Don't Interrupt Me When You Reconfigure my Service Function Chains

Adrien Gausseran, Andrea Tomassilli, Frederic Giroire, Joanna Moulierac

Don't Interrupt Me When You Reconfigure my Service Function Chains

Adrien Gausseran, Andrea Tomassilli, Frederic Giroire, Joanna Moulierac
*

November 16, 2021

Abstract

Software Defined Networking (SDN) and Network Function Virtualization (NFV) are complementary and core components of modernized networks. In this paper, 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 optimization 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.

Keywords: Software Defined Networks, Network Function Virtualization, Service Function Chaining, Network reconfiguration, optimization

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, Software Defined Networking (SDN) and Network Function Virtualization (NFV) are two of the most promising technologies for the Next-Generation Network.

SDN aims at simplifying network management by decoupling the control plane from the data plane. Network intelligence is logically centralized in an SDN

*Université Côte d'Azur, CNRS, Inria Sophia Antipolis, I3S, UMR 7271, 06900 Sophia Antipolis, France

controller that maintains a global view of the network state. As a consequence, the network becomes programmable and can be coupled to users' business applications. Indeed, network devices do not need to be configured individually anymore as it becomes easier to manipulate them through a software program [1].

With the NFV paradigm, network functions (e.g., firewall, load balancer, content filtering or deep packet inspection) can be implemented in software and executed on generic-purpose servers located in small cloud nodes. Virtual Network Functions (VNFs) can be instantiated and scaled on–demand without the need of installing new dedicated equipment. The goal is to shift from specialized hardware appliances to commoditized hardware in order to deal with the major problems of today's enterprise middlebox infrastructure, such as cost, capacity rigidity, management complexity, and failures [2]. For network operators, NFV is an opportunity to offer services in a more agile way, capable of operating on extremely large scales, but above all, in a faster way by exploiting the intrinsic properties of virtualization. The benefits of these two emerging technologies are many. An NFV can easily adapt to changing demand. This is particularly interesting at the launch of the service where you do not have to immobilize a large investment without knowing the subsequent evolution. Virtualization offers great flexibility and elasticity for deployment; in addition to the reduction in implementation costs (CAPEX), there are also reductions in operating costs (OPEX). With NFV, scaling can be done horizontally: for example, in the event of congestion of a service, it is easy to deploy a new occurrence of this service, thus allowing it to adapt to the scalability. Finally, virtualization permits to overcome geographical constraints by positioning the service independently of physical constraints.
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 [3].

To meet customers' demands, VNFs have often to be interconnected to form complete end-to-end services. Besides, network flows are often required to be processed by an ordered sequence of network functions. For example, an Intrusion Detection System may need to inspect the packet before compression or encryption are performed. This notion is known as Service Function Chaining (SFC) [4]. SDN has the potential to make the chaining of the network functions much easier.

Routing & Provisioning 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 paper, 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 minimize 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.

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 [5], 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 minimize the network's operator cost and to optimize 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 [6], when a request is rejected [7], when the physical network is modified [8], 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*.

This article is an extended version of the conference paper published in [9] and our contributions are as follows.

- We provide the first method, `Break-Free-ILP`, to reconfigure, with a *make-before-break mechanism*, a network routing a set of demands which have to go through service function chains. This mechanism allows to reach closer to optimal resource allocation while *not interrupting the demands* which have to be rerouted.
- We propose an efficient heuristic to reconfigure the SFCs as closely as possible to a given allocation.

Compared to the original version, this article presents some new and more detailed contents. The major extensions in this article can be summarised as follows.

- Previously, we only proposed solutions based on Integer Linear Programming (ILP). Now, we also present a scalable heuristic algorithm, `Break-Free-HEUR`. The heuristic is detailed in Section 4.5 and thoroughly evaluated in Section 5 (Numerical Results). The algorithm allowed us to present results for a larger network (with 65 nodes and 108 links) than with the ILP-based methods (where the largest evaluated network had 24 nodes and 55 links).
- Sections 4.2 and 4.3 were basically missing in the conference version and only reduced to a few lines. We present here the optimization models used to solve (i) the static routing and provisioning problem (Breaking-Bad) and (ii) the problem for a single demand used when a new demand arrives dynamically.
- The example of Section 4.1 in Figure 2 was added to clarify the notion of layered graph.
- We also extended our evaluation section.
  - First, all the figures include now the results for the heuristic and for a larger network named `ta2`.
  - Second, we present new metrics, number of VNFs deployed (see Figure 3) and bandwidth usage (see Figure 4) across time, to give the readers a more informed explanation of the cost reduction achieved by our solutions.
  - We also completed the study of the impact of the parameter β, that is the relative cost of the network function installation, on the results by looking at how it influences the bandwidth usage in the network, see Figure 8.

- We added a discussion on the average reconfiguration times of the different solutions in Section 5.5, see Figure 10.
- We carried out new experiments to understand the impact of the *reconfiguration rate* on the network cost and on the accepted demands (see Section 5.6 and Figures 12 and 13).
- Last, our model does not take into account the number of transient VNFs used only for reconfiguration, so we performed additional simulations to study this overhead and added the Section 5.8.

