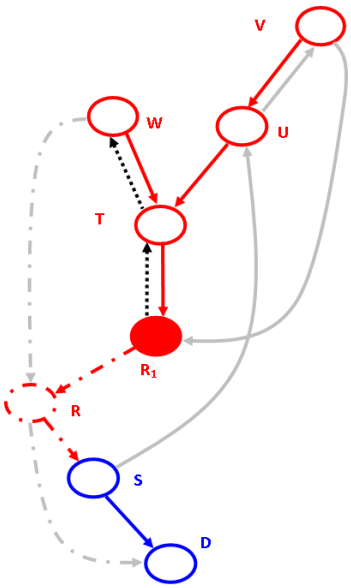
(a) Failure link SD



(b) Failure node S



(c) Failure node R case 1



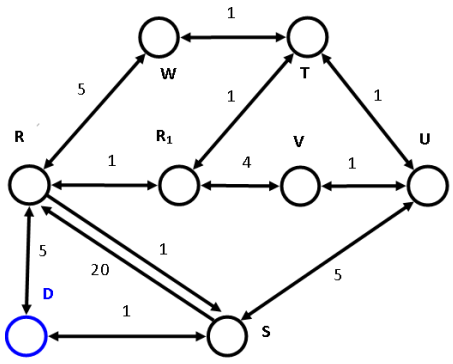
(d) Failure node R case 2

Figure 5.3: Unsolvable Conflict in case of node failure

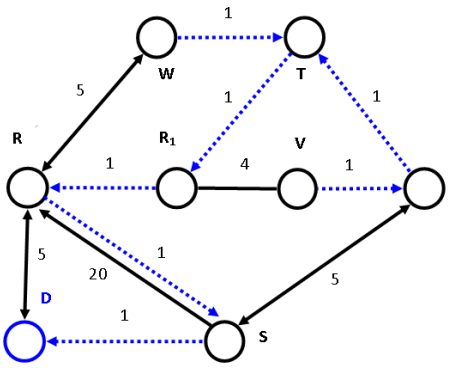
consider the failure of node S (Figure 5.3(b)). In this case the rerouting path is $(R,\ D)$. We consider now the failure of node R. We can see there is two solutions possible in this case. In the first case (Figure 5.3(c)), the rerouting path uses arc $(R_1,\ V)$, then it has to use the arc of the nominal routing tree until the node R_1. In this case, it causes then the looping problem and the traffic could not be rerouted to destination D. In the second case (Figure 5.3(d)), the rerouting path uses arc $(R_1,\ T)$. As it uses the same arc as the rerouting path in case of failure of link $(S,\ D)$, it has to follow this rerouting path until it reaches destination D. However, because node R fails, this rerouting path cannot reach destination D. This conflict is unsolvable. This example seems strange when we look at it in the tree form (Figure 5.3). However, it is not an exotic example at all when we transform into graph (Figure 5.4). We have an initial graph with metrics in the arcs (Figure 5.4(a)). When we calculate the shortest paths tree with Dijiktra, we have figure 5.4(b). In this figure, the shortest path tree is noted with the dotted arrow. When link $(S,\ D)$ fails (Figure 5.4(c)), the rerouting path is $(S,\ U,\ V,\ R_1,\ T,\ W,\ R,\ D)$ using shortest path. We can see that the nominal routing tree and the rerouting path are identical with example above (Figure 5.3). In brief, we cannot apply the idea about rerouting path with more liberty degree to node failure problem.

5.1.3 Multiple link failure situations

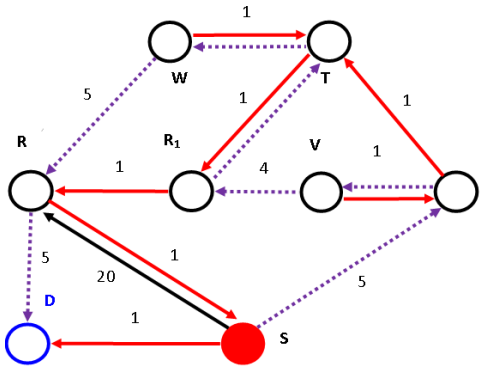
Because a situation of node failure is equivalent to several simultaneous link failures, it might be assumed that a free-conflict rerouting scheme for any multiple link failure situation can be constructed, given that the network remains connected. We have, however, found a counterexample showing that it would be wrong to make such an assumption (Figure 5.5).



