HAL
open science

Be Scalable and Rescue My Slices During Reconfiguration

Adrien Gausseran, Frederic Giroire, Brigitte Jaumard, Joanna Moulierac

To cite this version:

Adrien Gausseran, Frederic Giroire, Brigitte Jaumard, Joanna Moulierac. Be Scalable and Rescue My Slices During Reconfiguration. The Computer Journal, 2021, 64 (10), pp.1584-1599. 10.1093/comjnl/bxab108 . hal-03430584

HAL Id: hal-03430584

https://hal.science/hal-03430584v1

Submitted on 16 Nov 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Be Scalable and Rescue My Slices During Reconfiguration

Adrien Gausseran[1], Frederic Giroire[1], Brigitte Jaumard[2], and Joanna Moulierac[1]

[1]Université Côte d'Azur, I3S, CNRS, Inria, University of Nice Sophia Antipolis, France
[2]Concordia University, Canada

Abstract

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 Virtualization, 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. We propose new models and scalable algorithms (relying on column generation techniques) that solve large data instances in few seconds.

1 Introduction

The Network Function Virtualization (NFV) paradigm is a major technology of 5G networks. Over the past decade, it has been widely deployed and a large number of studies investigated its use and benefits. Its core principle is to break the dependence on dedicated hardware like traditional expensive middleboxes by allowing network functions (e.g., firewall, load balancing, Virtual Private Network (VPN) gateways, content filtering) to be virtualized and

implemented in software, and executed on generic servers. Virtual Network Functions (VNFs) can be instantiated and scaled on demand without the need to install new equipment, increasing flexibility with user demands [1]. In parallel, we also saw the emergence of Software-Defined Networking (SDN) that simplifies network monitoring and management. By decoupling the control plane from the data plane and abstracting network intelligence into a central controller, SDN allows a global vision and control of the network [2]. Combination of SDN and NFV leads to dynamic, programmable, and flexible networks in which the network infrastructure and resources are shared among network services.

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 [3, 4]. 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 [4]. With all these key features, Network slicing will thus be a fundamental feature of 5G networks [3].

Dynamic resource allocation is one of the key challenges of network slicing. In a dynamic scenario, the network state changes continuously due to the arrival and departure of requests (noted also as flows). As the granting of new flows is done without impacting the ongoing ones, we may end up with a non-optimal provisioning, and thus with an inefficient resource usage. Therefore, network operators must adjust network configurations in response to changing network conditions to fully exploit the benefits of the SDN and NFV paradigms, and to minimize the operational cost (e.g., software licenses, energy consumption, and Service Level Agreement (SLA) violations).

We here 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. Rerouting demands and migrating VNFs take several steps. Usually, network carriers/operators cannot afford traffic disruption, due to their SLAs, as this can have a significant impact on the Quality of Service (QoS) experienced by users. Their strategy is then to perform the reconfiguration by using a two–phase approach. First, a new route is established while keeping the initial one enabled (i.e., two redundant data streams are both active in parallel). Then, the transmission is done only on the new route and the resources used by the initial one are released. This strategy is often referred to as *make-before-break*. In

this work, 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.

Our contributions in this paper 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.

2 Related Work

In the last years, a large corpus of works has studied the deployment and management of network services, see [5] and [6] for surveys. 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 optimization models have been proposed in order to minimize setup cost [7, 8] or take into account the chaining constraints [9, 10]. 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 [11], it has been proposed to carry out reconfigurations of network and virtual resources regularly in order to bring the network closer to an optimal state of operation. The goals can be diverse: optimizing 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.* [12]. The latter formulate an ILP and a column generation model in order to jointly optimize 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. Noghani *et al.* [13] study the trade-off between the reconfiguration of SFCs and the optimality of the reconfigured routing and placement solution.

Gao and Rouskas [14] considered the reconfiguration of virtual networks. They proposed online algorithms to minimize the maximum utilization of substrate nodes and links while bounding the number of virtual nodes that have to be migrated.

Ayoubi *et al.* [15] propose an availability-aware resource allocation and reconfiguration framework for elastic services in failure-prone data center networks. Their work is limited to the case of Virtual Network scale-up requests such as resource demands increase, new network components arrival, and/or service class upgrade. The goal is to provide the highest availability improvement minimizing the overall reconfiguration cost which reflects the amount of resources as well as any service disruption/downtime.

Ghaznavi *et al.* [16] propose a consolidation algorithm that optimizes the placement of the VNFs in response to on-demand workload. The algorithm decides the VNF Instances to be migrated on the basis of the reconfiguration costs implied by the migration. However, they assume only one type of VNF and do not consider chaining requirements.

In [17], Eramo *et al.* study the problem of migrating VNFs in the dynamic scenario. The considered objective is to minimize the network operation cost which is the sum of the energy consumption costs and the revenue loss due to the bit loss occurring during the downtime. However, their model does not consider the bandwidth resources.

Recently, the problem has been studied for network slices. Wang *et al.* [18] propose a hybrid slice reconfiguration mechanism. The goal of the authors is to optimize the profit of a network slice provider, i.e., the total utility gained by serving slices minus the resource consumption and reconfiguration cost. The reconfiguration overhead of a slice includes two aspects: service interruption and reconfiguration resource cost.

Similarly, all works on reconfiguration of virtual resources (virtual net-

$G = (V, L)$	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 1: Notations

works, 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 [19]. However, we are the first to propose a scalable decomposition model based on column generation to solve it.

3 Problem Statement and Notations

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., [20, 21], 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 1 summarizes the notations used throughout the paper.

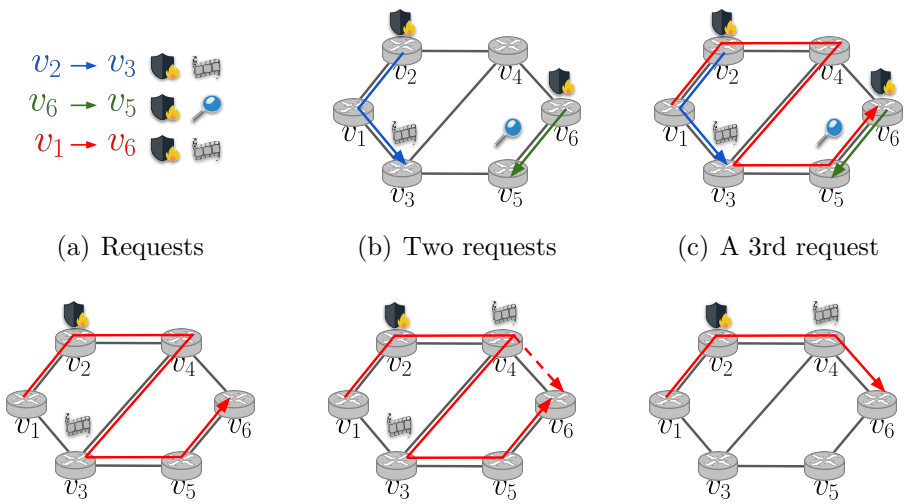
In a dynamic scenario with no information on future traffic, each demand is routed individually while *minimizing the network operational cost* defined by the weighted sum of link bandwidth and VNF usage costs (licenses, energy consumption, etc). As requests come and leave over time, allocations that 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

3.2 Example

Figure 1 illustrates an example for the reconfiguration of a request using a *make-before-break* process. Two requests, v_2 to v_3 and v_6 to v_5 are routed during step (b). Four VNFs have been installed in v_2, v_3, v_5 and v_6 to satisfy the needs of these requests. To avoid the usage cost of new VNFs, the route from v_1 to v_6 with minimum cost is a long 5–hops route (step (c)). When requests from v_2 to v_3 and from v_6 to v_5 leave, the request is routed on a non-optimal path (step (d)), which uses more resources than necessary. We compute one optimal 3-hop path and reroute the request on it (step (f)) with an intermediate make-before-break step (step (e)) in which both routes co-exist. In this example, the reconfiguration can be done in only one step of reconfiguration, but we will consider in the following up to 3 steps of reconfiguration.

