An Adaptive Load Balancing Scheme for Evolving Virtual Networks

Houda Jmila, Djamal Zeghlache

An Adaptive Load Balancing Scheme for Evolving Virtual Networks

Conference Paper · January 2015

DOI: 10.1109/CCNC.2015.7158024

2 authors:

Houda Jmila
Institut Mines-Télécom
7 PUBLICATIONS **5** CITATIONS
SEE PROFILE

Djamal Zeghlache
Institut Mines-Télécom
207 PUBLICATIONS **1,508** CITATIONS
SEE PROFILE

An Adaptive Load Balancing Scheme for Evolving Virtual Networks

Houda Jmila*, Djamal Zeghlache*
* *Institut Mines Telecom, Telecom SudParis and UMR*5157 *of CNRS, Evry, France*
{*houda.jmila, djamal.zeghlache*}*@telecom-sudparis.eu*

***Abstract*—An algorithm to adapt dynamically virtual networks to additional resource requirements is proposed and evaluated. The optimization is achieved while balancing load and avoiding fragmentation in the infrastructure (often referred as substrate or physical network). The algorithm focuses on virtual nodes requiring more resources by extending their allocations and maintaining their connectivity (even if the node is migrated) to other resources while tidying up (or consolidating) the infrastructure. The algorithm outperforms existing approaches.**

***Keywords*-Resource allocation, evolving virtual network, load balancing, Cloud, QoS;**

I. Introduction

Network Virtualization allows the co-existence of multiple Virtual Networks (VNs) on the same physical infrastructure (often referred as substrate network, SN). Virtual network embedding algorithms are typically used to enable the sharing of the infrastructure among multiple users or tenants. Virtual Network Embedding (VNE) maps the virtual nodes and links requests from users onto the graph representing the physical infrastructure nodes and links and their connectivity. Most VNE solutions allocate a *static* amount of resources to the VN during its lifetime, however, cloud environments require dynamic and elastic resource allocation in line with varying user resource requirements. Reactive or periodic re-configurations have nevertheless been proposed to re-optimize the SN utilization. Unfortunately, they lead to network instability and service disruptions. When adapting and extending virtual network allocations, stability and consolidation of the infrastructure have to be ensured to optimize utilization and maintain quality of service. The proposed algorithm aims at this simultaneous dynamic adaptation of VN allocations and optimal SN utilization to continuously support varying applications requirements. A load-balancing aware strategy is combined with the dynamic adaptation algorithm to deal with fluctuating demands while optimizing resource utilization.

More specifically this paper deals with user requests for additional resources from physical nodes to support virtual nodes elasticity requirements. In our previous work [1], we improved performance of prior art by reshuffling virtual resources across physical nodes (or hosts) but did not consider SN efficient or optimal utilization to increase profitability. To fill this gap, we propose a new bi-objective adaptive scheme that re-optimizes the SN utilization *while* responding to additional resource requests. This minimizes the need for making regular reconfigurations (or consolidations) to improve SN utilization. We take advantage of the demand fluctuation to simultaneously "tidy up " the substrate network by balancing the load among substrate links though migration of more congestion causing virtual resources to other more adequate physical nodes. This frees resources and makes room to meet evolving virtual nodes requirements in the affected substrate nodes. Note that this is achieved by selecting new links (attached to the new node) that respect (do not degrade) the applications quality of service requirements.

Section II of the paper describes related work on virtual network embedding. Section III analyzes and formulates the problem. Section IV presents our proposed heuristic algorithm to achieve minimum cost and load balancing. The results of performance evaluation and a comparison to prior art are reported in sections V and VI.

II. Related work

The problem of Virtual Network Embedding has been intensively studied [2]–[5] but adaptation of already embedded VNs to dynamically optimize resource utilization has received little attention. To the best of our knowledge only [2], [3], [6]–[8] addressed this problem. The presented adaptation strategies can be classified into two main families: i) periodic and ii) reactive approaches. The first family periodically selects and re-allocates the entire [2] or parts [3] of the underlying VNs but this induces high reconfiguration cost and network instability. The second family executes the re-allocation scheme only when a virtual network request is rejected thus affecting user satisfaction.

The authors of [2] propose an online virtual network reconfiguration algorithm that operates on VNs using congested substrate resources. A periodic scheme first marks the set of VNs to re-allocate by checking the overloaded physical nodes and links. The algorithm then reassigns the entire marked VN topology by re-running the initial embedding algorithm. Unfortunately, such *periodic* re-allocation is very costly and mapping again the *whole* VN topology disrupts more running services than needed because of the global rearrangements.

