Computer Communications 84 (2016) 12–24

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Novel adaptive virtual network embedding algorithm for Cloud's private backbone network

Ilhem Fajjari[a], Nadjib Aitsaadi[b,*], Boutheina Dab[b], Guy Pujolle[c]

[a] Orange Labs, 38-40, rue du General Leclerc, 92130 Issy-les-Moulineaux, France
[b] LiSSi - University of Paris Est Creteil (UPEC): 122, rue Paul Armangot, 94400 Vitry Sur Seine, France
[c] Sorbonne Universites, UPMC Univ Paris 06, CNRS, LIP6 UMR 7606: 4 place Jussieu Paris 75005, France

ARTICLE INFO

Article history:
Received 11 September 2015
Revised 5 February 2016
Accepted 14 March 2016
Available online 22 March 2016

Keywords:
Cloud computing
NaaS
Service provisioning
Optimization
K-supplier algorithm

ABSTRACT

In this paper, we study the adaptive virtual network embedding problem within Cloud's backbone. The main idea is to take profit from the unused bandwidth but allocated to virtual networks. Consequently, the acceptance rate of new clients will be maximized. However, the congestion rate of virtual links must be minimized in order to maximize the satisfaction of end-users. To do so, first we formulate the problem as *K*-supplier optimization problem. Then, we propose a novel virtual network embedding strategy denoted by `Adaptive-VNE`. It is based on the approximation-algorithm for bottleneck problems and backtracking strategy. The proposal is validated by simulations and experimental testbed. The results obtained show that `Adaptive-VNE` outperforms the most prominent strategies and reaches a good performance.

1. Introduction

Cloud computing is seen as a tremendous innovation and has attracted both academics and industrials. The global market for Cloud equipment will reach 79.1 billions by 2018[1]. Such success has leading to a massive usage of Cloud's physical resources. For example, CISCO forecasts that Cloud network traffic will grow more than threefold by 2017. In fact, the volume of traffic will progress from 2.6 zettabytes in 2012 to 7.7 zettabytes annually in 2017[2].

Whereas with the rapidly increasing number of clients, Cloud computing becomes victim of its own success. Unfortunately, CloudÇÖ popularity often outgrows the physical infrastructure which brings new challenges to Cloud Providers ($\mathcal{CP}$). To tackle this problem, $\mathcal{CP}$ must possess efficient techniques to supply clients with their required computational resources over scalable networks. An optimal and fast provisioning algorithm is fundamental to achieve the above objective. $\mathcal{CP}$s need to minimize their provision cost whilst guaranteeing the requested Service Level Agreements (SLA) of clients.

In this context, one of the main enabling technology for Cloud computing is virtualization. Thanks to it, physical resources can be shared by various independent applications hosted within data centers and Cloud's backbone. Such a hardware reuse delivers reduction in capital expenditure. Moreover, virtualization offers elastic scalability by dynamically calibrating (i.e., increasing and/or decreasing) the capacity of virtual resources. Unfortunately, the flexibility of virtualization has a cost and add complexity on resource provisioning stage within the physical Cloud infrastructure. Note that the resource provisioning process is considered as one of the most challenging issue to deal with in Cloud computing. Indeed, selecting the *most suitable* physical resources for any service while i) *guaranteeing* the QoS requirements (e.g., CPU, memory, bandwidth, etc.) and ii) *maximizing* the $\mathcal{CP}$'s revenue is complex and on which Cloud's success depends on. In this paper, we address the resource provisioning problem within Cloud's backbone. Our objective is to ensure the best embedding of Virtual Networks ($\mathcal{VN}$s) within the Substrate Network ($\mathcal{SN}$) of Cloud's backbone. In other words, we focus on the resource allocation problem of Networks as a Service (NaaS). The problematic of adaptive resource provisioning of bandwidth within Cloud's backbone network is also considered. In order to supply dedicated $\mathcal{VN}$ to an end-user typified with a customized traffic, the resource provisioning algorithm aims to guarantee an efficient and flexible share among all the instantiated $\mathcal{VN}$s upon the underlying $\mathcal{SN}$.

To reach our objective, we propose a new Adaptive Virtual Network Embedding algorithm (`Adaptive-VNE`[3]). It makes use of

* Corresponding author. Tel.: +33 141807310.
E-mail addresses: ilhem.fajjari@orange.com (I. Fajjari), nadjib.aitsaadi@u-pec.fr (N. Aitsaadi), boutheina.dab@u-pec.fr (B. Dab), guy.pujolle@lip6.fr (G. Pujolle).

[1] http://www.forbes.com/fdc/welcome_mjx.shtml.
[2] http://www.cisco.com/.

[3] Preliminary results are published in IEEE GLOBECOM 2012.

http://dx.doi.org/10.1016/j.comcom.2016.03.019
0140-3664/

"divide and conquer" strategy. The main idea behind our proposal is to i) take advantage of unused reserved bandwidth and ii) allocate them to $\mathcal{VN}$ demanding additional resources. Adaptive-VNE proceeds as following. First, the $\mathcal{VN}$ topology is subdivided into a set of star topologies called Solution Components $\mathcal{SC}s$. Afterwards, each $\mathcal{SC}$ embedding is formulated as *K*-supplier problem which is resolved based on the approximation-algorithm for bottleneck problems [1]. In fact, the solution consists of finding the best candidates to host the central virtual router of each $\mathcal{SC}$. The calculated candidates are sorted with respect to a proposed cost metric. The latter considers the residual physical resources such as processing power, memory and bandwidth. Finally, to perform the embedding of the global $\mathcal{VN}$ topology, Adaptive-VNE adopts a backtracking algorithm in order to minimize the virtual network mapping cost. To do so, the best virtual network mapping solution is built by selecting one (not necessary the best one) candidate for each $\mathcal{SC}$ while the mapping cost is minimized of the global $\mathcal{VN}$ topology.

Adaptive-VNE is validated by extensive simulations and with experimental testbed. In fact, we developed a $\mathcal{VN}$ embedding platform called ProvisionLab. It is based on Xen hypervisor and openVsiwtch. Based on the results obtained, Adaptive-VNE outperforms the related strategies in terms of i) reject rate of $\mathcal{VN}$ requests, ii) $\mathcal{CP}$ revenue, iii) congestion of virtual links and iv) satisfaction rate of congested virtual links.

The remainder of this paper is organized as follows. The next section will present the business model of Cloud computing by explaining the Cloud's actors and their relationships. Afterwards, in Section 3, we will detail the related work addressing the $\mathcal{VN}$ embedding problem. In Section 4, we will formulate both the network model and the $\mathcal{VN}$ optimization mapping problem. Then, we will describe our adaptive virtual network embedding algorithm, Adaptive-VNE in Section 5. Performance evaluation of both simulations and experimentations will be detailed in Section 6. Finally, Section 7 will conclude the paper.

2. Business model of Cloud computing

Cloud computing business model defines new actors. Indeed, the service providers are no more the owner of the physical infrastructure. The ***user actor*** is decoupled into two actors: ***Service providers*** ($\mathcal{SP}$) and ***Clients***. $\mathcal{SP}s$ create and run applications over the physical infrastructure offered by a ***Cloud provider***. In other words, $\mathcal{CP}s$ supply an Infrastructure as a Service (IaaS). As a consequence, the clients lease services from $\mathcal{SP}s$. The latters become customers of $\mathcal{CP}s$.

$\mathcal{CP}$'s own the physical infrastructure and manage the resources (e.g., bandwidth, CPU, memory, etc.). As described in [2], $\mathcal{CP}s$ can be classified into *Private, Community, Public* or *Hybrid* according to the type of hosting clients. Private $\mathcal{CP}$ offers services to one single organization (e.g., bank, company, etc.). Community $\mathcal{CP}$ serves a group of organizations having similar needs and interests (e.g. group of universities, etc.). Public $\mathcal{CP}$ provides services to the general public (i.e., open access). Finally, hybrid $\mathcal{CP}$ is a composition of the above types (i.e., private, community and public).

The physical infrastructure consists of a set of geographically distributed data centers interconnected through a Substrate backbone Network ($\mathcal{SN}$). It is formed by i) a set of geographically distributed routers (access and core) interconnected with wired connections such as fiber.

To illustrate the clients demands, hereafter we enumerate some scenarios:

- A commercial gaming service needs virtual nodes in several major cities with link constraints such as bandwidth size equal to 20 Mbps.

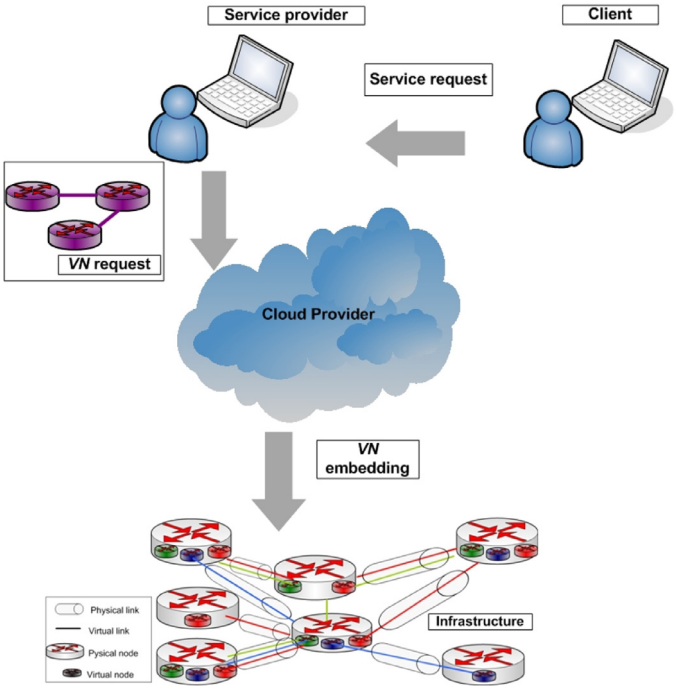


Fig. 1. NaaS business model.

- A researcher needs a specific network topology to evaluate the new proposed protocol.
- A company has one main office and many branches. It wants to set up a secure and confidential communication system between the main office and branches by deploying its private network and managing its network stack. It is worth noting that setting up Virtual Private Networks (VPN) is a complicated task and is subject to human error.

In this paper, we will investigate the Network as a Service (NaaS) within the $\mathcal{SN}$ of private $\mathcal{CP}$. The business model is the following. First, a client specifies the service requirements in terms of i) virtual network topology, ii) processing power and memory of virtual routers, iii) bandwidth of virtual links, iv) softwares, etc.. Then, the above requirements are communicated to a $\mathcal{SP}$. The latter is elected according to the choice of the client. Next, $\mathcal{SP}$ needs to deploy the softwares within the best data centers and to connect all the instances by the requested $\mathcal{VN}$ topology. In order to embed the $\mathcal{VN}$ and softwares, the $\mathcal{SP}$ delegates the instantiation process to a $\mathcal{CP}$ insofar as it owns the physical infrastructure. It is worth noting that in this paper we tackle only the embedding problem of $\mathcal{VN}$ within the $\mathcal{SN}$. Afterwards, based on the $\mathcal{VN}$ description, $\mathcal{CP}$ identifies the appropriate substrate resources and allocates them. $\mathcal{CP}$ also has the ability to migrate other $\mathcal{VN}s$ in order to free resources for new requests. Once the $\mathcal{VN}$ is instantiated, the client deploys the protocol stack and has all the privileges to administer the virtual topology. In fact, there is no limit or constraint on the choice of client's protocol stack. It is straightforward to see that a $\mathcal{CP}s$ must optimize the physical resource allocation in order to maximize the revenue while i) guaranteeing the required QoS and ii) ensuring low penalty in terms of congestion and rejection rate of new $\mathcal{VN}$ requests. Fig. 1 summarizes the NaaS business model.