(a) Initial graph



(b) Routing tree



(c) Failure link SD

Figure 5.4: Unsolvable Conflict in case of node failure in graph form

In this counterexample, we are given a routing tree to destination D and we consider a set composed of two failure situations for which the resident networks remain connected. We show that it is not possible to build a free-conflict rerouting scheme for destination D resilient to each of these failures.

The first failure situation is composed of links $(A,\ D)$ and $(C,\ D)$. When $(A,\ D)$

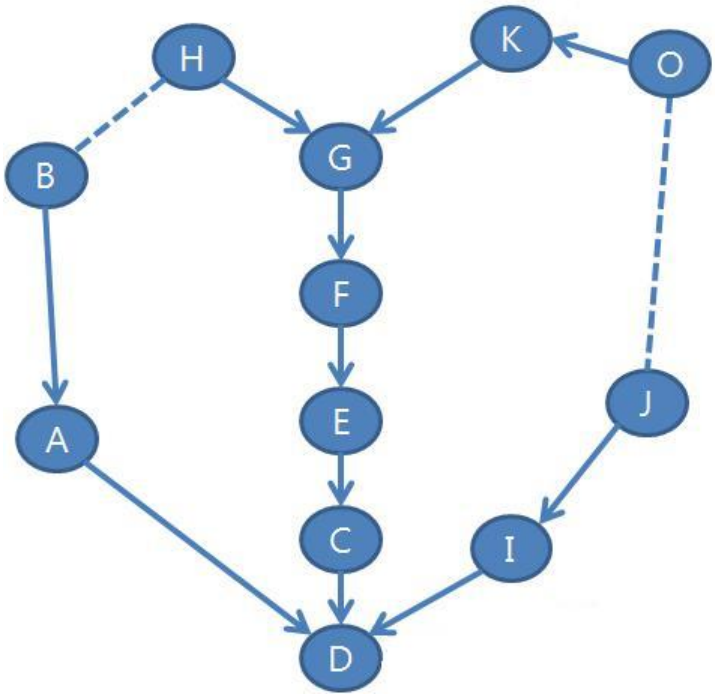


Figure 5.5: Counterexample from the generalized problem.

fails, the only available rerouting path is $(A,\ B,\ H,\ G,\ F,\ E,\ C,\ D)$ because if we choose to use the direction $(H,\ G,\ K)$ there would be a conflict at G. When $(C,\ D)$ is also down, the traffic that comes from the failure $(A,\ D)$ will stop at C and it will be rerouted by C. To reroute the traffic from the failure $(C,\ D)$, C has to transfer the traffic back to G, there are two possibilities here either take $(G,\ H,\ B,\ A,\ D)$ as the rerouting path, or transfer the traffic using $(G,\ K)$. We can not use $(G,\ H,\ B,\ A)$ because in that way the traffic will be transferred indeterminately between A and C. So, we have to use $(F,\ G,\ K)$ as the rerouting path in this situation.

Next, we consider the second situation where the two links $(E,\ C)$ and $(I,\ D)$ fail at the same time. When $(I,\ D)$ fails, using the same reasoning as for the previous case,

the only available rerouting path is $(I,\ J,\ O,\ K,\ G,\ F,\ E,\ C,\ D)$. As we choose to use $(F,\ G,\ K)$ in the previous case, in order to reroute the traffic of the failure $(E,\ C)$ and to assure there is no conflict with the previous case, we must also transfer the traffic through $F,\ G,\ K)$. Because both failures take place at the same time, the traffic will be transferred indeterminately between C and I, so the traffic can not be rerouted in this situation! So there is no rerouting scheme without conflict for destination D.