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 Section 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 paper is organized as follows. In Section 2, we discuss related work. In Section 3, we formally state the problem addressed in this paper. Section 4 presents the optimization framework and develops the optimization models for solving the problem of routing a demand and reconfiguring the network. In Section 5, we validate our proposed optimization models by various numerical results on two real–world network topologies of different sizes. Finally, we draw our conclusion in Section 6.

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 [10] and [11] for comprehensive surveys on the relevant state of the art.

Although a lot of effort has been made to develop efficient strategies to route demands and satisfy their chaining requirements [12, 13], 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.* [14] 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 minimized. 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.

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.

In [18], Noghani *et al.* study the trade-off between the reconfiguration of SFCs and the optimality of the reconfigured routing and placement solution.

In [19], Pozza et al proposes a reconfiguration algorithm for network slices. Their algorithm takes as input the network to be reconfigured and the network in its reconfigured state. In output it returns the different reconfiguration steps. Each reconfiguration step represents the migration of a vnf as well as the rerouting of the flow towards it. Each step must be a valid routing and allocation. This study is partly similar to our heuristic solution with the difference that between each step they do not check that the capacity of the flows can coexist, thus not doing a *make-before-break* reconfiguration and thus not being sure not to degrade the QoS.

In [20], 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 minimize the number of vnf migrations.

Sharma et al [21], propose a dynamic algorithm for network slicing. Their

goal is to minimize the number of slices by removing, adding or scaling existing slices to adapt to the current traffic. Their optimization is done in two steps, the first one by solving the problem by using the average of the values to instantiate the slices. The second by giving an uncertain value to each slice and then scaling them, in order to maximize the utility of every slice placed.

Wei et al [22] proposes a slice reconfiguration algorithm in the core network exploiting Deep Reinforcement Learning to maximize 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.

In [23], Wang et al proposes two slice reconfiguration algorithms to maximize the operator's profit. A dynamic algorithm that individually reconfigures slices to scale them . A static algorithm that is occasionally used to adapt the set of slices to incoming traffic by modifying their routing. They also study resource reservation for future requests to reduce the number of reconfigurations.

The closest study to our work is from Liu *et al.* [24]. They consider the problem of optimizing 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 maximization of the service provider's profit which is the total profit from the served requests minus the total deployment cost as an optimization 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 [17]. 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*. This paper is the long version of [9].

The table 1 presents the main differences of the publications presented in this related work.

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

Reference	Type of requests	# VNFs by Request	Allocation	Objective	Reconfiguration Done	Reconfiguration Cost	Uninterrupted Flow	ILP	Heuristic	Dynamic Programming
[12]	SFC	Up to 8	Dynamic	Max Accept	-	NC	NC	-	✓	
[13]	SFC	Unknown	Static	Min Cost	-	NC	NC	✓	✓	
[14]	Flow	0	Dynamic	Min Cost	✓	✓	-	✓	✓	
[15]	VN	Up to 20	Dynamic	Max Utility	✓	✓	-	✓	✓	
[16]	VN	1	Dynamic	Min Cost	✓	✓	-	-	✓	
[17]	SFC	Up to 3	Static	Max Accept Min Cost	✓	✓	-	✓	✓	
[18]	SFC	Up to 4	Dynamic	Min Cost	✓	✓	-	✓	-	
[19]	SFC	5	-	-	✓	✓	-	-	-	
[20]	SFC	3	Dynamic	Min Cost	✓	✓	-	✓	✓	
[21]	SFC	Unknown	Dynamic	Max Slices Utility	Scaling	-	-	-	-	
[22]	SFC	2-3	-	Min Cost	✓	✓	-	✓	-	
[23]	SFC	Unknown	Dynamic	Max Utility and Profit	✓	✓	-	-	-	
[24]	SFC	Up to 5	Dynamic	Max Profit	✓	-	-	✓	-	
Our Work	SFC	Up to 4	Dynamic	Min Cost	✓	-	✓	✓	✓	

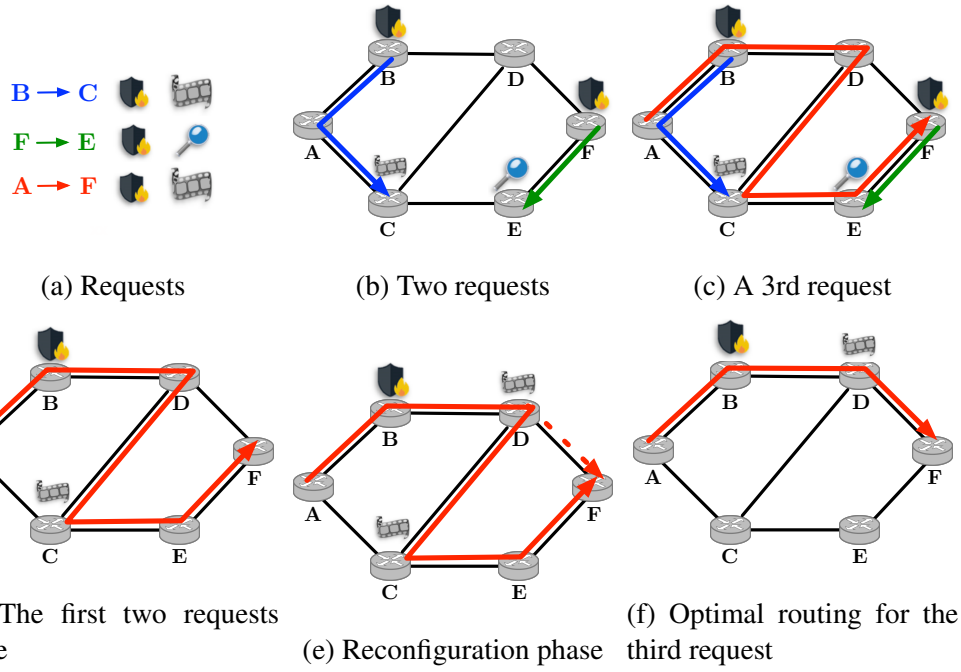
Table 1: Related Work Summary where NC means Not Concerned, '-' means that the paper does not deal with this parameter, and ✓means that this parameter is addressed in the paper.

