RSforEVN: Node Reallocation Algorithm for Virtual Networks Adaptation

Houda Jmila, Ines Houidi, Djamal Zeghlache

RSforEVN: Node Reallocation Algorithm for Virtual Networks Adaptation

Houda Jmila*, Ines Houidi† Djamal Zeghlache*
* *Institut Mines Telecom, Telecom SudParis and UMR*5157 *of CNRS, Evry, France*
{*houda.jmila, djamal.zeghlache*}*@telecom-sudparis.eu*
†*RedCAD Lab, Sfax University, National Engineering School of Sfax, Tunisia*
{*ines.houidi,wajdi.louati*}*@enis.rnu.tn*

***Abstract*—This paper addresses the dynamic adaptation of already embedded virtual network (VN) resources to respond to increasing network services demands and load on network nodes. The proposed algorithm focuses on the virtual nodes, of the embedded VN, requiring more resources. The adaptation scheme goes beyond the reallocation of the virtual nodes by considering their topological neighborhood. The proposed algorithm outperforms existing approaches in reallocation cost and in execution time (or convergence time) for larger graphs.**

***Keywords*-Virtual Network Embedding, Node reallocation, Cloud, NaaS, QoS;**

I. Introduction

Network Virtualization (NV) allows multiple virtual networks (VNs) to co-habit on shared physical networks (SNs, often referred as substrate network or network infrastructure). The current state of the art has extensively addressed the static allocation and placement problem of virtual resources (nodes and links) in physical hosts (nodes and substrate paths) but Cloud services and environments require the dynamic allocation of resources (elasticity services) according to applications and user demand resource requirements. Beyond the traditional Infrastructure/Platform/Software as a Service offers (IaaS/PaaS/SaaS), the need to also provide Networks and Networking as a a service has become essential (NaaS) [1].

This paper addresses this dynamic allocation of resources issue, especially of virtual networks, to support cloud services according to varying applications and user resource requirements. More specifically the paper focuses on virtual nodes of already embedded VNs when more resources are required from the hosting physical machine or node. The need for more resources may have multiple reasons such as increasing applications requirements, the need to maintain quality of service of multiple services sharing the same physical nodes, etc. Reacting to these dynamic changes and growing needs may require allocation of additional resources from the hosts themselves when feasible, or the reallocation and optimal reshuffling of virtual resources across physical nodes or hosts. When hosts do not have enough resources, prior work on Virtual Networks Embedding (VNE: that consists of mapping virtual resources onto physical resources in terms of nodes and links) [2], [3] move the virtual nodes requiring more resources to other physical nodes to maintain the service. This affects the active application or service running in the virtual resource. The service will experience a downtime or unavailability period that needs to be taken into account and minimized [4]. In real situations, VN users often impose Service Level Agreements with penalties for service disruptions caused by migration (e.g. penalty imposed to Amazon EC2 for violating VM availability SLA) [5] [6]. Avoiding such disruptions and penalties are essential. The migration of the virtual resource will also induce load on the physical network links proportionally to the size of the migrated virtual resource. In order to minimize these impacts, we propose to select the virtual nodes in the affected physical node that will incur the lowest cost and load during migration. Virtual resources that are intuitively candidates for such migration are those that are tolerant to disruptions and/or are of small size since the migration will be faster and will induce less load. When making migration decisions, the selected virtual resource connectivity has to be taken into account since it has to be maintained, actually all the links associated with the selected virtual resource have to be re-established.

The paper starts with the related work on virtual network embedding in Section II when a need for additional resources arises. Section III describes and formulates the problem. Section IV presents our proposed heuristic algorithm to achieve minimum cost and service interruption when additional resources are required. Performance evaluation of the proposed heuristic algorithm is presented and compared to prior art in sections V and VI.

