DYVINE: Fitness-Based Dynamic Virtual Network Embedding in Cloud Computing

Chinmaya Kumar Dehury and Prasan Kumar Sahoo, *Senior Member, IEEE*

***Abstract*— Virtual network embedding (VNE) is the process of embedding the set of interconnected virtual machines onto the set of interconnected physical servers (PSs) in the cloud computing environment. The level of complexity of VNE problem increases when a large number of virtual machines with a set of resource demand need to be embedded onto a network of thousands of PSs. The key challenge of VNE is the efficient mapping of virtual networks (VNs), which may have dynamic resource demands. Existing solutions mainly emphasize on the embedding of static VN resulting in poor resource utilization and very low acceptance rate. To tackle such level of complexity in VNE, a fitness-based dynamic virtual network embedding (DYVINE) algorithm is proposed with the goal to maximize the resource utilization by maximizing the acceptance rate. Local and global fitness values of the virtual machines and VN, respectively, are used to utilize the maximum amount of physical resources. The proposed VNE algorithm allows the VN to be dynamic, which indicates that the structure and resource demand can be changed during its execution time. Furthermore, in order to reduce the embedding time in each time slot, a set of PSs is selected to host the VN instead of considering thousands of PSs, which may significantly increase the embedding time. The proposed embedding mechanism is evaluated through extensive simulation and is compared with similar existing embedding algorithms, which outperforms over others.**

***Index Terms*— Cloud computing, virtual network embedding (VNE), dynamic VNE, virtual resource allocation.**

I. Introduction

Cloud platform is the most preferable computing paradigm among the business entities, research organization and academics. The computing resources are provided to the remote users on the rented basis as utility computing. Normally, the cloud resources are referred to as CPU, memory, storage, and network bandwidth. Multiple Virtual Machines (VMs) are created with specific resource configuration according to the users resource requirement. Virtualization is the backbone technology to create, delete, and manipulate a VM [1]. This allows the Cloud Service Provider (CSP) to assign single physical server to multiple users by creating dedicated VMs for each user without compromising the data confidentiality.

Infrastructure as a Service (IaaS) users send the request to the CSP in terms of number of VMs and the communication topology with bandwidth requirement among the VMs [2], [3]. For example, as given in Figure 1(a), a cloud user may send the request that consists of three VMs $a, b,$ and c configured with 20 units, 10 units, and 10 units of computing resource requirement, respectively. The request also consists of specific network topology among the VMs and network resource demand such as the resource demand between VMs a and b is 2 units.

The users request consisting of multiple interconnected VMs with specific network topology refers to as Virtual Network (VN). CSP is equipped with thousands of high-end physical servers interconnected through most popular and efficient Fat-tree [4] network topology. The network of such huge number of physical servers is referred to as physical network. The process of embedding the users' VN atop the CSP's physical network is known as the Virtual Network Embedding (VNE) [5], [6]. As shown in Figure 1(b), two VNs are embedded onto the physical network.

VNE refers to as the embedding of a set of VMs onto a set of physical servers and the assignment of virtual links to the physical paths. Each physical path may consist of multiple physical links and connecting devices. The resource configuration of VMs comprises of CPU, memory, and storage resource demands. Similarly, the resource configuration of virtual links comprises of bandwidth requirement and maximum propagation delay. VN embedding could be in sequential or parallel order. In sequential order, all VMs are embedded followed by the virtual links or vice-versa. In parallel order, either each VM is embedded followed by embedding the attached virtual links or each virtual link is embedded followed by embedding the VMs at both ends. Different approaches such as game theory [7], heuristic approach [8], [9] are followed to solve this NP-hard VNE problem.

VNE can be classified into two types [10], [11], such as static and dynamic VNE. In static VNE, users are not allowed to change the resource configuration of each VM and the network topology of the VN after embedding onto the physical network. For instance, huge amount of batch and streaming data are generated from the smart city environment, which need to be handled by the CSP. CSP needs to allow the cloud

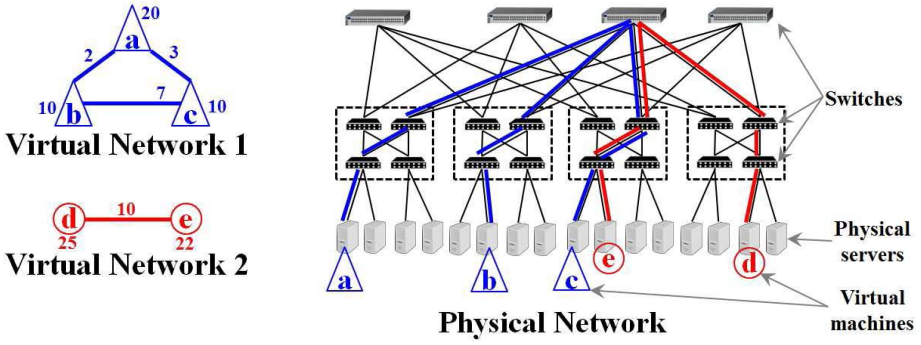
Manuscript received May 5, 2018; revised January 22, 2019 and March 8, 2019; accepted March 14, 2019. Date of publication March 21, 2019; date of current version April 16, 2019. This work was supported in part by the Ministry of Science and Technology (MOST), Taiwan, under Grant 107-2221-E-182-073 and in part by Chang Gung Medical Foundation, Taiwan under Grant CMRPD 2H0291. *(Corresponding author: Prasan Kumar Sahoo.)*

C. K. Dehury is with the Department of Computer Science and Information Engineering, Chang Gung University, Taoyuan City 333, Taiwan (e-mail: d0321009@stmail.cgu.edu.tw).