Work in [3] uses a periodic path migration algorithm to minimize used bandwidth to increase the VN acceptance ratio. The algorithms runs again the initial link-mapping

algorithm to select new underlying paths. The authors do not take advantage of migrating traffic sources and sinks (i.e. virtual nodes) and unfortunately limit the reconfiguration problem to path migration.

Authors of [6] propose a reactive reconfiguration scheme that first detects the unmapped virtual nodes and links causing the rejection of the VN request, and then moves the congested links and nodes to less critical hosts. Authors of [7], [8] propose a reactive strategy that takes into account the cost incurred by the service disruption during re-allocation.

All the cited previous work enhances SN utilization but leads to network instability and service disruption of reconfigured VNs. In order to minimize the negative impacts of such reconfigurations, we propose to "tidy up " the SN when responding to fluctuating (increasing) VN resources requirements at minimum cost and disruptions.

The rest of the literature focused on *static* allocation and placement of virtual resources in physical hosts. Only a handful of papers, such as our previous work in [1] and [9]–[12], tackled dynamic allocation in response to fluctuations in demand and elasticity requirements. The common approach is to use a heuristic algorithm to cost efficiently re-allocate virtual nodes and/or links to satisfy the new increasing demand. All this previous work [9]–[12] was not concerned by efficient SN utilization or profitability when satisfying new resource requirements. In this paper we fill this gap by combining the two objectives. We adapt resource allocations at minimum cost, respect quality of service of all running applications and simultaneously maximize utilization by balancing the load on the SN links while meeting new resource requirements of already embedded virtual nodes.

III. Problem formulation and Model

The problem to solve is that of allocating additional infrastructure resources, from a substrate network (SN), to virtual nodes of already embedded and active virtual networks. Our goal is to adapt previous assignments while minimizing nodes and links re-allocations costs and the average saturation of the links to ensure load balancing in the SN. We consequently propose a bi-objective function to minimize jointly i) the cost of re-allocations and ii) the average link saturation in SN to provide the required elasticity for evolving nodes while maximizing SN utilization (or profitability).

A. Network Model

The cloud infrastructure (referred in this work as SN) can be represented by a weighted undirected graph $G_s = (N_s, L_s)$, where N_s is the set of *substrate nodes* n_s (e.g. physical servers) and L_s is the set of *substrate links* l_s (e.g. intra and inter cloud physical links). We denote by $a^t_{n_s}$ the *available* capacity of node n_s (typically CPU and memory) and by $a^t_{l_s}$ the *available* bandwidth on a link l_s. We use φ to represent a substrate path (a single or a sequence of substrate links) between two substrate nodes and P_φ to represent the set of substrate paths. The available bandwidth a^t_φ associated to a substrate path φ can be evaluated as the smallest available bandwidth on the links along the substrate path.

B. VN Resource Request Model

A VN request, expressed and sent by users to the cloud providers, is a set of *virtual nodes* interconnected via *virtual links*. This set is hence also a graph that is represented by a weighted undirected graph $G_v = (N_v, L_v)$, where N_v and L_v are respectively the virtual nodes and links of the VN request. Each virtual node $n_v \in N_v$ is associated with a minimum required capacity denoted by $b^t_{n_v}$. Each virtual link $l_v \in L_v$ is associated with a minimum required bandwidth denoted by $b^t_{l_v}$. To complete the VN Resource Request Model, we denote VN^t the set of running VNs on G_s at time t. In addition, we represent by m^i_v an *evolving* node running on a virtual network $i \in VN^t$, i.e. a virtual node with more resource requirements $b^{t+1}_{m^i_v} > b^t_{m^i_v}$.

C. VN Mapping Model

For each VN request G^r_v in the substrate network, $(M^t_{N^r_v}, M^t_{L^r_v})$ describes its mapping in the substrate network at time t that respects the resource constraints expressed in the request. More precisely, $M^t_{N^r_v} : N^r_v \to N_s$ describes the node mapping and $M^t_{L^r_v} : L^r_v \to P_\varphi$ is the link mapping.