3. Related work

Virtual network embedding is one of the most challenging issues of Network as a Service (network virtualization) in Cloud

computing. In fact, a fundamental challenge in resource instantiation is how to afford optimal allocation so that the $\mathcal{CP}$ fulfills the required SLA of $\mathcal{CP}$ while operational cost is minimized and the global revenue is maximized. To do so, $\mathcal{CP}$ maps online $\mathcal{VN}$s in the $\mathcal{SN}$ using the minimum volume of physical resources while still satisfying the required QoS in terms of bandwidth, processing power and memory. This in turn minimizes the reject rate of $\mathcal{VN}$ requests and maximizes returns for the $\mathcal{CP}$.

$\mathcal{VN}$ embedding problem under node and link constraints has been proved to be NP-hard [3–5] even in the offline case. It is worth noting that the offline $\mathcal{VN}$ embedding problem can be reduced to the NP-hard multiway separator problem [6]. Moreover, even if all the virtual nodes are mapped, virtual link embedding problem can be reduced to the unsplittable flow problem [4,5], which is still NP-hard [7].

$\mathcal{VN}$ resource provisioning strategies can be classified into two main groups: i) ***Static*** and ii) ***Dynamic*** approaches.

The first group, static approaches [6–13], consists of $\mathcal{VN}$ embedding strategies which full guarantee the required client SLA in terms of bandwidth, processing power and memory. The advantages of this approach are the simplicity of deployment and a full customer satisfaction guarantee. Unfortunately, the static approach does not optimize physical resource usage rate. Even though the allocated resources of any embedded $\mathcal{VN}$ are free, they cannot be exploited by the rest of $\mathcal{VN}$s. The second group, dynamic approaches, provides a dynamic resource provisioning of virtual resources. In doing so, physical resource usage rate in the $\mathcal{SN}$ is maximized. The second group encompasses two sub-groups: i) ***reconfiguration based strategies*** and ii) ***adaptive strategies***. The reconfiguration strategies aim to "tidy up" the embedded $\mathcal{VN}$ within the $\mathcal{SN}$ in order to balance the load of physical resources. Hence, the acceptance rate of $\mathcal{VN}$ requests will be increased. The adaptive strategies take advantage of the unused allocated resources by allowing more than one $\mathcal{VN}$ to exploit them. In fact, adaptive mapping strategies embed $\mathcal{VN}$s with respect to their i) current demand, ii) prediction of circulating traffic in $\mathcal{VN}$s and iii) required performance (i.e., rate of congested links, reject rate of $\mathcal{VN}$s, etc.). Hereafter, we will give an in-depth overview of adaptive $\mathcal{VN}$ resource provisioning strategies proposed in literature.

In [14], the authors propose a dynamic and flexible allocation scheme for $\mathcal{VN}$ mapping based on a non-cooperative game theory model [11]. The proposed algorithm allocates the bandwidth among multiple $\mathcal{VN}$s based on defined payoff function. The latter function is convex and encompasses three functions: i) utility, ii) price to pay and iii) congestion cost according to the assigned bandwidth. Each $\mathcal{VN}$ is modeled as a player and characterized by the playing strategies. The authors assume that the individual $\mathcal{VN}$ does not communicate with others to modify its own playing strategy. Hence, the infrastructure provider is responsible for enforcing the instantiated $\mathcal{VN}$s to i) modify their strategies, ii) ensure the fairness and iii) maximize their revenue. The authors prove the convergence of their scheme through simulations. We notice that the authors do not consider how to embed $\mathcal{VN}$s in the $\mathcal{SN}$. In fact, they assume the mapping strategy of $\mathcal{VN}$s is known. Moreover, the proposal is validated only with small-size benchmark and the scalability issue has not been studied.

In [15], an adaptive bandwidth allocation algorithm, denoted by `Davinci`, was proposed for $\mathcal{VN}$s. The main idea is to periodically reassign bandwidth shares between instantiated virtual links in order to maximize aggregate performance across multiple traffic classes. To do so, each $\mathcal{VN}$ is assumed to run customized packet-delivery protocols optimizing a performance objective. The latter is formulated as a convex function of network parameters. Then, a traffic management protocol is designed to control the amount of traffic through each substrate path including current congestion levels and the performance objectives of $\mathcal{VN}$s. The main drawback of this proposal is that each substrate link must be aware of all objective functions of $\mathcal{VN}$s transiting through it. Moreover, virtual router allocation problems have not yet been considered.

In [16], the flexible bandwidth $\mathcal{VN}$ embedding problem was tackled. The proposed algorithm uses multi-commodity flow solver to embed virtual links. In order to avoid producing bottleneck substrate links, the proposal calls periodically the traffic predictor module to adjust the links' bandwidth allocation with the largest occupation. The results obtained show that the multi-commodity flow solver incorporated with a traffic prediction module widely improves the usage rate of bandwidth and reduces the rate of packet loss compared with the variant without prediction.

In [17], the mechanisms for adaptive bandwidth allocation are investigated. The main goal is to distribute fairly the bandwidth among virtual links in order to avoid bottleneck substrate links. To do so, the authors define two types of connections for virtual links: i) restricted connections consuming more bandwidth than reserved and ii) unrestricted connections limited by the reserved bandwidth for the current link. First, the proposed algorithm allocates bandwidth to restricted connections based on their requirements. Then, the remaining bandwidth is distributed equally among unrestricted connections crossing a substrate link. Unfortunately, we notice that only bandwidth allocation is investigated and the mapping of virtual nodes is not tackled. Moreover, the authors assume that only one connection crosses a virtual link which needs to be extended for multiple connections.

The main contribution of this paper consists in proposing an adaptive embedding algorithm. `Adaptive-VNE` algorithm aims to find the virtual network mapping that maximizes the minimum residual bandwidth in the network and simultaneously minimizes the total usage of physical links. Indeed, the main idea is to balance the load of resources while mapping the virtual network in order i) to avoid causing physical resources bottlenecks and ii) to favor the utilization of least loaded resources. By doing so, incoming virtual networks are most likely to find available physical resources. To reach our objective, `Adaptive-VNE` iteratively constructs the mapping solution by selecting physical nodes that i) have available resources (CPU, memory) and ii) optimize the load balancing in physical links. It worth noting that links load balancing is achieved thanks to the *K*-supplier based approach which aims to select the physical nodes that offer paths with the highest residual bandwidth.

Unlike [6–8], our proposal will analyze the traffic circulating over the virtual links in order to optimize the bandwidth usage and reuse available bandwidth for other $\mathcal{VN}$ requests. Unlike [8,9], we will consider limited substrate resources. Indeed, only a finite number of $\mathcal{VN}$s can be hosted in the $\mathcal{SN}$. Hence, the acceptance rate of $\mathcal{VN}$ requests can be determined. Finally, unlike [14–17], the virtual router mapping stage will be taken into account during the embedding of $\mathcal{VN}$ request. In the aim to optimize the embedding virtual topology, our proposal will coordinate the virtual routers and links mapping stages and exploit the unused allocated bandwidth.

4. Formulation of the adaptive virtual network embedding problem

We model the $\mathcal{SN}$ as an undirected graph denoted by $\mathcal{G} = (V(\mathcal{G}), E(\mathcal{G}))$ where $V(\mathcal{G})$ and $E(\mathcal{G})$ are respectively the sets of physical routers and their connected links. Each physical router, $w \in V(\mathcal{G})$, is characterized by its i) residual processing power $B(w)$, ii) residual memory $M(w)$, iii) type: access or core $X(w)$ and iv) geographic location $L(w)$. Note that if $X(w) = 1$, then w is an access router. Otherwise, $X(w) = 0$. Likewise, each physical link, $e \in E(\mathcal{G})$, is typified by its i) capacity $C(e)$, ii) distribution of usage rate $\mathcal{U}(e)$ $(0 \leq \mathcal{U}(e) \leq 1)$ and iii) used bandwidth $y(e)$.

Similarly, a $\mathcal{VN}$ request is modeled as an undirected graph, denoted by $\mathcal{D} = (V(\mathcal{D}), E(\mathcal{D}))$ where $V(\mathcal{D})$ and $E(\mathcal{D})$ are respectively the sets of virtual routers and their virtual links. Each virtual router, $v \in V(\mathcal{D})$, is associated with the i) required processing power $B(v)$ and ii) memory $M(v)$, iii) its type $X(v)$ and iv) geographic location $L(v)$ if is an access router. Moreover, each virtual link $d \in E(\mathcal{D})$ is characterized by its i) bandwidth capacity $C(d)$ and ii) distribution of usage rate $\mathcal{U}(d)$ $(0 \leq \mathcal{U}(d) \leq 1)$.

The problem is to embed the virtual graph $\mathcal{D}$ in the substrate graph $\mathcal{G}$ considering the same $\mathcal{VN}$ mapping constraints as in our previous work [18], except the constraint concerning bandwidth allocation. Hereafter, we will define the following notations:

- $W(v) \subseteq V(\mathcal{G})$ denotes the set of potential substrate routers for hosting virtual node $v \in V(\mathcal{D})$ taking into account its requirements such as: processing power $B(v)$, memory $M(v)$, geographical location $\mathcal{L}(v)$, etc.
- $V(w) \subseteq V(\mathcal{D})$ denotes the set of virtual routers that can be embedded on physical router $w \in V(\mathcal{G})$. Note that $v \in V(w) \equiv w \in W(v)$.
- x_{vw} is a binary variable indicating whether virtual router $v \in V(\mathcal{D})$ is assigned to physical router $w \in V(\mathcal{G})$ or not.
- P_{st} denotes the set of admissible physical paths from physical routers s to t, $(s,t) \in V(\mathcal{G})^2$. Note that A^2 denotes the family of all ordered pairs of elements within set A (i.e., $A = \{(s,t) \in A^2 : s \neq t\}$).
- $\mathcal{P}$ denotes the set of all admissible paths. Formally, $\mathcal{P} = \bigcup_{\{s,t\} \in V(\mathcal{G})^2} P_{st}$.
- u_{dp} is a binary variable indicating whether virtual link $d \in E(\mathcal{D})$ is mapped upon the physical path $p \in \mathcal{P}$ or not.
- $(a(d), b(d)) \in V^2(\mathcal{D})$ denotes the starting and terminating virtual routers of virtual link $d \in E(\mathcal{D})$.
- δ_{ep} is a binary coefficient determining whether physical link $e \in E(\mathcal{G})$ belongs to path $p \in \mathcal{P}$.

