VNR Algorithm: A Greedy Approach For Virtual Networks Reconfigurations

Ilhem Fajjari, Nadjib Aitsaadi, Guy Pujolle, Hubert Zimmermann

VNR Algorithm: A Greedy Approach For Virtual Networks Reconfigurations

Ilhem Fajjari[§†], Nadjib Aitsaadi[‡], Guy Pujolle[†] and Hubert Zimmermann[§]
[§]Ginkgo Networks: 2a, rue Danton, 92120 Montrouge, France
[‡]HIPERCOM – INRIA: Domaine de Voluceau - Rocquencourt - B.P. 105, 78153 Le Chesnay Cedex, France
[†]UPMC - University of Paris 6: 4 Place Jussieu, 75005 Paris, France
ilhem.fejjari@ginkgo-networks.com, nadjib.aitsaadi@inria.fr, guy.pujolle@lip6.fr, hubert.zimmermann@ginkgo-networks.com

***Abstract*—In this paper we address the problem of virtual network reconfiguration. In our previous work on virtual network embedding strategies, we found that most virtual network rejections were caused by bottlenecked substrate links while peak resource use is equal to 18%. These observations lead us to propose a new greedy Virtual Network Reconfiguration algorithm, VNR. The main aim of our proposal is to 'tidy up' substrate network in order to minimise the number of overloaded substrate links, while also reducing the cost of reconfiguration. We compare our proposal with the related reconfiguration strategy VNA-Periodic, both of them are incorporated in the best existing embedding strategies VNE-AC and VNE-Greedy in terms of rejection rate. The results obtained show that VNR outperforms VNA-Periodic. Indeed, our research shows that the performances of VNR do not depend on the virtual network embedding strategy. Moreover, VNR minimises the rejection rate of virtual network requests by at least $\simeq 83\%$ while the cost of reconfiguration is lower than with VNA-Periodic.**

Keywords: Network virtualization, Embedding problem, Reconfiguration.

I. Introduction

In the last few years, network virtualization has attracted a great deal of interest from industry and research communities as a means for designing the future Internet architecture. In fact, Network virtualization offers a promising way to share a physical network among many simultaneous, independent and isolated virtual networks [1], [2]. Each Virtual Network ($\mathcal{VN}$) allocates resources in the Substrate Network ($\mathcal{SN}$) such as processing power, memory and bandwidth. The network virtualization paradigm is confronted with many challenges [3], such as security, interoperability, resource scheduling, etc.

In this research paper, we will examine the problem of resource allocation in $\mathcal{SN}$s. The main objective is to maximise the $\mathcal{VN}$ provider's revenue by finding a judicious assignment of a $\mathcal{VN}$ request in the $\mathcal{SN}$. It is worth noting that each virtual node is hosted in only one substrate node. Furthermore, each virtual link is embedded into one substrate path. Indeed, in our previous work [4], we proposed an embedding strategy, denoted by VNE-AC. It is based on Max-Min Ant System metaheuristic and outperforms all the related strategies found in existing literature [5]–[8]. However, in spite of the efficiency of the proposed algorithm, we observed two drawbacks in the results obtained. The first one is the underuse of the $\mathcal{SN}$'s resources. In this regard, we noticed that the average rate of bandwidth use is low (i.e. $\simeq 15\%$). We can therefore conclude that the $\mathcal{SN}$'s profitability can be improved. The second drawback is bottlenecking in the substrate links. Therefore, in most cases, rejected $\mathcal{VN}$ requests are caused by a bandwidth shortfall. It is worth noting that, over time, the $\mathcal{SN}$'s resources become fragmented due to the arrival and departure of new and already hosted $\mathcal{VN}$s respectively. These observations have motivated us to explore the reconfiguration of $\mathcal{VN}$s mapped in the $\mathcal{SN}$ so as to overcome the problem of resource fragmentation. In other words, we aim to 'tidy up' the $\mathcal{SN}$ in order to minimise the rate of overloaded substrate links. The rejection rate of $\mathcal{VN}$ requests will in turn be minimised.

In this paper we will propose a new hybrid (i.e. reactive and proactive) Virtual Network Reconfiguration algorithm, denoted by VNR. The proposed approach is reactive so it is carried out only when an embedding strategy cannot assign a $\mathcal{VN}$ request in the $\mathcal{SN}$. Moreover, VNR is a proactive approach in that it operates only with the $\mathcal{SN}$ and its hosted $\mathcal{VN}$s in order to minimise the rejection rate of future $\mathcal{VN}$ requests. Note that VNR is not interrelated with any mapping algorithm and operates only with the $\mathcal{SN}$ and its hosted $\mathcal{VN}$s. A virtual node with its attached hanging virtual links form a star moving candidate, denoted by $\aleph_i$. Since migration is costly in terms of service interruption period, VNR does not migrate a whole $\mathcal{VN}$ topology but only a minimum set of $\{\aleph_i\}$ hosted in the $\mathcal{SN}$. The main idea behind our proposal is to relocate, using migration, star moving candidates $\{\aleph_i\}$ in the aim of minimising the number of congested substrate links. To do so, VNR first of all sorts, in decreasing order, star moving candidates $\{\aleph_i\}$ of all $\mathcal{VN}$s deployed in the $\mathcal{SN}$. Candidates are sorted based on a predefined metric, κ_i, quantifying the suitability for migration of each $\aleph_i$. Then, VNR selects only the first successful migrating $\aleph_i$ in the sorted selection, among the N_{max} highest (i.e. in terms of κ_i) star moving candidates $\{\aleph_1, \aleph_2, \cdots, \aleph_{N_{max}}\}$. If VNR does not succeed the migration, the embedding strategy rejects the $\mathcal{VN}$ request. Otherwise, the embedding strategy tries to map the $\mathcal{VN}$ request again. If it fails once more, the process is repeated from the sorting stage, at the most N_{mig} iterations.

