UNIVERSITÉ CÔTE D'AZUR

ÉCOLE DOCTORALE
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THÈSE DE DOCTORAT

Algorithmes d'optimisation pour le Network Slicing pour la 5G

Adrien GAUSSERAN

Laboratoire d'Informatique, de Signaux et Systèmes de Sophia Antipolis (I3S)
UMR7271 UCA CNRS

Présentée en vue de l'obtention du grade de docteur en Informatique **d'**Université Côte d'Azur

Dirigée par : Nicolas NISSE
Co-encadrée par : Joanna MOULIERAC
Soutenue le : 09/11/2021

Devant le jury, composé de :
Mathieu BOUET, Directeur du network software lab., HDR, Thales
Frédéric GIROIRE, Directeur de recherche, CNRS, I3S, Univ. Côte d'Azur
Brigitte JAUMARD, Professeur, Concordia University, Canada
Adlen KSENTINI, Professeur, EURECOM, Sophia Antipolis
Joanna MOULIERAC, Maître de conférence, encadrante, Univ. Côte d'Azur
Nicolas NISSE, Chargé de Recherche, HDR, directeur, Inria de l'Univ. Côte d'Azur
Géraldine TEXIER, Professeur, IMT Atlantique
Guillaume URVOY-KELLER, Professeur, Univ. Côte d'Azur, Sophia Antipolis

ALGORITHMES D'OPTIMISATION POUR LE NETWORK SLICING POUR LA 5G

Optimization algorithms for Network Slicing for 5G

Adrien GAUSSERAN

Jury :

Rapporteurs

Mathieu BOUET, Directeur du network software lab., HDR, Thales
Géraldine TEXIER, Professeur, IMT Atlantique

Examinateurs

Frédéric GIROIRE, Directeur de recherche, CNRS, I3S, Univ. Côte d'Azur
Brigitte JAUMARD, Professeur, Concordia University, Canada
Adlen KSENTINI, Professeur, EURECOM, Sophia Antipolis
Guillaume URVOY-KELLER, Professeur, Univ. Côte d'Azur, Sophia Antipolis

Encadrants

Joanna MOULIERAC, Maître de conférence, encadrante, Univ. Côte d'Azur
Nicolas NISSE, Chargé de Recherche, HDR, directeur, Inria de l'Univ. Côte d'Azur

---

Université Côte d'Azur

Algorithmes d'optimisation pour le Network Slicing pour la 5G

Résumé

Notre décennie voit l'accroissement de l'utilisation des réseaux mobiles pour les acteurs industriels et administratifs ainsi que pour le grand public grâce à l'introduction de la 5ème génération de réseaux mobiles : la 5G. La 5G apporte une diversité de cas d'utilisation des réseaux mobiles et un nombre croissant de demandes avec des besoins très hétérogènes mais toujours avec de fortes contraintes de qualité de service (QoS). La 5G a été développée pour utiliser les technologies de réseaux définis par logiciel (SDN) et de virtualisation de fonctions réseaux (NFV). SDN sépare les plans de contrôle et de données et offre une gestion centralisée du réseau. NFV dissocie les fonctions réseaux du matériel qui les exécute grâce à la virtualisation. Ces technologies automatisent la gestion du réseau et le rendent plus flexible et adaptable à l'évolution du débit du trafic ainsi que de ses besoins. L'introduction du paradigme de découpage du réseau entraîne une division du réseau en des réseaux virtuels indépendants cohabitant sur la même infrastructure. Ce paradigme permettra de répondre aux besoins très hétérogènes des futures demandes. Dans cette thèse nous nous intéressons à l'optimisation de l'utilisation des ressources des réseaux de nouvelle génération afin de diminuer les coûts opérationnels et d'accepter plus de demandes. Nous étudions d'abord l'allocation de chaînes de fonctions de services, pour accepter rapidement les requêtes et répondre aux demandes diverses et abondantes des réseaux mobiles. Nous étudions ensuite la faisabilité de la reconfiguration Make-Before-Break des requêtes, qui permet de reconfigurer sans dégrader la QoS. Nous adaptons ensuite notre reconfiguration au network slicing pour l'utiliser sur les futures réseaux 5G. Enfin nous optimisons les périodes de reconfiguration grâce à un algorithme d'apprentissage par renforcement, réduisant ainsi les coûts de gestion.

Mots-clés : SDN, NFV, SFC, 5G, Slicing, Reconfiguration.

Optimization algorithms for Network Slicing for 5G

Abstract

Our decade is marked by an increase in the use of mobile networks for industrial and administration actors as well as for the general public thanks to their evolution. The introduction of the 5th generation of mobile networks, 5G, brings a diversity of use cases for mobile networks and a growing number of demands with very heterogeneous needs and with strong quality of service (QoS) constraints. The development of 5G relies on new techniques such as Software Defined Networking (SDN) and Network Function Virtualisation (NFV) technologies. SDN allows the separation of control and data planes by providing centralised network management. NFV decouples network functions from the hardware that performs them through virtualisation. The use of these two technologies automates the management of the network and makes it much more flexible and adaptable to changing traffic flows and requirements. The introduction of the network slicing paradigm leads to a division of the network into multiple independent virtual networks cohabiting on the same infrastructure. This paradigm allows to meet the very heterogeneous needs of future applications. In this thesis, we focus on optimising the resource utilisation of next generation networks in order to decrease operational costs and to accept more demands. We first study the allocation of service function chains, to quickly accept requests and to meet the diverse and high-volume demands of mobile networks. We then study the feasibility of Make-Before-Break reconfiguration of requests, which allows to reconfigure without degrading the QoS. We then scale up our reconfiguration and adapt it to network slicing to be used on future 5G networks. Finally, we optimise the reconfiguration periods by implementing a reinforcement learning algorithm, minimising the management costs.

Keywords: SDN, NFV, SFC, 5G, Slicing, Reconfiguration.

Remerciements

Je tiens d'abord à remercier Joanna Moulierac et Frédéric Giroire pour avoir encadré mon travail et m'avoir aidé dans l'amélioration de mes compétences de recherche. Je tiens aussi à remercier Nicolas Nisse pour m'avoir accompagné durant ma thèse et m'avoir aidé à aboutir mon manuscrit.

Je remercie aussi mes rapporteurs Géraldine Texier et Mathieu Bouet pour le temps qu'ils ont pris pour la lecture ainsi que pour les remarques qu'ils m'ont faites et qui m'ont permis d'achever mon manuscrit. Je remercie les membres de mon jury d'avoir accepté d'évaluer mon travail, Adlen Ksentini, Guillaume Urvoy-Keller et surtout Brigitte Jaumard pour m'avoir permis de travailler avec elle pendant 3 mois au sein de l'université Concordia au Canada.

Je remercie tous les membres de l'équipe COATI de m'avoir accueilli et d'avoir approfondi ma culture par les discussions que j'ai pu avoir avec eux et par les conseils qu'ils m'ont apportés.

Je tiens à remercier tous mes anciens enseignants et collègues, aussi bien à l'Inria qu'à l'I3S, qu'à toutes les périodes de ma vie. Toutes ces personnes qui, par leur culture, leur travail et leur passion m'ont permis de développer ma curiosité, ma conscience et mes capacités en partageant une partie de leur savoir.

Un grand merci à tous mes amis, ceux que j'ai connu par la fac et ceux qui étaient aux bons endroits aux bons moments. Tous ces moches et ces moins moches, c'est gros et ces maigres, ces débiles et ces encore plus débiles avec qui j'ai dansé, chanté, mangé et bu. Tous ces gens avec qui j'ai profité dans des champs, des caves, sous des chapiteaux ou des plots. Parfois à 12 et parfois à plus, souvent peu vêtu, hagard et à l'œil peu vif. Ces personnes de tous bords et tous horizons qui m'ont permis de sortir de ma bulle, de profiter de la vie et d'accaparer des objets sans resurgir sur autrui. Ces prolos, ces médecins et ces futurs ministres, ces avocats, ces ingés et ces dentistes : tous unis pour profiter aussi bien dans les bons moments que dans l'adversité.

Un pensée toute particulière au gangs des pistaches citronnées, Arthur, Guillaume, Joseph et Mehdi : le réseau de neurones le moins efficace de la décennie.

Enfin je tiens à remercier ma famille pour m'avoir soutenu pendant tant d'années. Ma sœur m'a donné le goût du voyage, mon frère celui de l'informatique, ma mère m'a offert la passion et mon père la curiosité. À mes parents qui ont toujours été là pour moi, qui ont été patients et qui m'ont soutenu quand j'étais perdu. Ils ont eu peur du Gi-Joe, ils ont eu peur du Tanguy, mais maintenant ils savent qu'ils ont mis au point un adulte autonome, un peu à l'ouest et à peu près fonctionnel.

Table of contents

List of Abbreviations

3GPP	3rd Generation Partnership Project
5G-PPP	5G Infrastructure Public Private Partnership
5GC	5G Core
ADSL	Asymmetric Digital Subscriber Line
API	Application Programming Interface
BSS	Business Support System
Capex	Capital Expenditure
CG	Column Generation
CPU	Central Processing Unit
CU	Centralized Unit
DPI	Deep Packet Inspection
DU	Distributed Unit
eMBB	Enhanced Mobile Broadband
EPC	Evolved Packet Core
ETSI	European Telecommunications Standards Institute
ILP	Integer Linear Program
IMT	International Mobile Telecommunications
IoT	Internet of Things
IP	Internet Protocol
LP	Linear Program
MANO	Management And Network Orchestration
MIMO	Multiple-Input Multiple-Output
MIP	Mixed Integer Program
MME	Mobility Management Entity
mMTC	Massive Machine Type Communications
NFV	Network Function Virtualisation
NGMN	Next Generation Mobile Networks Alliance

ONF	Open Network Foundation
Opex	Operational Expenditure
OSPF	Open Shortest Path First
OSS	Operations Support Systems
PGW	Packet Data Network Gateway
PHY	Physical
PP	Pricing Problem
QoS	Quality of Service
RAN	Radio Access Network
RIP	Routing Information Protocol
RMP	Restricted Master Problem
RRU	Remote Radio Unit
SDN	Software-Defined Networking
SFC	Service Function Chain
SGW	Serving Gateway
SLA	Service-Level Agreement
uRLLC	Ultra-Reliable and Low-Latency Communications
VM	Virtual Machine
VNF	Virtual Network Function
WLAN	Wireless Local Area Network

Introduction

0.1 Motivations

Over the last two decades the use of telecommunication networks has seen very large changes, from the number of users connected, to the amount of data exchanged. In France, for example, 17% of households were connected to the Internet through fixed networks in 2001 compared to 90% in 2018. In recent years, the development of mobile networks has been even more important. In France, 4G data consumption averaged 4.6GB per user per month in 2017, compared to 10.2GB in 2020, an increase of over 30% each year [dT21]. The evolution of networks is also driven by the needs they have to meet. The users of mobile networks have been mainly people so far. In the coming years, the users will also largely be industrials, administrations, and a large number of connected objects (IoT: Internet of Things objects). These IoTs objects range from sensors in the city to watches or cars. As a result, a large number of very heterogeneous use cases are emerging, with different needs and constraints. To meet these needs in terms of new services and bandwidth, networks have significantly changed, both in their infrastructure and in the technologies they use.

5G is envisioned to enable a multi-service network supporting a wide range of communication scenarios with a diverse set of performance and service requirements. All of these advances come with the promise of more bandwidth, less delay, and more flexibility for an ever increasing number of users and applications. In addition to infrastructure changes, to meet these requirements network management must evolve and two technologies are at the root of 5G. The combination of Software Defined Networking (SDN) and Network Function Virtualisation (NFV) is at the root of 5G. These two technologies improve the management of the network which becomes programmable and therefore automatable. It also becomes much more flexible and adaptable to changing traffic flows and requirements. One network for so many different use cases is not feasible, so the network slicing paradigm has emerged. The objective is to divide the network into multiple virtual networks, independent and isolated from each other, each meeting specific needs while providing a high Quality of Service (QoS).

In this context, the efficient provisioning of network and resources for a wide variety of applications with dynamic user demands is a real challenge. Network management as we know it must change and be adapted to network slicing. New management methods and optimisation algorithms need to be developed in order to efficiently manage increasingly dense and heterogeneous networks.

This thesis aims at providing a humble part of the answers to this challenge and contributing to the construction of tomorrow's network management.

0.2 Context

In traditional networks, packet routing is distributed across the network and is handled by routers. Each router communicates with its neighbours to create and maintain enough information about the network to build a routing table. They must keep this information up to date in order to route each packet to its destination. As can be seen on Figure 0.2.1(a), on this distributed architecture

both data and control plane are located on the same equipment. This has some advantages. In case of failure, each router is responsible for itself, so there are as many failure points as routers. Each router has only a limited number of neighbours. The decision making logic is simple. Finally this architecture can easily be scaled for very large networks. It also has disadvantages. The change of a route can require a human intervention on multiple routers. The distributivity of the control does not allow a global vision as well as a fine-grained control of the routing. Finally the human interventions on the routers can induce errors.

To understand how routing in traditional networks works, a network can be modelled as a graph where nodes represent routers to which resources and/or users will be connected and edges represent links connecting routers. To route data within the network, an interior gateway protocol must be used. Data may also be routed between networks, using exterior gateway protocol such as Border Gateway Protocol [RHL06], but they are not presented in this thesis as they are out of scope.

There are two main types of protocols that can be used for intra-domain routing:

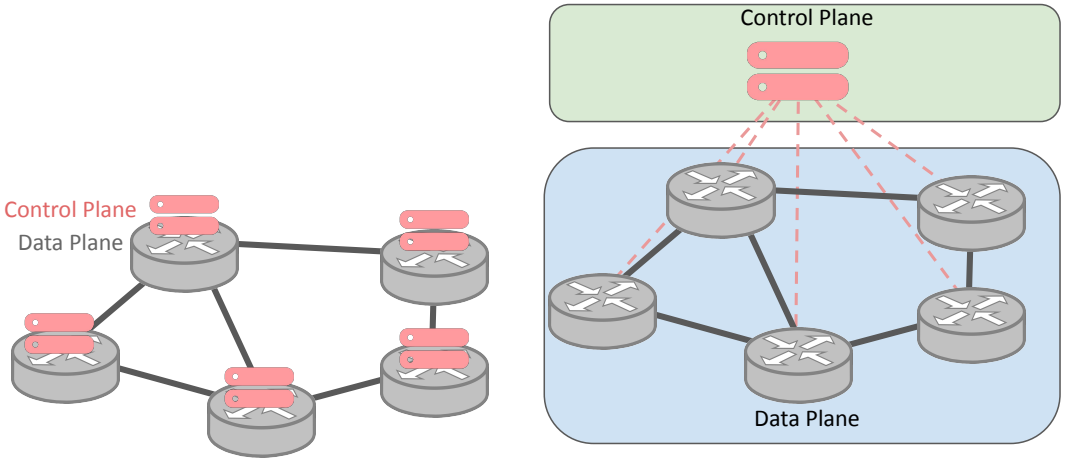
- In **distance vector routing** protocols, each router does not have information about the complete network topology. A router reports its distance values (number of hops to reach the destination) to its neighbours and receives similar reports from them. If changes occur, the router updates its routing table (a table that maps destination addresses to the router ports). If a router stops reporting, after some time its neighbours will stop forwarding packets on that route. Using these routing announcements, each router fills its routing table. During the next announcement cycle, a router announces the updated information in its routing table. This process continues until the routing table of each router converges to stable values (when the shortest paths are found). A widely used protocol is Routing Information Protocol (RIP) [Mal98]. For each known destination network, each router keeps the address of the neighbouring router with the lowest distance value. These best routes are broadcasted every 30 seconds. To avoid routing loops, the number of hops to reach the destination is limited to 15. RIP only takes into account the distance between two machines, but it does not consider the link state in order to choose the best possible bandwidth. RIP is based on the Bellman-Ford algorithm [Bel58].

- In **link-state routing** protocols, each router has information about the entire network topology. Using the local topology information, every router independently determines the best next hop for all possible destinations. The routing table is composed of the set of the best next hops. A good example is the Open Shortest Path First (OSPF) protocol [Moy98]. OSPF has been designed expressly for the TCP/IP internet environment. This routing protocol collects link state information (the bandwidth) from the routers in the network and determines the routing table information to forward packets. This happens by creating a topology map for the network and each router obtains it through flooding. Unlike RIP, OSPF only exchanges routing information when there is a change in the network topology. Every router then runs the Dijkstra algorithm on its database to build the shortest-path tree and, thus, the IP routing table. The OSPF protocol is adapted to complex networks that have multiple subnets and are intended to facilitate network administration and traffic optimisation. It efficiently calculates the shortest path with minimal network traffic when the change occurs.

Table 1 shows the routing table of a router on a traditional network. This table has six entries and each entry is divided into several parts. The letters in blue correspond to the protocol used to

	Destination	Distances	Next hop	Update	Interface
S*	0.0.0.0/0	[1/0]	via 10.0.0.2		
C	10.0.0.0/30		is directly connected,		Serial0/0
C	10.0.0.8/30		is directly connected,		FastEthernet0/1
R	10.0.10.32/27	[120/2]	via 10.0.0.8,	00:00:14,	FastEthernet0/1
O	10.0.10.0/24	[110/62]	via 10.0.0.1,	00:43:25,	Serial0/0
D	10.0.10.0/24	[90/186487]	via 10.0.0.5,	00:14:41,	Serial0/1

Table 1 – Routing table on a traditional network



(a) Traditional Network (b) SDN Network

Figure 0.2.1 – Standard VS SDN network

learn the route: O for OSPF, R for RIP, D for EIGRP, C if the destination is directly connected, and S to say that the route is static (the * means that it is the default route). On the right of the letters is the destination network address and its mask. The red part corresponds to the administrative distance (right) and to the distance metric of the route calculated by the protocol (left). The administrative distance is an arbitrary number assigned to rank the preferred routes according to the protocol (the preferred being one): 1 is for static, 90 for EIGRP, 110 for OSPF, and 120 for RIP. After the metrics comes the address of the next jump, then the time since the last route update in hours:minutes:seconds. Finally, the interface from which the destination network can be reached is specified.

0.2.1 Software Defined Networks

Software Defined Network (SDN) is an emerging paradigm that aims at simplifying network management. It is democratized and standardized by the Open Network Foundation [Fou16] (ONF) which specifies its architecture. The management of the network is simplified by separating the control plane of traditional networking devices (e.g.switches, routers) from the data plane, as shown in Figure 0.2.1(b). Network intelligence is logically centralized in an SDN controller that

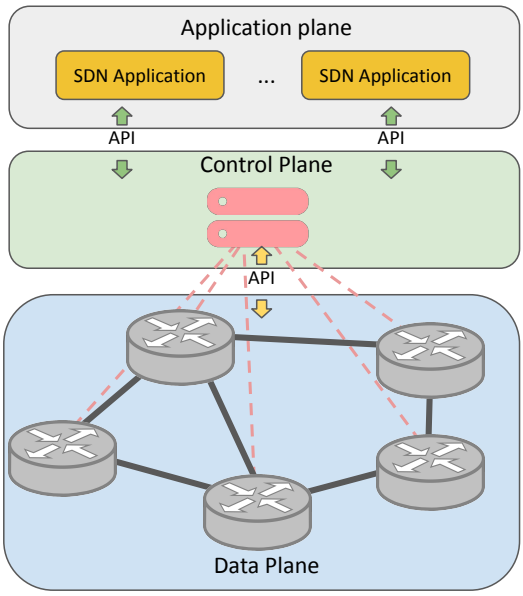


Figure 0.2.2 – SDN Architecture

maintains a global view of the network state. This centralization completely changes the management of the packets. The network becomes programmable and the routes can be changed dynamically without human intervention on the routers. This allows network administration to adapt on demand to different concurrent applications, enabling more precise control of the network and therefore better resource management [KF13]. In 2013, Google introduced its first operational SDN on its private WAN connecting all its data centres across the world [JKM+13]. Today SDN is used in many corporate networks. Its architecture aims at optimising traffic management between data centers and reduce over-provisioning of links between them. It has been shown the ability to use its data center network at full capacity by achieving close to 100% link utilisation, demonstrating the feasibility and benefits of SDN in a real production network. As developed in [KREV+15] and presented in Figure 0.2.2 the SDN architecture is composed of three main layers: data plane, control plane and application plane.

- **The data plane** is made up of all the network equipments forming the underlying network. It consists of switches and routers (and mobile equipment such as antennas for mobile networks) connected to each other by different types of links. The goal of this layer is to forward the network traffic and each device must expose its capabilities to the control plane using an Application Programming Interface (API).

- In **the control plane** the SDN controller is the core of the logic in the SDN architecture. It is a software entity that has exclusive control over a set of data plane resources. It can also provide abstract information about the attributes and operations that can be performed by the elements of the data plane to the application plane.
  The controller supervises the network forwarding behaviour through an open API by aggregating the various network information and application requests to enable the best routing for each application. It does it while making the most efficient use of the data plane resources. It also allows to dynamically reconfigure routes according to policy changes or to

respond to failures on the data plane.
The SDN controller is seen as a single logical entity but it may be implemented as many software components, distributed on many physical platforms. To be seen as one entity those components must maintain a synchronized and self-consistent view of information and status. A single centralized controller may be sufficient for a small network. On the other hand, it represents a risk because in case of failure, the control of the whole network is interrupted. It also may not be sufficient to resiliently manage the network elements of the data plane. The controller can also be distributed within a single cluster or to different nodes in the network. When a node fails, another node takes over the tasks.

- **The application plane** is the manager of the network. It is responsible for analysing all the information about the network topology and the network state. It manages the different control functions that will be given to the control plane (through APIs) to adapt the traffic on the data plane. The SDN applications in the application plane relay their network requirements to the controller plane. The SDN controller interprets the applications' requirements and provides low-level control over the network elements, while feeding relevant information to the SDN applications. An SDN controller can manage concurrent application requests for limited network resources.

Challenges

Like any new paradigm, SDN has raised challenges. The SDN controller must be able to interconnect with equipment already in use on traditional networks. The entire infrastructure does not have to be changed. The administration being centralized, routers must be able to report information about the state of the network. In [LHK+13] the authors propose a solution to deploy an SDN network and connect it to other IP networks seamlessly. They integrate a border gateway protocol process to the SDN controller that makes SDN-IP peering possible. Their application allows the SDN network to appear as a single router to the connected routers of other IP networks. Without breaking the traffic, the application updates the information received by the controller and then allows it to coordinate the actions of the routers to route the traffic.

Since SDN is used in new networks, the question of scaling arises. The control plane and the data plane being decoupled, when a flow management rule is added it must be forwarded to the routers, thus increasing control plane related traffic. This traffic increases the load on the network and at large scale it can potentially create a bottleneck. As updates to flow management rules and network information are also propagated over the network, this can create additional latency, with a risk of deteriorating the quality of service. Some studies focus on these scalability issues. In [YG12] the authors propose a SDN controller using two layers for the control plane. The top layer is the central controller which has a global view of the network. The bottom layer consists of several controllers used by local applications using a single switch. These local controllers can be instantiated when needed and handle the majority of frequent events, reducing the load on the top layer.

SDN has to work on large-scale networks with a large number of heterogeneous requests requiring a large number of different rules to handle all the flows. These rules are stored in flow tables inside the network devices, which use an expensive and small type of memory, limiting the number of rules to a few thousand. Since their size can be substantial, some works focus on compression techniques to reduce the number of flow entries. Giroire *et al.* [GHM15] develop a destination-based heuristic to address the problem of determining a compact two-dimension routing table using aggregation rules that has the same behaviour as the original routing table. They

manage to often save half the space of the table. One of the contributions of [GHMP18] is the reduction of the size of the forwarding tables using wildcard rules by aggregating rules with the same action on the corresponding fields. They develop an Integer Linear Program (ILP) and two heuristics that outperform a solution using default port compression by having a compression ratio between 15% and over 60% depending on the number of ports. They study multi-field compression, using extra wildcards compared to a default port compression. The additional wildcards are used to aggregate flows having a field with a specific value (such as a common destination).

Distributing the SDN controller across different nodes ensures that the network is resilient and does not become brainless in the case of a single failure. This distributivity poses a challenge because the controller must be synchronized. This requires exchanges on the control plane between the different devices that host the controller. Finally, in case of failure, the controller must be able to reorganize itself in an elastic way. These issues have been particularly studied in some works. Bari *et al.* [BRC+13] propose a framework that, instead of optimally placing SDN controllers, dynamically adjusts the number of active controllers. Each controller manages a subset of switches based on network dynamics. They develop an ILP and two heuristics to ensure minimal flow establishment and communication time. One of their heuristics, a greedy approach based on the knapsack problem, manage to balance these two objectives while running fast enough to be used in real time. Their second heuristic is a simulated annealing based meta-heuristic approach that provides better results, but takes longer to converge. In [DHM+13] the authors propose to dynamically scale the size of the SDN controller set according to traffic conditions. They also perform dynamic load migration between controllers to avoid overloading any of them. By monitoring switch CPU utilization, they determine which switches are most likely to send the most messages and overload a controller. They emulate their architecture using mininet [LHM10] and show that their solution enables load balancing between different controllers, resulting in better throughput and faster response time. In [KBK+12], the controller is not distributed but only one of its functions. The link and node failure detection protocol is distributed within the switches. Their method allows the switches to emit the messages making the restorations possible without inducing a load on the controller, thus allowing a scalable recovery of the data plan, in less than 50ms.

Advantages

Although SDN poses some challenges it also comes with many advantages [ONF14]:

- **Programmability**: Once the infrastructure is deployed, no human intervention is required on the equipment to modify the routes, all these actions are managed directly by the controller. Each SDN application can have a different policy which is programmed and transmitted to the controller. The controller can manage the heterogeneous needs of the different applications and adapt the traffic flow accordingly.

- **Centralization**: The network view being centralised, decisions are made with a global view and therefore they will be more efficient in managing network resources as well as in respecting the different SDN application policies. This allows easy manipulations of large amounts of traffic data as well as for dealing with very low latency requests.

- **Scalability**: As discussed, scalability can be a challenge with SDN because of the potentially large number of different rules that need to be managed. But centralized control brings a considerable advantage as it gives access to the complete network information. This makes

traffic routes adaptable to improve load balancing and prevent congestion and QoS degradation.

- **Modularity and cost reduction**: One of the purpose of SDN is to establish an open standard to simplify interconnection. This objective is supported by the ONF. The deployment of SDN enables interconnection of devices even if they are from different manufacturers. This heterogeneity of equipments allows network owners to meet their needs more effectively, as well as greater openness to competition and thus a reduction in costs and a stimulus to hardware innovation.

- **Security**: The centralization of control provides a quicker and more effective identification and response in case of an attack on the data plane or in case of failure of some equipment of the data plane. The resilience of the network can be improved through migration and restoration functions provided by the controller.

In recent years, the increase in traffic, the number of requests and their heterogeneity, and the constraints they impose such as very low latency or high mobility, have been significant. To address these developments, SDN is becoming an essential paradigm in the management of next generation networks.

0.2.2 Network Function Virtualisation and Service Function Chaining

The purpose of the data plane is not only to route traffic but also to provide network services. To this end, in addition to routers and switches, it is also composed of various network functions such as firewalls or video optimisers.

0.2.2.1 Network Function Virtualisation

In a traditional network, network functions are executed on specialised and proprietary hardware called middleboxes [CB02]. These devices are placed on the network and can only be added and/or moved by specialised technicians. These additions and moves are slow and fail to provide an adaptable network to traffic and demands. When a new service is added, new equipments must be purchased and placed on the network. Those equipments must be compatible with those already in use. In the case of a failure it takes a relatively long time to repair or change the hardware.

To address these issues a new paradigm has emerged, Network Function Virtualisation (NFV). The goal is to shift from specialised hardware appliances to general-purpose hardware in order to deal with the major problems of today's enterprise middlebox infrastructure. These problems range from cost to capacity limitations to management complexity and failures. [SHS+12]. NFV change the way network infrastructure is deployed and managed by virtualising network functions on standard equipments such as servers or switches. To install a new function, instead of buying a new machine and sending technicians to install it, the network manager simply runs the software for the network function on a virtual machine. As shown in Figure 0.2.3, virtual machines can be instantiated on servers and a server can execute a multitude of functions as long as its capacity is sufficient. These Virtual Network Functions (VNF) can be run on demand in a very short time making their placement adaptable to the traffic load and to the different demand requirements. The aim of virtualisation is therefore to make the management of network services more flexible. NFV adds dynamism by creating and orchestrating network functions in real time. This makes the management adaptable to the needs and traffic in an environment where function placement was

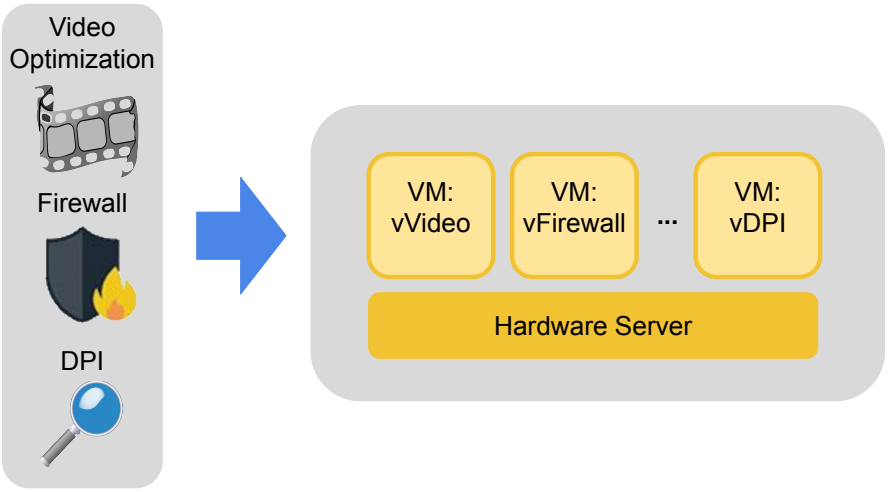


Figure 0.2.3 – Network Function Virtualisation means executing network functions on standard servers

previously static. It improves network resource utility and reduces costs. It also improves profits by being able to offer customised services on demand.

Similarly to the control and management layers that SDN has, NFV has its orchestrator. NFV-MANO (management and network orchestration) is a framework developed by the European Telecommunications Standards Institute (ETSI) [00114]. It is used to manage and orchestrate all resources in a virtualised data center, including computation, network, storage and virtual machine (VM) resources. NFV-MANO works within existing systems using standard VNF templates that define the function and profile details of VNFs. It gives users the ability to choose from existing NFV resources to deploy their functions. NFV-MANO is divided into three functional elements:

- **The Virtualised Infrastructure Manager** controls computing machines (CPU, memory), storage and network resources.
- **The NFV orchestrator** is responsible for discovering available services. It manages the availability/allocation/release of virtualised resources and the life cycle of network services. Finally, it monitors resource failures and performance.
- **The VNF Manager** is responsible for the instantiation and life cycle management of the VNF instances, scaling the VNFs (increasing or decreasing the capacity of the VNF) and updating them.

Challenges

Like SDN, NFV has raised challenges, many of which have been studied. The flexibility induced by the possibility of launching virtual functions on demand implies optimising the number and localisation of these functions in a dynamic way. In [CLX+10] the authors propose a heuristic based on the constraint set cover problem to dynamically migrate and allocate VNFs when the topology of the network changes (failure or addition of infrastructure). They modify the allocation of VNFs to satisfy the embedded virtual networks by taking into account the capacities of nodes and links as well as path delays. Their objective is to minimise migration costs as well as the cost of installing new VNFs. Their heuristic gives results ranging from 11% to 26% of the optimal.

The cost of using these virtual functions cannot be managed in the same way as purchasing hardware. Indeed, as several instances of the same software can be run at the same time, it is necessary to define a cost and licence management policy as recall in [01017].

As these licence costs become a significant part of expenditure, they are one of the two expenses we are reducing in our pursuit of operational cost minimisation. In chapters 3, 4 and 5, our solutions allow to decrease the VNFs licence costs by 20% to almost 40% depending on the topology used and the network load.

Finally, the security of virtual machines cannot be approached in the same way as on physical equipment. It is necessary to think about security strategies for the NFV paradigm and using the NVF paradigm. Jaeger [Jae15] described a security extension for the standard ETSI NFV architecture for a hybrid network (physical and virtual network functions are deployed). Its security orchestrator is intended to interact with Network Management and Orchestration ("MANO") entities while being independent of the ETSI NFV reference architecture. The roles of his orchestrator are automated control of the deployment and configuration of virtual security functions and trust management within network services.

Advantages

The use of NFV is now gaining popularity [GJPGA12] among network operators because of the opportunity it brings them. It brings an opportunity to offer services in a more agile way, capable of operating at extremely large scales. Above all, it enables services to be delivered more quickly by exploiting the intrinsic properties of virtualisation. There are many advantages of using NFV compared to the proprietary middlebox of the traditional networks:

- **Flexibility**: Networks can be modified without updating any hardware. Network operators can deploy new network services dynamically on the same hardware. Managing the installation of the various services can be done quickly and automatically without human intervention.
- **Scalability**: The servers can be switched off and on to follow the network load and new virtual functions can be launched to decongest routes and prevent bottlenecks.
- **Capital expenditure (Capex)**: The cost of buying and maintaining the middlebox for large network may rise to over one million dollars over five years [SHS+12]. By running the functions on standard servers, operators no longer need expensive proprietary hardware. This allows for greater openness to competition, as well as simplified maintenance resulting in lower costs. In addition, upgrading software running on multiple machines is much less expensive than changing a whole set of equipment.
- **Operational expenditure (Opex)**: Centralized and automated orchestration of service deployment simplifies management and reduces associated costs. It also reduces energy costs if equipment can be turned off by consolidating multiple functions on a single machine.

0.2.2.2 Service Function Chaining

Even though SDN and NFV are two different independent technologies, they are complementary and share many properties [Fou15]. With SDN optimising traffic routing and VFN optimising network function placement, the work of these two paradigms can be synergistic and the action of one can benefit the other to improve networks and service delivery over them [MGT+15]. A

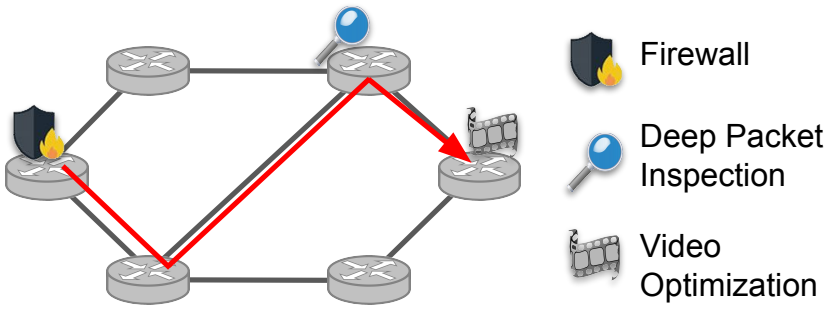


Figure 0.2.4 – An example of SFC allocation

customer's request within a network can be simply summarised as traffic going from a source to a destination and passing through a network service. As the European Telecommunications Standards Institute recalls, a service may requires the traffic flow to be processed by a sequence of network functions in a specific order [00114]. This sequence of functions can be composed of physical network functions and/or VNFs, linked together. This notion is known as Service Function Chaining (SFC) [QN15]. Users may have different requirements for network services. To meet these demands, various sets of ordered chains of network functions may be required. A multitude of SFCs can therefore be offered by a network operator to meet a range of heterogeneous needs.

An example of an SFC placed on a network is shown in Figure 0.2.4. On this network, a chain of three functions must be placed. The traffic must be processed in order by a firewall, a deep packet inspection function and a video optimisation function. The three functions must be placed on nodes that can host them. It is sometimes more convenient to place several functions on the same node, to reduce the energy consumption of a node for example. But it can also be more interesting to allocate the functions on different nodes because these functions can be used by other chains and it is necessary to avoid overloading the nodes and the links attached to them. Considering the traffic routing at the same time as the placement of the functions can also be interesting, as it avoids having too long routes and therefore does not saturate the network unnecessarily. In this example, the functions are placed in such a way that the flow goes through its shortest path.

The concept of SFC is not dependent on SDN and NFV. The implementation of function chaining can be done between physical network functions through the use of segment routing. This paradigm encapsulates the packet in headers in a recursive manner in order for the packet to be sent through a predetermined set of intermediate routers until its destination. By combining segment routing and NFV it is possible to make the management of SFCs much more flexible. In [ACF+17], when the packet is sent the VNFs are chosen and the packet is encapsulated in the corresponding headers to allow the service (the SFC) to be deployed. In [WBIC19], the authors propose an architecture for deployment on NFV networks using existing routing protocols (OSPF) and allowing the placement of SFCs without the use of SDN. Unlike [ACF+17], the ingress node is not in charge of setting the list of nodes to be visited to constitute the SFC. Instead, the decision is completely distributed within autonomous NFV routers. They propose a Linear Program (LP) with the objective of minimising costs (traffic and VNF) and compare their solution to a centralised SFC placement. The cost of their solution never exceeds 2 times the cost of the centralized solution

and is mostly between 1 and 1.4 times. SFCs can therefore be deployed using traditional networks and without the deployment of SDN.

However, even with this it is easy to see how difficult it is to orchestrate SFCs in a traditional network. Indeed, if too many users request the same service, the SFC that provides that service can quickly become a bottleneck. Rerouting requests (changing the route taken by the flow) to a new network function chain can be slow and complicated without SDN. If there are no other instances of the same network functions, it will also be impossible to create a new SFC dynamically without using the NFV paradigm as installing a new middlebox cannot be done in real time. As network requirements change rapidly, it seems complicated to be able to provide new services quickly taking into account the time to purchase and install new middleboxes [BJSE16]. Indeed, using SDN and NFV together bring significant advantages. In case of congestion, it is possible to quickly allocate new VNFs on the network, create a route between them and thus generate a new instance of an SFC producing the same service. New requests can be routed to this SFC and bottlenecks are avoided.

A notable improvement is also the addition of new services. Service requirements are evolving even more rapidly with the arrival of new technologies such as 5G and the network must therefore be able to adapt. Using SFC on a SDN and NFV based network eases the implementation of these new services by minimising the addition of network equipment. When a new service is required, a combination of network functions is needed to provide the service. Let us take for example a tracking service for a fleet of autonomous taxis. In [QKA18], an intrusion detection system is combined with a firewall for security. A service prioritisation function is added to ensure high priority is given to the service. Finally, a remote control function is added to allow the management of autonomous vehicles. The service can then be instantiated and the VNFs will be allocated automatically. The route between these VNFs will also be reserved automatically to create an SFC and users can now use this service without the need for technicians to intervene on the network infrastructure.

Managing and implementing SFC by combining SDN and NFV has many advantages but exposes also to many challenges. Since SDN and NFV are used, inherently all the related challenges are also included. Resource allocation for SFCs is an even more challenging problem because it links the optimisation of VNF allocation to path optimisation, taking into account multiple constraints such as link capacities, server capacities and execution and propagation times. In addition, multiple objectives can be taken into account such as improving the utility of the network, energy saving, increasing the acceptance of requests, reliability or simply reducing costs.

Gu *et al.* [GZT+19] jointly study the problem of online network flow scheduling and VNF resource allocation. They develop a dynamic distributed algorithm for flow and rate control by applying Lyapunov's optimisation theory. Their objective is to maximise the utility of the network while ensuring system stability and guaranteeing fairness among the SFCs. Their algorithm achieves performance close to the optimal solution, with fairness consideration on their scenario set.

In [HTGJ18] the authors propose 3 decomposition models based on column generation to minimise energy consumption when using SFCs in an SDN and NFV based environment. Their solutions reduce energy consumption by up to 60% during low load periods by consolidating VNFs on common devices and turning off unused equipment.

Tomassilli *et al.* [THGJ18] study the resilient allocation of SFCs while minimising the bandwidth used in the network. They develop an ILP and a column generation model to handle dedicated protection (having a one plus one path for each SFC). They again develop an ILP and a

column generation model to deal with shared protection (against single-link failure). Their solution is never further than 4% from the optimal.

The authors in [KHZ17] propose two solutions for placing the SFCs while minimising the bandwidth. A first approach is inspired by the Perfect 2-factor theory and transforms the SFCs into cycles and then find these cycles to have an optimal placement. Although the first approach is only valid for SFCs consisting of three VNFs, they propose a second approach to handle a larger number of VNFs. They propose a heuristic working on what we will present later as a layer graph (see section 2.4). They then solve the maximal flow problem to find a valid placement and routing. Their solutions are competitive in terms of performance and complexity compared to the current state of the art.

In this thesis, we will compare different methods for the SFC placement problem in chapter 2. Our interest is the use of a structure called layered graph (section 2.4), which allows to easily model the ordering of functions within SFCs. This structure is used throughout this thesis. In chapter 3, we study the problem of reconfiguring these SFCs. The study will be carried out on both static and dynamic scenarios. Our goal is to show the efficiency of a reconfiguration technique in increasing the acceptance of requests as well as in decreasing the costs of the network.

0.2.3 5G Networks

The evolution of networks is also taking place through mobile networks. Since 40 years, the deployment of the first generation of mobile networks has continuously evolved, both in the technology and in the provided services. For some years now, IP networks have considerably changed but also mobile networks in terms of bandwidth requirements and number of connections and these are still increasing. On the contrary, new needs cannot be met without an evolution of networks, which is driving the adoption of a new generation of mobile networks [ARS16]. In this context, 5G is envisioned to enable a multi-service network supporting a wide range of communication scenarios with a diverse set of performance and service requirements. It needs to meet all these requirements while promising more bandwidth, less delay and more flexibility for an ever increasing number of users and applications.

Since the introduction of 3G, mobile networks have supported multimedia content and although the speed was not high, it was sufficient for the video quality of the time for the general public. The arrival of 4G was even more promising. In addition to greatly increasing throughput and improving Quality of Service, 4G brought IP interoperability for seamless access to the mobile Internet. Mainstream users could therefore have the same mobile network usage as their domestic network [Var12]. This means being able to watch TV or high quality streaming video, make video calls or play video games, with some operators even providing 4G modems to connect homes to the internet rather than using copper lines like *ADSL*. Today the need to change mobile network is felt in particular for two reasons. The 4G network is reaching its limits in terms of bandwidth. The increase of users and their personal needs for bandwidth is causing congestion problems. The second reason is the increase in heterogeneous needs arising from the expansion and emergence of new special user groups. These include the expansion of connected objects, the development of autonomous cars, the emergence of smart cities, and industrials who intend to exploit mobile networks. The 4G network was not designed to be so adaptable and 5G is designed to meet these needs. 5G promises greater speed, capacity and density of connected devices with less interferences. It also delivers greater energy efficiency, greater mobility and lower latency. To

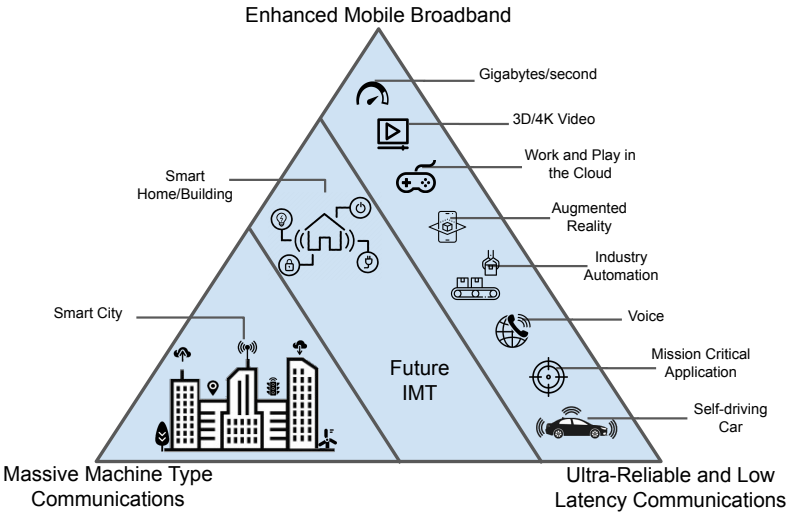
achieve this, new technologies are being introduced, such as new architectures and optimisation designed to meet these needs.

With 5G come new needs with very heterogeneous requirements that may vary considerably depending on the use case. This may involve very low latency, very high throughput or a massive amount of connections, all with a high QoS requirement.

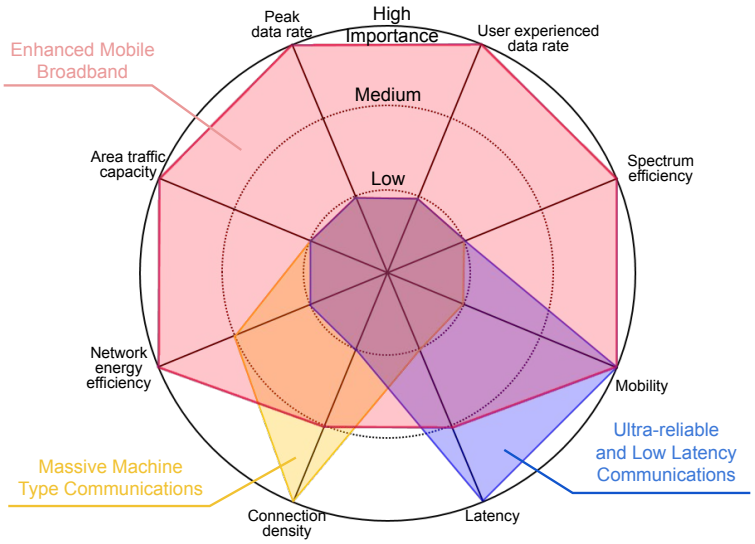
To have an overview of these use cases and their requirements refer to Figure 0.2.5(a). It shows the great disparity of the use cases but especially of their needs. The triangle represents the three main types of needs: enhanced mobile broadband, mass communication and reliable communication with very low latency. The position of the use cases in the triangle represents the importance of the shared requirements for each use case. For example, in the case of autonomous cars, reliable communications and very low latency are extremely important requirements compared to the other two. Unlike 4G, which had the principle of one network for everything, which does not allow for performance. 5G is designed to meet specific combinations of needs, making it highly efficient.

To address these issues, 5G has five main characteristics as outlined in these white papers [22.16, 38.20] and surveys [DGT+19, FZI16]:

- 5G promises **very high bandwidth** for both reception and transmission. To support uses such as 4k video-on-demand, 5G will deliver download speeds of over 1 gigabit per second, putting it on a par with some optical fibre connections.
- **Very low latency** is required by some real-time applications with very low delay tolerance, such as autonomous cars or critical life support systems. The latency target is as low as 1ms.
- The feature of **very high mobility** is promised by 5G, which means continuous coverage for a user moving around the network. This feature has to be implemented both in very dense areas of users and antennas like in cities for the autonomous car. It must also be implemented in much less dense areas and at much faster speeds to cover high speed trains up to 500km/h while respecting the QoS.
- **Massive connectivity** is required with the sharp increase in IoT. This includes connected objects used by the general public such as smart-watches but also autonomous devices used in the industry or sensors for smart cities.
- Some applications will require **high reliability** of connections, which implies very high availability and resilience of communications. These applications may include autonomous cars and critical life support systems.



(a) 5G Use Cases



(b) 5G Needs classes

Figure 0.2.5 – 5G Use Cases and Needs

Even with all these targeting features, 5G cannot accommodate all these demands at once for every application. But as mentioned earlier, the very heterogeneous use cases of 5G have very different needs. As can be seen in Figure 0.2.5(a) the majority of the applications can be classified into three groups each with different specifications and needs(Figure 0.2.5(b) [ITU18]). Use cases of class *Enhanced Mobile Broadband* have high requirements in energy efficiency, mobility, spectrum efficiency, data rate and traffic capacity. Use cases of class *Massive Machine Type Communications* have high requirements in connection density and energy efficiency. Finally, use cases of class *Ultra-reliable and Low Latency Communications* have high requirements in latency and mobility.

To meet these requirements, 5G network architectures need to be adapted. It is based on the use of SDN and NFV technologies. It includes multi-antenna transmission/reception technology, advanced inter-node coordination schemes and multi-hop technologies. 5G also uses a wide variety of frequency bands that vary to carefully addresses the needs of each use cases scenario.

As shown in Figure 0.2.6, the Radio Access Network (RAN) is divided and virtualised into server-based Distributed Units (DUs) and then centralised into server-based Centralised Units (CUs), reducing the proprietary hardware to Remote Radio Units (RRUs).

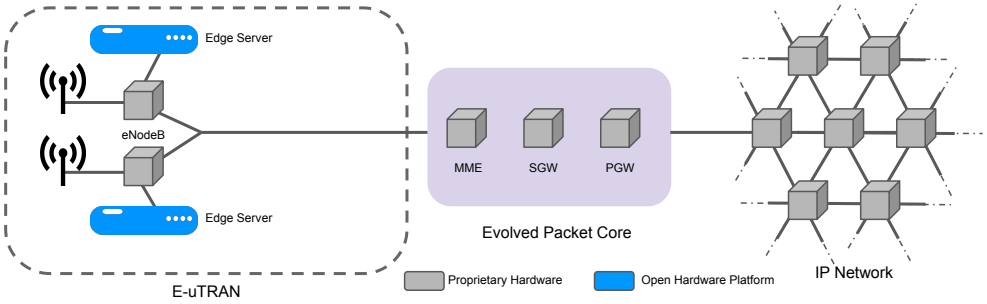
- **The RRU** handles parts of the PHY layer, analog to digital conversion, filtering, power amplification as well as the digital beamforming functionality.
- **The Distributed Unit (DU)** is close to the RRU and runs the Radio Link Control, MAC, and parts of the PHY layer. It provides digital processing, including signal modulation, encoding and scheduling. It is a logical node that includes a subset of the gNB (5G base station) functions, depending on the functional split option. Its operations are controlled by the CU.
- **The Centralized Unit (CU)** is a logical node that provides support for higher layers of the protocol stack. It includes the gNodeB functions like Transfer of user data, mobility control, radio access network sharing, positioning, session management, with the exception of the functions that are allocated exclusively to the DU. The CU controls the operation of several DUs.

This centralized deployment makes load-balancing between different RRUs possible. That is why, in most cases, the DU will be collocated with RRUs on-sit. Intelligent Edge servers will support real-time applications with computing and AI inferencing. 4G Evolved Packet Core (EPC) functions will be replaced by 5G Core components running as virtualised network functions in data centers and the cloud. 5G infrastructure will enable the interconnectivity among the different emerging technologies like Massive MIMO network, Cognitive Radio network, and mobile and static small-cell networks.

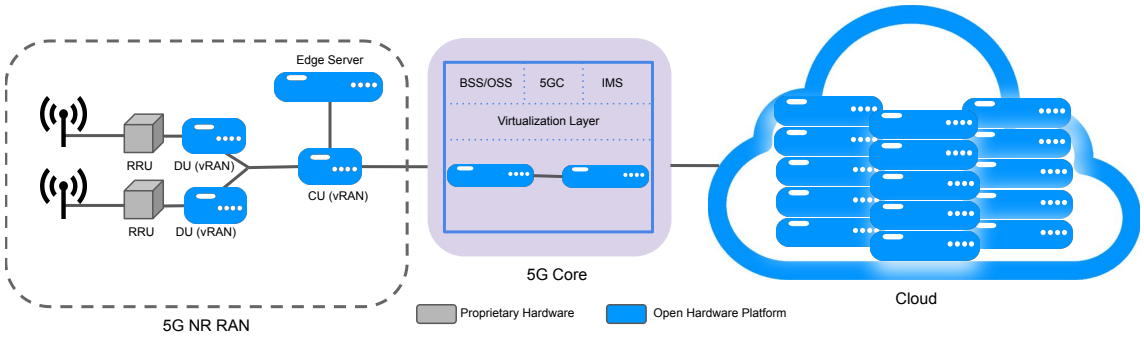
0.2.3.1 Network Slicing

As mentioned above, 5G will bring many improvements such as very high speeds and very low latency, but above all it promises to meet many new QoS requirements for a wide range of new heterogeneous use cases. These requirements can vary greatly depending on the use case. To cope with this diversity, the traditional monolithic network is inadequate. Through the joint use of SDN and NFV, a multitude of independent virtual networks can be abstracted from a single network. Each of these networks is specialised to meet the specific requirements of an end-to-end service. This idea of splitting the network into multiple independent networks is a new paradigm called network slicing [01118]. Its purpose is to logically adapt the management of network infrastructure and resources to meet the promise of 5G.

An example of the network slicing architecture can be seen on Figure 0.2.7. The network infrastructure is on the last layer and covers the access network, the transport network and the core network. In this example there are three slices, each of a different class and with different needs. Each slice has several entry points on the access network thanks to the Wireless Local Area Network (WLAN) infrastructure. Then every slices passe through network functions on edge



(a) 4G LTE Network



(b) 5G Network

Figure 0.2.6 – Architecture differences between 4G LTE and 5G

clouds in the transport network . Finally they connect to datacenters in the core network. Some parts of the network infrastructure are used by only one slice and others are shared by several.

Many proposals exist on what a network slice is, several organisations such as ETSI [01217], 3GPP, ONF, NGMN [All15] or 5G-PPP [PPP20] each have their own definition. To make it simple we can describe a slice as an independent virtual network composed of a set of network functions combined together to fulfil the requirements of a specific use case with resource guarantees or guaranteed service level. The use of both dedicated infrastructure resources for certain slices, as well as the use of shared infrastructure resources and functions between multiple slices is needed. Each slice must be seen by its users as an independent network. The purpose of a 5G slice is to provide only the traffic throughput required for the use case, and to avoid any other unnecessary functionality. Although some definitions of network slicing differ in certain aspects, some of the required principles are recurrent [ATS+18]:

- The creation of a slice can be done on demand, so their instantiation must be **managed automatically** without the need of human intervention. Each slice request has constraint requirements such as user positions (sources), bandwidth, maximum latency, network service or different network functions required, the target service, the lifetime of the slice, etc.

- During the use of a slice, some needs may change and a slice must be **elastic** and must provide the possibility to change according to these needs. In some cases, a slice may be sized according to the number of users and its main usage. This enables great flexibility, but this flexibility must not lead to a degradation of the QoS of the other slices using shared resources. This required elasticity may include changes in user location (mobility), changes in latency and changes in capacity. To meet these demands a slice must be able to modify its routing and change the allocation of its VNFs, by migrating them or allocating more resources.

- Coexisting on the same physical network, each slice must be isolated from the others. This **isolation** must provide a certain level of security and above all never violate the required Service-Level Agreement (SLA). The use of certain resources by one slice must not have an impact on the others. Because of the elasticity of the slices, it can be hard to maintain this isolation. Indeed, if two slices share the same resource and each of them requires an increase in traffic, security measures must be established to ensure that the slices do not suffer any degradation of the QoS.

- A slice must be **programmable and customizable**. Via APIs, the slice tenant (referred to as the customer paying for the slice and using it as his own virtual network) must be able to request changes to the slice's requirements. He must have access to statistics on its use and be able to manage the slice in a transparent manner, as if it was his own physical network.

- A slice must provide an **end-to-end service** and may therefore include part of the RAN [KN17], the transport network and the core network in order to reach the cloud. It must chain the relevant network functions, assigns the relevant performance configurations, and finally maps all of this onto the infrastructure resources.

In the literature a slice can be seen as a virtual network, in which case the slice allocation problem would be a Virtual Network Embedding problem. This problem involves allocating VNFs on the network and routing traffic between them but there is no order between the VNFs

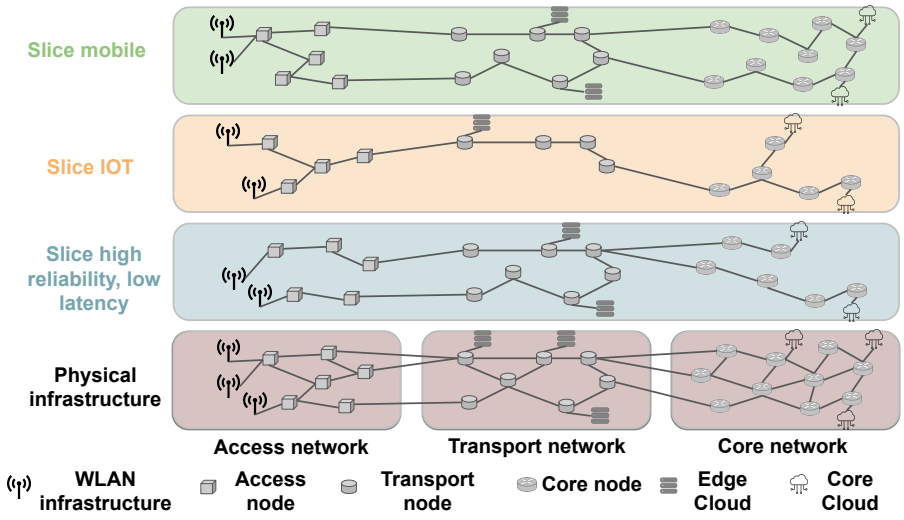


Figure 0.2.7 – Network Slicing Architecture

and the network created does not necessarily respond to an End to End service. Due to their requirements, the majority of slice implementations in the literature use one or a set of SFCs, as in [SZGS+18, TAM19, ZLF+17]. Similarly we use sets of SFCs to model slices in chapter 4 and chapter 5.