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 defines the notation used throughout the paper. We consider a setting with splittable flows as it is frequent to have load balancing in networks [25] and as it makes the model quicker to solve [26]. Following the model of [27], a demand can follow different paths and the network functions of its chain can be processed in different cloud nodes.
The optimization task consists in routing each demand while *minimizing 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-optimization process.

In practice, 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. (iv) Last, the flows are no more sent to their old routes.
An Example. Figure 1 illustrates an example for the reconfiguration of a request using a *make-before-break* process. 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.

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 minimizes the *additional network operational cost* to be paid. Indeed, in an SDN network, even if multiple



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

(d) The 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 the *make-before-break* procedure.

flows arrive simultaneously, they will be processed one by one by the SDN controller [28]. 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.

We present here three optimization models: (i) to solve the static problem (Section 4.2) used by `Breaking-Bad`, (ii) to route one demand when it arrives (Section 4.3), and (iii) to reconfigure the network with the make-before-break mechanism of `Break-Free-ILP` (Section 4.4). Our models are based on the concept of a layered graph explained in the following section. 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 Section 4.5.

4.1 Layered graph

Similarly as in [29], 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 for an example of a graph with three layers. Representing the original graph as a layered graph is a *modeling trick* first proposed in [30]. 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 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 will be 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.

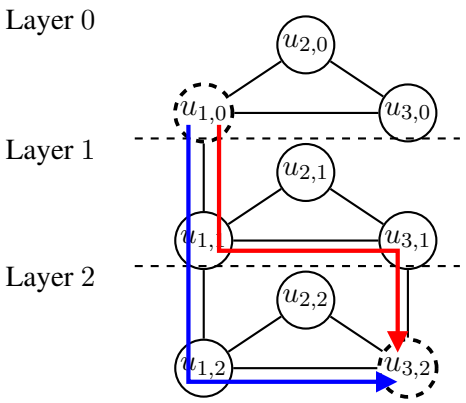


Figure 2: The layered network $G^L(d)$ associated with a demand d such that $v_s = u_1$, $v_d = u_3$, and $c_d = f_1, f_2$, within a triangle network. f_1 is allowed be installed on u_1 and f_2 on u_1 and u_3. Source and destination nodes of $G^L(d)$ are $u_{1,0}$ and $u_{3,2}$. Two possible SFCs that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

4.2 Static Routing and provisioning problem (R&P): `Breaking-Bad`

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 an ILP given below. The 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 minimization 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 will try 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. The 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.

[1]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.

$G = (V, E)$	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[1] 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 2: Notation used throughout the paper

• $\alpha^d_{u,i} \geq 0$ 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: minimize the amount of network resources consumed.

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

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

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

$$\sum_{(u,v) \in \omega^+(v)} \varphi^d_{uv,|c_d|} \; - \sum_{(v,u) \in \omega^-(v)} \varphi^d_{vu,|c_d|} - \alpha^d_{u,|c_d|-1} = \begin{cases} -1 \text{ if } v = v_d \\ 0 \text{ else} \end{cases} \tag{2}$$

$$\sum_{(u,v) \in \omega^+(u)} \varphi^d_{uv,i} \; - \sum_{(v,u) \in \omega^-(u)} \varphi^d_{vu,i} + \alpha^d_{u,i} - \alpha^d_{u,i-1} = 0. (0 < i < |c_d|) \tag{3}$$

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

$$\sum_{d \in D} bw_d \sum_{i=0}^{|c_d|-1} \Delta_{f_i^{c_d}} \cdot \alpha^d_{u,i} \leq C_u. \tag{4}$$

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 $(u, v) \in E$.

$$\sum_{d \in D} bw_d \sum_{i=0}^{|c_d|} \varphi^d_{uv,i} \leq C_{uv}. \tag{5}$$

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

$$\alpha^d_{u,i} \leq z_{u,f^{c_d}_i} \tag{6}$$

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.

$$\alpha^d_{u,i} = 0 \tag{7}$$

4.3 R&P for a single demand

Note first, that even routing a single demand is NP-hard as it is equivalent to find a shortest Weight-Constrained Path [31] in the layered graph as link and node capacities are shared between layers [29]. 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 minimizing 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.
Model. The 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).
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: minimize the additional increase in terms of network operational cost.