We incorporated our reconfiguration scheme, VNR, with the best embedding strategies, in term of rejection rate, found in existing literature i) VNE-Greedy [6] and ii) VNE-AC [4]. Based on extensive simulations, we found that VNR significantly decreases the rejection rate of $\mathcal{VN}$ requests. Moreover, we compared VNR with the reconfiguration algorithm VNA-Periodic [5]. The results obtained show that our proposal outperforms VNA-Periodic.

The remainder of this paper is organised as follows. In the next Section we will summarise the related approaches to solving the $\mathcal{VN}$ reconfiguration problem. In Section III we will outline the main reasons that motivated this research. Then, we will formulate the $\mathcal{VN}$ reconfiguration problem in Section IV, before describing our $\mathcal{VN}$ reconfiguration algorithm, VNR, and providing a performance evaluation in Section V and Section VI respectively. Finally, Section VII will conclude this

paper.

II. Related work

$\mathcal{VN}$ reconfiguration is a challenging problem that has only been tackled by a handful research papers. To the best of our knowledge, only the following $\mathcal{VN}$ reconfiguration algorithms can be found in existing literature.

In [5], the authors propose the $\mathcal{VN}$ reconfiguration algorithm `VNA-Periodic`. It is a periodic scheme and is mainly composed of two stages. The first stage involves marking a set of $\mathcal{VN}$s hosted in the $\mathcal{SN}$ that make use of at least one overloaded physical node or link. Then, in the second stage, `VNA-Periodic` uses the corresponding embedding strategy to relocate the marked $\mathcal{VN}$s. The authors claim that the cost of reconfiguration is reduced because only a subset of already mapped $\mathcal{VN}$s is marked. However, we noticed that the cost of `VNA-Periodic` reconfiguration is high since it is periodically triggered and reassigns the whole $\mathcal{VN}$ topology. However, a judicious approach should migrate only virtual nodes and links deployed in excessively loaded substrate nodes and links. Moreover, `VNA-Periodic` does not move the marked $\mathcal{VN}$s by itself. It 'sub-contracts' the work to the initial embedding strategy.

In [6], the authors restrain the reconfiguration problem. In fact, migration is allowed to the virtual links and prohibited to the virtual nodes. To do so, first the reconfiguration algorithm periodically detects the over-loaded substrate links. Then, it finds new substrate paths or updates the split ratio of virtual links that are in transit within overloaded substrate links. Note that the authors base their proposal on traffic path splitting. Moreover, the authors do not take advantage of migrating traffic sources and sinks (i.e. virtual nodes) to minimise bottlenecking in the $\mathcal{SN}$.

In [9], the authors propose an autonomic and distributed reconfiguration algorithm. It is run locally within all substrate nodes. The main idea is to shorten the physical path embedding a virtual link that overloads at least one substrate link according to its incoming/outgoing traffic. To do this, either the source or the destination of traffic (i.e. virtual node) is moved in order to shunt the overloaded substrate link. The reconfiguration algorithm is divided into five stages. First, each substrate node monitors and analyses the presence of overloading traffic. Then, it exchanges monitoring information with its neighbours. Next, each substrate node analyses the received information and decides whether to migrate one of its hosted virtual nodes or to receive a virtual node from its neighbours. After that, a receiving substrate node allocates the resources required to host the moving node. Finally, the virtual node is moved and hence the path length is reduced. The main criticism of this approach is that the migration of nodes is triggered without considering the notion of access and core routers. Besides, contracting a path may require a great deal of moving until the path's length becomes equal to one hop. Moreover, the migration frequency of routers depends on the traffic load, which is actually unstable and correlated to the running applications.