D. Metric for the Substrate Network

In order to quantify the amount of resources used by the substrate network to fulfill the VN requests, we use the notion of stress. As most VN request rejections are caused by bandwidth shortage [2], [3] we focus on avoiding substrate links saturation by balancing the load. Similarly to [4] we define the link stress of a substrate link l_s as the ratio of the total amount of bandwidth allocated to the virtual links whose substrate paths pass through l_s over the amount of bandwidth initially available in l_s. Formally:

$$s^t_{l_s} = \frac{\sum_{l_v \to l_s} b^t_{l_v}}{a^0_{l_s}} = \frac{a^0_{l_s} - a^t_{l_s}}{a^0_{l_s}} \tag{1}$$

where $l_v \to l_s$ indicates that the substrate path of virtual link l_v passes through the substrate link l_s, and $a^0_{l_s}$ is the initial available bandwidth in l_s. The average link stress ALS^t in SN is defined consequently as:

$$ALS^t = \frac{\sum_{l_s \in L_s} s^t_{l_s}}{|L_s|} \tag{2}$$

where $|L_s|$ is the total number of substrate links in SN.

E. Problem Formulation

For a running evolving node m^i_v requiring additional resources in a substrate host h (with $M^t_{N^r_v}(m^i_v) = h$) that has insufficient resources, we need a strategy to re-allocate resources onto other alternate candidate nodes (those having enough resources) to maintain the service. A trivial and

suboptimal strategy is to move the entire evolving node to another less loaded host [9]. We proposed a more elaborate strategy in [1] by reorganizing virtual nodes in the initial host in neighboring hosts while minimizing overall re-allocation cost without considering SN utilization. In this paper, we extend the work by moving some candidate virtual nodes in the affected physical node to make room for the additional needs and balance the load on the SN at the same time in order to also optimize SN utilization and profitability. This is achieved by minimizing the average link saturation of the entire SN in addition to making cost effective re-allocation and migration decisions. Intuitively, the most congestion causing virtual nodes in the initial host should be selected in priority as candidates for re-allocation and migration.

1) Optimization objective

As in [1], [9], we consider two phases for node re-allocation: *remapping* and *migration*. The remapping phase consists in finding alternate resources to host the candidate evolving virtual node and its associated virtual links. The migration phase tries in a second stage to migrate tasks running on the virtual node onto the selected destination node to resume these tasks. Note that this phase can affect the migrated application or service, that will experience a downtime or unavailability period that needs to be taken into account and minimized [4]. Hence the node re-allocation incurs a remapping cost $Cost_{remap}$ and a migration cost $Cost_{mig}$ to reflect all these effects.

To derive the expressions for the mapping and migration costs, we define $n_v^r \in N_v^r, r \in VN^t$ as a candidate virtual node selected for re-allocation and $\mathbb{S}_{n_v^r}$ as the star topology formed by n_v^r and its connected virtual links. We can as in our previous work [1] express the re-mapping and migration costs:

- **Re-mapping cost**:

$$Cost_{remap}\left(n_v^r\right) = b_{n_v^r}^{t+1} * cost\left(M_{N_v^r}^{t+1}\left(n^r\right)\right) + \sum_{l_v^r \in \mathbb{S}_{n_v^r}} \sum_{l_s \in M_{L_v^r}^{t+1}(l_v^r)} b_{l_v^r}^{t+1} * cost(l_s) \quad (3)$$

Where $cost\left(n_s\right)$ and $cost\left(l_s\right)$ are the unit costs of the substrate node and the substrate link respectively, and with $(M_{N_v^r}^{t+1}, M_{L_v^r}^{t+1})$ describing the new mappings of the re-allocated resources.

- **Migration cost**:

$$cost_{mig}\left(n_v^r\right) = \sum_{l_s \in p_{mig(n_v^r)}} minBW_{n_v^r} * cost(l_s) \quad (4)$$

Where n_v^r, $b_{n_v^r}^{t+1}$ is the size of the re-allocated virtual node, $p_{mig(n_v^r)} \in P_\varphi$ the temporary established substrate path between the old and new hosts to support task migration, and $minBW_{n_v^r}$ the minimum required bandwidth to migrate that task. This bandwidth is calculated according to the maximum downtime acceptable to the VN end-user and is denoted $downtime_r$. This leads to:

$$minBW_{n_v^r} = \frac{b_{n_v^r}^{t+1}}{downtime_r} \quad (5)$$

The re-allocation cost of a virtual node is the sum of its re-mapping and migration costs:

- **Re-allocation cost**

$$Cost_{realloc}\left(n_v^r\right) = Cost_{remap}\left(n_v^r\right) + Cost_{mig}\left(n_v^r\right) \quad (6)$$

To satisfy the demand of an evolving node m_v^i for additional resources, the re-allocation of more than one virtual node may be required. The global re-allocation cost $RealloCost_{m_v^i}$ related to an evolving node m_v^i is the sum of all re-allocation costs:

$$RealloCost_{m_v^i} = \sum_{n_v^r \ is \ re-allocated} Cost_{realloc}\left(n_v^r\right) \quad (7)$$