P. K. Sahoo is with the Department of Computer Science and Information Engineering, Chang Gung University, Taoyuan City 333, Taiwan, and also with the Division of Colon and Rectal Surgery, Chang Gung Memorial Hospital, Taoyuan City 33305, Taiwan (e-mail: pksahoo@mail.cgu.edu.tw).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSAC.2019.2906744



(a) Virtual networks (b) Physical network

Fig. 1. An example of virtual network embedding. (a) Virtual networks (b) Physical network.

users to change the resource requirement and the structure of the VN in order to handle such growth of data. It is possible, if the VNE is designed to handle the change in the structure of the VN, which is known as dynamic VNE. Change in the structure of VN includes addition and deletion of VMs and virtual links. This also includes the change in resource requirement of VMs and virtual links. While designing the dynamic VNE algorithm, the CSP must also consider other parameters of dynamic VNE such as frequency at which the VN can be changed by the users or the time interval between any two adjacent changes.

A. Motivation

One of the major advantages of the cloud computing is its ability to scale-up and scale-down its resources. Because of this feature, exact amount of resources can be assigned to the user's request to provide the better quality of service without engaging more number of physical resources. Normally, the user's requests arrive in the form of VNs, which is a set of virtual nodes and virtual links. The user's request can be classified into two categories such as static and dynamic. Static requests or VNs are the requests, where the resource requirement is fixed throughout the duration of execution. On the other hand, the resource requirement may change over time in case of Dynamic request or VN.

Based on our recent literature survey, it is observed that most of the research articles introduce different VNE mechanisms considering the static VN. The static VNE mechanism produces very efficient results due to fixed resource requirement for entire execution time. However, it is a cumbersome task for the cloud service provider to handle the VN, whose resource requirements changes with time. Let us consider an example of providing online game as a service, which allows multiple players to join the game at any time within one session and one VM can be assigned to only one player. This indicates that the number of VMs required for the one session of the game is decided by the number of players those who are playing currently. In this scenario, the underline resource requirement in the form of number of VMs will change at any time instance, since the number of players may change with time. This infers that the resource requirement is not static so as the requested VNs.

Similar situation can also be observed in a smart city scenario, which generates huge volume of data. To process the smart city big data, CSP installs a set of interconnected virtual nodes onto its available physical resources. Though the embedded VN may able to process the smart city data for few days or months, the increasing number of IoT devices and growth in volume of generated data force the CSP to reconfigure the VN and assign more resources to it. This indicates that the resource requirement of the VN is dynamic.

Embedding such kind of VNs with help of the static VNE algorithm is not an efficient approach. Though dynamic VN can be embedded onto the physical network using static VNE mechanism, the dynamic VN needs to be embedded repeatedly due to the changes in the resource requirement, which is a time-consuming process. This will also decrease the acceptance rate, if the resource requirement cannot be fulfilled by the CSP at any time instance during the execution time. Decrease in the acceptance rate has also direct impact onto the revenue of the CSP. The aforementioned research issue motivates us to revisit the design and embedding procedure of existing VNE mechanism. As a result, a novel DYnamic VIrtual Network Embedding (DYVINE) algorithm is proposed considering the dynamic resource requirement of the VNs.

The main contributions of our work are summarized as follows,

- A novel dynamic VNE algorithm is proposed based on the fitness values, which are calculated considering the resource demand of the VN and the resource availability of the Physical Servers (PSs) at different time instances. By doing so, the users are allowed to change the structure and resource configuration of the entire VN.
- The propagation delay of each physical path is considered during the embedding process of the virtual links along with the resource availability and demands at any instant of time.
- Two sub-algorithms of DYVINE are proposed with the goal to maximize the resource utilization. By doing so, the rate of acceptance of the VNs can be maximized, which may play a significant role in maximizing the revenue of the CSP.
- In order to reduce the total required embedding time of the VNs, a small set of most suitable physical servers are