As said before, even if 5G is full of promise, use cases do not need the full potential of 5G at the same time. Figure 0.2.5(b) shows us that use case requirements can be split into three classes, and since slices are made to address the use case, there are mainly three classes of slices as recalled in [BAMH20].

- **Enhanced Mobile Broadband (eMBB)** slices are those that meet the most common needs of the general public, they must support stable connections with very high peak data rates, as well as moderate data rates for users at the cell edge. The needs addressed range from video-on-demand to social networking and general internet surfing.
- **Ultra-Reliable and Low-Latency Communications (uRLLC)** slices will open up scenarios with very low latency and/or security needs, such as managing medical emergencies or performing remote operations, vehicle-to-vehicle communications to avoid accidents or virtual reality video games.
- **Massive Machine Type Communications (mMTC)** slices support a very large number of Internet of Things (IoT) devices, which are only intermittently active and send very little data. In the usage scenario, we will have a lot of sensors that have a very limited battery but consume very little power.

Because of the innovations they bring as well as the heterogeneity of their use cases and the multiple solutions they offer, network slicing and 5G in general are raising many new **challenges**.

Allocation: Network slice allocation is the first problem encountered. In [SZGS+18], to find a slice allocation the authors compute the k shortest paths offline between each base station and

each centralized unit. Then online, in two phases, they reserve resources and choose paths to maximise an operator's revenue by overbooking (subsection 0.2.4). Each path is characterised by a delay in order to respect the delay constraint of the incoming slices. Their solution does not only focus on allocation (subsection 0.2.4). In [TAM19], the authors use a MILP that, given a set of SFC requests, finds an optimal VNF placement and routing and wavelength assignment. Their allocation is offline and they aim to maximise the number of successfully routed SFCs. Their allocation is then used by a reinforcement learning algorithm to perform online reconfiguration (subsection 0.2.4). In [ZLF+17] the authors propose efficient penalty successive upper bound minimisation and rounding algorithms for slice allocation, as well as four heuristics for fast allocation in dynamic scenario but they don't take into account any users QoS requirements. Their heuristics give solutions with resource violations. One of their algorithms has a good balance between solution quality and execution speed.

Isolation: The second problem that can be mentioned is the problem of the slices isolation, because goal of network slicing is to have multiple virtual networks sharing some resources and having different needs, on the same infrastructure. Due to the slice elasticity, isolation can become very challenging if we don't want to degrade the QoS. Huin *et al.* [HMM+19] choose to physically isolate the slices to ensure that there is no interference between them. They rely on Flex Ethernet technology, which allows different slices to be isolated by allocating resources in the manner of Time-division multiple access. In [MF19], the authors propose a two-layer scheduler to reduce the complexity of a single layer in the context of RAN slicing. The first layer is an inter-slice scheduler that determines the amount of resources for each slice. The second layer is an intra-slice scheduler and allocates resources to end users. Their approach allows to obtain different trade-offs between isolation and efficiency depending on the chosen parameters. They compare their methods to other state-of-the-art techniques to show the flexibility of their approach. In [YLW+19], the authors demonstrate that the use of a user admission control mechanism ensures isolation in a dynamic network slicing scenario exploiting the capacity advantages of dynamic network slicing. For an overview of slice isolation, in [KNS+18], the authors discuss the different types of isolation as well as the isolation parameters. An important aspect of their discussion is the security achieved by inter-slice communication and they propose a set of challenges to realize end-to-end user's security based on slices isolation.

In this thesis, the isolation of slices is ensured because a slice is seen as a pipe with a reserved capacity. This method does not allow for overbooking by allocating less capacity than the tenant has requested when the slice is not used much. But by doing this we ensure that the SLA will always be respected, as well as the isolation of the slices.

Elasticity: Finally, we can be interested in the slices elasticity. The elasticity of a slice can be seen in two ways. Either the traffic of the slice is modified dynamically without modifying the quantity of resources payed by the tenant (to follow the dynamics of the traffic). Or it can be seen as a scaling of the slice: the tenant wants to modify the reserved capacities. The second is intrinsically linked to the scaling reconfiguration that we will discuss next. The first, on the other hand, must be taken into account only if the operator does not reserve the capacities requested by the tenant. Indeed, if the operator reserves less capacity during a period of low load on a slice to do overbooking, then he must be prepared to manage an increase in its load if he does not want to break the isolation. Multiple solutions are applicable, whether for the scaling of slices or their migration.

Wang *et al.* [WFQ+19] propose a hybrid slice reconfiguration mechanism that manages two types of reconfiguration. A first type of reconfiguration is made to adapt the slices to the current traffic and a second one is made to modify the flows traversing the slices. The first reconfiguration, which they call DSR, is done so that the capacity reserved for the slice is as close as possible to what it actually consumes, in order to maximise the operator's profit.

In [ZZC20] the authors use an algorithm to predict the traffic demanded by each slice at a time t. The prediction is used by an adaptive VNF scaling strategy that determines the number of VNFs and the network resources to be installed. The aim is to deploy slices with the lowest energy consumption costs. However, once a slice is deployed its resource consumption does not change.

The elasticity of slices is not taken into account in this thesis. When a slice is allocated, its resources are reserved. A change in slice traffic therefore does not need to be taken into account as the reserved capacity will not change. As mentioned for isolation, this prevents overbooking but ensures that the SLA is met. As for scaling, we have not studied it in our work. We could partially modify our simulations to decrease or increase the capacity of the slices at certain times. But this method would not take into account some challenges related to scaling. It could be interesting to add a scaling function to our reconfiguration, inspired by the work mentioned above.

0.2.4 Reconfiguration

To avoid the saturation that 5G could bring and to be able to manage ever larger networks without sacrificing performance and wasting resources, we need ever more effective optimisation techniques. These techniques must be adjusted to the problem they address and to the context to be as efficient as possible, but they can be adapted from existing methods.

In the scope of this thesis, a network configuration can be defined as the set of resources used, both the VNFs deployed and the links used and how they are used. A configuration takes into account how each resource is used by each slice or demand. To define it simply, a reconfiguration is a change in the configuration of the network. A reconfiguration neither adds nor removes demands. Reconfiguration is used to adapt the utilisation of network resources to the traffic and can have several objectives such as reducing the Opex or improving the QoS.

In a traditional network, reconfiguring can lead to a modification of the network infrastructure, because if a network function needs to be moved, it means moving the hardware, which takes time, possibly measured in days. Reconfiguration in this case is therefore not necessarily adapted to accommodate the network dynamically to traffic, but rather to accommodate the network in the long term. The use of SDN and NFV gives reconfiguration its full potential in our context and makes it possible to avoid modifying the infrastructure. The network can therefore be reconfigured to adapt to the dynamics of the traffic. Reconfiguration can be performed at several moments in time. It can be done as soon as a new request arrives [GR18], when a request is rejected [TTG13], when the physical network is modified [CLX+10], or it could also be done periodically while the network is not yet saturated.

0.2.4.1 Different types of reconfigurations

Multiple types of reconfiguration can be done within a network in our context, not necessarily involving the same resources.

- The first type of reconfiguration is **flow re-routing**. With this type of reconfiguration, a portion of the traffic is changed without impacting the network functions. If a request has

to go through specific network functions, its traffic can be rerouted in-between but the functions cannot migrated. Wang *et al.* [WFQ+19], develop an algorithm that handles two types of reconfiguration to maximise the operator's profits. A first one that modifies the routing of the flow inside the slices, and a second one that we will mention later. In this context, slices have a fractional allocation. Their algorithm reduces the number of reconfigured flows by over 85% compared to their baseline. This type of reconfiguration is rarely done alone in a SDN-NFV network in the literature, as performing migration of VNFs together with re-routing of flows lead to a better improvement of the utilization of the network resources.

- The second type of reconfiguration is the **VNFs migration within a datacenter**. Cho *et al.* [CTZB17] present an algorithm to migrate VNFs within VMs to minimise latency. As they mention, if two VNFs used one after the other are on the same VM on the same node, then it removes the communication delay between them. In [LZC+16], the authors present two algorithms for migration and consolidation of virtual machines within a datacenter. Their objective is to minimise energy consumption by doing as few migrations as possible. The authors point out that each node within a datacenter can consume up to 70% of its maximum energy consumption when it is only slightly used: the migration of VMs and their consolidation therefore provide the opportunity to shut down nodes and improve the energy efficiency of the data center.

- The reconfiguration can also take the form of a **scaling**. When allocating a virtual network, SFC or slice, the allocated capacities can be seen as fixed: the customer pays for a maximum bandwidth. But a slice may also need to be scaled. A tenant may ask for an increase in resources during high traffic periods. Alternatively, he may ask for a decrease in the capacity of his slice. Scaling and elasticity are also often linked. In many works, the maximum capacity of a slice is not reserved, only what it consumes at a given time. This allows to minimise the cost and to accept more demand. However, when traffic increases, it is necessary to be able to handle the traffic for which the tenant has paid without compromising the other slices using the same resources. In this case, a scaling reconfiguration can be used. Scaling allows to modify the allocated capacities to be always close to the current traffic or to foresee the traffic increase or decrease by scaling-up or down flow capacities.

  In [SGT20] the authors propose a dynamic slicing algorithm, which manages the placement of VNFs on the network. They also propose a stochastic optimisation formulation, which handles the uncertainty of service demands. For them a slice is not made for a tenant but for a specific service. If several requests are made for the same service their algorithm will try to place them in the same slice to maximise its utility. They try to minimise the number of slices by deleting, adding or scaling existing slices to fit the current traffic. For each slice their objective is to maximise its utility by not oversizing it. Their view of slices does not allow them to compare themselves to the literature. They test their algorithm with several parameters to observe its behaviour and confirm that it adapts correctly to the traffic.

  Salvat *et al.* [SZGS+18] use machine learning to overbook slices based on the prediction of resource usage in order to maximise the operator's revenue. At each time step the algorithm runs to scale the slices according to the given prediction. They use a Benders decomposition (also called row generation) for small and medium size instances. They also use a heuristic for large instances. They manage to get up to three times more profit with their concept and test it experimentally.

As mentioned earlier, in [WFQ+19] the authors develop an algorithm that handles two types of reconfiguration to maximise the operator's profits. Firstly, a scaling reconfiguration to adapt the slices to the current traffic. Secondly, a reconfiguration that modifies the routing of the slices. Their first algorithm reduces the number of reconfigurations performed by 5% compared to their baseline.

Ayoubi *et al.* [AZA16] propose a reliable embedding and reconfiguration algorithm for elastic services in failure-prone datacenter networks. They work on Virtual Network scale-up requests, such as increasing resource demands, adding new network components and/or upgrading the class of service. Their objective is to provide the greatest improvement in availability. Their algorithm, unlike existing work, promotes better resource utilisation as it avoids availability over-provisioning. They also propose an availability-aware reconfiguration module for elastic services that enables low-cost reconfiguration with minimal service disruption.

- The **VNF migration between datacenters** is also a type of reconfiguration. In this case it may be associated with re-routing of flows to route traffic between the new VNFs' positions.

  In [EMAL17] the authors study the problem of VNF migrations in a dynamic scenario with the aim of minimising the network operating cost which is the sum of the energy consumption costs and the loss of revenue due to the loss of data during downtime. They propose a heuristic based on the markov decision process theory. In the worst case they do not exceed 20% difference with the optimal and compared with a simple state of the art policy they improve the results by 27%. Ghaznavi *et al.* [GKS+15] propose a reconfiguration algorithm for VNF consolidation that optimises the placement of those functions in response to the workload on demand. Their solution accepts almost twice the workload in comparison to first-fit and random placements. As for the operational cost, their solution reduces it by %5 to 8%.

  As mentioned earlier, Troia *et al.* [TAM19] use a MILP to calculate the allocation of a set of SFCs. They use reinforcement learning to reconfigure the SFCs placed on the network by migrating VNFs and rerouting traffic. Their objective is to accept as many demands as possible while minimising the reconfiguration costs. Their algorithm learns how and when to reconfigure demands in order to route traffic requests with a lower blocking probability. It is also able to predict sudden changes in traffic patterns and trigger the reconfiguration of SFCs in advance.

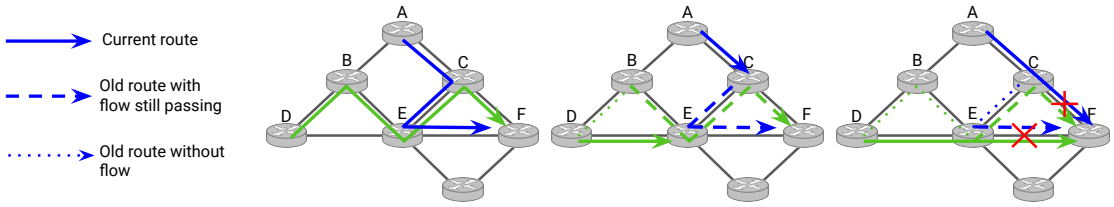
  Pozza *et al.* [PNL+20] propose to find the steps of a reconfiguration in which VNFs and routes are modified taking into account capacities and delays, based on an initial allocation of slices and a final pre-computed allocation. There are few papers dealing with multi-step reconfigurations and their method allows to calculate these steps quickly. Compared to our work in chapter 3, chapter 4 and chapter 5 they do not propose a method to compute the final allocation used for reconfiguration and their reconfiguration does not prevent from a degradation of the QoS. This type of reconfiguration is the one that will be developed in this thesis. Indeed, it allows to process requests from end to end with a global view of the network (with the exception of RAN which is outside the scope of this thesis). It also allows to control the processing of requests from the edge, through the transport network, to the core network, thus giving a large potential for management improvement.

0.2.4.2 Reconfiguration strategies

To recall, reconfiguring generally aims at maximising revenue by accepting more demands and/or at reducing costs by minimising the resources used. But there are also **reconfiguration costs** that have to be considered by the network operators. There are management costs but also disruption costs as during a reconfiguration, data losses or degradation of QoS can occur for some. The majority of the related work take into account the cost of reconfigurations. In [KBB19], there are migration costs related to the volume of data transmitted during the migration. In [WFQ+19], the cost of a reconfiguration is defined by its resource consumption during signalling and retransmissions. In [EMAL17], the loss of revenue caused by a reconfiguration is due to the loss of data during downtime. In [AZA16], they seek to minimise the overall cost of reconfiguration, which reflects the amount of resources used as well as any service disruption/downtime.

Different reconfiguration methods can be implemented for migration of rerouting of traffic. **Break-Before-Make** approach does not provide control over the possible cohabitation of the different traffic during the reconfiguration. The new routes are defined, the functions are migrated and then the traffic rerouted. A reconfiguration may not be perfectly synchronised and some migrations may take longer than others. As a result, new traffic sent may collide with old traffic that should no longer exist when crossing the same route.

In case of a collision, if a link is saturated, the router will queue packets, resulting in increased latency. But when operating on a core network with links with bandwidth greater than 1 gigabit/second, the router's buffers fill up quickly, which may also imply packet losses.



(a) Legend (b) Before the reconfiguration (c) After 1ms (d) Collisions

Figure 0.2.8 – An example of a failed *break-before-make* reconfiguration.

Example Figure 0.2.8 illustrates an example for the reconfiguration of two requests using a *break-before-make* process. In this example, there are no VNF for simplicity. Each link can only support one flow per direction and all links have the same latency (1ms). The legend (a) shows the three types of flow. The current route is the route the flow should take when it is sent. Old routes are routes that are no longer identified by the SDN controller as being in use, however either packets are still in transit on them or no traffic is present. In (b) the green and blue flows are not optimal. The green flow going from D to F uses four links instead of two and the blue flow going from A to F uses three links instead of two. To reconfigure them, paths need to change: the green flow now goes from D to E to F and the blue flow from A to C to F. In (c), routing on the old paths is stopped and routing on the new ones begins. The old flows do not disappear until their packets in transit reach their destination. Because of the latency, the old green traffic and the new one arrive at node E approximately at the same time. In (d) there are two collisions, the new green

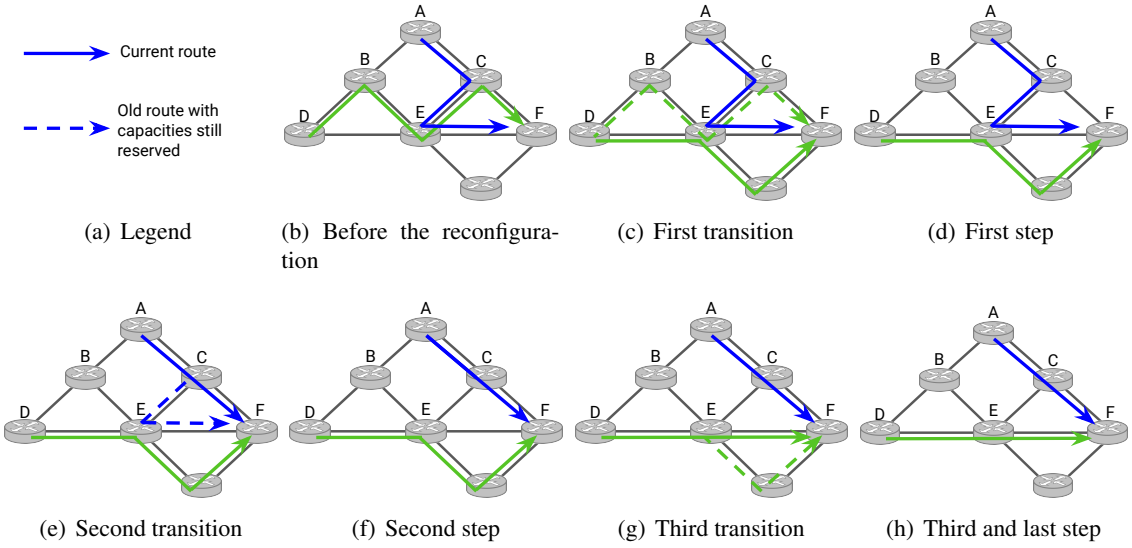
flow and the old blue flow try to use the E-F link at the same time. The same happens with the new blue flow and the old green flow on C-F. These collisions may lead to packet loss and therefore QoS degradation.

Even reconfiguring requests one by one (which may not be very efficient on a large network) may lead to these collisions. As Foerster *et al.* [FVW19] show, when reconfiguring a request, in each part of the network where the old and new routes cross (this may involve several links/nodes or just the destination node) there is a risk of a collision due to the latency differences on the old and new routes. If the traffic goes faster on the new route it may catch up with the old traffic and therefore collide.

To avoid these collisions, one strategy would be to wait for all the old packets to arrive at their destination, however this implies an interruption of the transmission and thus a degradation of the QoS. Most work on reconfiguration takes into account these packet losses and latency increases, which is why they put a cost on reconfiguration. Taking into account the high QoS requirements of 5G and network slicing, an important question is whether putting a price on reconfiguration is a good idea or whether these QoS degradations should be avoided instead.

The last reconfiguration method that will be addressed in this thesis is based on **Make-Before-Break** approach. Make-Before-Break comes from a spectral defragmentation mechanism in optical networks [WM13, DJCA18]. In an optical network, a traffic flow from a source to a destination keeps the same wavelength along its path. The same wavelength cannot be used twice at the same time on the same fiber. Flows sharing part of their path must have a different wavelength, and the number of wavelengths available is limited. The dynamicity of the demands will over time create fragmentation: "holes" in the network where certain wavelengths become unusable.

As the name suggests the new reconfigured route will be created before the old one is broken. To be more precise, the reconfiguration is done following several steps: (i) First, the reconfiguration to be carried out is computed by the controller. (ii) Second, the new needed VNF instances (if any) are deployed. (iii) Third, rerouted flows are sent towards their new routes in which we know the needed capacity is available (the capacities on the old route are still reserved) . (iv) Finally, the flows are no longer sent to their old routes, the routing entries corresponding to the old flow are removed and therefore capacities are no longer reserved on the old route. When a request is reconfigured, its two routes (the old and the new one) coexist, the traffic capacity is reserved on each of them: this reconfiguration consequently implies a stronger constraint in terms of capacity and may therefore require more steps to approach an optimal allocation.

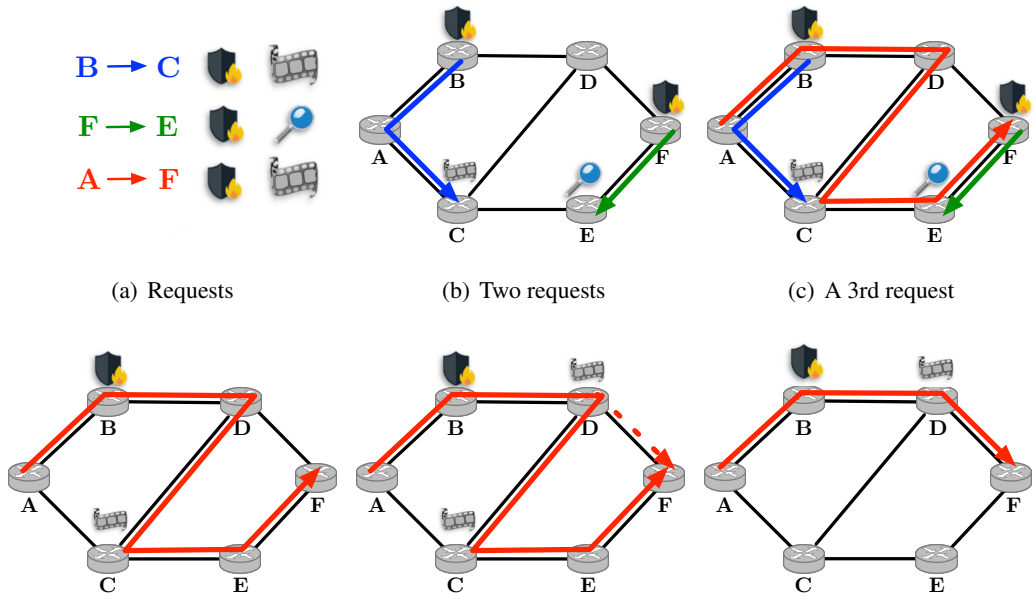


(a) Legend (b) Before the reconfiguration (c) First transition (d) First step

(e) Second transition (f) Second step (g) Third transition (h) Third and last step

Figure 0.2.9 – An example of *make-before-break* reconfiguration.

Example. Figure 0.2.9 shows the same scenario as in Figure 0.2.8 but with make-before-break reconfiguration. The legend (a) shows the two types of flow. The current route is the route initially followed by the flow. The old route is the one during transition, where packets are still in transit. This route still has its capacity reserved by the controller. In (b), the green and blue flows are not optimal. The green flow going from D to F uses four links instead of two and the blue flow going from A to F uses three links instead of two. To reconfigure them, paths need to change: the green flow now goes from D to E to F and the blue flow from A to C to F. (c) represents the first transition (between (b) and (d)), both old and new routes are active. Only the green demand is being reconfigured in a temporary route. The green flow cannot be directly in the final route oh (h) as there is not enough capacity on link EF. In (d) the transition is finished and all the packets of the old route have been forwarded. The capacities of the old routes are freed by the controller. (e) represents the second transition (between (d) and (f)), the blue demand is being reconfigured. (f) is the second step of the reconfiguration, the blue demand is in the optimal state. In (g) the green demand is being reconfigured in its optimal state, both old and new routes are active. Finally in (h) both flows take their optimal routes. The reconfiguration was computed in 3 steps (we don't count the transition steps). This example shows that the make-before-break reconfiguration is more constraining to implement than the break-before-make because of the capacities reserved during the transition phases. Nevertheless, it may allow to find ways to reconfigure in several steps.



(a) Requests (b) Two requests (c) A 3rd request

(d) First two requests leave (e) Reconfiguration phase (f) Optimal routing for the third request

Figure 0.2.10 – An example of the reconfiguration of a request using a *make-before-break* approach with one step.

Example. Figure 0.2.10 illustrates an example for the reconfiguration of a request using a *make-before-break* process. In addition to the traffic reconfiguration, this example shows the reconfiguration of the VNFs and the congestion caused by the dynamicity of the requests. When the request from A to F arrives in step (c), two requests have already been routed during step (b). To avoid the cost of installing new VNFs, the route from A to F with minimum cost is a long 5–hops route (step (c)). When requests from B to C and from F to E leave (step (d)), the request is routed on a non-optimal path which uses more resources than necessary. We reroute the request from A to F to one optimal 3-hops path (step (f)) with an intermediate make-before-break step (step (e)) in which both routes co-exist. Note that the VNF in C is removed, while a new one is installed in D. At the end of the reconfiguration, the red application uses fewer links on the network (which can be a goal when reconfiguring).

This type of reconfiguration can be run with several requests at the same time and over several steps. Thanks to the security it induces in terms of packet loss and latency, this technique is interesting to use on a 5G network with network slicing in consideration of the high QoS requirements.

0.3 Thesis plan and Contributions

In this section, I present the plan of the thesis by summarising the contributions, and listing all the published works and submitted ones.

In chapter 1, I introduce different optimisation tools and techniques that are used in this thesis. Firstly, I present an introduction to linear programming, secondly I develop the concept of column generation based on linear programming, and thirdly I introduce reinforcement learning.

The chapter 2 focuses on the SFC placement problem. The static placement problem of a set of SFCs and the dynamic placement problem of an SFC are studied. I describe the data structure called a layer graph that allows the routing and allocation problem to be transformed into a routing problem. This structure is used to formulate an ILP and a heuristic. The ILP is the model used throughout the thesis for the placement problem, and it is compared to an ILP from the literature.

The following chapters of the thesis focuse on the reconfiguration problem. The chapter 3 studies the make-before-break reconfiguration of SFCs using fractional flows in static and dynamic environments. I present an ILP that computes the best possible allocation by reconfiguring a set of requests with a given number of steps. This ILP also computes a reconfiguration that enables the allocation of an SFC that could not be placed. Then, I present a heuristic that computes a reconfiguration to get as close as possible to an optimal allocation. Finally, my solutions are compared to no-reconfiguration and to break-before-make reconfiguration solutions.

The chapter 4 develops the reconfiguration of the previous chapter in the context of network slicing. Other parameters such as daily dynamicity, delay, request design and the management of a much larger number of requests have to be taken into account. Therefore, the modelisation has to be adapted in order to be consistent with these new constraints. These modifications cannot be handled by common ILP due to the huge number of constraints, therefore, I set up a column generation algorithm with two different pricing problems (ILP and LP based approach). This modelisation gives the ability to manage the reconfiguration of a large number of slices in less than one minute and can take advantage of a parallelisation potential. We compare our new solutions against an adapted version of our previous ILP as well as against a non-reconfiguring solution and one using break-before-make reconfiguration. We show that our solution can handle a much larger number of demands while significantly improving costs compared to a solution without reconfiguration.

Finally, chapter 5 proposes an intelligent management of reconfigurations by choosing the best moment during the day to perform reconfigurations. By developing a deep reinforcement learning agent, I show how to reduce the number of reconfigurations within a day by choosing at which time to reconfigure, without reducing the improvement given by a fixed frequency reconfiguration method. This makes the network management easier and more programmable. The agent is adaptable to the dynamics of the day, and changes the frequency of reconfigurations in response to the number of slices on the network and their arrival/departure.

0.3.1 List of Publications

International Journals:

[GGJM22] **Adrien Gausseran**, Frédéric Giroire, Brigitte Jaumard and Joanna Moulierac, *"Be Scalable and Rescue My Slices During Reconfiguration"* The Computer Journal, Volume 65, 2022.

[GTGM21] **Adrien Gausseran**, Andrea Tomassilli, Frédéric Giroire, Joanna Moulierac, *"Don't interrupt me when you reconfigure my Service Function Chains"*, Computer Communications, Volume 171, 2021, Pages 39-53, ISSN 0140-3664, https://doi.org/10.1016/j.comcom.2021.02.008.

International Conferences:

[GGJM20] **Adrien Gausseran**, Frédéric Giroire, Brigitte Jaumard and Joanna Moulierac, *"Be Scalable and Rescue My Slices During Reconfiguration"* ICC 2020 - 2020 IEEE International Conference on Communications (ICC), 2020, pp. 1-6, doi: 10.1109/ICC40277.2020.9148871.

[GTGM19a] **Adrien Gausseran**, Andrea Tomassilli, Frédéric Giroire and Joanna Moulierac, *"No Interruption When Reconfiguring my SFCs"* 2019 IEEE 8th International Conference on Cloud Networking (CloudNet), 2019, pp. 1-6, doi: 10.1109/CloudNet47604.2019.9064115.

[GTGM19b] **Adrien Gausseran**, Andrea Tomassilli, Frédéric Giroire and Joanna Moulierac, *"Poster: Don't interrupt me when you reconfigure my service function chains"* 2019 IFIP Networking Conference (IFIP Networking), 2019, pp. 1-2, doi: 10.23919/IFIPNetworking46909.2019.8999470.

Submitted to an International Conference:

[GAL+22] **Adrien Gausseran**, Redha A. Alliche, Hicham Lesfari, Ramon Aparicio-Pardo, Frédéric Giroire and Joanna Moulierac, *"When to Reconfigure my Network Slices? A Deep Reinforcement Learning Approach"* ICC 2022 - 2022 IEEE International Conference on Communications (ICC), 2022, pp. 1-6.

National Conferences:

[GTGM19c] **Adrien Gausseran**, Andrea Tomassilli, Frédéric Giroire, Joanna Moulierac. *"Reconfiguration de chaînes de fonctions de services sans interruption"*. CORES 2019 - Rencontres Francophones sur la Conception de Protocoles, l'Évaluation de Performance et l'Expérimentation des Réseaux de Communication, Juin 2019, Saint Laurent de la Cabrerisse, France. (hal-02118989).

Research Reports:

[GGJM19] **Adrien Gausseran**, Frédéric Giroire, Brigitte Jaumard, Joanna Moulierac. *"Be Scalable and Rescue My Slices During Reconfiguration"*. [Research Report] Inria - Sophia Antipolis; I3S, Université Côte d'Azur; Concordia University. 2019. (hal-02416096).

[GTGM18] **Adrien Gausseran**, Andrea Tomassilli, Frédéric Giroire, Joanna Moulierac. *"Don't Interrupt Me When You Reconfigure my Service Function Chains"*. [Research Report] RR-9241, UCA, Inria; Université de Nice Sophia-Antipolis (UNS); CNRS; UCA,I3S. 2018. (hal-01963270v).

References

[00114] ETSI GS NFV-MAN 001. Etsi gs nfv-man 001 v1.1.1 (2014-12)network functions virtualisation (nfv);management and orchestration. https://www.etsi.org/deliver/etsi_gs/nfv-man/001_099/001/01.01.01_60/gs_nfv-man001v010101p.pdf, 12 2014.

[01017] ETSI GR NFV-EVE 010. Network functions virtualisation(nfv) release 3; licensing management; report on license management for nfv. https://www.etsi.org/deliver/etsi_gr/NFV-EVE/001_099/010/03.01.01_60/gr_nfv-eve010v030101p.pdf, 12 2017.

[01118] ETSI GR NGP 011. Next generation protocols(ngp);e2e network slicing reference framework and information model. https://www.etsi.org/deliver/etsi_gr/NGP/001_099/011/01.01.01_60/gr_ngp011v010101p.pdf, 09 2018.

[01217] ETSI GR NFV-EVE 012. Network functions virtualisation (nfv) release 3;evolution and ecosystem; report on network slicing support with etsi nfv architecture framework. https://www.etsi.org/deliver/etsi_gr/NFV-EVE/001_099/012/03.01.01_60/gr_NFV-EVE012v030101p.pdf, 12 2017.

[22.16] 3GPP TR 22.891. Study on new services and markets technology enablers. https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2897, 9 2016.

[38.20] 3GPP TR 38.913. Study on scenarios and requirements for next generation access technologies. https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2996, 7 2020.

[ACF+17] Ahmed Abdelsalam, Francois Clad, Clarence Filsfils, Stefano Salsano, Giuseppe Siracusano, and Luca Veltri. Implementation of virtual network function chaining through segment routing in a linux-based nfv infrastructure. In *2017 IEEE Conference on Network Softwarization (NetSoft)*, pages 1–5, 2017.

[All15] The Next Generation Mobile Networks Alliance. 5g white paper. https://www.ngmn.org/wp-content/uploads/NGMN_5G_White_Paper_V1_0.pdf, 02 2015.

[ARS16] Mamta Agiwal, Abhishek Roy, and Navrati Saxena. Next generation 5g wireless networks: A comprehensive survey. *IEEE Communications Surveys Tutorials*, 18(3):1617–1655, 2016.

[ATS+18] Ibrahim Afolabi, Tarik Taleb, Konstantinos Samdanis, Adlen Ksentini, and Hannu Flinck. Network slicing and softwarization: A survey on principles, enabling technologies, and solutions. *IEEE Communications Surveys Tutorials*, 20(3):2429–2453, 2018.

[AZA16] Sara Ayoubi, Yanhong Zhang, and Chadi Assi. A reliable embedding framework for elastic virtualized services in the cloud. *IEEE Transactions on Network and Service Management (IEEE TNSM)*, 13(3):489–503, 2016.

[BAMH20] Alcardo Alex Barakabitze, Arslan Ahmad, Rashid Mijumbi, and Andrew Hines. 5g network slicing using sdn and nfv: A survey of taxonomy, architectures and future challenges. *Computer Networks*, 167:106984, 2020.

[Bel58] Richard Bellman. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

[BJSE16] Deval Bh, Raj Jain, Mohammed Samaka, and Aiman Erbad. A survey on service function chaining. *Journal of Network and Computer Applications*, 75, 09 2016.

[BRC+13] Md. Faizul Bari, Arup Raton Roy, Shihabur Chowdhury, Qi Zhang, Mohamed Faten Zhani, Reaz Ahmed, and R. Boutaba. Dynamic controller provisioning in software defined networks. 10 2013.

[CB02] B. Carpenter and Scott Brim. Middleboxes: taxonomy and issues. 01 2002.

[CLX+10] Zhiping Cai, Fang Liu, Nong Xiao, Qiang Liu, and Zhiying Wang. Virtual network embedding for evolving networks. In *IEEE Global Telecommunications Conference - GLOBECOM*, pages 1–5. IEEE, 2010.

[CTZB17] Daewoong Cho, Javid Taheri, Albert Y. Zomaya, and Pascal Bouvry. Real-time virtual network function (vnf) migration toward low network latency in cloud environments. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pages 798–801, 2017.

[DGT+19] Tilemachos Doukoglou, Velissarios Gezerlis, Konstantinos Trichias, Nikos Kostopoulos, Nikos Vrakas, Marios Bougioukos, and Rodolphe Legouable. Vertical industries requirements analysis targeted kpis for advanced 5g trials. In *2019 European Conference on Networks and Communications (EuCNC)*, pages 95–100, 2019.

[DHM+13] Advait Dixit, Fang Hao, Sarit Mukherjee, T. Lakshman, and Ramana Kompella. Towards an elastic distributed sdn controller. volume 43, pages 7–12, 09 2013.

[DJCA18] Huy Duong, Brigitte Jaumard, David Coudert, and Ron Armolavicius. Efficient make before break capacity defragmentation. In *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*, pages 1–6, 2018.

[dT21] Fédération Française des Télécoms. Chiffres clés. https://www.fftelecoms.org/chiffres-cles/, 2021.

[EMAL17] Vincenzo Eramo, Emanuele Miucci, Mostafa Ammar, and Francesco Giacinto Lavacca. An approach for service function chain routing and virtual function network instance migration in network function virtualization architectures. *IEEE/ACM Transactions on Networking (ToN)*, 25(4):2008–2025, 2017.

[Fou15] Open Networking Foundation. Tr-518 relationship of sdn and nfv. https://opennetworking.org/wp-content/uploads/2014/10/onf2015.310_Architectural_comparison.08-2.pdf, 10 2015.

[Fou16] Open Networking Foundation. Tr-521 sdn architecture. https://opennetworking.org/wp-content/uploads/2014/10/TR-521_SDN_Architecture_issue_1.1.pdf, 2016.

[FVW19] Klaus-Tycho Foerster, Laurent Vanbever, and Roger Wattenhofer. Latency and consistent flow migration: Relax for lossless updates. In *2019 IFIP Networking Conference (IFIP Networking)*, pages 1–9, 2019.

[FZI16] Pingzhi Fan, Jing Zhao, and Chih-Lin I. 5g high mobility wireless communications: Challenges and solutions. *China Communications*, 13(Supplement2):1–13, 2016.

[GAL+22] A. Gausseran, R. Alliche, H. Lesfari, R. Aparicio-Pardo, F. Giroire, and J. Moulierac. When to reconfigure my network slices? a deep reinforcement learning approach. In *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, pages 1–6, 2022.

[GGJM19] Adrien Gausseran, Frédéric Giroire, Brigitte Jaumard, and Joanna Moulierac. Don't break network slices during reconfiguration. Technical report, Inria, Dec. 2019.

[GGJM20] A. Gausseran, F. Giroire, B. Jaumard, and J. Moulierac. Be scalable and rescue my slices during reconfiguration. In *IEEE ICC*, 2020.

[GGJM22] A. Gausseran, F. Giroire, B. Jaumard, and J. Moulierac. Be scalable and rescue my slices during reconfiguration. volume 65, 2022.

[GHM15] Frédéric Giroire, Frédéric Havet, and Joanna Moulierac. Compressing two-dimensional routing tables with order. In *7th Network Optimization Conference (INOC)*, 2015.

[GHMP18] Frédéric Giroire, Nicolas Huin, Joanna Moulierac, and Truong Khoa Phan. Energy-Aware Routing in Software-Defined Network using Compression. *The Computer Journal*, 61(10):1537–1556, 03 2018.

[GJPGA12] Aaron Gember-Jacobson, Prathmesh Prabhu, Zainab Ghadiyali, and Aditya Akella. Toward software-defined middlebox networking. pages 7–12, 10 2012.

[GKS+15] Milad Ghaznavi, Aimal Khan, Nashid Shahriar, Khalid Alsubhi, Reaz Ahmed, and Raouf Boutaba. Elastic virtual network function placement. In *IEEE International Conference on Cloud Networking (CloudNet)*, pages 255–260, 2015.

[GR18] Lingnan Gao and George N Rouskas. Virtual network reconfiguration with load balancing and migration cost considerations. In *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 2303–2311. IEEE, 2018.

[GTGM18] Adrien Gausseran, Andrea Tomassilli, Frédéric Giroire, and Joanna Moulierac. Don't Interrupt Me When You Reconfigure my Service Function Chains. Technical report, Inria, Dec. 2018.

[GTGM19a] A. Gausseran, A. Tomassilli, F. Giroire, and J. Moulierac. No interruption when reconfiguring my SFCs. In *IEEE International Conference on Cloud Networking (CloudNet)*, pages 1–6, 2019.

[GTGM19b] Adrien Gausseran, Andrea Tomassilli, Frederic Giroire, and Joanna Moulierac. Poster: Don't interrupt me when you reconfigure my service function chains. In *2019 IFIP Networking Conference (IFIP Networking)*, pages 1–2, 2019.

[GTGM19c] Adrien Gausseran, Andrea Tomassilli, Frédéric Giroire, and Joanna Moulierac. Reconfiguration de chaînes de fonctions de services sans interruption. In *CORES 2019 - Rencontres Francophones sur la Conception de Protocoles, l'Évaluation de Performance et l'Expérimentation des Réseaux de Communication*, Saint Laurent de la Cabrerisse, France, June 2019.

[GTGM21] A. Gausseran, A. Tomassilli, F. Giroire, and J. Moulierac. Don't interrupt me when you reconfigure my service function chains. *Computer Communications*, 2021.

[GZT+19] Lin Gu, Deze Zeng, Sheng Tao, Song Guo, Hai Jin, Albert Y. Zomaya, and Weihua Zhuang. Fairness-aware dynamic rate control and flow scheduling for network utility maximization in network service chain. *IEEE Journal on Selected Areas in Communications*, 37(5):1059–1071, 2019.

[HMM+19] Nicolas Huin, Paolo Medagliani, Sébastien Martin, Jérémie Leguay, Lei Shi, Shengming Cai, Jinchun Xu, and Hao Shi. Hard-isolation for network slicing. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 955–956, 2019.

[HTGJ18] Nicolas Huin, Andrea Tomassilli, Frederic Giroire, and Brigitte Jaumard. Energy-efficient service function chain provisioning. *IEEE/OSA Journal of Optical Communications and Networking*, 10(3):114–124, 2018.

[ITU18] International Telecommunication Union ITU. Itu-r m.2083; setting the scene for 5g: opportunities and challenges. http://handle.itu.int/11.1002/pub/811d7a5f-en, 2018.

[Jae15] Bernd Jaeger. Security orchestrator: Introducing a security orchestrator in the context of the etsi nfv reference architecture. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 1255–1260, 2015.

[JKM+13] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs

Hölzle, Stephen Stuart, and Amin Vahdat. B4: Experience with a globally-deployed software defined wan. *SIGCOMM Comput. Commun. Rev.*, 43(4):3–14, August 2013.

[KBB19] Nguyen Tuan Khai, Andreas Baumgartner, and Thomas Bauschert. Optimising virtual network functions migrations: A flexible multi-step approach. In *2019 IEEE Conference on Network Softwarization (NetSoft)*, pages 188–192, 2019.

[KBK+12] James Kempf, Elisa Bellagamba, András Kern, Dávid Jocha, Attila Takacs, and Pontus Sköldström. Scalable fault management for openflow. In *2012 IEEE International Conference on Communications (ICC)*, pages 6606–6610, 2012.

[KF13] Hyojoon Kim and Nick Feamster. Improving network management with Software Defined Networking. *IEEE Communications Magazine*, 51(2):114–119, 2013.

[KHZ17] Selma Khebbache, Makhlouf Hadji, and Djamal Zeghlache. Scalable and cost-efficient algorithms for vnf chaining and placement problem. In *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, pages 92–99, 2017.

[KN17] Adlen Ksentini and Navid Nikaein. Toward enforcing network slicing on ran: Flexibility and resources abstraction. *IEEE Communications Magazine*, 55(6):102–108, 2017.

[KNS+18] Zbigniew Kotulski, Tomasz Wojciech Nowak, Mariusz Sepczuk, Marcin Tunia, Rafal Artych, Krzysztof Bocianiak, Tomasz Osko, and Jean-Philippe Wary. Towards constructive approach to end-to-end slice isolation in 5g networks. *EURASIP Journal on Information Security*, 2018(1):2, Mar 2018.

[KREV+15] D. Kreutz, F.M.V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, Jan 2015.

[LHK+13] Pingping Lin, Jonathan Hart, Umesh Krishnaswamy, Tetsuya Murakami, Masayoshi Kobayashi, Ali Al-Shabibi, Kuang-Ching Wang, and Jun Bi. Seamless interworking of sdn and ip. volume 43, pages 475–476, 08 2013.

[LHM10] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: Rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, New York, NY, USA, 2010. Association for Computing Machinery.

[LZC+16] Hongjian Li, Guofeng Zhu, Chengyuan Cui, Hong Tang, Yusheng Dou, and Chen He. Energy-efficient migration and consolidation algorithm of virtual machines in data centers for cloud computing. *Computing*, 98(3):303–317, Mar 2016.

[Mal98] Gary S. Malkin. RIP Version 2. RFC 2453, November 1998.

[MF19] Dania Marabissi and Romano Fantacci. Highly flexible ran slicing approach to manage isolation, priority, efficiency. *IEEE Access*, 7:97130–97142, 2019.

[MGT+15] Jon Matias, Jokin Garay, Nerea Toledo, Juanjo Unzilla, and Eduardo Jacob. Toward an SDN-enabled NFV architecture. *IEEE Communications Magazine*, 53(4):187–193, 2015.

[Moy98] John Moy. OSPF Version 2. RFC 2328, April 1998.

[ONF14] Optical Interconnecting Forum Open Networking Foundation. Global transport sdn prototype demonstration. https://opennetworking.org/wp-content/uploads/2013/02/oif-p0105_031_18.pdf, 10 2014.

[PNL+20] M. Pozza, P. K. Nicholson, D. F. Lugones, A. Rao, H. Flinck, and S. Tarkoma. On reconfiguring 5g network slices. *IEEE Journal on Selected Areas in Communications*, 2020.

[PPP20] 5G PPP. View on 5g architecture. https://5g-ppp.eu/wp-content/uploads/2020/02/5G-PPP-5G-Architecture-White-Paper_final.pdf, 02 2020.

[QKA18] Long Qu, Maurice Khabbaz, and Chadi Assi. Reliability-aware service chaining in carrier-grade softwarized networks. *IEEE Journal on Selected Areas in Communications*, PP, 03 2018.

[QN15] P. Quinn and T. Nadeau. Problem statement for service function chaining. RFC 7498, RFC Editor, April 2015.

[RHL06] Yakov Rekhter, Susan Hares, and Tony Li. A Border Gateway Protocol 4 (BGP-4). RFC 4271, January 2006.

[SGT20] S. Sharma, A. Gumaste, and M. Tatipamula. Dynamic network slicing using utility algorithms and stochastic optimization. In *2020 IEEE 21st International Conference on High Performance Switching and Routing (HPSR)*, 2020.

[SHS+12] Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. Making middleboxes someone else's problem: network processing as a cloud service. *ACM SIGCOMM Computer Communication Review*, 42(4):13–24, 2012.

[SZGS+18] Josep Xavier Salvat, Lanfranco Zanzi, Andres Garcia-Saavedra, Vincenzo Sciancalepore, and Xavier Costa-Perez. Overbooking network slices through yield-driven end-to-end orchestration. In *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '18, New York, NY, USA, 2018. Association for Computing Machinery.

[TAM19] S. Troia, R. Alvizu, and G. Maier. Reinforcement learning for service function chain reconfiguration in nfv-sdn metro-core optical networks. *IEEE Access*, 2019.

[THGJ18] Andrea Tomassilli, Nicolas Huin, Frederic Giroire, and Brigitte Jaumard. Resource requirements for reliable service function chaining. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–7, 2018.

[TTG13] Phuong Nga Tran and Andreas Timm-Giel. Reconfiguration of virtual network mapping considering service disruption. In *IEEE International Conference on Communications - ICC*, pages 3487–3492. IEEE, 2013.

[Var12] Upkar Varshney. 4g wireless networks. *IT Professional*, 14(5):34–39, 2012.

[WBIC19] Adrien Wion, Mathieu Bouet, Luigi Iannone, and Vania Conan. Change in continuity: Chaining services with an augmented igp. *IEEE Transactions on Network and Service Management*, 16(4):1332–1344, 2019.

[WFQ+19] G. Wang, G. Feng, T.Q.S. Quek, S. Qin, R. Wen, and W. Tan. Reconfiguration in network slicing-optimizing the profit and performance. *IEEE Transactions on Network and Service Management*, 16(2):591–605, June 2019.

[WM13] R. Wang and B. Mukherjee. Provisioning in elastic optical networks with non-disruptive defragmentation. *IEEE Journal of Lightwave Technology*, 31(15):2491–2500, 2013.

[YG12] Soheil Yeganeh and Yashar Ganjali. Kandoo: A framework for efficient and scalable offloading of control applications. *HotSDN'12 - Proceedings of the 1st ACM International Workshop on Hot Topics in Software Defined Networks*, 08 2012.

[YLW+19] Xu Yang, Yue Liu, Ieok Cheng Wong, Yapeng Wang, and Laurie Cuthbert. Effective isolation in dynamic network slicing. In *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6, 2019.

[ZLF+17] Nan Zhang, Ya-Feng Liu, Hamid Farmanbar, Tsung-Hui Chang, Mingyi Hong, and Zhi-Quan Luo. Network slicing for service-oriented networks under resource constraints. *IEEE journal on Selected Areas in Communications*, 35(11):2512–2521, 2017.

[ZZC20] J. Zhou, W. Zhao, and S. Chen. Dynamic network slice scaling assisted by prediction in 5g network. *IEEE Access*, 2020.

CHAPTER 1

Preliminaries

In this Preliminary chapter we introduce different optimisation tools and techniques that are used in this thesis. Understanding all the models presented requires a basic knowledge of these optimisation techniques. Linear Programming is used in all the chapters of this thesis. Column generation is used in chapter 4 and 5. Finally reinforcement learning is used in chapter 5.

1.1 Linear Programming (LP)

Linear programming, also called linear optimisation, is, as the second name suggests, a method for solving optimisation problems. Since 1947 with the introduction of the *simplex algorithm* by Dantzig [Dan48] (the first practical approach to solving linear programs), linear programming has become one of the most widely used methods for solving optimisation problems. A linear program (LP) consists of three components:

- An **objective function**, which can be a cost to minimise or a profit to maximise. This is the main driver of decision making in optimisation problems.
- A set of **decision variables**. Each variable has a coefficient in the objective (in terms of cost or profit). The value of the objective is therefore dependent on the value of the set of decision variables. The value of the variables determine the output solution.
- A set of **linear constraints** which are in the form of equality and/or inequality. Each constraint restricts the value of one or more variables and thus reduces the set of possible solutions.

Linear programming is widely used to solve optimisation problems in operations research such as scheduling, flow routing, resource allocation, resource management, etc. In a LP, all variables must be fractional, however there are many problems that require integer or binary variables. For example, to model a delivery vehicle scheduling problem, it is impossible to use a fraction of a vehicle, when a truck is sent on delivery it is sent entirely. When all variables are binaries or integers the problem is an ILP (Integer Linear Program). When there is a mix of fractionals and integers/binaries variables the problem is a MILP (Mixed Integer Linear Program) but we will refer to it as ILP during this thesis to keep the nomenclature simple. Solving an ILP instead of an LP changes the complexity of the problem, as explained in subsection 1.1.2.

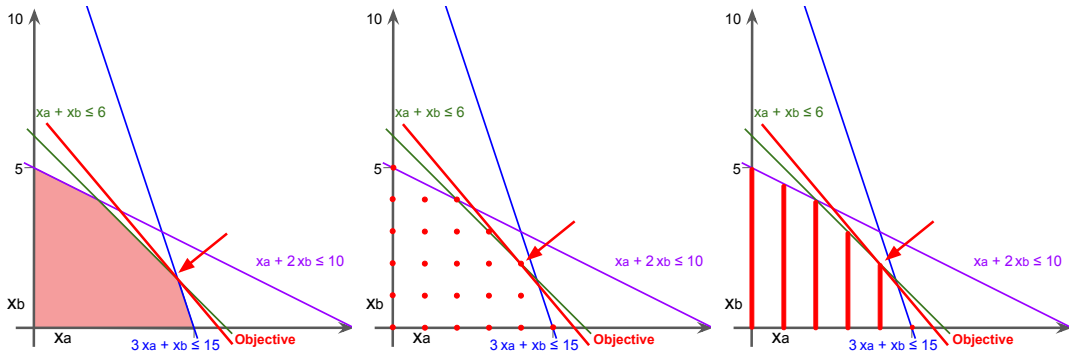
A very simple example allows to understand and visualise how it works. A company manufactures two products a and b. Two variables x_a and x_b represent the output quantity for products a and b. To produce a and b three resources are needed: r_1, r_2, r_3. There are 6 units of resource r_1, 15 units of resource r_2 and 10 units of resource r_3 in stock. To prepare one unit of a, 1 unit of resource r_1, 3 units of resource r_2 and 1 unit of resource r_3 are needed. To prepare one unit of b, 1 unit of resource r_1, 1 unit of resource r_2 and 2 units of resource r_3 are needed. On sale, product a (resp. b) has a profit of \$12 (resp. \$10). The aim of the problem is to determine the number of products a and b to produce in order to maximise the profit with the given number of resources.

This problem can be modelled as follows:

$$\begin{array}{lrcll} \text{Maximise} & 12x_a + 10x_b & & & \\ \text{Subject To:} & x_a + x_b & \leqslant & 6 & (1) \\ & 3x_a + x_b & \leqslant & 15 & (2) \\ & x_a + 2x_b & \leqslant & 10 & (3) \\ & x_a & \geqslant & 0 & \\ & x_b & \geqslant & 0 & \end{array} \tag{1.1}$$

The constraint (1), (2) and (3) respectively represent the amount of resources r_1, r_2 and r_3 to make products a and b.

This problem can also be represented by a plot. Figure 1.1 represents different versions of the problem.



(a) LP (b) ILP (c) MIP

Figure 1.1 – Representation of the graphical solution of a linear optimisation problem

- First, in Figure 1.1(a), each axis represents a variable, x_a or x_b, and each line corresponds to a constraint (except the red one which corresponds to the objective). The colours of the lines (green, blue and purple) correspond to those of the model 1.1. The red polytope is the area of all possible solutions and is delimited by all the constraints. To find the optimal solution, the objective line is moved until it reaches an extreme border of the possible solutions area. In this example, the solution found is 69, with a value of 4.5 for x_a and 1.5 for x_b. The variables are therefore part of $\mathbb{R}^+$.
- Depending on the problem, the output values should be integer, and not fractional. Indeed, if for example x_a and x_b represent a number of phones to be sold, it's impossible to sell half a phone. In Figure 1.1(b), x_a and x_b are part of $\mathbb{N}$ and therefore the sum of each can only be part of $\mathbb{N}$. The area of possible solutions is thus made up of a set of points and the best solution must be chosen using the objective line. The program is no longer an LP but an ILP. The result is 68 with 4 for x_a and 2 for x_b.
- Finally, a third version, if x_a is part of $\mathbb{N}$ and x_bb is part of $\mathbb{R}^+$. As in Figure 1.1(c), the set of solutions can be represented by lines and the solution is found by the objective line. In this case, the program is a MILP.

Linear programs are written in the form:

$$\begin{aligned} &\text{Minimise/Maximise} && \sum_{i=1}^{n} c_i x_i && && (1) \\ &\text{Subject To:} && \sum_{i=1}^{n} a_{j,i} x_i && \begin{cases} \leqslant \\ = \\ \geqslant \end{cases} b_j, \quad j \in [1,m] && (2) \\ & && x_i && \begin{cases} \leqslant \\ \geqslant \end{cases} 0, \quad i \in [1,n] && (3) \end{aligned} \tag{1.2}$$

Where n is the number of variables and m the number of constraints. $X = \{x_1, x_2, ..., x_n\}$ is the set of decision variables. $C = \{c_1, c_2, ..., c_n\}$ is the set corresponding to the coefficients of the variables in the objective function, it may be a cost to minimise or a profit to maximise. For

every linear constraint $j \in [1, m]$ there is a set of coefficient $a_{j,i}$ for every variables x_i and b_j is the *right-hand-side* of equation j.

In Equation 1.2, line (1) corresponds to the objective, line (2) corresponds to the set of constraints and finally line (3) corresponds to the restriction on variables: either unrestricted or negative or positive.

There is a **standard form** of writing LPs. The problem must be a maximisation and all constraints must be of the form lesser or equal. To switch to a minimisation problem the signs of the coefficients of the variables are changed and the constraints are switched to greater or equal.

Any linear program can be written in matrix form and in standard form:

$$\begin{array}{ll} \text{Maximise} & z = c^T x \\ \text{Subject To:} & Ax \leqslant b \\ & x \geqslant 0 \end{array} \tag{1.3}$$

- $c = \begin{pmatrix} c_1 \\ ... \\ c_n \end{pmatrix}, x = \begin{pmatrix} x_1 \\ ... \\ x_n \end{pmatrix}$ are column vectors of size n.
- $b = \begin{pmatrix} b_1 \\ ... \\ b_m \end{pmatrix}$ is a column vector of size m.
- $A = \begin{pmatrix} a_{1,1} & ... & a_{1,n} \\ ... & ... & ... \\ a_{m,1} & ... & a_{m,n} \end{pmatrix}$ is a matrix of size $m \times n$.
- c^T represents the transpose of vector c.