Our objective is to find the best re-allocation scheme that satisfies the evolving node additional resource request while minimizing all re-allocation costs (7) and the average

Table I
KEY NOTATIONS

Notation	Description
Network Model	
G_s	Substrate Network
N_s	Set of substrate nodes n_s
L_s	Set of substrate links l_s
$a_{n_s}^t$	Available capacity of substrate node n_s
$a_{l_s}^t$	Available bandwidth on substrate link l_s
P_φ	Set of loop-free substrate paths φ
a_φ	Available bandwidth associated to φ
$cost(n_s)$	Unit cost of substrate node n_s
$cost(l_s)$	Unit cost of substrate link l_s
Request Model	
G_v^r	Virtual Network r of VN^t
N_v^r	Set of virtual nodes n_v^r of VN G_v^r
L_v^r	Set of virtual links l_v^r of VN G_v^r
$b_{n_v^r}^t$	Minimum required capacity of n_v^r
$b_{l_v^r}^t$	Minimum required bandwidth on l_v^r
$downtime_r$	Maximum downtime imposed by en user VN G_v^r
$\mathbb{S}_{n_v^r}$	Star topology formed by virtual node n_v^r and its connected virtual links
$minBW_{n_v^r}$	Minimum required bandwidth to migrate n_v^r
Mapping Model	
$M_{N_v^r}^t : N_v^r \rightarrow N_s$	Node mapping related to VN G_v^r
$M_{L_v^r}^t : L_v^r \rightarrow P_\varphi$	Link mapping related to VN G_v^r
Measurement of SN	
$s_{l_s}^t$	Stress of substrate link l_s
ALS^t	Average link stress on SN
$OR(l_v^r, l_s)$	Occupancy rate of l_v^r on l_s
$CI^t(l_v^r, l_s)$	Congestion impact of l_v^r on l_s
$ACI^t(l_v^r)$	Average congestion impact of l_v^r
$CI^t(n_v^r)$	congestion impact of virtual node m_v^r

link saturation (2). This leads to the following "***Objective function***":

$$minimize(RealloCost_{m_v^i}, ALS^{t+1}) \tag{8}$$

IV. Heuristic algorithm design

The problem outlined above is a multi-objective optimization problem with conflicting objectives known to be NP-hard [13]. Since we are looking for practical, implementable and scalable solutions, we resort to a heuristic algorithm called Bi-RSforEVN (*Bi-objective Re-allocation Scheme for Evolving Virtual Node request*) to solve it. This heuristic algorithm proceeds in two steps which consist in first selecting the virtual nodes/links should that be re-allocated and then finds the best new hosts for them.

A. First step: Which virtual components should be re-allocated ?

Assume that m_v^i is the evolving node asking for additional resources and that $coloc_h^t$ is the set of all virtual nodes hosted in the same physical node h as m_v^i (i.e. $M_{N_v^r}^t(m_v^i) = h$). In order to satisfy the elasticity request for m_v^i, we will re-allocate one or more co-located virtual nodes to free resources and make room in the hosting substrate node. Recall that we also aim at simultaneously "tidy up" the substrate networks and balance the load. To do so we move (migrate) congestion causing virtual nodes to less saturated substrate nodes (hosts). To identify the virtual nodes causing the congestion, we define a "congestion impact " metric to use as the selection criterion. In fact, we use the notion of occupancy rate $OR(l_v^r, l_s)$ of a virtual link l_v^r passing through a substrate link l_s to evaluate the congestion impact. The occupancy rate $OR(l_v^r, l_s)$ is the ratio of the virtual link l_v^r required bandwidth $b_{l_v^r}^t$ to the total bandwidth of l_s:

$$OR(l_v^r, l_s) = \frac{b_{l_v^r}^t}{a_{l_s}^0} \tag{9}$$

We derive the congestion impact of l_v^r on l_s as the product of its occupancy rate and l_s's stress:

$$CI^t(l_v^r, l_s) = OR(l_v^r, l_s) * s_{l_s}^t \tag{10}$$

CI^t measures the "degree of involvement " of l_v^r in saturating l_s. The average congestion impact of a virtual link is the average of its congestion impacts on all substrate links hosting it:

$$ACI^t(l_v^r) = \frac{1}{|M_{L_v^r}^t|} \sum_{l_s \in M_{L_v^r}^t} CI(l_v^r, l_s) \tag{11}$$

where $|M_{L_v^r}^t|$ is the number of substrate links hosting l_v^r. Since the congestion impact of a virtual node n_v^r is the sum of the congestion impacts of its attached virtual links, we get:

$$CI^t(n_v^r) = \sum_{l_v^r \in \mathbb{S}_{n_v^r}} ACI^t(l_v^r) \tag{12}$$

Algorithm 1 Bi-RSforEVN

```
RequestSatisfied ← false
Step1: Sort nodes in coloc_h^t according to criterion Reem
Step2: n_v^r ← coloc_h^t.pop
if The re-allocation of n_v^r is succeeded then
    if The updated resources in h are sufficient to satisfy the request or n_v^r = the evolving node then
        RequestSatisfied ← true
        Update mapping
    else
        if coloc_h^t is not empty then
            goto 3
        end if
    end if
else
    if coloc_h^t is not empty then
        goto 3
    end if
end if
```

The virtual nodes are selected according to i) their size and QoS requirements, ii) their congestion impact. The size of a virtual node includes its intrinsic size and the aggregate bandwidth of its associated links. The QoS corresponds to the maximum acceptable downtime of the virtual node during migration, and the congestion impact of a virtual node n_v^r is defined by equation (12). To select the virtual nodes for re-allocation, we use a selection metric $Reem$ that ranks the nodes according to their contribution to the overall congestion in a decreasing order. The selection variable $Reem$ is defined as:

$$Reem(n_v^r) = \frac{CI^t(n_v^r)}{(b_{m_v^r}^t + \sum_{l_v^r \in \mathbb{S}_{n_v^r}} b_{l_v^r}^t) * downtime_r} \tag{13}$$

The $Reem$ expression is a fraction composed of three terms. The numerator is the virtual node's congestion impact. The denominator is the product of two terms: one term represents the "size " of the virtual node and the second one is related to the QoS requirements. The purpose behind considering the ranking criterion $Reem\,(n_v^r)$ is threefold. First, we favor re-allocation of candidate virtual nodes and their attached links requiring the smallest amount of resources to minimize the re-mapping cost (3). Second re-allocate in priority the smaller and more QoS degradation tolerant nodes to optimize the migration cost (5) since the amount of bandwidth required to perform task migration will be minimized. And finally, favor the re-allocation of the most congestion causing virtual links (10) by moving (migrating) them to less saturated hosts. As a result of this ranking, all virtual nodes in $coloc_h^t$ are sorted in a list $\vec{coloc}_h^t$ in decreasing order of their $Reem$ value.

B. Second Step: Where these selected components should be re-allocated ?

The aim of this second step is to re-allocate one or more of the virtual nodes having the lowest $Reem$ values to make room in the initial host, and to move their associated links to less saturated physical paths. The amount of requested additional resources by the evolving node dictates the number of virtual nodes that have to be re-allocated. Indeed, the sum of resources to be freed should be greater then this amount. After each node re-allocation, the algorithm checks if enough resources are freed to satisfy m_v^i's new demand, if it is the case, the request is satisfied. Otherwise, the next node in $coloc_h^t$ is selected and the process is repeated until the elasticity request is satisfied or m_v^i is re-allocated as long as $coloc_h^t$ is not empty.

C. Virtual node re-allocation scheme

To re-allocate a virtual node n_v^r, its associated star topology $\mathbb{S}_{n_v^r}$ (the node and its links) should be re-mapped, in order to maintain n_v^r's connectivity with all its peers and resume tasks through migration. In order to minimize the re-allocation cost (7), the new substrate host for the re-allocated node is chosen among the nearest neighbors, $near_h^t$ of the initial host h, that have enough resources and that can reconstruct all the links in $\mathbb{S}_{n_v^r}$. In order to balance the load, these virtual links are re-allocated using the shortest path algorithm, where the weight of each physical link is defined by its stress (1). Among the set $near_h^t$, the selected node is the one minimizing the total re-allocation cost and the average links saturation. The Virtual node re-allocation scheme is illustrated in Algorithm 2.

Algorithm 2 Node re-allocation Scheme

re-allocate($n_v^r, RemapCost_{m_v^i}, MigCost_{m_v^i}$)
$re-allocationResult \leftarrow failure$
$remapCost^{best} \leftarrow \infty, \quad ANS^{best} \leftarrow ANS^t$
Search $near_{n_v^r}^t$
if $near_{n_v^r}^t$ is not empty **then**
  for all $n_s \in near_{n_v^r}^t$ **do**
    map n_v^r in n_s
    for all $l_v^r \in \mathbb{S}_{n_v^r}$ **do**
      re-map virtual link l_v^r onto a substrate path φ using shortest path algorithm
    end for
    if $\mathbb{S}_{n_v^r}$'s mapping succeeds **then**
      $re-allocationResult \leftarrow success$
      if $remapCost(\mathbb{S}_{n_v^r}) * ANS^{t+1} < remapCost^{best} * ANS^{best}$ **then**
        $remapCost^{best} \leftarrow remapCost(\mathbb{S}_{n_v^r})$
        $ANS^{best} \leftarrow ANS^{t+1}$
      end if
    end if
  end for
  if $re-allocationResult = Success$ **then**
    Add $cost_{mig}(n_v^r)$ to $MigCost_{m_v^i}$
    Add $cost_{remap}(n_v^r)$ to $RemapCost_{m_v^i}$
  end if
end if
return $re-allocationResult$