The mapping of virtual routers is constrained so that for each $\mathcal{VN}$ request, $\mathcal{D}$, two virtual routers cannot be assigned to the same substrate router. Formally,

$$\sum_{v \in V(w)} x_{vw} \leq 1, \quad \forall w \in V(\mathcal{G}) \tag{4.1}$$

In addition, each virtual router must be assigned in only one physical router. Formally,

$$\sum_{w \in W(v)} x_{vw} = 1, \quad \forall v \in V(\mathcal{D}) \tag{4.2}$$

Virtual router, $v \in V(\mathcal{D})$, can be mapped in the substrate router, $w \in V(\mathcal{G})$, if i) the available residual resources (i.e., $B(w)$ and $M(w)$) are at least equal to those required (i.e., $B(v)$, $M(v)$) and if ii) v has the same type (i.e., access or core) as w. Formally,

$$\forall v \in V(w), \quad \begin{cases} (B(w) - B(v))x_{vw} \geq 0 \\ (M(w) - M(v))x_{vw} \geq 0 \\ (X(w) - X(v))x_{vw} = 0 \end{cases} \tag{4.3}$$

Each virtual link, $d \in E(\mathcal{D})$ between $a(d)$ and $b(d)$, is embedded in an **unsplittable** substrate **path** denoted by $p \in \mathcal{P}$ between s and t. Note that p is a set of substrate links that form a path. In fact, the current $\mathcal{SN}$s mainly make use of shortest-path-based routing protocols such as Open Shortest Path First (OSPF) and avoid the out-of order arrival problem of multi-path. Formally,

$$\sum_{p \in \mathcal{P}} u_{dp} = 1, \quad \forall d \in E(\mathcal{D}) \tag{4.4}$$

A virtual link $d \in E(\mathcal{D})$ must be embedded in only one path $p \in P_{st}$ (i.e., unsplittable) such as $a(d) \in V(\mathcal{D})$ is assigned to substrate router $s \in V(\mathcal{G})$ and $b(d) \in V(\mathcal{D})$ is assigned to substrate router $t \in V(\mathcal{G})$. Formally,

$$\forall d \in E(\mathcal{D}), \quad p \in P_{st} \begin{cases} u_{dp} \leq x_{a(d)s} \\ u_{dp} \leq x_{b(d)t} \end{cases} \tag{4.5}$$

On the other hand, the first objective consists in maximizing the acceptance rate of $\mathcal{VN}$ requests. Hence, the reject rate of requests is minimized and the provider's revenue is maximized. In this respect, our objective is to maximize the minimum residual bandwidth in the network (i.e., z) and then, **keeping the value of $z$at its maximum**, we minimize the total usage of physical links (i.e., maximize $-Z$). Note that z and Z are equal to:

$$z = \min_{e \in E(\mathcal{G})} (C(e) - y_e) \tag{4.6}$$

$$Z = \sum_{e \in E(\mathcal{G})} y_e \tag{4.7}$$

However, the flexible bandwidth $\mathcal{VN}$ mapping needs an additional objective concerning the usage rate of substrate links. Indeed, this approach allows a substrate link to host more than its capacity. In other words, the sum of all requested bandwidth transiting over the substrate link $e \in E(\mathcal{G})$ can be greater than $C(e)$. The new objective consists in minimizing the bottleneck rate in each substrate link e with respect to the traffic behavior transiting over it and with respect to all the virtual links. Formally,

$$\forall e \in E(\mathcal{G}), \quad \text{minimize}[P(\mathcal{U}(e) = 1\ )] \tag{4.8}$$

where $P(a = b)$ denotes the probability that a is equal to b. Note that $\mathcal{U}(e) = 1$ means that substrate link e is congested.

We will now outline our $\mathcal{VN}$ embedding optimization problem.

Problem 1 (Adaptive virtual network embedding problem)**.**

$$\begin{array}{ll} \texttt{lex max} & (z, -Z) \\ \texttt{minimize} & P(\mathcal{U}(e) = 1\ ), \quad \forall e \in E(\mathcal{G}) \\ \\ \texttt{subject to:} & \sum_{v \in V(w)} x_{vw} \leq 1, \quad \forall w \in V(\mathcal{G}) \\ & \sum_{w \in W(v)} x_{vw} = 1, \quad \forall v \in V(\mathcal{D}) \\ & (B(w) - B(v))x_{vw} \geq 0, \quad \forall v \in V(w) \\ & (M(w) - M(v))x_{vw} \geq 0, \quad \forall v \in V(w) \\ & (X(w) - X(v))x_{vw} = 0, \quad \forall v \in V(w) \\ & u_{dp} \leq x_{a(d)s}, \quad \forall d \in E(\mathcal{D}), \quad p \in P_{st} \\ & u_{dp} \leq x_{b(d)t}, \quad \forall d \in E(\mathcal{D}), \quad p \in P_{st} \\ & \sum_{p \in \mathcal{P}} u_{dp} = 1, \quad \forall d \in E(\mathcal{D}) \\ & y_e \leq C(e), \quad \forall e \in E(\mathcal{G}) \\ & x_{vw}, u_{dp} : \textit{binary} \\ & z, Z, y_e : \textit{continuous} \\ & \mathcal{U}(e) : \textit{distributionofusagerate} \end{array}$$

Note that lex max$(z, -Z)$ objective means lexicographical maximization. In other words, we first maximize z (i.e., minimum unused capacity on physical links) and then we maximize $-Z$ without deteriorating the maximization of z. The above problem is a multi-objective mixed-integer optimization problem. It has been proved to be NP-hard [3–5]. In the next section, we will propose a new scalable $\mathcal{VN}$ flexible embedding algorithm based on an approximation algorithm for bottleneck problems, denoted by `Adaptive-VNE`.

5. Proposal: `Adaptive-VNE` algorithm

`Adaptive-VNE` is a new adaptive $\mathcal{VN}$ embedding algorithm operating as follows. First, the $\mathcal{VN}$ topology (i.e., $\mathcal{D}$) is divided into a set of solution components, denoted by $\mathcal{SC}$, forming a star topology. Then, the generated solution components $\{\mathcal{SC}_i\}$ are sorted and

mapped sequentially. Note that the assignment of a $\mathcal{SC}$ is formulated as a K-supplier problem [19] and its resolution is equivalent to build a small part of the overall topology. Afterwards, based on an approximation algorithm for bottleneck problems [1], for each $\mathcal{SC}_i$ a set of candidate substrate routers that could host the $\mathcal{SC}_i$'s center virtual router are generated. The final assigned topology is constructed incrementally based on all the generated $\mathcal{SC}$s' candidates and using the backtracking strategy in order to minimize the overall mapping cost. Hereafter, we will detail the main `Adaptive-VNE` stages: i) *Formation of solution components*, ii) *Mapping of solution component*, iii) *Forecasting the usage rate of resources* and iv) *Selection of the global $\mathcal{VN}$ embedding topology*. The pseudo algorithm of `VNE-Adaptive` is illustrated in Algorithm 1.

Algorithm 1: Adaptive-VNE.

```
Input: 𝒟, 𝒢
Output: Mapping of 𝒟 in 𝒢
{𝒮𝒞_i} ← Generation-Solution-Component(𝒟)
for i = 1, i ≤ |{𝒮𝒞_i}|, i++ do
    W(v_i) ← Generation-Best-Candidates (𝒮𝒞_i, 𝒟, 𝒢 )
    /*Backtracking mechanism*/
    for j = 1, j ≤ i, j++ do
        Select best candidates w*_j ∈ W(v_j) such as the global
        mapping cost is minimized
Mapping-Edge-Routers-Links( 𝒟, 𝒢)
for j = 1, j ≤ |{𝒮𝒞_j}|, j++ do
    Mapping-Solution-Component (𝒮𝒞_j, w*_j, 𝒢)
```

5.1. Formation of solution components

In this stage, the $\mathcal{VN}$ request, $\mathcal{D}$, is split up into k sub-requests depending on $|V(\mathcal{D})|$ and $|E(\mathcal{D})|$, known as Solution Components, denoted by $\{\mathcal{SC}_i\}_{1\leq i\leq k}$. To do this, `Adaptive-VNE` first embeds all virtual access routers. Indeed, the virtual access router, v (i.e., $X(v)=1$), is mapped in the nearest substrate access router, w (i.e., $X(w)=1$), offering the most residual resources and giving priority to processing power. Remember that the available resources of substrate router w (i.e., $B(w)$ and $M(w)$) must be at least equal to those requested (i.e., $B(v)$ and $M(v)$). Then, virtual links between the virtual access routers will be assigned as explained in Section 5.2. The remaining virtual topology, lacking virtual access routers and links connecting them, is denoted by $\tilde{\mathcal{D}}$. Note that in $\tilde{\mathcal{D}}$, many virtual links missing one of their outermost virtual routers appear. In fact, the virtual routers had already been mapped. Insofar, we will refer to this type of links as a **"hanging links"**.

As depicted in Fig. 2, A $\mathcal{SC}_i$ is a star topology, composed of a center virtual node v_i and its direct connected links. $\{\mathcal{SC}_i\}_{1\leq i\leq k}$ are generated sequentially on the basis of the number of their hanging links. To do so, first the virtual router with the highest number of hanging links in $\tilde{\mathcal{D}}$ is selected. Then, $\mathcal{SC}_i$ is built by the selected virtual router and its whole attached virtual links without their outermost virtual routers. Next, $\mathcal{SC}_i$ is subtracted from $\tilde{\mathcal{D}}$ (i.e. $\tilde{\mathcal{D}} \leftarrow \tilde{\mathcal{D}} \backslash \mathcal{SC}_i$). The same process is repeated recursively to build the rest of the solution components until $\tilde{\mathcal{D}}$ becomes empty (i.e., $\tilde{\mathcal{D}}=\emptyset$). Note that the set $\{\mathcal{SC}_i\}_{1\leq i\leq k}$ obtained follows the same sequencing during the $\mathcal{SC}$ formation process.

5.2. Mapping of solution components

The potential substrate routers capable for hosting the $\mathcal{SC}_i$'s center virtual router v_i, denoted by $W(v_i)$, are those supplied with

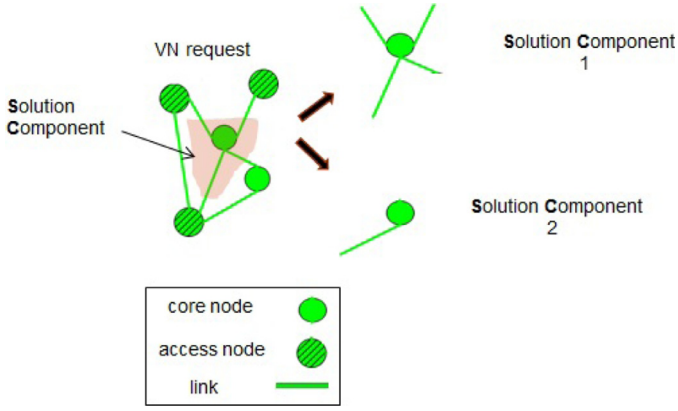


Fig. 2. Solution Component formation.

sufficient residual resources (i.e., processing power and memory) and guaranteeing the minimum mapping cost.

To find $W(v_i)$, the $\mathcal{SC}_i$'s mapping sub-problem is formulated as K-supplier problem [19]. Indeed, the K-supplier problem consists of n nodes that are partitioned into i) a set of suppliers and ii) a set of customers. The objective is to find the subset of suppliers that minimizes the cost of serving customers.

We define i) $\mathcal{N}_i$ as the set of substrate routers hosting the virtual routers of hanging links in $\mathcal{SC}_i$ and ii) $\mathcal{S}_i$ as the set of substrate routers not belonging to $\mathcal{N}_i$ and not hosting any virtual router of $\mathcal{VN}$ request. So, $\mathcal{N}_i$ models the set of customers and the objective is to find the best set of suppliers $W(v_i)$ within $\mathcal{S}_i$ which can host $\mathcal{SC}_i$'s center virtual router.