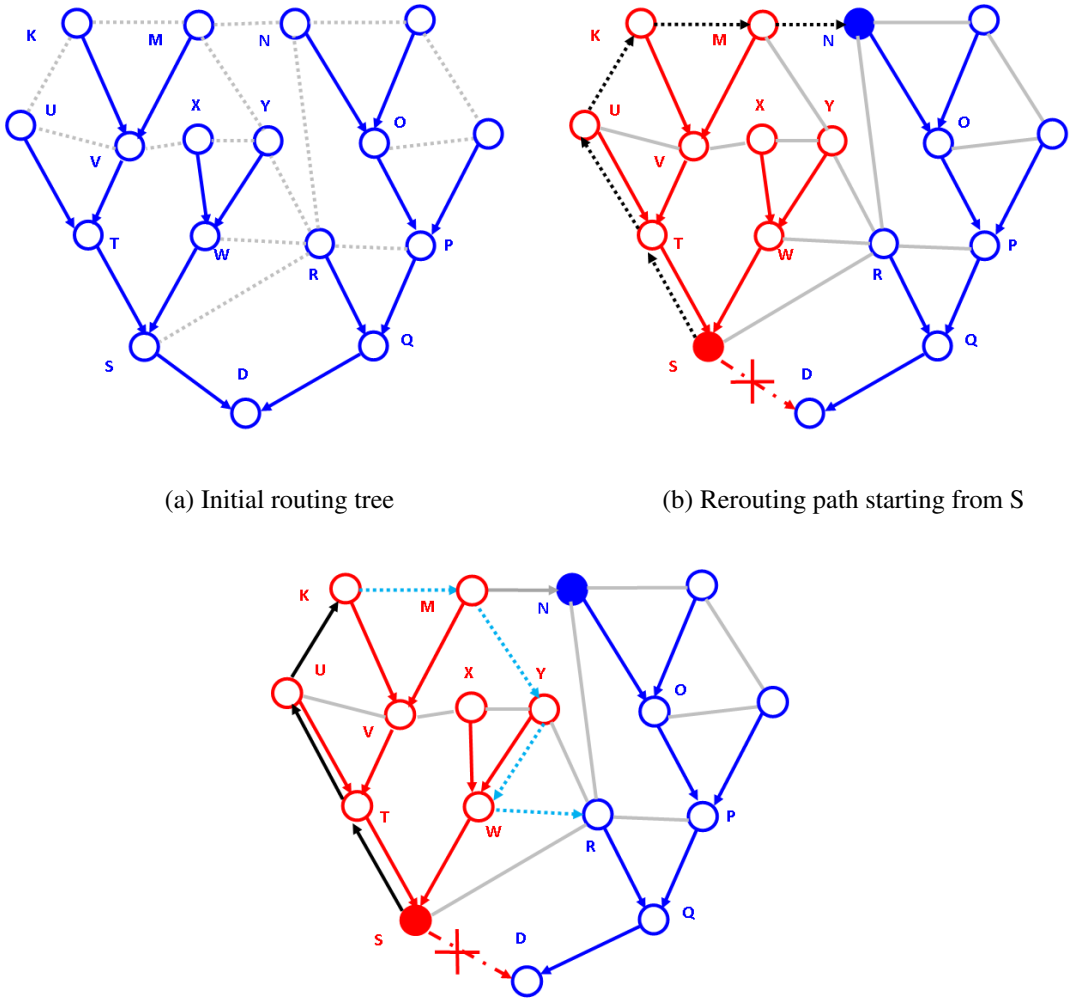
5.2 Update routing table

As we introduce at the beginning of this thesis, the reaction of the switches are automatic when they detect the failure. During this period, the traffic is rerouted following the precalculated rerouting path. When the network controller finds out that this failure is persistent, it has to recalculate the topology of the network and update the routing table of the nodes in the network. Here comes the question: Can we do this without affecting the routing of traffic in the network? As we know, even though the new routing path is calculated, we can not modify the routing table immediately. It will cause the lost of traffic in this case.

Looking at this question in literature, we find a solution which could solve a problem which is similar to our question. In this work, Lambert [42] has presented a method that could update the information in the routing table while avoiding the loss of packets and looping problem. The principle of this work is to update information in the routing table from the destination to the source. We could use this principle in our context. The routing table can be updated starting from the destination of the routing tree and going up until it reaches the node which detects the failure.

We will present here how to update the routing table in order to avoid the data loss

and looping problem.



(a) Initial routing tree

(b) Rerouting path starting from S

(c) The path recalculated from K

Figure 5.6: Update routing table

In this example, we have a routing tree to destination D (Figure 5.6(a)). In this figure, the arcs of the routing tree are described by the straight arrow while the link in the grape are described by the dotted link. When link $(S,\ D)$ fails (Figure 5.6(b)), the rerouting path uses the path $(S,\ T,\ U,\ K,\ M)$ in the red part, the bridge $(M,\ N)$, then

follow the nominal routing tree in the blue part until destination D. We can see that we do not have to update the routing table in the blue part of the routing tree because it would be the same as before. In the red part of the routing tree, for each node in the rerouting path, from an extremity of the bridge M to the node S, we have to calculate the new path to destination D. Also, updating the routing table for the nodes in these paths has to follow this order. We will show here an example of updating the path from K to destination D (Figure 5.6(c)). This path passes by $M,\ Y,\ W,\ R$. We can see that if the path for $Y,\ W$ is not updated, the packets will be transferred to S via the old routing tree. That is why we have to update the routing table for $W,\ Y,\ M$ in this order. After updating the path for the nodes in the old rerouting path (Figure 5.4(b)), we update other nodes in the red part whose routing tables have not been updated. We could do this process for all destination in the network.