In [10], the authors propose a reactive reconfiguration scheme that is executed only when the defined embedding strategy rejects the $\mathcal{VN}$ request. To achieve this, the proposed algorithm first detects the unmapped virtual nodes and links causing the $\mathcal{VN}$ request rejection. Then, candidate substrate nodes, $\mathcal{C}_i$, for each unmapped virtual node, n_i, are selected. Next, in order to map n_i, the algorithm tries to migrate one of the virtual nodes mapped in $\mathcal{C}_i$ to one node among the potential candidate nodes. The algorithm also detects the bottlenecked substrate links causing the embedding to be blocked. Then, the strategy tries to re-assign one of the virtual links transiting over the overloaded substrate link. To do so, the authors make use the maximum flow algorithm when setting the residual bandwidth of the bottlenecked link to zero. Note that the authors assume that candidate substrate nodes are predefined for each virtual node, which is not realistic. In fact, many embedding strategies do not return this information, such as [4]–[6]. Moreover, the reconfiguration strategy re-assigns virtual links without taking advantage of moving the source and destination (i.e. virtual node). Besides, the maximum flow algorithm does not minimise the path length and consequently bandwidth allocation is not optimised.

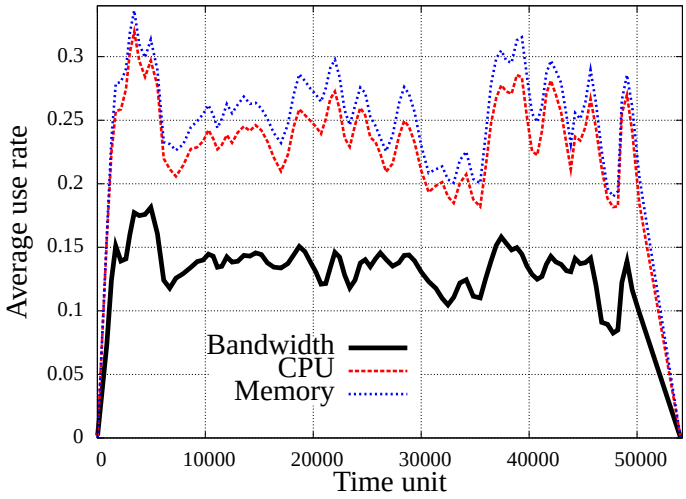


Fig. 1. `VNE-AC`: Average resources use

In this paper we will propose a new virtual network reconfiguration, `VNR`. Unlike [5], we do not relocate the whole $\mathcal{VN}$ topology but only a star in the aim of minimising the cost of reconfiguration. Moreover, our proposal is reactive and is not periodically executed as in [5], [6], [9]. Besides, it can be considered as a proactive approach since it also 'tidies up' the $\mathcal{SN}$ in order avoid future $\mathcal{VN}$ rejection. Unlike [5], [10], `VNR` is not interrelated with the embedding strategy and does not need any details about the $\mathcal{VN}$ request rejected (e.g. unmapped virtual nodes, etc). Finally, our proposal is not based on path splitting as in [6] and takes into account the access and core routers, unlike [9].

III. Motivation for $\mathcal{VN}$ Reconfiguration

In our previous work [4], we proposed the embedding strategy called `VNE-AC` which is based on Max-Min Ant System metaheuristic. The rejection rate of $\mathcal{VN}$ requests obtained with `VNE-AC` is equal to $4.56 \pm 0.40\%$. It is worth noting that the latter result is the lowest obtained result compared with related strategies and it is calculated according to the benchmark scenario assumed in [5]–[8]. We analysed the extensive simulation traces and determined the reasons for $\mathcal{VN}$ rejection. In fact, $\simeq 99\%$ of rejections were caused by a bandwidth shortage (i.e. substrate links). The remaining,

$\simeq 1\%$ was due to a lack of processing power or memory (i.e. substrate nodes).

We calculated the average resource use as illustrated in Fig. 1 and noticed that during simulations, the peak average bandwidth use in the $\mathcal{SN}$ is equal to 18%. Thus, we concluded that $\mathcal{SN}$ use is not optimised since $\simeq 99\%$ of $\mathcal{VN}$ rejections are caused by bandwidth deficiency while, at the most, only 18% of bandwidth is exploited in the $\mathcal{SN}$. Moreover, we concluded that many substrate links are overloaded (i.e. bottlenecked).

These findings motivated us to explore the reconfiguration of $\mathcal{VN}$s already hosted in the $\mathcal{SN}$. Thanks to migration techniques, moving virtual nodes [11] and paths [12] becomes easily feasible. Thus, our aim is to propose a new reconfiguration algorithm to balance the load within the $\mathcal{SN}$. As a result, the $\mathcal{VN}$ rejection rate will be minimised.

IV. Formulation of $\mathcal{VN}$ Reconfiguration Problem

We can model the $\mathcal{SN}$ as an undirected graph, denoted by $\mathcal{G}^s(\mathcal{N}^s, \mathcal{E}^s)$, where $\mathcal{N}^s$ and $\mathcal{E}^s$ are the sets of physical nodes and their connected links respectively. Each physical node, $n_i^s \in \mathcal{N}^s$, is characterised by its i) residual processing power ($\mathcal{C}_{n_i^s}$), ii) residual memory ($\mathcal{M}_{n_i^s}$) and iii) type: access or core ($\mathcal{X}_{n_i^s}$). Note that if $\mathcal{X}_{n_i^s} = 1$, then n_i^s is an access node. Otherwise, $\mathcal{X}_{n_i^s} = 0$. Likewise, each physical link, $e_x^s \in \mathcal{E}^s$, is typified by its available residual bandwidth, denoted by $\mathcal{B}_{e_x^s}$.