We remind that the main goal of `Adaptive-VNE` is to reduce the mapping cost of $\mathcal{VN}$ while maximizing residual $\mathcal{SN}$ resources. To do so, i) the allocated bandwidth in the $\mathcal{SN}$ must be minimized for each virtual link $d \in E(\mathcal{D})$ and ii) the selected substrate routers within the prospective candidates, $W(v_i)$ for any $v_i \in V(\mathcal{D})$, must maximize the residual resources and load balancing in terms of processing power, memory and bandwidth once the virtual links are mapped.

The prospective candidates within $\mathcal{S}_i$ forming $W(v_i)$ should be as close as possible to the substrate routers belonging to $\mathcal{N}_i$ and have available resources (at least those required by v_i) in terms of processing power and memory. Thus, we define the mapping cost of substrate router $w \in \mathcal{S}_i$ denoted by c_w as follows:

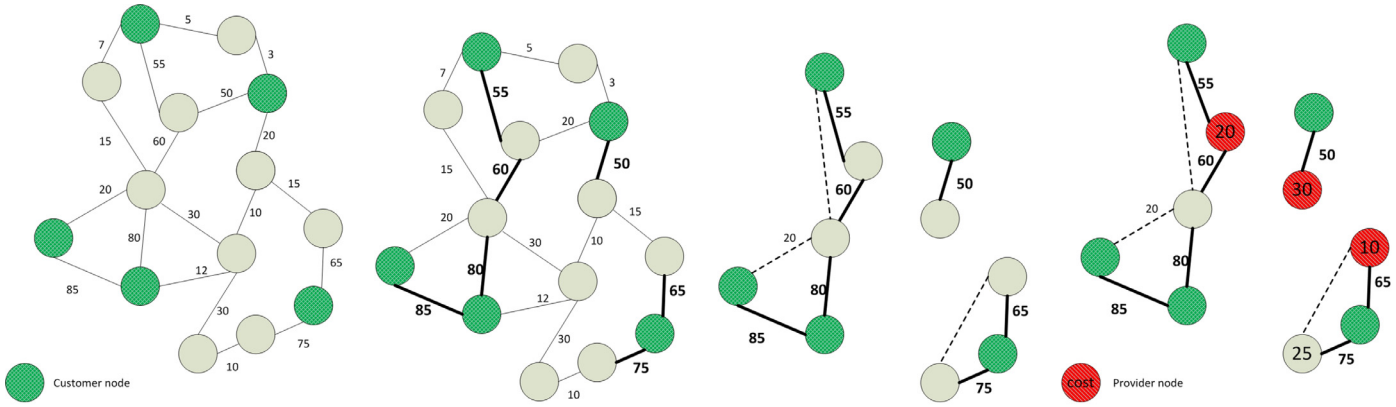
$$c_w = \frac{1}{B(w) + M(w) + \sum_{u\in\mathcal{N}_i} \mathcal{R}_{p_w^u} + \epsilon} \tag{5.9}$$

where p_w^u is the best path between substrate routers w and u in terms of the proposed metric defined in [18], $\mathcal{R}_{p_w^u}$ is the minimum residual bandwidth in the substrate path p_w^u and ϵ a small positive constant to avoid dividing by zero. We recall that $B(w)$ and $M(w)$ correspond to the residual processing power and the residual memory $M(w)$ respectively.

Since the sub-problem is formulated as K-supplier problem, the total mapping cost of the prospective candidates $W(v_i)$ must be at most equal to K. Formally,

$$c_{W(v_i)} = \sum_{w\in W(v_i)} c_w \leq K \tag{5.10}$$

To resolve the above K-supplier subproblem, we propound to adapt the proposed approximation algorithm in [1]. To do so, `Adaptive-VNE` constructs sequentially a set of subgraphs $\mathcal{G}_j$. For each $\mathcal{G}_j$, the algorithm tests whether there is a set $W(v_i)$ having a cost $c_{W(v_i)} \leq K$. If this latter does not exist, the new subgraph $\mathcal{G}_{j+1}$ is generated and j is incremented. The same process is repeated until $W(v_i)$ is found or $(j+1)$ is greater than the number of $\mathcal{SC}$'s



(a) Initial graph $\mathcal{G}$ (b) $\mathcal{G}_6 \subset \mathcal{G}$ (c) $\mathcal{G}_6^2$ (d) $\mathcal{G}_6^*$

Fig. 3. Generation of best candidates.

hanging links (i.e., number of customers). It is worth noting that the calibration of K in each $\mathcal{SC}$ is performed with respect to the mean value of its mapping cost.

The procedure that generates subgraph $\mathcal{G}_j$ proceeds as follows. First, the edge weighted complete graph based on shortest paths, denoted by $\mathcal{G}'$, is generated from the physical graph $\mathcal{G}$ (see Fig. 3(a)) . Then, the physical links $e \in E(\mathcal{G})$ are sorted according to their residual bandwidth.

Let $RB(e_i)$ be the residual bandwidth of the substrate link e_i. The new set substrate link is denoted by $E'(\mathcal{G})$ and is equal to:

$$E'(\mathcal{G}) = \{e_1, \ldots, e_m\} \setminus (RB(e_i + 1)) \geq (RB(e_i)) \tag{5.11}$$

where $m = |E(\mathcal{G})|$ and $0 \leq i \leq m-1$.

Afterwards, as shown in Fig. 3(b), the subgraph $\mathcal{G}_j(V(\mathcal{G}_j), E(\mathcal{G}_j))$ can be generated. In fact, $V(\mathcal{G}_j) = V(\mathcal{G})$ and $E(\mathcal{G}_j) = E_j = \{e \in E'(\mathcal{G}) : y(e) \leq y(e_j)\}$. It is worth noting that $\mathcal{G}_j$ represents the subgraph composed of the first jth best substrate links.

`Adaptive-VNE` tests whether $W(v_i)$ can be generated from $\mathcal{G}_j$ as following. First, $\mathcal{G}_j^2$ is generated. It consists in adding to $\mathcal{G}_j$ new links between nodes connected by 2-hop path. Then, graph $\mathcal{G}_j^2$ is reduced to $\mathcal{N}_i$ and denoted by $\tilde{\mathcal{G}}_j^2$. In other words, as depicted in Fig. 3(c), the graph $\mathcal{G}_j^2$ is reduced to the number of customers. Afterwards, the maximal independent set of $\tilde{\mathcal{G}}_j^2$ is generated, denoted by $\mathcal{IS}$. Indeed, $\mathcal{IS} \subset \mathcal{N}_i$ is a subset of customers. Note that an independent set of graph is a set of pairwise non-adjacent nodes. Besides, a maximal independent set is an independent set that is not properly contained in any independent set. It has been proved that the problem of maximal independent set can be solved in polynomial time [1]. Once $\mathcal{IS}$ is generated for each customer $u \in \mathcal{IS}$, its lowest 1-hop neighbor $w^* \in \mathcal{G}_j$ in terms of mapping cost is selected and added to $W(v_i)$. It is worth noting that $w*$ must belong to the eligible candidates belonging to $\mathcal{S}_i$. Finally, if $c_{W(v_i)} \leq K$ then the `Adaptive-VNE` converges. Otherwise, the next sub-graph $\mathcal{G}_{j+1}$ will be generated and the same process is repeated.

Let $\mathcal{G}_j^*$ be the subgraph on which $W(v_i)$ is found. $\mathcal{G}_j^*$ is called a bottleneck graph (see Fig. 3(d)). As detailed in [1], the above algorithm yields an approximate solution guaranteed to be within a constant factor **3 of the optimal solution**. The generation of the best candidates $W(v_i)$ that would hosts a $\mathcal{SC}_i$ is detailed in Algorithm 2 and illustrated in Fig. 3(b).

Algorithm 2: Generation-Best-Candidates.

```
Input: SC_i, N_i, S_i, D, G
Output: W(v_i) /*Set of best candidates for hosting v_i*/
W(v_i) ← ∅
v_i ← SC_i's center router
Generate E'(G)
j ← 1
repeat
    Generate graph G_j
    Generate graph G_j^2
    Generate graph G̃_j^2 /*G_j^2 is reduced to N_i*/
    Generate Independent Set IS of G̃_j^2
    foreach u ∈ IS do
        Select w* ∈ V(G_j) : (w* is one-hop neighbor of u) ∧
        (w* ∈ S_i) ∧ (c_w* is the lowest one)
        W(v_i) ← W(v_i) ∪ {w*}
    j ← j + 1
until [(c_W(v_i) ≤ K) ∧ (j > nbr hanging links of SC_i)];
```

5.3. Forecasting of the usage rate of resources

Contrarily to a static $\mathcal{VN}$ embedding strategy, a flexible approach aims to optimize the usage rate of resources by exploiting the unused allocated resources of already mapped $\mathcal{VN}$s. Hence, more $\mathcal{VN}$ requests can be accepted. However, the probability that a $\mathcal{VN}$ request cannot access to its allocated resources must be minimized. In this paper, we focus only on the flexibility of bandwidth allocation since the latter is a scarce resource. Virtual nodes' resources will be instantiated assuming a static approach.

The random distribution of usage rate $\mathcal{U}(d)$ for each virtual link $d \in E(\mathcal{D})$ is very hard to be typified and defined upstream of the mapping. Indeed, the distribution depends on many parameters such as: i) end-user traffic, ii) routing protocol, iii) failures, etc. To tackle the above problem, we propose to make use of a monitoring module in order to estimate an upper-bound of usage rate for each virtual link d mapped in the $\mathcal{SN}$.

Let $\tilde{\alpha}_{t_i}(d)$ be the average data traffic volume during the period $[t_i - T, t_i]$ where T is the size of the observing window. We define, so doing, the usage rate during the same period, denoted by $\mu_{t_i}(d)$, as:

$$\mu_{t_i}(d) = \frac{\tilde{\alpha}_{t_i}(d)}{C(d)} \tag{5.12}$$

we recall that $C(d)$ is the bandwidth capacity of virtual link d.

Consequently, the usage rate of the window $[t_i - T, t_i]$ of the substrate link $e \in E(\mathcal{G})$ is defined as follows:

$$\mu_{t_i}(e) = \frac{1}{|\mathcal{L}_e|} \sum_{d \in \mathcal{L}_e} \mu_{t_i}(d) \tag{5.13}$$

where $\mathcal{L}_e$ is the set of virtual links transiting over the substrate link e.

As a simple prediction module of usage resource rate, we propose setting the usage rate of substrate link e denoted $\mu^*_{t_i}(e)$ as:

$$\mu^*_{t_i}(e) = \bar{\mu}_{t_i}(e) + \delta \cdot \frac{\sigma}{\sqrt{n}} \tag{5.14}$$

where i) $\bar{\mu}_{t_i}(e)$ and σ respectively denote the mean and standard deviation of $\mu_{t_i}(e)$, ii) n is the set sample size of $\mu_{t_i}(e)$ and iii) δ is equal to normal inverse cumulative distribution in respect to the desired confidence level. So, the used bandwidth of the substrate link e can be estimated as:

$$y(e) = C(e) \cdot \mu^*_{t_i}(e) \tag{5.15}$$

The above module calculates the upper-bound of the usage rate according to the confidence level. However, this monitoring module could, in the future, be extended and deployed easily (i.e., without any modification of the proposal) with more sophisticated traffic prediction algorithms [20,21].