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

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

$$\sum_{(u,v)\in\omega^+(u)} \varphi_{uv,0} - \sum_{(v,u)\in\omega^-(u)} \varphi_{vu,0} + \alpha_{u,0} = \begin{cases} 1 \text{ if } u = v_s \\ 0 \text{ else} \end{cases} \tag{8}$$

$$\sum_{(u,v)\in\omega^+(v)} \varphi_{uv,|c_d|} - \sum_{(v,u)\in\omega^-(v)} \varphi_{vu,|c_d|} - \alpha_{u,|c_d|-1} = \begin{cases} -1 \text{ if } v = v_d \\ 0 \text{ else} \end{cases} \tag{9}$$

$$\sum_{(u,v)\in\omega^+(u)} \varphi_{uv,i} - \sum_{(v,u)\in\omega^-(u)} \varphi_{vu,i} + \alpha_{u,i} - \alpha_{u,i-1} = 0. \quad 0 < i < |c_d| \tag{10}$$

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 $u \in V$.

$$bw_d \sum_{i=0}^{|c_d|-1} \Delta_{f_i^{c_d}} \cdot \alpha_{u,i} \leq R_u. \tag{11}$$

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 $(u, v) \in E$.

$$bw_d \sum_{i=0}^{|c_d|} \varphi_{uv,i} \leq R_{uv}. \tag{12}$$

Another possibility 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 [32].

4.4 `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 Section 4.2. 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 optimization 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.

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 minimize 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 Section 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 Section 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 optimization 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 4.1 and it is written as follows.
Objective: minimize the amount of network resources consumed during the last

reconfiguration time step T.

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

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

$$\sum_{(u,v) \in \omega^+(u)} \varphi_{uv,0}^{d,t} - \sum_{(v,u) \in \omega^-(u)} \varphi_{vu,0}^{d,t} + \alpha_{u,0}^{d,t} = \begin{cases} 1 \text{ if } u = v_s \\ 0 \text{ else} \end{cases} \tag{13}$$

$$\sum_{(u,v) \in \omega^+(v)} \varphi_{uv,|c_d|}^{d,t} - \sum_{(v,u) \in \omega^-(v)} \varphi_{vu,|c_d|}^{d,t} - \alpha_{u,|c_d|-1}^{d,t} = \begin{cases} -1 \text{ if } v = v_d \\ 0 \text{ else} \end{cases} \tag{14}$$

$$\sum_{(u,v) \in \omega^+(u)} \varphi_{uv,i}^{d,t} - \sum_{(v,u) \in \omega^-(u)} \varphi_{vu,i}^{d,t} + \alpha_{u,i}^{d,t} - \alpha_{u,i-1}^{d,t} = 0. \qquad 0 < i < |c_d| \tag{15}$$

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 $t \in \{1, ..., T\}$.

$$\sum_{d \in D} bw_d \sum_{i=0}^{|c_d|-1} \Delta_{f_i^{c_d}} \cdot y_{u,i}^{d,t} \leq C_u. \tag{16}$$

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 $t \in \{1, ..., T\}$.

$$\sum_{d \in D} bw_d \sum_{i=0}^{|c_d|} x_{uv,i}^{d,t} \leq C_{uv}. \tag{17}$$

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 $t \in \{1, ..., T\}$.

$$\alpha_{u,i}^{d,t} = 0 \tag{18}$$

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

$$\alpha_{u,i}^{d,T} \leq z_{u,f_i^{c_d}}^{T}. \tag{19}$$

Note that we do not consider the cost of potential activations of VNFs during the reconfiguration process. Indeed, our goal is to minimize the network operational cost over time and the reconfiguration duration is very small in comparison in an SDN network [33].

4.5 Heuristic `Break-Free-HEUR`

As shown by the numerical evaluations in Section 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 1, 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 1 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 2). The difficulty derives from the fact that we consider *splittable* flows and that only *part of these flows* can be rerouted by the procedure.

Algorithm 1: `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);
```

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 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 2) 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.

Algorithm 2: 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;
```

The procedure `reconfSFC` (Algorithm 2) 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 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 3: 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* $\leftarrow$ 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 4). 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

Algorithm 4: 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$^{\top}$.

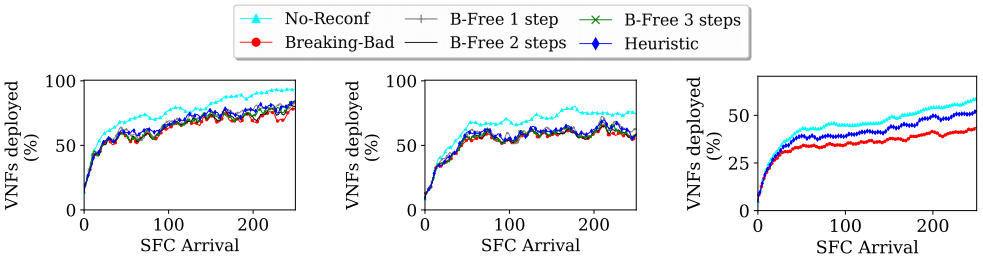
1 $E_p \leftarrow \emptyset$ ▷ #$E_p = \{(e,i) : (e,i) \in path^{\top} and (e,i) \in path^{\perp}\}$;
2 **for** $(e,i) \in path^{\top}$ **do**
3 **if** $(e,i) \in path^{\perp}$ **then**
4 E_p.append((e,i));
5 **for** e *in path*$^{\top}$ **do**
6 nbPassages $\leftarrow 0$;
7 **for** (e,i) *in path*$^{\top}$ **do**
8 **if** $(e,i) \notin E_p$ **then**
9 nbPassages $\leftarrow$ nbPassages$+1$;
10 $f_e^* = \frac{C_e}{nbPassages}$;
11 $f^* = \min_{e \in E} f_e^*$;
12 $flow = \min(flow^{\perp}, f^*, \min_{e \in path} \varphi_e^{\top})$;
13 return *flow*;

of the residual 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 1 (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.