3.3 Layered graph

As in [22], in order to model the chaining constraints of a demand, we associate with each demand d a layered graph $G^L(d)$. The principle is to consider as many 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 previous and next layers with an *inter-layer link*. Links within a layer corresponds to the physical network links. The use of an inter-layer link represents the use of a virtual function in the corresponding node. See Figure 2 for an example of a graph with three layers. Representing the original graph as a layered graph is a *modeling idea* first proposed in [23]. It allows a reduction of the problem to a routing problem with shared capacities. This leads to a drastic reduction in computational times compared to usual strategies using a large number of



(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 1: An example of the reconfiguration of a request using a *make-before-break* approach with one step.

binary variables due to the ordering constraints of the SFCs. Layered graphs can be used to solve different problems: *(i)* how to determine the placement and activation of NFVs, and the routing of demands; *(ii)* if the placement of NFVs has already been done, how to determine their activation and the routing of demands.

More formally, we associate with each demand d a layered graph $G^L(d)$ containing $|c_d|+1$ copies of G representing the layers of the graph. We denote by $v_{h,i}$ the copy of node v_h in layer i. As shown on Figure 2, the path for demand d from node $v_{\text{SRC}} = v_1$ to node $v_{\text{DST}} = v_3$ starts from node $v_{1,0}$ in layer 0 and ends at node $v_{3,|c_d|}$ in layer $|c_d|$. $|c_d|$ denotes the number of VNFs in the chain c_d of the demand. Given a link (v_h, v_j), each layer i has a link $(v_{h,i}, v_{j,i})$ defined. A node may be enabled to run only a subset of the virtual functions. To model this latter constraint, given a demand d we add a link $(v_{h,i}, v_{h,i+1})$ only if node v_h is enabled to run the $(i+1)-th$ function of the chain of d. A path on the layered graph corresponds to an assignment of a demand to both a path and the locations where functions are being run. Using a link $\ell = (v_{h,i}, v_{j,i})$ on G^L, implies using link $\ell = (v_h, v_j)$ on G. In addition, using link $(v_{h,i}, v_{h,i+1})$ implies using the $(i+1)-th$ function of the chain at node v_h. Capacities of both nodes and links are shared among layers.

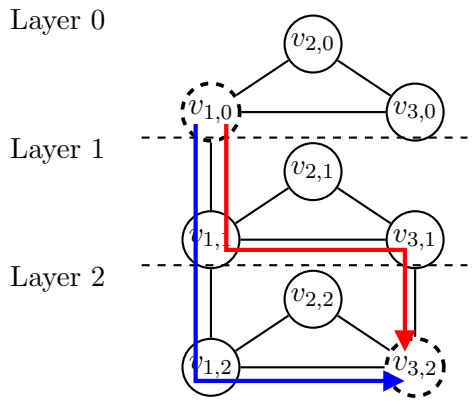


Figure 2: The layered network $G^L(d)$ associated with demand d such that $v_{\text{SRC}} = v_1$, $v_{\text{DST}} = v_3$, and $c_d = f_0^{c_d}, f_1^{c_d}$, within a triangle network. $f_0^{c_d}$ can be installed on v_1 and $f_1^{c_d}$ on v_1 and v_3. Two possible paths that satisfy d are drawn in red ($f_0^{c_d}$ in v_1 and $f_1^{c_d}$ in v_3) and blue (both functions are in v_1).

4 Optimization models

This section described first the compact ILP model (`slow-rescue`) to solve our problem, and then, the two models (`rescue-ILP` and `rescue-LP`) based on column generation methods.

4.1 ILP Model: `slow-rescue`

The compact ILP model, `slow-rescue`, is an Integer Linear Program based on the notion of layered graph described previously.

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 optimization 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 $\alpha_{u,i}^{d,0}$ (for each node $v \in V$, layer $i \in \{0, ..., |c_d|\}$) are known.

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

$$\min \sum_{d \in D} \sum_{\ell \in L} \sum_{i=0}^{|c_d|} \mathrm{BW}_d \, \varphi_{\ell,i}^{d,T} + \beta \sum_{v \in V} \sum_{f \in F} c_{v,f} \, z_{v,f} \tag{1}$$

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 $t \in T$.

$$\sum_{\ell \in \omega^+(v)} \varphi_{\ell,0}^{d,t} - \sum_{\ell \in \omega^-(v)} \varphi_{\ell,0}^{d,t} + \alpha_{v,0}^{d,t} = \begin{cases} 1 \text{ if } v = v_{\text{SRC}} \\ 0 \text{ else} \end{cases} \tag{2}$$

$$\sum_{\ell \in \omega^+(v)} \varphi_{\ell,|c_d|}^{d,t} - \sum_{\ell \in \omega^-(v)} \varphi_{\ell,|c_d|}^{d,t} - \alpha_{v,|c_d|-1}^{d,t} = \begin{cases} -1 \text{ if } v = v_{\text{DST}} \\ 0 \text{ else} \end{cases} \tag{3}$$

$$\sum_{\ell \in \omega^+(v)} \varphi_{\ell,i}^{d,t} - \sum_{\ell \in \omega^-(v)} \varphi_{\ell,i}^{d,t} + \alpha_{v,i}^{d,t} - \alpha_{v,i-1}^{d,t} = 0 \qquad 0 < i < |c_d|. \tag{4}$$

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 $y_{v,i}^{d,t}$ 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 $\alpha_{v,i}^{d,t}$ or $\alpha_{v,i}^{d,t-1}$ should be equal to 0. If d keeps the same allocation between t and $t-1$, then $\omega^{d,t} = 1$ and $y_{v,i}^{d,t}$ is forced to 1

if node v is used and can be equal to 0 otherwise.

$$\alpha_{v,i}^{d,t} \leq y_{v,i}^{d,t} \tag{5}$$
$$\alpha_{v,i}^{d,t-1} \leq y_{v,i}^{d,t} \tag{6}$$
$$\alpha_{v,i}^{d,t} + \alpha_{v,i}^{d,t-1} - \omega^{d,t} \leq y_{v,i}^{d,t}. \tag{7}$$

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.

$$\varphi_{\ell,i}^{d,t} \leq x_{\ell,i}^{d,t} \tag{8}$$
$$\varphi_{\ell,i}^{d,t-1} \leq x_{\ell,i}^{d,t} \tag{9}$$
$$\varphi_{\ell,i}^{d,t} + \varphi_{\ell,i}^{d,t-1} - \omega^{d,t} \leq x_{\ell,i}^{d,t}. \tag{10}$$

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 $t \in T$.

$$\sum_{d \in D} \mathrm{BW}_d \sum_{i=0}^{|c_d|-1} \Delta_{f_i^{c_d}} \, y_{v,i}^{d,t} \leq C_v. \tag{11}$$

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 $\ell \in L$, $t \in T$.

$$\sum_{d \in D} \mathrm{BW}_d \sum_{i=0}^{|c_d|} x_{\ell,i}^{d,t} \leq C_\ell. \tag{12}$$

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 $d \in D$, $t \in T$

$$\sum_{i=0}^{|c_d|} x_{\ell,i}^{d,t} \, \mathrm{DELAY}_\ell \leq \mathrm{DELAY}_d. \tag{13}$$

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 $i \in \{0, ..., |c_d| - 1\}$,

$$\alpha_{v,i}^{d,T} \leq z_{v,f_i^{c_d}}. \tag{14}$$

Reconfiguration - node modification constraints. To know if the allocation of a demand d is modified on nodes between two consecutive time periods.