TABLE I
COMPARISON OF RELATED WORKS

Algorithm	Request types	Resource reconfiguration	Resource types	Other major parameters	Embedding time	Findings	Limitations
LVRM [6]	Static request	No VN resource reconfiguration allowed	CPU, memory, and bandwidth	Round trip time, energy consumption, workload	Not optimize	Minimize the required network resource	VNs are not dynamic
VNE-NTANRC [9]	Static request	Not allowed	Bandwidth and PS capacity (no specific resource type)	Link propagation delay, Node location, and lifetime of VN	Not minimized	Minimize embedding monetary cost and maximize acceptance ratio	VNs are not dynamic
EAD-VNE and EAD-VNE-G [10]	Dynamic request	Yes	CPU, memory and network	Energy consumption of physical servers and switches	Not minimized	Minimize energy consumption	Do not minimize required amount of network resource
DVNMA [11]	Dynamic request	Yes	CPU, memory and network	Latency of physical and virtual links	Not minimized	Minimize the reconfiguration cost	No minimization of network resource requirement
PAME [16]	Static request	User can not modify the VN	Computing and network resource	Arrival and lifetime of the VN	Minimize through parallel embedding	Maximize revenue, acceptance ratio and revenue-to-cost ratio	Static VN request and dynamic resource configuration are not supported
SS [17]	Static request	Users not allowed	CPU and network bandwidth	Power consumption of physical servers	Not minimized	Maximize profit by considering load and energy management	Users are not allowed to modify the resource configuration of VNs
ALG-PERIOD and ALG-NO-PERIOD [18]	Static (known & unknown periodic resource demand)	CSP predict resource demand of VNs	Computing and network bandwidth	Lifetime and cost of VN	Not minimized	Maximize revenue of CSP	User cannot modify the resource demand of VN
SVC [19]	Static (Probabilistic demand prediction)	Uncertain bandwidth demand	Network bandwidth	No other parameter	Not minimized	Provide probabilistic bandwidth guarantee to user	VM's computing resource demand is not considered
Proposed DYVINE algorithm	Dynamic request	Resources are allowed to reconfigure	CPU, memory, and bandwidth	Link propagation delay, required time slots	Minimized	Maximize the resource utilization and acceptance rate	Implementation on real test-bed is required to verify the performance.

selected instead of considering all PSs that are available with the CSP.

- Extensive simulations are performed and compared with existing VNE schemes considering the resource utilization, acceptance rate, embedding time and other related parameters as performance metrics.

The rest of the paper is organized as follows. Section II presents the brief summary of the related works. The physical and virtual networks are modeled in Section III followed by the formulation of dynamic VNE problem and objective functions. The proposed DYnamic VIrtual Network Embedding (DYVINE) algorithm is presented with theoretical analysis in Section IV. Performance evaluation of the proposed algorithm is done in Section V followed by the concluding remarks in Section VI.

II. RELATED WORKS

Generally, the VNE problem is decomposed into two subproblems as virtual node mapping and virtual link mapping. Mostly, virtual node and link mappings are done separately and mapping of virtual nodes is followed by the mapping of virtual links. A number of VNE approaches have been proposed considering the dynamic resource requirement, topology of the data center, energy consumption, and revenue maximization [12]–[15]. An extensive comparison of the related works is presented in the Table I.

Considering the dynamic nature of the incoming VN, Zhang *et al.* [10] propose an energy-aware VNE (EAD-VNE) algorithm with the objective to minimize the energy consumption, which leads to the maximization of revenue. The basic and workload dependent power consumption by both physical server and networking devices are taken into account to achieve the objective. However, it is essential to consider the resource utilization in order to minimize the energy consumption. Besides, the proposed algorithm does not exploit the opportunity to map multiple similar inter-connected virtual machines onto a single physical server in order to reduce the energy consumption of the networking switches, which further can minimize the total energy consumption. Similarly, a cost-efficient VNE (DVNMA) algorithm is proposed by formulating the problem as a mixed integer linear programming problem in [11]. Authors address the problem of frequent changes in the structure and resource demand of the virtual network and the objective of the proposed

algorithm is to minimize the reconfiguration cost. Though various parameters are taken into account in the proposed heuristic method, assigning the VMs in one-to-one manner may not give efficient result in terms of reconfiguration cost. Further, the proposed algorithm does not consider the fact that by allowing multiple virtual machines to be embedded onto single physical server also leads to saving the cost of mapping or allocating the resources to the corresponding virtual links, resulting in further minimization of total reconfiguration cost.

Yan *et al.* [12] propose a novel VNE mechanism considering the congestion in the link mapping phase. Dividing the capacity of each server into slots with same capacity results in poor resource allocation, as the VMs in a VN are heterogeneous in nature, which is the major disadvantages of the proposed approach. A greedy VNE algorithm is proposed in [8], where the Euclidean distance shortest path approach is followed in the link mapping. Mapping all virtual nodes followed by all virtual links are some of the major disadvantages of the proposed VNE approach. This may also produce inefficient result in terms of embedding time when the reallocation is required for the efficient embedding solution. A novel approach is followed in embedding the VN in [20], where one virtual node is split into multiple virtual nodes for efficient resource utilization and acceptance rate maximization. However, the proposed VNE cannot be used in case of compute-intensive application. Further, CSP must have detail knowledge of the application. By splitting a virtual node into multiple virtual nodes will directly affect the network resource utilization.

With the goal to minimize the embedding cost and maximize the revenue of the infrastructure provider, Wang and Hamdi [14], Haeri and Trajkovic [21], and Hesselbach *et al.* [22] formulate the VNE problem as multi-objective linear programming problem. Embedding cost refers to as allocating exactly equal number of physical links. However, this objective would not be achieved if the virtual links would have mapped after mapping the virtual nodes. Hesselbach *et al.* [22] propose VNE mechanism with the objective to maximize the revenue, where the virtual nodes and virtual links are mapped simultaneously instead of sequentially. Considering the federated cloud environment, Aral and Ovatman [23] propose embedding algorithm for mapping multiple VM clusters. Though, the authors nearly achieve the objective of maximizing the acceptance ratio, they ignore the fact that data transfer incurs extra cost in terms of time in the geographically distributed cloud environment.