Chapter 6

Conclusion

The limits of the current Internet and the appearance of new requirements promote research in network virtualization, for which failure resilience and traffic overload need to be treated. In this thesis, we mainly study failure resilience in virtual networks. When we have two virtual networks, the quality of service must be guaranteed in one virtual network while the other one is "best effort". That is the reason why our research focuses on one virtual plane only.

In this thesis, we have presented the literature on network virtualization, SDN and OpenFlow. We have proved that, in the context of OpenFlow networks, there exists a restoration scheme without conflict which could reroute the traffic to a same destination in case of single link or single node failure. We have proposed a mathematical model which permits to optimize the dimensioning of the network while calculating the rerouting paths. We have also introduced two heuristics which permit to compute a feasible solution within a short computation time and which could be applied to large networks. There are some special points that we want to emphasize about our methods. The main point here is its simplicity. Only the extremity nodes of the failure

have to know the failure while other nodes transfer the traffic normally as described in their routing table. These extremity nodes will reroute autonomously the traffic as pre-planned in their routing table without informing other nodes. This local reaction reduces greatly the communication time between network nodes and simplifies the network management. Another important point is that we calculate the rerouting paths for all failures in the network while the other methods can only guarantee QoS for a part of the network. This is the reason why the network dimensioning is expensive with our method. However, this is also the strongest point because we guarantee full restoration in the network with very little management effort.

There are several possible improvements of our heuristics which could be studied in future works. The perspectives of our work could also be how the network controller manages the routing tables as explained at the end of chapter 5.

Our work can also be extended to the overload issue in switches based networks. When a link is overloaded or a partial failure has occurred, it is possible to deliver part of traffic on the rerouting path. In consequence, our method can be applied directly in this case. When there are several layers with different priorities in our network, the overload issue should be studied deeper. For example, when a link is overloaded and we have at least three layers with different priorities, each layer can be treated with different manners. We have to make sure that the one with greatest priority functions normally. For the second one with less priority, we can reroute the traffic with our method. A part of the traffic can be probably lost for the "best-effort" one. The future works in this direction are also necessary to assure quality of service in virtual networks.

At the end of chapter 4, we have shown how to apply our method to several virtual networks. Also, different virtual layers with different priorities can modify the way

that our method is applied. More study is also needed in this research direction.

Bibliography

[1] Fernandes, N., Moreira, M., Moraes, I., Ferraz, L., Couto, R., Carvalho, H., Campista, M., Costa, L., Duarte, O., *Virtual networks: isolation, performance, and trends*, Annals of Telecommunications, vol. 66, pp. 339–355, Oct. 2011.

[2] RFC 5286, *Basic Specification for IP Fast Reroute: Loop-Free Alternates*, http://tools.ietf.org/html/rfc5286, 2008.

[3] Internet-Draft, *IP Fast Reroute Using Not-via Addresses*, https://tools.ietf.org/html/draft-ietf-rtgwg-ipfrr-notvia-addresses-11, 2013.

[4] Ho, K.-H., Wang, N., Pavlou, G., Botsiaris, C., *Optimizing Post-Failure Network Performance for IP Fast Reroute using Tunnels*, QShine'08, Proceedings of the 5th International ICST Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness, Hong Kong, China, 2008.

[5] Kvalbein, A., Hansen, A., Cicic, T., Gjessing, S., Lysne, O., *Fast IP Network Recovery using Multiple Routing Configurations*, INFOCOM 2006, 25th IEEE International Conference on Computer Communications, pp. 1-15, April, 2006.

[6] Pham, T.S., Carlier, J., Lattmann, J., Lutton, J.-L., Nace, D., Valeyre, L., *A restoration scheme for virtual networks using switches*, ICUMT 2012, 4th In-

ternational Congress on Ultra Modern Telecommunications and Control Systems and Workshops, pp. 800-805, Saint Petersburg, Russia, 2012.

[7] Pham, T.S., Carlier, J., Lattmann, J., Lutton, J.-L., Nace, D., Valeyre, L., *A fully distributed resilient routing scheme for switch-based networks*, 3PGCIC'2013, Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, pp. 92-97, Compiegne, France, Oct, 2013.

[8] Wang, J., Nelakuditi, S., *IP Fast Reroute with Failure Inferencing*, Proceedings of the 2007 SIGCOMM workshop on Internet network management, pp. 268-273, ACM New York, USA, 2007.