Similarly, a $\mathcal{VN}$ request can be modelled as an undirected graph, denoted by $\mathcal{G}^v(\mathcal{N}^v, \mathcal{E}^v)$, where $\mathcal{N}^v$ and $\mathcal{E}^v$ are the sets of virtual nodes and their virtual links respectively. Within a $\mathcal{VN}$ request, each virtual node, $n_i^v \in \mathcal{N}^v$, is associated with the i) required processing power ($\mathcal{C}_{n_i^v}$), ii) required memory ($\mathcal{M}_{n_i^v}$), and iii) its type ($\mathcal{X}_{n_i^v}$). Moreover, each virtual link, $e_x^v \in \mathcal{E}^v$, requests $\mathcal{B}_{e_x^v}$ in terms of bandwidth.

As stated in Section III, the main aim is to decrease the number of congested substrate links since substrate nodes do not affect the rejection of $\mathcal{VN}$ requests. Let ζ denote the set of α-congested substrate links. Formally,

$$\zeta = \{e_x^s \in \mathcal{E}^s : \mathcal{B}_{e_x^s} \leq (1-\alpha)\mathcal{B}_{e_x^s}^{max}\} \quad (1)$$

where $\mathcal{B}_{e_x^s}^{max}$ is the bandwidth capacity of e_x^s and $0 \leq \alpha \leq 1$.

In fact, our first objective is to reconfigure the Accepted $\mathcal{VN}$ Requests, $\mathcal{AR}$, within the $\mathcal{SN}$ in the aim of minimising the number of α-congested links. Formally,

$$minimise_{\mathcal{AR}}\ (|\zeta|) \quad (2)$$

Our second objective is to minimise the cost of reconfiguration. Let ϕ denote the migration cost, based on the reconfiguration cost proposed in [5], then we can define ϕ as the weighted sum of migrated virtual nodes ϕ_n and links ϕ_e (i.e. $\phi = a\phi_n + b\phi_e$). Formally,

$$minimise_{\mathcal{AR}}\ (a\phi_n + b\phi_e) \quad (3)$$

where $0 \leq a \leq 1$ and $0 \leq b \leq 1$ are the weights of ϕ_n and ϕ_e respectively. Note that $a + b$ must be equal to 1.

In addition, a virtual node, n^v, can move to a substrate node, n^s, only if both nodes are of the same type (i.e. access or core). Besides, a substrate node cannot embed two virtual nodes belonging to the same $\mathcal{VN}$.

Algorithm 1: Embedding scheme of $\mathcal{VN}_x$

```
while VN queue Q is not empty do
    VN_x ← Q.pop
    stop_1 ← false
    i ← 1
    while ( (stop_1 = false) and (i ≤ N_mig) ) do
        Call the embedding strategy to assign VN_x
        if (Successful mapping) then
            stop_1 ← true
        else
            Call VNR
            if (Successful reconfiguration) then
                i ← i + 1
            else
                stop_1 ← true
```

The problem outlined above is a multi-objective optimisation problem with conflictual objectives. It is a combinatorial optimisation problem that is known as NP-hard [13]. In spite of the similarities between our problem and the MPLS routing problem, none of the MPLS-related strategies [14] can be used. Indeed, the MPLS problem can be formulated as a Multi-Commodity Flow problem [15] since a source/destination of traffic cannot be migrated. However, in our case, we have more flexibility and the possibility of reducing bottlenecking in the $\mathcal{SN}$ by migrating sources/destinations of traffic and/or reassigning virtual links. Hereafter, we will describe our new Virtual Network Reconfiguration algorithm, VNR.

V. Proposal: VNR Algorithm

In this section we will describe our $\mathcal{VN}$ hybrid reconfiguration algorithm, VNR. Our proposal is a reactive scheme, in that it is executed when the $\mathcal{VN}$ embedding strategy rejects a $\mathcal{VN}_x$ request. Moreover, VNR is not interrelated with any $\mathcal{VN}$ embedding strategy. Indeed, VNR is a proactive approach in that it operates only with the $\mathcal{SN}$ and its hosted $\mathcal{VN}$s in order to minimise the rejection rate of future $\mathcal{VN}$ requests. In other words, our proposal 'tidies up' the $\mathcal{SN}$ further to physical resource fragmentation due to the arrival and departure of new and already hosted $\mathcal{VN}$s respectively.