For $d \in D$, $v \in V$, $i \in \{0, ..., |c_d|\}$, $t \in T$.

$$\omega^{d,t} \leq 1 + \alpha_{v,i}^{d,t} - \alpha_{u,i}^{d,t-1} \tag{15}$$

$$\omega^{d,t} \leq 1 + \alpha_{v,i}^{d,t-1} - \alpha_{v,i}^{d,t}. \tag{16}$$

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

$$\omega^{d,t} \leq 1 + \varphi_{\ell,i}^{d,t} - \varphi_{\ell,i}^{d,t-1} \tag{17}$$

$$\omega^{d,t} \leq 1 + \varphi_{\ell,i}^{d,t-1} - \varphi_{\ell,i}^{d,t}. \tag{18}$$

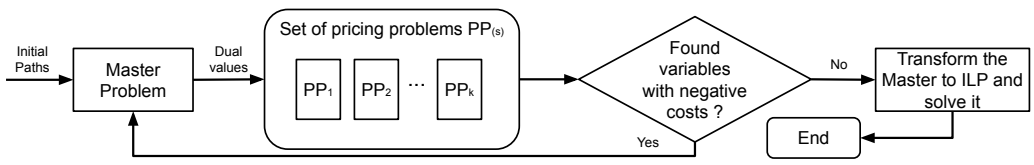


Figure 3: CG is a decomposition method dividing an optimization model into two parts: a master problem and a (set of) pricing problem(s) (PP). The restricted master problem (RMP) solves a linear relaxation of the problem with a restricted set of columns. Then the PPs compute the best columns to be added, based on prices given by the dual variables of the RMP. The RMP and PPs are then iteratively solved until no more columns can improve the solution of the RMP. Last, the original problem is solved subject to the integrality constraint using the columns of the RMP.

As we will see in Section 5, 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).

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

4.2.1 Key ideas of column generation technique

Column generation (CG) is a model allowing the solution of an optimization model without explicitly introducing all variables, see Figure 3 for an

explanation. It thus often allows the solution of larger instances of the problem than a compact model, in particular, with an exponential number of variables.

In the context of our problem, the master problem (MP) seeks a possible global reconfiguration for all slices with a path-formulation. In the restricted master problem (RMP), only a subset of potential paths is used for each slice. At the initialization, the set of paths is the one used before reconfiguration. Each pricing problem (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. For a more detailed explanation column generation techniques, see [24].

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

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

We assume an initial configuration is provided with fixed values for $\varphi_p^{d,0}$. The optimization model is written as follows.
Objective: minimize the amount of network resources consumed during the last reconfiguration time step T.

$$\min \sum_{d\in D}\sum_{p\in P_d}\sum_{\ell\in L} \mathrm{BW}_d\, \varphi_p^{d,T}\, \delta_\ell^p + \beta \sum_{V\in V^{\text{VNF}}}\sum_{f\in F} c_{v,f}\, z_{v,f} \tag{19}$$

Constraints:
One path constraint. For $d \in D$, time step $t \in T$.

$$\sum_{p\in P_d} \varphi_p^{d,t} = 1. \tag{20}$$

Path usage over two consecutive time periods. For $d \in D$, $p \in P_d$, $t \in T$.

$$\varphi_p^{d,t} \le y_p^{d,t} \text{ and } \varphi_p^{d,t} \le y_p^{d,t-1}. \tag{21}$$

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 $v \in V^{\text{VNF}}$, $t \in T$.

$$\sum_{d \in D} \sum_{p \in P_d} \sum_{i=0}^{|c_d|-1} y_p^{d,t} \cdot \theta_{i,v}^p \cdot \text{BW}_d \cdot \Delta_{f_i^{c_d}} \leq C_v. \tag{22}$$

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 $\ell \in L$, $t \in T$,

$$\sum_{d \in D} \sum_{p \in P_d} \text{BW}_d \, y_p^{d,t} \, \delta_\ell^p \leq C_\ell. \tag{23}$$

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

$$y_p^{d,T} \, \theta_{i,u}^p \leq z_{u,f_i^{c_d}}. \tag{24}$$

4.2.3 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: minimize the amount of network resources consumed for the demand d at time t.

$$\begin{aligned} \min \sum_{\ell \in L} \sum_{i=0}^{|c_d|} \varphi_{\ell,i} \, \text{BW}_d (1 + \mu_{\ell,t}^{(23)}) \\ + \text{BW}_d \sum_{v \in V^{\text{VNF}}} \mu_{v,t}^{(22)} \sum_{i=0}^{|c_d|-1} \Delta_{f_i^{c_d}} \, \alpha_{v,i} \\ - \mu_{d,t}^{(20)} + \beta \sum_{v \in V^{\text{VNF}}} \sum_{f \in F} c_{v,f} \, z_{v,f} \, \mu_{d,v,f}^{(24)}, \end{aligned} \tag{25}$$

where $\mu^{(24)}_{d,v,f} = 0$ when $t \neq T$, see constraints (24).
Constraints:
Flow conservation constraints for the demand d. For $v \in V^{\text{VNF}}$.

$$\sum_{\ell \in \omega^+(v)} \varphi_{\ell,0} - \sum_{\ell \in \omega^-(v)} \varphi_{\ell,0} + \alpha_{v,0} = \begin{cases} 1 \text{ if } v = v_{\text{SRC}} \\ 0 \text{otherwise} \end{cases} \tag{26}$$

$$\sum_{\ell \in \omega^+(v)} \varphi_{\ell,|c_d|} - \sum_{\ell \in \omega^-(v)} \varphi_{\ell,|c_d|} - \alpha_{v,|c_d|-1} = \begin{cases} -1 \text{ if } v = v_{\text{DST}} \\ 0 \text{ otherwise} \end{cases} \tag{27}$$

$$\sum_{\ell \in \omega^+(v)} \varphi_{\ell,i} - \sum_{\ell \in \omega^-(v)} \varphi_{\ell,i} + \alpha_{v,i-1} - \alpha_{v,i-1} = 0 \qquad 0 < i < |c_d|. \tag{28}$$

Delay constraints. The sum of the link delays of the flow must not exceed the delay requirement of demand d.

$$\sum_{i=0}^{|c_d|} \varphi_{\ell,i} \, \text{DELAY}_\ell \leq \text{DELAY}_d. \tag{29}$$

Function activation. To know which functions are activated on which nodes. For $v \in V^{\text{VNF}}$, $f \in F$, layer $i \in \{0, ..., |c_d| - 1\}$.

$$\alpha_{v,i} \leq z_{v,f_i^{c_d}}. \tag{30}$$

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

$$\alpha_{u,i} = 0. \tag{31}$$

4.2.4 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.