5.4. Selection of the global $\mathcal{VN}$ embedding topology

Once all the solution components are generated $\{\mathcal{SC}_i\}$, `Adaptive-VNE` starts the mapping of solution components as described in Section 5.2. To do so, backtracking mechanism is adopted. Indeed, at each iteration, the global mapping cost is evaluated and the previous mapping of solution components can be updated by choosing another candidate while the global mapping cost is reduced.

It is worth remarking that the global mapping topology is generated incrementally. In each step, only one $\mathcal{SC}$ is mapped but the final decision will be made at the end. So, `Adaptive-VNE` avoids being blocked in local optima by using a backtracking approach.

6. Performance evaluation

In this section, we will study the performance of our proposed adaptive $\mathcal{VN}$ embedding strategy `Adaptive-VNE`. To achieve this, first we will define the performance metrics to evaluate it. Then, we will detail the simulation environment and the generated results compared with prominent related strategies found in literature. Finally, we will describe our experimental platform ProvisionLab and we will present the performance obtained.

6.1. Performance metrics

To gauge the performance of our proposal, hereafter we define the following performance metrics:

- $\mathbb{Q}$ is the rejection rate of $\mathcal{VN}$ requests.
- $\mathbb{R}(\mathcal{D})$ is the revenue produced by a $\mathcal{VN}$ request $\mathcal{D}$.
- $\mathbb{B}$ is the bottleneck rate of embedded virtual links in the $\mathcal{SN}$.
- $\mathbb{D}$ is the average bottlenecking duration of embedded virtual links in the $\mathcal{SN}$.
- $\mathbb{S}$ is the satisfaction rate of mapped $\mathcal{VN}$s. It quantifies the percentage of allocated bandwidth resources with respect to the requested volume.
- **Residual** memory of substrate routers.
- **Allocated** processor power of substrate router.

6.2. Simulation performance

6.2.1. Simulation environment

We designed and implemented a discrete event $\mathcal{VN}$ embedding simulator. To generate $\mathcal{SN}$ and $\mathcal{VN}$ topologies, `GT-ITM` tool[4] is used. As in [7,22], (i) the arrival of $\mathcal{VN}$ requests is modeled by a Poisson Process with rate λ_A and (ii) $\mathcal{VN}$ lifetime is modeled by exponential distribution with mean μ_L.

We integrated in the simulator our proposal `Adaptive-VNE` and the related strategies: (i) `VNE-Greedy` [7], (ii) `VNE-AC` [23], (iii) `Static-VNE` which is the static variant of `Adaptive-VNE` that assigns the virtual links according to the peak demand and (vi) `GomoryHu-VNE` [24] which converges to the optimal solution while considering the peak demand. We compared the above embedding strategies with respect to the defined performance metrics. Moreover, we evaluated the robustness of `Adaptive-VNE`. Note that our proposal cannot be compared with related **adaptive** $\mathcal{VN}$ embedding strategies since the virtual router mapping stage is not considered. Only virtual link assignment is addressed.

As stated in [7,23,25], we make use of the following benchmark scenario. The $\mathcal{SN}$ size is set to 100 and, in this case, the ratio of access and core routers are respectively fixed at 20% and 80%. Furthermore, the $\mathcal{VN}$ size is set according to a discrete uniform distribution, using the values given in [2,10]. Since virtual access router are defined by customers, we can assume that each virtual router 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 routers 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 respectively fixed to 4 requests per 100 time units and 1000 time units. We calibrate the capacity of substrate routers and links (i.e., $B(w)$, $M(w)$, and $C(e)$) according to a continuous uniform distribution, taking the values in [50, 100]. Similarly, the required virtual resources (i.e., $B(v)$, $M(v)$, and $C(d)$) 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. Finally, the size of observing window $T = 10$ time units.

We assume exponential ON-OFF traffic sources as in [26,27]. The average ON time is set to 100 units and the OFF period varies within the range of [5, 50] units to simulate different traffic load conditions. This means that each virtual link can at most be off during half of the duration of the ON period. We assume that each virtual link d is typified by its usage rate distribution $\mathcal{U}(d)$ following a discrete uniform distribution using values [50%, 100%]. During each ON traffic period, the current usage rate is sampled according to its distribution. Besides, we assume that the extremity nodes of virtual link d transmit with a fixed rate leading to a fixed usage rate as generated by $\mathcal{U}(d)$. Finally, we consider that all virtual links are full-duplex.

Note that a realistic characterization of network traffic is outside the scope of this paper. The main goal is to evaluate the robustness and the effectiveness of the proposed algorithm in different network conditions. Moreover, all simulation results are calculated with confidence intervals corresponding to a confidence level of 99.70%. Tiny confidence intervals are not shown in the following figures.

Table 1
Performance of $\mathcal{VN}$ Embedding strategies.

Strategy	Reject rate (%)	Revenue ($\times 10^4$)
`VNE-AC`	5.1 ± 0.40	47.42 ± 0.28
`VNE-Greedy`	12.95	42.51
`Static-VNE`	4.8	47.78
`Adaptive-VNE`	0.25 ± 0.05	50.82
`GomoryHu-VNE`	0.55	50

6.2.2. Simulation results

Table 1 compares the reject rate $\mathbb{Q}$ and total revenue of the whole (i.e., 2000) $\mathcal{VN}$ requests. It shows that our proposal, `Adaptive-VNE`, significantly reduces the reject rate compared to other strategies and thus, improves the $\mathcal{CP}$'s revenue. As

[4] http://www.cc.gatech.edu/projects/gtitm/.

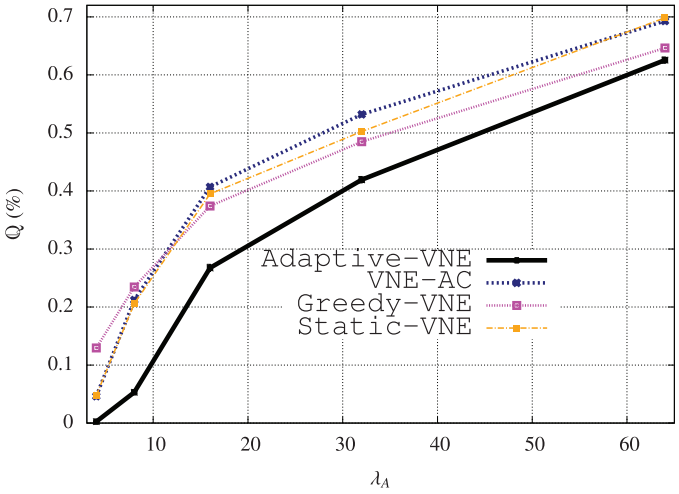
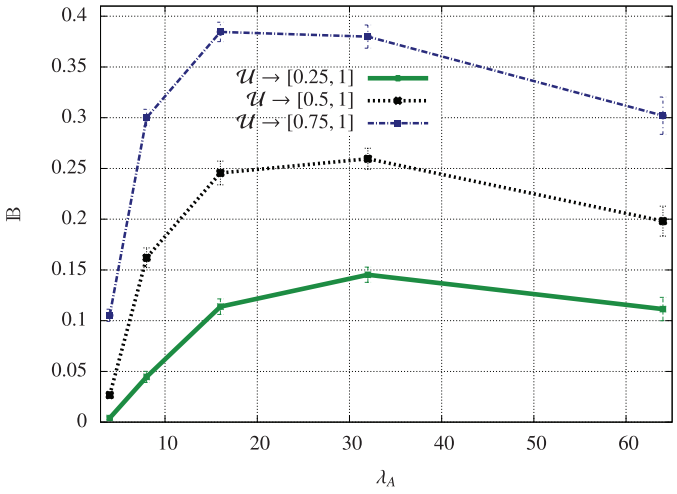
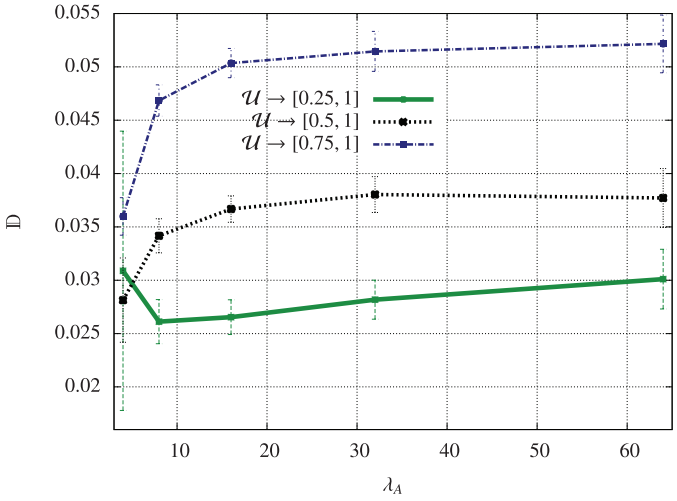


Fig. 4. Adaptive-VNE — Comparison of $\mathcal{VN}$ rejection rate $\mathbb{Q}$.



(a) Rate of bottlenecked virtual links ($\mathbb{B}$)



(b) Average duration of bottlenecked virtual links ($\mathbb{D}$)

Fig. 5. Adaptive-VNE — Congestion of virtual links.

illustrated in Table 1, `Adaptive-VNE` rejects only 3 ± 1 requests among 2000, which is the equivalent of 0.25 ± 0.05% of all of the $\mathcal{VN}$ requests. We notice that 5.1 ± 0.40%, 12.95%, 4.8% and 0.55% of requests are respectively rejected by static strategies `VNE-AC`, `VNE-Greedy`, `Static-VNE` and `GomoryHu-VNE`. Indeed, the static embedding strategies decline at least two times more of $\mathcal{VN}$ requests than the adaptive `Adaptive-VNE`. This is mainly due to the fact that the static mapping strategies do not reuse allocated bandwidth even though they are not exploited. We recall that this is also the case of `GomoryHu-VNE` which converges to the optimal mapping based on the peak demand. Consequently, this leads to large resource wastage and more rejection of $\mathcal{VN}$ requests.

It is worth noting that the **static variant of our proposal** (i.e., `Static-VNE`) **outperforms** our previous strategy `VNE-AC` and `VNE-Greedy`.