II. Related work

Authors in [2], [3], [7] addressed the problem of evolving resource requests in VN embedding with [2] listing four VN evolution cases: i) adding new nodes and links to an ongoing VN allocation ii) deleting no longer needed resources when services end iii) releasing resources when a task requires less resources to run iv) requesting more resources when VN nodes or/and links require more resources at specific stages of an application lifetime. The Authors optimally reconfigure the evolving VNs using a Mixed Integer Problem formulation with the objective of

minimizing the reconfiguration cost. Since the problem is NP-hard, they suggest heuristic algorithms to deal with each case to avoid exponential explosion. They unfortunately consider exhaustively all mapping combinations to adapt a virtual node by evaluating the cost for each substrate node and select finally the most effective one. This strategy is not suitable for large physical networks and can not meet the *swift and rapid* adaptation required by dynamic cloud applications and services.

In [3], authors propose an incremental re-embedding scheme for evolving VNs requirements relying on the notion of physical resource migration on nodes reported in [8] that distributes virtual resources across multiple interconnected physical resources. Their objective is to reduce the number of virtual nodes or resources that are reallocated but this leads to increased interconnection bandwidth usage that limits the acceptance ratio of new requests. They do not minimize the per-node reallocation. They minimize only the *number* of reallocated nodes.

In [7], authors address VN reconfiguration when the VN's resource requirements change according to services traffic patterns. Considering predictable traffic patterns, they propose an embedding algorithm that reduces the number of link migrations while achieving an acceptable load balancing over substrate links. They unfortunately reallocate virtual nodes randomly and focus only on virtual link reallocation.

In our case, we focus as mentioned on virtual nodes, of already embedded VNs, requiring more resources, and reduce both the per-node reallocation cost and the number of reallocated nodes compared to [3]. In addition, we take into account jointly the virtual node and its links as opposed to [7] that concentrates on links. Finally, unlike most of node reallocation solutions proposed in the VNE literature [9], [10] where the task migration phase is ignored, our approach minimizes service interruption during migration.

III. Problem formulation

This section presents a mathematical model to allocate additional resources to active VNs hosted by shared infrastructures (or SNs). Fulfilling the requests for more resources can be achieved by moving out of the physical host only the concerned virtual nodes themselves or by migrating other virtual nodes to other hosts. The goal is to derive from the model an objective function that will realize the re-allocation of virtual nodes at minimum overall adaptation cost. Re-mapping and migration costs, downtime and optimization performance need to be taken into account in the derivation.

A. Network Model

The 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 and L_s is the set of *substrate links* l_s. Graph G_s is used to represent the substrate. Let $a^t_{n_s}$ denote the *available* capacity of node n_s (typically CPU and memory) and $a^t_{l_s}$ the *available* bandwidth on link l_s. Variable φ represents a substrate path (a single or a sequence of substrate links) between two substrate nodes. Variable P_φ is the set of loop-free substrate paths. The available bandwidth a_φ associated to a substrate path φ can be evaluated as the smallest available bandwidth on the links along the substrate path.

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
$a^t_{n_s}$	Available capacity of substrate node n_s
$a^t_{l_s}$	Available bandwidth on substrate link l_s
P_φ	Set of loop-free substrate paths φ
a_φ	Available bandwidth associated to a substrate path φ
$cost(n_s)$	Unit cost of substrate node n_s
$cost(l_s)$	Unit cost of substrate link l_s
Request Model	
G^r_v	Virtual Network r of VN^t
N^r_v	Set of virtual nodes n^r_v of VN G^r_v
L^r_v	Set of virtual links l^r_v of VN G^r_v
$b^t_{n^r_v}$	Minimum required capacity of virtual node n^r_v
$b^t_{l^r_v}$	Minimum required bandwidth on virtual link l^r_v
$downtime_r$	Maximum downtime imposed for n^r_v
$\mathbb{S}_{n^r_v}$	Star topology formed by n^r_v and its connected links
$minBW_{n^r_v}$	Minimum required bandwidth to migrate n^r_v
Mapping Model	
$M^t_{N^r_v} : N^r_v \rightarrow N_s$	Node mapping related to VN G^r_v
$M^t_{L^r_v} : L^r_v \rightarrow P_\varphi$	Link mapping related to VN G^r_v

B. VN resource Request Model

Since a VN request is composed of *virtual nodes* interconnected via *virtual links*, the VN request topology can be represented by a weighted undirected graph $G_v = (N_v, L_v)$, where N_v is the set of required virtual nodes and L_v is the set of required virtual links. Each virtual node $n_v \in N_v$ is associated with a minimum required capacity $b^t_{n_v}$. Each virtual link $l_v \in L_v$ is associated with a minimum required bandwidth $b^t_{l_v}$. The set of active VNs on G_s at time t is defined as VN^t and the evolving node (requiring more resources) is represented by m^i_v with $i \in VN^t$ and with a new resource requirement $b^{t+1}_{m^i_v} > b^t_{m^i_v}$.