A cloud resource allocation and provisioning algorithm is proposed in [24], which works in a multi-service cloud environment considering the service cost, SLA, resource availability and users' demand. Though the proposed resource allocation algorithm performs well in the mentioned scenario, however, ignoring the network resource in resource allocation process may not be able give efficient result in real-life scenario. The importance of network resource is presented and the inefficient network resource reservation problem is addressed in [19] with the goal to maximize the revenue generated by the CSP. The above-mentioned VNE schemes share a common disadvantage, i.e., inability to reconfigure the VN during its execution. This infers that the users need to send the maximum resource requirement of the VMs and virtual links, which may lead to the resource over-subscription problem and cost ineffectiveness.

Reinforcement learning in machine learning approach is followed to embed the VN in [25]. Maximizing the revenue of the CSP by maximizing the acceptance ratio by using the historical usage data is the sole goal of the proposed VNE scheme. However, the proposed embedding algorithm does not support the dynamic VN and therefore may not effectively maximize the acceptance ratio. The membrane computing is exploited in order to embed the incoming VNs as proposed in [16]. Though the proposed approach achieves the mentioned goal of maximizing the revenue and acceptance ratio, the VNE algorithm does not support the dynamic nature of the incoming VNs and each VM is embedded onto separate physical servers, which may not maximize the acceptance ratio effectively. Pyoung and Baek [17] propose a VNE scheme taking the trade-off between the energy consumption and load balancing into account. The energy consumption can also be minimized by utilizing less network resource. However, in the proposed VNE scheme each VM is mapped onto single physical server, which indicates that the amount of network resource allocated is not less than its demand and hence the proposed embedding algorithm may not give efficient result in terms of power consumption.

The VNE problem can be formulated as a graph bisection problem as proposed in [26]. The proposed embedding algorithm allows the users to provide the location preference, which can be used for integrating the VMs and virtual link mapping. This also prohibits the physical servers to host multiple VMs from one VN. As a result, the network resource utilization is not optimized. Zhang *et al.* [27] propose the VNE algorithm considering the structure of the VN. The structure of the VN includes the clustering and degree information of the VMs. The objective of the proposed algorithm is to minimize the network utilization and maximize the acceptance ratio. This may give inefficient result as the dependencies among the VMs is not considered along with the dynamic nature of the VNs. Taking the network resource into consideration, the VNE algorithm in [28] uses the temporal and spatial network topology information to minimize the link interference. All the three above-mentioned algorithms suffer a common research issue, i.e., they do not support the dynamic nature of the incoming VNs. As a result, the acceptance ratio and resource utilization is not optimized.

III. Problem Formulation

In this section, the network model of physical servers and user's requests are presented. Afterward, the fitness matrix and the fitness value are formulated followed by the formulation of the objective function.

A. Physical Network

In the data center, a large number of physical servers are connected using specific network topology. The physical network consists of a set of interconnected physical machines

or physical servers and a set of connecting devices. The communication topology can be of two types such as switch-centric and server-centric. In this paper, we are considering switch-centric network topology, where all the physical servers are connected through the network devices such as switches. It is to be noted that fat-tree, VL2 and Jellyfish are some of the popular switch-centric network topologies [12]. The physical network is represented as an undirected weighted graph, $G^p(N^p, E^p)$, where $N^p = \{P_1, P_2, P_3, ..., P_m\}$ is the set of m number of physical servers and E^p is the set of physical paths. Each physical server $P_i \in N^p$ is associated with mainly two parameters, which are available computing resource (i.e. CPU and memory), and available network resource (i.e. network bandwidth). We use $A_i^x(t)$ to denote the available computing resource at time t of type $x \in \{memory, CPU\}$ in the physical server $P_i \in N^p$. Each physical path, denoted as $L_{ij}^p \in E^p$ between the physical servers P_i and P_j is associated with the bandwidth availability and propagation delay. The notation $N_{ij}^a(t)$ is used to denote the available network bandwidth at time t between the physical servers P_i and P_j. Using the value of $N_{ij}^a(t)$, the network bandwidth $N_i^a(t)$ available at the physical server P_i at time t can be calculated as,

$$N_i^a(t) = \max_{\forall P_j \in N^p} N_{ij}^a(t), \quad i \neq j \tag{1}$$

B. Virtual Network

Users send their resource requirement in the form of Virtual Network (VN). Each VN consists of a set of virtual nodes and a set of virtual links, which defines the communication among the virtual nodes. Here, each virtual node represents one VM. Since, in this paper, we are dealing with the dynamic nature of the VNs, the graph of VN is designed as a function of time. As a result, the undirected weighted graph of the VN is denoted by the notation $G_t^v(N_t^v, E_t^v)$, where $N_t^v = \{V_1^t, V_2^t, V_3^t, ..., V_n^t\}$ represents the set of n_t number of VMs at time t, and E_t^v represents the set of virtual edges at time t. The vertex of the graph represents one VM and the edge in the graph represents one virtual link. Further, each VN is associated with various parameters such as computing resource requirement, i.e., CPU and memory, network resource requirement, i.e., network bandwidth, and maximum link propagation delay. Let, $R_j^x(t)$ be the computing resource requirement of the VM $V_j^t \in N_t^v$. Here, $x \in \{memory, CPU\}$ represents the resource type, which includes the memory and CPU type. $L_{ij}^v(t)$ represents the virtual link between the VMs V_i^t, and V_j^t, $0 < i, j \leq n_t$, $i \neq j$. The network bandwidth requirement by the virtual link $L_{ij}^v(t)$ is represented by $N_{ij}^r(t)$. Using the value of $N_{ij}^r(t)$, the network bandwidth requirement $N_i^r(t)$ by VM $V_j \in N_t^v$ at time t can be calculated as follows.