As explained in Section III, the main problem is overloaded substrate links. In this respect, we propose to migrate in each iteration one star moving candidate, denoted by $\aleph_i$. Note that $\aleph_i$ is formed by one virtual node and its attached hanging virtual links. The main idea behind star topology moving is to reassign virtual links while migrating the central virtual node. In other words, we can see the migration as an elastic movement in which a moving virtual node n^v is fastened with its attached virtual links that can be considered like elastics. Indeed, in most cases, the moving virtual node can land in any substrate node since all $\mathcal{VN}$ rejections are caused by a lack of bandwidth, as explained in Section III. Thus, our challenge is to find the best hosting substrate node to reduce the number of α-congested links. In doing so, we will not re-embed all the $\mathcal{VN}$ topology but only the canonical star topology $\aleph_i$. Consequently, we will reduce the service interruption period

Algorithm 2: VNR

```
Generate the set of star moving candidates ℵ in the SN
for j=1 to |ℵ| do
    Calculate κ_j for ℵ_j
Sort out decreasingly {ℵ_j} according to {κ_j}
stop_2 ← false
j ← 1
while (stop_2 = false) & (j ≤ N_max) do
    Migrate (ℵ_j)
    if (Successful migration) then
        stop_2 ← true
    else
        j ← j + 1
```

of $\mathcal{VN}$s.

When a $\mathcal{VN}$ request is rejected, VNR begins by sorting out the star moving candidates $\{\aleph_i\}$, in decreasing order, of all $\mathcal{VN}$s mapped in the $\mathcal{SN}$. They are sorted according to κ_i a criterion which quantifies the $\aleph_i$'s suitability for migration. To this end, κ_i is defined according to i) the number, $\mathcal{A}_i$, of α-congested links in the paths embedding $\aleph_i$'s virtual links and ii) its residual lifetime, $\mathcal{T}_i$. Formally,

$$\kappa_i = \mathcal{A}_i \times \mathcal{T}_i \tag{4}$$

Note that we favour star moving candidates that will expire as late as possible and which whose virtual links pass through the greatest number of α-congested links. In fact, reconfiguring a $\{\aleph_i\}$ that is soon set to leave the $\mathcal{SN}$ is not judicious. For this reason, κ_i is proportional to the residual lifetime $\mathcal{T}_i$.

Initially, VNR selects only the first successful migrating $\aleph_i$, according to the sorting order, among the N_{max} highest ranking star moving candidates $\{\aleph_1, \aleph_2, \cdots, \aleph_{N_{max}}\}$. If VNR does not succeed the migration, the embedding strategy rejects the $\mathcal{VN}$ request. Otherwise, the embedding strategy tries to map the $\mathcal{VN}$ request again. If it fails once more, the process is repeated from the sorting stage, at the most N_{mig} iterations. The VNR strategy is summarised in Algorithms 1 and 2.

To reconfigure $\aleph_i$, our proposal proceeds as follows. First, VNR generates a new substrate network, denoted by $\widehat{\mathcal{SN}}$. It is similar to $\mathcal{SN}$ in terms of i) topology and ii) residual resources **but without** substrate links with residual bandwidth lower than the **minimum** bandwidth required by $\aleph_i$'s virtual links. Next, VNR verifies whether $\widehat{\mathcal{SN}}$ contains a connected component including **all** the substrate nodes hosting the one-hop neighbours of $\aleph_i$'s star-centre virtual node. If the connected component does not exist, it is impossible to migrate $\aleph_i$ because at least one $\aleph_i$'s virtual link cannot be mapped. Otherwise, VNR selects substrate node n_s within the connected component, in the aim of migrating $\aleph_i$'s virtual node to n_s and its attached virtual links thus minimising $\varrho_{\aleph_i}^{n_s}$. It is equal to:

$$\varrho_{\aleph_i}^{n_s} = \frac{\text{Average bandwidth of substrate paths hosting } \aleph_i}{\mathcal{A}_i + 1} \tag{5}$$

Note that the reassignment of $\aleph_i$'s virtual links is based on a shortest path algorithm using our proposed path definition metric in [4]. The star moving candidate $\aleph_i$ migration method is summarised in Algorithm 3.

Algorithm 3: Migration of $\aleph_i$

```
SN̂ ← SN/{e_x^s : B_{e_x^s} < min_{B_{e_x^v} ∈ ℵ_i} (B_{e_x^v})}
N̂_v ← {set of 1-hop virtual neighbours of ℵ_i's central virtual node}
N̂_s ← {set of susbsrate nodes hosting virtual nodes N̂_v }
CC ← connected components of SN̂
if (∃ CC_x ∈ CC / N̂_s ⊆ CC_x) then
    if ∄ n_s ∈ CC_x / migration of ℵ_i's virtual node to n_s and its attached virtual links succeeds then
        Failed migration
    else
        Select the best substrate node n_s ∈ CC_x:
        migration of ℵ_i's virtual node to n_s and its
        attached virtual links minimises ϱ_{ℵ_i}^{n_s}
        Successful reconfiguration
else
    Failed migration
```

VI. Performance Evaluation

In this section, we will study the efficiency of our proposal, VNR. To achieve this, we will first describe our discrete event $\mathcal{VN}$ embedding simulator. Then, we will define several metrics in the aim of showing the advantages of VNR compared to the proactive reconfiguration algorithm VNA-Periodic [5]. Note that we cannot compare our proposal with the reconfiguration algorithms proposed in [6], [9], [10] since the assumptions and constraints are different. In [6] the authors assume a path splitting approach, while in [9] the type of router (i.e. access and core) is not taken into consideration and in [10] the algorithm requires as inputs the virtual nodes and links that the embedding algorithm failed to assign (see Section II).