[9] Médard, M., Finn, S.G., Barry, R.A., Gallager, R.G., *Redundant trees for preplanned recovery in arbitrary vertex-redundant or edge-redundant graphs*, IEEE/ACM Transactions on Networking, vol. 7, issue 5, pp. 641-652, Oct, 1999.

[10] Doverspike, R., Li, G., Oikonomou, K., Ramakrishnan, K.K., Wang, D., *IP Backbone Design for Multimedia Distribution: Architecture and Performance*, IEEE INFOCOM 2007, pp. 1523-1531, Anchorage, AK, May, 2007.

[11] Xi, K., Chao, J., *IP Fast Rerouting for single-link/node failure recovery*, BROADNETS 2007, Fourth International Conference on Broadband Communications, Networks and Systems, pp. 142-151, Raleigh, NC, USA, Sept, 2007.

[12] Kamamura, S., Shimazaki, D., Hiramatsu, A., Nakazato, H., *Autonomous IP Fast Rerouting with Compressed Backup Flow Entries Using OpenFlow*, IEICE TRANSACTIONS on Information and Systems, Vol.E96-D, No.2, pp.184-192, February, 2013.

[13] He, J., Zhang-shen, R., Li, Y., Lee, C-y., Rexford, J., Chiang, M., *DaVinci: Dynamically adaptive virtual networks for a customized internet*, CoNEXT '08, Proceedings of the 2008 ACM CoNEXT Conference, ACM New York, USA, 2008.

[14] Yhu, Y., Zhang-shen, R., Rangarajan, S., Rexford, J., *Cabernet: connectivity architecture for better network services*, CoNEXT '08, ACM New York, USA, 2008.

[15] Wang, Y., Keller, R., Biskerborn, B., Merwe, J., Rexford, J., *Virtual routers on the move: live router migration as a network management primitive*, Proceedings of the ACM SIGCOMM, pp. 231-242, ACM New York, USA, August, 2008.

[16] Cherkaoui, O., Halima, E.,*Network virtualization under user control*, International Journal of Network Management, vol. 18, issue 2, pp. 147-158, March 2008.

[17] PlanetLab Project, http://www.planet-lab.org

[18] Geni Project, http://www.geni.net

[19] Bavier, A., Feamster, N., Huang, M., Peterson, L., Rexford, J., *In VINI Veritas: Realistic and Controlled Network Experimentation*, Proceeding of ACM SIGCOMM, pp. 3-14, October, 2006.

[20] Fire Projet, http://cordis.europa.eu/fp7/ict/fire

[21] Akari Project, http://www.nict.go.jp/en/photonic_nw/archi/akari/akari-top_e.html.

[22] VXLAN Gateway for Cisco Nexus 1000V Series Switches Deployment Guide, http://www.cisco.com/en/US/prod/collateral/switches/ps9441/ps9902/guide_c07-728864.html

[23] Chari, A., Morin, T., *NaaS4Cloud Opportunity of a Network-as-a-Service Layer for Cloud services*, Research Paper Orange, 2011.

[24] Chari, A., Houatra, D., Meulle, M., *Network for Clouds, a key Telco differentiator*, Research Paper Orange, 2010.

[25] Stern, T.E., Bala, K., *Multiwavelength Optical Networks: A Layered Approach*, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1999.

[26] Rahman, M.R., Aib, I., Boutaba, R., *Survivable Virtual Network Embedding*, in Proc. NetWORKing 2010, vol. 6091, pp.40-52, May, 2010.

[27] Zheng, Q., Shin, K.G., *Fault-tolerant real-time communication in distributed computing systems*, IEEE Trans. Parallel Distrib. Syst, vol. 9, issue 5, pp. 470-480, May 1998.

[28] Nascimento, M.R., Rothenberg, C.E., Salvador, M.R., Correa, C.A., Lucena, S.C., Magalhaes, M.F., *Virtual Routers as a Service: The RouteFlow Approach Leveraging Software-Defined Networks*, CFI '11, Proceedings of the 6th International Conference on Future Internet Technologies, pp. 34-37, ACM New York, USA, 2011.

[29] Juniper, JunosV App Engine, http://www.juniper.net/techpubs/en_US/release-independent/junosv-app-engine/.

[30] Federica Project, FEDERICA. http://www.fp7-federica.eu/.

[31] Ofelia Project, http://www.fp7-ofelia.eu/.

[32] Sherwood, R., Gibb, G., Yap, K.-K., Appenzeller, G., Casado, M., McKeown, N., Parulkar, G., *FlowVisor: A Network Virtualization Layer*, 2009. http://www.openflow.org/downloads/technicalreports/openflow-tr-2009-1-flowvisor.pdf.