C. VN Mapping Model

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

D. Problem formulation

1) Reallocation strategy: When an evolving node m^i_v requiring additional resources and a substrate host h with $M^t_{N^r_v}(m^i_v) = h$ has insufficient resources a strategy for re-allocation of resources is needed to maintain the service. This may require a migration of the virtual node or other

nodes in the host. A trivial and suboptimal strategy is to move the evolving node to another less loaded host. A more elaborate strategy should take into account multiple criteria such as migration and re-mapping costs. We accordingly adopt a strategy where we reorganize and redistribute virtual nodes in the initial host and its neighbors while minimizing overall re-allocation cost. Intuitively, the nodes inducing the smallest migration cost and disruptions should be selected in priority to find a good solution.

2) Optimization objective: With this strategy in mind, we implement the virtual node re-allocation in two phases: *Re-mapping and Migration*.

The *Re-mapping (remap)* phase consists in finding alternative substrate resources to host the reallocated components. The virtual node would be remapped onto another substrate node found to have enough available resources. The links associated to the original (or source) virtual node will be also remapped to restore connectivity with the new hosting (destination) node. Secondly, the Migration phase ($migrate$) will move tasks or jobs previously running on the source virtual node onto the selected destination virtual node to resume tasks. Moving tasks requires the establishment of a temporary connection between the old and new hosts to support task migration. This induces a transfer cost that we take into account in the reallocation cost assessment. The resource re-allocation incurs both a re-mapping cost $Cost_{remap}$ and a migration cost $Cost_{mig}$.

Re-mapping cost:

Similar to previous work in [2], the mapping/re-mapping cost of a VN request is equal to the sum of the costs of allocating/re-allocating its virtual nodes and links from the data center or infrastructure resources (physical nodes and substrate paths). Let $cost\left(n_s\right)$ (and $cost\left(l_s\right)$) be the cost unit of substrate node (and substrate link) respectively. Let $n_v^r \in N_v^r, r \in VN^t$ denote a virtual node selected to be re-allocated related to request r and let $\mathbb{S}_{n_v^r}$ represent the star topology formed by n_v^r and its connected virtual links. We define the cost of re-mapping n_v^r as the sum of total substrate resources reallocated to the node n_v^r and its attached virtual links. Formally:

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

Where $(M_{N_v^r}^{t+1}, M_{L_v^r}^{t+1})$ describes the mapping of re-allocated elements.

Migration Cost: During the migration step, migrated tasks experience a downtime that depends on i) the migration technique [4], ii) the size of the migrated task and iii) the bandwidth allocated for task migration. Migration is a topic on its own that is beyond the scope of this paper. For our study, we consider that the downtime depends primarily on the size of the migrated task and the bandwidth available during the migration. In our model, a maximum downtime for each virtual node n_v^r, $downtime_r$, is imposed by each VN end-user. To respect this condition, sufficient resources should be allocated from the target host depending on the size of the task to migrate. Formally, we define $minBW_{n_v^r}$ as the minimum required bandwidth to migrate a virtual node n_v^r :

$$minBW_{n_v^r} = \frac{b_{n_v^r}^{t+1}}{downtime_r} \quad (2)$$

Where $b_{n_v^r}^{t+1}$ is the size of the re-allocated virtual node. The cost of task migration $cost_{mig}\left(n_v^r\right)$ is the sum of all resources allocated (needed) for migration. Formally, if $p_{mig(n_v^r)} \in P_\varphi$ denotes the substrate path used for migrating the node n_v^r, the migration cost is defined as:

$$cost_{mig}\left(n_v^r\right) = \sum_{l_s \in p_{mig(n_v^r)}} minBW_{n_v^r} * cost(l_s) \quad (3)$$