Our reconfiguration scheme VNR and VNA-Periodic will be incorporated with the best embedding strategies found in existing literature i) ii) VNE-AC [4] and ii) VNE-Greedy [6]. Finally, we will build on the outputs of the above simulations to assess our proposal and comment the results obtained.

A. Simulation Environment

We implemented a discrete event $\mathcal{VN}$ embedding simulator. In this respect, the GT-ITM tool is used to generate random $\mathcal{SN}$ and $\mathcal{VN}$ topologies. The arrival of $\mathcal{VN}$ requests is modelled by a Poisson Process with rate λ_A and $\mathcal{VN}$ lifetime is modelled by exponential distribution with mean μ_L.

As stated in [4], [6], [10], we make use the following benchmark scenario. The $\mathcal{SN}$ size is set to 100 and, in this case, the ratio of access and core nodes is fixed at 20% and 80%, respectively. Furthermore, the $\mathcal{VN}$ size is set according to a discrete uniform distribution, using the values given in $[2, 10]$. Since virtual access nodes are defined by customers, we can assume that each virtual node could be access or core with a probability of 0.5. It is worth noting that in both cases ($\mathcal{VN}$ and $\mathcal{SN}$), each pair of nodes is randomly connected with a probability of 0.5. The arrival rate, λ_A, and the average lifetime, μ_L, of $\mathcal{VN}$s are fixed to 4 requests per 100 time unit

TABLE I
REJECTION RATE OF $\mathcal{VN}$ REQUESTS - $\mathbb{Q}$ (%)

Strategy	VNE-AC	VNE-Greedy
Without reconfiguration	4.56 ± 0.40	12.95
VNA-Periodic	4.31 ± 0.38	11.55
VNR	0.37 ± 0.098	2.2

TABLE II
DECREASE OF REJECTION RATE (%)

Strategy	VNE-AC	VNE-Greedy
VNA-Periodic	5.48	10.81
VNR	91.88	83.01

and 1000 time units respectively. We calibrate the capacity of substrate nodes and links (i.e. $\mathcal{C}_{n_i^s}$, $\mathcal{M}_{n_i^s}$, and $\mathcal{B}_{e_x^s}$) according to a continuous uniform distribution, taking the values in $[50, 100]$. Similarly, the required virtual resources (i.e. $\mathcal{C}_{n_i^v}$, $\mathcal{M}_{n_i^v}$, and $\mathcal{B}_{e_x^v}$) are set according to a continuous uniform distribution, using the values given in $[10, 20]$. The number of $\mathcal{VN}$ requests are set to 2000, and, based on extensive simulations, N_{mig} and N_{max} are set to 5% and 10% of currently mapped virtual nodes respectively.

We set the parameters of VNE-AC and VNA-Periodic as calibrated in [5] and [4] respectively. Note also that each performance value of pseudo-random strategies is equal to the average of 30 simulations. Moreover, simulation results are always presented with confidence intervals corresponding to a confidence level of 99.70%. Only minute confidence intervals are not shown in the following figures.

B. Performance Metrics

In this section, we will define the performance metrics used to assess our proposal.

1) $\mathbb{Q}$*:* is the final rejection rate of $\mathcal{VN}$ requests.

2) ϕ*:* is the cost of migration as defined in equation 3, $\phi = (a\phi_n + b\phi_e)$. We set $a = b = \frac{1}{2}$.

3) $\mathbb{U}(t)$*:* measures the average use rate of resources (i.e. processing power, memory, bandwidth) in the $\mathcal{SN}$ substrate links at time t. For example, the average bandwidth use rate can be expressed as:

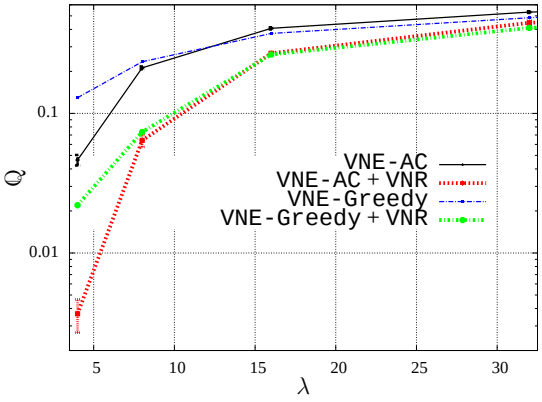
$$\mathbb{U}(t) = \frac{1}{|\mathcal{E}^s|} \times \sum_{e_s^x \in \mathcal{E}^s} \left(\frac{\mathcal{B}_{e_s^x}}{\mathcal{B}_{e_s^x}^{max}} \right) \quad (6)$$

In the following section, we will present the results of our simulations and summarise the key observations.