[33] White paper Cisco, http://www.cisco.com/en/US/solutions/collateral/ns341/ns524/ns562/ns573/white_paper_c11-512753.pdf.

[34] Cisco Cloud Services Router 1000V Series, http://www.cisco.com/c/en/us/products/routers/cloud-services-router-1000v-series/index.html

[35] Sgambelluri, A., Giorgetti, A., Cugini, F., Paolucci, F., Castoldi, P., *OpenFlow-based segment protection in Ethernet networks*, Optical Communications and Networking, vol.5, no.9, pp. 1066-1075, Sept, 2013.

[36] Sharma, S., Staessens, D., Colle, D., Pickavet, M., Demeester, P., *A demonstration of automatic bootstrapping of resilient OpenFlow networks*, Integrated Network Management (IM 2013), pp.1066-1067, Ghent, May 2013.

[37] Sharma, S., Staessens, D., Colle, D., Pickavet, M., Demeester, P., *OpenFlow: Meeting Carrier-Grade Recovery Requirements*, Computer Communication, vol. 36, issue 6, pp. 656-665, March, 2013.

[38] Simon, J.L., Saint Jalme, C., Sol, D., Lattmann J., Valeyre, L., *Technical study on virtualization WAN*, Research paper Orange, 2013.

[39] Lattmann, J., *Virtualization*, Research Paper Orange, 2008.

[40] Popek, G. J., Goldberg, R. P., *Formal requirements for virtualizable third generation architectures*, Communications of the ACM, vol. 17, issue 7, July, 1974.

[41] Sharma, S., Staessens, D., Colle, D., Pickavet, M., Demeester, P., *Enabling fast failure recovery in OpenFlow networks*, Design of Reliable Communication Networks (DRCN 2011), pp. 164-171, Krakow, Oct, 2011.

[42] Lambert, A., Buob, M.-O., Uhlig, S.,*Improving internet-wide routing protocols convergence with MRPC timers*, CoNEXT'09, pp. 325-336, ACM New York, USA, 2009.

Appendix A

Technologies permitting networking virtualization

In this appendix, we analyze the main network technologies that provide functions for network virtualization. We distinguish several classes:

- Techniques that offer solutions for transfer plan virtualization. These technologies partially meet the requirements of virtualization. In general they allow the isolation the transfer plans but through a common control plan.

- Techniques that offer solutions for multiple control plans allow meeting the requirements of full virtualization of the network with a conventional approach.

- Emerging technologies around Software-Define-Network (SDN) and OpenFlow are an alternative and contributions to the network virtualization.

A.1 Techniques for transfer plan virtualization

A.1.1 Ethernet VLAN technology.

Description

Ethernet VLAN technology allows partitioning an Ethernet switch into several areas. Each area is associated with a single VLAN (Virtual Local Area Network) number and can have its own address space and its own broadcast domain. The lookup is performed on a common address table (VLAN, MAC). Therefore the VLAN Ethernet technology can create end- to-end virtual networks. Isolation is provided by the transfer scheme, the frames are identified by a VLAN number. This solution is not scalable, because it is limited to a maximum of 4094 VLANs.

A.1.2 MPLS technologies

MPLS [13] which is a traffic routing protocol based on label switching, can be also used as a protocol for transfer plan. It is based on the use of marking of packets by labels which permit switching along a predetermined path between a router and a home end router.

It relies on control plan protocols to distribute the labels. Thus, the LDP (Label Distribution Protocol) is used to distribute the labels along the shortest path given by the classical IGP. Several extensions to MPLS can create levels of virtualization in the data plan and/or control plan. Thus, we describe in this session:

- MPLS -TE and PCE that achieve traffic -engineering and virtualization paths in the network data plan level.

- MPLS VPN, which allows to create virtual private networks in the data plan to the control plan virtualization at the end: the VRF

MPLS -TE and PCE

Description

MPLS -TE (MPLS Traffic Engineering) [14] [15] is a technology that allows tunneling –that is, the traffic is routed explicitly and based on constraints in the network. MPLS -TE allows routing of network traffic through traffic -engineering techniques. It is based on a control plan IGP- TE (IGP link state routing with traffic -engineering extensions) and RSVP -TE (signaling protocol). The transfer plan uses MPLS .

Compared to the conventional IGP routing which follows the shortest path, MPLS -TE introduces the TED (Traffic Engineering Database), which is a network topology different from the LSDB (Link State Database). Indeed, IGP- TE adds new link parameters (bandwidth, TE metric , color, priority service class , ...) and route traffic based on these additional constraints and those of the user (explicit by specifying one or more intermediate nodes routing) .