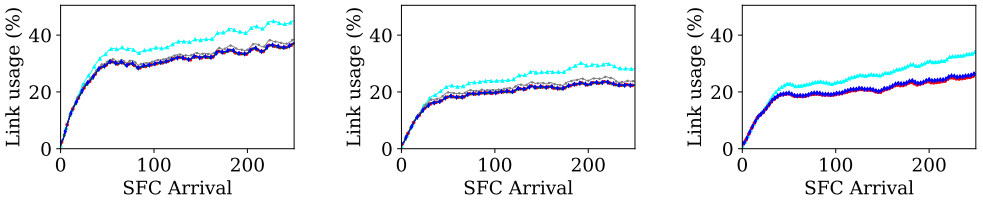
5 Numerical Results

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 Section 4.2) for each SFC arrival, and with `No-Reconf`, which computes the R&P problem for a single demand, the newly arrived SFC (Section 4.3).
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



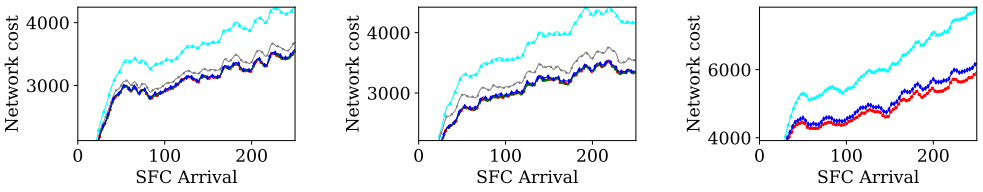
(a) `pdh` (b) `ta1` (c) `ta2`

Figure 3: Low-Traffic scenario - Number of VNFs deployed across time.



(a) `pdh` (b) `ta1` (c) `ta2`

Figure 4: Low-Traffic scenario - Bandwidth usage across time.



(a) `pdh` (b) `ta1` (c) `ta2`

Figure 5: Low-Traffic scenario - Network operational cost.

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.
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.

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: Three-real world topologies

5.1 Data sets

We conduct experiments on three real-world topologies from SNDlib [34] of different sizes: `pdh` (11 nodes, 34 links), `ta1` (24 nodes, 55 links), and `ta2` (65 nodes, 108 links). The table 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 [35], 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.

5.2 Low-traffic scenario - Resource usage

In Figure 5, we show the network cost for the *low-traffic scenario*. This cost is the result the weighted addition of Bandwidth (Figure 4) and of VNF costs (Figure 3). 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.

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 5(a)) while using 7% fewer VNFs (Figure 3(a)) and 18% less link bandwidth (Figure 4(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.

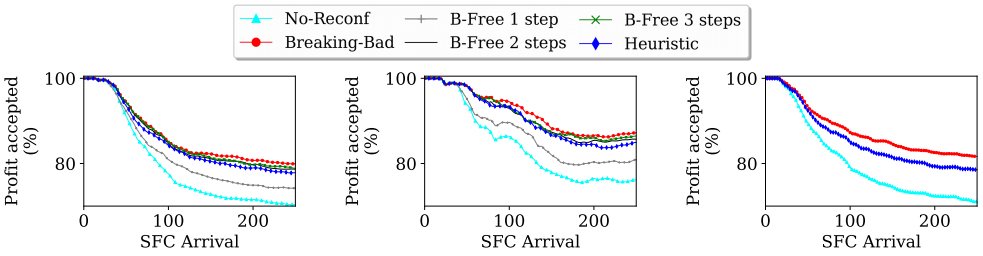
For ta2 and unlike pdh and ta1, Break-Free-HEUR deployed two times more VNFs than Breaking-Bad (Figure 3(c)). But, in the same time, Break-Free-HEUR reduces drastically the bandwidth usage (Figure 4(c)), leading in the end to a good improvement of network cost (Figure 5(c)).

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.

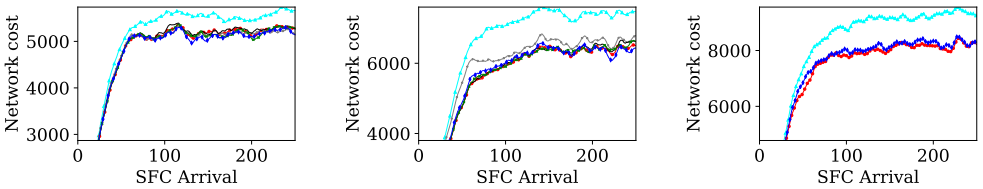
5.3 High-Traffic scenario - Acceptance Rate

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 6, 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.



(a) `pdh` (b) `ta1` (c) `ta2`

Figure 6: High-Traffic scenario - Percentage of accepted profit across time.



(a) `pdh` (b) `ta1` (c) `ta2`

Figure 7: High-Traffic scenario - Cost gain across time.