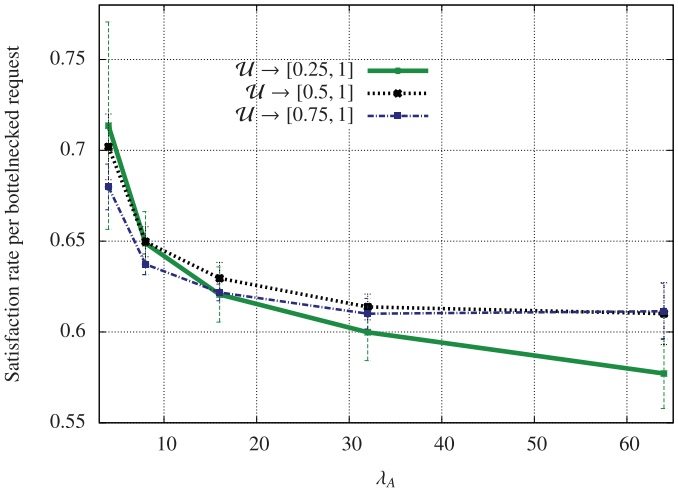
On the other hand, we can see, in Table 1, that `Adaptive-VNE` maximizes the $\mathcal{CP}$'s revenue. In fact, as defined in [7,8,22], the revenue of a $\mathcal{VN}$ request, $\mathbb{R}(\mathcal{D})$, is proportional to the amount of its allocated resources. This can be explained due to the fact that a larger number of accepted requests leads to the increase in allocated resources. $\mathbb{R}(\mathcal{D})$ can be expressed as:

$$\mathbb{R}(\mathcal{D}) = \sum_{v \in V(\mathcal{D})} B(v) + \sum_{v \in V(\mathcal{D})} M(v) + \sum_{d \in E(\mathcal{D})} C(d) \tag{6.16}$$

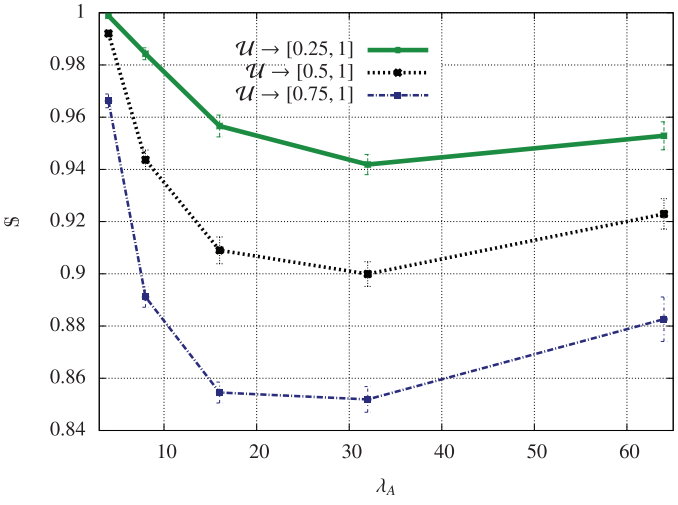
In Fig. 4, we evaluate the robustness of `Adaptive-VNE` compared with static strategies `VNE-AC`, `VNE-Greedy` and `Static-VNE` with respect to the arrival rate of $\mathcal{VN}$ requests, λ_A. Note that the average lifetime of $\mathcal{VN}$ is fixed. As shown in the figure, our proposal `Adaptive-VNE` keeps a lower rejection rate while the arrival rate of $\mathcal{VN}$s grows exponentially in the range [4, 64]. Recall that $\lambda_A = n$ means that n $\mathcal{VN}$ requests arrive per 100 units. It is clear to see that for low λ_A values (i.e., $4 \leq \lambda_A \leq 32$), our proposal significantly reduces the rejection rate. For example, at $\lambda_A = 8$ the rejection rate is equal to 5.66 ± 0.43%. Hence, `Adaptive-VNE` reduces the rejection rate by at least 4 times compared with the best related `Greedy-VNE` embedding strategy (reject rate equal to 23.45%). Note that the rejection rate improvement stays high while the arrival rate grows exponentially. Moreover, with high values of λ_A, the $\mathcal{SN}$ becomes saturated and no matter the chosen embedding strategy no more requests can be accepted. Even though substrate resources are congested, the improvement remains noticeable even with $\lambda_A = 64$, `Adaptive-VNE` rejects almost 63.05 ± 1.30% of $\mathcal{VN}$ requests when `VNE-AC`, `Greedy-VNE` and `Static-VNE` respectively decline 69.3% ± 0.7, 65% and 69.85%.

To investigate the impact of $\mathcal{VN}$'s traffic load on `Adaptive-VNE`'s performance, we simulate within the $\mathcal{VN}$ three scenarios according to the discrete uniform usage rate distribution $\mathcal{U}(d)$: i) [25%, 100%], ii) [50%, 100%] and iii) [75%, 100%]. In Fig. 5(a), we illustrate the rate of bottlenecked virtual links with respect to λ_A and $\mathcal{U}(d)$.

It is clear that whatever the range of $\mathcal{U}(d)$, the rate of congested virtual links is low and less than 50%. In fact, for low occupation rates (i.e., $\mathcal{U}(d) \rightarrow [25\%, 100\%]$), at most only 15% of mapped virtual links are congested (i.e., bottlenecked traffic) when $\lambda_A = 16$. Moreover by increasing the usage rate, (i.e., $\mathcal{U}(d) \rightarrow [50\%, 100\%]$ and $\mathcal{U}(d) \rightarrow [75\%, 100\%]$), the bottleneck rate of virtual links increases and this is a consequence of the high level of traffic circulation. Whereas this increase stays limited and, at most, reaches 42%. It is worth noting that this bottlenecking is considerably short-lived. In fact, as displayed in Fig. 5(b), the duration of congestion is infinitely short since it does not exceed 5.5% of the lifetime of $\mathcal{VN}$s whatever the arrival rate and the load rate. For example, with a congestion duration around to 2% of $\mathcal{VN}$ lifetime and $\mathcal{U}(d) \rightarrow [50\%, 100\%]$, the end-user may notice a slight perturbation of the traffic, which can sometimes even be transparent in



(a) Satisfaction level of bottlenecked virtual links



(b) Satisfaction level of all mapped virtual links ($\mathbb{S}$)

Fig. 6. Adaptive-VNE — Satisfaction level of mapped virtual links.

the case of elastic traffic such as TCP-based applications (e.g., HTTP, STMP, FTP, etc.). We have noticed that for high values of λ_A (i.e., $\lambda_A \geq 32$), the rate of bottlenecked virtual links decreases. This is expected since the acceptance rate declines with high $\mathcal{VN}$ arrival rates.

As crucial and important performance metric, to evaluate the effectiveness of our proposal, is the satisfaction level of congested mapped $\mathcal{VN}$s. Indeed, even though the bottleneck duration is tiny, we evaluate the percentage of reserved bandwidth volume compared to the required bandwidth during the congestion period.

Fig. 6 (a) shows the satisfaction level of bottlenecked $\mathcal{VN}$s. One can clearly see that the satisfaction level is high since it varies between 60% and 80%. This signifies that a virtual link, congested during its lifetime, will see at least 60% of its bandwidth requirements satisfied whatever the occupation rate distribution $\mathcal{U}(d)$ and the arrival $\mathcal{VN}$ requests rate λ_A.

Moreover, Fig. 6(b) reports the global satisfaction of all mapped $\mathcal{VN}$s. Note that this performance metric $\mathbb{S}$ is considerably satisfying. Indeed, $84\% \leq \mathbb{S} \leq 100\%$.

6.3. Experimental performance

We evaluate the performance of our proposal `Adaptive-VNE` with a testbed. To accomplish our goal, we designed and implemented an experimental virtualization platform, called ProvisionLab, in order gauge the efficiency of `Adaptive-VNE`. Hereafter, we will describe the technical details of the $\mathcal{VN}$ embedding experimental platform. Then, we will illustrate the ProvisionLab's results.

6.3.1. Experimental environment

ProvisionLab routers are running with "ARCH Linux" operating system characterized by "Linux kernel" release 3.2. Upon the above operating system, we install the hypervisor "Xen" environment release 4.1.2 characterized by the virtualization library "Libvirt"[5]. Virtual routers are created using Xen and Library "Libvirt". The operating system of virtual routers is "ARCH Linux" characterized by "Linux kernel" release 3.4.6. The substrate network topology is illustrated in Fig. 7. The platform is composed of 7 network devices: 2 core routers and 5 access routers. The access routers denoted by "PhyN0", "PhyN1", "PhyN2", "PhyN3" and "PhyN4" are characterized by i) processor power (CPU): 1.5 GHz and ii) Random Access Memory (RAM): 150 Mo. Core routers denoted by "PhyN5" and "PhyN6" are characterized by i) CPU: 2.4 GHz and ii) RAM: 500 Mo.

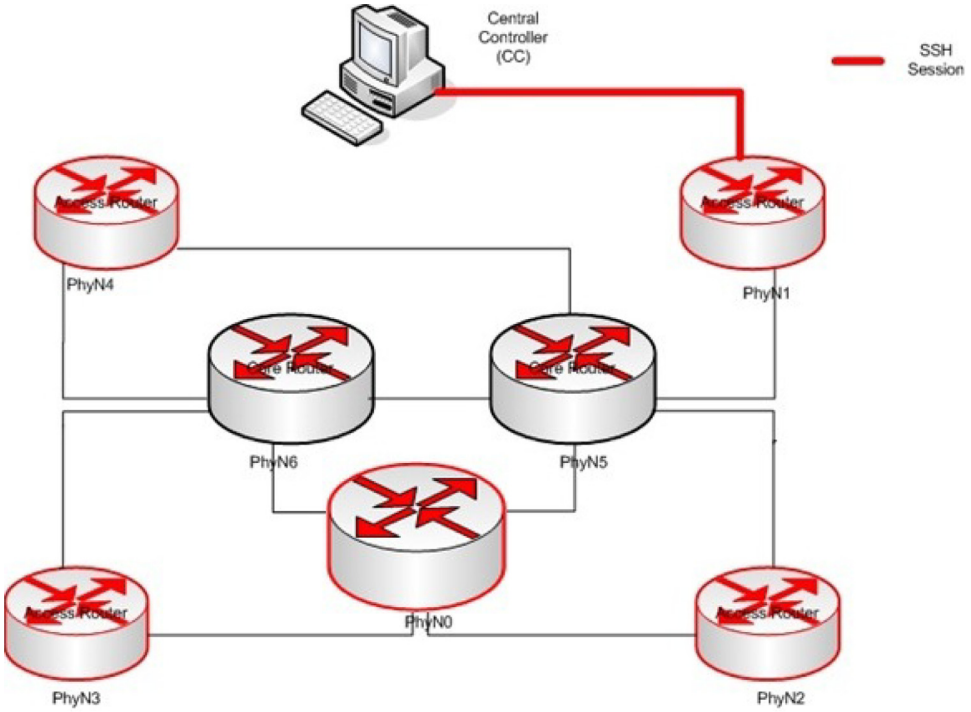
Core routers are characterized by 6 Gigabit Ethernet ports, while edge routers are characterized by 4 Gigabit ports. The capacity of physical cables interconnecting physical routers is 100 Mbps. The Ethernet ports of different routers are connected using i) cables with capacity of 100 Mbps and ii) bridge network mode. We recall that a network bridge is a 2-layer device used to create a connection between two distant network devices. It forwards data between the network's routers based on the MAC address of both sender and receiver. We configured Xen in the bridge mode in order to monitor the transmission between the routers.

ProvisionLab is managed by a Central Controller (CC) which is responsible for i) virtual and substrate network devices, ii) admission of new $\mathcal{VN}$s and iii) the $\mathcal{VN}$s mapping process. The latter strongly depends on state (i.e., residual resources) of the $\mathcal{SN}$ which is continuously monitored. Instantiated $\mathcal{VN}$s are monitored by Xen hypervisor. Note that hosting and hosted systems are classified by domains. Domain zero, abbreviated by "Dom0", is the initial domain launched by the hypervisor on the boot. "Dom0" plays the role of **host** operating system. The **hosted** virtual machines represent the unprivileged domains abbreviated by "domU".

Each $\mathcal{VN}$ is able to allocate one or more virtual machines (i.e., domU). Dom0 is responsible for the communication monitoring between different domUs. Dom0 has a direct access to the hardware. Moreover, Dom0 is the starter of the new domains. Hosted domUs are created and/or killed using the library Libvirt. Each $\mathcal{VN}$ is typified by its own i) IPv4 address and ii) identifier. It is worth noting that each virtual machine is qualified by a unique identifier (uuid). The latter is randomly generated and permits to distinguish a virtual machine from any others within a $\mathcal{VN}$.