V. Simulation results and evaluation

This section describes the simulation settings used for a performance evaluation of our heuristic algorithm, Bi-RSforEVN, and provides the results of this assessment and of a comparison with relevant prior art. The evaluation focuses on the observed total re-allocation cost and the average link saturation when accepting requests for additional resources.

A. Simulation environment

A custom VN embedding simulator is combined with the GT-ITM tool [14] that generates random topologies of the substrate and VN requests. We adopt similar conditions as in [1], [4] to compare our algorithm with previous work using equivalent settings and scenarios. The SN size is set to 50 nodes and each pair of substrate nodes is randomly connected with probability 0.5. The node resource capacity and link resource capacity are drawn randomly in $[0, 50]$. With no loss of generality, we set the per unit node and link resources costs to 1 unit. The requested VNs contain between 2 and 10 virtual nodes in their topologies with an average connectivity also set to 50%. The virtual node and link resource capacities are random within $[0, 20]$ and $[0, 50]$.

In order to initialize the scenario and start the system from a typical situation, we first map the virtual nodes greedily and follow with the k-shortest path algorithm to map links (we set $k = 5$ and select the longest path). This step leads to suboptimal embedding that can reflect (or emulate) a SN state subject to multiple virtual nodes evolutions. To create a highly dynamic environment and unpredictable states or situations, we select randomly N evolving nodes among the virtual nodes hosted in the SN as nodes that require additional resources. We define r, the ratio of the number of evolving nodes to the total number of virtual nodes in SN (i.e., $r = \frac{N}{|SN|}$). The additional resources requests are expressed using the parameter "Increase Factor" (IF):

$$b_{m_v^i}^{t+1} = IF * \ b_{m_v^i}^t \tag{14}$$

where $b_{m_v^i}^{t+1}$ is the new resource requirement of the evolving node m_v^i.

B. Simulation results

As stated earlier, among previous work [9] [10] [11], [12] dealing with the problem of evolving VN, only the authors of [9] used similar assumptions and objective function to our proposal (**Bi-RSforEVN**). It is consequently more relevant and appropriate to compare performance with their algorithm named **DVNMA_NS**. In addition, to measure the effectiveness of our double objective heuristic algorithm with

Table II
COMPARED ALGORITHMSII

Notation	Re-allocated virtual nodes	Chosen new host	Link re-allocation strategy
DVNMA_NS	The evolving node (systematically)	The most cost effective node among *all* substrate nodes	Shortest path (all weights=1)
RSforEVN	The smallest and more QoS degradation tolerant virtual nodes	The most cost effective node among nearest neighbors	Shortest path (all weights=1)
LB-RSforEVN	Virtual nodes with highest congestion impact	The node leading to minimum ALS among nearest neighbors	Shortest path (weight=link stress)
Bi-RSforEVN	The smallest and more QoS degradation tolerant virtual nodes with highest congestion impact	The most cost effective node leading to minimum ALS among nearest neighbors	Shortest path (weight=link stress)

respect to re-allocation costs and load balancing, we use two variants of **Bi-RSforEVN** each focusing on one and only one objective while neglecting the other one. Namely, **LB-RSforEVN** is a re-allocation scheme that aims exclusively at balancing the SN load at the expense of cost and **RSforEVN** is an algorithm (used in our previous work [1]) that minimizes re-allocation cost but without any regard to the SN state or load. Both **LB-RSforEVN** and **RSforEVN** serve as bounds on performance to benchmark our new algorithm **Bi-RSforEVN** that provides the best possible tradeoff between re-allocation cost and load balancing. Using this bounds, we actually show that the algorithm performance is very close to these bounds. Table II summarizes the characteristics of the compared algorithms.