1.1.1 A general example

For a more general example and to facilitate the understanding of the next section, an ILP modelling of the shortest path problem between two points with capacity constraints is considered. Using an ILP is not the most efficient way to solve it, but it is a simple way to understand the modelling of constraints.

To summarise, on a graph $G = (V, E)$, a request from a source $v_s \in V$ must be routed to a destination $v_d \in V$ and it consumes an amount bw of bandwidth. The objective is to find a shortest path between v_s and v_d that respects the capacity C_{uv} of each link $uv \in E$. The list of parameters and variables is noted in Table 1.1.

Parameters

$G = (V, E)$	The network where V represents the set of nodes and E the set of links.
C_{uv}	Capacity of a link $(u, v) \in E$ expressed as its total bandwidth available.
(v_s, v_d, bw)	The demand is modeled by a triplet with $v_s \in V$ the source, $v_d \in V$ the destination, and bw the required units of bandwidth.

Variables

x_{uv}	Utilisation of link $(u, v) \in E$. $x_{uv} \in \{0, 1\}$ is equal to 1 if the link (u, v) is used, 0 otherwise.

Table 1.1 – Notation for the shortest path problem

Objective: Minimise the number of links used

$$\min \sum_{(u,v)\in E} x_{uv} \tag{1.4}$$

Flow conservation constraints: Constrains the flow to go from v_s to v_d. For each Node $u \in V$.

$$\sum_{(u,v)\in\omega^+(u)} x_{uv} - \sum_{(v,u)\in\omega^-(u)} x_{vu} = \begin{cases} 1 \text{ if } u = v_s \\ -1 \text{ if } u = v_d \\ 0 \text{ else} \end{cases} \tag{1.5}$$

Link capacity constraints: For each Link $(u, v) \in E$.

$$bw \cdot x_{uv} \leq C_{uv} \tag{1.6}$$

This simple example provides a general understanding of modelling. Equation 1.4 is the objective and minimises the number of links used, we are looking for the shortest path. Equation 1.5 is a set of 3 constraints for each node, it forces one and only one link to leave if the node is v_s and one and only one link to reach the node if it is v_d. For each intermediate node, every incoming link implies an outgoing link. There is no anti-cycle constraint as, thanks to the shortest path objective, a cycle cannot be part of the shortest path.

Finally, Equation 1.6 forbids the use of any link with not enough capacity to pass the flow. With a single request, the links with not enough capacity can be removed before the modelling and therefore, this constraint is not useful anymore. There is no non-negativity constraint because we specify in Table 1.1 that the x variables can only take the value 0 or 1.

1.1.2 Linear Programming properties

The *simplex algorithm* is the most common method to solve LPs. To summarise, the simplex method proceeds by going through the polytop (a compact convex set with a finite number of extreme points) from one vertex to another. At each step, it chooses the best vertex relative to the objective function. Either the algorithm determine that the constraints are unsatisfiable, or it determine that the objective function is unbounded, or finally it reach a vertex from which it cannot progress, which optimises the objective function.

Other algorithms exist, without going into details we can mention the *ellipsoid method* proposed by Khachiyan [Kha80] who proves that an LP can be solved in polynomial time. But

this algorithm was only effective in theory. Finally, in 1984, Karmarkar [Kar84] developed a new polynomial algorithm for practical use, based on the *interior-point method*. Thanks to these advances, LPs can be solved in polynomial time. While solving an ILP is NP-hard in general, it can be useful even in large problems. As explained in Section 1.2, some problems can be broken down into several smaller problems and solved quickly by an ILP. However, these solutions may not always be optimal. ILPs can also be relaxed by removing the integrality constraint from each variable. This method provides an upper-bound for maximisation and a lower-bound for minimisation, which is useful for approximation algorithms. Using the previous example, the LP scores 69 against 68 for the ILP. By using the LP solution and rounding the value of the variables, the optimal solution is found. This does not work in every case and the optimal solution may be far from the relaxation solution (there may be no solution to the ILP while there is one for the LP).

One of the key properties of LP used in the following section is the **Duality**. For each LP written in standard form, called the *Primal*, there is another LP called the *Dual*. The objective of the dual is the opposite of the primal one, i.e. if the primal is a minimisation problem, then the dual is a maximisation one, and vice-versa. For every variable in the primal there is a constraint in the dual. And for every constraint in the dual there is a variable in the dual. The dual problem of the dual is the primal.

The dual problem (left) of the precedent primal 1.3 (right) is modelled as follows:

$$\begin{array}{lll} \text{Minimise} & b^T y & \\ \text{Subject To:} & A^T y \geqslant c & \\ & y \geqslant 0 & \end{array} \qquad \begin{array}{lll} \text{Maximise} & c^T x & \\ \text{Subject To:} & Ax \leqslant b & \\ & x \geqslant 0 & \end{array} \tag{1.7}$$

Where y is the vector variables of size m of the dual problem.

There are several rules for transforming the constraints and bounds of a primal maximisation problem to its dual:

- For an inferiority constraint and a non-negativity bound, the dual has a superiority constraint and a non-negativity bound.
- For an equality constraint and a non-negativity bound, the dual has a superiority constraint and no bound.
- For an inferiority constraint with no bound, the dual has an equality constraint and a non-negativity bound.

For example, the dual problem (left) of the precedent primal 1.1 (right) is modelled as follows:

$$\begin{array}{lll} \text{Minimise} & 6y_a + 15y_b + 10y_c & \\ \text{Subject To:} & y_a + 3y_b + y_c \geqslant 12 & \\ & y_a + y_b + 2y_c \geqslant 10 & \\ & y_a \geqslant 0 & \\ & y_b \geqslant 0 & \\ & y_c \geqslant 0 & \end{array} \qquad \begin{array}{lll} \text{Maximise} & 12x_a + 10x_b & \\ \text{Subject To:} & x_a + x_b \leqslant 6 & \\ & 3x_a + x_b \leqslant 15 & \\ & x_a + 2x_b \leqslant 10 & \\ & x_a \geqslant 0 & \\ & x_b \geqslant 0 & \end{array} \tag{1.8}$$

Two theorems emerge from the duality [Dan63]:

- **The weak duality theorem:** The objective value of any feasible solution of the dual is an upper bound on the optimal objective value of the primal solution. And reciprocally the objective value of any feasible solution of the primal is a lower bound on the optimal objective value of the dual solution. This implies that if the dual is unbounded, then the primal has no feasible solution, and reciprocally if the primal is unbounded, then the dual has no feasible solution.
  If x^* is a feasible solution of the primal maximisation and y^* is a feasible solution of the dual minimisation then the weak duality theorem can be stated as :
  $$\text{Maximise} \quad c^T x^* \quad \leqslant \quad \text{Minimise} \quad b^T y^*$$
- **The strong duality theorem:** If the primal problem has an optimal solution with a finite objective value then so does the dual (and vice versa). This objective value is the optimal value for both problems. The bounds given by the weak duality theorem are tigh. If x^* is an optimal solution of the primal maximisation, there exists y^* an optimal solution of the dual minimisation and:
  $$\text{Maximise} \quad c^T x^* \quad = \quad \text{Minimise} \quad b^T y^*$$

The comprehension of the duality in LP is important to understand the functioning of the column Generation (CG).

1.2 Column Generation

ILPs can be an interesting method for solving small problems but they quickly become inefficient when the size of the problem increases. Some large problems can still benefit from ILP modelling under certain conditions.

Column Generation (CG) [DDS05] is a decomposition method coming from the field of operational research and which divides an optimisation model into two parts:

- A **Restricted Master Problem** (RMP) which consists of the original problem but with a restricted number of variables and consequently of possible solutions.
- A set of **Pricing Problems** (PP) which consists of a smaller problem by focusing on one (or a subset) of the RMP variables, thus allowing a faster execution. Each pricing create columns (decision variables) to feed the RMP. Using an ILP is not mandatory for modelling PPs. Any exact optimisation algorithm suffice as long as the objective can be modified. Using an approximation algorithm or a heuristic can improve the execution time of the PPs but can potentially reduce the quality of the RMP solution.

In order to use column generation, a problem needs to be divided into several pricing problems.

To recall, an LP uses a matrix in which the rows are the constraints and the columns are the decision variables. The CG works by gradually adding columns (decision variables) to find a good solution.

For example Three PPs, red, blue and green are the sub-problems of a RMP. At the start of the algorithm the constraint matrix of the RMP is:

$$A = \begin{pmatrix} a_{1,(1,1)} & a_{1,(2,1)} & a_{1,(3,1)} \\ \dots & \dots & \dots \\ a_{m,(1,1)} & a_{m,(2,1)} & a_{m,(3,1)} \end{pmatrix}$$

Where $a_{x,(y,z)}$ represents the coefficients of the column z of the pricing y in the x^{th} constraint. After each pricing has produced a valid column, the matrix evolves.

$$A = \begin{pmatrix} a_{1,(1,1)} & a_{1,(1,2)} & a_{1,(2,1)} & a_{1,(2,2)} & a_{1,(3,1)} & a_{1,(3,2)} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ a_{m,(1,1)} & a_{m,(1,2)} & a_{m,(2,1)} & a_{m,(2,2)} & a_{m,(3,1)} & a_{m,(3,2)} \end{pmatrix}$$

Figure 1.2 is a representation of how CG works. In order to be solved, CG needs first a valid solution (even a very bad one). Usually the algorithm starts with the smallest possible number of variables (down to one per pricing problem). This first set of variables may be found either through using a heuristic or by using the PPs a first time if it works. The quality of this first solution can have an impact on the quality of the final solution found by the RMP. Variables will then be added at each iteration. In order to add variables that can potentially improve the solution of the master, the relaxation of the master problem is solved. The use of an LP allows, thanks to the simplex algorithm, to obtain the dual values of the constraints. These dual values indicate for each row if it constrains the solution of the problem. The dual values are then injected in the pricing problems. For each pricing problem, when a column related to it is present in a constraint, the dual of this constraint influences the pricing objective.

The objective $\overline{C_j}$ of the PP for the variable j is called the reduced cost and is expressed as follows:

$$\overline{C_j} = C_j - \sum_i a_{ij} \cdot \mu_i \tag{1.9}$$

Where C_j corresponds to the original cost of the variable j in the RMP objective. The sum on i is the sum of all constraints i where the variable j appears. a_{ij} is the coefficient of the variable j in the constraint i. μ_i is the dual value of the constraint i.

Each PP is then solved. If the reduced cost of a PP is less than 0, the resulting column can potentially improve the solution of the RMP relaxation, so it is added to the set of columns of the RMP. The algorithm then iterates between executing the RMP relaxation and executing the PPs. When no more columns are created the integral RMP is executed. This execution gives an optimal solution for the restricted set of columns of the RMP. The end of the iteration can also be triggered using a timer or when the improvement of the RMP relaxation objective is lower than a parameterised threshold. Finally, even if the RMP has a restricted number of variables, in some problems this number may still be so large that it cannot be executed in a reasonable time. In this case, a column filtering method can be added to remove the least useful columns. As an example, the least used columns in relation to their number of iterations in the RMP may be removed. However, removing columns may decrease the quality of the RMP solution, as variables not used when running the LP may become useful when running the ILP.

The solution given by the RMP may be close to or far from the actual optimal solution, depending on the problem. But, the CG at least gives a solution, while it is not possible for an ILP due to time execution constraints (as we will see in chapter 4). The quality of the CG solution (objective = π^*_{CG}) can be estimated by comparing it to the optimal solution of the relaxed problem (objective = π^*_{LP}). By calculating $(\pi^*_{CG} - \pi^*_{LP})/\pi^*_{LP}$ we obtain a ratio. If this ratio is 0, then the solution is optimal, and the closer it is to 0, the better the solution.

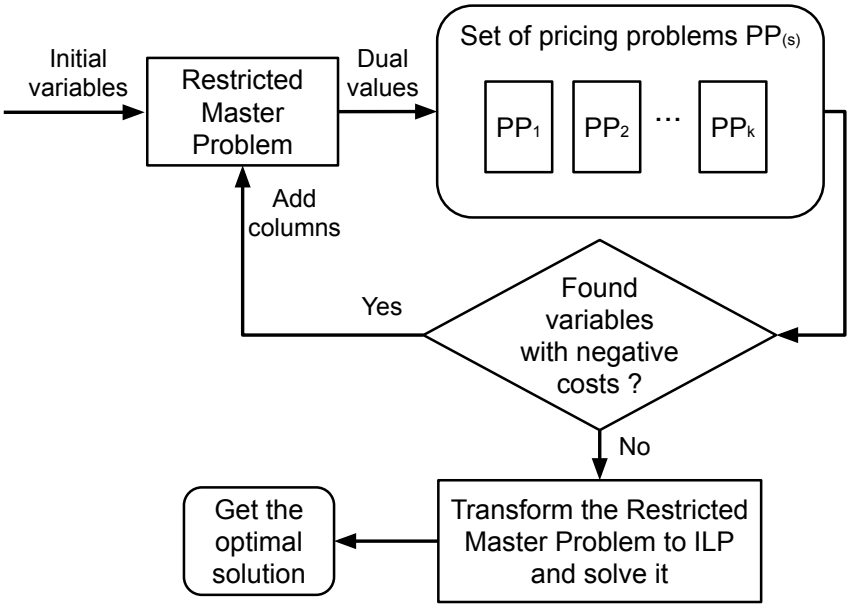


Figure 1.2 – How column generation works

The **advantage** of column generation is its good optimisation acceleration capabilities. In the last few years, the number of cores per CPU has increased drastically. Since the PPs are independent, they can be solved in parallel and take advantage of this technological advance.

The **drawbacks** of column generation are as follows. Finding a first set of variables to start the algorithm may be difficult, and the quality of this first set may have an impact on the quality of the solution. In some problems, the speed up may be negligible or the execution time may be longer than ILP. Pricing problems can take too long to solve if the problem they address is too complex. Finally, the implementation of a column generation model is more complex than an ILP one.

1.2.1 A general example

To get a better understanding of how CG works, the previous example of the shortest path problem is generalised. This problem is called the Multi-commodity flow problem. It is extremely similar to the previous one except that this time there are several requests that must be satisfied on the same network. The objective changes slightly, instead of minimising the length of paths, the bandwidth is minimised (which implies that between two requests with different throughput, it is better to prioritise the optimisation of the larger ones and to take longer routes for the smaller ones. The variables and parameters are almost the same as the ones for the shortest path problem. Each triplet identifies the needs of a request (source, destination, bandwidth) is now linked to a demand $d \in D$. The same applies to variables x which are associated to a request $d \in D$ in addition to being associated to a link $l \in E$. This ILP can therefore be used to solve our new problem, but as we said before using an ILP (especially if the number of requests is very large) can quickly become inefficient. To solve this problem with a large number of requests, the CG technique is used.

To understand the modelling of the problem, consider Figure 1.2. Instead of having an ILP finding the solution for all requests, a set of pricing problems (PPs) is defined. Each one finds a possible path for a request at every iteration. These paths are then given to the RMP which instead of having as many variables as there are links and requests, has a reduced number of paths.

Table 1.2 adds notations used in the RMP.

Parameters

D	The set of demands d
P_d	The set of paths p from demand $d \in D$
δ_{uv}^p	Is equal to 1 if the link (u, v) is in path p, 0 otherwise

Variables

z_p	Utilisation of path p, its equal to 1 if path p is used, 0 otherwise.

Table 1.2 – Additional notation for the RMP

Restricted Master Problem

Objective: minimise the bandwidth used

$$\min \sum_{d \in D} \sum_{p \in P_d} \sum_{(u,v) \in E} z_p \cdot \delta_{uv}^p \cdot bw_d \tag{1.10}$$

One path used per demand. For each Demand $d \in D$.

$$\sum_{p \in P_d} z_p = 1 \tag{1.11}$$

Link capacity constraints. For each Link $(u, v) \in E$.

$$\sum_{d \in D} \sum_{p \in P_d} z_p \cdot bw_d \cdot \delta_{uv}^p \leq C_{uv} \tag{1.12}$$

The objective 1.10 is a little different, the total bandwidth used must be minimised and to do this the bandwidth for the path used by each request must be minimised. There are no longer any flow conservation constraints which are managed by the PPs. Constraint 1.11 forces the use of one and only one path per request. And constraint 1.12 prohibits the overloading of links.

For the **Pricing Problems**, the model is the same as for the shortest path except that the objective must take into account the dual values of 1.11 and 1.12 constraints, represented by μ. The original cost C_j (see Equation 1.9) of a column j is the bandwidth used by the demand:

$$\sum_{(u,v) \in E} x_{uv} \cdot bw_d$$

Before writing the objective function, it is necessary to verify the form of the constraints in the RMP. Indeed, Equation 1.9 is valid when the RMP is written in standard form. For a minimisation problem, all constraints must be in the form geq or the dual of the constraints will have negative values.

Link capacity constraints in standard form. For each Link $(u, v) \in E$.

$$\sum_{d \in D} \sum_{p \in P_d} -bw_d \cdot \delta_{uv}^p \geq -C_{uv} \tag{1.13}$$

The new *objective* is therefore:

$$\min \quad -\mu_d^{1.11} + \sum_{(u,v) \in E} x_{uv} \cdot bw_d \cdot (\mu_{uv}^{1.13} + 1) \tag{1.14}$$

For constraints 1.11, there is one constraint per demand d. In the PP objective, the dual value is therefore the one associated with the PP demand, resulting in the notation $\mu_d^{1.11}$. On the contrary for constraints 1.13, there is one constraint per link (u, v), which means that the dual value are used by all PPs resulting in the notation $\mu_{uv}^{1.13}$. $\mu_{uv}^{1.13}$ is positive because in 1.13 the variable δ_{uv}^p has a negative coefficient.

For a more complete understanding of column generation, the reader may refer to [DDS05].

The implementation of the Multi-commodity flow problem can be found at [Gaub]

1.3 Reinforcement Learning

Machine learning is a branch of Artificial Intelligence and Computer Science which focuses on the use of data and algorithms to learn and adapt themselves without following explicit instructions. Machine learning uses statistical models to analyse and draw inferences from patterns found in data samples, in order to make predictions or decisions without relying on a predetermined equation as a model. The algorithms adaptively improve their accuracy as the number of training samples increases.

Algorithms used in machine learning basically fall into four categories:

- **Supervised Learning:** algorithms use a labelled and classified dataset. When a prediction is made, a function calculates an error by comparing the prediction and the label to determine how accurate the prediction is. The error result is used to adapt the prediction function.
- **Unsupervised Learning:** algorithms use a dataset that only contains inputs. They look for hidden patterns or intrinsic structures in the data. They learn based on the presence or absence of such patterns in each new dataset.
- **Semisupervised Learning:** algorithms fall between the above categories. The dataset contains both labelled and unlabelled data, which allows the algorithms to learn how to label data. This guides the algorithms to draw independent conclusions and can improve the learning accuracy.
- **Reinforcement Learning:** algorithms are presented in the following subsection.

1.3.1 Definition

Reinforcement Learning exists in a context where there is an environment with a state and where taking an action change the state of that environment. It has the ability to learn how to map a series

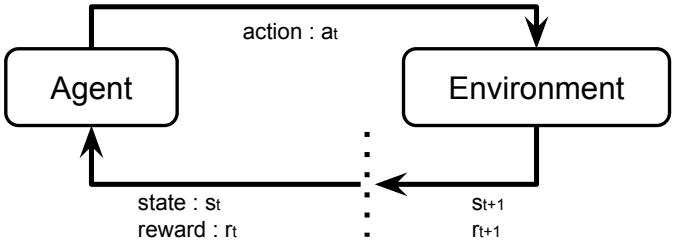


Figure 1.3 – Interaction between the agent and the environment

of inputs to outputs with dependencies. This model progressively learns through trials and errors. A sequence of positive outcomes are reinforced to build the best recommendation or policy for a given problem.

A reinforcement learning algorithm consists of an environment and an agent. The agent is the entity making the decisions. The environment evolves as the agent makes decisions, and gives the agent the state on which to perform the action and a reward as feedback for its previous action.

As can be seen in Figure 1.3, at an instant t, the environment gives the state s_t to the agent. The agent makes a decision by sending the action a_t to the environment. The environment modifies its state according to the transition probabilities between states (which itself depends on the action) and sends the state s_{t+1} and the reward r_{t+1} to the agent.

1.3.2 Markov Decision Process

Almost all the RL problems can be modeled as Markov Decision Processes.

The model is composed by five elements :

- S a set of **states**.
- A a set of **actions** of the agent.
- P the **transition probability function** with $P(s', r|s, a)$ the probability of transition from state s to state s' under action a while receiving reward r.
  With $\mathbb{P}$ the symbol of probability:

$$P(s', r|s, a) = \mathbb{P}[s_{t+1} = s', r_{t+1} = r|s_t = s, a_t = a]$$

  Thus, the state-transition function can be defined as a function of this probability:

$$P^a_{ss'} = P(s'|s, a) = \mathbb{P}[s_{t+1} = s'|s_t = s, a_t = a] = \sum_{r \in R} P(s', r|s, a)$$

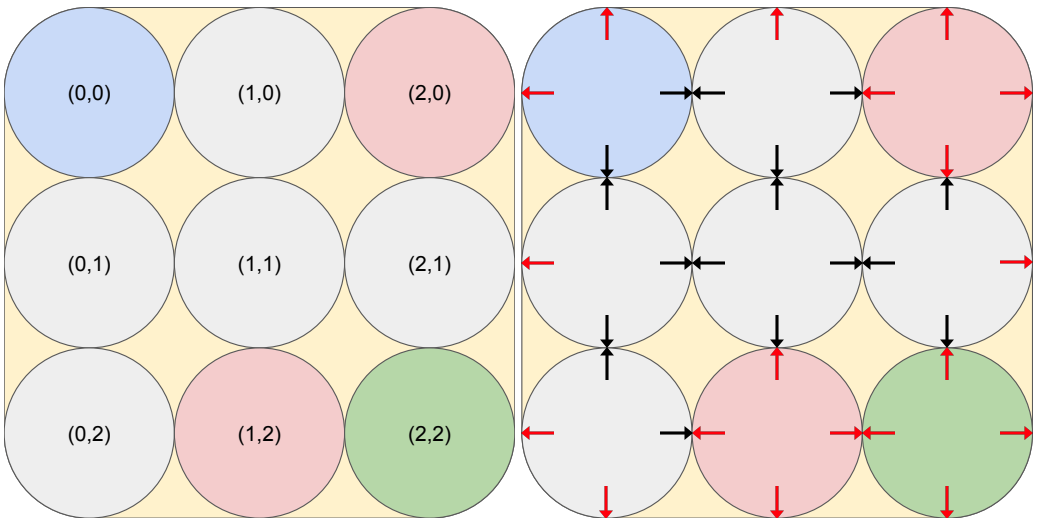
- R the **reward function** which gives the value of the next reward when taking the action a on state s:

$$R(s, a) = \mathbb{E}[r_{t+1}|s_t = s, a_t = a] = \sum_{r \in R} r \sum_{s' \in S} P(s', r|s, a)$$

- $\gamma \in [0, 1]$ is the **discount factor** for future rewards. It specifies the extent to which future rewards impact on the outcome of the current action.

All states transition have the Markovian property: given the current state s and action a, the next state s' is conditionally independent of all the previous states and actions.

For example: As a toy example, let us consider an agent that moves in a labyrinth (which is modeled by a small grid here for sake of simplicity). On each cell, the agent may gain some reward or some penalty. Starting from a cell, the goal is then for the agent to progressively learn an optimal trajectory (i.e., guaranteeing it a best final gain starting from this cell). Intuitively, the agent begins by following random trajectories. The following trajectories are increasingly guided by the memory of its previous trials. More precisely, the environment is represented by a 3*3 grid (Figure 1.4(a)) and each state is represented by a cell. The cells are identified by their position (x,y) on the grid with x the horizontal axis and y the vertical axis. The agent starts from cell (0,0), it can only move to the adjacent cells and must learn to go to cell (2,2). It must not pass through cells (2,0) and (1,2). When the agent reaches the cell (2,2), (2,0), or (1,2) the game ends. To help the agent make a decision, he is given a reward of 1 when he reaches the state (2,2) and a reward of -1 when he reaches the state (2,0) and (1,2).
$S = \{(0,0), (0,1), (0,2), ..., (2,1), (2,2)\}$.
The agent has four possible actions at his disposal, go up, down, left or right.
$A = \{left, right, up, down\}$.
$R(2,2) = 1, R(1,2) = -1, R(2,0) = -1$.
The transition probability function is simple, for each transition the probability is either 1 when the transition involves two adjacent cells with the appropriate action or 0. The possible movements are represented by arrows in Figure 1.4(b). For each state the agent can perform all actions. The red arrows represent actions that are impossible. Either because the state is a terminal state and the agent cannot perform any action when the game is over. Either because theses actions do not change the agent's state and we forbid them for simplicity.



(a) Reward of each state (b) Transitions of the agent

Figure 1.4 – Environment example

1.3.3 Policy and Value function

By interacting with its environment, the RL agent determines which actions produce the greatest reward and uses this experience to improve its performance on future trials. The agent's main objective is therefore to maximise the total amount of reward received (good actions are reinforced). To this end, the agent's behaviour is determined by a **policy** π. It provides a direction on the action to take in a certain state. $\pi(a, s)$ gives the probability of taking action a in state s
$\pi(a, s) \in [0, 1]$
$\pi(a, s) = \mathbb{P}(a_t = a | s_t = s)$

The return G_t is a weighted sum of the future rewards starting from time t:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}.$$

Not all rewards have the same impact on G_t. Even if it is important to take into account potential rewards that are far in the future, rewards that are closer in time may have more meaning. By setting the γ parameter to a value smaller than 1, we ensure that the further into the future a potential reward is, the less impact it has on the G_t return.
To choose the direction offering the best reward, a **value function** is associated $V(s)$ to each state. It predicts the expected value of the future rewards related to a state. The bigger the value function for a state, the better the state is.
The **value function** $V^\pi(s)$ is the expected return starting with state s by following policy π.

$$V^\pi(s) = \mathbb{E}_\pi[G_t | s_t = s]$$

The **action-value** (or Q-value) of a state action pair is the expected return starting with state s and first performing action a, before following policy π.

$$Q^\pi(s, a) = \mathbb{E}_\pi[G_t | s_t = s, a_t = a]$$

By following the target policy π, the state-value can be rewritten using the probability distribution over the possible actions and the Q-values.

$$V^\pi(s) = \sum_{a \in A} \pi(a, s) \cdot Q^\pi(s, a) \tag{1.15}$$

As the return G_t is recursively built, the Q and V functions can also be expressed recursively using the Bellman equations. Bellman's equations are related to dynamic programming and allow for the consideration of future states. A decision at a state s_t is recursively composed by the decisions at states s_{t+1}

$$\begin{aligned} V(s) &= \mathbb{E}[G_t | s_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... | s_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + ...) | s_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma G_{t+1} | s_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma V(s_{t+1}) | s_t = s] \end{aligned} \tag{1.16}$$

Similarly for Q-value,

$$\begin{aligned} Q(s,a) &= \mathbb{E}[R_{t+1} + \gamma V(s_{t+1}) | s_t = s, a_t = a] \\ &= \mathbb{E}[R_{t+1} + \gamma \mathbb{E}_{a \sim \pi} Q(s_{t+1}, a) | s_t = s, a_t = a] \end{aligned}$$

The **optimal policy** π^* achieves the optimal value functions: it maximises the expected cumulative reward. The optimal value function produces the maximum return.

$$V^{\pi *}(s) = \max_{\pi} V^{\pi}(s)$$
$$Q^{\pi *}(s,a) = \max_{\pi} Q_{\pi}(s,a)$$

The goal of a reinforcement learning agent is to learn the optimal policy to take the best decisions.

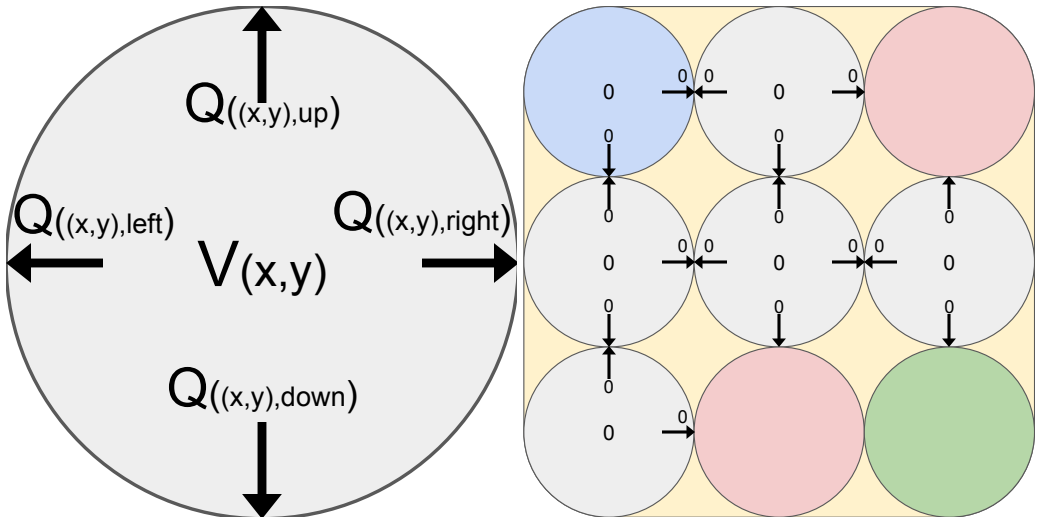
For example: To complete the previous example, the value function can be represented as a 3*3 matrix with one value per state.

$$V = \begin{pmatrix} V_{(0,0)} & V_{(0,1)} & V_{(0,2)} \\ V_{(1,0)} & V_{(1,1)} & V_{(1,2)} \\ V_{(2,0)} & V_{(2,1)} & V_{(2,2)} \end{pmatrix}$$

The Q-Value can be represented as a 3*3*4 matrix with one value per (state, action) pair.

$$Q = \begin{pmatrix} \begin{pmatrix} Q_{((0,0),left)} & Q_{((0,0),right)} \\ Q_{((0,0),up)} & Q_{((0,0),down)} \end{pmatrix} & \dots & \dots \\ \dots & \dots & \dots \\ \dots & \dots & \begin{pmatrix} Q_{((2,2),left)} & Q_{((2,2),right)} \\ Q_{((2,2),up)} & Q_{((2,2),down)} \end{pmatrix} \end{pmatrix}$$

Each state can therefore be represented as Figure 1.4(b) to visualise its value function and Q-Value. At the beginning of the algorithm their value is shown in Figure 1.5(b).



(a) Value-function and Q-Value (b) Start of the algorithm

Figure 1.5 – Example of V and Q tables

1.3.4 Exploration vs Exploitation

To find the optimal policy $\pi*$ and to be able to exploit it, the agent has to first explore the different states with every actions. However, this is not a good method. Indeed, it does not allow for scaling when the number of states and actions are too large. The trade-off between exploitation and exploration is one of the main challenges in reinforcement learning [SB18].

A simple and very common method is the ϵ-greedy policy. This method allows to control the exploration rate in relation to the exploitation rate. The value $\epsilon \in [0, 1]$ is the probability for the agent to choose a random action at each step : it is the exploration probability. Similarly, 1-ϵ is the probability of exploitation: the probability of following the policy.

The value ϵ can be a fixed or varied value. A commonly used method is the decay. The value of epsilon is close to 1 at the start of the algorithm to encourage a strong exploration at the beginning. Epsilon then decreases with each iteration to reach a minimum value (down to 0) to encourage exploitation. A good configuration of the decay allows not to fall in a local optimal at the beginning and to converge faster at the end of the training of the agent.

1.3.5 Q-Learning

There are different reinforcement learning algorithms and in this thesis we will focus on Q-Learning. The Q-Learning algorithm was introduced in 1992 by Watkins and Dayan [WD92] and aims at learning the Q-values $Q(s, a)$. It is used when both the action space and the state space are discrete.

It is an off-policy algorithm. Instead of having a single policy $\pi(s, a)$ (the target policy), there is a second policy $\beta(s, a)$ (the behaviour policy). The behaviour policy is used by an agent to select actions during exploration and the target policy is used during exploitation. The target policy is learnt independently of the agent's actual behaviour. Being an off-policy algorithm means that $\pi(s, a)$ is not the same as $\beta(s, a)$: the policy used during the evaluation of the agent is different from the one used during the training of the agent.

Q-Learning is a model-free algorithm which means it does not use the transition probability distribution and the reward function associated with the Markov decision process. For a model-free agent, to change the action associated with a state, it moves to that state, acts from it, possibly several times, and experiences the consequences of its actions. Model-free reinforcement learning algorithms can be described as trial and error learning methods.

In addition to the discount factor, Q-learning uses a parameter called the **learning rate** $\alpha \in [0, 1]$. It defines the extent to which newly acquired information overrides old ones. Like ϵ, α can be fixed or can vary depending on the problem being addressed.
The equation for updating the Q-value is written as follows:

$$Q(s_t, a_t) = (1 - \alpha)Q(s_t, a_t) + \alpha\Big[r + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})\Big] \tag{1.17}$$

Q-Learning learns it's Q-values from a sequence of agent interactions with the environment. Unfortunately, when the state and action spaces are huge, Q-learning requires a prohibitive computation time.

For example: By completing the previous example, an agent can be trained to solve this maze. The discount factor γ is set to 0.9 and the learning rate α to 0.5. The policy used is the ϵ-greedy with an ϵ value of 1 which decreases by 5% per round In the first round, the agent plays in a totally random way. In the second round, it has 5% chance of choosing the action with the best Q-value.

The first run can be seen in Figure 1.6(a): the agent goes right and then down and then right and then down and reaches the state (2,2). On reaching the state (2,2), it receives a reward of 1 and the Q-value of the previous state (2,1) is changed and the game is reset. During this reset the value function of all states is updated, which is visible (as well as the Q-function) in Figure 1.6(b).

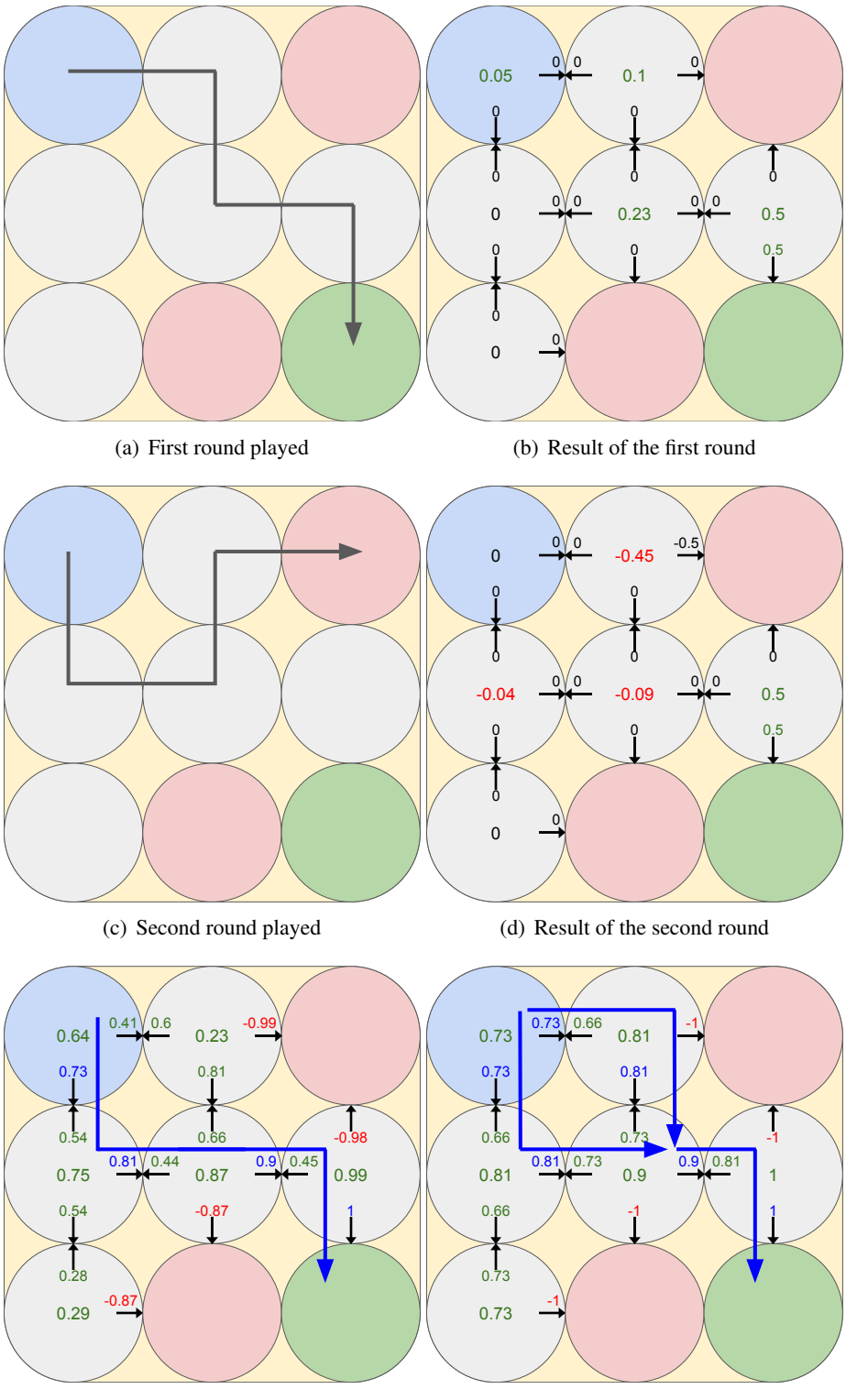
During the second run Figure 1.6(c), the agent plays down, right, up, right and reaches the state (2,0). On reaching the state (2,0), it receives a reward of -1 and the Q-value of the previous state (1,0) is changed and the game is reset. The negative reward propagates to the states through which the agent has passed Figure 1.6(d).

Finally, Figure 1.6(e) shows the Q-Value and the value function when the training is finished (the epsilon has reached a parameter value of 0.05). If the agent now follows its target policy, it will take the actions with the maximum Q-value (in blue) and will always arrive at the state (2,2). Even if the agent has finished training and is no longer in the exploration phase but in the exploitation phase (ϵ = 0), this does not mean that the optimal policy π^* has been found. The behavioural policy being based on partially random choices, it is possible that some states are rarely visited or not visited at all. For example, the state (0,2) at the bottom left has only been visited a few times and its value function is far from its optimal value. Nevertheless, it can be seen that even if the epsilon greedy policy does not necessarily give the optimal policy, it allows to converge towards it. Figure 1.6(f) represents the optimal policy, the two paths (in blue) (through (1,0) and through (0,1)) are optimal, and the value function converges. It is therefore equal to the Equation 1.15.

The agent training algorithm is represented by Algorithm 1. At each new step, the agent starts by calling Algorithm 2 to chooses an action following its behaviour policy (ϵ-greedy in this example). Then it applies its action to the environment and the environment returns the reward and the new state. A global variable of type LIFO is used to record all the transitions. This transition variable is used to update the value function at the end of a round. The agent then chooses the best action to take for the new state by using its target policy to update the Q-value. The agent then updates the Q-value of the current state by following Equation 1.17. Finally, the current state is updated and if the reward is different from 0, it means that the round is over and the environment is reset by calling Algorithm 3.

Algorithm 2 choose an action and takes as parameters the current state and the value of ϵ. A random value is drawn between 0 and 1 and if this value is lower than ϵ then a random action is chosen. Otherwise it is the best action known to the agent that is returned.

Finally Algorithm 3 resets the environment to play a new round. All transitions are processed one by one in their reverse order of arrival to update the value function by following Equation 1.16. Finally, ϵ decays and the current state is reset to the initial state.



(a) First round played

(b) Result of the first round

(c) Second round played

(d) Result of the second round

(e) End of the training

(f) Optimal policy

Figure 1.6 – Example of training the agent

Algorithm 1: step

Data: s_t, ϵ
Result: Play one step
$a_t \longleftarrow$ takeAction(s_t, ϵ);
$s_{t'}$, $r \longleftarrow$ Environement.play(s_t, a_t);
transition.push([s_t, r, $s_{t'}$]);
$a_{t'} \longleftarrow$ takeAction($s_{t'}$,0);
Q[s_t][a_t] $\longleftarrow$ Q[s_t][a_t] + α*(r+γ*Q[$s_{t'}$][$a_{t'}$]-Q[s_t][a_t]);
$s_t \longleftarrow s_{t'}$;
if *r != 0* **then**
    $s_t \longleftarrow$ reset(transition, α, γ, ϵ);
return s_t, r;

Algorithm 2: takeAction

Data: s_t, ϵ
Result: Choose the action to take
if *random(0,1)* < ϵ **then**
    $a_t \longleftarrow$ random(left, right, up, down);
else
    $a_t \longleftarrow$ argMax(Q[s_t]);
return a_t;

Algorithm 3: reset

Data: transition, α, γ, ϵ
Result: Reset the position of the agent to start a new round
while *transition.lenght* > *0* **do**
    s_t, r, $s_{t'}$, $\longleftarrow$ transition.pop();
    V[s_t] $\longleftarrow$ V[s_t] + α*(r+γ*V[$s_{t'}$]-V[s_t]);
$\epsilon \longleftarrow \epsilon$*0.5;
$s_t \longleftarrow$ (0,0);
return s_t;

The implementation of this algorithm can be found at [Gaua].

1.3.6 Deep Q-Learning

To overcome the Q-Learning limitation, Deep Q-learning Network (DQN) [MKS+15] makes use of a deep neural network to approximate the Q-value function for potentially high-dimensional or continuous state-space problems. The state is given as the input and the Q-value of all possible actions is generated as the output of the neural network. The Q-value Q_{θ_t} is a vector. A second neural network $\overline{\theta}$ called the Target network (usually a copy of the first Q-network) is used to calculate a target Q-value "y" (the value of the best possible choice in the target network). The

target is used to train the network and compute the loss function.

$$y_t = r_t + \gamma \max_{a_{t+1}} Q_{\overline{\theta_t}}(s_{t+1}, a_{t+1})$$

Q_{θ_t} is updated by a gradient descent on its parameters θ_t:

$$\theta_{t+1} = \theta_t - \alpha \nabla_{\theta_t} Q_{\theta_t}(s_t, a_t)\Big(Q_{\theta_t}(s_t) - y_t\Big)$$

The target network is not updated at each iteration.
A **loss function** is computed during the Q-learning update at iteration t:

$$L_t(\theta_t) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\Big(y_t - Q_{\theta_t}(s_t, a)\Big)^2\right]$$

The aim of the learning algorithm is not only to maximise the reward, it is above all to learn, and therefore to be able to predict the reward. As the training progresses, the value of the sum of the rewards of each episode should increase while the value of the loss function should decrease and converge to a minimum value.

Lin [Lin91] introduced an important method for speeding up learning called the **experience replay**. In Q-Learning, experiments obtained through trials and errors are used only once to adjust the Q-values and then discarded. This is wasteful, as some experiments may be rare and others expensive to obtain. To overcome this problem, the agent's experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ are stored at each time step in a dataset $D_t = \{e_1, ..., e_t\}$, aggregated over many episodes in a replay buffer. Regularly, a learning update is performed on a sample of experiences $(s, a, r, s') \sim U(D)$ randomly drawn from the dataset By doing this, the agent remembers its past experiences, as if it were experiencing again and again the same situation. This method has a second advantage. Reinforcement learning may become unstable or divergent when a neural network is used to represent the Q-values. Each small update of Q can significantly change the agent's policy and the distribution of the data because of correlations in the observation sequence. The experience replay removes these correlations and smoothes the changes in the data distribution.

Figure 1.7 summarises the operation of the DQN architecture. The DQN is trained in several steps over many episodes. It goes through a sequence of operations at each time step. First, the agent selects an $\epsilon - greedy$ action from the current state, executes it in the environment which returns the reward and the next state. This operation is saved in the experience buffer.

A random batch of samples is then formed by recent and older samples. This batch of training data is given as inputs to both networks. The Q-network takes the current state and action of each data sample and predicts the Q-value for that action. The target network takes the next state of each data sample and predicts the best Q-value (target Q-value) of all actions that can be taken from that state. The Q-value, the target Q-value and the observed reward of the data sample are used to calculate the loss to train the Q-network. The processing is repeated for the next time steps. After T time steps, the weights of the Q-network are copied to the target network. The iteration continues until the training ends: the convergence of the loss function is a good indicator of the end of the agent's learning.
For a more complete understanding of reinforcement learning, the reader may refer to [SB18].

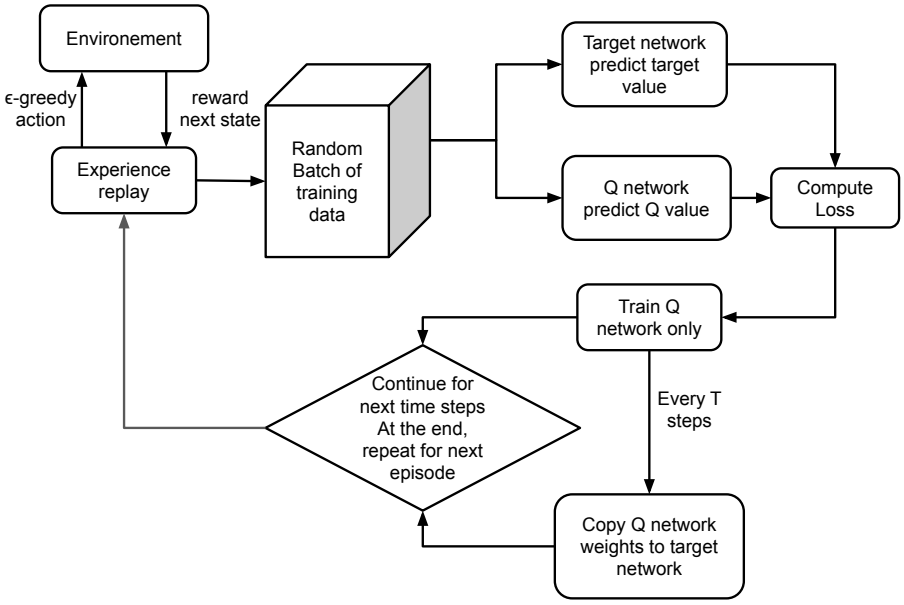


Figure 1.7 – Deep Q Network operation

References

[Dan48] George B Dantzig. Programming in a linear structure. *Washington, DC*, 1948.

[Dan63] George Bernard Dantzig. *Linear Programming and Extensions*. RAND Corporation, Santa Monica, CA, 1963.

[DDS05] Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon, editors. *Column Generation*. Number 978-0-387-25486-9 in Springer Books. Springer, May 2005.

[Gaua] Adrien Gausseran. Maze qlearning example. https://github.com/AdrienGausseran/MazeQLearningExample.

[Gaub] Adrien Gausseran. Multi commodity flow column generation. https://github.com/AdrienGausseran/MultiCommodityFlow_ColumnGeneration.

[Kar84] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, Dec 1984.

[Kha80] L.G. Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980.

[Lin91] Long-Ji Lin. Programming robots using reinforcement learning and teaching. In *Proceedings of the Ninth National Conference on Artificial Intelligence - Volume 2*, AAAI'91, page 781–786. AAAI Press, 1991.

[MKS+15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[WD92] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 1992.

CHAPTER 2

Service Function Chain Placement

In this first chapter we study the problem of SFCs placement. The main contribution of this thesis deals with reconfiguration, but before reconfiguring requests they must be placed on the network. Two placement problems are studied in this section. The dynamic placement problem of a single SFC on the network must allow to accept automatically and instantaneously a SFC when a request is received. In order to be valid, this placement must take into account the resources already in use on the network. The second problem is the static placement problem. It does not have to be instantaneous, although there may still be a time constraint, but it concerns a set of SFCs and is therefore more difficult to solve.

Part of this chapter is contained in the articles [GTGM19a, GTGM21].

2.1 Introduction

The introduction of SDN and NFV has paved the way for a transformation in network management. SDN decouples the control plane from the data plane by centralising routing management in a controller. Thanks to this centralisation of intelligence, the controller has a global view of the network and can make the network management programmable and dynamic [KF13]. NFV allows network functions to be virtualised within generic hardware. This gives network operators greater freedom to customise their networks and offers a chance to reduce both Capex and Opex. Network management thus becomes more flexible and even more dynamic [M. 13]. A more complete description of SDN and NFV is given in subsection 0.2.1 and subsubsection 0.2.2.1.

SDN centralises control of routing and is an important technology for optimising routing and increasing the acceptance of requests. NFV, on the other hand, allows control over the placement of functions and is therefore more interesting in optimising the scaling of these functions or in reducing costs. However, the joint use of these two technologies leads to synergy [Fou15], both in the independent optimisation of the above-mentioned objectives and in multi-ojective optimisations.

This improvement in the management of routes and network functions makes it easier to handle end-to-end service requests through ordered chains of network functions [00114]. This notion of service is called Service Function Chaining (SFC) [QN15] and allows the modelling of various service requests by changing the order of the functions as well as the bandwidth or delay constraints linked to the path. A more complete description of SFC is given in subsubsection 0.2.2.2.

Many works are mainly focused on optimising the placement of VNFs, omitting route optimisation and reducing routing to a simple constraint [CLBB+15, SJG+17]. Yet routing optimisation is an important factor in avoiding network congestion and improving request acceptance [MTC+19] as reported by [FAPZ11], 99% of rejections were caused by bandwidth shortage even though there were enough resources to satisfy the request. Jointly optimising routing and function placement is therefore an advantageous strategy to consider.

In this context, a fundamental problem that arises is how to map these VNFs to nodes (servers) in the network to satisfy all demands, while routing them through the right sequence of functions and meeting service level agreements. In doing this, the capacity constraints on both nodes and links must be respected.

In this chapter, we consider the problem of providing, for each demand, a path through the network in the case of dynamic traffic while respecting capacities on both links and nodes. Moreover, the problem also consists in provisioning VNFs in order to ensure that the traversal order of the network functions by each path is respected. Our goal is to minimise the network operational cost, defined as the sum of the bandwidth cost to route the demands and the cost for all the VNFs running in the network. We present in this chapter, several layer graph based placement models and compare one of them with a state of the art ILP. Our goal is to show that this model can easily transform the allocation and routing problem into a routing problem, without compromising on computation time.

2.2 Related Work

The problem of how to deploy and manage network services conceived as a chain of VNFs has received a significant interest in the research and industrial community. We refer to [HB16] and [M+16] for comprehensive surveys on the relevant state of the art. Some solutions are designed to work on traditional networks that don't use SDN, like in [WBIC19] where the authors use segment routing. Their solution is to totally distribute the choice of functions to be used on the VNF server. When a packet is processed by a network function, the router to which the function is connected decides where the next fonction will be located according to the function by which the packet must be executed. Other solutions are based on SDN and a lot of works have been conducted to develop efficient solutions that respect the functions chaining constraints [KLLT18, TGHP18]. In [SMGZ17] the authors study the placement of SFCs in order to consolidate VNFs by reducing the use of different servers to improve energy efficiency. Their solution is based on the Monte Carlo search tree method to select the best nodes and then optimise the communication between the nodes, but they do not take into account the delays.

The authors of [SJG+17] study the placement of SFCs with splitable requests. Each request is a flow that can go through several paths and can be executed by several instances of the same VNF. Their goal is to minimise the number of VNF instances and they are not interested in minimising links at all. They propose an ILP and two approximation algorithms.

Addis *et al.* [ABBS15] focus on the SFC placement problem and propose two solutions, a MILP and a heuristic. Two objectives are used separately for the MILP: a minimisation of instantiated VNFs or a minimisation of the bandwidth usage. Their heuristic is multi-objective and compromised by first minimising the maximum link utilisation. It then uses the solution found as a parameter to minimise the number of instantiated VNFs. Both approaches improve performance in accordance with the intended objective. In [MTC+19] the authors study the problem of placing SFCs dynamically on single and multi-tenant scenarios. Their objective is to maximise the acceptance of SFCs. It is characterised by minimising three parameterised factors, bandwidth, and node capacities. It allows to prioritize the use of resourceful links and nodes, and to reserve the more loaded ones for potential future heavy requests. They implement an ILP and a heuristic that divides the network into multiple clusters and computes an offline abstraction of the network. They compare their solution to a solution aiming only at minimising bandwidth and obtain an average of 5% to 10% better performance.

The last two works take into account potential compression/decompression functions that modify the bandwidth requirement between two functions. They also take into account different delay constraints between certain types of functions. Bandwidth and delay requirements are therefore consider for the all SFC and locally (between VNFs).

In this chapter we compare our layered graph modelling with the modelling of Morin *et al.* [MTC+19]. Although their work focuses on the study of the objective, we only keep the ILP constraints. In addition, some constraints such as delay constraints are removed and only the constraints that allow to model an SFC taking into account the capacity of links and nodes are kept. All the constraints removed from their ILP can be added to our model, but we maintain the model as simple as possible.

$G = (V, E)$	the network where V represents the set of nodes and E the set of links.
C_{uv}	capacity of a link $(u, v) \in E$ expressed as its total bandwidth available.
C_u	available resources* such as CPU, memory, and disk of a node $u \in V$.
Δ_f	number of cores required per unit of bandwidth required by the function $f \in F$.
$c_{u,f}$	installation cost of the function $f \in F$ which also depends on the node u.
(v_s, v_d, c_d, bw_d)	each demand $d \in D$ is modeled by a quadruple with v_s the source, v_d the destination, c_d the ordered sequence of network functions that need to be performed, and bw_d the required units of bandwidth.
$Succ(f \in c_d)$	Outgoing neighbouring VNF of $f \in c_d$ form demand $d \in D$.

Table 2.1 – Notation used throughout the chapter

2.3 Problem Statement and Notations

We model the network as a directed graph $G = (V, E)$, where V represents the set of nodes and E the set of links. Both nodes and links have associated a capacity. The capacity of a link $(u, v) \in E$ is denoted by C_{uv} and defines the total bandwidth of the link. For a node $u \in V$, the capacity C_u denotes the available resources such as CPU, memory, and disk; it is expressed as the number of CPU cores. For this purpose, given the set of VNFs F, each $f \in F$ has associated a value Δ_f defining the number of cores required by function f per unit of bandwidth. Also, each function f has associated an installation cost $c_{u,f}$ which also depends on the node u. D represents the set of demands. Each demand $d \in D$ is modeled by a quadruple with v_s the source, v_d the destination, c_d the ordered sequence of network functions that need to be performed, and bw_d the required units of bandwidth. Table 2.1 defines the notation used throughout this chapter.

The optimisation task consists in routing each demand while *minimising the network operational cost* defined in terms of bandwidth and VNFs cost (licenses, energy consumption, etc).

2.4 Layered Graph

2.4.1 Layered Graph

Similarly as in [HJG18], in order to model the chaining constraints of a demand, we associate to each demand d a layered graph $G^L(d)$. See Figure 2.1 for an example of a graph with three layers. Representing the original graph as a layered graph is a *modeling trick* first proposed in [DW16]. It allows to simplify the problem by reducing it to a routing problem with shared capacities. This allows a drastic reduction of computation time compared to usual strategies using a large number of binary variables due to the ordering constraints of SFCs. The principle is to consider as many

*A node u with a strictly positive number of cores (i.e., $C_u \in \mathbb{N}^+ = \{1, 2, \cdots\}$) represents a cloud location with the capability to execute VNFs, while a node with $C_u = 0$ is a node that serves only as an SDN router.

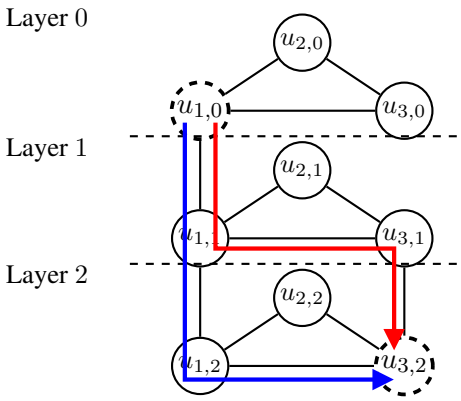


Figure 2.1 – The layered network $G^L(d)$ associated with a demand d such that $v_s = u_1$, $v_d = u_3$, and $c_d = f_1, f_2$, within a triangle network. f_1 is allowed be installed on u_1 and f_2 on u_1 and u_3. Source and destination nodes of $G^L(d)$ are $u_{1,0}$ and $u_{3,2}$. Two possible SFCs that satisfy d are drawn in red (f_1 is in u_1, f_2 in u_3) and blue (f_1 and f_2 are in u_1).

copies of the network as VNFs in an SFC plus one. Copies of a node in a layer are then connected to the ones in the above and below layers with a *vertical link*. Using a horizontal link in a layer corresponds to the use of a physical network link, when using a vertical link joining layers represents the use a virtual function in the corresponding node. Layered graphs can be used to solve different problems: (i) determine the placement and activation of NFVs, and the routing of demands; (ii) If the placement of NFVs has already been done, determine their activation and the routing of demands.