MPLS -TE includes PCE [16] (Path Computation Element) . PCE is a functional block from architecture MPLS -TE. It is designed to perform path computation in one or more network elements which can be centralized (on a server) or distributed (on a router).

To summarize the MPLS -TE enables virtualization paths over a physical network infrastructure. In this way, it allows a network topology different from that offered by conventional IGP. However: - Virtualization is limited to virtualization paths - It does not include the control plan and therefore it offers only a partial network virtualization.

The ability of PCE to retrieve information from the network topology is a brick useful for network orchestration.

- It is particularly interesting in the centralized mode as it allows deporting some of the control plan in an external network element. It allows a better coordination between

the established TE - tunnels. - It helps to have an interface to another component designed to provide requests from applications and services. Thus, it could be triggered as needed, and in this sense it could also be programmable from the underlying network (reactivity following a change in resources or a change in the network topology).

MPLS -VPN solutions

MPLS VPN technology allows construction of virtual private networks (VPNs) over a shared network infrastructure. The network elements are implemented in edge routers (PE) of a core network routers (P) routers and clients (CE).

The only job of P routers ("Provider" or "Label Switch Routers") in the core network is MPLS label switching.

MPLS L3 VPN:

Transfer Plan

Packets transported in this mode between “Provider Edge Routers” are the IP packets encapsulated in MPLS frames, this is what justifies the name L3. Data transfer requires two levels of MPLS labels: a level that describes the tunnels between routers "Provider Edge" and in this tunnel, the IP packets are distinguished between customers at a second level label. To enable the switch by the PE there is a routing table and a table of virtual transfer "Virtual Routing and Forwarding table" (VRF) that connects the prefixes of customers with a PE destination and a second level label for distinguished clients. Therefore, plans addressing are separated and can overlap between multiple clients.

Control plan

Exchanges roads are made by MP-BGP (RFC-4760) which is an extension of BGP selected for its ability to advertise VPN routes. The extensions consist of adding, announcements of prefixes customers, namely information labels allocated by the EP,

the identifier of the VRF ("Route Distinguisher"), as well as specific BGP communities through the fields "route target".

Advantages:

Operator side • This is a standard built on IP solution. • The PE routers support hundreds of client connections , allowing adaptation at all levels ; • MPLS tunnels are shared on the core (LSP PE to PE) for all customers of a given PE a given output PE packets , the IGP on the core is shared for all customers (no IGP VPN) • VPN routing information conveyed by MP-BGP concern only PE and “Route Reflectors”, which is a network component used by BGP. This information is not maintained on all routers in the backbone.

Client side • Establishing an MPLS tunnel is independent of the level 2 technology • It is adaptable to all levels and therefore it is particularly well suited to the needs of small and medium enterprises solution ; • Transparent routing of IP packets between sites.

Disadvantages:

Operator side • This solution requires, in addition to the IGP, MPLS and LDP or RSVP protocols and MP-BGP and requires the installation of "Route Reflectors» to avoid "full- meshes" MP- MPMO between PE. • When not using «Traffic Engineering», MPLS tunnels follow the IGP shortest paths. During the fall of a link or IGP and LDP or RSVP router protocols follow the IGP routing protocol and therefore must converge to restore connectivity through a IGP step convergence and recovery of a path LDP or RSVP. The mechanism of «MPLS Fast Reroute» overcomes these difficulties and ensures continuity of service. However, its implementation requires the provision and preparation of bypass roads which complicates the administration and engineering of the core network. • The QoS guarantee is not the default mode when establish-

ing a VPN. You have to manage the establishment of links by the RSVP- TE. • It is not possible to specify that certain client packets must go through other defined paths through the tunnels established in the heart. • There is no mechanism to isolate the use of memory resources / CPU customer site (VRF): If a problem occurs on one of the VRF and that it increases its CPU consumption, lack of resources can affect other customers.

Client side • Because the MP-BGP protocol takes over the IGP customer network from the PE router, the client does not have a vision of the topology of the IGP network, which can cause problems in optimal paths use.

MPLS L2 VPN:

Packages transported in this fashion between the PE packets are level 2 issued by the EC, encapsulated in MPLS frames.