5 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, and
- `slow-rescue`, our (slower) compact ILP reconfiguring slices without interruptions.

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 Section 5.2. We discuss the impact of the number of reconfiguration steps in Section 5.3. Then, we consider dynamic scenarios in which requests arrive and leave over time in Section 5.4. We discuss the gains provided by the reconfiguration by studying the impact on several metrics while varying the reconfiguration frequency in Section 5.5. The scalability of our solutions are proven in 5.6. The gains of parallelization are shown in Section 5.7 and the impact of slice delay constraints in Section 5.8.

5.1 Data sets

Topologies. We conduct simulations on three real-world topologies from SNDlib [25] 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

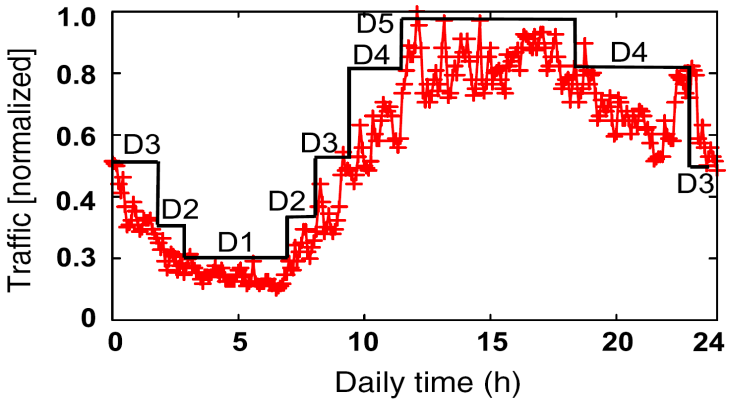


Figure 4: Period approximation of traffic variation

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 2: Characteristics of network slices

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. We use cplex [26] for the mathematical linear programming solver.

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 2 and are taken from [27]. The bandwidth usage was chosen according to the distribution of Internet traffic described in [28]. The latency requirements are expressed in milliseconds and represent the maximum delay between the source and destination. Simulations have been conducted on an Intel Xeon E5520 with 24GB of RAM.

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 (Fig. 4). 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-section 5.3.

5.2 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 Fig. 4. 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 $D5$.

5.2.1 Execution times

We report the execution times of a reconfiguration in two steps for `slow-rescue`, `rescue-ILP` and `rescue-LP` in Figure 5. 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.

5.2.2 Gains in network cost

We now compare the improvement in terms of network cost obtained after a reconfiguration in Figure 6. 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 7 and 8, respectively).

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.

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 7 and 8 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 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

	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 3: Accuracy of the column generation models (%)

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`.

5.2.3 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 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.

5.2.4 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 9 (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.

5.3 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 Figures 10 and 11. 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 10 and Table 4, whether on `rescue-LP` or `rescue-ILP`, over all periods: an increase in numbers implies an improvement in the objective. This phenomenon is 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 11 and Table 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.

5.4 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) in order to model the variation of traffic over 24 hours. In our scenario, the net-

	rescue-ILP				rescue-LP			
# Steps	1	2	3	4	1	2	3	4
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: Average percentages of improvement for each period and each number of steps for `rescue-LP` and `rescue-ILP` on `ta1`.

	rescue-ILP				rescue-LP			
# Steps	1	2	3	4	1	2	3	4
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 5: Computation times (seconds) on `ta1`

work 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 `no-reconf` 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 Section5.5, in which we vary the reconfiguration frequency, and show that 15 is a good trade-off between network management and all the studied metrics.

5.4.1 Network Cost

In Figure 12 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. 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.

5.4.2 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 13. 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,`no-reconf` 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 con-

clusion of Figures 12 and 13, `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.

5.4.3 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 2), we report the percentage of the bandwidth of the accepted slices compared to the one of the requested slices. The plot in Figure 14 represent 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. 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.

5.4.4 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 15. 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.

5.5 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 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.

5.5.1 Network Cost

We study in Figure 16 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.

5.5.2 Throughput

Figure 17 shows the network throughput over time as defined in 5.4.2. 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.

5.5.3 Accepted Slices

In Figure 18, we plot the accepted bandwidth over time as defined in 5.4.3. Each curve is more easily identifiable compared to previous figures. For reconfiguration frequency of 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.

5.5.4 Cost per MBit

Figure 19 shows the network operational cost per MBit over time as defined in 5.4.4. 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 frequency 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).

5.6 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 20, we are interested in the scalability of our solution based on our experiences in 5.2. 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 9 (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 20 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 20 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 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 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.

5.7 Parallelization of the pricing problem

One of the advantages of column generation is the ability to parallelize 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 parallelization can save time. In Figure 21 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.

5.8 Impact of the delay constraints

Being able to ensure strict delay constraints for some applications is one of the key element of network slicing [4]. As an example, each of the slice we considered had a maximum latency corresponding to its service as shown in Table 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 ms, 5 ms and 10 ms.

5.8.1 Stricter delays lead to lower improvements

The improvement of the objective due to reconfiguring is plotted in Figure 22 for the 3 different latency constraints. We observe that larger gains are obtained when the delay constraints are looser. For a 2.5 ms latency, the improvement is of 16% in average, while it is of 27% and 27.5% for 5 ms and 10 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 ms and the 10 ms are similar.

5.8.2 Stricter delays makes it harder to solve

In Figure 23 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 ms, in addition to have a lower improvement, we have much longer computation times with 428 sec on average, compared to 228 sec and 91 sec for 5 ms and 10 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.

6 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 work, 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%.

Acknowledgements. This work has been supported by the French government through the UCA JEDI [ANR-15-IDEX-01] and EUR DS4H [ANR-17-EURE-004] Investments in the Future projects, and by Inria associated team EfDyNet.

References

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

[2] Kim, H. and Feamster, N. (2013) Improving network management with software defined networking. *IEEE Communications Magazine*, **51**, 114–119.

[3] Rost, P. et al. (2017) Network slicing to enable scalability and flexibility in 5G mobile networks. *IEEE Communications magazine*, **55**, 72–79.

[4] Bega, D., Gramaglia, M., Banchs, A., Sciancalepore, V., Samdanis, K., and Costa-Perez, X. (2017) Optimising 5G infrastructure markets: The business of network slicing. *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pp. 1–9.

[5] Herrera, J. and Botero, J. (2016) Resource allocation in NFV: A comprehensive survey. *IEEE Transactions on Network and Service Management*, **13**, 518–532.

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

[7] Kuo, T.-W., Liou, B.-H., Lin, K. C.-J., and Tsai, M.-J. (2018) Deploying chains of virtual network functions: On the relation between link and server usage. *IEEE/ACM Transactions on Networking (TON)*, **26**, 1562–1576.

[8] Cohen, R., Lewin-Eytan, L., Naor, J., and Raz, D. (2015) Near optimal placement of virtual network functions. *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, Kowloon, Hong-Kong, pp. 1346–1354.