1) Re-allocation cost for large size evolving virtual nodes

The first simulation assesses the re-allocation cost of our algorithm for evolving virtual nodes of large sizes (Equation 13). To produce scenarios with large virtual nodes instances to re-allocate, 14 ($r = 1/12$) virtual nodes are selected randomly from the top 50 largest virtual nodes currently hosted in the SN among a total of 112 nodes. The re-allocation cost is measured for variable Increase Factors, representing the amount of additional resources that will be required by the 14 selected virtual nodes. Figure 1 depicts the results of 100 averaged runs and indicates that **Bi-RSforEVN** and **RSforEVN** have the lowest re-allocation cost. In fact, these algorithms reduce the re-allocation cost by selecting small virtual nodes as candidates for re-allocation. This also makes them less sensitive and more robust to increasing IF values, as opposed to **DVNMA_NS** that always re-allocates the evolving nodes themselves and induces consequently high

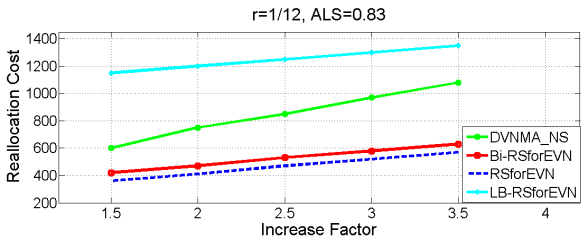

Figure 1. Re-allocation Cost

re-mapping costs when the evolving nodes are of large size. Note that **LB-RSforEVN**has the highest re-allocation cost. In fact, this algorithm selects the virtual nodes to re-allocate regardless of their size and considers only their congestion impact, besides, when re-allocating $\mathbb{S}_{n_v^r}$, the new host is chosen as the one minimizing ALS no matter its re-allocation cost. Our algorithm **Bi-RSforEVN**, as expected, is slightly outperformed by **RSforEVN** that only focuses on minimizing re-allocation cost and disregards the impact of its decisions on the SN load. The marginal difference with this bound, that provides clearly suboptimal solutions, confirms the efficiency of our algorithm in achieving the best re-allocation cost and load tradeoff.

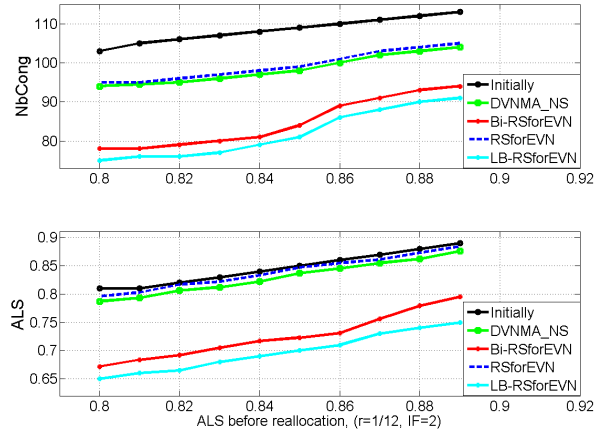

Figure 2. Load balancing

2) Load balancing

A substrate link is called *congested* if its is over stressed regarding the average link stress in SN ($s_{l_s}^t > ALS^t$). The number of congested substrate links is denoted $nbCongested$. We measure ALS and $nbCongested$ observed after re-allocating N evolving nodes (while maintaining $r = 1/12$), for different *initial* ALS values. Figure2 shows that **Bi-RSforEVN** (resp.**LB-RSforEVN**) reduces by 17% (resp. 19%) the average link saturation and 24% (resp. 27%) the number of congested substrate links, leading to

a better load balancing compared to **DVNMA_NS** and **RSforEVN**. The gap is more significant when the SN is slightly saturated, in fact, in such situation these algorithms find more easily less saturated hosts for re-allocated resources as a part of substrate resources is still available. This task is more difficult when the SN is saturated as almost all resources are congested, but they still perform well, reducing the ALS by 11% (resp. 13%) and $nbCongested$ by 17% (resp 19%). Note that **DVNMA_NS** and **RSforEVN** minimize slightly the ALS thanks to the use of the shortest path algorithm, compared to the k-shortest path algorithm in the initial embedding. We also observe that **Bi-RSforEVN** is very close to the bound provided by **LB-RSforEVN** that focuses only on load balancing, showing again the efficiency of **Bi-RSforEVN** in load balancing and re-allocation costs tradeoff.