We denote by $u_{i,l}$ the copy of node u_i in layer l. The path for demand d starts from node $v_{s,0}$ in layer 0 and ends at node $v_{d,|c_d|}$ in layer $|c_d|$ where $|c_d|$ denotes the number of VNFs in the chain of the demand.
Given a link (u_i, v_j), each layer l has a link $(u_{i,l}, v_{j,l})$ defined. This property does not hold for links of the kind $(u_{i,l}, u_{i,l+1})$. Indeed, a node may be enabled to run only a subset of the virtual functions. To model this constraint, given a demand d we add a link $(u_{i,l}, u_{i,l+1})$ only if Node u is enabled to run the $(l+1) - th$ function of the chain of d. The $l - th$ function of the chain of d are denoted by $f_l^{c_d}$.
A path on the layered graph corresponds to an assignment to a demand of both a path and the locations where functions are being run. Using a link $(u_{i,l}, v_{j,l})$ on G^L, implies using link (u, v) on G. On the other hand, using link $(u_{i,l}, u_{i,l+1})$ implies using the $(l+1) - th$ function of the chain at node u. Capacities of both nodes and links are shared among layers.

2.5 Static routing and provisioning problem (R&P)

2.5.1 State of the Art ILP formulation `state-of-art-ILP`

Model.
This ILP which we call `state-of-art-ILP` is derived from [MTC+19], with a modification of the objective and the deletion of constraints not used in this context.
It takes as an input the set of demands D.

Variables:
• $\varphi_{uv}^{f,f',d} \in \{0,1\}$ where $\varphi_{uv}^{f,f',d} = 1$ if Link (u,v) is used by demand d between VNF f and f'.
• $\alpha_u^{f,d} \in \{0,1\}$ where $\alpha_u^{f,d} = 1$ if Node u is used by VNF f from demand d.
• $z_{u,f} \in \{0,1\}$, where $z_{u,f} = 1$ if function f is activated on Node u.
Objective: minimise the amount of network resources consumed.

$$\min \quad \sum_{d \in D} \sum_{(u,v) \in E} \sum_{i=0}^{|c_d|-1} bw_d \cdot \varphi_{uv}^{f_i,f_{i+1},d} + \beta \cdot \sum_{u \in V} \sum_{f \in F} c_{u,f} \cdot z_{u,f}$$

Constraints:
Entry and Exit. The entry and exit nodes of the chain are represented by two fictive VNFs (inF and $outF$) without any resource needs, so $\alpha_{v_s}^{inf,d} = 1$ and $\alpha_{v_d}^{outf,d} = 1$.
Path continuity. For each Node $u \in V$, Demand $d \in D$, VNF number $i \in \{0, ..., |c_d| - 1\}$

$$\sum_{(v,u) \in \omega^-(u)} \varphi_{vu}^{f_i,f_{i+1},d} - \sum_{(u,v) \in \omega^+(u)} \varphi_{uv}^{f_i,f_{i+1},d} + \alpha_u^{f_i,d} - \alpha_u^{f_i+1,d} = 0. \tag{2.1}$$

Node capacity constraints. The capacity of a node u in V is shared between each path between VNFs and cannot exceed C_u. For each Node $u \in V$.

$$\sum_{d \in D} bw_d \sum_{i=0}^{|c_d|-1} \Delta_{f_i} \cdot \alpha_u^{f_i,d} \leq C_u. \tag{2.2}$$

Link capacity constraints. The capacity of a link $(u,v) \in E$ is shared between each path between VNFs and cannot exceed C_{uv}. For each Link $(u,v) \in E$.

$$\sum_{d \in D} bw_d \sum_{i=0}^{|c_d|-1} \varphi_{uv}^{f_i,f_{i+1}d} \leq C_{uv}. \tag{2.3}$$

Functions activation. To know which functions are activated on which nodes. For each Node $u \in V$, Function $f \in F$, Demand $d \in D$, VNF number $i \in \{0, ..., |c_d| - 1\}$.

$$\alpha_u^{f_i,d} \leq z_{u,f_i} \tag{2.4}$$

Location constraints. A node may be enabled to run only a subset of the virtual network functions. For each Demand $d \in D$, Node $u \in V$, VNF number $i \in \{0, ..., |c_d| - 1\}$, if the $(i+1) - th$ function of c_d cannot be installed on Node u, we add the following constraint.

$$\alpha_{u,i}^d = 0 \tag{2.5}$$

2.5.2 Our ILP formulation `layer-ILP`

To solve the static R&P (Routing and Provisioning) problem in which a routing and a provisioning of VNF is given for each SFC, we use the ILP (`layer-ILP`) given below.`layer-ILP` routes the demands by finding a path on the layered graph for each of them. In doing this, both node and link capacities must be respected as they are shared among all the demands. The ILP has the minimisation of the network operational cost (i.e., bandwidth cost and network function activation cost) as an objective. As network functions can be shared, the ILP tries to activate a small number

of network functions. The parameter $\beta \geq 0$ specified by the network administrator accounts for different scales over which the functions' activation cost is put in relationship with the network bandwidth cost. β represents how many *TB/s* of data can be sent when using a dollar. Its dimension thus is $TB/dollars$, giving that our objective function formally expresses a bandwidth.

Model.
`layer-ILP` takes as an input the set of demands D. The output corresponds to the *minimum cost* SFC-R&P.
Variables:
• $\varphi^d_{uv,i} \geq 0$ is the amount of flow on Link (u, v) in Layer i for Demand d.
• $\alpha^d_{u,i} \geq 0$ is the fraction of flow of Demand d using Node u in Layer i.
• $z_{u,f} \in \{0, 1\}$, where $z_{u,f} = 1$ if function f is activated on Node u.

Objective: minimise the amount of network resources consumed.

$$\min \quad \sum_{d \in D} \sum_{(u,v) \in E} \sum_{i=0}^{|c_d|} bw_d \cdot \varphi^d_{uv,i} + \beta \cdot \sum_{u \in V} \sum_{f \in F} c_{u,f} \cdot z_{u,f}$$

Constraints:
Flow conservation constraints. For each Demand $d \in D$, Node $u \in V$.

$$\sum_{(u,v) \in \omega^+(u)} \varphi^d_{uv,0} - \sum_{(v,u) \in \omega^-(u)} \varphi^d_{vu,0} + \alpha^d_{u,0} = \begin{cases} 1 \text{ if } u = v_s \\ 0 \text{ else} \end{cases} \tag{2.6}$$

$$\sum_{(u,v) \in \omega^+(v)} \varphi^d_{uv,|c_d|} - \sum_{(v,u) \in \omega^-(v)} \varphi^d_{vu,|c_d|} - \alpha^d_{u,|c_d|-1} = \begin{cases} -1 \text{ if } v = v_d \\ 0 \text{ else} \end{cases} \tag{2.7}$$

$$\sum_{(u,v) \in \omega^+(u)} \varphi^d_{uv,i} - \sum_{(v,u) \in \omega^-(u)} \varphi^d_{vu,i} + \alpha^d_{u,i} - \alpha^d_{u,i-1} = 0. (0 < i < |c_d|) \tag{2.8}$$

Node capacity constraints. The capacity of a node u in V is shared between each layer and cannot exceed C_u. For each Node $u \in V$.

$$\sum_{d \in D} bw_d \sum_{i=0}^{|c_d|-1} \Delta_{f^{c_d}_i} \cdot \alpha^d_{u,i} \leq C_u. \tag{2.9}$$

Link capacity constraints. The capacity of a link $(u, v) \in E$ is shared between each layer and cannot exceed C_{uv}. For each Link $(u, v) \in E$.

$$\sum_{d \in D} bw_d \sum_{i=0}^{|c_d|} \varphi^d_{uv,i} \leq C_{uv}. \tag{2.10}$$

Functions activation. To know which functions are activated on which nodes. For each Node $u \in V$, Function $f \in F$, Demand $d \in D$, Layer $i \in \{0, ..., |c_d| - 1\}$.

$$\alpha^d_{u,i} \leq z_{u,f^{c_d}_i} \tag{2.11}$$

Location constraints. A node may be enabled to run only a subset of the virtual network functions. For each Demand $d \in D$, Node $u \in V$, layer $i \in \{0, ..., |c_d| - 1\}$, if the $(i+1) - th$ function of c_d cannot be installed on Node u, we add the following constraint.

$$\alpha_{u,i}^{d} = 0 \tag{2.12}$$

2.6 R&P for a single demand

Note first that even routing a single demand is NP-hard. Indeed, it is equivalent to finding a shortest Weight-Constrained Path [GJ02] in the layered graph as link and node capacities are shared between layers [HJG18]. A solution is to use the ILP for static R&P in which all the demands routed in the past are fixed. The ILP routes the demand (if possible) with the goal of minimising the additional needed cost without exceeding the available network resources. To deal with the already installed network function, the *current cost* $\tilde{c}_{u,f}$ of installing a network function f on a Node u is defined as follows. Let $\mathcal{I}$ be the set with the already installed network function, then $\tilde{c}_{u,f} = 0$ if $(u, f) \in \mathcal{I}$, and $c_{u,f}$otherwise.

ILP takes as an input a demand $d = (v_s, v_d, c_d, bw_d)$ and the network. We denote by R_u the residual capacity of a Node u, and finally by R_{uv} the residual capacity of a link (u, v).

2.6.1 State of the Art ILP formulation, single demand

Variables:
- $\varphi_{uv}^{f,f',d} \in \{0, 1\}$ where $\varphi_{uv}^{f,f',d} = 1$ if Link (u, v) is used by demand d between VNF f and f'.
- $\alpha_{u}^{f,d} \in \{0, 1\}$ where $\alpha_{u}^{f,d} = 1$ if Node u is used by VNF f from demand d.
- $z_{u,f} \in \{0, 1\}$, where $z_{u,f} = 1$ if function f is activated on Node u.

Objective: minimise the amount of network resources consumed.

$$\min \quad \sum_{d \in D} \sum_{(u,v) \in E} \sum_{i=0}^{|c_d|-1} bw_d \cdot \varphi_{uv}^{f_i, f_{i+1}, d} + \beta \cdot \sum_{u \in V} \sum_{f \in F} \tilde{c}_{u,f} \cdot z_{u,f}$$

Constraints:

Entry and Exit. The entry and exit nodes of the chain are represented by two fictive VNFs (inF and $outF$) without any resource needs, so $\alpha_{v_s}^{inf,d} = 1$ and $\alpha_{v_d}^{outf,d} = 1$.

Path continuity. For each Node $u \in V$, Demand $d \in D$, VNF number $i \in \{0, ..., |c_d| - 1\}$

$$\sum_{(v,u) \in \omega^-(u)} \varphi_{vu}^{f_i, f_{i+1}, d} - \sum_{(u,v) \in \omega^+(u)} \varphi_{uv}^{f_i, f_{i+1}, d} + \alpha_{u}^{f_i, d} - \alpha_{u}^{f_i+1, d} = 0. \tag{2.13}$$

Node capacity constraints. The capacity of a node u in V is shared between each layer and cannot exceed R_u. For each Node $u \in V$.

$$\sum_{d \in D} bw_d \sum_{i=0}^{|c_d|-1} \Delta_{f_i} \cdot \alpha_{u}^{f_i, d} \leq R_u. \tag{2.14}$$

Link capacity constraints. The capacity of a link $(u, v) \in E$ is shared between each layer and cannot exceed R_{uv}. For each Link $(u, v) \in E$.

$$\sum_{d \in D} bw_d \sum_{i=0}^{|c_d|-1} \varphi_{uv}^{f_i, f_{i+1} d} \leq R_{uv}. \tag{2.15}$$

2.6.2 Our ILP formulation, single demand

Variables:
- $\varphi_{uv,i} \geq 0$ is the amount of flow on Link (u, v) in Layer i.
- $\alpha_{u,i} \geq 0$ is the fraction of flow of the demand using Node u in Layer i at time step t.

Objective: minimise the additional increase in terms of network operational cost.

$$\min \quad \sum_{(u,v)\in E} \sum_{i=0}^{|c_d|} bw_d \cdot \varphi_{uv,i} + \beta \cdot \sum_{u\in V} \sum_{i=0}^{|c_d|-1} \tilde{c}_{u,f_i^{c_d}} \cdot \alpha_{u,i}$$

Constraints:
Flow conservation constraints. For each Node $u \in V$.

$$\sum_{(u,v)\in\omega^+(u)} \varphi_{uv,0} - \sum_{(v,u)\in\omega^-(u)} \varphi_{vu,0} + \alpha_{u,0} = \begin{cases} 1 \text{ if } u = v_s \\ 0 \text{ else} \end{cases} \tag{2.16}$$

$$\sum_{(u,v)\in\omega^+(v)} \varphi_{uv,|c_d|} - \sum_{(v,u)\in\omega^-(v)} \varphi_{vu,|c_d|} - \alpha_{u,|c_d|-1} = \begin{cases} -1 \text{ if } v = v_d \\ 0 \text{ else} \end{cases} \tag{2.17}$$

$$\sum_{(u,v)\in\omega^+(u)} \varphi_{uv,i} - \sum_{(v,u)\in\omega^-(u)} \varphi_{vu,i} + \alpha_{u,i} - \alpha_{u,i-1} = 0. \qquad 0 < i < |c_d| \tag{2.18}$$

Node capacity constraints. The capacity of a node u in V is shared between each layer and cannot exceed the residual capacity R_u. For each Node $u \in V$.

$$bw_d \sum_{i=0}^{|c_d|-1} \Delta_{f_i^{c_d}} \cdot \alpha_{u,i} \leq R_u. \tag{2.19}$$

Link capacity constraints. The capacity of a link $(u, v) \in E$ is shared between each layer and cannot exceed the residual capacity R_{uv}. For each Link $(u, v) \in E$.

$$bw_d \sum_{i=0}^{|c_d|} \varphi_{uv,i} \leq R_{uv}. \tag{2.20}$$

2.7 Weight Constrained Shortest Path based heuristic

Another possibility that we have studied is to adapt the pseudo-polynomial algorithms proposed for the shortest Weight-Constrained Path problem such as the Label-setting algorithm based on dynamic programming [ID05]. We present in this section the algorithms implemented for the static problem. Although the solution in subsection 2.6.2 is a ILP, the execution time for a single demand is negligible (less than 10ms for the largest networks tested). This heuristic (Algorithm 4) was

therefore made for the static placement problem for a large number of SFCs when the execution time of subsection 2.5.2 is too long. It calls the sub-procedures 5, 6, and 7 described in the following subsections.

2.7.1 Algorithm 4: Finding a good static placement

Algorithm 4 is the algorithm called to find a good placement for a set of SFCs. It takes as input the graph G, the list of SFCs to be placed, the list of functions, the β parameter of the objective, and a parameter *nbVnfToShut*. *nbVnfToShut* represents the maximum number of VNFs to try to turn off to improve the objective, the larger it is the more time the algorithm takes, but the greater the chance of a good placement there is. The principle of the algorithm is to iteratively place each SFC through a configuration of functioning VNFs. As the VNFs are turned off to improve the objective, a new placement is computed for all SFCs. In line 2 the layer graph G' is created and in line 3, the variable *adj* is pointing to the adjacency dictionary of G'.

Between lines 4 and 9 a first placement for the set of SFCs is computed by leaving all VNFs on. This placement is iteratively calculated using Algorithm 5 and the capacities of G' are decreased at each iteration. The parameter β is not given as an input to Algorithm 5, therefore the algorithm only minimise links utilisation (path size). If an allocation is impossible then the algorithm returns false and no allocation for the whole set of SFCs is found. This first allocation and its objective is saved in line 8 and 10. In line 11 the usage of every VNF on every datacenter is recovered from the allocation in addition to the number of every VNF instances. Then these information are given as input to Algorithm 7 which returns a list of VNFs to be turned off (the length of this list is *nbVnfToShut*). This list is added to a stack: *stackToShut*. From line 15 the algorithm iterates until *stackToShut* is empty.

The last list in the stack is removed and a VNF is taken from this list. The list is returned to the stack in line 19. In line 20 it is checked that the current configuration of switched off VNFs (counting the one removed from the list) has not already been tried. If the configuration is not already tried, it is saved and the VNF to be turned off is stack in *stackToTurn* to be turned on later to tried other configurations.

In lines 25 and 26 the VNF is turned off by removing the link in G' and the capacities of G' are reset. Between lines 27 and 33 the placement for the set of SFCs is computed with the new configuration of VNFs turned on. If the allocation is possible, the objective is checked and if it is better than the older one, the allocation is saved and a new list of VNFs to turn off is added to the stack. If the allocation is not possible, in lines 42 and 43 the last VNF turned off is turned back on, and another VNF from the queue will be tested, until *stackToShut* is empty. The best placement is then returned.

2.7.2 Algorithm 5: Finding the best routing

Algorithm 5 is the algorithm of the Weight Constrained Shortest Path problem and it is called to find the best placement for a single SFC for a configuration of activated VNFs. It takes as input *adj* (the adjacency dictionary of G'), the list of SFCs to be placed, the list of functions, the β parameter of the objective, and a parameter *nbVnfToShut*. The algorithm does not need G' because *adj* is already pointing to its links. This algorithm uses labels: a label is the identification of a path from the source of the SFC to a given node of G'. A label consists of the path, its cost, *resourceUsed* (a dictionary of resources used by the path), and a list of pointers to its child labels.

Algorithm 4: FindGoodAllocation

```
Data: G, listSfc, functions, β, nbVnfToShut
Result: Find a good allocation for a list of SFCs
bestAlloc, currentAlloc ⟵ void dictionaries;
G' ⟵ Create layer graph of G;
adj ⟵ Adjacency dictionary of G';
for sfc in listSfc do
    possibleRouting, routing ⟵ FindRouting(adj, sfc, 0);
    if not possibleRouting then
        return False, void;
    bestAlloc[sfc] ⟵ routing;
    G'.updateCapacity(routing);
bestObj ⟵ Objective of bestAlloc;
usedDC, nbVnfPresent ⟵ Datacenter usage and number of each vnf used by bestAlloc;
listVnfToShut ⟵ FindVnfToShut(usedDC, nbVnfPresent, nbVnfToShut);
stackToShut, stackToTurn, configurationsTried ⟵void lists;
stackToShut.push(listVnfToShut);
while stackToShut.length > 0 do
    currentList ⟵ stackToShut.pop();
    if currentList.length > 0 then
        (v,f) ⟵ currentList.pop();
        stackToShut.push(currentList);
        if currentConfiguration in configurationsTried then
            continue;
        else
            configurationsTried.append(currentConfiguration);
        stackToTurn.push((v, f));
        G'.removeVnf(v,f);
        G'.resetCapacities();
        for sfc in listSfc do
            possibleRouting, routing ⟵ FindRouting(adj, sfc, 0);
            if not possibleRouting then
                break;
            currentAlloc[sfc] ⟵ routing;
            G'.updateCapacity(routing);
        if possibleRouting then
            obj ⟵ Objective of currentAlloc;
            if obj < bestObj then
                bestObj ⟵ obj;
                bestAlloc ⟵ currentAlloc;
                usedDC, nbVnfPresent ⟵ Datacenter usage and number of each vnf
                  used by currentAlloc;
                listVnfToShut ⟵ FindVnfToShut(usedDC, nbVnfPresent,
                  nbVnfToShut);
                stackToShut.push(listVnfToShut);
    else
        (v,f) ⟵ stackToTurn.pop();
        G'.addVnf(v,f);
return True, bestAllocation;
```

For each link or datacenter used by the path, *resourceUsed* records the amount of resources used for that link/datacenter.

An important property of labels is dominance, represented by Algorithm 6. One label dominates another if its cost is lower and if for each resource used, the second label consumes at least as much.

At the start of Algorithm 5 a first label is created, starting from the source of the SFC, and is placed on the stack *labelToTreat*. The algorithm iterates until the stack is empty. When a label is taken from the stack, if it is still active (none of the ancestor is dominated), the algorithm iterates on its adjacent edges. For every edge, the capacity is checked (the multiplier is greater than 0 if the edge represents a VNF) and a new cost is created.

If the edge is a VNF, β is added to the cost, but is the real β from the objective was used. This algorithm would find the best placement with the best allocation and a big part of Algorithm 4 would be useless. On the contrary, in lines 5 and 28 of Algorithm 4 0 is given as a parameter instead of β. By running the algorithm with β instead of 0, the computation time becomes too important and does not allow the use of the algorithm. Instead Algorithm 4 controls the VNF allocations.

In lines 23 to 25, the path and the *resourceUsed* of the new label are created. If the destination is reached, the cost and the path are saved in line 28. The new child label of the current label is created and its dominance is checked against all the labels already saved on the edge in line 33. If the new label is dominated, it is discarded, but if it dominates a label, this one is turned off (and all of its childs are turned off recursively). The new label is then added to the edge and to the stack. At the end, the algorithm returns the best placement for the SFC.

2.7.3 Algorithm 7: Choosing the VNFs to turn off

Algorithm 7 is the algorithm used to find a list of VNFs to turn off. It takes as input *UsedDC*: the usage of every VNF on every datacenter, *nbVnfPresent*, the number of every VNF instances, and *nbVnftoShut*, the number of VNFs to turn off. The objective of the algorithm is to minimise the cost. The cost being defined by the bandwidth used on the paths and the number of VNFs deployed, it is relevant to try to turn off as many VNFs as possible.

To turn off a VNF, the algorithm takes the least used VNF as long as it is not the last instance of a function.

2.8 Numerical Results for `layer-ILP` and `state-of-art-ILP`

In this section we compare the efficiency of `layer-ILP` and `state-of-art-ILP` to observe whether modelling using the layered graph improves or impairs computation times.

We conduct experiments on two real-world topologies from SNDlib [OWPT10] of different sizes: `pdh` (11 nodes, 34 links), and `ta1` (24 nodes, 55 links). We generate our problem instances as follows. To create the scenarios we vary the number of VNFs used by the demands and the number of SFCs to be placed. Each demand is associated with an ordered sequence of 4 to 6 functions (depending on the scenario) uniformly chosen at random from a set of 6 different functions. Each demand set contains 50 or 100 SFCs to be placed.

To identify the difference in performance between the two algorithms each scenario is run with different maximum computation times, 5, 10, 30, 60 and 300 seconds. At the end of the simulation, if the two algorithms have given a solution, their two objectives are compared

and a ratio is calculated. The ratio is the percentage improvement of one algorithm over the other. When the two objectives are equal, the ratio is 0. If the objective of `layer-ILP` is better than `state-of-art-ILP` then the ratio is negative and represents the percentage improvement of `layer-ILP` over `state-of-art-ILP`. Conversely, if the objective of `state-of-art-ILP` is better than that of `layer-ILP`, then the ratio is positive and represents the percentage improvement of `state-of-art-ILP` over `layer-ILP`

Each scenario is performed on 10 instances and the results are summarised in tables 2.2 for `pdh` and 2.3 for `ta1`. For `pdh` the difference between the two solutions is rarely more than 5%: it reaches 8.3% in favour of `layer-ILP` for 100SFCs composed of 6VNFs to be placed in less than 10 seconds. The average ratio is less than 1% difference and the number of instances that did not give a result due to lack of time is 5 for `layer-ILP` as for `state-of-art-ILP`.

For `ta1` again the ratio rarely exceed 5%, it reaches 18.4% in favour of `layer-ILP` for 50 SFCs composed of 5 VNFs in 10 seconds but this gap would surely be reduced if we did a larger number of experiments. The average ratio is less than 1% difference and the number of instances that did not give a result due to time constraints is slightly higher for `layer-ILP` with 58 iterations without result against 56 for `state-of-art-ILP`.

These results do not show any difference in effectiveness between `layer-ILP` and `state-of-art-ILP`.

2.9 Conclusion

After extensive simulations, the heuristic presented in section 2.7 did not prove to be effective. Algorithm 4 when used for a single SFC with the beta parameter does not allow to find a dynamic placement as quickly as the two ILPs from section 2.6.

Algorithm 5, sometimes gives results considerably faster than the ILPs from section 2.5: up to a factor of 10 when the ILPs take more than 600 seconds. Unfortunately the quality of the solutions is very uneven, the objective is up to 2.5 times higher. Although Algorithm 7 turns off the less used VNFs, they are sometimes essential to a good solution and are used a lot more in an optimal solution. In low congestion scenarios where many of the VNFs are not used, turning off the VNFs step by step is often a bad solution. In such a scenario a better heuristic idea would be to maybe on the contrary turn on the VNFs gradually until a possible solution is founded and the objective does not decrease any more.

Furthermore, since the heuristic searches for a solution iteratively for each SFC, it does not always find a solution for a set of SFCs when the ILPs return an optimal solution. Finally, when the problem has many SFCs and no solution, the heuristic can take several seconds to detect it, whereas it takes less than a second for the ILPs.

In this chapter we have seen several ways to model the ordering of network functions within an SFC. The formulation `layer-ILP` developed in this chapter, based on the layer graph, transforms the routing and allocation problem into a routing problem. This formulation, while not improving or degrading execution times, allows for an intuitive modelling of ordered chains and will be used throughout this thesis.

Algorithm 5: FindRouting

Data: adj, sfc, β
Result: Find a weight constrained shortest path in the layer graph

```
labelToTreat ⟵ void stack;
currentLabel ⟵ newLabel(path = [sfc.src_layer0], cost = 0, resourceUsed = dictionary);
bestCost ⟵ +∞ ;
bestRouting ⟵ void;
labelToTreat.push(currentLabel);
while labelToTreat.length > 0 do
    currentLabel ⟵ labelToTreat.pop();
    if not currentLabel.isOn() then
        continue;                                   ▷ # If an antecedent of this label is dominated
    for edge in adj[currentLabel.path.LastNode] do
        if edge.dst in currentLabel.path then
            continue;
        if currentLabel.resourceUsed[edge] + sfc.bandwidth*edge.multiplier >
         edge.capacity then
            continue;                               ▷ # We check the capacity of the edge
        cost ⟵ currentLabel.cost;
        if edge.isLink() then
            cost ⟵ cost + sfc.bandwidth;
        else
            if edge.isVnf() then
                cost ⟵ cost + β ;
        if cost > bestCost then
            continue;          ▷ # We check if a routing with a better cost is already found
        resourceUsed ⟵ currentLabel.resourceUsed;
        resourceUsed[edge] ⟵ resourceUsed[edge] + sfc.bandwidth*edge.multiplier;
        path ⟵ currentLabel.path + edge;                ▷ # We find the destination
        if edge.dst is sfc.dst_lastLayer then
            if cost < bestCost then
                bestCost ⟵ cost;
                bestRouting ⟵ path;
        else
            labelToAdd ⟵ newLabel(path = path, cost = cost, resourceUsed =
             resourceUsed);
            addLabel ⟵ True;
            for label in edge.dst.labelSaved do
                if dominate(label, labelToAdd)          ▷ # If the new label is dominated
                 then
                    addLabel ⟵ False;
                    break;
                if dominate(labelToAdd, label)  ▷ # If the new label dominate the old one
                 then
                    label.turnOff();
            if addLabel then
                labelToTreat.push(labelToAdd);
                edge.dst.labelSaved.append(labelToAdd);
if bestRouting = void then
    return False, bestRouting;
return True, bestRouting;
```

Algorithm 6: dominate

```
Data: label1, label2
Result: Indicates if label1 dominate label2
if label1.cost >= label2.cost then
    return False;
for r in label1.resourceUsed do
    if not r in label2.resourceUsed then
        return False;
    if not label1.resourceUsed[r] >= in label2.resourceUsed[r] then
        return False;
return True;
```

Algorithm 7: FindVnfToShut

```
Data: usedDC, nbVnfPresent, nbVnfToShut
Result: Find the next VNFs to turn off
listVnfToShut ⟵ void list;
for (i=0, i<nbVnfToShut, i++) do
    smallerUse ⟵ +∞ ;
    smallerVnf ⟵ void;
    for v in usedDC do
        for f in usedDC[v] do
            if (usedDC[v][f] < smallerUse) And (nbVnf[f] > 1) And ((v,f) not in
              smallerVnf) then
                smallerUse ⟵ usedDC[v][f];
                smallerVnf ⟵ (v,f);
    listVnfToShut.append(smallerVnf);
    nbVnf[smallerVnf.f] ⟵ nbVnf[smallerVnf.f] - 1;
return listVnfToShut;
```

50 SFCs																
Time	5s			10s			30s			60s			300s			
# of VNFs	4	5	6	4	5	6	4	5	6	4	5	6	4	5	6	
Ratio (%)	0.4	0.4	-1.8	0.7	1.1	0.9	0.1	-0.3	0	0	0	0	0	0	0	
Average Ratio (%)	0.1															
# of instances without solution	layer-ILP: 0 state-of-art-ILP: 0															
100 SFCs																
Time	5s			10s			30s			60s			300s			
# of VNFs	4	5	6	4	5	6	4	5	6	4	5	6	4	5	6	
Ratio (%)	-2.2	0	0.9	-4.0	1.9	-8.3	0	-0.1	-0.1	-2.2	-0.4	0.3	-0.1	-0.4	-0.3	
Average Ratio (%)	-1															
# of instances without solution	layer-ILP: 5 state-of-art-ILP: 5															

Table 2.2 – Efficiency ratio between `layer-ILP` and `state-of-art-ILP` in percentage on the `pdh` network (numbers in blue are in favour of `layer-ILP` and numbers in red are in favour of `state-of-art-ILP`.)

50 SFCs

Time	5s			10s			30s			60s			300s		
# of VNFs	4	5	6	4	5	6	4	5	6	4	5	6	4	5	6
Ratio (%)	2.1	5.8	-4.4	-0.2	-18.4	-0.4	-0.5	-0.2	-0.1	-0.9	0.1	1.0	-0.1	0	0
Average Ratio (%)	-1.1														
# of instances without solution	layer-ILP: 0 state-of-art-ILP: 0														

100 SFCs

Time	5s			10s			30s			60s			300s		
# of VNFs	4	5	6	4	5	6	4	5	6	4	5	6	4	5	6
Ratio (%)	7.7	0	0	3.7	0	0	0.3	-0.8	2.3	1.3	-3.2	2.1	-0.7	-2.1	-0.3
Average Ratio (%)	0.7														
# of instances without solution	layer-ILP: 58 state-of-art-ILP: 56														

Table 2.3 – Efficiency ratio between `layer-ILP` and `state-of-art-ILP` in percentage on the `ta1` network (numbers in blue are in favour of `layer-ILP` and numbers in red are in favour of `state-of-art-ILP`.)

References

[00114] ETSI GS NFV-MAN 001. Etsi gs nfv-man 001 v1.1.1 (2014-12)network functions virtualisation (nfv);management and orchestration. https://www.etsi.org/deliver/etsi_gs/nfv-man/001_099/001/01.01.01_60/gs_nfv-man001v010101p.pdf, 12 2014.

[ABBS15] Bernardetta Addis, Dallal Belabed, Mathieu Bouet, and Stefano Secci. Virtual network functions placement and routing optimization. In *2015 IEEE 4th International Conference on Cloud Networking (CloudNet)*, pages 171–177, 2015.

[CLBB+15] Marcelo Caggiani Luizelli, Leonardo Bays, Luciana Buriol, Marinho Barcellos, and Luciano Gaspary. Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions. 05 2015.

[DW16] Abhishek Dwaraki and Tilman Wolf. Adaptive service-chain routing for virtual network functions in software-defined networks. In *Proceedings of the 2016 workshop on Hot topics in Middleboxes and Network Function Virtualization*, pages 32–37, 2016.

[FAPZ11] Ilhem Fajjari, Nadjib Aitsaadi, Guy Pujolle, and Hubert Zimmermann. VNR algorithm: A greedy approach for virtual networks reconfigurations. In *IEEE Global Telecommunications Conference - GLOBECOM*, pages 1–6. IEEE, 2011.

[Fou15] Open Networking Foundation. Tr-518 relationship of sdn and nfv. https://opennetworking.org/wp-content/uploads/2014/10/onf2015.310_Architectural_comparison.08-2.pdf, 10 2015.

[GJ02] Michael R Garey and David S Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002.

[HB16] Juliver Gil Herrera and Juan Felipe Botero. Resource allocation in NFV: A comprehensive survey. *IEEE Transactions on Network and Service Management (IEEE TNSM)*, 13(3):518–532, 2016.

[HJG18] Nicolas Huin, Brigitte Jaumard, and Frédéric Giroire. Optimal network service chain provisioning. *IEEE/ACM Transactions on Networking (ToN)*, 26(3):1320–1333, June 2018.

[ID05] Stefan Irnich and Guy Desaulniers. Shortest path problems with resource constraints. In *Column generation*, pages 33–65. Springer, 2005.

[KF13] Hyojoon Kim and Nick Feamster. Improving network management with Software Defined Networking. *IEEE Communications Magazine*, 51(2):114–119, 2013.

[KLLT18] Tung-Wei Kuo, Bang-Heng Liou, Kate Ching-Ju Lin, and Ming-Jer Tsai. Deploying chains of virtual network functions: On the relation between link and server usage. *IEEE/ACM Transactions on Networking (TON)*, 26(4):1562–1576, 2018.

[M. 13] M. Chiosi *et al.* Network functions virtualisation (NFV) network operator perspectives on industry progress. In *SDN & OpenFlow World Congress*, Dusseldorf, Germany, October 2013.

[M+16] Rashid Mijumbi et al. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 18(1):236–262, 2016.

[MTC+19] Cédric Morin, Geraldine Texier, Christelle Caillouet, Gilles Desmangles, and Cao-Thanh Phan. Vnf placement algorithms to address the mono-and multi-tenant issues in edge and core networks. In *2019 IEEE 8th International Conference on Cloud Networking (CloudNet)*, pages 1–6, 2019.

[OWPT10] Sebastian Orlowski, Roland Wessäly, Michal Pióro, and Artur Tomaszewski. Sndlib 1.0—survivable network design library. *Networks: An International Journal*, 55(3):276–286, 2010.

[QN15] P. Quinn and T. Nadeau. Problem statement for service function chaining. RFC 7498, RFC Editor, April 2015.

[SJG+17] Yu Sang, Bo Ji, Gagan R Gupta, Xiaojiang Du, and Lin Ye. Provably efficient algorithms for joint placement and allocation of virtual network functions. In *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 1–9. IEEE, 2017.

[SMGZ17] Oussama Soualah, Marouen Mechtri, Chaima Ghribi, and Djamal Zeghlache. Energy efficient algorithm for vnf placement and chaining. pages 579–588, 05 2017.

[TGHP18] Andrea Tomassilli, Frédéric Giroire, Nicolas Huin, and Stéphane Pérennes. Provably efficient algorithms for placement of Service Function Chains with ordering constraints. In *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 774–782, Honolulu, Hawai, US, 2018. IEEE.

[WBIC19] Adrien Wion, Mathieu Bouet, Luigi Iannone, and Vania Conan. Change in continuity: Chaining services with an augmented igp. *IEEE Transactions on Network and Service Management*, 16(4):1332–1344, 2019.

CHAPTER 3

Service Function Chains Reconfiguration

Software Defined Networking (SDN) and Network Function Virtualisation (NFV) are complementary and core components of modernised networks. In this chapter, we consider the problem of reconfiguring Service Function Chains (SFC) with the goal of bringing the network from a sub-optimal to an optimal operational state.
We propose optimisation models based on the *make-before-break* mechanism, in which a new path is set up before the old one is torn down. Our method takes into consideration the chaining requirements of the flows and scales well with the number of nodes in the network.
We show that, with our approach, the network operational cost defined in terms of both bandwidth and installed network function costs can be reduced and a higher acceptance rate can be achieved, while not interrupting the flows.

This chapter is part of the articles [GTGM19b, GTGM19a, GTGM21].

3.1 Introduction

The last decade has seen the development of new paradigms to pave the way for a more flexible, open, and economical networking. In this context, SDN and NFV are two of the most promising technologies for the Next-Generation Network.

- SDN which aims at simplifying network management by decoupling the control plane from the data plane. See subsection 0.2.1 for more details on SDN.
- NFV which allows network functions (e.g., firewall, load balancer, content filtering or deep packet inspection) to be implemented in software and executed on generic-purpose servers located in small cloud nodes. See subsubsection 0.2.2.1 for more details on NFV.

Even though SDN and NFV are two different independent technologies, they are complementary. Each one can leverage off the other to improve networks and service delivery over them [MGT+15]. Service Function Chaining (SFC) [QN15] can take advantage of these technologies, with VNFs being strategically placed to reach the service destination, and SDN allowing VNFs to be chained easily together. See subsubsection 0.2.2.2 for more details on SFC.

Reconfiguring The network state changes continually due to the arrival and departure of flows. Moreover, the allocation of a demand is performed individually without having full knowledge of the incoming traffic. This may lead to a sub-optimal utilization of the resources of the network. For instance, requests may be routed on long paths, and there may be more active network functions than needed. An optimal or near-optimal resource allocation may result after a lapse of time in over-provisioning or in an inefficient resource usage. Also, it may lead to a higher blocking probability even though there are enough resources to serve new demands. Indeed, as reported by [FAPZ11], 99% of rejections were caused by bandwidth shortage even though there were enough resources to satisfy the request.

Therefore, operators must take it into consideration and adjust network configurations in response to changing network conditions to fully exploit the benefits of the SDN and NFV paradigms, and to avoid undue extra cost (e.g., software licenses, energy consumption, and Service Level Agreement (SLA) violation).

Thus, another problem is how to reroute traffic flows through the network and how to improve the mapping of network functions to nodes. In order to minimise the network's operator cost and to optimise the usage of network resources, we consider the problem of reconfiguring regularly the demands, i.e., moving them from a *local optimal* allocation to a *global optimal* one.

Make-Before-Break Reconfiguration can be performed at several moments in time. It can be done as soon as a new request arrives [GR18], when a request is rejected [TTG13], when the physical network is modified [CLX+10], or it could also be done periodically when the network is not yet saturated.

Rerouting demands and migrating VNFs may take several time steps. If during this time, traffic is interrupted, it may have a non-negligible impact on the QoS experienced by the users. To tackle this issue, our strategy performs the reconfiguration by using a two–phase approach. First, a new route for the transmission is established while keeping the initial one enabled (i.e., two redundant data streams are both active in parallel), and after the network has been updated to the new state, the transmission moves on the new route and the resources used by the initial one

are released. This strategy is often referred to as *make-before-break*.
See subsection 0.2.4 and subsubsection 0.2.4.2 for more details on reconfiguration and the *make-before-break* technique.

The results of our numerical evaluations lead to the following conclusions.

- `Break-Free-ILP` allows to *reduce the network cost* and *increase the acceptance rate*. It can achieve, in most of the considered cases, a gain close to the one of a reconfiguration algorithm that interrupts the requests (referred to as `Breaking-Bad` in the following), as proposed in the literature.
- It is important to consider mechanisms limiting the impact on the demands. Indeed, as we show in subsection 3.5.7, the percentage of demands which have to be rerouted to achieve a significant gain in terms of network cost or acceptance rate may be very high.
- Network reconfiguration needs to be performed frequently in order to achieve a significant gain. However, this reconfiguration can be quickly computed and carried out, making it possible to be put into practice in real time.

The rest of this chapter is organized as follows. In section 3.2, we discuss related work. In section 3.3, we formally state the problem addressed in this Chapter. The section 3.4 presents the optimisation framework and develops the optimisation models for reconfiguring the network. In section 3.5, we validate our proposed optimisation models by various numerical results on two real–world network topologies of different sizes. Finally, we draw our conclusion in section 3.6.

3.2 Related Work

The problem of how to deploy and manage network services conceived as a chain of VNFs is summarised in chapter 2

Although a lot of effort has been made to develop efficient strategies to route demands and satisfy their chaining requirements [KLLT18, TGHP18], not enough has been made to improve resources usage during network operation.

Recently, some research work has started to explore SDN capabilities for a more efficient usage of the network resources by dynamically adapting the routing configuration over time. For instance, Paris *et al.* [PDM+16] study the problem of online SDN controllers to decide when to perform flow reconfigurations for efficient network updating such that the flow reallocation cost is minimised. However, the network function requirements are not considered in their work. Indeed, the traffic of a request may need to be steered to traverse middleboxes implementing the required network functions.

In [NKT19], Noghani *et al.* study the trade-off between the reconfiguration of SFCs and the optimality of the reconfigured routing and placement solution.

In [HFS+19], Harutyunyan et al proposes a MILP to solve the problem of slices embedding, modeled by SFCs. They compare different placement strategies and study their trade-offs. They then propose a slice embedding heuristic to minimise the number of vnf migrations.

Wei et al [WFS+20] proposes a slice reconfiguration algorithm in the core network exploiting Deep Reinforcement Learning to maximise the use of network resources. Deep Reinforcement Learning is used to predict when to make reconfigurations. A reconfiguration consists in re-routing the slice from the VNFs it uses to other VNFs, while taking into account a reconfiguration cost.

The closest study to our work is from Liu *et al.* [LLZ+17]. They consider the problem of optimising VNFs deployment and readjustment to efficiently orchestrate dynamic demands. When a new request arrives, the service provider can serve it or change the provisioning schemes of the already deployed ones at time instances with a fixed interval in between. They consider the maximisation of the service provider's profit which is the total profit from the served requests minus the total deployment cost as an optimisation task. For this purpose, they formulate an Integer Linear Programming (ILP) model. Then, to reduce the time complexity, they design a column generation model. An important unaddressed issue concerns the revenue loss of an operator due to the QoS degradation occurring when demands are reconfigured [EMAL17]. Indeed, in their model, transmissions may need to be interrupted in order to be moved to the new computed state. Different from the above mentioned works, our aim is to provide efficient mechanisms to dynamically reallocate the demands *without the consequential QoS deterioration due to the traffic interruption, but instead using make-before-break strategy*.

The Table 3.1 presents the main differences of the publications presented in this related work.

3.3 Problem Statement and Notations

We model the network as a directed graph $G = (V, E)$, where V represents the set of nodes and E the set of links. Both nodes and links have associated a capacity. The capacity of a link $(u, v) \in E$ is denoted by C_{uv} and defines the total bandwidth of the link. For a node $u \in V$, the capacity C_u denotes the available resources such as CPU, memory, and disk; it is expressed as the number of CPU cores. For this purpose, given the set of VNFs F, each $f \in F$ has associated a value Δ_f defining the number of cores required by function f per unit of bandwidth. Also, each function f has associated an installation cost $c_{u,f}$ which also depends on the node u. D represents the set of demands. Each demand $d \in D$ is modeled by a quadruple with v_s the source, v_d the destination, c_d the ordered sequence of network functions that need to be performed, and bw_d the required units of bandwidth. Table 3.2 defines the notation used throughout the chapter.

We consider a setting with splittable flows as it is frequent to have load balancing in networks [AFT07] and as it makes the model quicker to solve [GK07]. Following the model of [SJG+17], a demand can follow different paths and the network functions of its chain can be processed in different cloud nodes.

The optimisation task consists in routing each demand while *minimising the network operational cost* defined in terms of bandwidth and VNFs cost (licenses, energy consumption, etc). Also, as the dynamics related to the arrival and departure of demands may leave the network in a sub-optimal operational state, we want to reconfigure the network to improve resources usage and to be able to accommodate new incoming traffic. In doing this, we use the *make-before-break* mechanism to avoid network services disruption due to traffic rerouting resulting from the re-optimisation process.

See Figure 0.2.10 for an example.

3.4 Modeling

In the considered setting, demands arrive and leave the network. To route them, we consider them one by one, and find the route which minimises the *additional network operational cost* to be paid. Indeed, in an SDN network, even if multiple flows arrive simultaneously, they will be processed

Reference	Type of requests	# VNFs by Request	Allocation	Objective	Reconfiguration Done	Reconfiguration Cost	Uninterrupted Flow	ILP	Heuristic	Dynamic Programming	Reinforcement Learning
[KLLT18]	SFC	Up to 8	Dynamic	Max Accept	-	NC	NC	-	✓	✓	-
[TGHP18]	SFC	Unknown	Static	Min Cost	-	NC	NC	✓	✓	-	-
[PDM+16]	Flow	0	Dynamic	Min Cost	✓	✓	-	✓	✓	✓	-
[EMAL17]	SFC	Up to 3	Static	Max Accept Min Cost	✓	✓	-	✓	✓	-	-
[NKT19]	SFC	Up to 4	Dynamic	Min Cost	✓	✓	-	✓	-	-	-
[HFS+19]	SFC	3	Dynamic	Min Cost	✓	✓	-	✓	✓	-	-
[WFS+20]	SFC	2-3	-	Min Cost	✓	✓	-	✓	-	-	✓
[LLZ+17]	SFC	Up to 5	Dynamic	Max Profit	✓	-	-	✓	-	✓	-
Our Work	SFC	Up to 4	Dynamic	Min Cost	✓	-	✓	✓	✓	-	-

Table 3.1 – Related Work Summary where NC means Not Concerned, '-' means that the paper does not deal with this parameter, and ✓means that this parameter is addressed in the paper.

$G = (V, E)$	the network where V represents the set of nodes and E the set of links.
C_{uv}	capacity of a link $(u, v) \in E$ expressed as its total bandwidth available.
C_u	available resources* such as CPU, memory, and disk of a node $u \in V$.
Δ_f	number of cores required per unit of bandwidth required by the function $f \in F$.
$c_{u,f}$	installation cost of the function $f \in F$ which also depends on the node u.
(v_s, v_d, c_d, bw_d)	each demand $d \in D$ is modeled by a quadruple with v_s the source, v_d the destination, c_d the ordered sequence of network functions that need to be performed, and bw_d the required units of bandwidth.

Table 3.2 – Notation used throughout the Chapter

one by one by the SDN controller [MSB+17]. We then reconfigure the network to improve the network operational cost when one of the following conditions holds:

- Periodically, after a given period of time;
- When the set of requests has changed significantly (after a given number of SFC arrivals and departures);
- When a request arrives and cannot be accepted with the current provisioning and routing solution.

The solution we propose, called `Break-Free-ILP` (for Break-Free Reconfiguration algorithm), implements a *make-before-break* mechanism to avoid the interruption of the flows. In our experiments, we compare its results with a reconfiguration algorithm which does not implement such a mechanism, called `Breaking-Bad` (for Breaking-Bad Reconfiguration algorithm). This algorithm breaks the flows before rerouting them, leading to packet losses and QoS degradation for these flows. When a reconfiguration has to be carried out, `Breaking-Bad` considers basically a static setting with the requests present in the network and finds an optimal Routing & Provisioning solution (R&P) without considering the current setting.

The algorithms presented in this chapter will use the concept of a layered graph, see the section 2.4 in chapter 2 for a full description. To solve the problem of static routing and compute the optimal allocation to a set of SFCs we used the ILP `Breaking-Bad` presented in subsection 2.5.2 of chapter 2. To route one demand when it dynamically arrives we use an ILP derived from the previous one and working on the residual capacities of the network. It is presented in subsection 2.6.2 of chapter 2. These models differ in the use of fractional variables, which allow requests to be processed as flows rather than paths.

In this Section we present one optimisation model to reconfigure the network with the make-before-break mechanism of `Break-Free-ILP` (subsection 3.4.2). For large networks, the models may take a prohibitive time to be solved. We thus also propose a heuristic algorithm, `Break-Free-HEUR`, to solve large instances in subsection 3.4.3.

3.4.1 Objective

For all models used here (both reconfiguration and allocation of SFCs), we need to find valid SFC allocations. The demands are routed by finding a path on the layered graph for each of them. In doing this, both node and link capacities must be respected as they are shared among all the demands. The objective is to minimise the network operational cost (i.e., bandwidth cost and network function activation cost). As network functions can be shared, it may be better to activate a small number of network functions. The parameter $\beta \geq 0$ specified by the network administrator accounts for different scales over which the functions' activation cost is put in relationship with the network bandwidth cost. β represents how many ***TB/s*** of data can be sent when using a dollar. Its dimension thus is $TB/dollars$, giving that our objective function formally expresses a bandwidth.

3.4.2 `Break-Free-ILP` Reconfiguration (Make-before-break)

A first way to perform the reconfiguration at a given time t is to try to *reconfigure to optimal*. This is done in two phases. In the first one, we compute a minimum cost routing for the set of demands present at time t. This can be done by using the model for static R&P presented in subsection 2.5.2. In the second one, we compute the transitions from the current routing to the optimal routing for each demand, taking into account the intermediate make-before-break steps during which two paths may co-exist for demands that need to be moved. This can be done using the ILP presented in this section, taking as inputs the current and the minimum cost solutions.

However, the transitions to an optimal solution may be long to compute or even impossible to carry out. Indeed, in complex scenarios (which occur when the network is saturated), the transition to the new routes cannot be performed directly. This is mainly due to two reasons.
First, in an intermediate step of the reconfiguration, two routes are provided, leading to an increased use of the network resources. Second, a request may need to be in an intermediate routing state before reaching its final one in order to free space for another request. This needs to be done during distinct reconfigurations steps. Because of this, several intermediate steps of reconfigurations may be necessary, and each additional step of reconfiguration significantly increases the number of variables and constraints of the problem, and thus the time needed to obtain an optimal solution. Therefore, we propose a *best effort reconfiguration*.

Best Effort Reconfiguration. The idea here consists in improving the R&P as much as possible instead of setting a final R&P as a target. To this end, we set a number of intermediate reconfiguration steps, T, (how to set T is discussed below) and the goal of the optimisation is to find a routing with minimal cost which can be reached from the current routing using T reconfiguration steps. Note that the best effort reconfiguration has several advantages compared to the reconfiguration to optimal. It will give a solution as good as the reconfiguration to optimal when such a reconfiguration is possible. Indeed, several optimal solutions may exist, and only part of them could be reached using reconfiguration. Reconfiguration to optimal is focusing on only one, when Best Effort reconfiguration could reach any of those. Second, when reconfiguration to optimal is not possible, Best Effort reconfiguration may still be able to find a solution better than the current one. This is why we used Best Effort reconfiguration in our experiments.

*A node u with a strictly positive number of cores (i.e., $C_u \in \mathbb{N}^+ = \{1, 2, \cdots\}$) represents a cloud location with the capability to execute VNFs, while a node with $C_u = 0$ is a node that serves only as an SDN router.

Best Effort reconfiguration can be modeled using the ILP presented in the following. At time 0, the R&P is set to the current one. Then, at each step of reconfiguration, a set of demands can be rerouted as long as there are enough link and node capacities to satisfy the intermediate make-before-break reconfiguration steps. This can be modeled linearly by defining a variable which is equal to 1 if a resource is used by a request either at time $t-1$ or at time t. As a single step of reconfiguration may not be enough, the ILP has several intermediate reconfiguration steps, each corresponding to a solution of the R&P problem. The objective function is to minimise the cost of the R&P of the final state.

Note that reconfiguration to optimal can be modeled using the same ILP with a few changes. We just have to set the variables corresponding to the final state to the minimum cost R&P computed previously.

Choosing T, the number of reconfiguration steps. The value of T is an important parameter. Indeed, a value too small may lead to models with no solution, while a value too large to models with prohibitive execution times. This is why we tested different values in our experiments. We observe that when the network is not congested, corresponding to the low-traffic scenarios of subsection 3.5.2, a single reconfiguration step is enough to provide optimal (or close to optimal) solutions while it leads to solutions almost as bad as without reconfiguration in the high-traffic scenarios of subsection 3.5.3. In the later scenarios, at least 2 reconfiguration steps are necessary. A good way to find the right value is to start with $T = 1$, which is the fastest model, and then to increase progressively the value of T until either the solution does not improve any more or the model solving time is too long. Note that, when a maximum solving time is set, the largest possible value of T leading to a lower solving time can also be found by dichotomy.

Model.

The ILP takes as an input both the current configuration (i.e., paths and function locations for all the demands) and the number of time steps T to be used in the reconfiguration process. The output corresponds to both the final SFC-R&P at time T after the reconfiguration process and the intermediate SFC-R&P to be used to reach the final state. Between two consecutive time steps $t_0 < ... < t_i < t_{i+1} < ... < T$, a subset of the demands may be moved to a new route. In doing this, resources of both nodes and links must not be exceeded in order to not interrupt connections (*make-before-break*).

Variables:

• $\varphi^{d,t}_{uv,i} \geq 0$ is the amount of flow on Link (u, v) in Layer i at time step t for Demand d.

• $\alpha^{d,t}_{u,i} \geq 0$ is the fraction of flow of Demand d using Node u in Layer i at time step t.

• $x^{d,t}_{uv,i} \geq 0$ is the maximum amount of flow on Link (u, v) in Layer i at time steps t and $t-1$ for Demand d.

• $y^{d,t}_{u,i} \geq 0$ is the maximum fraction of flow of demand d using Node u in Layer i at time steps t or $t-1$.

• $z^{T}_{u,f} \in \{0, 1\}$, where $z^{f}_{u} = 1$ if function f is activated on Node u at time step T in the final routing.

The optimisation model starts with the initial configuration as an input. Thus, for each demand $d \in D$ the variables $\varphi^{d,0}_{uv,i}$ (for each node $u \in V$, layer $i \in \{0, ..., |c_d|\}$) and $\alpha^{d,0}_{u,i}$ (for each link $(u, v) \in E$, layer $i \in \{0, ..., |c_d|\}$) are known. The ILP is based on the layered graph described in

section 2.4 and it is written as follows.

Objective: minimise the amount of network resources consumed during the last reconfiguration time step T.

$$\min \quad \sum_{d \in D} \sum_{(u,v) \in E} \sum_{i=0}^{|c_d|} bw_d \cdot \varphi_{uv,i}^{d,T} + \beta \cdot \sum_{u \in V} \sum_{f \in F} c_{u,f} \cdot z_{u,f}^{T}$$

Constraints:

Flow conservation constraints. For each Demand $d \in D$, Node $u \in V$, time step $t \in \{1, ..., T\}$.

$$\sum_{(u,v) \in \omega^+(u)} \varphi_{uv,0}^{d,t} - \sum_{(v,u) \in \omega^-(u)} \varphi_{vu,0}^{d,t} + \alpha_{u,0}^{d,t} = \begin{cases} 1 \text{ if } u = v_s \\ 0 \text{ else} \end{cases} \tag{3.1}$$

$$\sum_{(u,v) \in \omega^+(v)} \varphi_{uv,|c_d|}^{d,t} - \sum_{(v,u) \in \omega^-(v)} \varphi_{vu,|c_d|}^{d,t} - \alpha_{u,|c_d|-1}^{d,t} = \begin{cases} -1 \text{ if } v = v_d \\ 0 \text{ else} \end{cases} \tag{3.2}$$

$$\sum_{(u,v) \in \omega^+(u)} \varphi_{uv,i}^{d,t} - \sum_{(v,u) \in \omega^-(u)} \varphi_{vu,i}^{d,t} + \alpha_{u,i}^{d,t} - \alpha_{u,i-1}^{d,t} = 0. \qquad 0 < i < |c_d| \tag{3.3}$$

Node usage over two consecutive time periods. For each Demand $d \in D$, Node $u \in V$, Layer $i \in \{0, ..., |c_d| - 1\}$ time step $t \in \{1, ..., T\}$.

$$\begin{aligned} \alpha_{u,i}^{d,t} &\leq y_{u,i}^{d,t} \\ \alpha_{u,i}^{d,t-1} &\leq y_{u,i}^{d,t} \\ \alpha_{u,i}^{d,t} + \alpha_{u,i}^{d,t-1} &\geq y_{u,i}^{d,t} \end{aligned}$$

Link usage over two consecutive time periods. For each Demand $d \in D$, Link $(u, v) \in E$, Layer $i \in \{0, ..., |c_d|\}$ time step $t \in \{1, ..., T\}$.

$$\begin{aligned} \varphi_{uv,i}^{d,t} &\leq x_{uv,i}^{d,t} \\ \varphi_{uv,i}^{d,t-1} &\leq x_{uv,i}^{d,t} \\ \varphi_{uv,i}^{d,t} + \varphi_{uv,i}^{d,t-1} &\geq x_{uv,i}^{d,t} \end{aligned}$$

These two last constraints state that the flow $y_{u,i}^{d,t}$ for the node (or $x_{uv,i}^{d,t}$ for the link) is equal to the maximum flow between two intermediate reconfiguration steps. If there is no flow in u and uv during steps $t-1$ and t then $y_{u,i}^{d,t}$ (and $x_{uv,i}^{d,t}$) are equal to 0.