[9] Huin, N., Jaumard, B., and Giroire, F. (2018) Optimal network service chain provisioning. *IEEE/ACM Transactions on Networking*, **26**, 1320–1333.

[10] Tomassilli, A., Giroire, F., Huin, N., and Pérennes, S. (2018) Provably efficient algorithms for placement of service function chains with ordering constraints. *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pp. 774–782.

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

[12] Liu, J., Lu, W., Zhou, F., Lu, P., and Zhu, Z. (2017) On dynamic service function chain deployment and readjustment. *IEEE Transactions on Network and Service Management*, **14**, 543–553.

[13] Noghani, K., Kassler, A., and Taheri, J. (2019) On the cost-optimality trade-off for service function chain reconfiguration. *IEEE International Conference on Cloud Networking (CloudNet)*, pp. 1–6.

[14] Gao, L. and Rouskas, G. N. (2018) Virtual network reconfiguration with load balancing and migration cost considerations. *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, April, pp. 2303–2311.

[15] Ayoubi, S., Zhang, Y., and Assi, C. (2016) A reliable embedding framework for elastic virtualized services in the cloud. *IEEE TNSM*, **13**, 489–503.

[16] Ghaznavi, M. et al. (2015) Elastic virtual network function placement. *IEEE CloudNet*, pp. 255–260. IEEE.

[17] Eramo, V. et al. (2017) An approach for SFC routing and virtual function network instance migration in NFV architectures. *IEEE/ACM Transaction in Networking*, **25**, 2008–2025.

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

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

[20] Leconte, M., Paschos, G., Mertikopoulos, P., and Kozat, U. (2018) A resource allocation framework for network slicing. *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pp. 2177–2185.

[21] Pozza, M., Patel, A., Rao, A., Flinck, H., and Tarkoma, S. (2019) Composing 5G network slices by co-locating VNFs in μslices. *IFIP Networking Conference*, May, pp. 1–9.

[22] Huin, N., Jaumard, B., and Giroire, F. (2018) Optimal network service chain provisioning. *IEEE/ACM Transactions on Networking (ToN)*, **26**, 1320–1333.

[23] Dwaraki, A. and Wolf, T. (2016) Adaptive service-chain routing for virtual network functions in software-defined networks. *Proceedings of the 2016 workshop on Hot topics in Middleboxes and Network Function Virtualization*, pp. 32–37.

[24] Desaulniers, G., Desrosiers, J., and Solomon, M. (2005) *Column Generation* Cahiers du GERAD. Springer US.

[25] Orlowski, S., Wessäly, R., Pióro, M., and Tomaszewski, A. (2010) SNDlib 1.0–survivable network design library. *Wiley Networks*, **55**, 276–286.

[26] Cplex, I. I. (2009) V12. 1: User's manual for cplex. *International Business Machines Corporation*, **46**, 157.

[27] Savi, M., Tornatore, M., and Verticale, G. (2015) Impact of processing costs on service chain placement in network functions virtualization. *IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, pp. 191–197.

[28] CISCO (2015) *Cisco Visual Networking Index: Forecast and Methodology, 2014–2019.*

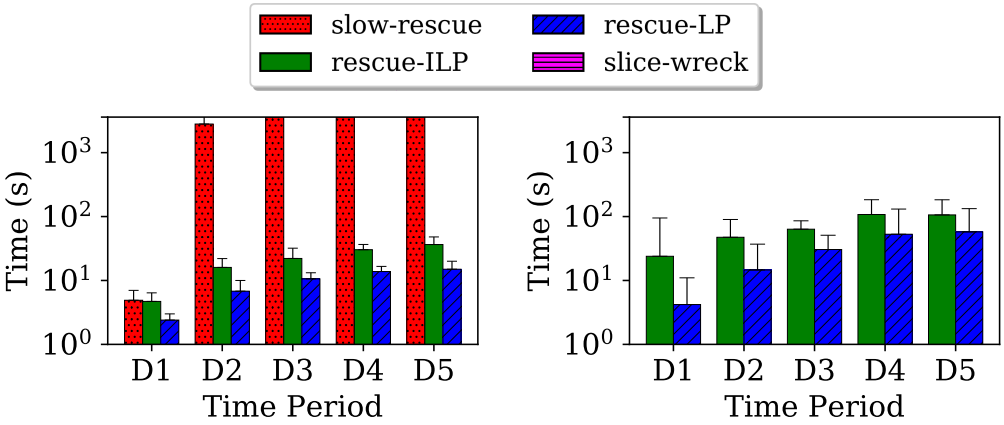


Figure 5: Execution times for `pdh` (left) and for `ta1` (right).

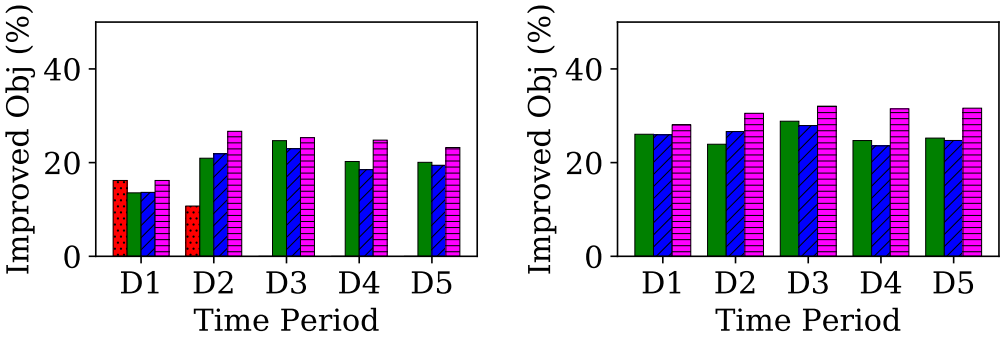


Figure 6: Gains in network cost for `pdh` (left) and for `ta1` (right).

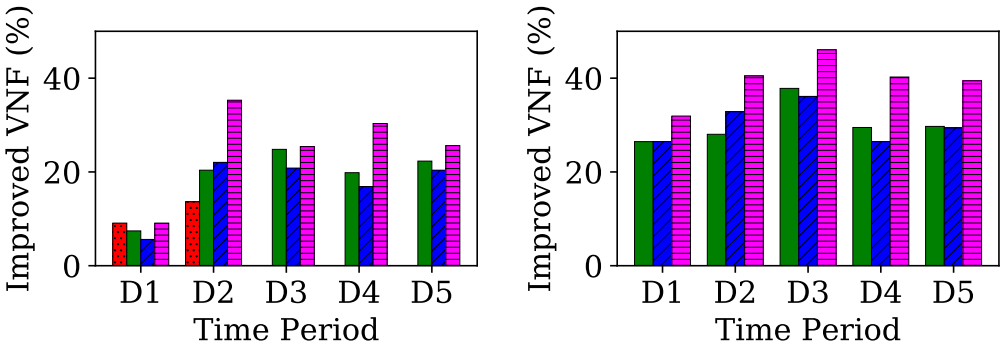


Figure 7: Gains in VNF cost for `pdh` (left) and for `ta1` (right).