In order to deploy `Adaptive-VNE` in the ProvisionLab, we designed and implemented several processes in the CC proceeding as follows. First, the CC receives the new $\mathcal{VN}$ request which needs to be instantiated in the $\mathcal{SN}$. Then, `Adaptive-VNE` is launched to embed the $\mathcal{VN}$ request based on the already collected information from the $\mathcal{SN}$. Afterwards, if `Adaptive-VNE` succeeds to map the $\mathcal{VN}$ request then notification messages are sent to the hosting substrate routers in order to execute the calculated mapping solution. In fact, the allocation process will be triggered within the concerned substrate routers. Finally, killing virtual machines process will be executed as soon as a $\mathcal{VN}$'s lifetime expires in order to unfreeze the allocated substrate resources. Hereafter, we will enumerate the main processes and will describe their tasks.

[5] http://libvirt.org/.



(a) Substrate network topology

(b) Picture of ProvisionLab

Fig. 7. Network devices of ProvisionLab.

- *Resource Discovery Process*: is responsible for monitoring $\mathcal{SN}$ resources (i.e., collecting information) in order to determine the availability of substrate resources and their suitability to host virtual resources. This information is periodically exchanged between the substrate routers and the CC. Resource Discovery Process is composed of four functional processes:
  1. *Topology Update* (TU): It is executed in the substrate routers. This process collects network information from its neighbors. This information describes the state of $\mathcal{SN}$ equipments such as: i) IP address, ii) residual resources (i.e., CPU, memory and bandwidth), iii) packet error rate, iv) byte error rate. TU process is responsible for periodically update of residual resources and takes into account the current $\mathcal{SN}$ state. Collected information is structured based on **Resource Description Framework** ("rdf") format. So additional monitored data can be easily added.
  2. *Topology Distribution* (TD): Thanks to this process, updated information is sent to the CC agent. To do so, "rdf" files generated by TU process in each substrate router are transmitted using **HTTP** protocol.
  3. *Topology Request* (TR): To collect topology files "rdf" from substrate routers, the CC agent periodically broadcasts "GET HTTP request". Upon receiving this request, each TD process is triggered and the topology file is transmitted.
- *Resources Allocation*: Once the mapping decision is made by `Adaptive-VNE`, the CC broadcasts resource reservation messages to the assigned substrate routers. These messages indicate for each virtual router in the $\mathcal{VN}$ request its: i) hosting substrate router in the $\mathcal{SN}$, ii) type (i.e., edge or core) and iiii) required resources (i.e., CPU and memory). Moreover, these messages contain for each virtual link the substrate path supporting it and its allocated bandwidth.

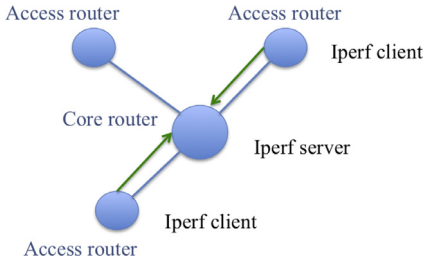


Fig. 8. Generation of UDP virtual traffic.

The $\mathcal{VN}$ request instantiation process is executed in two sequential stages.

1. *Virtual Router Creation* (VRC): Once the mapping decision is received by all the routers in the $\mathcal{SN}$, each substrate router checks whether it is concerned by the instantiation process or not. If it is the case, then VRC process is trigged to create the virtual router according the information received in "rdf" file. The latter contains the following information: i) $\mathcal{VN}$ identifier, ii) uuid (i.e., unique node identifier) identifying the virtual router belonging to the $\mathcal{VN}$ request, iii) amount of required CPU capacity and iv) amount of required memory. The router's agent constructs the **XML** description file by adding information concerning i) bridge id, ii) MAC address, iii) OS kernel image, etc. The XML file is exploited by "**Libvirt**" library in order to instantiate the virtual router.
2. *Virtual Link Creation* (VLC): In order to allocate bandwidth resources along the substrate paths supporting the virtual links, **Spanning Tree Protocol** (STP) is launched. It is worth noting that STP is included by default in **openVsiwtch** tool[6] which is deployed in our platform. We recall that, STP selects the shortest path while preventing bridge loops.

In order to evaluate the performance of `Adaptive-VNE`, we test the following scenario. The arrival rate of $\mathcal{VN}$s λ_A is fixed to 4 requests per 100 time units (seconds). The average lifetime μ_L is fixed to 1000 time units. The capacity of substrate routers and links are calibrated depending on the capacity of physical network devices previously outlined. The required virtual resources of routers (i.e., $B(v)$ and $M(v)$) are set according to a continuous uniform distribution. $B(v)$ and $M(v)$ are respectively sampled between [10, 17] KHz and [45, 45.1] KBytes. Similarly, we set required virtual bandwidth (i.e., $C(d)$) according to a continuous uniform distribution, using values in [40, 50] Mbps. Due to the hardware resource limitation, the number of $\mathcal{VN}$ requests in the current scenario is set to 50.

To investigate the impact of $\mathcal{VN}$'s traffic load on `Adaptive-VNE`'s performance, we generate a Constant Bit Rate (**CBR**) traffic circulating through each $\mathcal{VN}$. The usage rate of these flows are set according to a discrete uniform distribution taking values in [0.25, 1]. That mean the volume of traffic circulating over a virtual link d follows a discrete uniform distribution taking values in $[\frac{\mathcal{U}(d)}{4}, \mathcal{U}(d)]$. The traffic in each virtual link starts to circulate at a moment generated according to a discrete uniform distribution taking values in $[\frac{\mathcal{L}}{2}, \mathcal{L}]$. Note that $\mathcal{L}$ is the lifetime of the $\mathcal{VN}$ generated according to the exponential distribution with average μ_L. The **UDP** traffic is generated with "**iPerf**" tool. Note that traffic should be generated by end-users who are directly connected to access routers. As a consequence, the traffic must necessarily i) transit through core routers and ii) converge to an access router which transmits the traffic to the destination. For further details, Fig. 8 illustrates an example of UDP traffic circulating through a $\mathcal{VN}$ topology.

[6] http://openvswitch.org/.

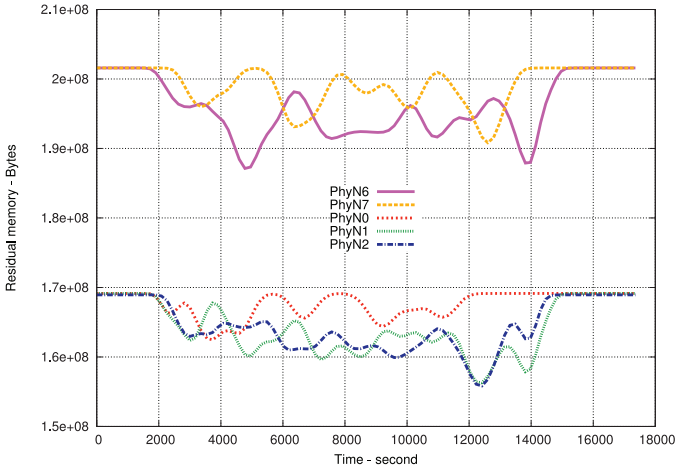


Fig. 9. ProvisionLab — Memory usage.

6.3.2. Experimental results

Based on the above scenario, `Adaptive-VNE` rejects only one $\mathcal{VN}$ request. In other words, the reject rate $\mathbb{Q}$ is equal to 2%. This obtained result shows the good performances of `Adaptive-VNE` in real conditions.

Fig. 9 evaluates the residual memory of all the substrate routers during the experimentation. We can easily differentiate three main stages.

The first stage expresses the volume of residual memory before any usage of substrate routers. It corresponds to the maximal capacity where no $\mathcal{VN}$ is embedded in the $\mathcal{SN}$. Indeed, the first $\mathcal{VN}$ request arrives at $t = 2000$ s. Hence, during the first period [0, 2000] s, the residual memory amount remains stationary at each substrate router. It is worth noting that, as previously explained, substrate core routers "PhyN5" and "PhyN6" are characterized by higher capacity of memory compared with the access routers.

During the second stage, $\mathcal{VN}$ requests pseudo-randomly (i.e., Poisson process) arrive in the $\mathcal{SN}$. The last $\mathcal{VN}$ request leaves the $\mathcal{SN}$ at $t = 15{,}000$ s. As we can see in Fig. 9, during the period [2000, 15,000] s, virtual resources are equally dispatched among the substrate routers. In fact, we remark that all substrate routers are elected to host virtual routers and the memory load is balanced among the substrate hosts. Consequently, we conclude that `Adaptive-VNE` maximizes the load balancing during the embedding decision process.

The third stage starts at $t = 15{,}000$ s. The volume of residual memory in the $\mathcal{SN}$ is evaluated after the departure of all $\mathcal{VN}$. It is clear to see that all memory resources are restored as in the first stage and the $\mathcal{SN}$ is free once again.

Fig. 10 illustrates the variation of used CPU during the embedding of $\mathcal{VN}$ requests. Similarly to the residual memory performance, it is clear to distinguish three stages: before, during and after the mapping of $\mathcal{VN}$s.

Note that the amount of CPU exploited does not only depend on allocated virtual resources. In fact, daemons and processes, described in Section 6.3.1, strongly impact the consumption of processing power. For this reason, CPU consumption is high during the first stage. Indeed, some substrate routers run multiple "iPerf" processes. We recall that "iPerf" tool is used to measure circulating throughput and residual bandwidth of physical links.

During the second stage (i.e., $\mathcal{VN}$ embedding phase), all substrate routers consume CPU due to i) the instantiation of virtual nodes and ii) the generation of traffic within $\mathcal{VN}$. In Fig. 10, we notice that the highest CPU use is measured in substrate router "PhyN1". In fact, the latter plays the role of Central Controller and runs continually "Resource Discover" process.

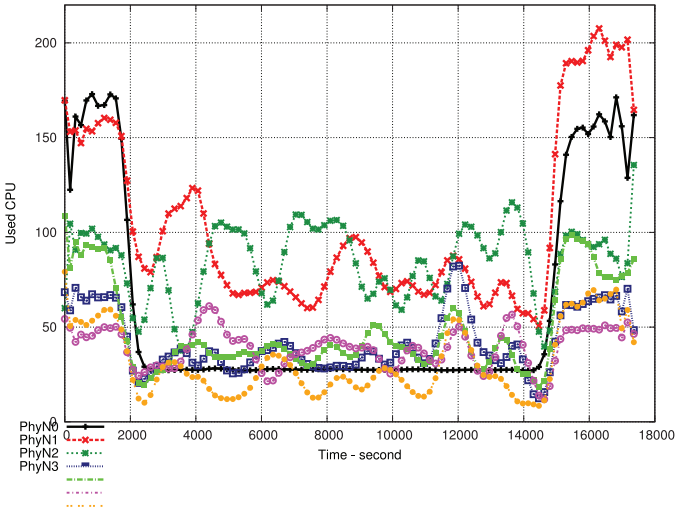


Fig. 10. ProvisionLab — CPU usage.