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 7, 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.

5.4 Low-Traffic scenario - Impact of Parameter β

In Figures 8 and 9, 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 Section 5.2 for a specific value of $\beta = 25$ (our default value) is valid in more general settings for a wide range of β.

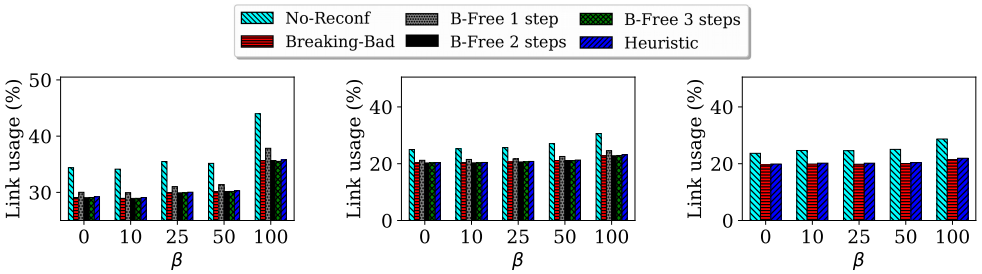
Another important observation is that the gain of reconfiguration is higher for larger values of β. The reason is that, when β is large, the requests tend to use longer routes as the cost of bandwidth is less important compared to the one of VNFs. This leads to requests routed in a very suboptimal way when other requests using the same VNFs leave (as shown in the example of Figure 1). 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 Section 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.

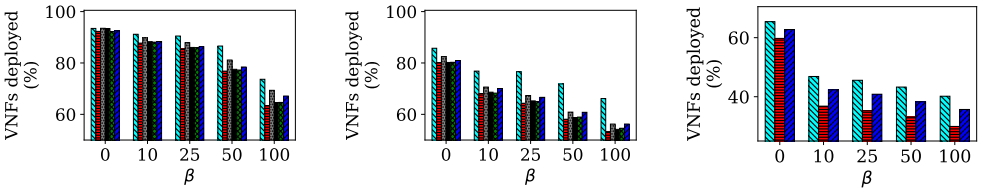
5.5 Execution Times to Compute the Reconfiguration

Figure 10 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



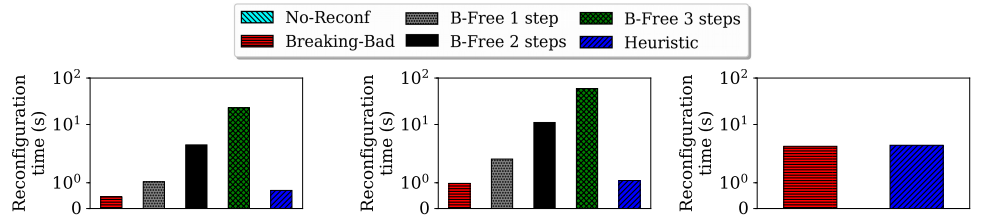
(a) `pdh` (b) `ta1` (c) `ta2`

Figure 8: Low-Traffic scenario - Impact of parameter β - Bandwidth usage as a function of β.



(a) `pdh` (b) `ta1` (c) `ta2`

Figure 9: Low-Traffic scenario - Impact of parameter β - VNFs deployed as a function of β.

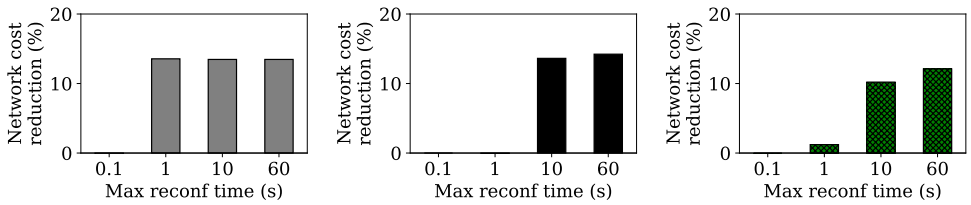


(a) `pdh` (b) `ta1` (c) `ta2`

Figure 10: High-Traffic scenario - Average reconfiguration times

`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.



(a) `B-Free` (1 step) (b) `B-Free` (2 steps) (c) `B-Free` (3 steps)

Figure 11: Low-Traffic scenario - Gains of network operational costs for different time limits for the optimization process.

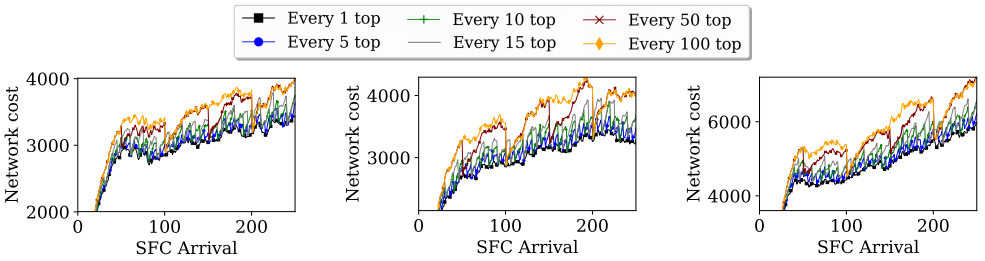
Figure 11 shows the gains of network cost (compared to `No-Reconf`) in percentage for `Break-Free-ILP` (1 to 3 steps) when limiting the time spent for the reconfiguration. `Break-Free-ILP` with 1 step needs only 1 second to reach its best solution. This variant of the algorithm is almost as fast as `Breaking-Bad` (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.