Reallocation cost

Finally, the reallocation cost of a virtual node is the sum of its re-mapping cost and its migration cost:

$$Cost_{realloc}\left(n_v^r\right) = Cost_{remap}\left(n_v^r\right) + Cost_{mig}\left(n_v^r\right) \quad (4)$$

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:

$$RealloCost_{m_v^i} = \sum_{n_v^r \ isReallocated} Cost_{realloc}\left(n_v^r\right) \quad (5)$$

Our objective is to find the best re-allocation scheme in order to satisfy the evolving node additional resource request while minimizing all re-allocation costs. This leads to the following objective function:

Objective function:

$$minimize(RealloCost_{m_v^i}) \quad (6)$$

IV. Heuristic algorithm design

Finding the optimal re-allocation for an evolving node while minimizing cost is NP-Hard [11]. We resort to a heuristic algorithm called RSforEVN (*Re-allocation Scheme for Evolving Virtual Node request*) to reduce complexity, improve convergence times and to provide a scalable solution. The heuristic algorithm must decide which virtual nodes to reallocate and where to move them. This reorganization should incur minimum overall reallocation and migration cost. The heuristic algorithm proceeds in two optimization steps. It first finds the best set of virtual nodes to reallocate and migrate to free resources for the benefit of the evolving node (unless the best solution is to move the evolving node itself, in which case the objective is to find a new host for it). In this step, the heuristic algorithm selects the minimum

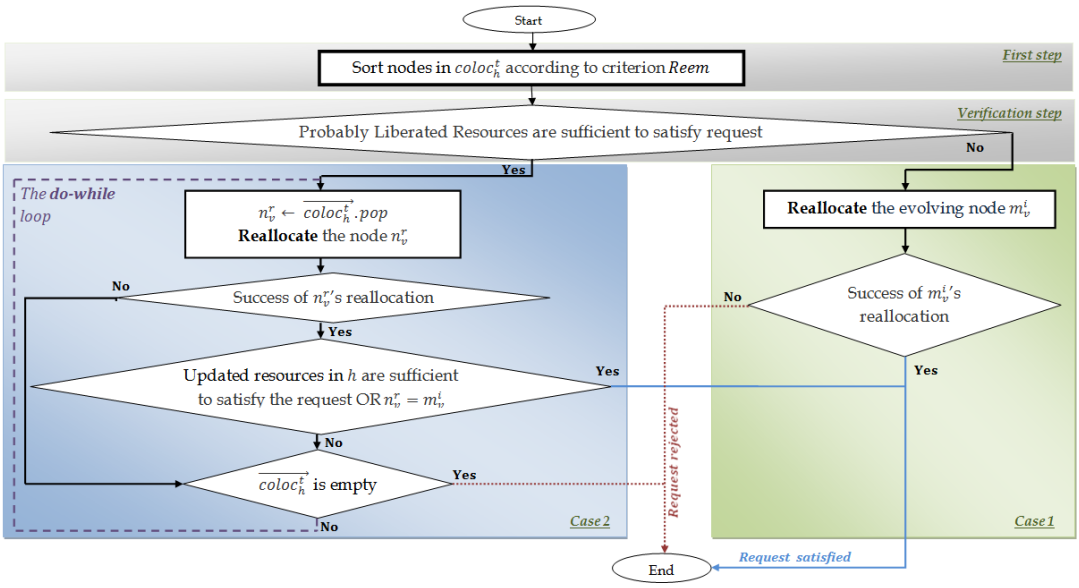

Figure 1. Main Algorithm steps

number of less constraining virtual nodes (typically of small sizes and most tolerant to disruptions and QoS degradations). The second step consists of finding the best destination or target hosts for the selected virtual nodes. The heuristic algorithm will have to map efficiently nodes and links to meet the the minimum reallocation and migration cost objective. The two steps are described in more detail in the sequel