$$N_j^r(t) = \sum_{\forall V_i^t \in N_t^v} N_{ij}^r(t), \quad i \neq j \tag{2}$$

In order to access the better quality of service, each virtual link $L_{ij}^v(t)$ is associated with the maximum propagation delay demand $R_{ij}^q(t)$ at any particular time t. This indicates the maximum propagation delay allowed on any physical link, which is selected to be used by a virtual link. In addition to that, user must provide the number of time slots Y that is required to execute the VN.

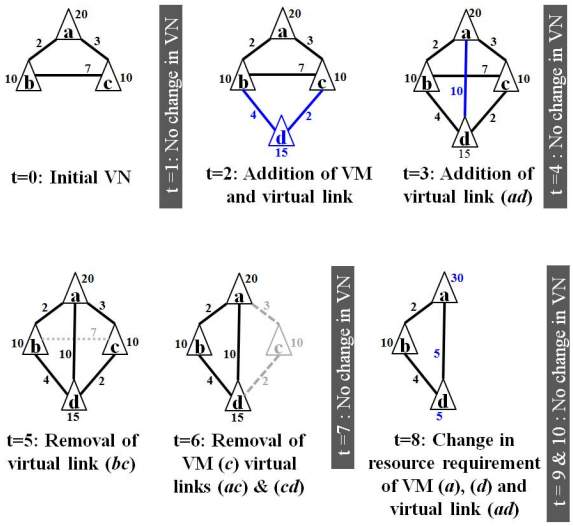


Fig. 2. An example of dynamic virtual network.

C. Dynamic Virtual Network

A VN is said to be dynamic, if the resource configuration and structure of the VN change over time within the specified time slots. Here, the resource configuration is referred to as the computation and network resource requirement of the virtual nodes and virtual links, respectively. The structure of the VN is referred to as the topology of the VN. The dynamic VN that is considered in this paper can be defined as the addition of new virtual nodes or virtual links, change in resource requirement of virtual nodes and virtual links, and removal of virtual nodes and virtual links.

For the better understanding, let us consider an example as presented in the Figure 2. The initial VN is presented in Figure 2(a) with three VMs having the resource demand of 20units, 10units, and 10units, respectively. The virtual links $(ab), (ac),$ and (bc) require the network resource of 2units, 3units, and 7units, respectively. Let, 10 units of time slots be required by the VN to finish its execution. At time $t = 1$, the structure and resource configure remain unchanged. However, at time $t = 2$ as shown in Figure 2(b), one new VM d is added with 15units of resource requirement. As it is assumed that the VN is represented as an undirected graph with no isolated VM, VM d is connected with VMs b, and c with 4, and 2units of network resource demand, respectively. At time $t = 3$, only one new virtual link (ad) is added with 10units of network resource demand as depicted in Figure 2(c). Similarly, dynamic nature can also be referred to as the removal of VMs or only the virtual links. For example, as shown in Figure 2(d) and Figure 2(e), only one virtual link (ad) and one VM c is removed, respectively. Removal of VM c also indicates the removal of virtual links, i.e., (ac) and (cd) as shown in Figure 2(e). The dynamic nature of the VN refers to as the change in the resource requirement of the VMs or the virtual

links. As shown in Figure 2(f), the resource requirement of VM a and d is changed along with the resource requirement of virtual link (ad).

It is believed that the existing VN may change its structure and the resource configuration under two circumstances. Firstly, due to the workload fluctuation, where the workload on the virtual node or on the virtual link may increase or decrease. As a result, the resource requirement of the VN will change. Secondly, due to the user's intervention, where the user may intentionally change the resource configuration of the virtual nodes and the virtual link along with the structure of the VN.

D. Fitness Matrix

Before embedding any VM onto a PS, it is essential to see the left out resources on that PS. For better understanding, let us take an example of two PSs, $PS1$ and $PS2$ with the resource availability of 65 and 60 units, respectively. Let, $VM1$ be the virtual machine with resource demand of 58 units, which needs to be embedded onto either $PS1$ or $PS2$. The remaining resource availability on $PS1$ would be 7 units, if the $VM1$ is embedded onto $PS1$. However, if the $VM1$ is embedded onto $PS2$, the remaining resource would be 5 units. It is obvious that the remaining resources cannot be used in both embedding solutions to create any VM and therefore remain idle. The latter solution is preferred to be the better embedding solution as the remaining idle resource is minimized.