Make Before Break - Node capacity constraints. The capacity of a node u in V is shared between each layer and cannot exceed C_u considering the resources used over two consecutive time periods.

For each Node $u \in V$, time step $t \in \{1, ..., T\}$.

$$\sum_{d \in D} bw_d \sum_{i=0}^{|c_d|-1} \Delta_{f_i^{c_d}} \cdot y_{u,i}^{d,t} \leq C_u. \tag{3.4}$$

Make Before Break - Link capacity constraints. The capacity of a link $(u, v) \in E$ is shared between each layer and cannot exceed C_{uv} considering the resources used over two consecutive time periods. For each Link $(u, v) \in E$, time step $t \in \{1, ..., T\}$.

$$\sum_{d \in D} bw_d \sum_{i=0}^{|c_d|} x_{uv,i}^{d,t} \leq C_{uv}. \tag{3.5}$$

Location constraints. A node may be enabled to run only a subset of the virtual network functions. For each Demand $d \in D$, Node $u \in V$, layer $i \in \{0, ..., |c_d| - 1\}$, if the $(i+1) - th$ function of c_d cannot be installed on Node u, we add the following constraint for each time step $t \in \{1, ..., T\}$.

$$\alpha_{u,i}^{d,t} = 0 \tag{3.6}$$

Functions activation. To know which functions are activated on which nodes in the final routing. For each Node $u \in V$, Function $f \in F$, Demand $d \in D$, and Layer $i \in \{0, ..., |c_d| - 1\}$,

$$\alpha_{u,i}^{d,T} \leq z_{u,f_i^{c_d}}^{T}. \tag{3.7}$$

Note that we do not consider the cost of potential activations of VNFs during the reconfiguration process. Indeed, our goal is to minimise the network operational cost over time and the reconfiguration duration is very small in comparison in an SDN network [B+14].

3.4.3 Heuristic `Break-Free-HEUR`

As shown by the numerical evaluations in subsection 3.5.5, the ILP models may take a long time to be solved for large networks. We thus present a heuristic algorithm, `Break-Free-HEUR`, able to provide good solutions for them. The algorithm reconfigures the requests as closely as possible to a given (optimal if possible) configuration in a given number of steps.

`Break-Free-HEUR` is an iterative algorithm presented in Algorithm 8, which starts from an initial allocation and tries to reconfigure as many as possible the SFCs to a given allocation. In the best case, all SFCs are reconfigured to the new allocation, in the worst case no SFCs are reconfigured. Algorithm 8 takes as inputs the graph G, the initial allocation (allocInit given by the flow values φ and α at time 0) the allocation to which we reconfigure (allocFinal given by the flow values φ and α at time T) and the number of steps allowed to reconfigure (nbSteps). The main technical point of the algorithm is concentrated in the procedure `reconfSFC` (Algorithm 9). The difficulty derives from the fact that we consider *splittable* flows and that only *part of these flows* can be rerouted by the procedure.

In Lines 1 and 2, the initial allocation of each SFC is divided into two allocations: currentOpti is the set of flows already reconfigured (noted as φ^{d,t^*}) and currentNonOpti is the set of flows that remains to be reconfigured (noted as $\varphi^{\perp} = \varphi^{d,t} - \varphi^{d,t^*}$). These two allocations represent the current allocation of each SFC. We also consider the set of flows of the final allocation to which

Algorithm 8: `Break-Free-HEUR`

```
   Data: G, listSfc, allocInit, allocFinal, nbSteps
   Result: Reconfigure the allocation as close as possible to the optimal
1  currentOpti ⟵ void allocation;
2  currentNonOpti ⟵ copy of allocInit;
3  remainingFinal ← allocFinal;
4  listSfcToReconf ⟵ List of SFCs whose initial allocation is not the final allocation;
5  for step in nbSteps do
6  |  G' ⟵ Residual graph of G for the current step;
7  |  for s in listSfcToReconf do
8  |  |  reconfSFC(G', currentOpti[s], currentNonOpti[s], allocFinal[s], s);
9  |  |  if currentOpti[s] = allocFinal[s] then
10 |  |  |  remove s from listSfcToReconf;  ▷ # We no longer need to reconfigure this sfc
11 |  if no chains have been moved during this step then
12 |  |  break;                                  ▷ # We can no longer reconfigure sfc
13 return fusion(currentOpti, currentNonOpti);
```

the initial flow was not yet reconfigured, named remainingFinal. The corresponding flow is noted $\varphi^{\top} = \varphi^{d,T} - \varphi^{d,t^*}$.

In Line 4 we list the SFCs to be reconfigured: those of which allocInit is different from allocFinal. In Line 6 we compute G', the residual graph of G using the current allocation of each SFC for the current step. The capacities of G' are given by $c_{uv}(G') = C_{uv} - \sum_d \sum_i^{c_d} \varphi_{uv,i}^{d,t}$ for edges and $c_u(G') = C_u - \sum_d \sum_i^{c_d} \alpha_{u,i}^{d,t}$ for nodes. In Line 8, for each SFC to be reconfigured, the procedure `reconfSFC` (presented in Algorithm 9) moves as much flow as possible from currentNonOpti to currentOpti. The procedure `reconfSFC` updates currentOpti, currentNonOpti and the residual graph G' to take into account the additional capacity used at each reconfiguration step. And it will return if it has successfully changed the current allocation of the SFC or if no move is possible at this step. The algorithm stops either if the final allocation is reached and there are no more SFCs to reconfigure, or when no more SFCs can be modified, or when the maximum number of steps has been reached. At the end of the algorithm in Line 13, currentNonOpti and currentOpti of each SFC are merged to return the current allocation: in the best case, currentNonOpti is empty for each SFC and the current allocation is the final allocation. In the worst case no SFC has been reconfigured and the current allocation is the same as the initial allocation.

The procedure `reconfSFC` (Algorithm 9) takes as inputs the residual graph, d (the SFC to be moved), the current allocation of the SFC and the final allocation to which we want to move the SFC. Its goal is to move as much flow as possible from currentNonOpti to remainingFinal. It returns if a reconfiguration has taken place or if the SFC is blocked at this step. In Line 1, we create the layer graph for the flow that is not yet reconfigured by taking only the links and nodes present in currentNonOpti and taking as capacity the flow passing through currentNonOpti. In Line 2, we create the layer graph for the final allocation by taking only the links and nodes present in remainingFinal and taking the capacities of G'. In Line 3, we find a path on currentNonOpti, that is a non splitted subflow from the flow of d which still has to be rerouted. In fact, a flow can be easily decomposed into paths. The procedure `findPath` returns such a path using a

Algorithm 9: reconfSFC(sfc d)

Data: G', currentOpti, currentNonOpti, allocFinal, sfc
Result: Pushes as much flow of currentNonOpti as possible into allocFinal for sfc during one step

```
1  G^⊥ ⟵ layer graph of currentNonOpti(d);
2  G^⊤ ⟵ layer graph of remainingFinal(d);
3  while path^⊥ ← findPath(G^⊥) ≠ Null do
4  |                                  ▷ #Find a path to be rerouted;
5  |  flow^⊥ ← min. value of φ^⊥ over edges and nodes of path  ▷ #Max. flow which can be
   |   rerouted from path;
6  |  while path^⊤ ← findPath(G^⊤)) ≠ Null do
7  |  |                          ▷ #Find a destination path for part of flow^⊥;
8  |  |  flow^⊤ ← maxFlow(G^⊤,path^⊤);
9  |  |  remove flow^⊤ from the edges of path^⊥ on G^⊥;
10 |  |  remove flow^⊤ from currentNonOpti and remainingFinal;
11 |  |  reduce the residual capacity of G^⊤ by flow^⊤;
12 |  |  add flow^⊤ into currentOpti;
13 if If currentOpti has not changed then
14 |  return False;
15 return True;
```

depth-first-search from the source of the SFC d to its destination. The value of the flow which can be rerouted from this path is the minimum value (over all edges of the path) of the flow passing through currentNonOpti (Line 4).

Algorithm 10: findPath(G)

Data: a graph G
Result: Find an s-t path

```
1  Carry out a Bread First Search in G starting from s and stopping at t;
2  if t never reached then
3  |  return Null;
4  path ← path from s to t given by the DFS;
5  return path;
```

We now want to reroute this subflow. We do it iteratively from Lines 6 to 12. In Line 6, we compute a target path, $path^{\top}$, to which we reroute some flows. Then, we compute the maximum value of flow which can be rerouted using the procedure `maxFlow` (given in Algorithm 11). The computation of the maximum *flow* which can be rerouted on $path^{\perp}$ is not direct due to layers sharing capacities. First, if $path^{\top}$ has an edge in the layered graph, (e, i), which is common with path$^{\perp}$, we know that $flow^{\perp}$ can be completely rerouted on (e, i). We note E^p the set of such edges. Then, for each edge e of $path^{\top}$, we compute the maximum flow, f_e^*, which can pass on the edge. The flow f_e^* is equal to the capacity C_e divided by the number of times $path^{\top}$ goes through e in different layers ($f_e^* = C_e/|\{(e, i)$ with (e, i) in *path* and $(e, i) \notin E_p\}|$). Then, we

set $f^* = \min_{e \in E} f_e^*$. We have *flow* $\leq f^*$. Second, we should not reroute more flow than the one of the target solution on path $path^\top$. Thus, we have $flow^\top \leq \min_{e \in path} \varphi_e^\top$. Last, the value of the rerouted flow cannot be larger than the flow which is rerouted, that is $flow^\top \leq flow^\perp$. This gives

$$flow^\top = \min(flow^\perp, f^*, \min_{e \in path} \varphi_e^\top).$$

We now have the value of $flow^\top$ and its path. We update the flows and capacities of the residual

Algorithm 11: maxFlow(G,path$^\top$,path$^\perp$,flow$^\perp$)

Data: a graph G,path$^\top$,path$^\perp$,flow$^\perp$
Result: The maximum value of flow which can be rerouted from path$^\perp$ to path$^\top$.

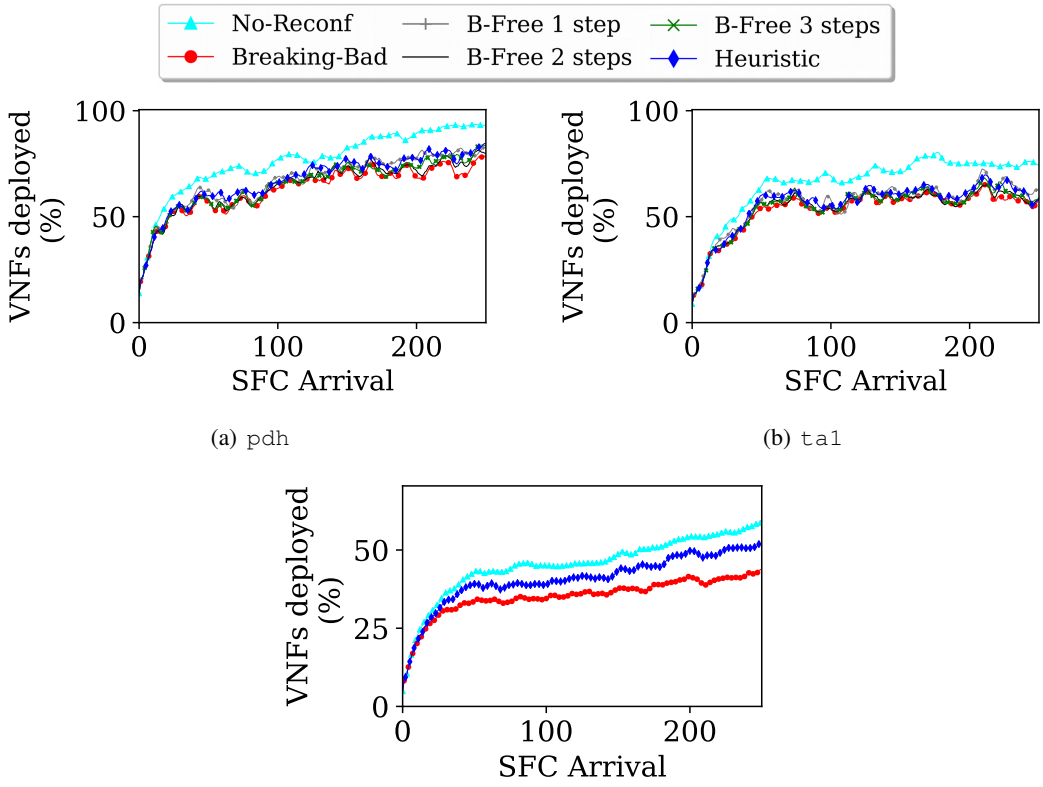
```
1  E_p ← ∅                          ▷ #E_p = {(e,i) : (e,i) ∈ path^⊤ and (e,i) ∈ path^⊥};
2  for (e,i) ∈ path^⊤ do
3  |  if (e,i) ∈ path^⊥ then
4  |  |  E_p.append((e,i));
5  for e in path^⊤ do
6  |  nbPassages ← 0;
7  |  for (e,i) in path^⊤ do
8  |  |  if (e,i) ∉ E_p then
9  |  |  |  nbPassages ← nbPassages+1;
10 |  f_e^* = C_e / nbPassages;
11 f^* = min_{e∈E} f_e^*;
12 flow = min(flow^⊥, f^*, min_{e∈path} φ_e^⊤);
13 return flow;
```

graphs $G^\perp$ and $G^\top$ for the remaining of the procedure `reconfSFC` (Lines 7 and 8). We also accordingly update the allocations currentNonOpti, currentOpti, remainingFinal which are used in the main Algorithm 8 (Lines 9 and 10). We then iterate on all the paths in $G^\perp$.

Finally, in Line 13, we check that we have succeeded in reconfiguring at least part of the flow. Otherwise, we return that sfc is blocked at this step.

3.5 Numerical Results

In this section, we evaluate the performance of `Break-Free-ILP` and `Break-Free-HEUR`. We study the impact of the reconfiguration on different metrics such as cost savings, acceptance rate, and resource usage. We first present the data sets used for the experiments. Then, we compare the results with the ones of `Breaking-Bad`, which computes an optimal R&P for the whole set of requests (ILP of subsection 2.5.2) for each SFC arrival, and with `No-Reconf`, which computes the R&P problem for a single demand, the newly arrived SFC (subsection 2.6.2).
We consider two scenarios, one with low traffic in which basically all demands can be accepted and one with high traffic in which some of them have to be rejected in order to satisfy the capacity constraints. In the low traffic scenario, we can fairly compare resource usage using the different algorithms `Break-Free-ILP`, `Breaking-Bad`, and `No-Reconf`, as they are accepting the same demands. In the high traffic scenario, we can compare them in terms of acceptance rate.



(a) `pdh` (b) `ta1`

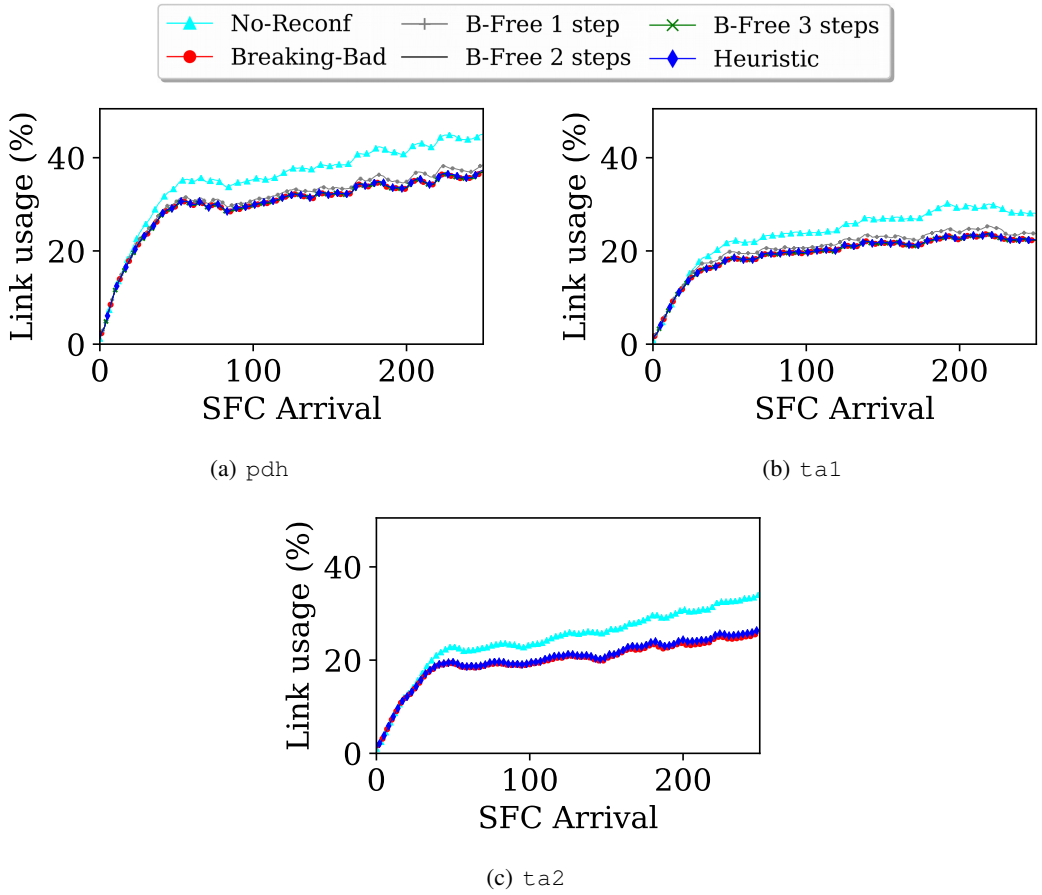
(c) `ta2`

Figure 3.1 – Low-Traffic scenario - Number of VNFs deployed across time.

We show in particular that `Break-Free-ILP` allows to lower the network cost and increases the acceptance rate almost as much as `Breaking-Bad`. For both algorithms, a large number of demands have to be rerouted, showing that it is crucial to implement a mechanism to avoid impacting them. Network reconfiguration has to be done often to attain a significant gain, however, this reconfiguration can be quickly computed. This allows reconfiguration mechanisms to be put into practice.

3.5.1 Data sets

We conduct experiments on three real-world topologies from SNDlib [OWPT10] of different sizes: `pdh` (11 nodes, 34 links), `ta1` (24 nodes, 55 links), and `ta2` (65 nodes, 108 links). The Table 3.3 summarizes the properties of each network topology. We generate our problem instances as follows. We considered 250 demands for each network. The source and destination of each demand are chosen using the given traffic matrices. Following [S+15], the lifetime of a demand is exponentially distributed with mean $\mu = 20$ for the low-traffic scenario and with mean $\mu = 45$ for the high-traffic scenario. We then round this lifetime to an integral number of time steps. The volume of the demands is chosen randomly. Also, each demand is associated with an ordered sequence of 2 to 4 functions uniformly chosen at random from a set of 5 different functions. Experiments have been conducted on an Intel Xeon E5520 with 24GB of RAM. `Break-Free-ILP` is not studied on `ta2` due to excessive runtime.



(a) `pdh` (b) `ta1`

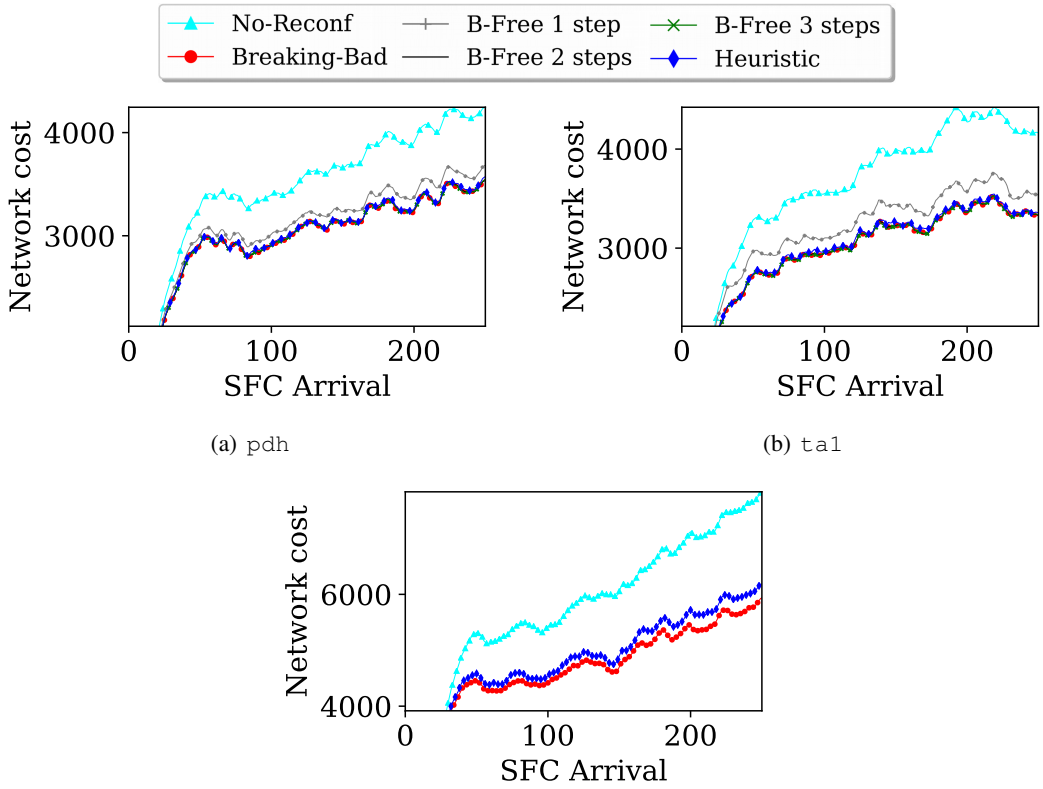
(c) `ta2`

Figure 3.2 – Low-Traffic scenario - Bandwidth usage across time.

3.5.2 Low-traffic scenario - Resource usage

In Figure 3.3, we show the network cost for the *low-traffic scenario*. This cost is the result the weighted addition of Bandwidth (Figure 3.2) and of VNF costs (Figure 3.1). The results are given for `No-Reconf` and `Breaking-Bad` as a measure of comparison, for several variants of `Break-Free-ILP` with different numbers of reconfiguration steps from 1 to 3, and for `Break-Free-HEUR` with 10 steps of reconfiguration. We focus on the low-traffic scenario as the compared algorithms accept the same requests and therefore, we can have a comparison for the same global volume of traffic

We first see that `Break-Free-ILP` has similar performances to `Breaking-Bad` in terms of network operational cost. Recall that `Breaking-Bad` interrupts the requests during reconfiguration. This means that `Breaking-Bad` provides a lower bound for `Break-Free-ILP`. As `Break-Free-ILP` does not interrupt the requests, it won't be able to reach a better solution than `Breaking-Bad`. Moreover, `Break-Free-ILP` achieves this performance for any number of time steps (even 1). This leads to a very fast algorithm as discussed below. Indeed, when the network is not congested, there is enough capacity to host both the old and new routes. Nevertheless in `ta2` the efficiency of `Break-Free-HEUR` is slightly below `Breaking-Bad`.



(a) `pdh` (b) `ta1`

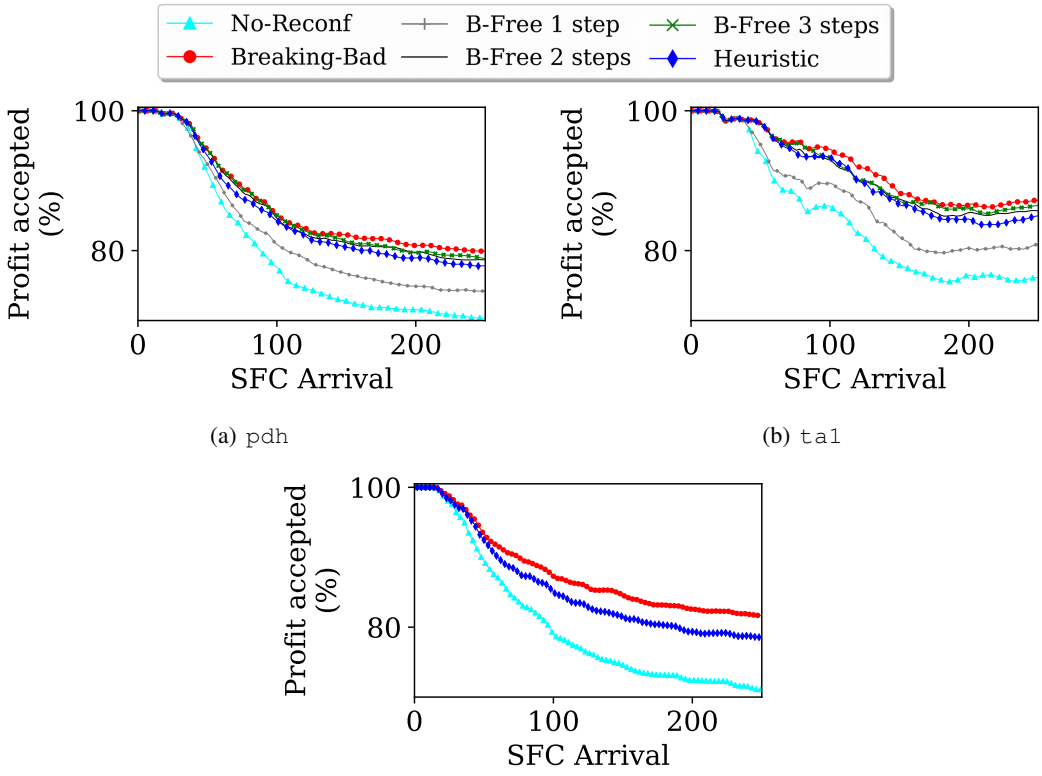
(c) `ta2`

Figure 3.3 – Low-Traffic scenario - Network operational cost.

Reconfiguration leads to a better resource utilization and reduces the network operational cost compared to `No-Reconf`, and this given a same volume of traffic (note that no demand is rejected in this scenario). Indeed, reconfiguring the network regularly permits a reduction of 15% of network operational cost (Figure 3.3(a)) while using 7% fewer VNFs (Figure 3.1(a)) and 18% less link bandwidth (Figure 3.2(a)) compared to the no-reconfiguration case on `pdh`. For `ta1`, we have a reduction of 20% of network operational cost while using 17% fewer VNFs and 21% less link bandwidth compared to `No-Reconf`. Finally, for `ta2` we have a reduction of 19% of network operational cost while using 10% fewer VNFs and 22% less link bandwidth compared to the no-reconfiguration.

topology	nb Nodes	nb Links	degree min	degree max	degree avg	diameter
pdh	11	34	4	8	6.18	3
ta1	24	55	2	11	4.58	4
ta2	65	108	1	10	3.32	8

Table 3.3 – Three-real world topologies



(a) pdh

(b) ta1

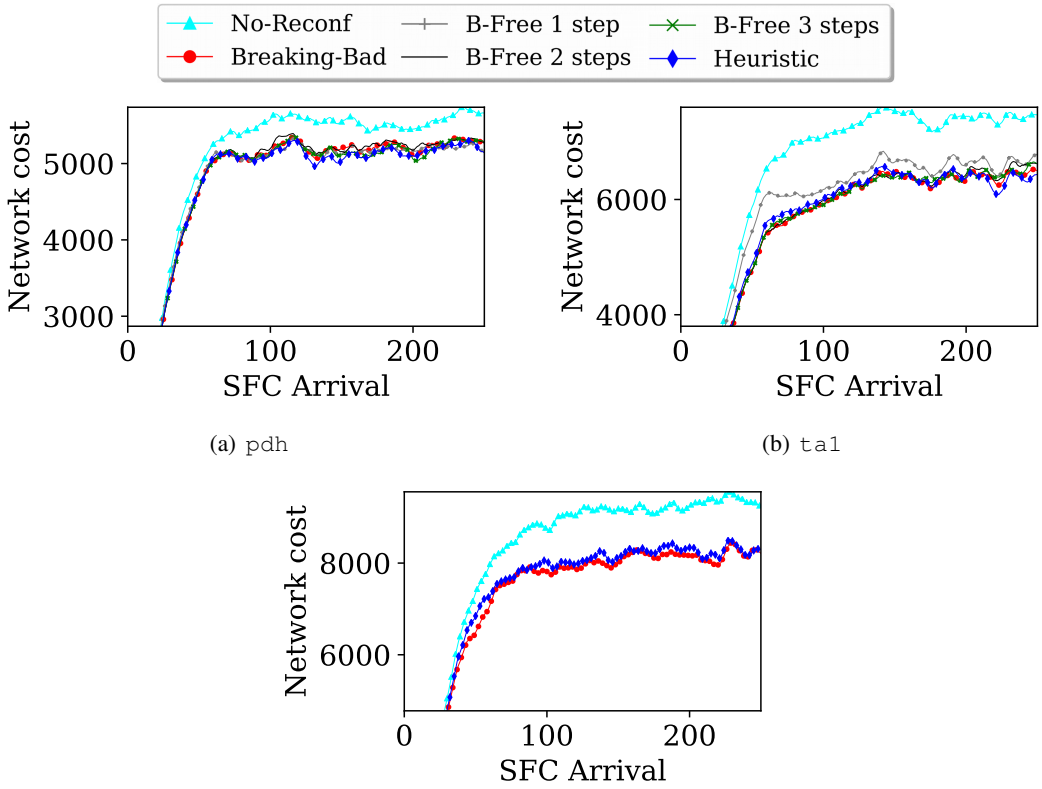
(c) ta2

Figure 3.4 – High-Traffic scenario - Percentage of accepted profit across time.

For ta2 and unlike pdh and ta1, Break-Free-HEUR deployed two times more VNFs than Breaking-Bad (Figure 3.1(c)). But, in the same time, Break-Free-HEUR reduces drastically the bandwidth usage (Figure 3.2(c)), leading in the end to a good improvement of network cost (Figure 3.3(c)).

The results for Break-Free-HEUR on ta2 show that we can reduce the whole network operational cost, but not equally between the bandwidth usage and the VNF cost. The diameter on this graph is 8, and the average degree connectivity is low compared to pdh and ta1. This implies that finding alternative paths reducing the number of links is more interesting to reduce the global network operational cost than moving the VNFs to other data centers. Recall that there are a fixed number of data centers where the VNFs can be installed, and with a large network, there are less opportunities for changing the VNFs. Moreover, deleting a VNF in one data center implies moving all the SFCs using it, and due to the possible longer paths, it is not always an interesting option.

We can hypothesize that Break-Free-HEUR has more difficulty in stopping using VNFs during reconfiguration because the graph is larger, its diameter is larger too (3 for pdh, 4 for ta1, 8 for ta2) and the average node's degree is also smaller making it more difficult to reconfigure completely every SFCs using a specific VNF.



(a) `pdh` (b) `ta1`

(c) `ta2`

Figure 3.5 – High-Traffic scenario - Cost gain across time.

3.5.3 High-Traffic scenario - Acceptance Rate

In our *high-traffic scenario*, there are not enough resources to satisfy all the demands. As a consequence, some requests cannot be accepted. We show, in Figure 3.4, the profit achieved by `Break-Free-ILP`, `Breaking-Bad`, `Break-Free-HEUR` and `No-Reconf`. We define the profit of a demand as the asked volume of bandwidth multiplied by its duration.
The global profit is defined as the sum of all the accepted requests' profits. This metric is of high importance. Indeed, in case of High-Traffic scenario, some requests will be rejected. However, we want to ensure that our algorithm will accept equally the requests when they arrive. If we consider only the number of accepted requests, one can think of an heuristic accepting only short and low-bandwidth in order to get an higher acceptance rate.

We show the profit as a percentage in terms of maximum achievable profit. In other words, 100% of profit means that all the demands (and their requested bandwidth) have been accepted (100% represents the global profit of all the requests).

It can be seen that `No-Reconf` and `Break-Free-ILP` (with 1-step) lead to equivalent profit, around 70% for `pdh` (and between 78 and 81% for `ta1`), while `Break-Free-ILP` (with 2, 3, and 4 steps), `Break-Free-HEUR` and `Breaking-Bad` have similar performances (around 79% for `pdh` and 87% for `ta1`). On `ta2` the results are the following: 71% for `No-Reconf` against 79% for `Break-Free-HEUR` and 82% for `Breaking-Bad`.

For this congested scenario, one step of reconfiguration is not enough as there is not enough place to move the requests. Therefore, some requests are rejected. Allowing to use more steps in our *make-before-break* reconfiguration process, without interrupting the requests, we can reach the same performances as `Breaking-Bad`.

In Figure 3.5, we show the network operational cost for `Break-Free-ILP`, `Breaking-Bad`, and `No-Reconf` as a function of the number of demands arrived. The first observation is that `Break-Free-HEUR` and `Break-Free-ILP` (with more than 2-steps) lead to a smaller network operational cost than `No-Reconf`. It accepts more, with less cost. The second observation is that even if `Break-Free-ILP` (with 1-step) has a similar profit to `No-Reconf`, it has substantially less network operational cost than all the other algorithms.

3.5.4 Low-Traffic scenario - Impact of Parameter β

In Figure 3.6 and Figure 3.7, we study the impact of the β parameter on the resources required in the network in terms of bandwidth and number of deployed VNFs, respectively.

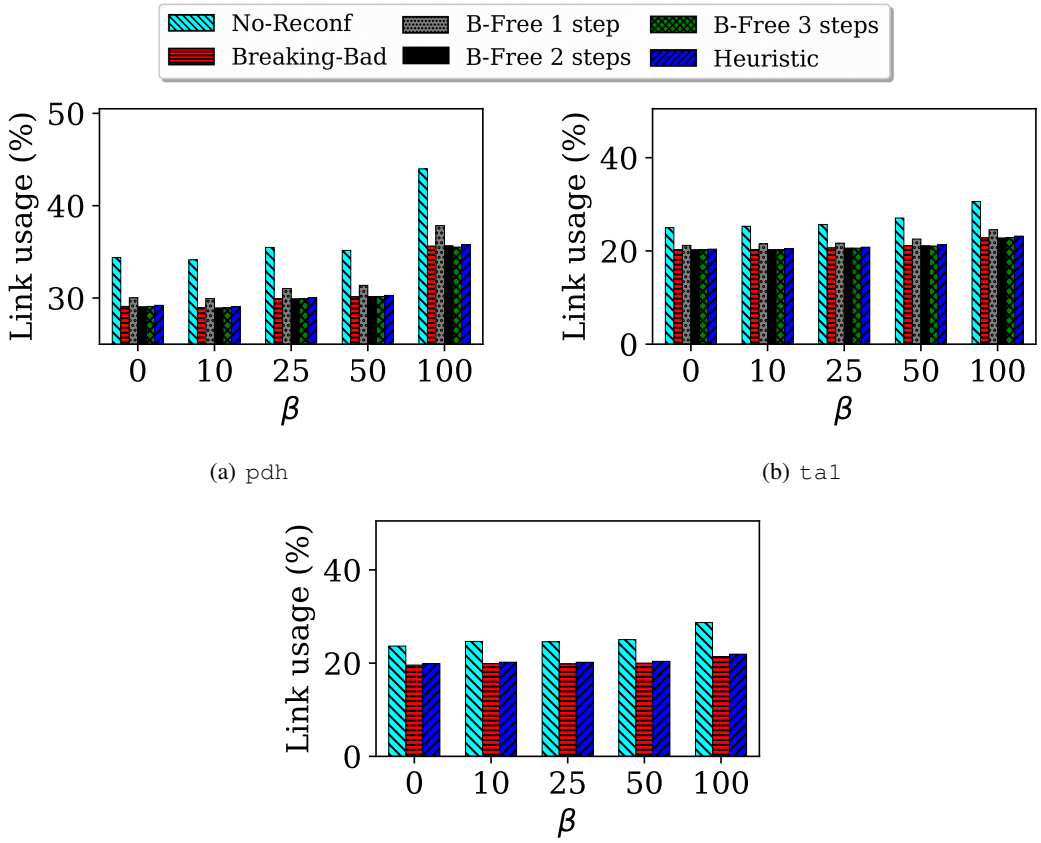
As β increases, the impact of the VNF cost on the total cost is greater. As a consequence, the number of deployed VNFs decreases, leading to longer routes, and thus, to an increased amount of bandwidth usage.

Note also that, for all values of $beta$, reconfiguration using `Break-Free-ILP` (for any number of steps) leads to similar gains to reconfiguration using `Breaking-Bad`. This shows that the conclusion discussed in subsection 3.5.2 for a specific value of $\beta = 25$ (our default value) is valid in more general settings for a wide range of β.

Another important observation is that the gain of reconfiguration is higher for larger values of β. The reason is that, when β is large, the requests tend to use longer routes as the cost of bandwidth is less important compared to the one of VNFs. This leads to requests routed in a very suboptimal way when other requests using the same VNFs leave (as shown in the example of Figure 0.2.10). On the contrary, when β is small, the routes try to always use close to shortest path solutions, leading to lower gains. There is still a gain as a shortest path is not always available (due to nodes and link capacities).

Finally, we can see that, as we said earlier in subsection 3.5.2, the `Break-Free-HEUR` is not as effective in reducing the deployment of VNFs: For `pdh` and `ta1` it is comparable to `Break-Free-ILP` with 1 step. For `ta2` we can see that its efficiency is very limited compared to `Breaking-Bad`, even if it does as well in reducing the use of links.

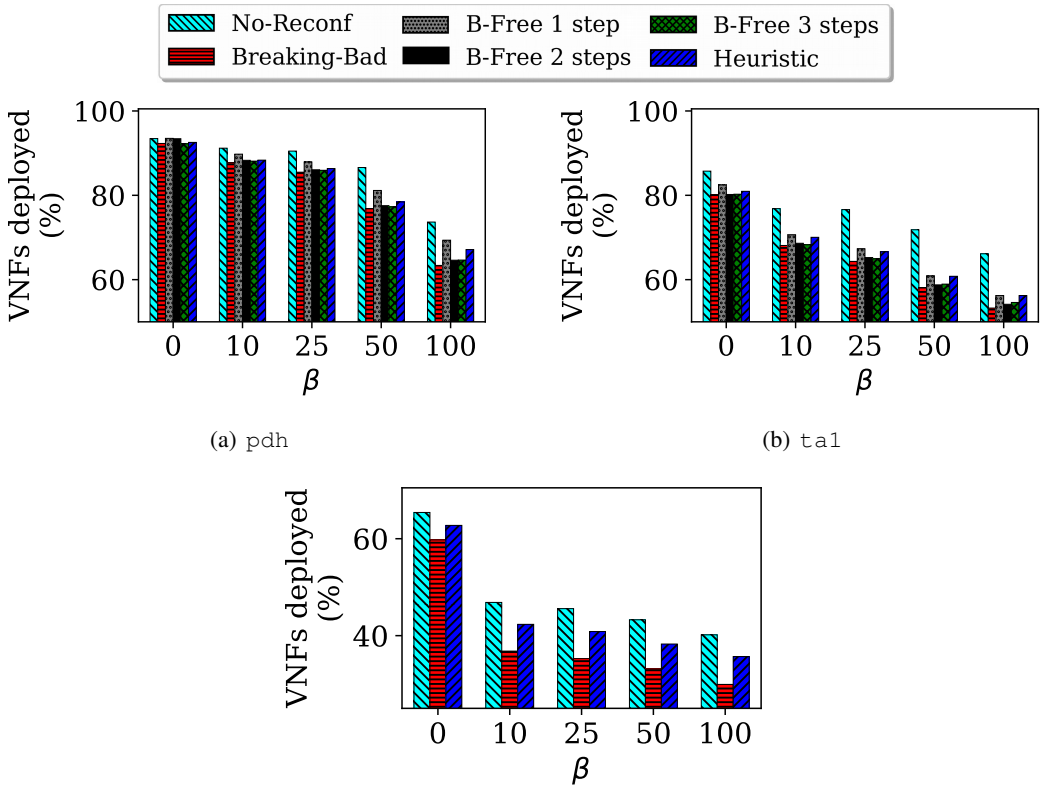
In the following, we use $\beta = 25$, as this is a good compromise between link utilization and number of VNFs deployed.



(a) `pdh` (b) `ta1`

(c) `ta2`

Figure 3.6 – Low-Traffic scenario - Impact of parameter β - Bandwidth usage as a function of β.



(a) `pdh` (b) `ta1`

(c) `ta2`

Figure 3.7 – Low-Traffic scenario - Impact of parameter β - VNFs deployed as a function of β.

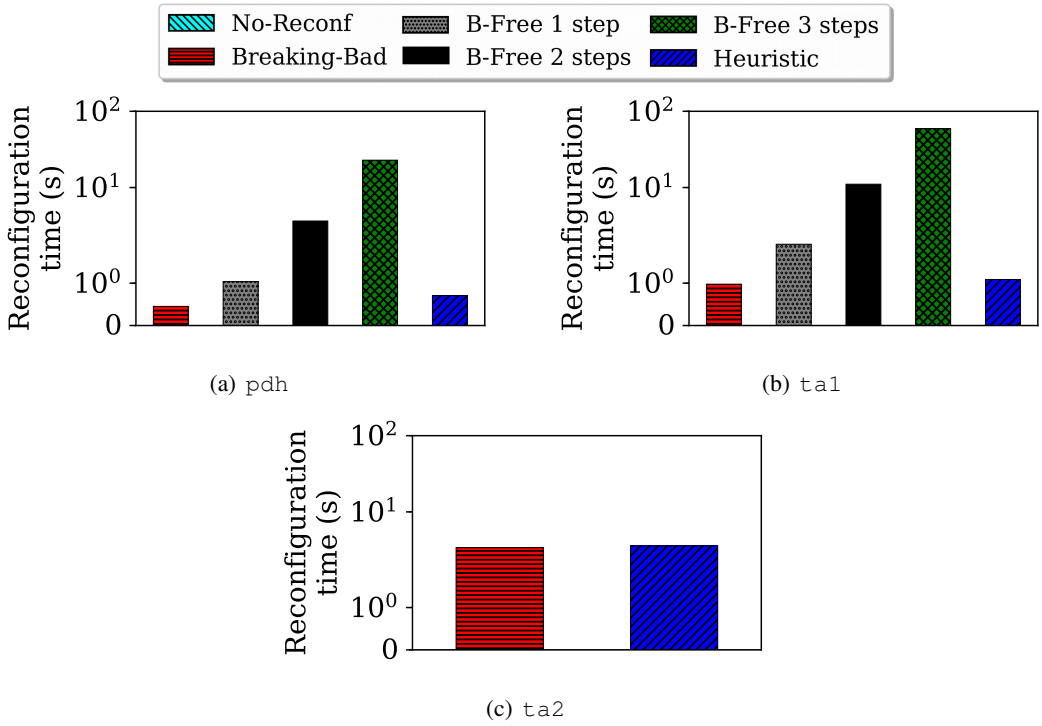


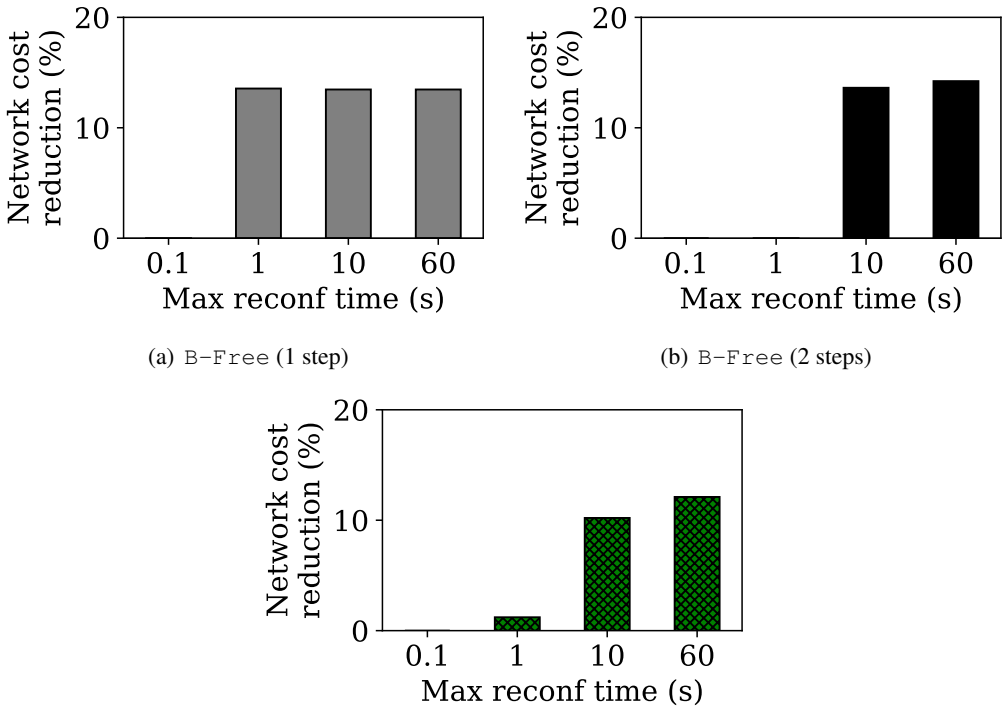
Figure 3.8 – High-Traffic scenario - Average reconfiguration times

3.5.5 Execution Times to Compute the Reconfiguration

Figure 3.8 shows the average times to reconfigure with a logarithmic scale. We can see that the reconfiguration time of `Breaking-Bad` and `Break-Free-HEUR` are within the same order of magnitude. Indeed, recall that in the first steps of `Break-Free-HEUR`, a routing is computed. During the simulations, this routing is computed using `Breaking-Bad`. This explains the identical reconfiguration times.

For `Break-Free-ILP`, even if the computation time is not much longer with one step, it increases with 2 and 3 steps and can not be used on large networks such as `ta2`. `Break-Free-ILP` with one step being far less effective on high-traffic scenarios than `Breaking-Bad` and `Break-Free-HEUR`, it also seems to be of little use on large networks.

Figure 3.9 shows the gains of network cost (compared to `No-Reconf`) in percentage for `Break-Free-ILP` (1 to 3 steps) when limiting the time spent for the reconfiguration. `Break-Free-ILP` with 1 step needs only 1 second to reach its best solution. This variant of the algorithm is almost as fast as `Breaking-Bad` (which does not compute an intermediate make-before-break step). 10 seconds are needed to reach a close to optimal solution for the 2–step variant, and a good solution for the 3–step variant. The best solution is attained after 1 minute. We remind the reader that in the low-traffic scenario, the 1–step variant is enough to achieve solutions close to optimal, while in the high-traffic scenario, this is the case of the 2 step variant. It is thus possible to reconfigure a network without interruption and with significant gain in a few seconds.



(a) `B-Free` (1 step)

(b) `B-Free` (2 steps)

(c) `B-Free` (3 steps)

Figure 3.9 – Low-Traffic scenario - Gains of network operational costs for different time limits for the optimisation process.

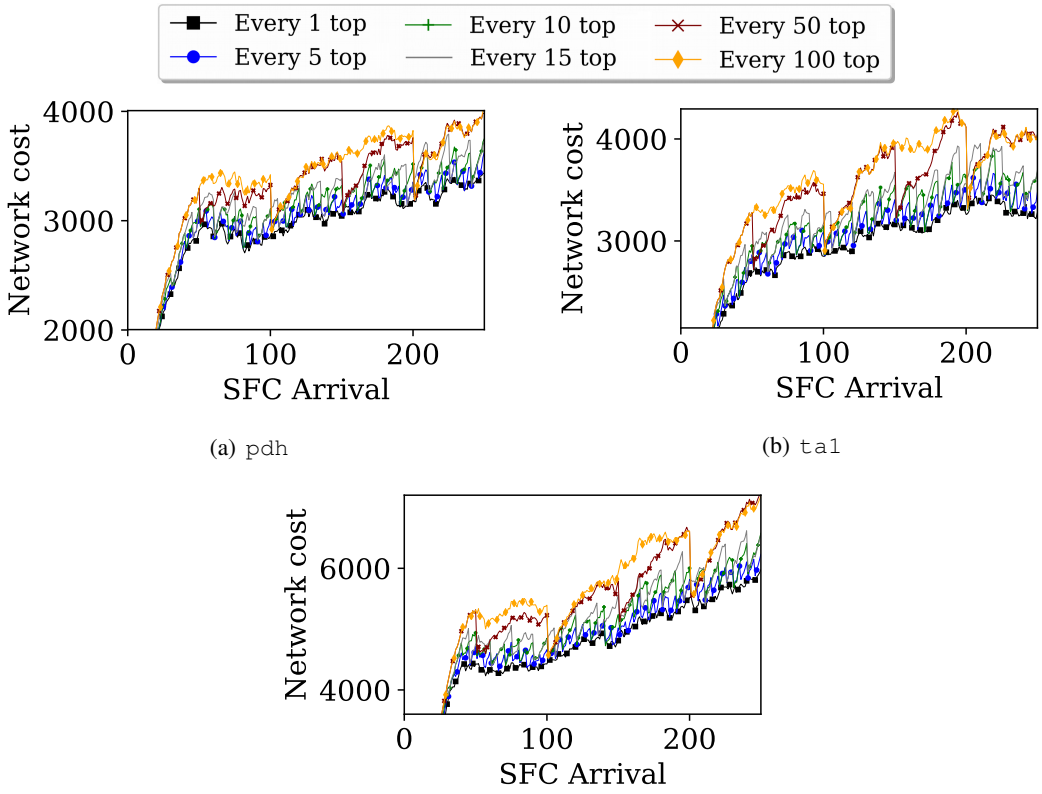
3.5.6 Reconfiguration Rate

In this experiment, we test different reconfiguration rates. Note that during the previous simulations, we reconfigured the network considering the three conditions defined in the beginning of section 3.4. Here, the only condition to reconfigure is the first one, i.e., periodically, after a given number of time steps, defined as the reconfiguration rate.

The faster rate is to reconfigure every time step, while the slowest one in our setting would be to reconfigure every 100 time steps (only 1 or 2 reconfigurations are performed during the whole test). We thus present the results for reconfiguration rates of every 1, 5, 10, 15, 50, and 100 time steps using `Break-Free-HEUR`, the results with `Breaking-Bad` and `Break-Free-ILP` are similar. In Figure 3.10, we provide the network cost in the low-traffic scenario. The minimum cost is as expected achieved when reconfiguring at each time step. However, in this setting similar gain can be obtained when reconfiguring every 10 and 15 time steps for `pdh`, `ta1` and `ta2`.

Results for the high-traffic scenarios can be seen in Figure 3.11, in which we report the profit generated by the accepted demands. In this setting, the network is congested. This means that very frequently the demand arriving at a time step cannot be routed directly.

For `pdh`, not reconfiguring at every time step leads to poor performance, whatever the value of the reconfiguration rate. For `ta1`, this effect is not as stringent. Different reconfiguration rates lead to different values of profit. However, only a reconfiguration every time step leads to an



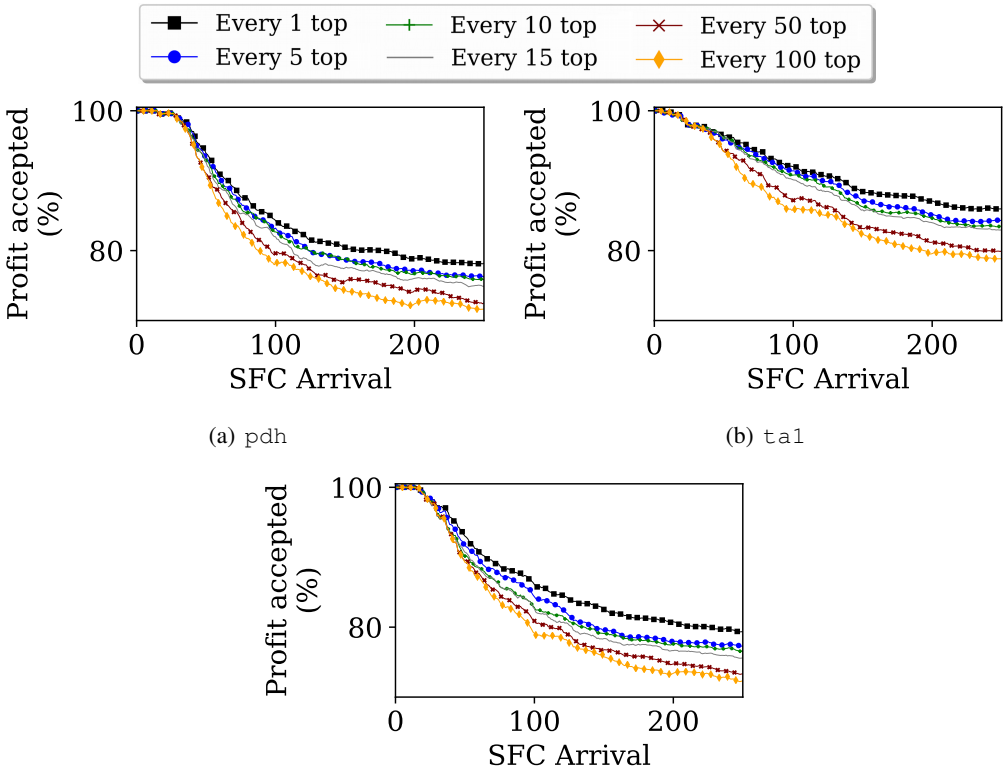
(a) `pdh` (b) `ta1`

(c) `ta2`

Figure 3.10 – Low-Traffic scenario - Impact of the reconfiguration rate on the network cost.

optimal performance. Choosing a rate between 5 and 15 can achieve a high efficiency without reconfiguring too much.

Thus, the reconfiguration should be well chosen by network operators, depending on their network usage. The higher the congestion, the higher the rate should be.



(a) `pdh` (b) `ta1`

(c) `ta2`

Figure 3.11 – High-Traffic scenario - Impact of the reconfiguration rate on the percentage of profit accepted.

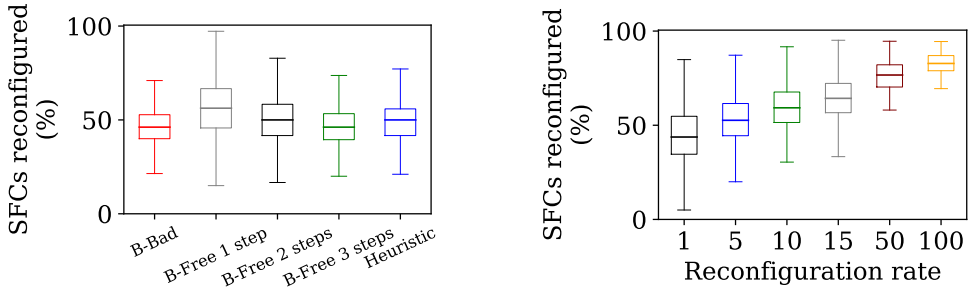
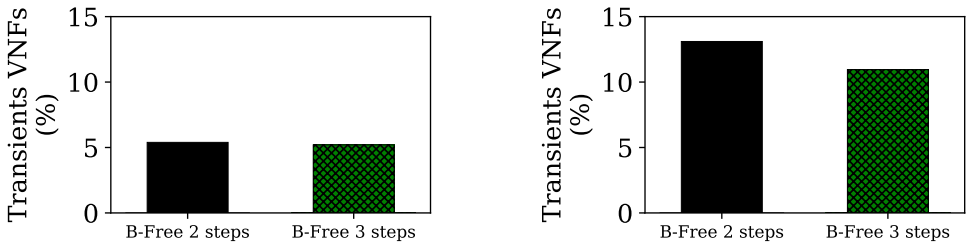


Figure 3.12 – Percentage of rerouted requests for `ta1`, considering (left) different intermediate reconfiguration steps and (right) different reconfiguration rates.