Transfer Plan

The L2VPN technology comes in two main forms: VPWS and VPLS

• VPWS (Virtual Private Wire Service): This is a technology that allows emulating point-to- point packet on a network. It provides the ability to connect two nodes regardless of the technology used by Level 2. The established tunnels behave towards routers "Provider Edge" as physical links or "Pseudo Wire" (PW). It is necessary to create a link PW PE by destination and by client. • VPLS (Virtual Private LAN Service): This architecture allows the emulation of a traditional local area network (LAN). It offers the ability to connect multiple remote LAN segments through a common network. The network acts as a "switch Ethernet». The links between the PE routers are of "Pseudo Wires» type.

Control Plan

There are two methods of signaling «Pseudo Wires»

• A method (called method «Martini ") implemented by all manufacturers, which is to get a session «Targeted- LDP» PE to PE. This session makes the exchange of MPLS labels on the way. This method has the disadvantage of requiring the maintenance of LDP sessions "full mesh" and their contexts. • A standard method (called method «Kompella ") implemented by Juniper, uses a variant of MP-BGP protocol. This method is intended to get a MP- BGP PE to PE.

Benefits of L2VPN

• Provider provides only the connection and client traffic is Level 2 • If layer 3 is used, only the client is master (management protocols, routing policy, and addressing plan). • Benefit related to VPWS: EP does not handle client MAC address tables, so there is little chance that a client can exhaust CPU / memory of a PE.

Disadvantages

• The absence of MP-BGP in the method "Martini" which is implemented by the majority of manufacturers makes the configuration tedious because there is no "self-discovery" (without the MP-BGP Route Reflectors).

a.2.2.3) Interest for the network virtualization.

Overall, MPLS VPN technologies allow creating isolated logical networks sharing a common infrastructure. The transfer plan is virtualized with MPLS in the core of the network and to the periphery through a label for references to independent contexts VPN (L3 VPN case and E-VPN). However, all these logical networks share the same level of control in the network core. MPLS VPN technologies thus only partially cover the full virtualization of the network.

a.3) Overlays over IP solutions

a.3.1) Description

GRE key (19) NVGRE (20) VxLAN (21) STT (22) encapsulation providing tech-

niques provide insulation layers in the virtual network Data Center (DC).

Thus, in the highly dynamic DC context, the tendency is to start overlays networks servers. OpenFlow, through the Open vSwitch (switch Opensource software integrated into the Linux kernel) is used for the construction of the overlay networks. This meets the need for automation and simplification of the business model, helping to reduce operational costs. a.3.2) Interest for the network virtualization.

These techniques can be used to create a large number of isolated virtual networks only in the context of DC. As MPLS technologies mentioned above, these technologies do not allow for full network virtualization (control plan shared between all virtual networks in the core of the network).

A.2 Techniques to run multiple control plans

Here we investigate technologies that have multiple control plans on the same network element. Several options are available: - The logic sharing control plan by instantiating several independent process that perform the function of each control plan . - The logic sharing control plan by independent contexts management through a process which performs the function of each control plan. - Physical sharing plans control by adding CPU cards.

A.2.1 Logic sharing

b.1.1) The IS-IS routing multi- topology

The IS-IS routing multi- topology (26) is a mechanism that enables a router to combine several independent IP topologies within a single IS-IS instance and independent decisions of routing packets, depending on the topology in which they are placed

.

b.1.2) The IS-IS routing multi-instance

The multi-instance routing is described in (27). This mechanism allows a router to run multiple routing instances and make routing decisions packets belonging to each of these instances independently.

.

b.1.3) Interest in relation to model full network virtualization

These technologies allow to implement just enough (minimal extensions of existing routing protocols) multiple instances and / or multiple routing topologies IS- IS or OSPF on a network. Used in conjunction with a mechanism of the transfer plan to identify the packets belonging to different bodies or topologies , these technologies are a first step towards full virtualization of the network since it allows the execution of multiple control plans IS- IS.

However, the scalability of these solutions must be analyzed as: - For multi- topology the maximum number of topologies is limited to 127. - Reservations concern CPU and memory resources required for the implementation of multi-instance. But solutions to deport instances of control plans to servers could be used.

A.2.2 Physical sharing: The “logical routers”

b.2.1) Definitions

A logical router (LR logical router) is a partition of a physical router (Router Physical PR) who has the same organs and the same functionality as a physical router. This is a subset of the physical machine which behaves as a full router. This is not a standard concept. Each manufacturer has different implementation choices. Each LR operates independently of other LRs in the same physical chassis.

A virtual router or a VRF (Virtual Routing and Forwarding instance) is an instance of simplified routing with a single routing table. A logical router can contain multiple virtual routers VRF.

b.2.2) Interest for the network virtualization

These solutions are not scalable and do not meet the need for flexibility and provision on demand. The insulation level transfer plan is provided but is not systematically a control plan.