Applying the above scenario onto VNE process, we use the *BestFit* and *Propagation Delay* matrices. The notation $D^x(t) = [d^x_{ij}(t)]$ is used to represent the BestFit matrix i number of rows and j number of columns at time t. Rows refer to as the PSs and the columns refer to as the VMs in the VN. Hence, the dimension of the BestFit matrix would depend on the number of VMs present in the VN at time t and the number of PSs that are eligible and are selected in the embedding process. Computing resource availability of the PSs and computing resource demand of the VMs are taken into account to construct the BestFit matrix. The value $d^x_{ij}(t)$ is calculated by the following equation, which indicates the difference between the resource requirement and resource availability.

$$d^x_{ij}(t) = A^x_i(t) - R^x_j(t), \quad x \in \{memory, CPU\} \tag{3}$$

All the negative values of $D^x_{ij}(t)$ are then replaced with ∞ value, which indicates that the amount of resource available is less than the resource requirement by the corresponding VM, which is represented as follows.

$$d^x_{ij}(t) = \begin{cases} \infty, & \text{if } d^x_{ij}(t) \leq 0 \\ d^x_{ij}(t), & \text{Otherwise} \end{cases} \tag{4}$$

Similar to the BestFit matrix, a three-dimensional propagation delay matrix is used to embed the virtual links onto the physical paths. Propagation delay is defined as the length of time taken for a single data packet to reach at its destination physical server. It is assumed that multiple paths exist between any two physical servers. Let, h be the maximum number of paths between any two physical servers and F^k_{ij} be the $k^{th} \leq h$ path between PSs P_i and P_j. $\mu^k_{ij}(t)$ represents the propagation delay of the k^{th} path between PSs P_i and P_j at time t. Considering different propagation delay values at different time instances among all physical servers, the three-dimensional propagation delay matrix $PD(t)$ is constructed.

$$PD(t) = [pd_{ijk}(t)], 1 \leq i, j \leq |B| \tag{5}$$

$$pd_{ijk}(t) = \begin{cases} \infty, & \text{if } i = j, \forall k \\ \mu^k_{ij}(t), & \text{Otherwise} \end{cases} \tag{6}$$

E. Fitness Values

A new feature called *Local Fitness Value (LFV)* is calculated to quantify the suitability of a PS for a particular VM. Similarly, *Global Fitness Value (GFV)* of the VN is calculated by considering the local fitness value of each VMs, network resource demand and availability of the virtual links and VMs, respectively. The local fitness value of VM V^t_j at time t is calculated as below.

$$\alpha_j(t) = \sum_{i=1}^{m} \kappa^j_i(t) * \frac{d^x_{ij}(t)}{A^x_i(t)} \tag{7}$$

where, $\kappa^j_i(t)$ is the boolean variable that represents if the VM V^t_j is embedded on PS P_i. Mathematically,

$$\kappa^j_i(t) = \begin{cases} 1, & \text{if VM } V^t_j \text{ is embedded on PS } P_i \\ 0, & \text{Otherwise} \end{cases} \tag{8}$$

In the calculation of LFV, only the computing resource availability of type x and computing resource demand by the VMs are considered. However, the GFV is calculated, which includes the LFV of all the VMs, network resource availability and demand of the VMs and PSs, respectively. Mathematically,

$$\beta(t) = \sum_{j=1}^{n_t} \alpha_j(t) + \sum_{i=1}^{m} \sum_{j=1}^{n_t} \kappa^j_i(t) * \frac{N^a_i(t) - N^r_j(t)}{N^a_i(t)} \tag{9}$$

F. Objective Functions

The major objective of the proposed algorithm is to maximize the resource utilization by taking the local fitness values and global fitness values into consideration. This also infers the maximization of the acceptance rate. Resource utilization mainly refers to as the utilization of the computing and network resource. Equations 7 and 9 are used to formulate the objective function mathematically as given below.

Objective:

$$Minimize \sum_{j=1}^{n_t} \min_{1 \leq i \leq m} \quad \kappa^j_i(t) \left(\frac{d^x_{ij}(t)}{A^x_i(t)} + \frac{N^a_i(t) - N^r_j(t)}{N^a_i(t)} \right) \tag{10}$$

Constraints:

$$\sum_{i=1}^{N^p} \kappa^j_i(t) = 1, \quad \forall V^t_j \in N^v_t \tag{11}$$

$$\forall P_i \in N^p, \nexists V^t_j \in N^v_t, A^x_i(t) < R^x_j(t), \quad x \in \{Memory, CPU\} \tag{12}$$

$$\forall L_{ij}^{p} \in E^{p}, \nexists L_{mn}^{v}(t) \in E_{t}^{v}, N_{ij}^{a}(t) < N_{mn}^{r}(t) \quad (13)$$

$$\forall L_{ij}^{p} \in E^{p}, \nexists L_{mn}^{v}(t) \in E_{t}^{v}, \text{ and } \mu_{ij}^{k}(t) > R_{mn}^{g}(t) \quad (14)$$

$$d_{ij}^{x}(t) = A_{i}^{x}(t) - R_{j}^{x}(t) \quad (15)$$

$$0 < d_{ij}^{x}(t) \quad (16)$$

$$0 < t \leq Y \quad (17)$$

$$N^{p} \neq \phi, E^{p} \neq \phi, N_{t}^{v} \neq \phi, E_{t}^{v} \neq \phi \quad (18)$$

$$\forall L_{ij}^{v} \in E_{t}^{v}, N_{ij}^{r}(t) > 0; \quad \forall V_{j}^{t} \in N_{t}^{v}, R_{j}^{x}(t) > 0 \quad (19)$$

The objective function in Equation 10 depends on the minimization of two key values, which needs to be minimized for all the VMs. The first term focuses on the minimizing the remaining computing resources. In other words, the VMs are embedded onto the PSs, where the remaining computing resource after embedding the current VM can be minimized. Similarly, the second term emphasizes on the minimization of the remaining network resource while embedding the virtual links onto the physical paths. Constraint (11) ensures that each VM is processed and is embedded onto exactly one PS. Constraint (12) ensures that the computing resource demand of each VM is valid and no VM exists in the current VN with the computing resource demand more than the maximum resource availability of same type at any PS.