A. First step: Selection of virtual nodes for reallocation

We use the following notations to describes the selection process: m_v^i identifies the evolving node asking for additional resources and $coloc_h^t$ the set of all virtual nodes hosted in the same physical node h as m_v^i (i.e. $M_{N_v^r}^t(m_v^i) = h$). The heuristic algorithm main idea is to re-allocate one or more co-located virtual nodes from the substrate node, hosting the evolving node, to free resources (or make room) for the evolving node (needing additional resources). The virtual nodes are selected according to their size and QoS requirements. 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:

$$Reem(n_v^r) = (b_{m_v^r}^t + \sum_{l_v^r \in \mathbb{S}_{n_v^r}} b_{l_v^r}^t) * downtime_r \qquad (7)$$

Hence, the $Reem$ expression is the product of two terms: the first term represents the "size" of the virtual node, whereas the second one is related to QoS requirements. The purpose behind considering the ranking criterion $Reem\,(n_v^r)$ is twofold. First favor reallocation of candidate virtual nodes and their attached links that require the smallest amount of resources to minimize re-mapping cost (1). Second re-allocate the smaller and more QoS degradation tolerant nodes to optimize the migration cost (3) since the amount of bandwidth required to perform task migration will be minimized. As a result of this ranking, all virtual nodes in $coloc_h^t$ are sorted in a list $\overrightarrow{coloc_h^t}$ in increasing order of their $Reem$ value.

B. Second Step: finding the best physical hosts for resources selected for reallocation

The next step consists in re-allocating virtual nodes that have the lowest $Reem$ values. Thus, one or more virtual nodes from the ranked list $\overrightarrow{coloc_h^t}$ should be re-allocated with their associated virtual links. The number of virtual nodes to re-allocate is dictated by the amount of requested additional resources by the evolving nodes. The sum of resources to free by migrating virtual nodes should be equal or greater than the amount of required additional resources for the evolving node. Virtual nodes will not be migrated if there are enough resources in the original physical node since the evolving node receive additional resource directly from its host. When remaining resources are insufficient, co-located virtual nodes will be migrated to free the needed resources for the evolving node. If virtual nodes can not be migrated for QoS reasons, the evolving node will be moved if possible otherwise the request is rejected. In fact, this will be the case each time all virtual nodes ranked ahead of the evolving node in $\overrightarrow{coloc_h^t}$ vector can not offer enough resources to satisfy the evolving node. As presented in Figure 1 our proposed algorithm takes into account two special cases:

i) If the amount of resources that could be freed after multiple re-allocations is not sufficient to satisfy the request,

our algorithm tries to re-allocate the evolving node. If the re-allocation succeeds, the request is satisfied, otherwise it is rejected.

ii) Else, our proposal remaps the first node in the ranked list. If it succeeds, the algorithm verifies if the resources released after this re-allocation are sufficient to satisfy m_v^i's new demand, if it is the case, the elasticity request is satisfied. Otherwise, the next node is selected and the process is repeated until the elasticity request is satisfied or the evolving node is re-allocated, as long as $\vec{coloc}_h^t$ is not empty (The do-while loop in Figure 1.

C. Virtual node reallocation scheme

After selecting virtual nodes for reallocation, the algorithm has to find the optimal nodes to host these selected virtual nodes and restore their connectivity with all their peers (pervious neighbors) by finding new substrate paths to restore all the broken links. To reallocate a virtual node n_v^r, the star $\mathbb{S}_{n_v^r}$ (the node and its links) should be re-mapped, and task migration should be performed. To find the best new substrate hosts, our heuristic algorithm explores the nearest neighbors of the initial host h to find nodes that have enough resources and can reconstruct all the links associated with each virtual node candidate to migration. Links must also be established to ensure migration respecting the downtime constraints of each virtual node. If $near_h^t$ is the set of potential (candidate) hosts for the re-allocated node, this neighbor set $near_h^t$ has to minimize migration cost (3). The shortest path algorithm is used to find the optimal substrate paths. The Virtual node reallocation scheme is illustrated is Procedure1