3.5.7 Percentage of rerouted requests

To see the importance of implementing a make-before-break process, we study the percentage of rerouted requests during the reconfiguration process. We report in Figure 3.12 (left) the percentage of reconfigured SFCs for `Break-Free-ILP` (1 to 3 steps), `Break-Free-HEUR` and `Breaking-Bad` for the high-traffic scenario. Firstly, `Breaking-Bad` has to interrupt, on av-



(a) pdh (b) ta1

Figure 3.13 – Percentage of transient VNFs used during the intermediate steps of the reconfiguration.

erage, 48% of the requests (between 20% and 70%) to maintain an optimal solution. This is thus of *crucial importance to avoid impacting this large number of requests when reconfiguring.*
Break-Free-ILP and Break-Free-HEUR change the routing of approximately the same number of requests (except for one step which is less efficient) but without any interruption of traffic.
Note that the number of reconfigured requests depends on the frequency of the reconfiguration, as shown in Figure 3.12 (right). Reconfiguring regularly permits to impact less SFCs at each reconfiguration process. Indeed, around 48% of SFCs are reconfigured when the reconfiguration rate equals 1, while around 80% of SFCs need to be reconfigured if this rate reach 100.

3.5.8 Percentage of Transient VNFs instantiated during reconfiguration

Our objective is to minimise the network operational cost at the final step of the reconfiguration. Since the transient VNFs used during reconfiguration are instantiated for a short period of time, our model did not take them into account. We considered their cost to be marginal compared to the cost of the VNFs that are used before and after the reconfiguration. Nevertheless, we plot in Figure 3.13 the percentage of transient VNFs that are used only for the aim of the reconfiguration. A VNF is considered as transient if it is deployed neither before, nor after the reconfiguration, but during the steps of the reconfiguration.

Breaking-Bad has no reconfiguration step and therefore do not activate transient VNFs. As for Break-Free-HEUR, by design it does not activate any either: indeed, each reconfiguration step is only a transition from the initial state to the final state. Therefore, no transient VNF is needed for Break-Free-HEUR. By analogy to Break-Free-HEUR, there is also none with Break-Free-ILP with one step, since there is no intermediate step between the initial an the final state.

In Figure 3.13, we can see that Break-Free-ILP (with 2 and 3 steps) uses on average about 5% temporarily VNFs for pdh and between 11% and 12% for ta1. We can especially notice that the use of transient VNFs is stable between 2 and 3 reconfiguration steps and does not increase. Although our model does not minimise the use of transient VNFs, it deploys an acceptable number of them during reconfiguration. If this happens to be critical, then constraints in the model could be added to restrain the use of these VNFs. Another solution would be to use Break-Free-HEUR that has no transient VNF and similar performance.

3.6 Conclusion

In this chapter, we provide two solutions, `Break-Free-ILP` and `Break-Free-HEUR`, to reconfigure a set of requests which have to go through service function chains. The requests are routed greedily when they arrive, leading to a sub-optimal use of network resources, bandwidth, and virtual network functions. We compared our strategies with `Breaking-Bad` (that reconfigures to an optimal placement and routing solution with interruption of the requests) and `No-Reconf` (that never performs reconfiguration). For our 2 solutions, we study their impact on bandwidth usage, the deployment of VNFs as well as on the increase in the acceptance of requests during periods of heavy network congestion. We also study their efficiency according to the variation of reconfiguration frequencies and the maximum time limit allowed for each reconfiguration. For small and medium sized networks, `Break-Free-ILP` is fast and efficient. It reroutes the requests to an optimal or close to optimal solution in a few seconds while providing a make-before-break mechanism to avoid impacting the rerouted requests. The reconfiguration frequency can be adapted depending on the needs and the number of SFC arrivals and departures. The network operational cost is already greatly improved with only two steps of reconfigurations.

`Break-Free-HEUR` needs as an input the final desired placement and routing solution, and tries to greedily move the requests to that state. Therefore, it does not instantiate transient VNFs during reconfiguration steps. It is almost as efficient as `Break-Free-ILP` and moreover, it allows to solve efficiently large network instances, for which `Break-Free-ILP` cannot provide any solution.

The considered setting had splittable flows to allow a fast execution of the model. It is relevant to ask whether the solution provided works when each SFC can only pass on a single path. A partial answer is given in chapter 4, `slow-rescue` takes the problem with paths and adds latency constraints. The algorithm works in the same way and the solution is not degraded by the use of paths rather than splittable flows. However, the computation time does not allow it to be used in our next experiments. To complete the answer, we need to look at `Break-Free-HEUR`, which is not included in the next chapter. Its functioning is based on moving the flow of each SFC little by little towards an optimal allocation. By imposing the use of paths, it would be necessary to be able to reconfigure each SFC in one step, which is difficult to do when the network is congested. This method was therefore not adapted because it would have required a complete change of the heuristic.

Ensuring the performance of *make-before-break* reconfiguration in a SDN-NFV network using SFCs is essential before studying its performance in a more constrained context. In the next chapter we develop this technique in the context of network slicing.

Our algorithm could be improved by taking into account the capacity reserved for the backup paths. Indeed, the intermediate steps of the reconfiguration being relatively short, it would be interesting to use these paths during these steps. Taking these paths into account would also ensure that the reconfiguration respects the resilience constraints by not reconfiguring a request on one of its backup paths.

References

[AFT07] Brice Augustin, Timur Friedman, and Renata Teixeira. Measuring load-balanced paths in the internet. In *ACM Internet Measurement Conference (IMC)*, pages 149–160. ACM, 2007.

[B+14] Pankaj Berde et al. Onos: towards an open, distributed sdn os. In *Workshop on Hot topics in software defined networking*, pages 1–6. ACM, 2014.

[CLX+10] Zhiping Cai, Fang Liu, Nong Xiao, Qiang Liu, and Zhiying Wang. Virtual network embedding for evolving networks. In *IEEE Global Telecommunications Conference - GLOBECOM*, pages 1–5. IEEE, 2010.

[EMAL17] Vincenzo Eramo, Emanuele Miucci, Mostafa Ammar, and Francesco Giacinto Lavacca. An approach for service function chain routing and virtual function network instance migration in network function virtualization architectures. *IEEE/ACM Transactions on Networking (ToN)*, 25(4):2008–2025, 2017.

[FAPZ11] Ilhem Fajjari, Nadjib Aitsaadi, Guy Pujolle, and Hubert Zimmermann. VNR algorithm: A greedy approach for virtual networks reconfigurations. In *IEEE Global Telecommunications Conference - GLOBECOM*, pages 1–6. IEEE, 2011.

[GK07] Naveen Garg and Jochen Koenemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM Journal on Computing*, 37(2):630–652, 2007.

[GR18] Lingnan Gao and George N Rouskas. Virtual network reconfiguration with load balancing and migration cost considerations. In *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 2303–2311. IEEE, 2018.

[GTGM19a] A. Gausseran, A. Tomassilli, F. Giroire, and J. Moulierac. No interruption when reconfiguring my SFCs. In *IEEE International Conference on Cloud Networking (CloudNet)*, pages 1–6, 2019.

[GTGM19b] Adrien Gausseran, Andrea Tomassilli, Frederic Giroire, and Joanna Moulierac. Poster: Don't interrupt me when you reconfigure my service function chains. In *2019 IFIP Networking Conference (IFIP Networking)*, pages 1–2, 2019.

[GTGM21] A. Gausseran, A. Tomassilli, F. Giroire, and J. Moulierac. Don't interrupt me when you reconfigure my service function chains. *Computer Communications*, 2021.

[HFS+19] D. Harutyunyan, R. Fedrizzi, N. Shahriar, R. Boutaba, and R. Riggio. Orchestrating end-to-end slices in 5g networks. In *15th International Conference on Network and Service Management (CNSM)*, 2019.

[KLLT18] Tung-Wei Kuo, Bang-Heng Liou, Kate Ching-Ju Lin, and Ming-Jer Tsai. Deploying chains of virtual network functions: On the relation between link and server usage. *IEEE/ACM Transactions on Networking (TON)*, 26(4):1562–1576, 2018.

[LLZ+17] Junjie Liu, Wei Lu, Fen Zhou, Ping Lu, and Zuqing Zhu. On dynamic service function chain deployment and readjustment. *IEEE Transactions on Network and Service Management (IEEE TNSM)*, 14(3):543–553, 2017.

[MGT+15] Jon Matias, Jokin Garay, Nerea Toledo, Juanjo Unzilla, and Eduardo Jacob. Toward an SDN-enabled NFV architecture. *IEEE Communications Magazine*, 53(4):187–193, 2015.

[MSB+17] Wenrui Ma, Oscar Sandoval, Jonathan Beltran, Deng Pan, and Niki Pissinou. Traffic aware placement of interdependent NFV middleboxes. In *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 1–9, 2017.

[NKT19] Kyoomars Alizadeh Noghani, Andreas J. Kassler, and Javid Taheri. On the Cost-Optimality Trade-off for Service Function Chain Reconfiguration. In *IEEE International Conference on Cloud Networking (CloudNet)*, 2019.

[OWPT10] Sebastian Orlowski, Roland Wessäly, Michal Pióro, and Artur Tomaszewski. Sndlib 1.0—survivable network design library. *Networks: An International Journal*, 55(3):276–286, 2010.

[PDM+16] Stefano Paris, Apostolos Destounis, Lorenzo Maggi, Georgios S Paschos, and Jérémie Leguay. Controlling flow reconfigurations in SDN. In *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 1–9. IEEE, 2016.

[QN15] P. Quinn and T. Nadeau. Problem statement for service function chaining. RFC 7498, RFC Editor, April 2015.

[S+15] Sahel Sahhaf et al. Network service chaining with optimized network function embedding supporting service decompositions. *Computer Networks*, 93:492–505, 2015.

[SJG+17] Yu Sang, Bo Ji, Gagan R Gupta, Xiaojiang Du, and Lin Ye. Provably efficient algorithms for joint placement and allocation of virtual network functions. In *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 1–9. IEEE, 2017.

[TGHP18] Andrea Tomassilli, Frédéric Giroire, Nicolas Huin, and Stéphane Pérennes. Provably efficient algorithms for placement of Service Function Chains with ordering constraints. In *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 774–782, Honolulu, Hawai, US, 2018. IEEE.

[TTG13] Phuong Nga Tran and Andreas Timm-Giel. Reconfiguration of virtual network mapping considering service disruption. In *IEEE International Conference on Communications - ICC*, pages 3487–3492. IEEE, 2013.

[WFS+20] F. Wei, G. Feng, Y. Sun, Y. Wang, S. Qin, and Y. C. Liang. Network slice reconfiguration by exploiting deep reinforcement learning with large action space. *IEEE Transactions on Network and Service Management*, 2020.

CHAPTER 4

Network Slices Reconfiguration

Modern 5G networks promise more bandwidth, less delay, and more flexibility for an ever increasing number of users and applications, with Software Defined Networking, Network Function Virtualisation, and Network Slicing as key enablers. Within that context, efficiently provisioning the network and cloud resources of a wide variety of applications with dynamic user demand is a real challenge. We study here the network slice reconfiguration problem. Reconfiguring network slices from time to time reduces network operational costs and increases the number of slices that can be managed within the network. However, this affect the *Quality of Service* of users during the reconfiguration step. To solve this issue, we study solutions implementing a *make-before-break* scheme. In this chapter we propose new models and scalable algorithms (relying on column generation techniques) that solve large data instances in few seconds.

This chapter is part of the articles [GGJM20, GGJM22].

4.1 Introduction

The 5G technology is envisioned to allow a multi-service network supporting a wide range of communication scenarios with a diverse set of performance and service requirements. The concept of network slicing has been proposed to address these diversified service requirements. A network slice is an end-to-end logical network provisioned with a set of isolated virtual resources on a shared physical infrastructure [R+17, BGB+17]. Moreover, slicing allows an efficient usage of resources, as VNFs can be instantiated and released on demand by slices. Besides, slices can be deployed whenever there is a service request, reducing the network operator costs [BGB+17]. With all these key features, Network slicing will thus be a fundamental feature of 5G networks [R+17]. See subsection 0.2.3 and subsubsection 0.2.3.1 for more details on 5G networks and network slicing.

Slice description As the Network slicing paradigm is still relatively recent, the modelling of a slice can vary according to the context of the work in which it is used. In [01118] the ETSI defines a slice as *a description of a service-aware logical network composed of different physical or virtual network elements, resources and functions*. The *instance of a slice has resources allocated to it from the underlying network infrastructure and is independently managed and monitored by the tenant*. The tenant being *the entity that consumes a network slice instance from network slice providers*. This definition of a slice leaves a wide variety of choices for its modelling, however the description of its role specifies that it is designed to deliver a service. *Network slices provide a network through which a consumer can achieve his or her service delivery objectives*. The use of SFCs to model end-to-end services is already widespread as in [SZGS+18, TAM19, ZLF+17] and is consistent with the definition of a slice. The definition of the dynamicity of a slice can also lead to different interpretations. In [WFS+20, GZL20] the authors do not fix the capacities used by a slice, which may vary over time depending on the use made of it by its users. On the contrary, in our model, the capacities allocated to a slice are fixed and represent a maximum limit for it. If the slice is underused, the unused capacities cannot be used by another slice, they are reserved to allow perfect isolation and respect of the capacities constraints. The tenant of a slice is responsible for the use of the resources he reserves, in accordance with the ETSI definition. *Each slice being an independent entity should not impact the operations or lead service disruptions for other slices. In addition, it should be possible for a tenant operator of a slice to control and manage resources as well as allocate them to different users or flows with in its own network slice*. Taking these different parameters into account, in this chapter we model a slice by using a set of SFCs with a fixed resource demand throughout the slice's lifespan.

Dynamic resource allocation is one of the key challenges of network slicing. In this chapter we consider the problem of both rerouting traffic flows and improving the mapping of network functions onto nodes in the presence of dynamic traffic, with the objective of bringing the network back to a close to optimal operating state, in terms of resource usage. We adapt our *make-before-break* reconfiguration method in the context of network slicing. To the best of our knowledge, we are the first to propose scalable models to reconfigure network slices while implementing such mechanisms to avoid QoS degradation. See subsection 0.2.4 and subsubsection 0.2.4.2 for more details on reconfiguration and the *make-before-break* technique.

Our contributions in this chapter are as follows:

- We propose an Integer Linear Program (`slow-rescue`) to reconfigure, with a *make-before-break mechanism*, the routing and provisioning of a set of slices.
- We propose two *scalable* models, `rescue-ILP` and `rescue-LP`, with `rescue` standing for "REconfiguration of network Slices with ColUmn gEneration without interruption". Both are based on a decomposition model and are solved using column generation. Our algorithms reconfigure a given set of network slices from an initial routing and placement of network functions to another solution that improves the usage of the network resources (both in terms of links and VNFs). Our solutions scale on large networks as we succeeded in solving data instances with 65 nodes and 108 links, and a hundred of network slices in few seconds, a lot faster than with a classic compact Integer Linear Program (ILP) formulation such as `slow-rescue`.
- We show that our solutions allow *the decrease of the network cost* without degrading the QoS (as the network slices are not interrupted thanks to the *make-before-break* approach) in moderate running times. Moreover, we can manage more network slices when the network is congested compared to solutions without any reconfiguration.

In this chapter the optimisation problem is similar to the one in the previous chapter. The modelling of the slices is done by using SFC and the objective of the model is the minimisation of the link usage and the deployment of VNFs. However, in this chapter the flow of each SFC is not splittable and must follow a single path. The addition of a delay constraints makes the problem harder to solve and allows for different latency classes depending on the type of slices. The ILP `slow-rescue` is the adaptation of the `Break-Free-ILP` ILP of the previous chapter and allows to compare our new solutions with the one of the previous chapter. Finally, the test scenarios are more realistic and based on a typical daily variation of traffic in an ISP network (Figure 4.1). It allows us to test our solutions with an evolving network load.

4.2 Related Work

In the last years, a large corpus of works has studied the deployment and management of network services.In particular, the problem of jointly routing demand and provisioning them with their needed VNFs has attracted a lot of attention. A large number of efficient algorithms and optimisation models have been proposed in order to minimise setup cost [CLENR15] or take into account the chaining constraints [HJG18]. The problem of how to deploy and manage network services conceived as a chain of VNFs is summarised in chapter 2 In [LPMK18] the authors study a multi-objective slice placement algorithm. They take into account fairness of delay, computational power, traffic usage and total utility maximisation. Their objective can be set to balance different needs while being pareto optimal. Pozza *et al.* [PPR+19] study a slice placement algorithm collocating VNFs with the objective of minimising inter-VNFs traffic and latency within a slice.

Most of these works have only considered scenarios in which, when a service is deployed, its route and used virtual resources are not changed during its lifetime. However, the churn of network services makes that even an optimal service deployment may lead to sub-optimal use of network resources after a certain time, when some services are no longer there.

Inspired by the classic defragmentation mechanism in optical networks [WM13], it has been proposed to carry out reconfigurations of network and virtual resources regularly in order to bring

$G = (V, L)$	Network: V represents the node set and L the link set.
C_ℓ	Bandwidth link capacity of $\ell \in E$.
DELAY_ℓ	Link delay of $\ell \in L$.
C_v	Resource node capacity (e.g., CPU, memory, and disk) of node $v \in V$.
Δ_f	Number of bandwidth units required by function $f \in F$.
$c_{v,f}$	Usage cost of function $f \in F$, which also depends on node v.
Each demand $d \in D$ is modeled by a quintuplet	:
$(v_{\text{SRC}}, v_{\text{DST}})$	Source and destination nodes,
c_d	Ordered network function sequence for demand d,
$f_i^{c_d}$	$i-th$ function of chain c_d,
BW_d	Required bandwidth units,
DELAY_d	Maximum required delay for the slice.

Table 4.1 – Notations

the network closer to an optimal state of operation. The goals can be diverse: optimising network usage, granting more requests, modifying the capacities of flows already allocated on the network or even to overcome network failures.

The readjustment of Service Function Chains (SFCs) has been studied in Liu *et al.* [LLZ+17]. The latter formulates an ILP and a column generation model in order to jointly optimise the deployment of the SFCs of new users and the readjustment of the SFCs already provisioned in the network while considering the trade-off between resource consumption and operational overhead.

Gao and Rouskas [GR18] considered the reconfiguration of virtual networks. They proposed online algorithms to minimise the maximum utilization of substrate nodes and links while bounding the number of virtual nodes that have to be migrated.

Similarly, all works on reconfiguration of virtual resources (virtual networks, slices or service function chains) include a cost expressing the degradation of the client's QoS. On the contrary, our goal is to *avoid this QoS degradation* by proposing a *make-before-break* mechanism, in which the new route is reserved and the new virtual resources are installed before the slice is reconfigured. A similar mechanism has been proposed in chapter 3. However, this chapter if the first to present a scalable decomposition model based on column generation to solve it.

4.3 Problem Statement and Notations

4.3.1 Definitions

We consider the network as a directed capacitated graph $G = (V, L)$ where V represents the node set and L the link set. The resource node capacity (e.g., CPU, memory, and disk) of node $v \in V$ is denoted by C_v. Link transport capacity is represented by C_ℓ and DELAY_ℓ is the delay of link $\ell \in L$. $t \in T$ is the number of steps used for the reconfiguration. Δ_f is the number of bandwidth units required by function $f \in F$.

Following, e.g., [LPMK18, PPR+19], a slice can be modeled by a set of requests. Each demand request $d \in D$ is modeled with a quintuplet: v_{SRC} the source, v_{DST} the destination, c_d the ordered sequence of network functions that need to be performed, where $f_i^{c_d}$ is the $i-th$ function of chain c_d. BW_d denotes the required units of bandwidth of demand d, and DELAY_d the delay requirement of demand d. Table 4.1 summarizes the notations used throughout the chapter.

In a dynamic scenario with no information on future traffic, each demand is routed individually while *minimising the network operational cost* defined by the weighted sum of link bandwidth and VNF usage costs (licenses, energy consumption, etc). As requests come and leave over time, allocations that are locally optimal at a given instant can bring the network in a global sub-optimal state. Our goal is to reconfigure the network to improve resource usage and therefore the operational costs. In doing so, we use the *make-before-break* mechanism to avoid network service disruption due to traffic rerouting. Reconfiguring a demand involves rerouting its path and/or reallocating the VNFs it's using to other locations

$|c_d|$ denotes the number of VNFs in the chain c_d of the demand.

4.4 ILP Model: `slow-rescue`

Model.

The compact ILP model, `slow-rescue`, is an Integer Linear Program based on the notion of layered graph described in section 2.4. This model is similar to the `Break-Free-ILP` model explained in subsection 3.4.2 from chapter 3. It differs by the addition of delays, the use of binary variables and by a more constraining reconfiguration. Unlike `Break-Free-ILP`, when a reconfigured request uses the same link twice between two steps, the link is counted twice.

Variables:

- $\varphi_{\ell,i}^{d,t} \in [0,1]$ is the amount of flow on Link ℓ in Layer i at time step t for demand d.
- $\alpha_{v,i}^{d,t} \in [0,1]$ is the amount of flow on node v in layer i at time step t for demand d.
- $x_{\ell,i}^{d,t} \in [0,1]$ is the maximum amount of flow on Link ℓ in Layer i at time steps t and $t-1$ for demand d.
- $y_{v,i}^{d,t} \in [0,1]$ is the maximum amount of flow on node v in layer i at time steps t and $t-1$ for demand d.
- $\omega^{d,t} \in [0,1]$, where $\omega^{d,t} = 0$ if the allocation of demand d is modified between time steps t or $t-1$.
- $z_{v,f} \in [0,1]$, where $z_v^f = 1$ if function f is activated on node v at time step $|T|$ in the final routing.

The optimisation model starts with the initial configuration (an initial placement of VNFs on the nodes, and a valid routing for the slices) as an input. Thus, for each demand $d \in D$, at initial time step 0, variables $\varphi_{\ell,i}^{d,0}$ (for each link $\ell \in L$, layer $i \in \{0,...,|c_d|\}$) and $\alpha_{v,i}^{d,0}$ (for each node $v \in V$, layer $i \in \{0,...,|c_d|\}$) are known.

Objective: minimise the amount of network resources consumed during the last reconfiguration time step $|T|$.

$$\min \sum_{d \in D} \sum_{\ell \in L} \sum_{i=0}^{|c_d|} \text{BW}_d \, \varphi_{\ell,i}^{d,T} + \beta \sum_{v \in V} \sum_{f \in F} c_{v,f} \, z_{v,f} \tag{4.1}$$

The parameter $\beta \geq 0$ specified by the network administrator accounts for different scales over which the functions' activation cost is put in relationship with the network bandwidth cost. β represents how many *TB/s* of data can be sent when using a dollar. Its dimension thus is $TB/dollars$, giving that our objective function formally expresses a bandwidth.

Constraints:

Flow conservation constraints. The following equations are the usual flow conservation constraints considering the graph layer technique as explained previously. Note that the traffic can enter at the top layer, and only exits at the bottom layer. For each demand $d \in D$, node $v \in V$, time step $t \in T$.

$$\sum_{\ell \in \omega^+(v)} \varphi^{d,t}_{\ell,0} - \sum_{\ell \in \omega^-(v)} \varphi^{d,t}_{\ell,0} + \alpha^{d,t}_{v,0} = \begin{cases} 1 \text{ if } v = v_{\text{SRC}} \\ 0 \text{ else} \end{cases} \tag{4.2}$$

$$\sum_{\ell \in \omega^+(v)} \varphi^{d,t}_{\ell,|c_d|} - \sum_{\ell \in \omega^-(v)} \varphi^{d,t}_{\ell,|c_d|} - \alpha^{d,t}_{v,|c_d|-1} = \begin{cases} -1 \text{ if } v = v_{\text{DST}} \\ 0 \text{ else} \end{cases} \tag{4.3}$$

$$\sum_{\ell \in \omega^+(v)} \varphi^{d,t}_{\ell,i} - \sum_{\ell \in \omega^-(v)} \varphi^{d,t}_{\ell,i} + \alpha^{d,t}_{v,i} - \alpha^{d,t}_{v,i-1} = 0 \qquad 0 < i < |c_d|. \tag{4.4}$$

Node usage over two consecutive time periods. For $d \in D$, $v \in V$, $i \in \{0, ..., |c_d| - 1\}$, $t \in T$. If d used link ℓ either at time step t or $t-1$, then $y^{d,t}_{v,i}$ is forced to 1. If d is modified between these two steps, then $\omega^{d,t} = 0$ and one (or both) of the two variables $\alpha^{d,t}_{v,i}$ or $\alpha^{d,t-1}_{v,i}$ should be equal to 0. If d keeps the same allocation between t and $t-1$, then $\omega^{d,t} = 1$ and $y^{d,t}_{v,i}$ is forced to 1 if node v is used and can be equal to 0 otherwise.

$$\alpha^{d,t}_{v,i} \leq y^{d,t}_{v,i} \tag{4.5}$$

$$\alpha^{d,t-1}_{v,i} \leq y^{d,t}_{v,i} \tag{4.6}$$

$$\alpha^{d,t}_{v,i} + \alpha^{d,t-1}_{v,i} - \omega^{d,t} \leq y^{d,t}_{v,i}. \tag{4.7}$$

Link usage over two consecutive time periods. For $d \in D, \ell \in L$, Layer $i \in \{0, ..., |c_d|\}$, $t \in T$. The arguments to justify these constraints are the same as the ones for node usage over two consecutive time periods.

$$\varphi^{d,t}_{\ell,i} \leq x^{d,t}_{\ell,i} \tag{4.8}$$

$$\varphi^{d,t-1}_{\ell,i} \leq x^{d,t}_{\ell,i} \tag{4.9}$$

$$\varphi^{d,t}_{\ell,i} + \varphi^{d,t-1}_{\ell,i} - \omega^{d,t} \leq x^{d,t}_{\ell,i}. \tag{4.10}$$

Make Before Break - Node capacity constraints. The capacity of a node $v \in V$ is shared between each layer and cannot exceed C_v considering the resources used over two consecutive time periods. For each Node $v \in V$, time step $t \in T$.

$$\sum_{d \in D} \text{BW}_d \sum_{i=0}^{|c_d|-1} \Delta_{f^{c_d}_i} \, y^{d,t}_{v,i} \leq C_v. \tag{4.11}$$

Make Before Break - Link capacity constraints. The capacity of a link $\ell \in L$ is shared between each layer and cannot exceed C_ℓ considering the resources used over two consecutive time periods. For $\ell \in L$, $t \in T$.

$$\sum_{d \in D} \text{BW}_d \sum_{i=0}^{|c_d|} x_{\ell,i}^{d,t} \leq C_\ell. \tag{4.12}$$

Delay constraint. The sum of the delays of all links traversed by the flow of a demand d must not exceed the maximum delay accepted by the demand. For $d \in D$, $t \in T$

$$\sum_{i=0}^{|c_d|} x_{\ell,i}^{d,t} \text{ DELAY}_\ell \leq \text{DELAY}_d. \tag{4.13}$$

Function activation. To know which functions are activated on which nodes in the final routing. For $v \in V$, $f \in F$, $d \in D$, and $i \in \{0, ..., |c_d| - 1\}$,

$$\alpha_{v,i}^{d,T} \leq z_{v,f_i^{c_d}}. \tag{4.14}$$

Reconfiguration - node modification constraints. To know if the allocation of a demand d is modified on nodes between two consecutive time periods.
For $d \in D$, $v \in V$, $i \in \{0, ..., |c_d|\}$, $t \in T$.

$$\omega^{d,t} \leq 1 + \alpha_{v,i}^{d,t} - \alpha_{v,i}^{d,t-1} \tag{4.15}$$

$$\omega^{d,t} \leq 1 + \alpha_{v,i}^{d,t-1} - \alpha_{v,i}^{d,t}. \tag{4.16}$$

Reconfiguration - link modification constraints. To know if the routing of a demand d is modified on links between two consecutive time periods.
For $d \in D$, $\ell \in L$, $i \in \{0, ..., |c_d|\}$, $t \in T$.

$$\omega^{d,t} \leq 1 + \varphi_{\ell,i}^{d,t} - \varphi_{\ell,i}^{d,t-1} \tag{4.17}$$

$$\omega^{d,t} \leq 1 + \varphi_{\ell,i}^{d,t-1} - \varphi_{\ell,i}^{d,t}. \tag{4.18}$$

As we will see in section 4.6, although effective, the compact ILP model `slow-rescue` does not scale on large networks or with many slices. We therefore propose an alternative using column generation: `rescue-ILP` and `rescue-LP` (for REconfiguration of network Slices with ColUmn gEneration with ILP or LP pricing).

4.5 The column generation technique and our model

Column generation (CG) is a model allowing the solution of an optimisation model without explicitly introducing all variables, see section 1.2 for an explanation. It thus often allows to solve larger instances of the problem than a compact model, in particular, with an exponential number of variables.

To model our problem using column generation we tested two different solutions.

4.5.1 A first CG-based algorithms

In the first model each PP handles the reconfiguration of a single demand. The PPs are therefore a reformulation of `slow-rescue` which seeks to find a valid reconfiguration for one demand and where only the objective changes. The capacity constraints are kept inside the PPs because if a request passes several times on a link during several steps of the reconfiguration, it can saturate the link by itself and thus there is a risk to have a column which is invalid and which will not be used by the RMP. The RMP on the other hand combines columns, each of which is the complete reconfiguration of a demand. The RMP also contains capacity constraints to ensure that capacities are respected between different requests.

4.5.1.1 First Master Problem

Model.
In this model the role of the RMP is to connect the different reconfigurations of the demands. Each demand has a set of columns representing a valid reconfiguration,generated by a PP. The RMP must find the optimal configuration of the columns to ensure the best global reconfiguration of the set of slices by choosing on column per demand.

Parameters:
• $\delta_\ell^{r,t}$ is the number of times the link ℓ appears on reconfiguration r between time step t and $t+1$. At $t = T$ only the time step T is used
• $\theta_{i,v}^{r,t}$ is the number of times the bode v is used as a VNF on reconfiguration r on layer i between time step t and $t+1$. At $t = T$ only the time step T is used
Variables:
• $\varphi_r^d \in [0,1]$ is the amount of flow of demand d on reconfiguration r.

We assume an initial configuration is provided with fixed values for φ_r^d. The optimisation model is written as follows.

Objective: minimise the amount of network resources consumed during the last reconfiguration time step T.

$$\min \sum_{d\in D}\sum_{p\in P_d}\sum_{\ell\in L} \mathrm{BW}_d\, \varphi_r^d\, \delta_\ell^{r,T} + \beta \sum_{V\in V^{\mathrm{VNF}}}\sum_{f\in F} c_{v,f}\, z_{v,f} \tag{4.19}$$

Constraints:

One reconfiguration constraint. For $d \in D$.

$$\sum_{r\in R_d} \varphi_r^d = 1. \tag{4.20}$$

Make Before Break - Node capacity constraints. The capacity of a node v in V is shared between each layer and cannot exceed C_v considering the resources used over two consecutive time periods. For $v \in V^{\mathrm{VNF}}$, $t \in T$.

$$\sum_{d\in D}\sum_{r\in R_d}\sum_{i=0}^{|c_d|-1} \varphi_r^d \cdot \theta_{i,v}^{r,t} \cdot \mathrm{BW}_d \cdot \Delta_{f_i^{c_d}} \le C_v. \tag{4.21}$$

Make Before Break - Link capacity constraints. The capacity of a link $\ell \in L$ is shared between each layer and cannot exceed C_ℓ considering the resources used over two consecutive time periods. For $\ell \in L$, $t \in T$,

$$\sum_{d \in D} \sum_{r \in R_d} \text{BW}_d \, \varphi_r^d \, \delta_\ell^{r,t} \leq C_\ell. \tag{4.22}$$

Function activation. To know which functions are activated on which nodes in the final routing. For $v \in V$, $f \in F$, $d \in D$, $i \in \{0, ..., |c_d| - 1\}$,

$$\varphi_r^d \cdot \theta_{i,v}^{r,T} \leq z_{v,f_i^{c_d}}. \tag{4.23}$$

4.5.1.2 First Pricing Problem

Model.
The variables are the same as `slow-rescue`, with the exception that they are only related to the demand handled by the PP.
Variables:
- $\varphi_{\ell,i}^t \in [0, 1]$ is the amount of flow on Link ℓ in Layer i at time step t.
- $\alpha_{v,i}^t \in [0, 1]$ is the amount of flow on node v in layer i at time step t.
- $x_{\ell,i}^t \in [0, 1]$ is the maximum amount of flow on Link ℓ in Layer i at time steps t and $t - 1$.
- $y_{v,i}^t \in [0, 1]$ is the maximum amount of flow on node v in layer i at time steps t and $t - 1$.
- $\omega^t \in [0, 1]$, where $\omega^t = 0$ if the allocation is modified between time steps t or $t - 1$.
- $z_{v,f} \in [0, 1]$, where $z_v^f = 1$ if function f is activated on node v at time step $|T|$ in the final routing.

The constraints are also the same as `slow-rescue`, but only related to the demand handled by the PP.
As in `slow-rescue`, the optimisation model starts with the initial configuration (an initial placement of VNFs on the nodes, and a valid routing for the demand) as an input. Thus, at initial time step 0, variables $\varphi_{\ell,i}^0$ (for each link $\ell \in L$, layer $i \in \{0, ..., |c_d|\}$) and $\alpha_{v,i}^0$ (for each node $v \in V$, layer $i \in \{0, ..., |c_d|\}$) are known.

The objective, however, changes. It takes into account the dual values of the RMP constraints, as explained in Equation 1.9.

Objective: minimise the amount of network resources consumed during the last reconfiguration time step $|T|$.

$$\begin{aligned}
\min \sum_{\ell \in L} \sum_{i=0}^{|c_d|} \varphi_{\ell,i}^T \, \text{BW}_d (1 + \mu_{\ell,T}^{(4.22)}) + \sum_{t=1}^{T} \sum_{\ell \in L} \sum_{i=0}^{|c_d|} x_{\ell,i}^t \, \text{BW}_d \, \mu_{\ell,t-1}^{(4.22)} \\
+ \text{BW}_d \sum_{t=1}^{T} \sum_{v \in V^{\text{VNF}}} \mu_{v,t-1}^{(4.21)} \sum_{i=0}^{|c_d|-1} \Delta_{f_i^{c_d}} \, y_{v,i}^t \\
- \mu_{d,t}^{(4.20)} + \beta \sum_{v \in V^{\text{VNF}}} \sum_{f \in F} c_{v,f} \, z_{v,f} \, \mu_{d,v,f}^{(4.23)}
\end{aligned} \tag{4.24}$$

4.5.2 Description of our CG-based algorithms: `rescue-ILP` and `rescue-LP`

In the second model the reconfiguration computation is transferred to the RMP. The PPs only find one valid allocation for each request. A column is therefore only one path and the RMP combines the paths to ensure that all requests can be reconfigured from one path to another. In this model there are no capacity constraints within the PPs simply because saturating a link or node with a single request in one step is highly unlikely, if not impossible in our scenario. The PPs are therefore much simpler problems than in the first model. The RMP is more complicated to solve.

We have chosen to study only the second model after testing both. In the case of the second model the majority of the execution time comes from the PPs and even with a more complex RMP, the time spent executing it at each iteration is negligible. For the first model the PPs take longer to execute and this is deleterious for the convergence time as well as for the quality of the final solution when the execution time is bounded.

In the context of our problem, the master problem (MP) seeks a possible global reconfiguration for all slices with a path-formulation. In the RMP, only a subset of potential paths is used for each slice. At the initialisation, the set of paths is the one used before reconfiguration. Each (PP) then generates a new path for a request, together with the placement of the VNFs. During a reconfiguration, slices are migrated from one path to another. Note that, as the execution of each pricing problem is independent of the others, their solutions can be obtained in parallel.

It should be noted that in the majority of the state of the art of column generation, for each new column added to the RMP, only one set of variables is added. In our case two sets of variables are added as well as a constraint. The 4.27 constraint is used to bind the φ_p and y sets of variables. Its dual value does not appear in the pricing objective because it applies to a particular column and even if it did appear in the pricing, the left and right signs of the constraint would cancel each other.

This addition of a constraint should not be considered as row generation. Without going into detail, row generation is an optimisation technique in which constraints (rows) are added to the problem in order to constrain it progressively and find a valid solution quickly. In our case the added constraints do not constrain the problem further but simply binds the sets of added variables.

4.5.2.1 Master Problem of `rescue-ILP` and `rescue-LP`

Model.
This master problem is used both by `rescue-ILP` and `rescue-LP`.

Parameters:
- δ_ℓ^p is the number of times the link ℓ appears on path p.
- $\theta_{i,v}^p$ = 1 if node v is used as a VNF on path p on layer i.

Variables:
- $\varphi_p^{d,t} \in [0,1]$ is the amount of flow of demand d on path p at time step t.
- $y_p^{d,t} \in [0,1]$ is the maximum amount of flow of demand d on path p between time step $t-1$ and t.

We assume an initial configuration is provided with fixed values for $\varphi_p^{d,0}$. The optimisation model is written as follows.

Objective: minimise the amount of network resources consumed during the last reconfiguration time step T.

$$\min \sum_{d \in D} \sum_{p \in P_d} \sum_{\ell \in L} \text{BW}_d \, \varphi_p^{d,T} \, \delta_\ell^p + \beta \sum_{V \in V^{\text{VNF}}} \sum_{f \in F} c_{v,f} \, z_{v,f} \tag{4.25}$$

Constraints:

One path constraint. For $d \in D$, time step $t \in T$.

$$\sum_{p \in P_d} \varphi_p^{d,t} = 1. \tag{4.26}$$

Path usage over two consecutive time periods. For $d \in D$, $p \in P_d$, $t \in T$.

$$\varphi_p^{d,t} \leq y_p^{d,t} \text{ and } \varphi_p^{d,t} \leq y_p^{d,t-1}. \tag{4.27}$$

Make Before Break - Node capacity constraints. The capacity of a node v in V is shared between each layer and cannot exceed C_v considering the resources used over two consecutive time periods. For $v \in V^{\text{VNF}}$, $t \in T$.

$$\sum_{d \in D} \sum_{p \in P_d} \sum_{i=0}^{|c_d|-1} y_p^{d,t} \cdot \theta_{i,v}^p \cdot \text{BW}_d \cdot \Delta_{f_i^{c_d}} \leq C_v. \tag{4.28}$$

Make Before Break - Link capacity constraints. The capacity of a link $\ell \in L$ is shared between each layer and cannot exceed C_ℓ considering the resources used over two consecutive time periods. For $\ell \in L$, $t \in T$,

$$\sum_{d \in D} \sum_{p \in P_d} \text{BW}_d \, y_p^{d,t} \, \delta_\ell^p \leq C_\ell. \tag{4.29}$$

Function activation. To know which functions are activated on which nodes in the final routing. For $v \in V$, $f \in F$, $d \in D$, $i \in \{0, ..., |c_d| - 1\}$,

$$y_p^{d,T} \, \theta_{i,u}^p \leq z_{v,f_i^{c_d}}. \tag{4.30}$$

4.5.2.2 ILP Pricing Problem of `rescue-ILP`

The pricing problem searches for a possible placement for the slice. Since a reconfiguration can be done in several steps, a pricing problem is launched for each demand, at each time step.

Parameters:

• μ are the dual values of the master's constraints. The number written in upperscript is the reference of the master's constraints.

Variables:

• $\varphi_{\ell,i} \in [0, 1]$ is the amount of flow on link ℓ in layer i.

• $\alpha_{v,i} \in [0, 1]$ is the amount of flow on node v in layer i.

Objective: minimise the amount of network resources consumed for the demand d at time t.

$$\min \sum_{\ell \in L} \sum_{i=0}^{|c_d|} \varphi_{\ell,i} \, \text{BW}_d (1 + \mu_{\ell,t}^{(4.29)}) + \text{BW}_d \sum_{v \in V^{\text{VNF}}} \mu_{v,t}^{(4.28)} \sum_{i=0}^{|c_d|-1} \Delta_{f_i^{c_d}} \, \alpha_{v,i} - \mu_{d,t}^{(4.26)} + \beta \sum_{v \in V^{\text{VNF}}} \sum_{f \in F} c_{v,f} \, z_{v,f} \, \mu_{d,v,f}^{(4.30)} \quad (4.31)$$

where $\mu_{d,v,f}^{(4.30)} = 0$ when $t \neq T$, see constraints (4.30).

Constraints:

Flow conservation constraints for the demand d. For $v \in V^{\text{VNF}}$.

$$\sum_{\ell \in \omega^+(v)} \varphi_{\ell,0} - \sum_{\ell \in \omega^-(v)} \varphi_{\ell,0} + \alpha_{v,0} = \begin{cases} 1 \text{ if } v = v_{\text{SRC}} \\ 0 \text{ otherwise} \end{cases} \quad (4.32)$$

$$\sum_{\ell \in \omega^+(v)} \varphi_{\ell,|c_d|} - \sum_{\ell \in \omega^-(v)} \varphi_{\ell,|c_d|} - \alpha_{v,|c_d|-1} = \begin{cases} -1 \text{ if } v = v_{\text{DST}} \\ 0 \text{ otherwise} \end{cases} \quad (4.33)$$

$$\sum_{\ell \in \omega^+(v)} \varphi_{\ell,i} - \sum_{\ell \in \omega^-(v)} \varphi_{\ell,i} + \alpha_{v,i-1} - \alpha_{v,i-1} = 0 \qquad 0 < i < |c_d|. \quad (4.34)$$

Delay constraints. The sum of the link delays of the flow must not exceed the delay requirement of demand d.

$$\sum_{i=0}^{|c_d|} \varphi_{\ell,i} \, \text{DELAY}_\ell \leq \text{DELAY}_d. \quad (4.35)$$

Function activation. To know which functions are activated on which nodes. For $v \in V^{\text{VNF}}$, $f \in F$, layer $i \in \{0, ..., |c_d| - 1\}$.

$$\alpha_{v,i} \leq z_{v,f_i^{c_d}}. \quad (4.36)$$

Location constraints. A node may be enabled to run only a subset of the virtual network functions. For $v \in V^{\text{VNF}}$, $i \in \{0, ..., |c_d| - 1\}$, if the $(i+1)^{\text{th}}$ function of c_d cannot be installed on v, we have

$$\alpha_{v,i} = 0. \quad (4.37)$$

4.5.2.3 LP Pricing Problem of `rescue-LP`

The difference between `rescue-ILP` and `rescue-LP` comes from the pricing problem, which is integer for `rescue-ILP` and fractional for `rescue-LP`. Indeed, the execution time of the CG algorithm is divided into the resolutions of: (1) the multiple PPs, (2) the multiple relaxations of the RMP, and (3) the ILP of the MP. In our experiments, the time spent in (1) represents more than 90% of the whole execution time. To reduce this computational time, we propose `rescue-LP` that solves a relaxation of the pricing problem with fractional flows. The Master Problem of `rescue-LP` is the same as previously described. In the vast majority of cases, even with no

constraint to force integral flows, the PP outputs an integral path that can be directly integrated into the RMP. If the LP gives a fractional flow, we use the ILP PP of `rescue-ILP` to get an integral path.

4.6 Numerical Results

We conducted several experiments in order to show the efficiency of our Column Generation algorithms, `rescue-ILP` (with ILP pricing) and `rescue-LP` (with LP pricing). We compare their results with three solutions:

- `no-reconf` which places and removes the slices without reconfiguring the network,
- `slice-wreck` which regularly reconfigures the network but with interruptions. It solves the problem of static routing and computes the optimal allocation to a set of slices (reformulation of `Breaking-Bad` from section 3.4 of chapter 3 explained in subsection 2.5.2 from chapter 2)
- `slow-rescue`, our (slower) compact ILP reconfiguring slices without interruptions (similar to `Break-Free-ILP` explained in subsection 3.4.2 from chapter 3).

The solution `slice-wreck` computes an optimal (static) routing and placement solution and reconfigures to that new solution. This algorithm gives a bound for the best solution we can reach with the make-before-break approach.

We first show the efficiency of the CG models in terms of execution times and gains in network costs compared to the ILP, and of accuracy using static scenarios in subsection 4.6.2. We discuss the impact of the number of reconfiguration steps in subsection 4.6.3. Then, we consider dynamic scenarios in which requests arrive and leave over time in subsection 4.6.4. We discuss the gains provided by the reconfiguration by studying the impact on several metrics while varying the reconfiguration frequency in subsection 4.6.5. The scalability of our solutions are proven in subsection 4.6.6. The gains of parallelisation are shown in subsection 4.6.7 and the impact of slice delay constraints in subsection 4.6.8.

4.6.1 Data sets

Topologies. We conduct simulations on three real-world topologies from SNDlib [OWPT10] of different sizes: `pdh` (11 nodes, 34 links), `ta1` (24 nodes, 55 links), and `ta2` (65 nodes, 108 links). The compact model, `slow-rescue`, succeeds to find solutions only for small networks like `pdh`. We thus first compare the results on `pdh` and `ta1` to show the efficiency of the CG models in terms of execution times and gains in network costs. We then use the two larger networks `ta1` and `ta2` for our study of large dynamic scenarios.

Slice demands. Each slice is composed of a random number of demands chosen uniformly between 1 and 5. Each of the demands has to implement a chain of up to 5 VNFs, requires a specific amount of bandwidth, and has a latency constraints. We consider four different types of demands corresponding to four services: Video Streaming, Web Service, VoIP, and online gaming. The characteristics of each service are reported in Table 4.2 and are taken from [STV15]. The bandwidth usage was chosen according to the distribution of Internet traffic described in [CIS15]. The latency requirements are expressed in milliseconds and represent the maximum delay between the source and destination.

Slice Types	VNF chain	Latency	bw (Mbps)
Web Service	NAT-FW-TM-WOC-IDPS	10ms	100
Video Streaming	NAT-FW-TM-VOC-IDPS	5ms	256
VoIP	NAT-FW-TM-FW-NAT	3.5ms	64
Online Gaming	NAT-FW-VOC-WOC-IDPS	2.5ms	50

Table 4.2 – Characteristics of network slices

Simulations have been conducted on an Intel Xeon E5520 with 24GB of RAM.

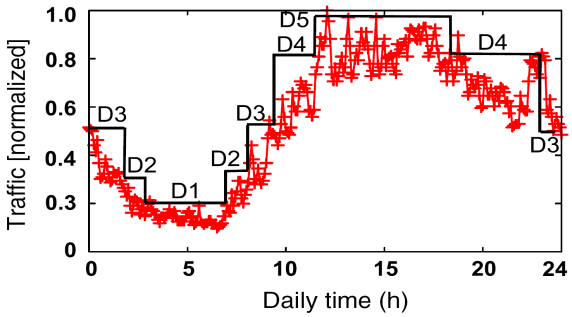


Figure 4.1 – Period approximation of traffic variation

Traffic distribution. Our goal is to study the impact of reconfiguration for different network usages. Indeed, when the traffic is low or medium, all slices can be served and reconfigurations improve the network usage (links and VNFs). However, when the traffic is high and if some links are congested, reconfiguration also helps to prevent rejecting slices. To model the typical daily variation of traffic in an ISP network, we used the traffic distribution from a trace of the Orange network (Figure 4.1). We adapted the churn rate of slices during time in order to obtain a similar level of traffic. This distribution is decomposed into five different levels of traffic demands: D1 to D5, D1 being the lowest one (from 3 to 6 am) and D5 the highest one (from 11 am to 6 pm). Each level of traffic corresponds to a different average number of slices: from 10 for D1, 22 for D2, 35 for D3, 52 for D4 to 60 for D5 with an average of 3 SFCs per slice.

Finally we will mostly use 3-steps reconfiguration, except for `pdh` where it will be a 2-steps reconfiguration (to be able to compare our algorithms with `slow-rescue` which does not give results with 3 steps). The reason for the choice of the 3-steps reconfiguration is developed in sub-subsection 4.6.3.

4.6.2 Efficiency of our algorithms with different traffic matrices

We evaluate the efficiencies of `rescue-ILP` and `rescue-LP` by comparing them with `slow-rescue`. We consider the `pdh` and `ta1` networks for the five different traffic levels during the day of Figure 4.1. We consider here a static scenario. For each network and for each level of traffic, we first place a corresponding number of slices one by one. We then carry out a reconfiguration to reroute the slices in order to improve the network usage. First, all the slices of $D1$ are placed, and then all reconfigured at once. Then, the same process is repeated for $D2$ until $D5$.

4.6.2.1 Execution times

We report the execution times of a reconfiguration in two steps for `slow-rescue`, `rescue-ILP` and `rescue-LP` in Figure 4.2. Each value is an average over 10 experiments. We set a time limit of one hour. When the time limit is reached, the algorithms return the best solution found during this delay. This solution is often not too far from the optimal solution, or even optimal as the solver tries to prove the optimality of the solution. For `pdh`, `slow-rescue` finds the optimal solution only for the period D1 and a small number of runs for D2. For all the other ones, it reaches the time limit. For the larger network `ta1`, the compact ILP was not able to find any feasible solution, even for D1 with the lowest number of slices. Column generation models are a lot faster. The execution times are below 120 s for both networks for any time period. Moreover, the models scale well as their execution times increase in a linear way. We observe that `rescue-LP` is a lot faster than `rescue-ILP` (beware of the log y-scale): for `ta1`, `rescue-LP` needs from 4 s to around 70 s, while the execution times of `rescue-ILP` are between 20 s and 120 s. It confirms that using LPs instead of ILPs when possible very significantly speeds up the resolution of the pricing problems, and, then, of the whole method.

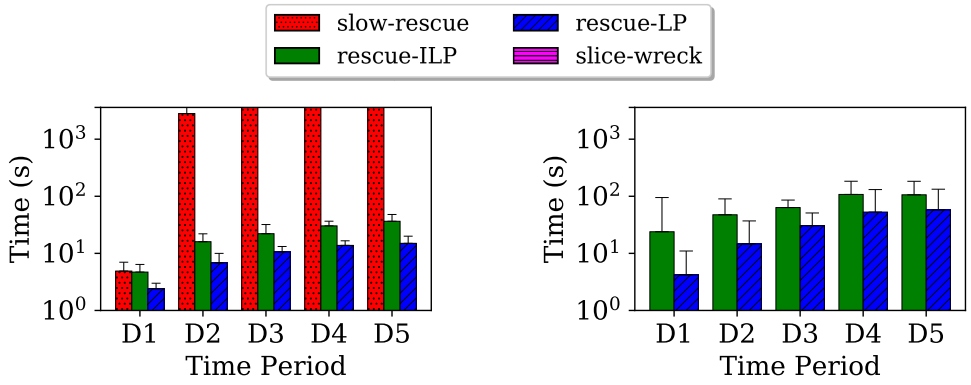


Figure 4.2 – Execution times for `pdh` (left) and for `ta1` (right).

4.6.2.2 Gains in network cost

We now compare the improvement in terms of network cost obtained after a reconfiguration in Figure 4.3. Results are given for each time period for `pdh` and `ta1`. Recall that the network cost is a weighted sum of the VNF and network costs (which are also plotted in Figure 4.4 and Figure 4.5, respectively).

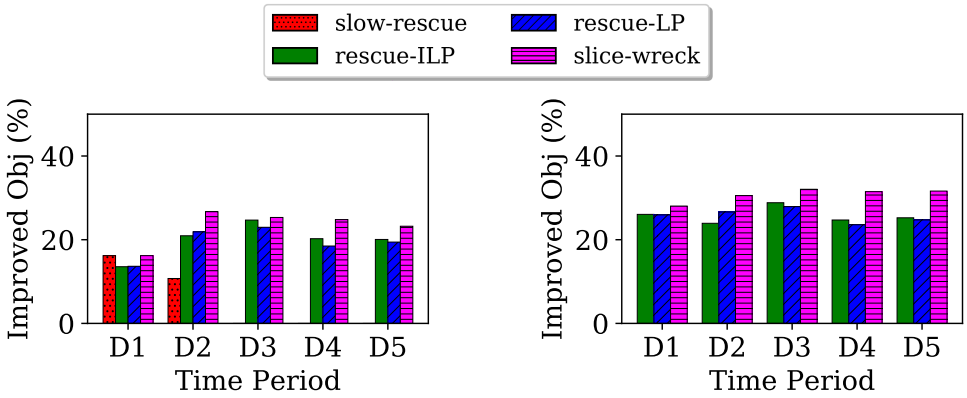


Figure 4.3 – Gains in network cost for `pdh` (left) and for `ta1` (right).

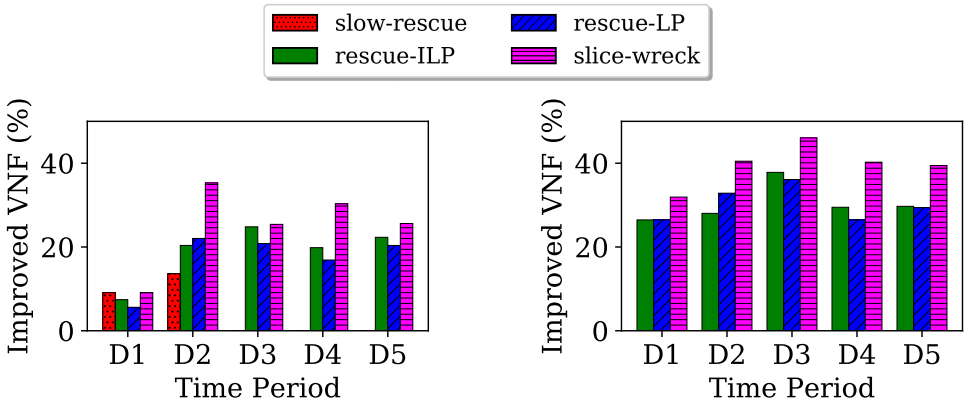


Figure 4.4 – Gains in VNF cost for `pdh` (left) and for `ta1` (right).

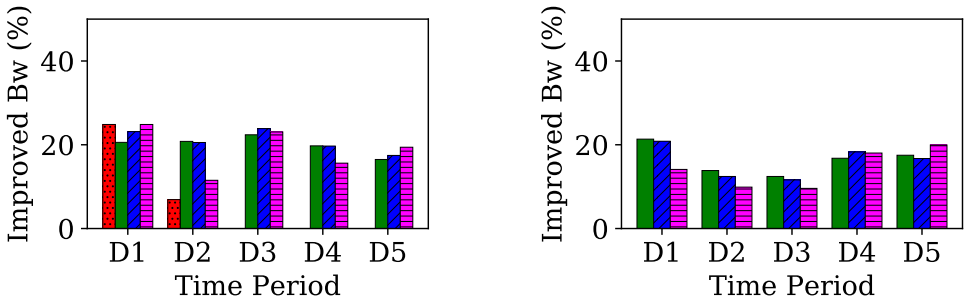


Figure 4.5 – Gains in bandwidth cost for `pdh` (left) and for `ta1` (right).

For `pdh` and for traffic matrix D2, `slow-rescue` reached the time limit, but succeeds in finding a feasible solution, whose improvement in terms of network cost is only half of the improvement of the Column Generation based methods. For the other traffic periods (except for the smallest one D1), not even a feasible solution can be found during the time limit.

	pdh		ta1	
	rescue-ILP	rescue-LP	rescue-ILP	rescue-LP
D1	3.11	4.03	1.82	1.38
D2	19.14	17.15	10.67	6.64
D3	11.22	13.88	8.19	9.55
D4	15.30	17.87	12.39	15.60
D5	12.52	13.28	12.16	13.09

Table 4.3 – Accuracy of the column generation models (%)

For both networks, we see that `rescue-ILP` and `rescue-LP` achieve comparable results. As `rescue-LP` is faster, we use it as our *preferred solution* in the following.

Last, we compare the results of our models with `slice-wreck`, which does not use the make-before-break mechanism. `slice-wreck` can achieve a better network improvement but at the cost of breaking slices and, thus, of a degraded QoS for users. We report its results as an upper bound on what our algorithms can achieve. We see that `rescue-ILP` and `rescue-LP` results are within few percent of the ones of `slice-wreck`, showing their efficiency. The difference is higher for heavy load periods (D4 and D5). Indeed, when the traffic is high, some links are almost saturated. It thus is harder to ensure that the bandwidth for both the current path and the one targeted by the reconfiguration can be reserved during the process.

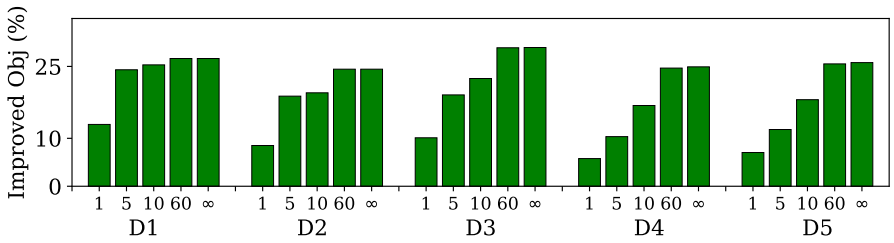
Figure 4.4 and Figure 4.5 show how the improvement of objective is decomposed between the number of VNFs and the bandwidth usage. We considered a setting (and accordingly set the value of β in our objective function, Equation 4.1) in which the bandwidth and the VNFs have the same weight in the objective: using 100% of the available bandwidth has the same cost as using 100% of the available VNFs.