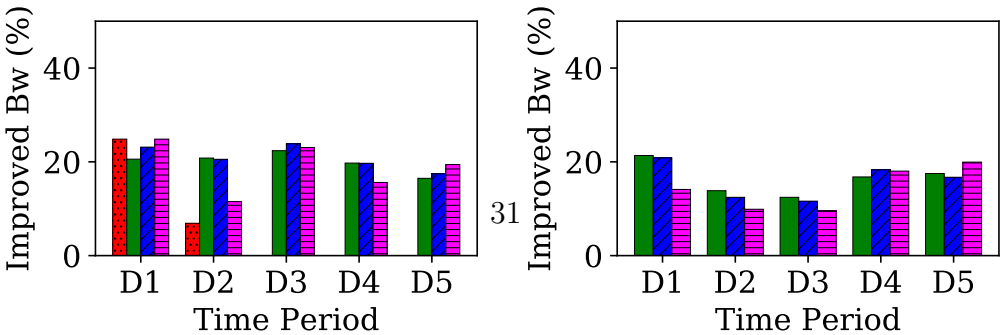
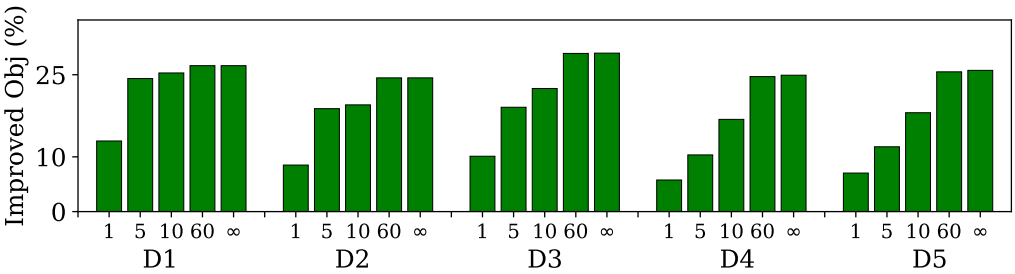
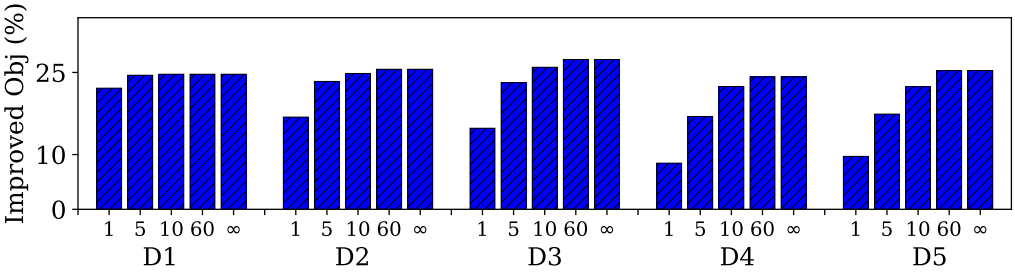


Figure 8: Gains in bandwidth cost for `pdh` (left) and for `ta1` (right).



(a) `rescue-ILP`



(b) `rescue-LP`

Figure 9: Improvement due to the reconfiguration for different model time limits on `ta1`.

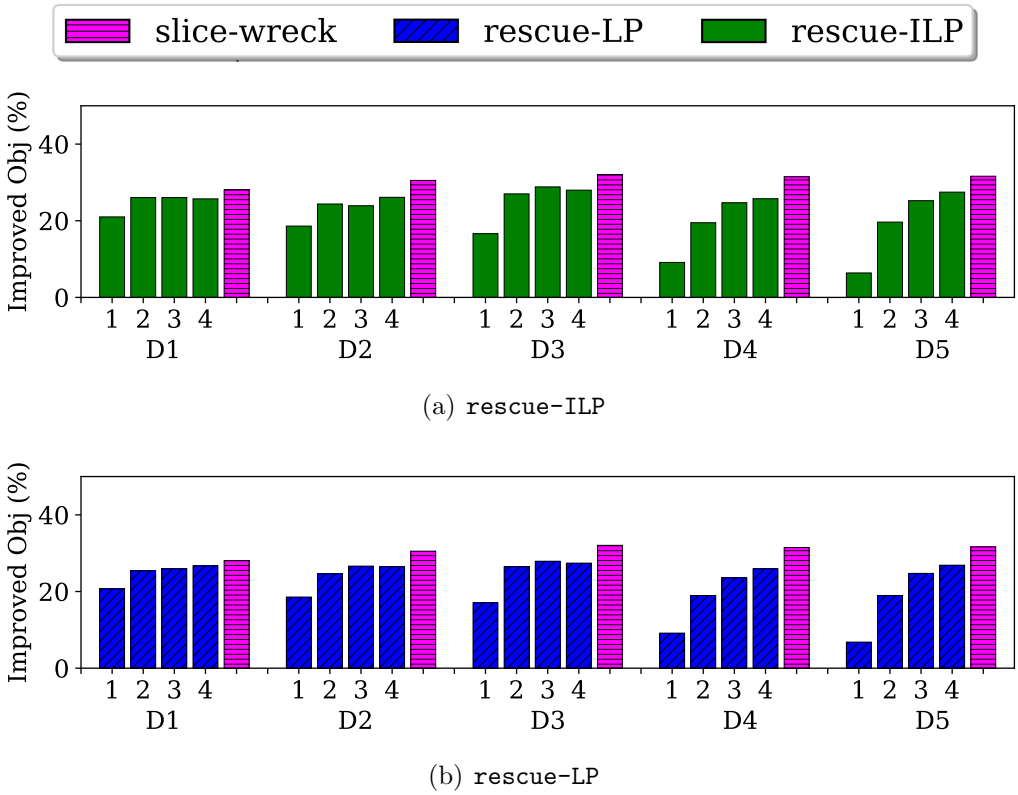
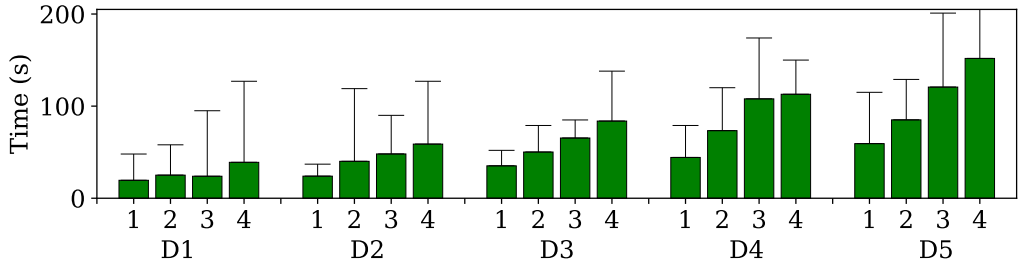
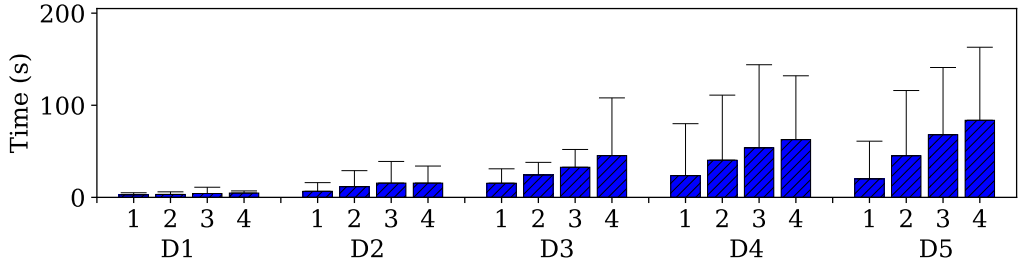