Procedure1:One Node Reallocation steps
Reallocate($n_v^r, RealoCost_{m_v^i}$)
$ReallocationResult \leftarrow failure$
$remapCost^{best} \leftarrow \infty$
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**
      $ReallocationResult \leftarrow success$
      if $remapCost\left(\mathbb{S}_{n_v^r}\right) < remapCost^{best}$ **then**
        $remapCost^{best} \leftarrow remapCost(\mathbb{S}_{n_v^r})$
      end if
    end if
  end for
  if $ReallocationResult = Success$ **then**
    Add $cost_{mig}\left(n_v^r\right) + cost_{remap}\left(n_v^r\right)$ to $RealoCost_{m_v^i}$
  end if
end if
return $ReallocationResult$

V. Simulation results and evaluation

We compare our algorithm with relevant prior art to assess performance with a focus on re-mapping and migration costs, execution time (or convergence time) and the acceptance rate of requests for additional resources. We also describe the settings, conditions and scenarios used to conduct the evaluation.

A. Simulation environment

The GT-ITM [13] tool is used to generate random topologies of the substrate and VN networks. Similar parameter settings and simulation conditions to existing work was adopted to be able to compare in equivalent scenarios the performance of our algorithm [12] [10]. The SN (Substrate Network) size is set to 50 nodes and each pair of substrate nodes is randomly connected with probability 0.5 (a realistic value for typical deployed and operational networks, since they are seldom fully meshed and often have connectivity below 50%).The node resource capacity and edge resource capacity are real numbers uniformly distributed between 0 and 50 in order to span reasonably the search without making any specific assumption on the statistical characteristic of this parameter. Without loss of generality, we set the per unit node and edge resources costs to 1 (one)unit. The requested VNs have between 2 and 10 virtual nodes in their topologies with an average connectivity also set to 50%. The node resource capacity is uniformly distributed between 0 and 20 and the edge resource capacity is uniformly distributed between 0 and 50.

In order to initialize the scenario and start the system from a typical situation we map the virtual nodes greedily and follow with the shortest path algorithm to map edges. This step leads to suboptimal embedding that can reflect or mimic the state of a SN subject to multiple virtual nodes evolutions.

To create a highly dynamic environment and unpredictable states or situations, we select randomly N virtual nodes among those hosted by the SN as nodes that require additional resources. The increasing resource requests are measured using the parameter "Increase Factor" (IF):

$$b_{m_v^i}^{t+1} = IF * \ b_{m_v^i}^t \tag{8}$$

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

B. Simulation results

Only [2] [3] [7] deal with the problem of evolving VN. Since the objective functions in [3] [7] differ and are not sufficiently close to our proposed algorithm, we do not retain them for performance comparison. In [3] authors minimize