We see that reconfiguration allows to decrease the usage of *both* network bandwidth and VNFs. In terms of network bandwidth usage, the gains are similar between `pdh` and `ta1` and vary between 12% and 24%. For the deployment of VNFs the gain on `pdh` is lower and is between 6% and 25% while for `ta1` it varies between 23.5% and 38%.

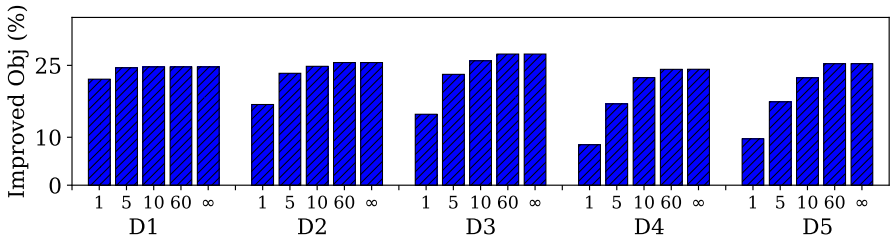
Indeed, `pdh` is a smaller network with a smaller diameter compared to `ta1` and fewer available datacenters. The routes of new slices are therefore more likely to be close to an already deployed VNF and of length not too far from the shortest one. Therefore, the reconfiguration is not as efficient on `pdh` compared to `ta1`.

4.6.2.3 Accuracy of the Column Generation Models

The accuracy ε of a column generation model is classically defined as $\varepsilon = (\tilde{z}_{\text{ILP}} - z^{\star}_{\text{LP}})/z^{\star}_{\text{LP}}$, where $z^{\star}_{\text{LP}}$ represents the optimal value of the relaxation of the Restricted Master Problem, and $\tilde{z}_{\text{ILP}}$ the integer solution obtained at the end of the column generation algorithm. We provide the accuracy of `rescue-ILP` and `rescue-LP` in Table 4.3. We see that, if the accuracy increases with the number of slices, it is always lower than 20% for both networks. The solutions thus are not far from optimal.



(a) `rescue-ILP`



(b) `rescue-LP`

Figure 4.6 – Improvement due to the reconfiguration for different model time limits on `ta1`.

4.6.2.4 Time limits for the reconfiguration

The reconfiguration of the network has to be done dynamically in real time. In this context, the time to compute the reconfiguration is an important element towards the adoption of such solutions. We thus compare the results of the algorithms for `ta1` for different maximum execution times: 1, 5, 10, 60 seconds and without limits, see Figure 4.6 (with `rescue-ILP` at the top and `rescue-LP` at the bottom). In period D1, `rescue-LP` is almost optimal in 1 s. We need at least 10 s to get closer to the optimal (no time limit) in the other periods, at 3% at most in D5. As for `rescue-ILP`, it is almost optimal in D1 in 5 s but needs at least 60 s to reach near optimal results for the other periods.

It confirms that `rescue-LP` is the most scalable method while reaching similar performance as `rescue-ILP`. It thus is the best solution to use in practice: `rescue-LP` is fast and reaches a very good performance level in only 10 s for all the periods.

4.6.3 Impact of the number of reconfiguration steps

A specificity of our *make-before-break* scheme is that the reconfiguration is done in a given number of steps. The more steps the more possibilities to improve the network operating state, however the more complex the models and the longer to solve them. In this section, we are interested in the impact of the number of steps on the improvements achieved by the reconfiguration and on the execution time. We use the same scenario as in the previous section. The simulations are done on `ta1` for a number of reconfiguration steps varying from 1 to 4. Results are reported in Figure 4.7 and Figure 4.8. As a measure of comparison, we reported the results of `slice-wreck` which are the same in all cases, as the method does not have reconfiguration steps.
As can be seen in Figure 4.7 and Table 4.4, whether on `rescue-LP` or `rescue-ILP`, over all periods: an increase in numbers implies an improvement in the objective. This phenomenon is

	rescue-ILP				rescue-LP			
	1 step	2 steps	3 steps	4 steps	1 step	2 steps	3 steps	4 steps
D1	21.0	26.1	26.1	25.7	20.7	25.4	25.9	26.7
D2	18.6	24.4	23.9	26.1	18.6	24.6	26.6	26.4
D3	16.6	27.0	28.8	28.0	17.0	26.5	27.9	27.4
D4	9.1	19.5	24.7	25.8	9.1	18.9	23.6	25.9
D5	6.4	19.6	25.2	27.5	6.8	19.0	24.7	26.9
AVG	**14.4**	**23.3**	**25.8**	**26.6**	**14.4**	**22.9**	**25.8**	**26.7**

Table 4.4 – Average percentages of improvement for each period and each number of steps for `rescue-LP` and `rescue-ILP` on `ta1`.

	rescue-ILP				rescue-LP			
	1 step	2 steps	3 steps	4 steps	1 step	2 steps	3 steps	4 steps
D1	19.5	25.1	23.9	39.0	2.8	3.2	4.2	4.7
D2	24.0	40.1	48.1	58.8	6.6	11.6	15.5	15.5
D3	35.2	50.2	65.4	83.8	15.4	24.3	32.7	45.3
D4	44.3	73.4	107.9	112.9	23.6	40.4	53.8	62.7
D5	59.3	85.1	120.7	151.8	20.1	45.2	68.0	83.7
AVG	**36.5**	**54.8**	**73.2**	**89.3**	**13.7**	**24.9**	**34.8**	**42.4**

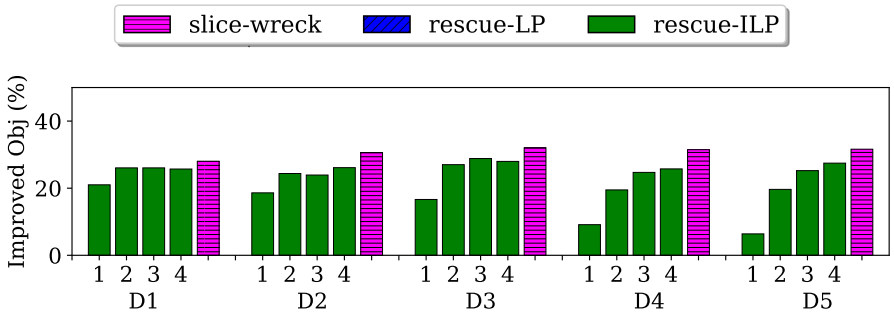
Table 4.5 – Computation times (seconds) on `ta1`

even more noticeable in periods D4 and D5. Nevertheless we can see a strong improvement between 1 step and 2 steps, a weaker improvement between 2 and 3 steps and finally a negligible improvement between 3 and 4 steps. In order to compare the interest of different numbers of reconfiguration steps, we must also look at the execution times. Figure 4.8 and Table 4.5 shows that, like the objective, an increase in the number of reconfiguration steps implies a higher computing time. But unlike the objective, the increase in computing time is not reduced as much by increasing the number of steps. By averaging over all time periods and between `rescue-LP` and `rescue-ILP`:

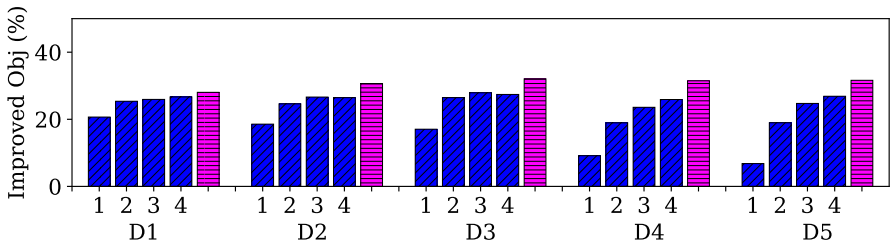
Going from 1 to 2 steps, the balance is undeniable, we increases the objective improvement by 60% against 59% additional execution time. Passing from 2 to 3 steps increases the objective improvement by 11.7% for 35.3% more computing time. Finally, moving from 3 to 4 steps we increases the objective improvement by only 3.1% for 21.9% more computing time. Seeing this we decided to use a 2-step reconfiguration for `pdh` (mainly so that we could compare our algorithms to `slow-rescue`) and a 3-steps reconfiguration for all the other experiments because it seems to us to be the most balanced configuration.

4.6.4 Gains over Time

We now study the gains provided by the reconfiguration over time. To this end, we consider a scenario in which the traffic is dynamic (requests arrive and leave over time) and some reconfigurations are regularly performed. We use a traffic distribution from a trace of Orange network (Figure 4.1) in order to model the variation of traffic over 24 hours. In our scenario, the network

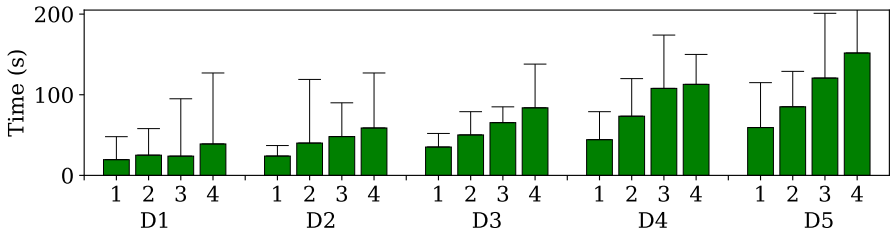


(a) `rescue-ILP`

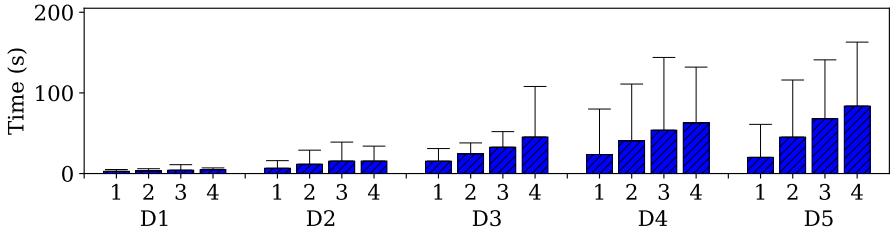


(b) `rescue-LP`

Figure 4.7 – Improvement of the Objective (in %) with different numbers of reconfiguration steps on `ta1`



(a) `rescue-ILP`



(b) `rescue-LP`

Figure 4.8 – Reconfiguration time with different numbers of reconfiguration steps on `ta1`

experiences periods of high congestion during which some slices may be rejected and periods with lower traffic.

To assess the reconfiguration gains, we compare `rescue-LP` (our best algorithm as it is as efficient as `rescue-ILP` but much faster) with
noreconfChapFive which does not carry out reconfigurations for a medium (`ta1`) and a large (`ta2`) networks. We study the following metrics: the network operational cost, the throughput of the accepted slices, the accepted number of slices, and the operational cost per Mbits of accepted traffic.

`rescue-LP` performs reconfigurations every 15 minutes. We choose this value as it seems a reasonable one for a network operator which does not want to change its routes too frequently. This choice is discussed in subsection 4.6.5, in which we vary the reconfiguration frequency, and show that 15 is a good trade-off between network management and all the studied metrics.

4.6.4.1 Network Cost

In Figure 4.9 we study the network operational cost over time. Recall that the network costs are defined by the weighted sum of link bandwidth and VNF usage costs. The network cost follows the traffic variation depicted in Figure 4.1. Of course, the figures shows that the more traffic, the more network operational cost. Our solution is more reactive to traffic variations thanks to the reconfigurations that are regularly performed. Throughout the entire execution and for both networks, `rescue-LP` significantly reduces the network operational costs: 22% of reduction on `ta1` and 18% on `ta2` compared to `no-reconf` case. This reduction is particularly substantial when the network is loaded (between 10am and 6pm). Reconfiguration allows a better management of the network and a more efficient resource usage.

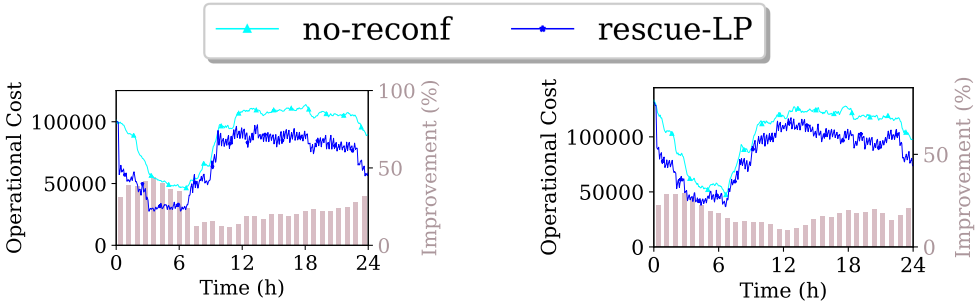


Figure 4.9 – Network cost for `ta1` (left) and for `ta2` (right).

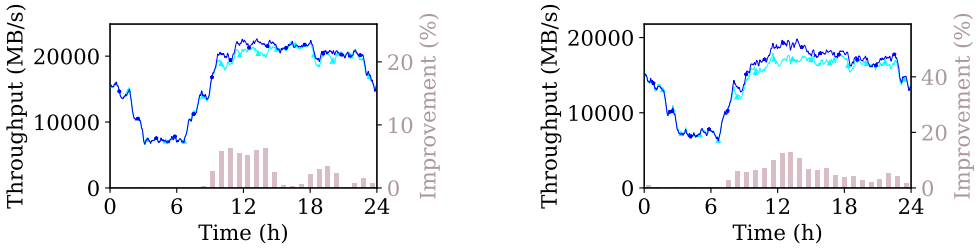


Figure 4.10 – Throughput for `ta1` (left) and for `ta2` (right).

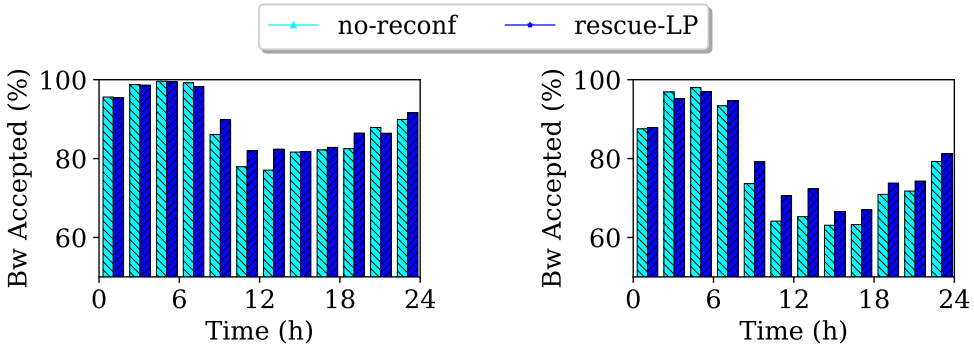


Figure 4.11 – Percentage of Bandwidth accepted for `ta1` (left) and for `ta2` (right).

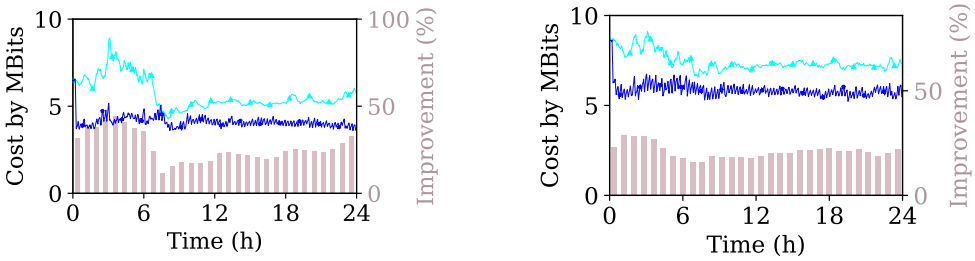


Figure 4.12 – Network cost per accepted bandwidth for `ta1` (left) and for `ta2` (right).

4.6.4.2 Throughput

The objective of our solution is to reduce operational costs. However, we should not reduce these costs at the price of rejecting slices. Therefore, we present the global throughput of the network in Figure 4.10. This throughput is defined as the sum of the requested bandwidth of the accepted slices. During the first 5 hours of execution there is almost no congestion because the traffic decreases, thus,
noreconfChapFive and `rescue-LP` accept the same number of slices and get roughly the same throughput for both networks. The next 3 hours, traffic increases and `rescue-LP` improves the throughput by up to 13% for `ta2` when the network is the most saturated (traffic period D5). For a period of 24 hours, `rescue-LP` allows an average throughput improvement of 3% on `ta1` and 5% on `ta2`. Therefore, as a combined conclusion of Figure 4.9 and Figure 4.10, `rescue-LP` succeeds in reducing the network operational costs while, at the same time, improving the network throughput. These gains are reached without impacting users' Quality of Service as resources are reserved before any changes of network configurations thanks to our make-before-break mechanism.

4.6.4.3 Accepted Slices

The difference in terms of throughput discussed above comes from different slice acceptance rates of both solutions. As the slices of different types do not require the same reserved bandwidth (see Table 4.2), we report the percentage of the bandwidth of the accepted slices compared to the one of the requested slices. The Figure 4.11 represents incremental acceptance, each bar corresponds to the percentage of accepted bandwidth averaged over 2 hours. The evolution of the curve reflects

the inverse of the network load as shown in Figure 4.1. Between midnight and 5:00 a.m. the network load decreases from period D3 to D2 and then to D1, we can therefore see that we are able to accept almost all of the demands. Then the load rises until noon to reach period D5 and remains stable until about 7 p.m., thus, the percentage of demands acceptance declines, which is even more noticeable on `ta2`. Finally, the load decreases until midnight to reach period D3, implying an increase in the acceptance percentage. `rescue-LP` allows an improvement in slice acceptance for both networks: 2% and 4% more bandwidth for `ta1` and `ta2`, respectively.

4.6.4.4 Cost per MBit

As discussed above, reconfiguration allows to reduce the network operational cost and, at the same time, to accept more slices. To measure both advantage with a single metric, we report the cost per MBit to obtain a fair comparison in Figure 4.12. The improvement in percent is given by the light red bars. The gain is of 25% for `ta1` and 22% for `ta2`. This shows that our solution is significantly efficient. We observe that the gain is lower when the traffic is low (period D1), but similar for the other periods (D2, D3, D4, D5). We also see that reconfiguring the network keeps the cost per MBit more stable during time, showing a better usage of the network resources which adapt when the traffic varies.

4.6.5 Impact of the reconfiguration time interval

In the previous section, we measured the effects of regularly reconfiguring the network in a dynamic scenario. The reconfiguration interval was set to 15 minutes. We now study the effects of different reconfiguration frequencies: 5 , 15 , 30 , and $60\,\mathrm{min}$. Indeed, reconfiguring more regularly can improve the usage of the network resources, but at the same time lead to more difficult management. Reconfiguring less regularly eases management, but reduces the reconfiguration gains. Results are reported in Table 4.6

4.6.5.1 Network Cost

We study in Figure 4.13 the network operational cost of the network considering different reconfiguration frequencies. For frequency of 60 , 30 , 15 and 5 respectively, we have improvements of 15.5%, 18.2%, 22% and 23.9% on `ta1` and 9.4%, 14%, 18% and 21% on `ta2`. Even if a frequency of 5 leads to better improvement in network costs, good improvement is already obtained with a reconfiguration frequency of 60 , meaning a reconfiguration every hour.

4.6.5.2 Throughput

Figure 4.14 shows the network throughput over time as defined in subsubsection 4.6.4.2. For reconfiguration frequency of 60 , 30 , 15 and 5 respectively, there are improvements of 0%, 1%, 3.1% and 5.1% on `ta1` and 0.1%, 2.4%, 5% and 7% on `ta2`. For both networks, a reconfiguration frequency every 15 minutes seems to be a good trade-off between throughput and network management.

4.6.5.3 Accepted Slices

In Figure 4.15, we plot the accepted bandwidth over time as defined in subsubsection 4.6.4.3. Each curve is more easily identifiable compared to previous figures. For reconfiguration frequency of

	ta1				ta2			
Time Interval	60	30	15	5	60	30	15	5
Network Cost	15.5	18.2	22	23.9	9.4	14	18	21
Throughput	0	1	3.1	5.1	0.1	2.4	5	7
Accepted Slices	0	0.7	2.2	4	0	1.5	3.8	5.3
Cost per MBit	14.4	20.5	25	28.5	10.2	16.2	22	26.8

Table 4.6 – Summary of the improvement percentage from `rescue-LP` with a 3-steps reconfiguration according to the reconfiguration time interval (number of minutes between every reconfiguration).

60 , 30 , 15 and 5 respectively we have improvements of 0%, 0.7%, 2.2% and 4% on `ta1` and 0%, 1.5%, 3.8% and 5.3% on `ta2`. Here again, reconfiguring every 15 minutes seems to be a good trade-off for the accepted number of slices.

4.6.5.4 Cost per MBit

Figure 4.16 shows the network operational cost per MBit over time as defined in subsubsection 4.6.4.4. We can easily distinguish the above curve without reconfiguration among all the curves. For reconfiguration frequency of 60 , 30 , 15 and 5 respectively there are improvements of 14.4%, 20.5%, 25% and 28.5% on `ta1` and 10.2%, 16.2%, 22% and 26.8% on `ta2`. Reconfiguring once an hour leads to strong peaks of cost, while when we reconfigure every 5 minutes, the cost per Mbit is more stable.

To summarize, a reconfiguration time interval of 15 is a good trade-off to balance the cost, stability and ease of network management. It leads to an improvement of 20.7% (respectively 17%) of network cost, 3.5% (respectively 8.9%) of throughput, 2.4% (respectively 6.4%) of accepted bandwidth, and of 25.5% (respectively 23.2%) of cost per Mbit on `ta1` (respectively on `ta2`).

4.6.6 Scalability

In this section we study the scalability potential of our approach. Indeed the interest of column generation is to be able to use reconfiguration with many requests. We must recall that a slice is composed of an average of three SFCs requests and therefore 480 slices represent about 1440 requests. In Figure 4.17, we are interested in the scalability of our solution based on our experiences in subsection 4.6.2. We want to show that our solution can manage a large number of slices in few seconds only. We vary the number of slices from 60 to 480, as well as the capacity of the network to keep the same percentage of network load. We impose a maximum time of 60 seconds. Note that only `rescue-ILP` and `rescue-LP` are compared, and recall that `slow-rescue` did not find any feasible solution with 2 steps of reconfiguration, with less than 30 slices in 3600 seconds on `ta1` (Figure 4.6 (right)). For each of the networks `ta1` and `ta2` we perform a 3-steps reconfiguration. As we can see, even with a large number of slices and a limited time, our solution still allows a significant improvement of the objective. The left side of Figure 4.17 shows us the results on `ta1` where `rescue-ILP` gets an improvement of 27.1% with 120 slices and on average it improves by 19%, while `rescue-LP` improves at best by 29.5% with 120 slices with an average of improvement of 22.6%. The right side of Figure 4.17 shows the results on `ta2` where `rescue-ILP` improves at best by 22.6% with 120 slices and at worst by 12.2% with 480 slices

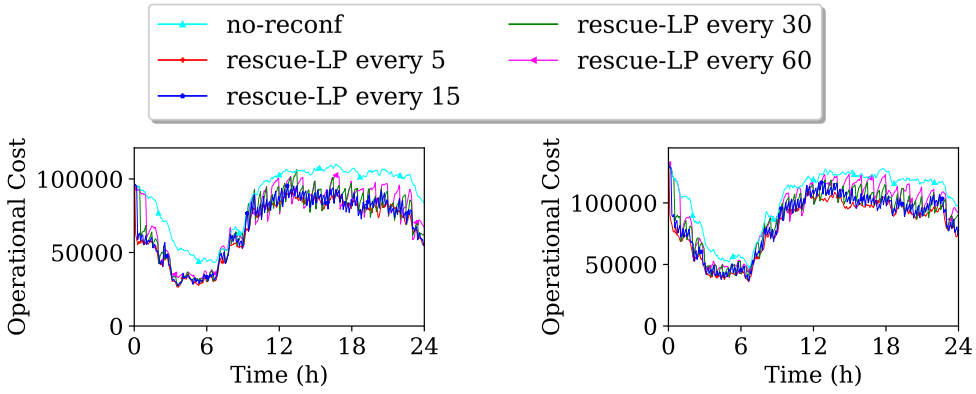


Figure 4.13 – Network cost for `ta1` (left) and for `ta2` (right).

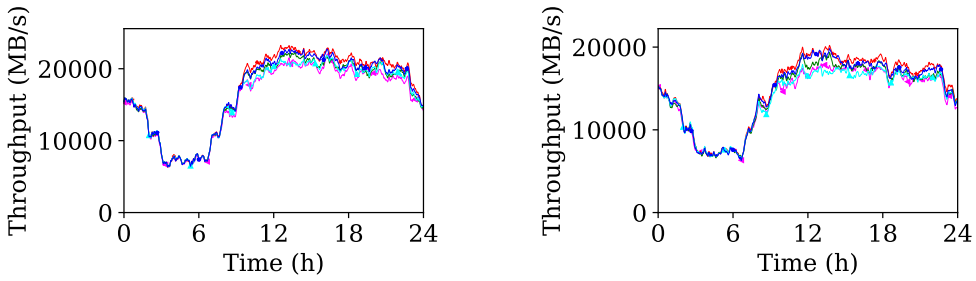


Figure 4.14 – Throughput for `ta1` (left) and for `ta2` (right).

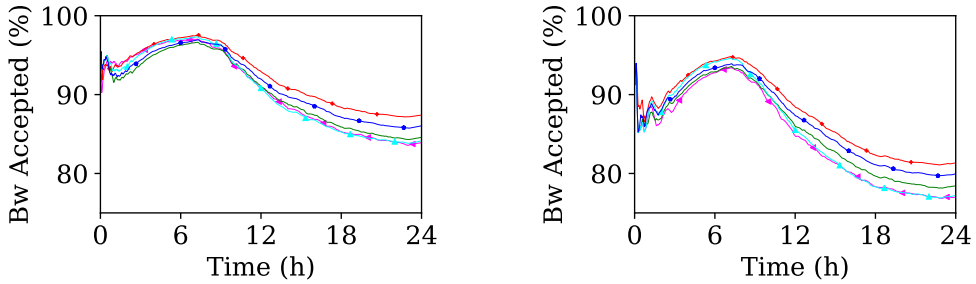


Figure 4.15 – Percentage of Bandwidth accepted for `ta1` (left) and for `ta2` (right).

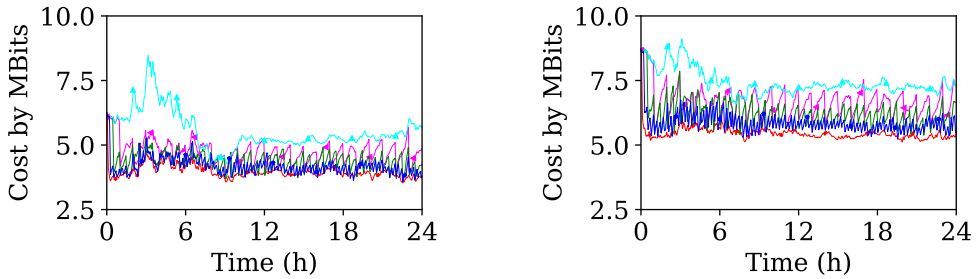


Figure 4.16 – Network cost per accepted bandwidth for `ta1` (left) and for `ta2` (right).

and on average it improves by 18.1%, while `rescue-LP` improves at best by 24% with 120 slices and at worst by 17.9% with 480 slices and on average it improves by 20%. Finally we can see here

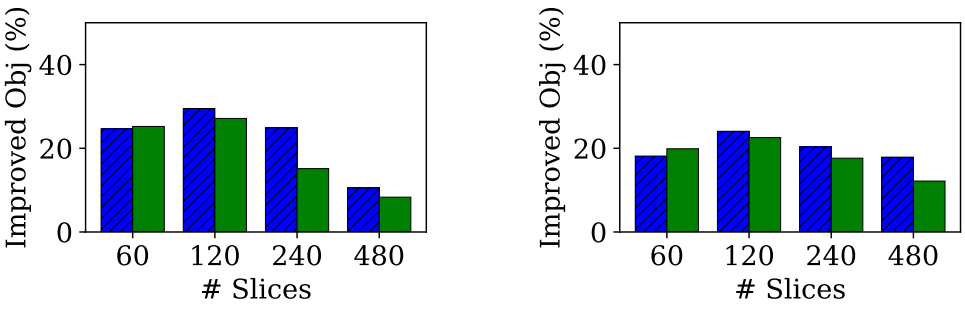


Figure 4.17 – Gains in network cost for `ta1` (left) and for `ta2` (right) with different numbers of slices during `D5` period.

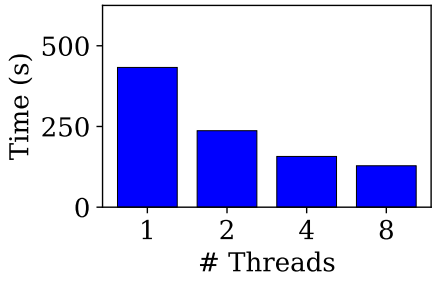


Figure 4.18 – Time to execute the pricing problems according to the number of threads on `ta2` in D5 period (60 slices).

the advantage of `rescue-LP` over `rescue-ILP` which allows a better improvement and is less affected by the lack of time on large instances.

4.6.7 Parallelisation of the pricing problem

One of the advantages of column generation is the ability to parallelise the execution of pricing problems on several CPUs cores or machines. In our experiments about 70% of the execution time is spent on solving pricing problems, which means that parallelisation can save time. In Figure 4.18 we show the execution times of `rescue-LP` to reconfigure 60 slices in D5 period in `ta2`. For this experiment we put no time limit and let the column generation create as many columns that can potentially improve the solution. The average computation time is 433 seconds with 1 thread and 237 seconds with 2 threads (45% improvement). With 4 threads `rescue-LP` is faster and computes a solution on 157 seconds. The difference between 4 and 8 threads is less pronounced, 29 seconds less, but our computer, although having 8 threads, has only 4 CPU cores. As pricing execution already uses CPUs to their full potential, additional threads have only a limited impact.

4.6.8 Impact of the delay constraints

Being able to ensure strict delay constraints for some applications is one of the key element of network slicing [BGB+17]. As an example, each of the slice we considered had a maximum latency

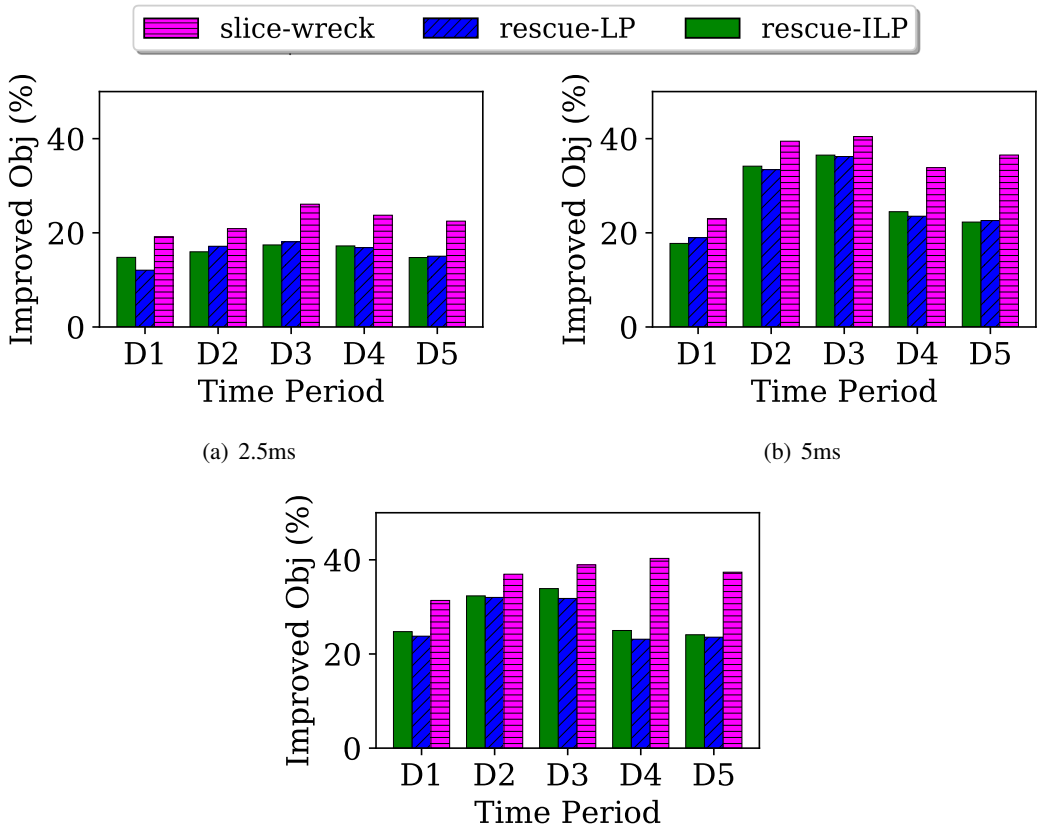
corresponding to its service as shown in Table 4.2. In this section, we study the impact of different delay constraints on the reconfiguration gains. We carried out three sets of reconfigurations for `ta1` setting the delay constraints of each slice successively to 3 different values: $2.5\,\mathrm{ms}$, $5\,\mathrm{ms}$ and $10\,\mathrm{ms}$.

4.6.8.1 Stricter delays lead to lower improvements

The improvement of the objective due to reconfiguring is plotted in Figure 4.19 for the 3 different latency constraints. We observe that larger gains are obtained when the delay constraints are looser. For a $2.5\,\mathrm{ms}$ latency, the improvement is of 16% in average, while it is of 27% and 27.5% for $5\,\mathrm{ms}$ and $10\,\mathrm{ms}$ latencies, respectively. Indeed, when the maximum delay is small, the number and diversity of potential paths to choose from for a demand are smaller. This leads to fewer opportunities for the reconfiguration. However, we also see that, when the maximum allowed delay reaches a threshold, such constraints are no more an important limiting factor. For example, for `ta1` the improvements for $5\,\mathrm{ms}$ and the $10\,\mathrm{ms}$ are similar.

4.6.8.2 Stricter delays makes it harder to solve

In Figure 4.20 we study the time taken to compute the reconfigurations: The stricter the latency constraints, the slower to compute a reconfiguration. With a very tight delay constraint of $2.5\,\mathrm{ms}$, in addition to have a lower improvement, we have much longer computation times with $428\,\mathrm{sec}$ on average, compared to $228\,\mathrm{sec}$ and $91\,\mathrm{sec}$ for $5\,\mathrm{ms}$ and $10\,\mathrm{ms}$, respectively, which allowed similar gains. Indeed, the higher the maximum allowed delay, the larger the opportunities for reconfiguration and the easier it is to find paths satisfying the delay constraints.



(a) 2.5ms

(b) 5ms

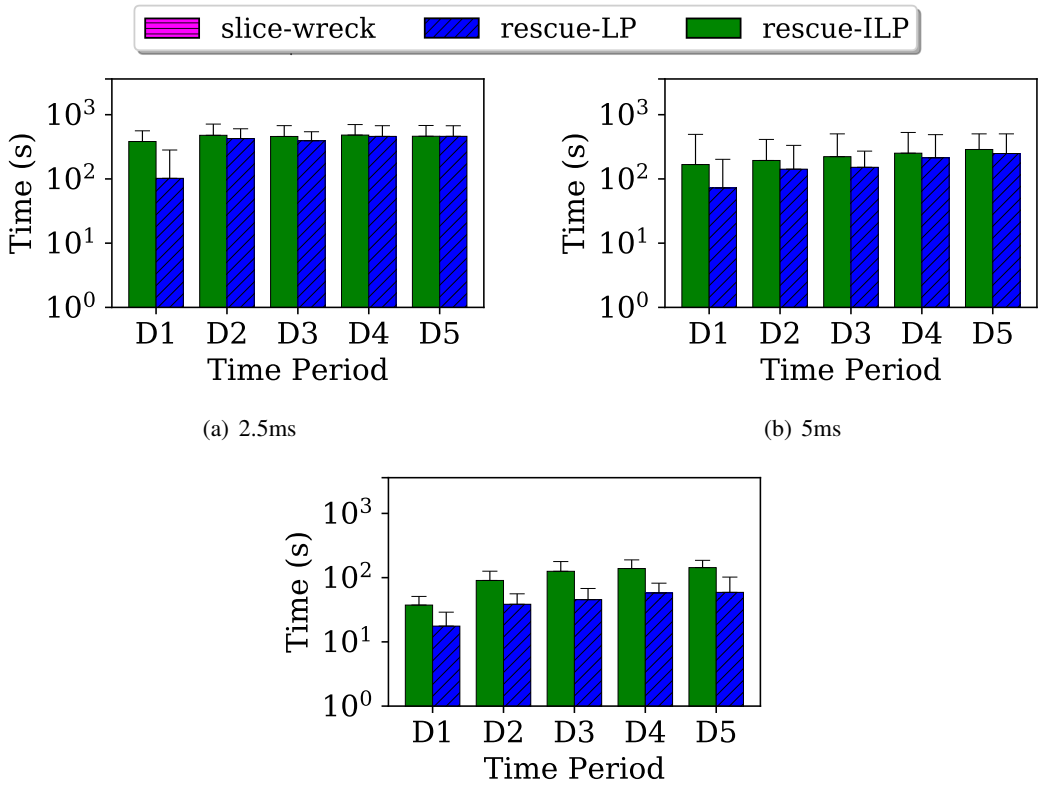
(c) 10ms

Figure 4.19 – Improved Objective with different delay constraints on `ta2`.

4.7 Conclusion

Modern 5G networks will see an increase in the number of users and an ever-growing need for flexibility and efficiency. Reconfiguring requests regularly can lead to significant improvements in the use of network resources. In this chapter, we provide solutions, `rescue-ILP` and `rescue-LP`, to reconfigure a set of requests using a make-before-break approach. Our algorithms, based on column generation, reroute the requests to an optimal or close to optimal solution without impacting the rerouted requests. Both our solutions are scalable and allow to reconfigure several hundred of Slices in one minute. The use of column generation also allows us to effectively parallelise part of the problem, which will increase its efficiency in the coming years with the development of computer with a larger number of CPU cores. `rescue-LP` is the solution to be chosen in practice as we observed during simulations that it scales better with the network size and the number of slices. Reconfiguring regularly the network with `rescue-LP` allows a slight increase in throughput when the network is congested as well as a significant reduction in operating costs of around 20% to 25%.

Tables summarising the percentage improvement of `rescue-ILP` and `rescue-LP` with different numbers of reconfiguration steps and their computation times can be found in Table 4.4



(a) 2.5ms

(b) 5ms

(c) 10ms

Figure 4.20 – Reconfiguration time with different delay constraints on `ta2`.

and Table 4.5. The summary of the different percentage improvement of `rescue-LP` with different reconfiguration intervals can be found in Table 4.6.

It would be interesting to extend our work to include the edge network infrastructure. The specificities in terms of latency and computing power of this part of the network would allow us to better adapt our solution and its objective to the different types of slices. The uRLLC slices could be given priority for reconfiguration at the edge due to their latency constraints, while the eMBB slices could be given priority for reconfiguration in the core in order to have access to its computing power.

References

[01118] ETSI GR NGP 011. Next generation protocols(ngp);e2e network slicing reference framework and information model. https://www.etsi.org/deliver/etsi_gr/NGP/001_099/011/01.01.01_60/gr_ngp011v010101p.pdf, 09 2018.

[BGB+17] D. Bega, M. Gramaglia, A. Banchs, V. Sciancalepore, K. Samdanis, and X. Costa-Perez. Optimising 5G infrastructure markets: The business of network slicing. In *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 1–9, 2017.

[CIS15] CISCO. *Cisco Visual Networking Index: Forecast and Methodology, 2014–2019*, May 2015.

[CLENR15] R. Cohen, L. Lewin-Eytan, J.S. Naor, and D. Raz. Near optimal placement of virtual network functions. In *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 1346–1354, Kowloon, Hong-Kong, 2015.

[GGJM20] A. Gausseran, F. Giroire, B. Jaumard, and J. Moulierac. Be scalable and rescue my slices during reconfiguration. In *IEEE ICC*, 2020.

[GGJM22] A. Gausseran, F. Giroire, B. Jaumard, and J. Moulierac. Be scalable and rescue my slices during reconfiguration. volume 65, 2022.

[GR18] Lingnan Gao and George N Rouskas. Virtual network reconfiguration with load balancing and migration cost considerations. In *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 2303–2311. IEEE, 2018.

[GZL20] W. Guan, H. Zhang, and V. C. M. Leung. Slice reconfiguration based on demand prediction with dueling deep reinforcement learning. In *IEEE GLOBECOM*, 2020.

[HJG18] Nicolas Huin, Brigitte Jaumard, and Frédéric Giroire. Optimal network service chain provisioning. *IEEE/ACM Transactions on Networking (ToN)*, 26(3):1320–1333, June 2018.

[LLZ+17] Junjie Liu, Wei Lu, Fen Zhou, Ping Lu, and Zuqing Zhu. On dynamic service function chain deployment and readjustment. *IEEE Transactions on Network and Service Management (IEEE TNSM)*, 14(3):543–553, 2017.

[LPMK18] Mathieu Leconte, Georgios S Paschos, Panayotis Mertikopoulos, and Ulaş C Kozat. A resource allocation framework for network slicing. In *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 2177–2185. IEEE, 2018.

[OWPT10] Sebastian Orlowski, Roland Wessäly, Michal Pióro, and Artur Tomaszewski. Sndlib 1.0—survivable network design library. *Networks: An International Journal*, 55(3):276–286, 2010.

[PPR+19] M. Pozza, A. Patel, A. Rao, H. Flinck, and S. Tarkoma. Composing 5G network slices by co-locating VNFs in μslices. In *IFIP Networking Conference*, pages 1–9, May 2019.

[R+17] Peter Rost et al. Network slicing to enable scalability and flexibility in 5G mobile networks. *IEEE Communications magazine*, 55(5):72–79, 2017.

[STV15] Marco Savi, Massimo Tornatore, and Giacomo Verticale. Impact of processing costs on service chain placement in network functions virtualization. In *IEEE Conference NFV-SDN*, 2015.

[SZGS+18] Josep Xavier Salvat, Lanfranco Zanzi, Andres Garcia-Saavedra, Vincenzo Sciancalepore, and Xavier Costa-Perez. Overbooking network slices through yield-driven end-to-end orchestration. In *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '18, New York, NY, USA, 2018. Association for Computing Machinery.

[TAM19] S. Troia, R. Alvizu, and G. Maier. Reinforcement learning for service function chain reconfiguration in nfv-sdn metro-core optical networks. *IEEE Access*, 2019.

[WFS+20] F. Wei, G. Feng, Y. Sun, Y. Wang, S. Qin, and Y. C. Liang. Network slice reconfiguration by exploiting deep reinforcement learning with large action space. *IEEE Transactions on Network and Service Management*, 2020.

[WM13] R. Wang and B. Mukherjee. Provisioning in elastic optical networks with non-disruptive defragmentation. *IEEE Journal of Lightwave Technology*, 31(15):2491–2500, 2013.

[ZLF+17] Nan Zhang, Ya-Feng Liu, Hamid Farmanbar, Tsung-Hui Chang, Mingyi Hong, and Zhi-Quan Luo. Network slicing for service-oriented networks under resource constraints. *IEEE journal on Selected Areas in Communications*, 35(11):2512–2521, 2017.

CHAPTER 5

Reinforcement Learning Driven Reconfiguration

The emerging 5G induces a great diversity of use cases, a multiplication of the number of connections, an increase in throughput as well as stronger constraints in terms of quality of service such as low latency and isolation of requests. To support these new constraints, Network Function Virtualisation (NFV) and Software Defined Network (SDN) technologies have been coupled to introduce the network slicing paradigm. Due to the high dynamicity of the demands, it is crucial to regularly reconfigure the network slices in order to maintain an efficient provisioning of the network. A major concern is to find the best frequency to carry out these reconfigurations, as there is a trade-off between a reduced network congestion and the additional costs induced by the reconfiguration. In this chapter, we tackle the problem of deciding the best moment to reconfigure by taking into account this trade-off. By coupling Deep Reinforcement Learning (DRL) for decision and a Column Generation (CG) algorithm to compute the reconfiguration, we propose `Deep-REC` and show that choosing the best time during the day to reconfigure allows to maximise the profit of the network operator, minimise the use of network resources and the congestion of the network. Moreover, our proposition allows to decrease the number of needed reconfigurations compared to an algorithm doing regular reconfigurations during the day.

Part of this chapter is submitted in [GAL+22].

5.1 Introduction

The increasing importance of wireless networks and the emergence of 5G bring out new needs such as massive device connectivity, high mobility and a great diversity in the QoS requirements. By dividing the network infrastructure into multiple logical isolated networks, network slicing allows the support of a wide range of communication scenarios with a diversified set of service demands, requirements, and performance. A slice must be deployed in real time and, needs to fulfill an end-to-end service, thus, the corresponding provisioning of network, computing, and storage resources has to be done dynamically. A network slice can be modeled as SFC containing the necessary VNFs provided by the network, allowing for a fast management of the flows/demands [ZLF+17]. See subsection 0.2.3 and subsubsection 0.2.3.1 for more details on 5G networks and network slicing.

In 5G networks, traffic is considered to be highly dynamic and network requests may be subject to frequent changes such as arrivals and departures. After a while this dynamicity may fragment the slice resource usage and make the use of network resources less efficient. As we have seen in chapter 3 and chapter 4, thanks to SDN and NFV, this effect can be counter using reconfiguration. The slice allocation can thus be adjusted in order to reduce the resource utilization with the goal of minimising operational costs. In this chapter, we use the reconfiguration method presented in chapter 4 and based on the *make-before-break* mechanism. This allows to not disrupt the traffic and, therefore, does not impact the Quality of Service (QoS) of the slices. The computations is done using a column generation approach allowing to deal with a large set of network slices. See subsection 0.2.4 and subsubsection 0.2.4.2 for more details on reconfiguration and the *make-before-break* technique.

Even when using such a mechanism to avoid degrading the QoS, network operators do not want to reconfigure their network too frequently, as it may lead to additional management costs. On the opposite, reconfiguring too rarely during the day may lead to a sub-optimal network usage. A simple policy with low computational cost is to regularly reconfigure every x minutes. However, reconfiguring in response to variation of traffic can reduce the number of reconfigurations required each day without impacting the overall improvement obtained. In this chapter, we propose a *reconfiguration management agent* that chooses when to initiate reconfiguration as a function of different parameters such as the traffic dynamics and the level of network congestion. We use a *Deep Reinforcement Learning technique* by implementing it with Tensorflow [A+15] and their DQN agent. We then show that our agent improves the efficiency of reconfigurations by performing less reconfigurations while still minimising the network operational costs compared to doing periodic and frequent reconfigurations.

Motivation In a dynamic scenario, due to the frequent arrival and departure of slices, the network is regularly in a sub-optimal state. We have therefore previously worked on a reconfiguration algorithm which, if called regularly, minimises the operational costs of the network. Nevertheless, the frequency to run this algorithm was found on a empirical manner. Indeed, we deduced in chapter 4 after several trials that reconfiguring every 15 minutes allowed a good ratio between cost reduction, quality of reconfigurations, acceptance rate and computation time. A fixed frequency is easy to set up but in practice, at some specific time of a day, reconfiguration may not be needed as traffic remains stable and the network is already in an optimal state. A new reconfiguration at this time won't bring any gain. On the opposite, during high-dynamic traffic period,

more frequent reconfigurations may be suitable to maintain an acceptable state of the network with efficient network resource usage. Therefore, a network operator might be interested to adapt the reconfiguration frequencies depending on the congestion of the network, and the nature of the traffic. This is the main goal of this chapter. We present a deep reinforcement learning model named `Deep-REC` to choose when to reconfigure in order to optimally adapt to the evolving network state. Our objective is to maximise the cumulative profit as presented before: the sum of the instantaneous profits p_t (See (5.1)).

The rest of this chapter is organized as follows. In section 5.2, we discuss related work. section 5.3 presents the formal definition of our problem, and section 5.4 the optimisation models based on reinforcement learning for solving our problem. In section 5.6, we validate our proposed optimisation models by various numerical results. Finally, we draw our conclusion in section 5.7.

5.2 Related Work

Routing and provisioning of slices. The dynamic nature of network traffic raises a range of problems concerning the acceptance of incoming slices, resource management, and SLA compliance. Cheng *et al.* [CWM+20] use a two-phase method to deploy and manage slice provisioning using deep learning and Lyapunov stability theories. A first deployment phase allows to choose which slices to accept, where to allocate VNFs, and how to route the flows of slices. A second management phase adapts the flow of the deployed slices to match the current traffic. Their solution learns from past traffic and adapt to current traffic. In [HFS+19] the authors present a Mixed Integer Linear Program and a heuristic to add new slices by minimising the bandwidth consumption and the slice provisioning cost while taking into account the VNF migrations.

Reconfiguration using standard techniques. The reconfiguration of SFCs and/or slices aims to maintain a near-optimal state of the network over time in order to optimise the network usage and the acceptance of demands. In [PNL+20] the authors propose a slice reconfiguration technique in which the new state of the network is pre-computed. The reconfiguration is done in several steps in which the VNFs and routes are modified while taking into account capacities and delays. In [GTGM19a, GTGM21] (chapter 3), we proposed an integer linear program and an heuristic to efficiently reconfigure SFCs using a *make-before-break* strategy.

5.2.1 Predict and Learn

Many works use prediction to help decision making. In [SZGS+18] the authors use machine learning to predict future traffic, they compute the k-shortest paths in offlines and then in 2 phases in online, reserve the resources and choose the paths to maximise the revenues of an operator by overbooking. Zhou *et al.* [ZZC20] use an algorithm to predict the traffic requested by each slice, which assists an adaptive VNF scaling strategy to determine the number of VNFs and network resources. With the goal of deploying slices with the lowest power consumption costs. In [SW19] the authors predict the sources and destinations of future SFCs in an optical network using a machine learning classification method. Their goal is to maximise the prediction ratio of SFCs.

Machine learning and more precisely reinforcement learning is taking an increasingly important place in the resolution of network problems. In [LHM20] deep reinforcement learning is used coupled with a decomposition technique to learn the best policy for resource allocation for each user in order to maximise slices utility in a heterogeneous network where SFCs coexist with flows

whose traffic does not use VNFs. Their solution is divided into two parts, one that manages the resource orchestration policy for all agents to ensure SLA compliance. And the agents that learn the resource demands of the slices and manage the resources accordingly to optimise their performance while respecting network capacities. In [PHY20] the authors use a Q-learning algorithm to find and optimal SFC deployment path in edge computing environment, minimising SFCs length and latency.

Learning-based reconfiguration Some recent works use reconfiguration techniques based on reinforcement learning and try to predict the dynamicity of the network. Liu *et al.* [LFC+20] propose a VNF migration strategy based on Double-Deep Q-Network. Their goal is to equally place VNFs between Mobile Edge clusters and core clouds in order to avoid congestion at the Edge. The migration takes into account future traffic and tries to reduce the number of migrations while minimising the number of congested links. In [WFS+20] the authors use deep reinforcement learning to predict when to reconfigure in order to minimise the resources consumed. Unlike our work, the authors focus on intra-slice reconfigurations. Guan *et al.* [GZL20] use a Markov Renewal Process to predict changes in the resource occupancy of slices and reserve resources for slices that obtain higher revenues at lower cost. They use deep dueling neural network combined with Q-Learning to choose for each slice whether to reconfigure it or not. Their goal is to maximise long term revenue: the increased user utility minus a cost for the resource utilization and the service interruptions.

Similar to the last work mentioned, we establish an agent based on deep reinforcement learning to choose when to reconfigure the slices. To the best of our knowledge, we are the first to propose a methodology to place the network slices without booking capacity in advance. In contrast, our deep reconfiguration learning algorithm adapts its behaviour based on the variations of the network traffic. Moreover, we do not fix a limit of the reconfigurations per slice or per day. The agent determines the best time to reconfigure and performs the needed number of reconfigurations depending on the traffic variations. The reconfiguration is computed using a column generation algorithm based on a *make-before-break* reconfiguration that chooses which slices to reconfigure. This allows our method to deal with a large number of slices, taking only a few seconds to compute the reconfiguration.

5.3 System Model and Problem Formulation

We consider the network as a directed capacitated graph $G = (V, L)$ where V represents the node set and L the link set. Using the resources available in this network, we must allocate a set of slices requests D. These slices constitute network service chains supported by the set of functions F deployed at the network. A slice request $d \in D$ is modeled with a quadruplet: (i) the source v_{SRC}, (ii) the destination v_{DST}, (iii) the required bandwidth BW_d in traffic units, and, (iv) the ordered sequence of network functions c_d that need to be performed, where $f_i^{c_d}$ is the $i-th$ function of chain c_d. Each network function instance* $f \in F$ has a installation cost c_f accounting for all the VNF usage costs (licenses, energy consumption, etc). Each slice $d \in D$ provides a revenue u per bandwidth unit.

We aim to find an allocation of the slice requests such that the network operator profit accumulated over a certain time window is maximised. This cumulative profit is the sum of the instantaneous profits p_t at each observation time (i.e. minute). The profits p_t are computed as the

*A function instance is a CPU process running at a given data center.

difference between the overall revenue of the allocated slices and the overall cost of the deployed VNFs at time t:

$$p_t = \sum_{d \in D_t} u \cdot \mathrm{BW}_d - \sum_{f \in F_t} c_f \tag{5.1}$$

where D_t and F_t is the set of slices and function instances allocated at observation time t, respectively.

In a dynamic scenario with no information on future traffic, the impact of recently arrived requests onto the cumulative profit (at the operator time horizon) is still not known: slices routed using long paths will consume too many resources preventing the allocation of future requests. To take into account that, at each time, we target to place new requests on paths such that resources (i.e. bandwidth at each link and network functions at each node) are minimised. The exact method is commented in subsection 5.3.1.

As a consequence of this lack of information about the future, the trivial mechanics of requests coming and leaving over time will bring the network in a global sub-optimal state, since optimal allocations previously computed are not optimal anymore. Hence, we are forced to periodically reconfigure the network. To do that, we use the *make-before-break* mechanism [GGJM20] that avoids network service disruption due to traffic rerouting. See Figure 0.2.10 for an example.

5.3.1 Optimisation Model

To model the reconfiguration we use the column generation model that we previously presented in section 4.5 [GGJM20, GGJM22]. The objective of the reconfiguration is to minimise the network operational costs.

Column generation is an optimisation method based on a decomposition model that consists of a master problem combining solutions of pricing sub-problems, see section 1.2 for an explanation. This method allows computing the reconfiguration of several hundred slices in only a few seconds. This is a scalable approach that can be used by a network operator to take a quick decision for reconfiguring when the network is congested.

Our algorithm reconfigures a given set of network slices from an initial routing and placement of network functions to another solution that improves the usage of the network resources (both in terms of link bandwidth and VNFs). This reconfiguration is done with a make-before-break approach to avoid interruptions of the flows.

5.4 Deep Reinforcement Learning Algorithm

The reinforcement learning paradigm formalises a discrete time stochastic control process (as our networking problem) where an *agent* interacts with an *environment* (in our case, the network). At each time step t, the *agent* interacts with its *environment* by (i) observing from the *environment* the current state s, (ii) accordingly, taking a decision (an *action*) a, (iii) receiving a reward $r(s, a)$, and, (iv) observing a new state s' (the network has transitioned from s to s'). The agent can repeat this process for a potential infinite number of time steps, giving rise to a *trajectory*. The sum of the discounted rewards over a trajectory from time t, or *discounted return*, is calculated as $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$, where $\gamma \leq 1$. The expectation of G_t over all possible trajectories initiated at a state s after taking an action a is the so-called *Q-value function* $Q(s, a)$. We aim to take, at each state s, the action a maximising the *Q-value function*. Then, we need to estimate

$Q(s,a)$. In [WD92], authors proposed the *Q-learning* algorithm to learn $Q(s,a)$ from a sequence of agent interactions with the environment. Unfortunately, when the state and action spaces are huge, Q-learning needs a prohibitive computation time. To overcome that, Deep Q-learning Network (DQN) [MKS+15] makes use of a deep neural network to approximate the *Q-value function* for high-dimensional state-space problems, as our case. Finally, we also opt for DQN since, conversely to other reinforcement learning algorithms, it can learn efficiently from past experiences without introducing bias. For an introduction to reinforcement learning, see section 1.3.