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 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.

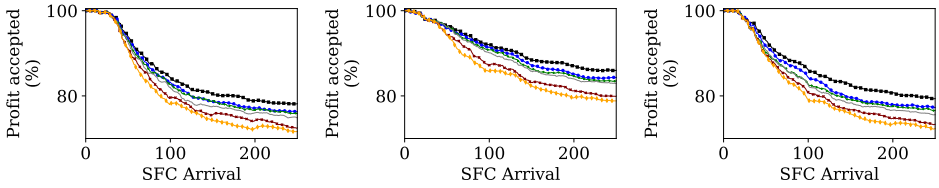
The faster rate is to reconfigure every time step, while the slowest one in our setting would be to reconfigure every 100 time steps (only 1 or 2 reconfigurations are performed during the whole test). We thus present the results for reconfiguration rates of every 1, 5, 10, 15, 50, and 100 time steps using `Break-Free-HEUR`, the results with `Breaking-Bad` and `Break-Free-ILP` are similar. In Figure 12, 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 13, in which we report the profit generated by the accepted demands. In this setting, the network



(a) pdh (b) ta1 (c) ta2

Figure 12: Low-Traffic scenario - Impact of the reconfiguration rate on the network cost.



(a) pdh (b) ta1 (c) ta2

Figure 13: High-Traffic scenario - Impact of the reconfiguration rate on the percentage of profit accepted.

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 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.

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 14 (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 average, 48% of the requests

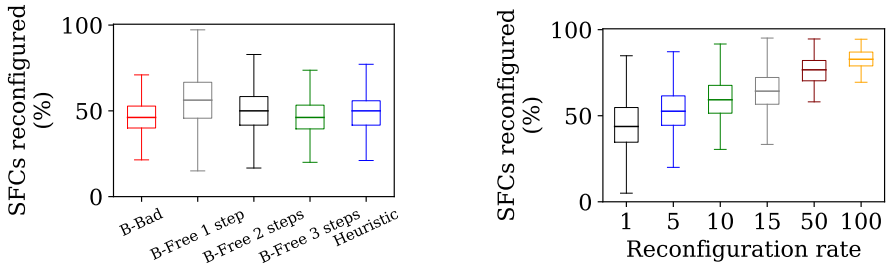
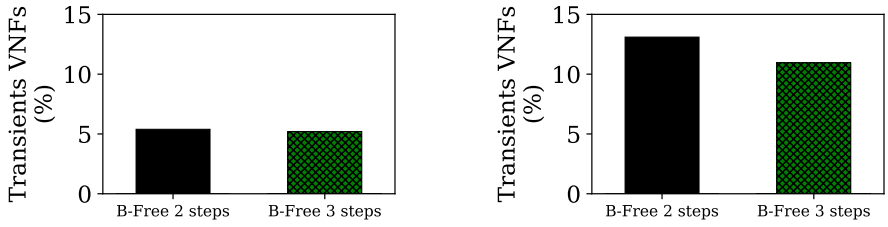


Figure 14: Percentage of rerouted requests for `ta1`, considering (left) different intermediate reconfiguration steps and (right) different reconfiguration rates.



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

Figure 15: Percentage of transient VNFs used during the intermediate steps of the reconfiguration.

(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 14 (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.

5.8 Percentage of Transient VNFs instantiated during reconfiguration

Our objective is to minimize 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 15 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 15, 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 minimize 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.

6 Conclusion

In this work, 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.

As a future work, we would like to develop more scalable optimization models able to provide guarantees on solutions for large networks.

7 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] H. Kim, N. Feamster, Improving network management with Software Defined Networking, IEEE Communications Magazine 51 (2) (2013) 114–119.

[2] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, V. Sekar, Making middleboxes someone else's problem: network processing as a cloud service, ACM SIGCOMM Computer Communication Review 42 (4) (2012) 13–24.

[3] J. Matias, J. Garay, N. Toledo, J. Unzilla, E. Jacob, Toward an SDN-enabled NFV architecture, IEEE Communications Magazine 53 (4) (2015) 187–193.

[4] P. Quinn, T. Nadeau, Problem statement for Service Function Chaining, Tech. rep. (2015).

[5] I. Fajjari, N. Aitsaadi, G. Pujolle, H. Zimmermann, VNR algorithm: A greedy approach for virtual networks reconfigurations, in: IEEE Global Telecommunications Conference (GLOBECOM), IEEE, 2011, pp. 1–6.

[6] L. Gao, G. N. Rouskas, Virtual network reconfiguration with load balancing and migration cost considerations, in: IEEE International Conference on Computer Communications (INFOCOM), IEEE, 2018, pp. 2303–2311.

[7] P. N. Tran, A. Timm-Giel, Reconfiguration of virtual network mapping considering service disruption, in: IEEE International Conference on Communications (ICC), IEEE, 2013, pp. 3487–3492.