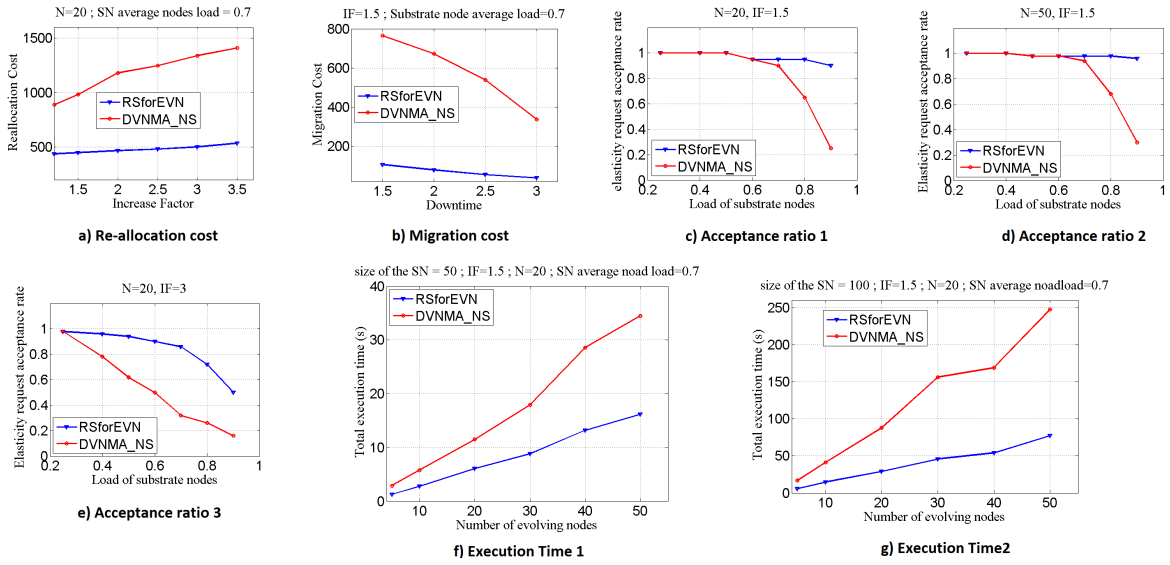

Figure 2. Simulation Results

the number of re-allocated virtual nodes, while in [7] authors minimize the number of virtual link reconfigurations after a VN evolves. The authors of [2] considered the same objective function as that of our proposal and it is more relevant and appropriate to compare performance with their algorithm named DVNMA_NS. The algorithms compared in our simulations are listed in Table II.

Table II
COMPARED ALGORITHMS

Notation	Algorithm description
RSforEVN	Makes a convenient choice of virtual nodes to re-allocate and selects the most cost effective new host among nearest neighbors
DVNMA_NS	Systematically re-allocates the evolving node, and selects the most cost effective new host among *all* substrate nodes

In the simulations, the following performance metrics are used: 1)**Re-allocation Cost**, that reports $RealloCost$ of all evolving nodes if their new demands are successfully satisfied. 2)**Migration Cost** measuring the amount of resource (bandwidth) required to achieve task migrations to fulfill the evolving node requests. This corresponds to the sum of all $cost_{mig}$ of re-allocated nodes. 3)**Acceptance ratio of elasticity requests** that measures the percentage of accepted additional resources requests for evolving nodes 4)**Total execution time (or convergence time)** that measure the algorithms convergence time to assess how fast the algorithms find a solution to fulfill the additional resource requests.

All reported results are obtained by averaging the collected performance from 100 independent runs for each simulation point.

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 7). To produce scenarios with large virtual nodes instances to re-allocate, 20 virtual nodes are selected randomly from the top 100 largest virtual nodes currently hosted in the SN among a total of 214 nodes. The reallocation cost is measured for variable Increase Factors, representing the amount of additional resources that will be required by the 20 selected virtual nodes.

Fig2.a depicts the results of 100 averaged runs and indicates that our algorithm (RSforEVN) outperforms the DVNMA_NS algorithm in terms of re-allocation cost by 50%. Our algorithm reduces the re-allocation cost by selecting primarily small virtual nodes as candidates before resorting to re-mapping virtual nodes of large sizes. This also makes our algorithm less sensitive and more robust to increasing IF values while DVNMA_NS re-allocation cost increases significantly for increasing IF values. The RSforEVN algorithm always selects the smallest virtual nodes first as opposed to the DVNMA_NS always re-allocates the evolving nodes themselves and this induces high re-mapping costs when the evolving nodes are of large size.

2) Migration cost: As depicted in Fig2.b, our algorithm, RSforEVN, performs also much better than DVMA_NS, in migration cost as a function of downtime tolerance of the virtual nodes. Without loss of generality, we assumed that all virtual nodes have the same downtime in the simulations. This is again due to the small virtual nodes selected by RSforEVN since these smaller nodes require less bandwidth for migration according to the downtime constraint. In addition, RSforEVN selects the nearest neighbors to the substrate node hosting the evolving nodes that require more resources

whereas DVNMA_NS searches for the best new hosting node in the entire substrate network and has to do so for the evolving nodes inducing high penalty and even higher cost if the evolving nodes are large. Once the best node is found, DVNMA_NS *deduces* the migration substrate path using the shortest path algorithm. Migration cost increases for both algorithms when the downtime migration constraints become tighter as more link resources (bandwidth) are needed (is needed) to achieve faster migration.