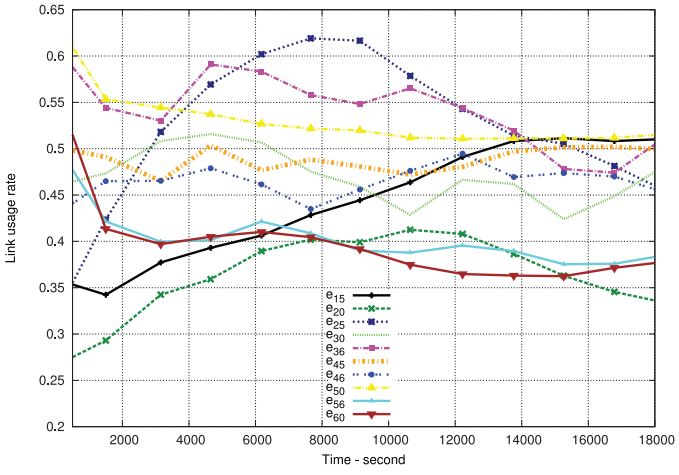


Fig. 11. ProvisionLab — Bandwidth usage.

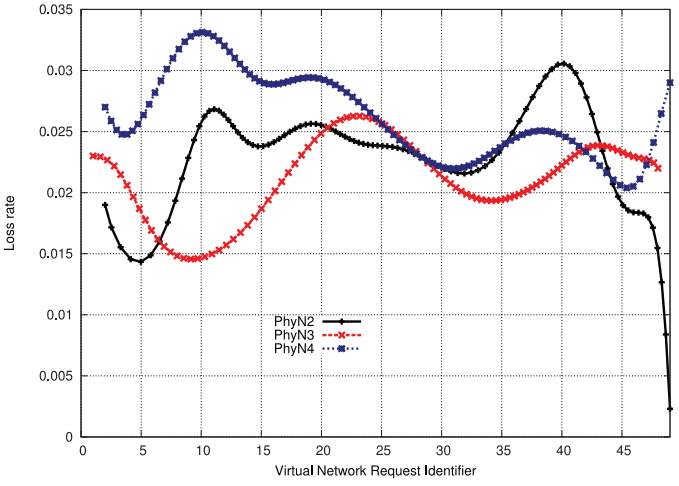
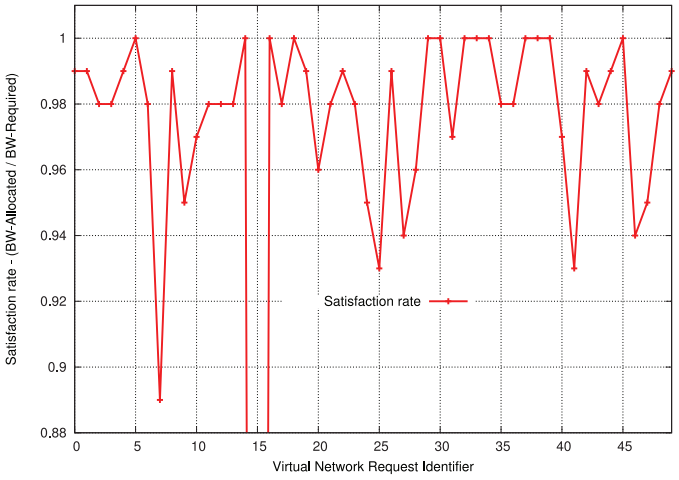
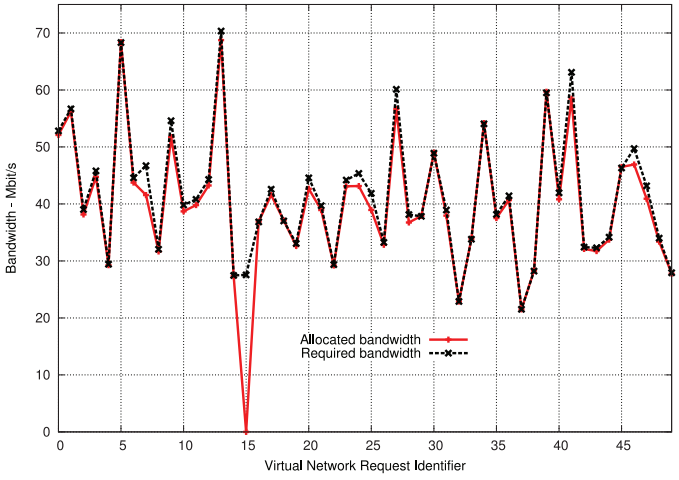


Fig. 12. ProvisionLab — Loss rate.

Fig. 11 depicts the link usage rate within the substrate network. It is straightforward to see that the traffic load is balanced and fairly shared between physical links. This can be explained by the fact that the distance path metric aims to make a trade-off between minimizing the length of substrate paths and maximizing residual bandwidth in the substrate network.



(a) Satisfaction Bandwidth rate per $\mathcal{VN}$



(b) Volume of requested and allocated bandwidth

Fig. 13. ProvisionLab — Bandwidth performance.

Fig. 12 illustrates the loss rate occurred during the lifetime of each instantiated $\mathcal{VN}$. Node that this network-relative metric is measured inside the different access nodes by "iPerf". It is worth noting that the loss rate of instantiated $\mathcal{VN}$ is low since it takes values in [0, 3.5]%. Besides, we notice that the loss rate is stationary. Indeed, it is straightforward to see that the latter is independent of the number of mapped $\mathcal{VN}$s which enhances the QoS of end-users. This obtained result confirms the good performances of `Adaptive-VNE` in real conditions.

Fig. 13 (a) illustrates the bandwidth satisfaction level (i.e., percentage of reserved bandwidth volume compared to the requested one) of $\mathcal{VN}$s. It is worth noting that, aside $\mathcal{VN}$ request 15 which is rejected, all virtual links belonging to other $\mathcal{VN}$s are satisfied at least 95%. Indeed, our proposal `Adaptive-VNE` aims to select always the shortest substrate paths having the most available bandwidth in order to maximize the satisfaction of incoming $\mathcal{VN}$ requests. Fig. 13(b) illustrates the volume of required and allocated bandwidth during the experimentation.

7. Conclusion

In this paper, we addressed the adaptive virtual network embedding problem. The main idea is to take advantage of the unused reserved bandwidth within the embedded virtual networks. In doing so, the rejection rate of new virtual network requests

will be minimized. To this end, we proposed a new adaptive virtual network embedding algorithm denoted by `Adaptive-VNE`. The problem is formulated as *K*-supplier problem. `Adaptive-VNE` makes use of i) the approximation-algorithm for bottleneck problems and ii) backtracking strategy to resolve the problem. Based on extensive simulations, `Adaptive-VNE` outperforms the most prominent strategies. Based on experimental testbed, the result obtained show that `Adaptive-VNE` optimizes the usage of physical resources and maximizes the satisfaction rate of end-users.

References

[1] D.S. Hochbaum, D.B. Shmoys, A unified approach to approximation algorithms for bottleneck problems, J. ACM 33 (3) (1986) 533–550.
[2] NIST, Definition of cloud computing v15, csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc
[3] D.G. Andersen, Theoretical approaches to node assignment, 2002. http://www.cs.cmu.edu/?dga/papers/andersen-assign.
[4] J. Kleinberg, in: Approximation algorithms for disjoint paths problems, 1996.
[5] S.G. Kolliopoulos, C. Stein, in: Improved approximation algorithms for unsplittable flow problems, 1997.
[6] M. Chowdhury, M. Rahman, R. Boutaba, Vineyard: virtual network embedding algorithms with coordinated node and link mapping, Netw. IEEE/ACM Trans. 20 (1) (2012) 206–219.
[7] M. Yu, Y. Yi, J. Rexford, M. Chiang, Rethinking virtual network embedding: substrate support for path splitting and migration, SIGCOMM Comput. Commun. Rev. 38 (2008) 17–29.
[8] Y. Zhu, M. Ammar, Algorithms for assigning substrate network resources to virtual network components, IEEE INFOCOM, 2006, pp. 1–12.
[9] J. Lu, J. Turner, Efficient Mapping of Virtual Networks onto a Shared Substrate, Technical Report, Washington University in St. Louis, 2006.
[10] C. Guo, G. Lu, H.J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, Y. Zhang, Secondnet: a data center network virtualization architecture with bandwidth guarantees, in: Proceedings of the 6th International Conference, in: Co-NEXT '10, ACM, New York, NY, USA, 2010, pp. 15:1–15:12.
[11] J. Nash, Non-cooperative games, in: Second Series, 54, Annals of Mathematics, 1951, pp. pp.286–295.
[12] J. Lischka, H. Karl, A virtual network mapping algorithm based on subgraph isomorphism detection, SIGCOMM VISA Workshop, 2009, pp. 81–88.
[13] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Neural Networks, 1995. Proceedings., IEEE International Conference on, 4, 1995.1942–1948 vol.4
[14] Y. Zhou, Y. Li, G. Sun, D. Jin, L. Su, L. Zeng, Game theory based bandwidth allocation scheme for network virtualization, IEEE Globecom, 2010, pp. 1–5.
[15] J. He, R. Zhang-shen, Y. Li, C. yen Lee, J. Rexford, M. Chiang, DAVINCI: dynamically adaptive virtual networks for a customized internet, in: Proceedings of CoNEXT, 2008.
[16] Y. Wei, J. Wang, C. Wang, X. Hu, Bandwidth allocation in virtual network based on traffic prediction, in: Wireless Communications Networking and Mobile Computing (WiCOM), 2010 6th International Conference on, 2010, pp. 1–4.
[17] J. Botero, X. Hesselbach, The bottlenecked virtual network problem in bandwidth allocation for network virtualization, in: Communications, 2009. LATINCOM '09. IEEE Latin-American Conference on, 2009, pp. 1–5.
[18] I. Fajjari, N. Aitsaadi, M. Piro, G. Pujolle, A new virtual network static embedding strategy within the clouds private backbone network, Comput. Netw. 62 (0) (2014) 69–88.
[19] Q. Wang, K. Cheng, A heuristic algorithm for the k-center problem with cost and usage weights, Technical report (University of Houston. Dept. of Computer Science).
[20] N. Sapankevych, R. Sankar, Time series prediction using support vector machines: a survey, Comput. Intell. Mag. IEEE 4 (2) (2009) 24–38.
[21] V. Alarcon-Aquino, J. Barria, Multiresolution fir neural-network-based learning algorithm applied to network traffic prediction, Syst. Man Cybern. Part C: Appl. Rev. IEEE Trans. 36 (2) (2006) 208–220.
[22] N. Chowdhury, M. Rahman, R. Boutaba, Virtual network embedding with coordinated node and link mapping, IEEE INFOCOM, 2009, pp. 783–791.
[23] I. Fajjari, N. Aitsaadi, G. Pujolle, H. Zimmermann, in: VNE-AC: Virtual Network Embedding Algorithm based on Ant Colony Metaheuristic, 2011.
[24] O. Soualah, I. Fajjari, M. Hadji, N. Aitsaadi, D. Zeghlache, in: A novel virtual network embedding scheme based on gomory-hu tree within cloud's backbone, 2016.
[25] N.F. Butt, M. Chowdhury, R. Boutaba, Topology-awareness and reoptimization mechanism for virtual network embedding, IFIP NETWORKING Conference, 2010, pp. 27–39.
[26] J. Elias, F. Martignon, A. Capone, G. Pujolle, A new approach to dynamic bandwidth allocation in quality of service networks: performance and bounds, Comput. Netw. 51 (10) (2007) 2833–2853.
[27] R.R.-F. Liao, A.T. Campbell, Dynamic core provisioning for quantitative differentiated services, IEEE/ACM Trans. Netw. 12 (3) (2004) 429–442.