Description. For the implementation we use the DQN agent from *tf_agents.agents.dqn.dqn_agent*, and the neural network from *tf_agents.networks.q_network* [A+15]. The network is composed of a pre-processing layer from keras [C+15] used for batch normalization and 2 layers of 64 neurons each. The batch size is 288, which is large enough to properly normalize. We use Adam optimiser with a learning rate of 1e-3 and we update the network every 16 states.
The discount factor γ is set to 0.9, a value large enough to show the importance of future actions. We use a epsilon-greedy policy, where ϵ is set to 0.99 and decay to 0 in 200 instances. The replay buffer has a size of 50 instances and we train the agent on 250 instances.

Context. A 24-hour day consists of 1440 minutes. We decided to discretize it into 288 periods of 5 minutes in order to optimise the training of our agent. It is recalled that the objective is to maximise the profit, while reconfiguring as efficiently as possible. The agent can potentially choose to reconfigure 288 times. To make the agent aware of the implicit trade-off between reconfiguring now or later, an artificial cost per reconfiguration v_R is introduced. Reconfiguring a network will never decrease the profit, but the agent has to learn when a reconfiguration really worth it.

The agent will then learn the optimal number of reconfigurations to maximise the profit with this artificial cost. This cost can be real (management cost) or it can be fixed to get a given number of reconfigurations per day. The advantage of this technique compared to having a maximum number of reconfigurations allowed is that allows the agent to make more or less reconfigurations, adapting its behaviour to the current period of the day.

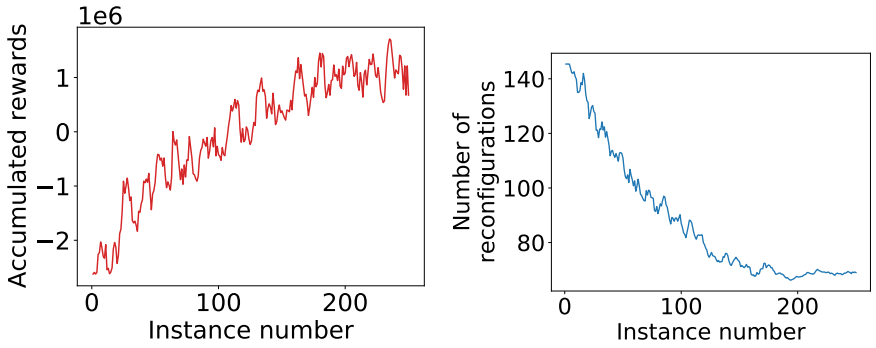
State and Action Spaces. The network state can be described based on the next five quantities: (i) the number of minutes since the last reconfiguration ΔT, (ii) the number of slices added since the last reconfiguration λ, (iii) the number of slices released since the last reconfiguration μ, (iv) the current profit p_t (5.1) and, (v) the current time t. ΔT represent the current allocation oldness, λ and μ estimate the current network load.

The action space consists of two actions: to perform or not a reconfiguration at current time based on the decision of our agent.

Reward Function. If the agent has chosen not to perform a reconfiguration, the reward is 0. Otherwise, the agent selects the reconfiguration, and two scenarios are possible:

1. *The reconfiguration was worth it.* A reconfiguration at time t is computed. To have a long-term vision, we simulate the network behaviour (slices arrivals and departures) with the new network configuration for the next three time slots (in training, we can simulate the future requests). Finally, we estimate the accumulated profit gained with the reconfiguration as $\Delta p_R = \{\sum_{k=t}^{t+3} p_t | \text{reconf at } t\}$.

2. *The reconfiguration was actually not worth it.* The reward is estimated differently. We suppose that no reconfiguration was performed at t and we also simulate the network



(a) Reward (b) Number of reconfigurations

Figure 5.1 – Learning Curves

behaviour (slices arrivals and departures) for the next three time slots. Again, we estimate the accumulated profit gained, this case, without the reconfiguration as $\Delta p_{NR} = \{\sum_{k=t}^{t+3} p_t | \text{no reconf at } t\}$.

Finally, we compute the reward as $r = \Delta p_R - \Delta p_{NR} - v_R$. We therefore have a positive reward when the profit increases Δp_R (if *reconfiguring* was the good decision) compensates both the profit increase Δp_{NR} (if *not reconfiguring* was the good decision) and the reconfiguration cost.

Training. We are now studing the efficiency of the learning of our agent trained on 250 instances. In the Figure 5.1(a) the return of the agent on the training environment increases during the 250 trained instances, which implies that it learns to maximise the reward accumulated on each instance.
We should not reconfigure too often during a day, so in Figure 5.1(b) we study the variation of the number of reconfigurations made by the agent on each instance. The agent starts by reconfiguring randomly: 1 time out of 2 and thus about 144 times per instance, and it learns that it must reduce the number of reconfigurations to maximize the accumulated reward. When the agent has trained on around 200 instances, the epsilon reaches 0 and the number of reconfigurations converges to around 70 reconfigurations per instance.

5.5 Data Set

Topology. We conduct simulations on a real-world topology from SNDlib [OWPT10], `ta1` (24 nodes, 55 links), which includes 6 datacenters on which all VNFs can be instantiated. The cost of VNF c_f is equal to the revenue of 2000 times the revenue u of a megabyte served.

Slice demands Each slice is composed of a chain of up to 5 VNFs, requires a specific amount of bandwidth, and has latency constraints. We consider four different types of demands corresponding to four services: Video Streaming, Web Service, VoIP, and online gaming. The characteristics of each service are reported in Table 5.1 and have been already used by [STV15]. The bandwidth usage was chosen according to the distribution of Internet traffic described in [CIS15]. The latency requirements are expressed in milliseconds and represent the maximum delay between the source and destination.
Each minute, 1 to 5 slice requests arrive (uniform random distribution) and slices that have reached

Slice Types	VNF chain	Latency	bw (Mbps)
Web Service	NAT-FW-TM-WOC-IDPS	10ms	100
Video Streaming	NAT-FW-TM-VOC-IDPS	5ms	256
VoIP	NAT-FW-TM-FW-NAT	3.5ms	64
Online Gaming	NAT-FW-VOC-WOC-IDPS	2.5ms	50

Table 5.1 – Characteristics of network slices

the end of their life are removed from the network. By varying the lifetime of the slices, we can vary the maximum number of slices present at the same time on so that the load on the network follows the curve in Figure 5.2. This figure represents a real distribution of traffic measured on a dedicated network operator. We divided this traffic in five different periods, where D1 is a low-traffic period, and in D5, the network is highly congested. There are between 30 and 180 slices present at each moment on the network and in 24 hours, there are about 4320 arrivals of slices.

Reconfiguration Cost. To train `Deep-REC` we define a fixed and artificial cost to the reconfiguration. This cost can be adapted to reconfigure more or less. In our study, it is equal to the cost of deploying a VNF for 15 minutes, which implies that a reconfiguration is useful if it allows to shut down a VNF for at least 15 minutes. To be usable in practice, a reconfiguration must be done quickly. Thanks to the column generation, we can limit the computation time of each reconfiguration to 15 seconds without affecting its efficiency.

5.6 Numerical Results

We compare the results obtained with three solutions in this section:

- `No-REC`: the slices are added in and removed from the network over time, and no reconfiguration is performed,
- `REC-15`: the reconfiguration is carried out every 15 minutes using a *make-before-break* strategy,
- `Deep-REC`: our deep-learning reconfiguration proposal.

We first show that reconfiguring the network leads to significant gains in terms of profit. We then discuss the importance of selecting the best moments to carry out the reconfigurations, allowing to perform fewer reconfigurations while achieving similar gains.

5.6.1 Improved network usage

Figure 5.3 presents the network cost per megabyte of data sent over the network throughout the day. We observe that the costs achieved by `REC-15` and `Deep-REC` are very similar with a clear improvement compared to
noreconfChapSix: `REC-15` allows an improvement of 36.82% when `Deep-REC` performs a little better with 38.05%. We also see that the cost is rather stable throughout the day while with noreconfChapSix the cost increases strongly during low-congestion period (periods D1 and D2, between 2am and 7am). The global cost improvement through a day of `REC-15` is 34.1% versus 35.55% with `Deep-REC`.
Figure 5.4 shows the achieved profit, whose maximisation is the objective of the reconfiguration: `Deep-REC` and `REC-15` have similar performance and improve the profit compared to noreconfChapSix. Indeed, the profit improvement of `REC-15` is 32.75% versus 32.53% for `Deep-REC`. Note that the improvement in bandwidth acceptance for `REC-15` is 1.42%, against 0.65% for `Deep-REC`.
Finally, Figure 5.5 shows that even when minimising VNF costs, reconfiguring the network does not lead to an increase of congestion: we observe a slightly higher utilization of links during periods D1-D2, but when the network is heavily loaded (periods D4-D5), there is a reduction of the congestion of the network.

As a conclusion, reconfiguring the network reduces congestion while reducing costs. Moreover, we validate with these results the performance of `Deep-REC` as it leads to similar profit as a regular and fixed reconfiguration strategy such as `REC-15`. We show in the following that `Deep-REC`, by performing reconfigurations at the ideal moment, achieves this efficiency while reducing the total number of reconfigurations through a day compared to a regular and fixed reconfiguration strategy such as `REC-15`.

5.6.2 Number of reconfigurations

Figure 5.6 shows the distribution of the number of reconfigurations during a day over two-hour periods. The green line on the figure represents `REC-15` which does a constant number of reconfiguration, namely 8 (a reconfiguration every 15-minutes). In contrast, `Deep-REC` adapts its actions to the network load and does not carry out reconfigurations when they are not necessary,

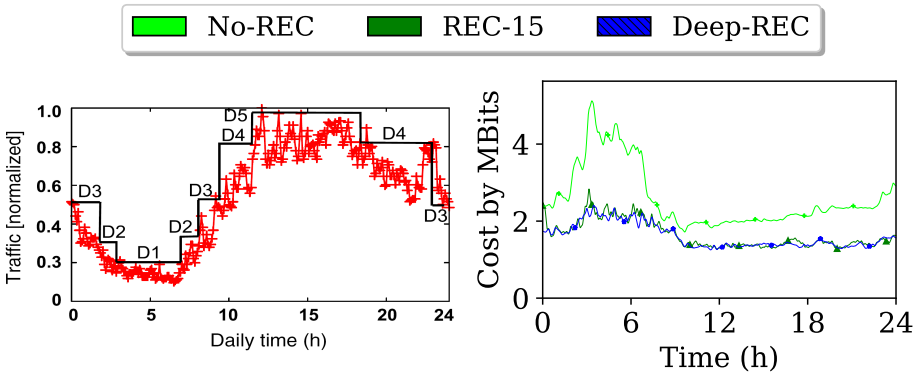


Figure 5.2 – Distribution of traffic

Figure 5.3 – Cost by MB

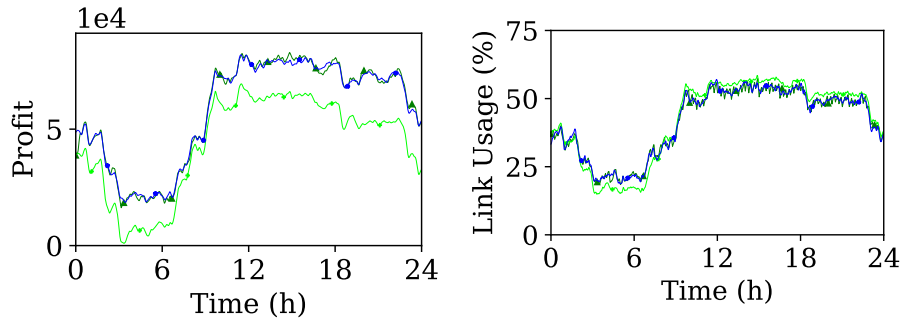


Figure 5.4 – Profit

Figure 5.5 – % of links capacity used

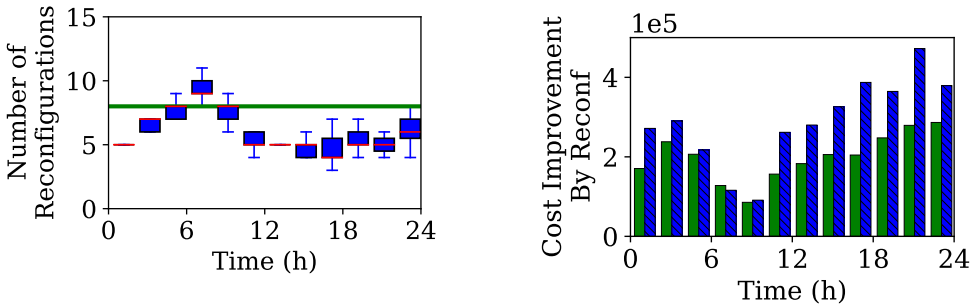


Figure 5.6 – Reconfiguration distribution

Figure 5.7 – Cost gain per Reconf

leading to a reduction of their number during some periods. From 00am to 6am the network load decreases and stabilises at a very low level, `Deep-REC` does less reconfigurations than `REC-15`, nevertheless the number of reconfiguration increases, indeed make-before-break reconfiguration is made easier when the network has more unused capacity and it is simpler to co-locate the VNFs and thus minimise the costs. Moreover, `Deep-REC` performs more reconfigurations than `REC-15` during the ascending phase (6am and 10am) in order to react to the rapid change of the network and, thus, to maintain a good profit. During this period, the improvement given by each reconfiguration is lower than that of `REC-15` as can be seen in Figure 5.7. This may be due to the increased difficulty of the make-before-break reconfiguration to get closer to the optimal allocation but this increase in the number of reconfigurations may also be essential to allow them to be

limited in the following period. During the busiest phase between 10am and 6pm, the number of reconfigurations is at its lowest, as the capacity of the network is so constrained that it becomes harder to co-locate VNFs and reduce costs, thus a small number of reconfigurations maintains the operational costs of the network. Finally, at the end of the day, the network load decreases and `Deep-REC` slightly increases the number of reconfigurations to adapt the allocation to the residual capacity increase. With 96 reconfigurations during a day (against 73.2 in average for `Deep-REC`), `REC-15` has 31.15% more reconfigurations, for only 0.22% profit improvement.

Figure 5.7 presents the cost improvement divided by the number of reconfigurations over periods of two hours. This shows that `Deep-REC` performs more efficient reconfigurations than `REC-15`. Each reconfiguration leads to a better improvement in terms of the network costs.

5.7 Conclusion

We presented in this chapter a deep reinforcement learning strategy to carry out an efficient reconfiguration of network slices with dynamic network demands. Our proposal, `Deep-REC`, chooses adequately the best time to reconfigure, it reconfigures few times during low-congestion periods compared to a fixed-frequency reconfiguration strategy. Moreover, when the network is highly congested, `Deep-REC` adapts his behavior, and reconfigures more in order to maximize the network profit. Finally, `Deep-REC` reduces by almost a quarter the number of reconfigurations needed over a day, while achieving similar performances in terms of network operational costs, achieved profit, and congestion of the network. As a future work we plan to extend the simulations of `Deep-REC` for several networks and to study the efficiency of our proposal by experimentation on a real platform.

References

[A+15] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[C+15] Francois Chollet et al. Keras, 2015.

[CIS15] CISCO. *Cisco Visual Networking Index: Forecast and Methodology, 2014–2019*, May 2015.

[CWM+20] X. Cheng, Y. Wu, G. Min, A. Y. Zomaya, and X. Fang. Safeguard network slicing in 5g: A learning augmented optimization approach. *IEEE JSAC*, 2020.

[GAL+22] A. Gausseran, R. Alliche, H. Lesfari, R. Aparicio-Pardo, F. Giroire, and J. Moulierac. When to reconfigure my network slices? a deep reinforcement learning approach. In *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, pages 1–6, 2022.

[GGJM20] A. Gausseran, F. Giroire, B. Jaumard, and J. Moulierac. Be scalable and rescue my slices during reconfiguration. In *IEEE ICC*, 2020.

[GGJM22] A. Gausseran, F. Giroire, B. Jaumard, and J. Moulierac. Be scalable and rescue my slices during reconfiguration. volume 65, 2022.

[GTGM19] A. Gausseran, A. Tomassilli, F. Giroire, and J. Moulierac. No interruption when reconfiguring my SFCs. In *IEEE International Conference on Cloud Networking (CloudNet)*, pages 1–6, 2019.

[GTGM21] A. Gausseran, A. Tomassilli, F. Giroire, and J. Moulierac. Don't interrupt me when you reconfigure my service function chains. *Computer Communications*, 2021.

[GZL20] W. Guan, H. Zhang, and V. C. M. Leung. Slice reconfiguration based on demand prediction with dueling deep reinforcement learning. In *IEEE GLOBECOM*, 2020.

[HFS+19] D. Harutyunyan, R. Fedrizzi, N. Shahriar, R. Boutaba, and R. Riggio. Orchestrating end-to-end slices in 5g networks. In *15th International Conference on Network and Service Management (CNSM)*, 2019.

[LFC+20] Y. Liu, G. Feng, Z. Chen, S. Qin, and G. Zhao. Network function migration in softwarization based networks with mobile edge computing. In *IEEE ICC*, 2020.

[LHM20] Q. Liu, T. Han, and E. Moges. Edgeslice: Slicing wireless edge computing network with decentralized deep reinforcement learning. In *IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, 2020.

[MKS+15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015.

[OWPT10] Sebastian Orlowski, Roland Wessäly, Michal Pióro, and Artur Tomaszewski. Sndlib 1.0—survivable network design library. *Networks: An International Journal*, 55(3):276–286, 2010.

[PHY20] S. Pandey, J. W. Hong, and J. H. Yoo. Environment aware adaptive q-learning to deploy sfc on edge computing. In *16th International Conference on Network and Service Management (CNSM)*, 2020.

[PNL+20] M. Pozza, P. K. Nicholson, D. F. Lugones, A. Rao, H. Flinck, and S. Tarkoma. On reconfiguring 5g network slices. *IEEE Journal on Selected Areas in Communications*, 2020.

[STV15] Marco Savi, Massimo Tornatore, and Giacomo Verticale. Impact of processing costs on service chain placement in network functions virtualization. In *IEEE Conference NFV-SDN*, 2015.

[SW19] D. Szostak and K. Walkowiak. Machine learning methods for traffic prediction in dynamic optical networks with service chains. In *2019 21st International Conference on Transparent Optical Networks (ICTON)*, 2019.

[SZGS+18] Josep Xavier Salvat, Lanfranco Zanzi, Andres Garcia-Saavedra, Vincenzo Sciancalepore, and Xavier Costa-Perez. Overbooking network slices through yield-driven end-to-end orchestration. In *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '18, New York, NY, USA, 2018. Association for Computing Machinery.

[WD92] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 1992.

[WFS+20] F. Wei, G. Feng, Y. Sun, Y. Wang, S. Qin, and Y. C. Liang. Network slice reconfiguration by exploiting deep reinforcement learning with large action space. *IEEE Transactions on Network and Service Management*, 2020.

[ZLF+17] Nan Zhang, Ya-Feng Liu, Hamid Farmanbar, Tsung-Hui Chang, Mingyi Hong, and Zhi-Quan Luo. Network slicing for service-oriented networks under resource constraints. *IEEE journal on Selected Areas in Communications*, 35(11):2512–2521, 2017.

[ZZC20] J. Zhou, W. Zhao, and S. Chen. Dynamic network slice scaling assisted by prediction in 5g network. *IEEE Access*, 2020.

Conclusion and Perspectives

Traditional networks are now reaching their limits. The increase in throughput and in the number of users cannot be satisfied with a simple hardware evolution. The management of networks also needs to evolve. The introduction of SDN and NFV started this evolution. These two paradigms enable a programmable network and an abstraction of resources, thus allowing an automatic management and a much greater control of traffic and resources. 5G further increases the need for evolution and for innovation in network management. It introduces strong constraints on latency, mobility, throughput, energy efficiency, security, and above all QoS. The infrastructure of mobile networks is also evolving, but this must be coupled with a change in the network management approach. The great heterogeneity of use cases and of their requirements led to the introduction of network slicing. This new paradigm involves the partitioning of the physical network into multiple independent virtual networks, each of which provides a specific service to meet precise requirements. With the new paradigm comes new algorithms with as many objectives as use cases.

In chapter 2, we have studied a data structure allowing a simple modelling of the ordering of network functions within a service chain. This structure is used throughout the thesis and allows to transform a routing and allocation problem into a simple routing problem. An ILP based on this structure has been developed for static placement problems of a set of SFCs, as well as for the dynamic placement problem of an SFC. We have compared our formulation to an ILP from the literature, and it allows an intuitive modelling of the service chain without degrading the computation time.

In the following of the thesis we looked at the reconfiguration of networks and demands. The reconfiguration allows to limit the congestion and fragmentation effects induced by the dynamicity of the arrivals and departures of requests over time. Reconfiguring improves traffic and resource management, minimises Opex, and increases request acceptance.

In chapter 3, we investigated the feasibility and performance of the make-before-break reconfiguration of SFCs within a network. This type of reconfiguration allows to modify the allocation of VNFs and to reroute traffic without interrupting it. A secondary route is created, its capacity is reserved, and the two routes co-exist during the reconfiguration. Not interrupting the traffic ensures that the QoS is not degraded. We have shown that this type of reconfiguration, although constraining, allows to significantly decrease Opex and to improve the acceptance of requests without interrupting the traffic flows. An ILP and a heuristic have been developed to process up to fifty SFCs in less than a minute on networks with more than a hundred links.

A relevant extension of this work would be to take into account path protection in our algorithm. Indeed, the reconfiguration does not implement protection for the new routes. Furthermore, reconfiguration could be potentially more effective if it used the capacity allocated to backup paths during intermediate steps. As the reconfiguration steps are fast, the probability of a failure occurring during this time is low, and this reserved capacity would allow better reconfiguration performances during heavy loaded periods.

In chapter 4, we extended the make-before-break reconfiguration to network slices. This type of reconfiguration is relevant because it allows to respects the QoS which is an important requirement of network slicing. Our study focused on very dense scenarios in terms of number of slices, over a 24-hours period, following the dynamicity in terms of load of a typical day. We considered an important scaling of our reconfiguration technique to cope with the very high density of future 5G networks. We reconfigured up to about 500 slices in one minute, representing approximately 1500 different traffic flows, and more than a hundred slices in a few seconds. To achieve this, we developed two column generation models, one based on an ILP based pricing problem and one with an LP based pricing problem.

A possible evolution of this work would be to take into account the specific features of the slices as well as the differences in treatment between the edge and the core network. Indeed, the edge does not have the same computing power as the core, but it can process requests with lower latency. uRLLC slices have much stronger latency constraints than eMBB slices and could have priority on the edge servers. The eMBB slices, on the contrary, have much higher computing power requirements and should be allocated in the core. Placement and reconfiguration techniques which take into account theses priorities would further improve the use of resources and ensure greater fairness in the acceptance of slices. The uRLLC slices would have a better chance of being routed if edge servers are not too heavily used by eMBB slices. The latter may be processed in the core during peak periods.

Finally, in chapter 5, we have extended our reconfiguration method. Although reconfiguring at fixed intervals improves the utilisation of network resources, it does not adapt to the variation of the traffic. Some periods of the day do not need as much reconfiguration and reconfiguring at those moments can add more management cost than it saves in Opex. On the contrary, during certain periods, reconfiguring more regularly can significantly improve cost reduction and acceptance of requests. To achieve this adaptability we have developed a reconfiguration agent based on Deep Reinforcement Learning which finds the best moment during the day to reconfigure. The agent adapts to the traffic load and time of day to reduce the number of reconfigurations and make each one more effective. The reconfiguration algorithm and the agent are completely independent of each other and the agent uses the reconfiguration as a black box. This makes it completely independent of the reconfiguration method and allows for a potential easy deployment.

An interesting improvement that could be studied would be to couple the possible improvement of the previous chapter, i.e. to treat the type of slice differently, with a new reinforcement learning agent. This agent would have controlled over the placement and reconfiguration objective. It would prioritise slices acceptance in periods of high load and it would prioritise cost reduction in off-peak periods. Furthermore, it could choose to process slices at the edge when few slices are active. This would allow to turn off part of the transport network and save energy. When the load increases, it could reconfigure the eMBB slices to reroute them to the core network to accept more new uRLLC slices.

In this thesis, we have investigated the optimisation potential of make-before-break reconfiguration of slices. This technique allows to preserve the QoS of requests, a very important requirement in the context of network slicing. The reconfiguration can be coupled with efficient placement techniques and allows to maintain the network in a near optimal state. It can be useful in the event of a change in placement policy. It may allow the new policy to be propagated to the slices already placed and thus to adapt the network management in real time. The adaptation of this optimisation technique to the edge and RAN are possible issues to consider. But in general

we believe that this technique can be easily applied to 5G networks in deployment and to SDN-NFV networks already in use. It can improve the utility of the networks and ultimately reduce the energy consumption and financial expenditure they cause.

We have focused on Opex reduction and demand acceptance, but it would be worth investigating other objectives. Optimisation could potentially focus on reducing energy consumption, route resilience, etc. Optimisation algorithms for network slicing must take into account the specific features of 5G and network slicing such as end-to-end service management, user mobility and resource sharing. They must also consider the differences in performance in terms of computing power and latency between edge and core networks as well as the respect of QoS.

In addition to having different needs, slices may have different priorities. An interesting perspective would be to take into account the possibility of booking slices in advance. The placement of these requests could be pre-computed and capacity could be reserved. This reserved capacity should not remain unused as this would reduce the network's utility. It would be interesting to provide admission control algorithms that would allow as many requests as possible to be accepted while ensuring that reserved slices are accepted. These admission controls could be coupled with reconfiguration algorithms whose objective would be to ensure the acceptance of these slices (and not to minimise the Opex).

Another interesting extension of our work would be the adaptation of the reconfiguration within the datacenters. The dispersion of virtual functions on different server racks can lead to a high bandwidth usage. In addition, the latency constraint between two functions of the same slice can be very low and locating two functions on the same rack removes the transmission delay. A reconfiguration algorithm designed to manage flows within the datacenter would allow for more efficient use of resources.

Finally, a last interesting perspective would be to take into account fog computing. In addition to distributing the processing on devices at the edge of the network, we could also distribute the processing on units in the local network. This would allow to support more efficiently uRLLC and mMTC slices which are very dependent on low delays and/or hyper connectivity. As the make-before-break reconfiguration is very capacity constrained, taking this additional capacity into account would make this method much more effective.

Bibliography

[00114] ETSI GS NFV-MAN 001. Etsi gs nfv-man 001 v1.1.1 (2014-12)network functions virtualisation (nfv);management and orchestration. https://www.etsi.org/deliver/etsi_gs/nfv-man/001_099/001/01.01.01_60/gs_nfv-man001v010101p.pdf, 12 2014.

[01017] ETSI GR NFV-EVE 010. Network functions virtualisation(nfv) release 3; licensing management; report on license management for nfv. https://www.etsi.org/deliver/etsi_gr/NFV-EVE/001_099/010/03.01.01_60/gr_nfv-eve010v030101p.pdf, 12 2017.

[01118] ETSI GR NGP 011. Next generation protocols(ngp);e2e network slicing reference framework and information model. https://www.etsi.org/deliver/etsi_gr/NGP/001_099/011/01.01.01_60/gr_ngp011v010101p.pdf, 09 2018.

[01217] ETSI GR NFV-EVE 012. Network functions virtualisation (nfv) release 3;evolution and ecosystem; report on network slicing support with etsi nfv architecture framework. https://www.etsi.org/deliver/etsi_gr/NFV-EVE/001_099/012/03.01.01_60/gr_NFV-EVE012v030101p.pdf, 12 2017.

[22.16] 3GPP TR 22.891. Study on new services and markets technology enablers. https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2897, 9 2016.

[38.20] 3GPP TR 38.913. Study on scenarios and requirements for next generation access technologies. https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2996, 7 2020.

[A+15] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[ABBS15] Bernardetta Addis, Dallal Belabed, Mathieu Bouet, and Stefano Secci. Virtual network functions placement and routing optimization. In *2015 IEEE 4th International Conference on Cloud Networking (CloudNet)*, pages 171–177, 2015.

[ACF+17] Ahmed Abdelsalam, Francois Clad, Clarence Filsfils, Stefano Salsano, Giuseppe Siracusano, and Luca Veltri. Implementation of virtual network function chaining through segment routing in a linux-based nfv infrastructure. In *2017 IEEE Conference on Network Softwarization (NetSoft)*, pages 1–5, 2017.

[AFT07] Brice Augustin, Timur Friedman, and Renata Teixeira. Measuring load-balanced paths in the internet. In *ACM Internet Measurement Conference (IMC)*, pages 149–160. ACM, 2007.

[All15] The Next Generation Mobile Networks Alliance. 5g white paper. https://www.ngmn.org/wp-content/uploads/NGMN_5G_White_Paper_V1_0.pdf, 02 2015.

[ARS16] Mamta Agiwal, Abhishek Roy, and Navrati Saxena. Next generation 5g wireless networks: A comprehensive survey. *IEEE Communications Surveys Tutorials*, 18(3):1617–1655, 2016.

[ATS+18] Ibrahim Afolabi, Tarik Taleb, Konstantinos Samdanis, Adlen Ksentini, and Hannu Flinck. Network slicing and softwarization: A survey on principles, enabling technologies, and solutions. *IEEE Communications Surveys Tutorials*, 20(3):2429–2453, 2018.

[AZA16] Sara Ayoubi, Yanhong Zhang, and Chadi Assi. A reliable embedding framework for elastic virtualized services in the cloud. *IEEE Transactions on Network and Service Management (IEEE TNSM)*, 13(3):489–503, 2016.

[B+14] Pankaj Berde et al. Onos: towards an open, distributed sdn os. In *Workshop on Hot topics in software defined networking*, pages 1–6. ACM, 2014.

[BAMH20] Alcardo Alex Barakabitze, Arslan Ahmad, Rashid Mijumbi, and Andrew Hines. 5g network slicing using sdn and nfv: A survey of taxonomy, architectures and future challenges. *Computer Networks*, 167:106984, 2020.

[Bel58] Richard Bellman. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

[BGB+17] D. Bega, M. Gramaglia, A. Banchs, V. Sciancalepore, K. Samdanis, and X. Costa-Perez. Optimising 5G infrastructure markets: The business of network slicing. In *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 1–9, 2017.

[BJSE16] Deval Bh, Raj Jain, Mohammed Samaka, and Aiman Erbad. A survey on service function chaining. *Journal of Network and Computer Applications*, 75, 09 2016.

[BRC+13] Md. Faizul Bari, Arup Raton Roy, Shihabur Chowdhury, Qi Zhang, Mohamed Faten Zhani, Reaz Ahmed, and R. Boutaba. Dynamic controller provisioning in software defined networks. 10 2013.

[C+15] Francois Chollet et al. Keras, 2015.

[CB02] B. Carpenter and Scott Brim. Middleboxes: taxonomy and issues. 01 2002.

[CIS15] CISCO. *Cisco Visual Networking Index: Forecast and Methodology, 2014–2019*, May 2015.

[CLBB+15] Marcelo Caggiani Luizelli, Leonardo Bays, Luciana Buriol, Marinho Barcellos, and Luciano Gaspary. Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions. 05 2015.

[CLENR15] R. Cohen, L. Lewin-Eytan, J.S. Naor, and D. Raz. Near optimal placement of virtual network functions. In *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 1346–1354, Kowloon, Hong-Kong, 2015.

[CLX+10] Zhiping Cai, Fang Liu, Nong Xiao, Qiang Liu, and Zhiying Wang. Virtual network embedding for evolving networks. In *IEEE Global Telecommunications Conference - GLOBECOM*, pages 1–5. IEEE, 2010.

[CTZB17] Daewoong Cho, Javid Taheri, Albert Y. Zomaya, and Pascal Bouvry. Real-time virtual network function (vnf) migration toward low network latency in cloud environments. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pages 798–801, 2017.

[CWM+20] X. Cheng, Y. Wu, G. Min, A. Y. Zomaya, and X. Fang. Safeguard network slicing in 5g: A learning augmented optimization approach. *IEEE JSAC*, 2020.

[Dan48] George B Dantzig. Programming in a linear structure. *Washington, DC*, 1948.

[Dan63] George Bernard Dantzig. *Linear Programming and Extensions*. RAND Corporation, Santa Monica, CA, 1963.

[DDS05] Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon, editors. *Column Generation*. Number 978-0-387-25486-9 in Springer Books. Springer, May 2005.

[DGT+19] Tilemachos Doukoglou, Velissarios Gezerlis, Konstantinos Trichias, Nikos Kostopoulos, Nikos Vrakas, Marios Bougioukos, and Rodolphe Legouable. Vertical industries requirements analysis targeted kpis for advanced 5g trials. In *2019 European Conference on Networks and Communications (EuCNC)*, pages 95–100, 2019.

[DHM+13] Advait Dixit, Fang Hao, Sarit Mukherjee, T. Lakshman, and Ramana Kompella. Towards an elastic distributed sdn controller. volume 43, pages 7–12, 09 2013.

[DJCA18] Huy Duong, Brigitte Jaumard, David Coudert, and Ron Armolavicius. Efficient make before break capacity defragmentation. In *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*, pages 1–6, 2018.

[dT21] Fédération Française des Télécoms. Chiffres clés. https://www.fftelecoms.org/chiffres-cles/, 2021.

[DW16] Abhishek Dwaraki and Tilman Wolf. Adaptive service-chain routing for virtual network functions in software-defined networks. In *Proceedings of the 2016 workshop on Hot topics in Middleboxes and Network Function Virtualization*, pages 32–37, 2016.

[EMAL17] Vincenzo Eramo, Emanuele Miucci, Mostafa Ammar, and Francesco Giacinto Lavacca. An approach for service function chain routing and virtual function network instance migration in network function virtualization architectures. *IEEE/ACM Transactions on Networking (ToN)*, 25(4):2008–2025, 2017.

[FAPZ11] Ilhem Fajjari, Nadjib Aitsaadi, Guy Pujolle, and Hubert Zimmermann. VNR algorithm: A greedy approach for virtual networks reconfigurations. In *IEEE Global Telecommunications Conference - GLOBECOM*, pages 1–6. IEEE, 2011.

[Fou15] Open Networking Foundation. Tr-518 relationship of sdn and nfv. https://opennetworking.org/wp-content/uploads/2014/10/onf2015.310_Architectural_comparison.08-2.pdf, 10 2015.

[Fou16] Open Networking Foundation. Tr-521 sdn architecture. https://opennetworking.org/wp-content/uploads/2014/10/TR-521_SDN_Architecture_issue_1.1.pdf, 2016.

[FVW19] Klaus-Tycho Foerster, Laurent Vanbever, and Roger Wattenhofer. Latency and consistent flow migration: Relax for lossless updates. In *2019 IFIP Networking Conference (IFIP Networking)*, pages 1–9, 2019.

[FZI16] Pingzhi Fan, Jing Zhao, and Chih-Lin I. 5g high mobility wireless communications: Challenges and solutions. *China Communications*, 13(Supplement2):1–13, 2016.

[GAL+22] A. Gausseran, R. Alliche, H. Lesfari, R. Aparicio-Pardo, F. Giroire, and J. Moulierac. When to reconfigure my network slices? a deep reinforcement learning approach. In *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, pages 1–6, 2022.

[Gaua] Adrien Gausseran. Maze qlearning example. https://github.com/AdrienGausseran/MazeQLearningExample.

[Gaub] Adrien Gausseran. Multi commodity flow column generation. https://github.com/AdrienGausseran/MultiCommodityFlow_ColumnGeneration.

[GGJM19] Adrien Gausseran, Frédéric Giroire, Brigitte Jaumard, and Joanna Moulierac. Don't break network slices during reconfiguration. Technical report, Inria, Dec. 2019.

[GGJM20] A. Gausseran, F. Giroire, B. Jaumard, and J. Moulierac. Be scalable and rescue my slices during reconfiguration. In *IEEE ICC*, 2020.

[GGJM22] A. Gausseran, F. Giroire, B. Jaumard, and J. Moulierac. Be scalable and rescue my slices during reconfiguration. volume 65, 2022.

[GHM15] Frédéric Giroire, Frédéric Havet, and Joanna Moulierac. Compressing two-dimensional routing tables with order. In *7th Network Optimization Conference (INOC)*, 2015.

[GHMP18] Frédéric Giroire, Nicolas Huin, Joanna Moulierac, and Truong Khoa Phan. Energy-Aware Routing in Software-Defined Network using Compression. *The Computer Journal*, 61(10):1537–1556, 03 2018.

[GJ02] Michael R Garey and David S Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002.

[GJPGA12] Aaron Gember-Jacobson, Prathmesh Prabhu, Zainab Ghadiyali, and Aditya Akella. Toward software-defined middlebox networking. pages 7–12, 10 2012.

[GK07] Naveen Garg and Jochen Koenemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM Journal on Computing*, 37(2):630–652, 2007.

[GKS+15] Milad Ghaznavi, Aimal Khan, Nashid Shahriar, Khalid Alsubhi, Reaz Ahmed, and Raouf Boutaba. Elastic virtual network function placement. In *IEEE International Conference on Cloud Networking (CloudNet)*, pages 255–260, 2015.

[GR18] Lingnan Gao and George N Rouskas. Virtual network reconfiguration with load balancing and migration cost considerations. In *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 2303–2311. IEEE, 2018.

[GTGM18] Adrien Gausseran, Andrea Tomassilli, Frédéric Giroire, and Joanna Moulierac. Don't Interrupt Me When You Reconfigure my Service Function Chains. Technical report, Inria, Dec. 2018.

[GTGM19a] A. Gausseran, A. Tomassilli, F. Giroire, and J. Moulierac. No interruption when reconfiguring my SFCs. In *IEEE International Conference on Cloud Networking (CloudNet)*, pages 1–6, 2019.

[GTGM19b] Adrien Gausseran, Andrea Tomassilli, Frederic Giroire, and Joanna Moulierac. Poster: Don't interrupt me when you reconfigure my service function chains. In *2019 IFIP Networking Conference (IFIP Networking)*, pages 1–2, 2019.

[GTGM19c] Adrien Gausseran, Andrea Tomassilli, Frédéric Giroire, and Joanna Moulierac. Reconfiguration de chaînes de fonctions de services sans interruption. In *CORES 2019 - Rencontres Francophones sur la Conception de Protocoles, l'Évaluation de Performance et l'Expérimentation des Réseaux de Communication*, Saint Laurent de la Cabrerisse, France, June 2019.

[GTGM21] A. Gausseran, A. Tomassilli, F. Giroire, and J. Moulierac. Don't interrupt me when you reconfigure my service function chains. *Computer Communications*, 2021.

[GZL20] W. Guan, H. Zhang, and V. C. M. Leung. Slice reconfiguration based on demand prediction with dueling deep reinforcement learning. In *IEEE GLOBECOM*, 2020.

[GZT+19] Lin Gu, Deze Zeng, Sheng Tao, Song Guo, Hai Jin, Albert Y. Zomaya, and Weihua Zhuang. Fairness-aware dynamic rate control and flow scheduling for network utility maximization in network service chain. *IEEE Journal on Selected Areas in Communications*, 37(5):1059–1071, 2019.

[HB16] Juliver Gil Herrera and Juan Felipe Botero. Resource allocation in NFV: A comprehensive survey. *IEEE Transactions on Network and Service Management (IEEE TNSM)*, 13(3):518–532, 2016.

[HFS+19] D. Harutyunyan, R. Fedrizzi, N. Shahriar, R. Boutaba, and R. Riggio. Orchestrating end-to-end slices in 5g networks. In *15th International Conference on Network and Service Management (CNSM)*, 2019.

[HJG18] Nicolas Huin, Brigitte Jaumard, and Frédéric Giroire. Optimal network service chain provisioning. *IEEE/ACM Transactions on Networking (ToN)*, 26(3):1320–1333, June 2018.

[HMM+19] Nicolas Huin, Paolo Medagliani, Sébastien Martin, Jérémie Leguay, Lei Shi, Shengming Cai, Jinchun Xu, and Hao Shi. Hard-isolation for network slicing. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 955–956, 2019.

[HTGJ18] Nicolas Huin, Andrea Tomassilli, Frederic Giroire, and Brigitte Jaumard. Energy-efficient service function chain provisioning. *IEEE/OSA Journal of Optical Communications and Networking*, 10(3):114–124, 2018.

[ID05] Stefan Irnich and Guy Desaulniers. Shortest path problems with resource constraints. In *Column generation*, pages 33–65. Springer, 2005.

[ITU18] International Telecommunication Union ITU. Itu-r m.2083; setting the scene for 5g: opportunities and challenges. http://handle.itu.int/11.1002/pub/811d7a5f-en, 2018.

[Jae15] Bernd Jaeger. Security orchestrator: Introducing a security orchestrator in the context of the etsi nfv reference architecture. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 1255–1260, 2015.

[JKM+13] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. B4: Experience with a globally-deployed software defined wan. *SIGCOMM Comput. Commun. Rev.*, 43(4):3–14, August 2013.

[Kar84] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, Dec 1984.

[KBB19] Nguyen Tuan Khai, Andreas Baumgartner, and Thomas Bauschert. Optimising virtual network functions migrations: A flexible multi-step approach. In *2019 IEEE Conference on Network Softwarization (NetSoft)*, pages 188–192, 2019.

[KBK+12] James Kempf, Elisa Bellagamba, András Kern, Dávid Jocha, Attila Takacs, and Pontus Sköldström. Scalable fault management for openflow. In *2012 IEEE International Conference on Communications (ICC)*, pages 6606–6610, 2012.

[KF13] Hyojoon Kim and Nick Feamster. Improving network management with Software Defined Networking. *IEEE Communications Magazine*, 51(2):114–119, 2013.

[Kha80] L.G. Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980.

[KHZ17] Selma Khebbache, Makhlouf Hadji, and Djamal Zeghlache. Scalable and cost-efficient algorithms for vnf chaining and placement problem. In *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, pages 92–99, 2017.

[KLLT18] Tung-Wei Kuo, Bang-Heng Liou, Kate Ching-Ju Lin, and Ming-Jer Tsai. Deploying chains of virtual network functions: On the relation between link and server usage. *IEEE/ACM Transactions on Networking (TON)*, 26(4):1562–1576, 2018.

[KN17] Adlen Ksentini and Navid Nikaein. Toward enforcing network slicing on ran: Flexibility and resources abstraction. *IEEE Communications Magazine*, 55(6):102–108, 2017.

[KNS+18] Zbigniew Kotulski, Tomasz Wojciech Nowak, Mariusz Sepczuk, Marcin Tunia, Rafal Artych, Krzysztof Bocianiak, Tomasz Osko, and Jean-Philippe Wary. Towards constructive approach to end-to-end slice isolation in 5g networks. *EURASIP Journal on Information Security*, 2018(1):2, Mar 2018.

[KREV+15] D. Kreutz, F.M.V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, Jan 2015.

[LFC+20] Y. Liu, G. Feng, Z. Chen, S. Qin, and G. Zhao. Network function migration in softwarization based networks with mobile edge computing. In *IEEE ICC*, 2020.

[LHK+13] Pingping Lin, Jonathan Hart, Umesh Krishnaswamy, Tetsuya Murakami, Masayoshi Kobayashi, Ali Al-Shabibi, Kuang-Ching Wang, and Jun Bi. Seamless interworking of sdn and ip. volume 43, pages 475–476, 08 2013.

[LHM10] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: Rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, New York, NY, USA, 2010. Association for Computing Machinery.

[LHM20] Q. Liu, T. Han, and E. Moges. Edgeslice: Slicing wireless edge computing network with decentralized deep reinforcement learning. In *IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, 2020.

[Lin91] Long-Ji Lin. Programming robots using reinforcement learning and teaching. In *Proceedings of the Ninth National Conference on Artificial Intelligence - Volume 2*, AAAI'91, page 781–786. AAAI Press, 1991.

[LLZ+17] Junjie Liu, Wei Lu, Fen Zhou, Ping Lu, and Zuqing Zhu. On dynamic service function chain deployment and readjustment. *IEEE Transactions on Network and Service Management (IEEE TNSM)*, 14(3):543–553, 2017.

[LPMK18] Mathieu Leconte, Georgios S Paschos, Panayotis Mertikopoulos, and Ulaş C Kozat. A resource allocation framework for network slicing. In *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 2177–2185. IEEE, 2018.

[LZC+16] Hongjian Li, Guofeng Zhu, Chengyuan Cui, Hong Tang, Yusheng Dou, and Chen He. Energy-efficient migration and consolidation algorithm of virtual machines in data centers for cloud computing. *Computing*, 98(3):303–317, Mar 2016.

[M. 13] M. Chiosi *et al.* Network functions virtualisation (NFV) network operator perspectives on industry progress. In *SDN & OpenFlow World Congress*, Dusseldorf, Germany, October 2013.

[M+16] Rashid Mijumbi et al. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 18(1):236–262, 2016.

[Mal98] Gary S. Malkin. RIP Version 2. RFC 2453, November 1998.

[MF19] Dania Marabissi and Romano Fantacci. Highly flexible ran slicing approach to manage isolation, priority, efficiency. *IEEE Access*, 7:97130–97142, 2019.

[MGT+15] Jon Matias, Jokin Garay, Nerea Toledo, Juanjo Unzilla, and Eduardo Jacob. Toward an SDN-enabled NFV architecture. *IEEE Communications Magazine*, 53(4):187–193, 2015.

[MKS+15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015.

[Moy98] John Moy. OSPF Version 2. RFC 2328, April 1998.

[MSB+17] Wenrui Ma, Oscar Sandoval, Jonathan Beltran, Deng Pan, and Niki Pissinou. Traffic aware placement of interdependent NFV middleboxes. In *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 1–9, 2017.

[MTC+19] Cédric Morin, Geraldine Texier, Christelle Caillouet, Gilles Desmangles, and Cao-Thanh Phan. Vnf placement algorithms to address the mono-and multi-tenant issues in edge and core networks. In *2019 IEEE 8th International Conference on Cloud Networking (CloudNet)*, pages 1–6, 2019.

[NKT19] Kyoomars Alizadeh Noghani, Andreas J. Kassler, and Javid Taheri. On the Cost-Optimality Trade-off for Service Function Chain Reconfiguration. In *IEEE International Conference on Cloud Networking (CloudNet)*, 2019.

[ONF14] Optical Interconnecting Forum Open Networking Foundation. Global transport sdn prototype demonstration. https://opennetworking.org/wp-content/uploads/2013/02/oif-p0105_031_18.pdf, 10 2014.

[OWPT10] Sebastian Orlowski, Roland Wessäly, Michal Pióro, and Artur Tomaszewski. Sndlib 1.0—survivable network design library. *Networks: An International Journal*, 55(3):276–286, 2010.

[PDM+16] Stefano Paris, Apostolos Destounis, Lorenzo Maggi, Georgios S Paschos, and Jérémie Leguay. Controlling flow reconfigurations in SDN. In *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 1–9. IEEE, 2016.

[PHY20] S. Pandey, J. W. Hong, and J. H. Yoo. Environment aware adaptive q-learning to deploy sfc on edge computing. In *16th International Conference on Network and Service Management (CNSM)*, 2020.

[PNL+20] M. Pozza, P. K. Nicholson, D. F. Lugones, A. Rao, H. Flinck, and S. Tarkoma. On reconfiguring 5g network slices. *IEEE Journal on Selected Areas in Communications*, 2020.

[PPP20] 5G PPP. View on 5g architecture. https://5g-ppp.eu/wp-content/uploads/2020/02/5G-PPP-5G-Architecture-White-Paper_final.pdf, 02 2020.

[PPR+19] M. Pozza, A. Patel, A. Rao, H. Flinck, and S. Tarkoma. Composing 5G network slices by co-locating VNFs in μslices. In *IFIP Networking Conference*, pages 1–9, May 2019.

[QKA18] Long Qu, Maurice Khabbaz, and Chadi Assi. Reliability-aware service chaining in carrier-grade softwarized networks. *IEEE Journal on Selected Areas in Communications*, PP, 03 2018.

[QN15] P. Quinn and T. Nadeau. Problem statement for service function chaining. RFC 7498, RFC Editor, April 2015.

[R+17] Peter Rost et al. Network slicing to enable scalability and flexibility in 5G mobile networks. *IEEE Communications magazine*, 55(5):72–79, 2017.

[RHL06] Yakov Rekhter, Susan Hares, and Tony Li. A Border Gateway Protocol 4 (BGP-4). RFC 4271, January 2006.

[S+15] Sahel Sahhaf et al. Network service chaining with optimized network function embedding supporting service decompositions. *Computer Networks*, 93:492–505, 2015.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[SGT20] S. Sharma, A. Gumaste, and M. Tatipamula. Dynamic network slicing using utility algorithms and stochastic optimization. In *2020 IEEE 21st International Conference on High Performance Switching and Routing (HPSR)*, 2020.

[SHS+12] Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. Making middleboxes someone else's problem: network processing as a cloud service. *ACM SIGCOMM Computer Communication Review*, 42(4):13–24, 2012.

[SJG+17] Yu Sang, Bo Ji, Gagan R Gupta, Xiaojiang Du, and Lin Ye. Provably efficient algorithms for joint placement and allocation of virtual network functions. In *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 1–9. IEEE, 2017.

[SMGZ17] Oussama Soualah, Marouen Mechtri, Chaima Ghribi, and Djamal Zeghlache. Energy efficient algorithm for vnf placement and chaining. pages 579–588, 05 2017.

[STV15] Marco Savi, Massimo Tornatore, and Giacomo Verticale. Impact of processing costs on service chain placement in network functions virtualization. In *IEEE Conference NFV-SDN*, 2015.

[SW19] D. Szostak and K. Walkowiak. Machine learning methods for traffic prediction in dynamic optical networks with service chains. In *2019 21st International Conference on Transparent Optical Networks (ICTON)*, 2019.

[SZGS+18] Josep Xavier Salvat, Lanfranco Zanzi, Andres Garcia-Saavedra, Vincenzo Sciancalepore, and Xavier Costa-Perez. Overbooking network slices through yield-driven end-to-end orchestration. In *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '18, New York, NY, USA, 2018. Association for Computing Machinery.

[TAM19] S. Troia, R. Alvizu, and G. Maier. Reinforcement learning for service function chain reconfiguration in nfv-sdn metro-core optical networks. *IEEE Access*, 2019.

[TGHP18] Andrea Tomassilli, Frédéric Giroire, Nicolas Huin, and Stéphane Pérennes. Provably efficient algorithms for placement of Service Function Chains with ordering constraints. In *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 774–782, Honolulu, Hawai, US, 2018. IEEE.

[THGJ18] Andrea Tomassilli, Nicolas Huin, Frederic Giroire, and Brigitte Jaumard. Resource requirements for reliable service function chaining. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–7, 2018.

[TTG13] Phuong Nga Tran and Andreas Timm-Giel. Reconfiguration of virtual network mapping considering service disruption. In *IEEE International Conference on Communications - ICC*, pages 3487–3492. IEEE, 2013.

[Var12] Upkar Varshney. 4g wireless networks. *IT Professional*, 14(5):34–39, 2012.

[WBIC19] Adrien Wion, Mathieu Bouet, Luigi Iannone, and Vania Conan. Change in continuity: Chaining services with an augmented igp. *IEEE Transactions on Network and Service Management*, 16(4):1332–1344, 2019.

[WD92] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 1992.

[WFQ+19] G. Wang, G. Feng, T.Q.S. Quek, S. Qin, R. Wen, and W. Tan. Reconfiguration in network slicing-optimizing the profit and performance. *IEEE Transactions on Network and Service Management*, 16(2):591–605, June 2019.

[WFS+20] F. Wei, G. Feng, Y. Sun, Y. Wang, S. Qin, and Y. C. Liang. Network slice reconfiguration by exploiting deep reinforcement learning with large action space. *IEEE Transactions on Network and Service Management*, 2020.

[WM13] R. Wang and B. Mukherjee. Provisioning in elastic optical networks with non-disruptive defragmentation. *IEEE Journal of Lightwave Technology*, 31(15):2491–2500, 2013.

[YG12] Soheil Yeganeh and Yashar Ganjali. Kandoo: A framework for efficient and scalable offloading of control applications. *HotSDN'12 - Proceedings of the 1st ACM International Workshop on Hot Topics in Software Defined Networks*, 08 2012.

[YLW+19] Xu Yang, Yue Liu, Ieok Cheng Wong, Yapeng Wang, and Laurie Cuthbert. Effective isolation in dynamic network slicing. In *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6, 2019.

[ZLF+17] Nan Zhang, Ya-Feng Liu, Hamid Farmanbar, Tsung-Hui Chang, Mingyi Hong, and Zhi-Quan Luo. Network slicing for service-oriented networks under resource constraints. *IEEE journal on Selected Areas in Communications*, 35(11):2512–2521, 2017.

[ZZC20] J. Zhou, W. Zhao, and S. Chen. Dynamic network slice scaling assisted by prediction in 5g network. *IEEE Access*, 2020.

Algorithmes d'optimisation pour le Network Slicing pour la 5G

Adrien GAUSSERAN

Résumé

Notre décennie voit l'accroissement de l'utilisation des réseaux mobiles pour les acteurs industriels et administratifs ainsi que pour le grand public grâce à l'introduction de la 5ème génération de réseaux mobiles : la 5G. La 5G apporte une diversité de cas d'utilisation des réseaux mobiles et un nombre croissant de demandes avec des besoins très hétérogènes mais toujours avec de fortes contraintes de qualité de service (QoS). La 5G a été développée pour utiliser les technologies de réseaux définis par logiciel (SDN) et de virtualisation de fonctions réseaux (NFV). SDN sépare les plans de contrôle et de données et offre une gestion centralisée du réseau. NFV dissocie les fonctions réseaux du matériel qui les exécute grâce à la virtualisation. Ces technologies automatisent la gestion du réseau et le rendent plus flexible et adaptable à l'évolution du débit du trafic ainsi que de ses besoins. L'introduction du paradigme de découpage du réseau entraîne une division du réseau en des réseaux virtuels indépendants cohabitant sur la même infrastructure. Ce paradigme permettra de répondre aux besoins très hétérogènes des futures demandes. Dans cette thèse nous nous intéressons à l'optimisation de l'utilisation des ressources des réseaux de nouvelle génération afin de diminuer les coûts opérationnels et d'accepter plus de demandes. Nous étudions d'abord l'allocation de chaînes de fonctions de services, pour accepter rapidement les requêtes et répondre aux demandes diverses et abondantes des réseaux mobiles. Nous étudions ensuite la faisabilité de la reconfiguration Make-Before-Break des requêtes, qui permet de reconfigurer sans dégrader la QoS. Nous adaptons ensuite notre reconfiguration au network slicing pour l'utiliser sur les futures réseaux 5G. Enfin nous optimisons les périodes de reconfiguration grâce à un algorithme d'apprentissage par renforcement, réduisant ainsi les coûts de gestion.

Mots-clés : SDN, NFV, SFC, 5G, Slicing, Reconfiguration.

Abstract

Our decade is marked by an increase in the use of mobile networks for industrial and administration actors as well as for the general public thanks to their evolution. The introduction of the 5th generation of mobile networks, 5G, brings a diversity of use cases for mobile networks and a growing number of demands with very heterogeneous needs and with strong quality of service (QoS) constraints. The development of 5G relies on new techniques such as Software Defined Networking (SDN) and Network Function Virtualisation (NFV) technologies. SDN allows the separation of control and data planes by providing centralised network management. NFV decouples network functions from the hardware that performs them through virtualisation. The use of these two technologies automates the management of the network and makes it much more flexible and adaptable to changing traffic flows and requirements. The introduction of the network slicing paradigm leads to a division of the network into multiple independent virtual networks cohabiting on the same infrastructure. This paradigm allows to meet the very heterogeneous needs of future applications. In this thesis, we focus on optimising the resource utilisation of next generation networks in order to decrease operational costs and to accept more demands. We first study the allocation of service function chains, to quickly accept requests and to meet the diverse and high-volume demands of mobile networks. We then study the feasibility of Make-Before-Break reconfiguration of requests, which allows to reconfigure without degrading the QoS. We then scale up our reconfiguration and adapt it to network slicing to be used on future 5G networks. Finally, we optimise the reconfiguration periods by implementing a reinforcement learning algorithm, minimising the management costs.

Keywords: SDN, NFV, SFC, 5G, Slicing, Reconfiguration.