3) Elasticity Request Acceptance ratio benefits for saturated SN: The next set of simulations address the performance of the algorithms with respect to the acceptance ratio of requests for additional resources and their speed in finding solutions (or execution/convergence time) to fulfill such requests for more resources. The evaluation is conducted for several scenarios as a function of the number of involved evolving nodes, the Increase factor that measures the amount of requested additional resources and the load in the substrate network or the SNs.

Figures 2.c and 2.d show close performance in percentage of accepted requests for both algorithms when the substrate network is not heavily loaded. However, when the substrate network is saturated our algorithm accepts 3 times more requests than DVNMA_NS that has difficulty in finding hosts available for large evolving virtual nodes. RSforEVN that moves smaller virtual nodes can instead find more easily some space available in new hosts for these small resource requests. Figure 2.e confirms that RSforEVN outperforms DVNMA_NS when the required amount of additional resources increases with the RSforEVN algorithm resisting much better the increased stress for $IF = 3$ compared to $IF = 1.5$ (looking at Fig 2.c and Fig 2.d joinlty). The acceptance rate for RSforEVN degrades smoothly while that of DVNMA_NS is more significant and rather abrupt. RSforEVN performs consistently better for overloaded substrate networks.

4) Reduced execution time, especially for large Substrate Networks: The convergence time of the algorithm also matters in terms of swift response to additional resources requests since some applications require elasticity services and high availability and can thus put very stringent requirements on extended resource allocations. Figure 2.f and 2.g present the collected required time to find a solution for the resource requests for both algorithms and depict better performance in convergence time for the RSforEVN algorithm that finds solutions 2 to 3 times faster for the simulated scenarios with increasing number of substrate and evolving nodes. Figures 2.f and 2.g corresponding to $SN = 50$ and $SN = 100$ respectively for involved virtual nodes ranging from 5 to 50 nodes. This gap in speed performance for DVNMA_NS is expected as it searches for new hosting nodes amongst all substrate nodes while SFforEVN searches only in the vicinity or neighborhood of the host currently hosting the evolving nodes. RSforEVN does in addition favors migration of smaller virtual nodes. In fact when analyzing all the performance results for the simulated scenarios and settings, RSforEVN performs consistently better and provides the best trade-offs in reallocation cost, migration cost, downtime and speed of convergence.

VI. Conclusion and future works

This paper addresses the allocation of additional resources for virtual nodes in virtual networks provided by shared substrate networks and proposes an algorithm that offers the best trade-off in terms of re-mapping and migration costs, service downtime and convergence speed when compared to prior art. The performance of the proposed algorithm, RSforEVN, is compared to the DVNMA_NS and shown to be consistently superior in all reported performance metrics. Future work will explore simultaneous or concurrent requests for additional resources from multiple virtual networks to address sub-graphs rather than nodes only.

References

[1] P. Costa, M. Migliavacca, P. Pietzuch, and A. L. Wolf, "Naas: Network-as-a-service in the cloud," in *Hot-ICE*, 2012.

[2] 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.

[3] 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.

[4] D. Kapil, E. Pilli, and R. Joshi, "Live virtual machine migration techniques: Survey and research challenges," in *IACC*, 2013.

[5] M. Zhani, Q. Zhang, G. Simon, and R. Boutaba, "Vdc planner: Dynamic migration-aware virtual data center embedding for clouds," in *IM*, 2013.

[6] [Online]. Available: http://aws.amazon.com/fr/ec2/

[7] A. Blenk and W. Kellerer, "Traffic pattern based virtual network embedding," in *Student Workshop*, 2013.

[8] Y. Zhou, Y. Li, D. Jin, L. Su, and L. Zeng, "A virtual network embedding scheme with two-stage node mapping based on physical resource migration," in *ICCS*, 2010.

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

[10] I. Fajjari, N. Aitsaadi, G. Pujolle, and H. Zimmermann, "Vnr algorithm: A greedy approach for virtual networks reconfigurations," in *GLOBECOM*. IEEE, 2011, pp. 1–6.

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

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

[13] E. Zegura, K. Calvert, and S. Bhattacharjee, How to model an internetwork, in Proc. IEEE INFOCOM, 1996.