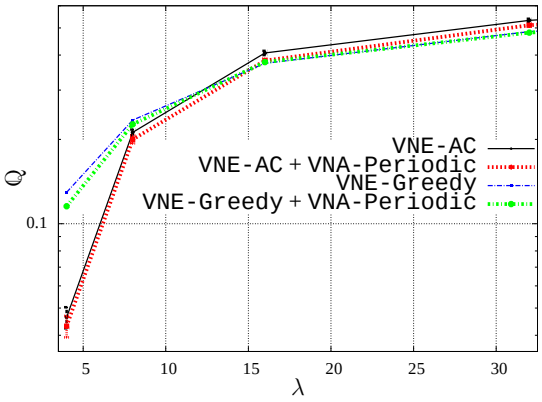
C. Evaluation Results

Table I shows a comparison of the rejection rate of $\mathcal{VN}$ requests obtained by the embedding strategies VNE-AC and VNE-Greedy when the reconfiguration algorithms (i.e. VNA-Periodic and VNR) are incorporated. We can first of all see that our reconfiguration algorithm VNR and VNA-Periodic decrease the rejection rate of $\mathcal{VN}$ requests whatever the embedding strategy used (i.e. VNE-AC and VNE-Greedy). This means that reconfiguration algorithms enhance the acceptance of $\mathcal{VN}$s in the $\mathcal{SN}$. Moreover, the results show that our proposal, VNR, significantly reduces the rejection rate with both mapping strategies and more so than VNA-Periodic. In fact, as illustrated in Table II, our VNR strategy reduces the rejection rate by at least 83% . This is consistently better than VNA-Periodic since VNR is at least 10 times more effective in terms of reducing $\mathcal{VN}$ rejection rate.

An important factor to consider when evaluating the effectiveness of a reconfiguration-strategy is the incurred cost of migrating virtual nodes and links. In fact, in spite of the flexibility that virtualization offers in terms of moving virtual nodes and links, the service disruption caused by migrations can have a negative impact on running applications (e.g. video streaming, VoIP, etc.). Indeed, in [16] we evaluated the average migration delay in testbed platform to $2ms$, which can disturb real-time applications. By triggering reconfiguration only when needed, VNR significantly reduces the cost of reconfiguration. As illustrated in Table III, our proposal considerably reduces the cost of migration. In fact, the results show that VNR reduces the cost between 2 and 5 times more than VNA-Periodic. We can therefore conclude that our proposal outperforms VNA-Periodic in terms of rejection rate and cost of migration.



(a) VNR



(b) VNA-Periodic

Fig. 2. Robustness of $\mathcal{VN}$ reconfiguration algorithms

TABLE III
COST OF MIGRATION - ϕ

Strategy	VNE-AC	VNE-Greedy
VNA-Periodic	2598.13 ± 134.74	3625
VNR	500.67 ± 35.83	1866

Fig. 2 evaluates the robustness of VNR and VNA-Periodic when the arrival rate of $\mathcal{VN}$ requests, λ, increases while their average lifetime is fixed. It is clear

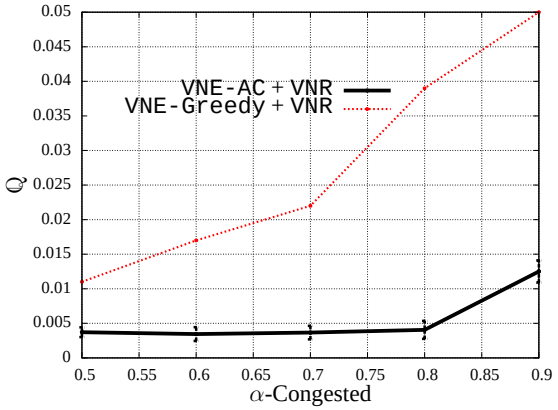


Fig. 3. VNR: Impact of α-congested level

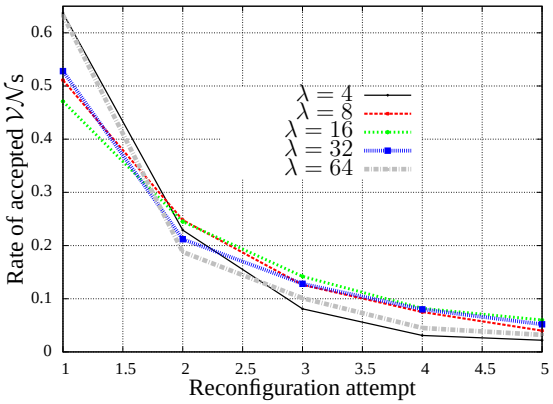


Fig. 4. VNE-AC + VNR: Impact of N_{mig}

to see that our proposal significantly reduces the rejection rate with both $\mathcal{VN}$ embedding strategies (i.e. VNE-AC and VNE-Greedy) while the arrival rate of $\mathcal{VN}$s grows. For example, with VNE-AC and λ equal to 8, 16 and 32, VNR reduces the rejection rate by 69.81%, 33.69% and 16.17% respectively. Note that the rejection rate improvement stills high while the arrival rate grows exponentially. Moreover, with high values of λ, the $\mathcal{SN}$ is saturated. On the other hand, VNA-Periodic does not resist to the $\mathcal{VN}$ request flow. For example with VNE-AC and λ is equal to 8 and 32, VNA-Periodic reduces the rejection rate by 3.41% and 0.8% respectively. Besides, our tests showed that when $\lambda = 16$, VNA-Periodic slightly increases the rejection rate.