[8] Z. Cai, F. Liu, N. Xiao, Q. Liu, Z. Wang, Virtual network embedding for evolving networks, in: IEEE Global Telecommunications Conference (GLOBECOM), IEEE, 2010, pp. 1–5.

[9] A. Gausseran, A. Tomassilli, F. Giroire, J. Moulierac, Don't Interrupt Me When You Reconfigure my SFCs, in: IEEE International Conference on Cloud Networking (CloudNet), 2019.

[10] J. G. Herrera, J. F. Botero, Resource allocation in NFV: A comprehensive survey, IEEE Transactions on Network and Service Management (IEEE TNSM) 13 (3) (2016) 518–532.

[11] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, R. Boutaba, Network function virtualization: State-of-the-art and research challenges, IEEE Communications Surveys & Tutorials 18 (1) (2016) 236–262.

[12] T.-W. Kuo, B.-H. Liou, K. C.-J. Lin, M.-J. Tsai, Deploying chains of virtual network functions: On the relation between link and server usage, IEEE/ACM Transactions on Networking (TON) 26 (4) (2018) 1562–1576.

[13] A. Tomassilli, F. Giroire, N. Huin, S. Pérennes, Provably efficient algorithms for placement of Service Function Chains with ordering constraints, in: IEEE International Conference on Computer Communications (INFOCOM), IEEE, Honolulu, Hawai, US, 2018, pp. 774–782.

[14] S. Paris, A. Destounis, L. Maggi, G. S. Paschos, J. Leguay, Controlling flow reconfigurations in SDN, in: IEEE International Conference on Computer Communications (INFOCOM), IEEE, 2016, pp. 1–9.

[15] S. Ayoubi, Y. Zhang, C. Assi, A reliable embedding framework for elastic virtualized services in the cloud, IEEE Transactions on Network and Service Management (IEEE TNSM) 13 (3) (2016) 489–503.

[16] M. Ghaznavi, A. Khan, N. Shahriar, K. Alsubhi, R. Ahmed, R. Boutaba, Elastic virtual network function placement, in: IEEE International Conference on Cloud Networking (CloudNet), 2015, pp. 255–260.

[17] V. Eramo, E. Miucci, M. Ammar, F. G. 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) (2017) 2008–2025.

[18] K. A. Noghani, A. J. Kassler, J. Taheri, On the Cost-Optimality Trade-off for Service Function Chain Reconfiguration, in: IEEE International Conference on Cloud Networking (CloudNet), 2019.

[19] M. Pozza, P. K. Nicholson, D. F. Lugones, A. Rao, H. Flinck, S. Tarkoma, On reconfiguring 5G network slices, IEEE Journal on Selected Areas in Communications 38 (7) (2020) 1542–1554.

[20] D. Harutyunyan, R. Fedrizzi, N. Shahriar, R. Boutaba, R. Riggio, Orchestrating end-to-end slices in 5G networks, in: 2019 15th International Conference on Network and Service Management (CNSM), 2019, pp. 1–9.

[21] S. Sharma, A. Gumaste, 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, pp. 1–8.

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

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

[24] J. Liu, W. Lu, F. Zhou, P. Lu, Z. Zhu, On dynamic service function chain deployment and readjustment, IEEE Transactions on Network and Service Management (IEEE TNSM) 14 (3) (2017) 543–553.

[25] B. Augustin, T. Friedman, R. Teixeira, Measuring load-balanced paths in the internet, in: ACM Internet Measurement Conference (IMC), ACM, 2007, pp. 149–160.

[26] N. Garg, J. Koenemann, Faster and simpler algorithms for multicommodity flow and other fractional packing problems, SIAM Journal on Computing 37 (2) (2007) 630–652.

[27] Y. Sang, B. Ji, G. R. Gupta, X. Du, L. Ye, Provably efficient algorithms for joint placement and allocation of virtual network functions, in: IEEE International Conference on Computer Communications (INFOCOM), IEEE, 2017, pp. 1–9.

[28] W. Ma, O. Sandoval, J. Beltran, D. Pan, N. Pissinou, Traffic aware placement of interdependent NFV middleboxes, in: IEEE International Conference on Computer Communications (INFOCOM), 2017, pp. 1–9.

[29] N. Huin, B. Jaumard, F. Giroire, Optimal network service chain provisioning, IEEE/ACM Transactions on Networking (ToN) 26 (3) (2018) 1320–1333. doi:10.1109/TNET.2018.2833815.

[30] A. Dwaraki, T. 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, 2016, pp. 32–37.

[31] M. R. Garey, D. S. Johnson, Computers and intractability, Vol. 29, wh freeman New York, 2002.

[32] S. Irnich, G. Desaulniers, Shortest path problems with resource constraints, in: Column generation, Springer, 2005, pp. 33–65.

[33] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, et al., ONOS: towards an open, distributed SDN OS, in: Proceedings of the third workshop on Hot topics in software defined networking, ACM, 2014, pp. 1–6.

[34] S. Orlowski, R. Wessäly, M. Pióro, A. Tomaszewski, Sndlib 1.0—survivable network design library, Networks: An International Journal 55 (3) (2010) 276–286.

[35] S. Sahhaf, W. Tavernier, M. Rost, S. Schmid, D. Colle, M. Pickavet, P. Demeester, Network service chaining with optimized network function embedding supporting service decompositions, Computer Networks 93 (2015) 492–505.