Figure 10: Improvement of the Objective (in %) with different numbers of reconfiguration steps on `ta1`



(a) `rescue-ILP`



(b) `rescue-LP`

Figure 11: Reconfiguration time with different numbers of reconfiguration steps on `ta1`

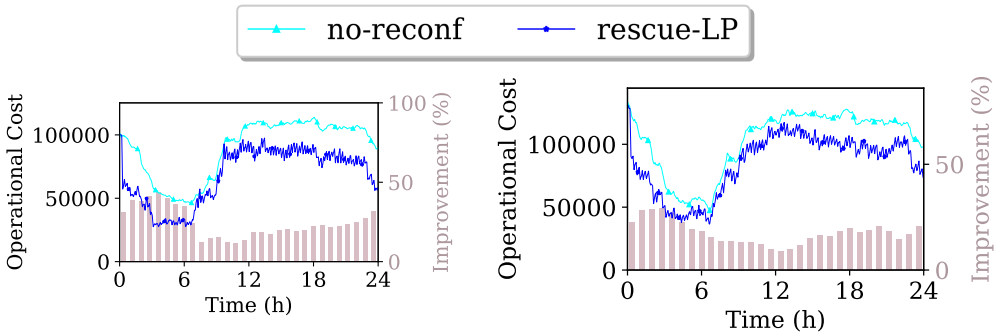


Figure 12: Network cost for `ta1` (left) and for `ta2` (right).

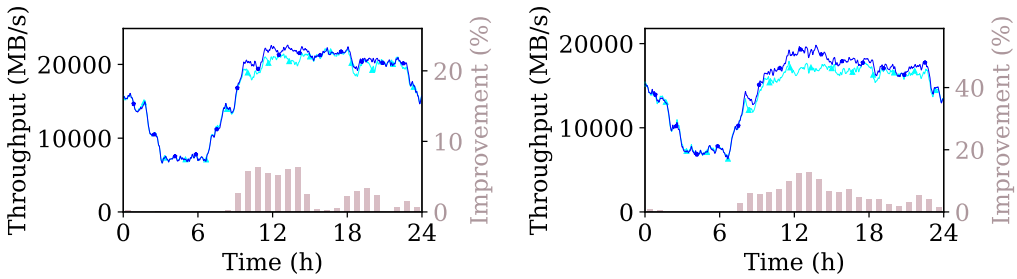


Figure 13: Throughput for `ta1` (left) and for `ta2` (right).

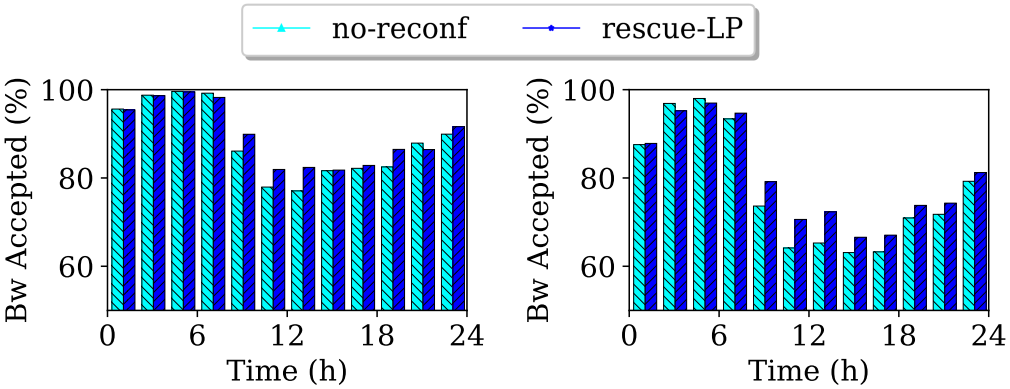


Figure 14: Percentage of Bandwidth accepted for `ta1` (left) and for `ta2` (right).

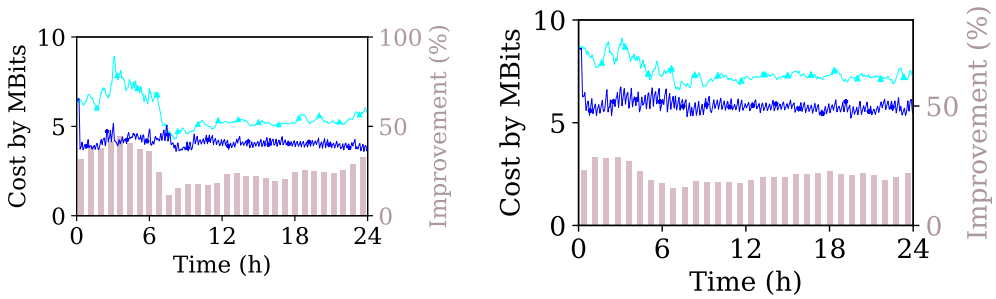


Figure 15: Network cost per accepted bandwidth for `ta1` (left) and for `ta2` (right).

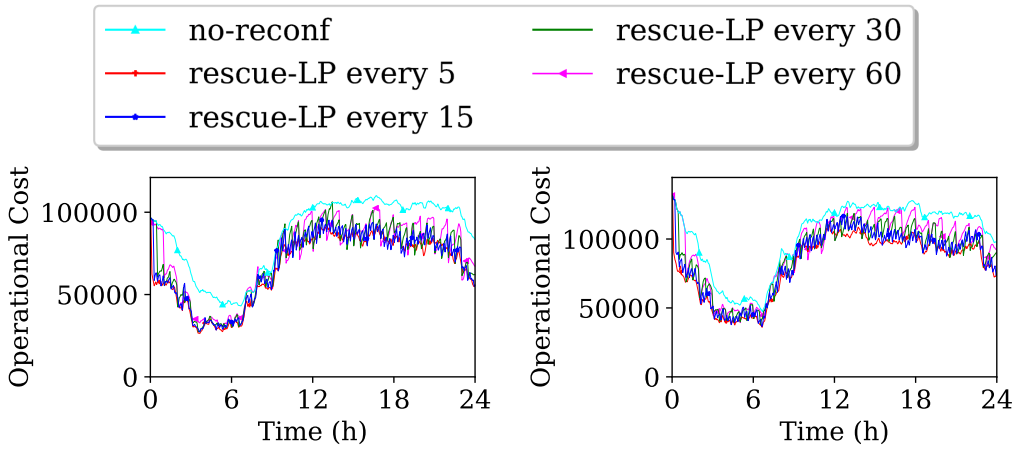


Figure 16: Network cost for `ta1` (left) and for `ta2` (right).