Constraint (13) and Constraint (14) ensures the feasibility of network resource allocation and satisfies the propagation demand of each virtual link, respectively. According to Constraint (15), the resource demand of the VMs and resource availability of the physical servers are considered for calculation of $d_{ij}^{x}(t)$. Additionally, Constraint (16) ensures that the value of $d_{ij}^{x}(t)$ is greater than 0. Negative value indicates the insufficient resource of the corresponding PS. According to the Constraint (17), the current time t must not exceed the value of total number of required time slots, Y at any time instance.

IV. Proposed DYVINE Algorithm

In this section, a novel DYnamic VIrtual Network Embedding (DYVINE) algorithm is proposed, which considers the dynamic nature of the user's VN. The dynamic nature is referred to as the change in the structure and resource configuration of the VMs and virtual links. The VN is embedded in two stages: (a) initial placement of the VMs and virtual links (b) dynamic placement of the VMs and virtual links. For the initial placement of the VN, initial VN Embedding (iVNE) algorithm is designed and is presented in Algorithm 1. For dynamic placement of the VMs and virtual links, DYVINE algorithm is presented in Algorithm 2.

Upon arrival of each VN request at the CSP, the iVNE algorithm is invoked with resource requirement and the required number of time slots Y as the input. iVNE algorithm also requires the resource availability at each PS and physical link. Following the initial embedding of the VNs, the current embedding solution and the updated resource availability at each PS are forwarded to the DYVINE algorithm as the input. DYVINE algorithm remaps the VMs with the objective to minimize the Local Fitness Value (LFV) and Global Fitness Value (GFV). Upon execution of the DYVINE algorithm, value of Y is checked in each time slot. If the value of Y is greater than 0, the VN is checked if there is any change in its structure or in the resource requirement of the VMs or virtual links. It is assumed that the CSP is able to obtain the information regarding any change in the VN. Based on such information, the DYVINE is invoked again in order to remap the existing VMs and virtual links or map the new VMs or virtual links. Following the DYVINE algorithm, the resource availability of the PSs and physical links are updated. However, the current embedding solution remains unchanged if no changes are made in the VNs. In the following sub-sections, the details of the iVNE and DYVINE algorithms are discussed.

A. Initial VN Embedding (iVNE)

In this section, the proposed iVNE algorithm is presented in Algorithm 1. The resource configuration of VN is analyzed in order to embed the VN onto the physical servers and physical links. Before embedding, user must provide the total number of time slots required to finish the execution of the VN, denoted as Y. Since, the physical network is very large in terms of number of physical servers and physical links, it is essential to preprocess the physical network graph G^{p}. Preprocessing is done by removing the physical servers and physical links. The physical servers are removed from the graph G^{p}, which cannot satisfy the network resource requirement of any VM or virtual link. Mathematically,

$$\text{Remove } P_{i}, \text{iff } N_{i}^{a}(t) \leq \max_{\forall V_{j}^{t} \in N_{t}^{v}} N_{j}^{r}(t), \quad \forall P_{i} \in N^{p}, 1 \leq j \leq n_{t} \quad (20)$$

Similar to the removal process of the PS based on the network resource as mentioned in Equation 20, the computing resource availabilities of all PSs are checked and are removed if the corresponding PS cannot host any VM. Mathematically,

$$\text{Remove } P_{i}, \text{iff } A_{i}^{x}(t) \leq \min_{\forall V_{j}^{t} \in N_{t}^{v}} R_{j}^{x}(t), \quad \forall P_{i} \in N^{p} \quad (21)$$

Similarly, the physical link L_{ij}^{p} is removed if the available network bandwidth N_{ij}^{a} is not enough to fulfill the bandwidth demand N_{uv}^{r} of any virtual link. Mathematically,

$$\text{Remove } L_{ij}^{p}, \text{iff } N_{ij}^{a}(t) \leq N_{uv}^{r}(t), \quad 1 \leq i, j \leq m, \; i \neq j, 1 \leq u, v \leq n, u \neq v \quad (22)$$

The resultant preprocessed graph is then stored as graph $G^{p'}$. In order to further reduce the search space or the size of the physical network, a new set of physical servers is formed based on the Equations 23 and 24. Let, B be the set of physical servers, whose resource availability is more than the maximum computing resource requirement among the VMs. Mathematically,

$$B = \left\{ P_{i} | A_{i}^{x}(t) > \max_{\forall V_{j}^{t} \in N_{t}^{v}} R_{j}^{x}, \; N_{j}^{a}(t) > \max_{\forall V_{j} \in N^{v}} N_{j}^{r}(t), P_{i} \in G^{p'} \right\} \quad (23)$$

Let, B' be the set of physical servers, whose resource availability is more than the minimum computing resource requirement among all the VMs. Mathematically,

$$B' = \Bigg\{ P_i | A_i^x(t) > \min_{\forall V_j^t \in N_t^v} R_j^x(t), \\ N_j^a(t) > \max_{\forall V_j^t \in N_t^v} N_j^r(t), P_i \in G^{p'} \Bigg\} \quad (24)$$

The set B and B' are sorted in ascending order based on the availability of the computing resource. From the sorted set, union operation is performed on $|N_t^v|$ number of physical servers from each of the sets B and B'. Hence, the resultant set B contains maximum $2 * |N_t^v|$ number of physical servers.