Fig. 3 shows the impact of the value of α-congested substrate links on the rejection rate of $\mathcal{VN}$s. We can see that when α is high (e.g. 90%), the rejection rate increases. This can be explained by the fact that VNR only deals with bottlenecked substrate links. Hence, VNR cannot easily 'tidy up' the $\mathcal{SN}$ since it acts late. However, with small α values VNR prevents the overloading of substrate links by moving the virtual links mapped to bottlenecked substrate links. Based on extensive simulations, we can see that good performances are achieved with a small value of α.

Fig. 4 illustrates the impact of maximal number of iterations, N_{mig}, on VNR's performance. Note that Fig. 4 depicts the rate of accepted $\mathcal{VN}$ requests that needed to be reconfigured in the $\mathcal{SN}$ to be mapped. This is carried out according to the maximum number of reconfigurations allowed per $\mathcal{VN}$ request (i.e. N_{mig}). The results show that most (i.e. $\simeq 50\%$) of the embedded $\mathcal{VN}$ requests require only one reconfiguration. Moreover, the arrival rate of $\mathcal{VN}$ requests has no impact on these results.

VII. Conclusion

Our research studied the problem of virtual network reconfiguration. The main objective was to reduce the rejection of virtual networks. To do so, we proposed a new greedy reconfiguration algorithm, VNR whose main aim is to reassign canonical star virtual topologies hosted in the overloaded substrate nodes and links. We incorporated our proposal with the best existing embedding strategies (i.e. VNE-AC and VNE-Greedy) and compared it with the related virtual network reconfiguration algorithm VNA-Periodic. The results obtained show that VNR minimises the virtual network rejection rate at least by $\simeq 83\%$ while the cost of reconfiguration is also lower (five times lower than VNA-Periodic).

References

[1] N. M. M. K. Chowdhury and R. Boutaba, "Network virtualization: State of the art and research challenges," *IEEE Communications Magazine*, vol. 47, 2009.

[2] J. Carapinha and J. Jiménez, "Network virtualization: a view from the bottom," *ACM workshop on Virtualized infrastructure systems and architecture*, pp. 73–80, 2009.

[3] N. M. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," *Computer Networks*, vol. 54, 2010.

[4] I. Fajjari, N. Aitsaadi, G. Pujolle, and H. Zimmermann, "VNE-AC: Virtual Network Embedding Algorithm based on Ant Colony Metaheuristic," *IEEE ICC*, 2011.

[5] Y. Zhu and M. Ammar, "Algorithms for assigning substrate network resources to virtual network components," *IEEE INFOCOM*, pp. 1–12, 2006.

[6] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: substrate support for path splitting and migration," *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 17–29, 2008.

[7] J. Lu and J. Turner, "Efficient mapping of virtual networks onto a shared substrate," Washington University in St. Louis, Tech. Rep., 2006. [Online]. Available: http://www.arl.wustl.edu/~jl1/research/tech_report_2006.pdf

[8] N. Chowdhury, M. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," *IEEE INFOCOM*, pp. 783–791, 2009.

[9] C. C. Marquezan, L. Z. Granville, G. Nunzi, and M. Brunner, "Distributed autonomic resource management for network virtualization," *IEEE Network Operations and Management Symposium, NOMS*, pp. 463–470, 2010.

[10] N. F. Butt, M. Chowdhury, and R. Boutaba, "Topology-awareness and reoptimization mechanism for virtual network embedding," *IFIP NETWORKING Conference*, pp. 27–39, 2010.

[11] Y. Wang, E. Keller, B. Biskeborn, J. van der Merwe, and J. Rexford, "Virtual routers on the move: Live router migration as a network-management primitive," *ACM SIGCOMM*, pp. 231–242, 2008.

[12] M. Agrawal, S. R. Bailey, A. Greenberg, J. Pastor, P. Sebos, S. Seshan, K. van der Merwe, and J. Yates, "Routerfarm: towards a dynamic, manageable network edge," *ACM SIGCOMM*, pp. 5–10, 2006.

[13] M. Ehrgott and X. Gandibleux, "A survey and annotated bibliography of multiobjective combinatorial optimization," *OR Spektrum*, vol. 22, no. 4, pp. 425–460, 2000.

[14] M. Kodialam and T. Lakshman, "Minimum interference routing with applications to mpls traffic engineering," *IEEE Infocom*, pp. 884–893, 2000.

[15] A. Srinivasan, "A survey of the role of multicommodity flow and randomization in network design and routing," *American Mathematical Society, Series in Discrete Mathematics and Theoretical Computer Science*, pp. 271–302, 1999.

[16] I. Fajjari, M. Ayari, G. Pujolle, and H. Zimmermann, "Towards an autonomic piloting virtual network architecture," *IFIP International Conference on New Technologies, Mobility and Security - NTMS*, pp. 1–5, 2011.