3) Load balancing Vs re-allocation cost

In this simulation, we aim at measuring the effectiveness of our algorithm in meeting *simultaneously* the objectives of re-allocation cost minimization and load balancing. We measure the ALS observed after accepting all elasticity requests for an increasing number of evolving virtual nodes and note also the total re-allocation cost. Figure 3 shows that, for all algorithms, when the number of re-allocated nodes increases, the total re-allocation cost naturally increases and the ALS decreases, in fact, the more reconfigurations we make, the more opportunity we have to "tidy up " the SN and resolve eventual congestion problems. Algorithm **RSforEVN** realizes the best re-allocation cost but has the worst performance in terms of load balancing. **LB-RSforEVN** is the best load balancing algorithm but is the most costly. **DVNMA_NS** is less costly then **LB-RSforEVN**, but it is outperformed by **Bi-RSforEVN**, and does not reduce significantly the ALS. Our algorithm **Bi-RSforEVN** has good performance in both objectives, in fact it reduces by 46% the ALS (for r=9/20) with a reasonable cost (30% less then **DVNMA_NS**). In conclusion **Bi-RSforEVN** offers the best trade-off between re-allocation cost and load balancing strategy.

VI. Conclusion and Future Works

We study the problem of allocating additional resources for virtual nodes in virtual networks while maximizing the profitability of the substrate networks and propose an algorithm that offers the best trade-off in re-allocation cost and load balancing when compared to prior art. Future work will explore dynamic bandwidth allocation when the resource demand of embedded virtual networks fluctuates according to end users' application needs.

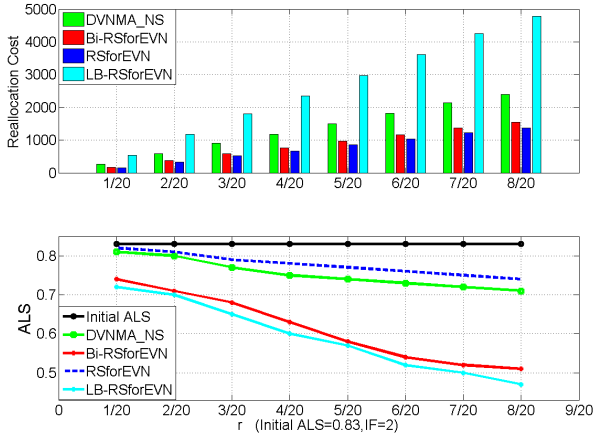

Figure 3. Re-allocation cost Vs Load balancing

References

[1] H. Jmila, I. Houidi, and D. Zeghlache, "Rsforevn: Node reallocation algorithm for virtual networks adaptation," *19th IEEE Symposium on Computers and Communications (IEEE ISCC 2014)*.

[2] Y. Zhu and M. H. Ammar, "Algorithms for assigning substrate network resources to virtual network components." in *INFOCOM*, 2006.

[3] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: Substrate support for path splitting and migration," *SIGCOMM Comput. Commun. Rev.*, 2008.

[4] M. Chowdhury, M. Rahman, and R. Boutaba, "Vineyard: Virtual network embedding algorithms with coordinated node and link mapping," *Networking, IEEE/ACM Transactions on*, 2012.

[5] J. F. Botero, X. Hesselbach, M. Duelli, D. Schlosser, A. Fischer, and H. de Meer, "Energy efficient virtual network embedding." *IEEE Communications Letters*, 2012.

[6] N. Farooq Butt, M. Chowdhury, and R. Boutaba, "Topology-awareness and reoptimization mechanism for virtual network embedding," in *NETWORKING 2010*, 2010.

[7] P. N. Tran, L. Casucci, and A. Timm-Giel, "Optimal mapping of virtual networks considering reactive reconfiguration," in *CLOUDNET, 2012*.

[8] P. Tran and A. Timm-Giel, "Reconfiguration of virtual network mapping considering service disruption," in *ICC, 2013*, 2013.

[9] G. Sun, H. Yu, V. Anand, and L. Li, "A cost efficient framework and algorithm for embedding dynamic virtual network requests," *Future Generation Comp. Syst.*, 2013.

[10] Y. Zhou, X. Yang, Y. Li, D. Jin, L. Su, and L. Zeng, "Incremental re-embedding scheme for evolving virtual network requests," *Communications Letters, IEEE*, 2013.

[11] A. Blenk and W. Kellerer, "Traffic pattern based virtual network embedding," in *Proceedings of the 2013 Workshop on Student Workhop*, 2013.

[12] R. Mijumbi, J.-L. Gorricho, J. Serrat, M. Claeysy, F. D. Turcky, and S. Latr, "Design and evaluation of learning algorithms for dynamic resource management in virtual networks," *NOMS 2014*, 2014.

[13] Z. Cai, F. Liu, N. Xiao, Q. Liu, and Z. Wang, "Virtual network embedding for evolving networks." in *GLOBECOM*, 2010.

[14] E. Zegura, K. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *INFOCOM*, 1996.