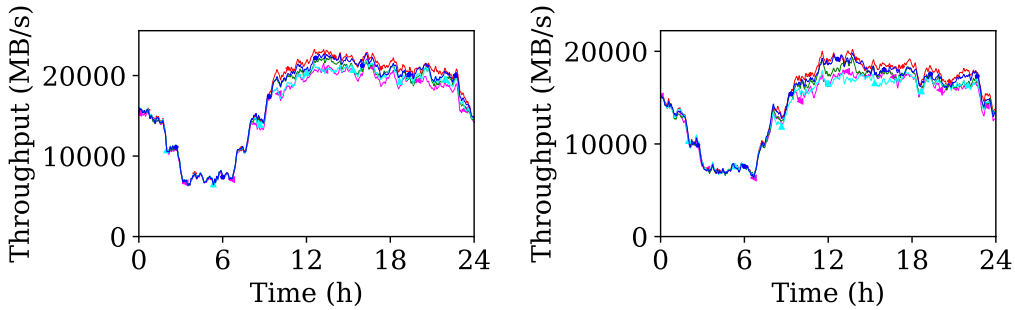


Figure 17: Throughput for `ta1` (left) and for `ta2` (right).

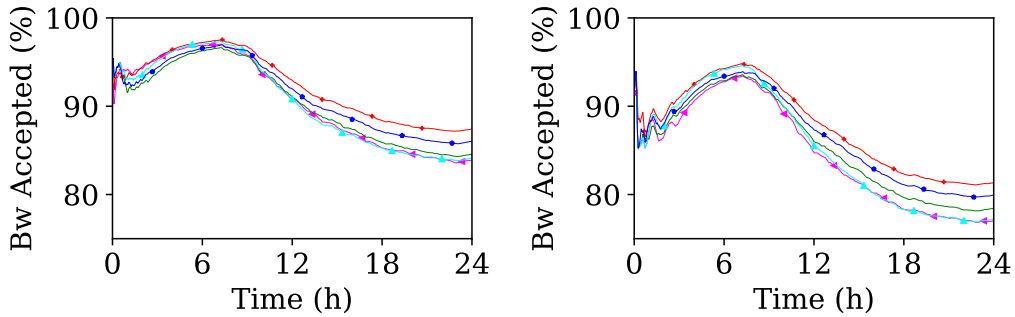


Figure 18: Percentage of Bandwidth accepted for `ta1` (left) and for `ta2` (right).

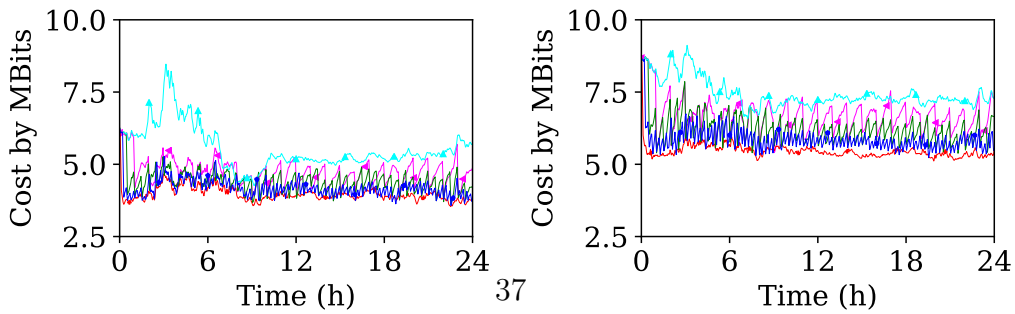


Figure 19: Network cost per accepted bandwidth for `ta1` (left) and for `ta2` (right).

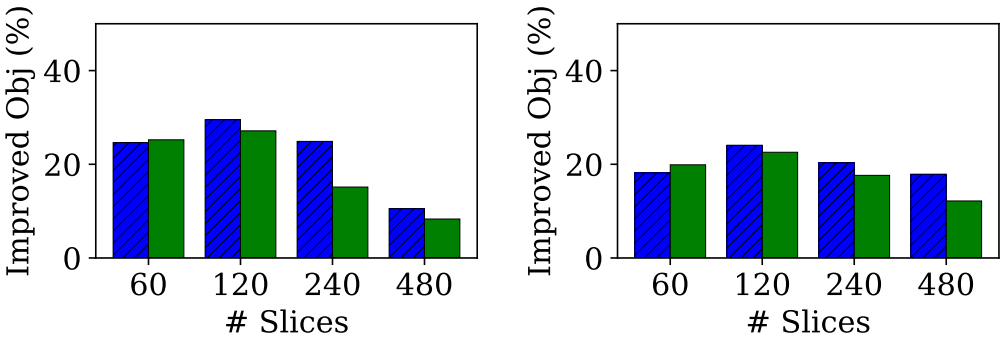


Figure 20: Gains in network cost for `ta1` (left) and for `ta2` (right) with different numbers of slices during `D5` period.

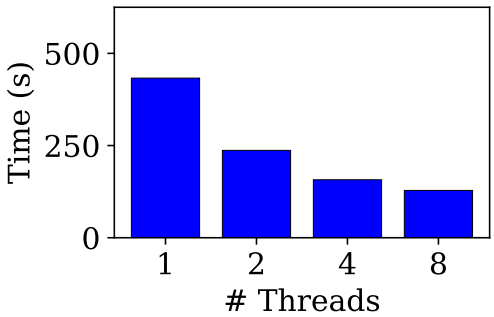


Figure 21: Time to execute the pricing problems according to the number of threads on `ta2` in D5 period (60 slices).

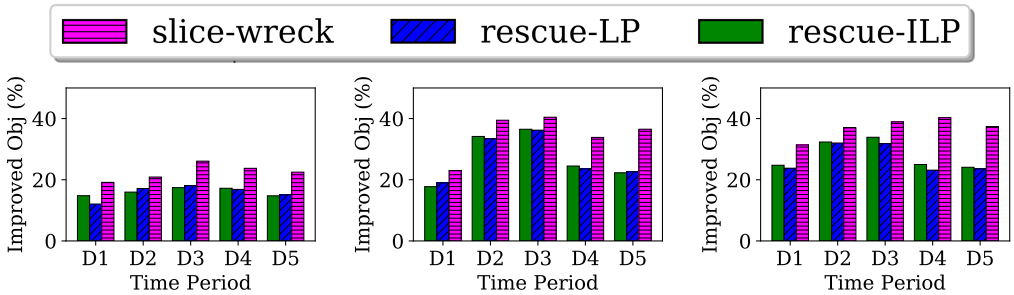


Figure 22: Improved Objective with 2.5ms delay (left), 5ms delay (middle) and 10ms delay (right) reconfiguration on `ta2`.

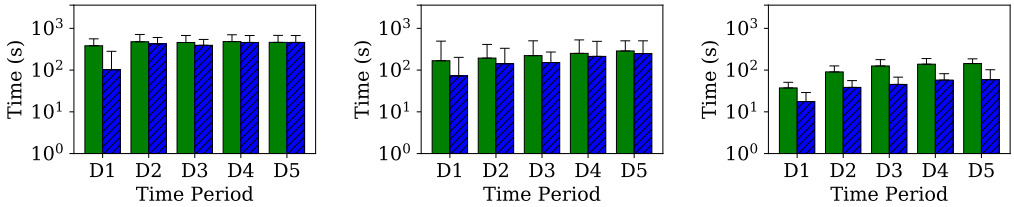


Figure 23: Reconfiguration time with 2.5ms delay (left), 5ms delay (middle) and 10ms delay (right) reconfiguration on `ta2`.