The iVNE algorithm is executed by receiving the dynamic VN and physical network as inputs. The bandwidth requirement of each VM is calculated as given in Line 1 followed by finding the maximum bandwidth requirement, as in Line 2. The physical servers that cannot host any VMs are removed and are stored in the set $N^{p'}$ as in Line 5. A new set of PS is formed by applying the Equation 23 and 24 on the set $N^{p'}$ as in Line 10. BestFit matrix is constructed in Line 11–12. Similarly, a three-dimensional propagation delay matrix is constructed as given in Line 13. Using the BestFit and propagation delay matrix, the VMs and virtual links are embedded. The embedding process starts by finding the minimum value in the BestFit matrix as in Line 18. The VM in the corresponding column is now embedded onto the PS in the corresponding row as in Line 20. After each VM embedding, the iVNE algorithm checks if any virtual link is not yet embedded, whereas the attached two VMs are already embedded. Such virtual links are embedded by finding a suitable path based on the propagation delay value in the matrix $PD(t)$ as given in Line 23–26. Following the embedding of each VM and virtual link, the resource availability of the PS and physical links are updated as given in Line 29.

Theorem 1: The time complexity of iVNE algorithm with n_t number of VMs and m number of PSs at any time t is $\mathcal{O}(m \log m + n_t^3)$.

Proof: Upon arrival of each VN at the CSP, iVNE algorithm is invoked for the initial placement of the VMs and virtual links. Calculation of bandwidth requirement of each VM is given in Algorithm 1 in Line 1, followed by searching the VMs with maximum bandwidth requirement, which requires a running time of $\mathcal{O}(n_t)$ as given in Line 2. Similarly, the running time of calculation of bandwidth availability of each PS is $\mathcal{O}(m)$. In the worst case scenario, all PS may fulfill the requirement of each VM and therefore no PS is removed as given in Line 4. However, the running time to check each PS's resource availability is $\mathcal{O}(m)$. Presuming that no PS is removed due to resource availability, the running time of sorting the set of PS is $\mathcal{O}(m \log m)$ as given in Line 6. This can be achieved by using $HEAPSORT$. In order to reduce the embedding time and PS search space, a new set B is formed with the running time $\mathcal{O}(2n_t)$. Since, the maximum number of PSs in the set B is $2 * n_t$, the running time for constructing and updating the BestFit matrix is $\mathcal{O}(2n_t * n_t) = \mathcal{O}(2n_t^2)$, which is given in Lines 11 and 12. Similarly, construction of the propagation delay matrix would require a running time of $\mathcal{O}(2n_t * 2n_t * h) = \mathcal{O}(4hn_t^2)$. Here, it is assumed that all PSs have a maximum of h number of physical paths. The *while* loop given in Lines 16-33 repeat for n_t number of times. Further, to find minimum value in BestFit matrix and corresponding value of i and j requires a running time of $\mathcal{O}(2n_t^2)$ as given in Line 18. Hence, the running time for embedding all VMs and adjacent virtual links can

Algorithm 1: Initial Virtual Network Embedding (iVNE) Algorithm

Data: Dynamic VN :$G_t^V(N_t^V, E_t^V)$, Physical network:$G^P(N^P, E^P)$

1 Calculate bandwidth requirement of each VM (Equation 2) ;
2 $maxBWreq = \max\{N_j^r(t)\}$;
3 Calculate bandwidth availability at each PS (Equation 1) ;
4 $N^{p'}$ = Remove the PS using Equation 20 ;
5 Update the set $N^{p'}$ using Equation 21 ;
6 Sort the set $N^{p'}$ on computing resource availability ;
7 Form set B and B' using Equation 23 and 24, respectively ;
8 B = Extract $|N_t^v|$ number of PSs from set B ;
9 B' = Extract $|N_t^v|$ number of PSs from set B' ;
10 $B = B \cup B'$;
11 Construct BestFit matrix D_t^x using Equation 3 ;
12 Update the D_t^x using Equation 4 ;
13 Construct Propagation Delay matrix PD using Equation 5 ;
14 $TG(tv, te)$ = Temporary empty graph; $tv = \{\}$ be the empty set of vertex; $te = \{\}$ be the empty edges set ;
15 $totUnMappedVMs = |N_t^v|$;
16 **while** $totUnMappedVMs > 0$ **do**
17 Initialize $i = 0, j = 0$;
18 Find i and j, such that $d_{ij}^x(t)$ in the matrix $D^x(t)$ is minimum ;
19 **if** *VM V_j^t is NOT embedded onto any PS* **then**
20 Embed VM V_j^t on PS P_i ;
21 $tv = tv \cup V_j^t$;
22 $\forall V_k^t \in tv, if\ (L_{jk}^v(t) = 1, L_{jk}^v(t) \in E_t^v)$, $te = te \cup L_{jk}^v(t)$;
23 **foreach** virtual link $L_{jk}^v(t) \in te$
24 s = PS on which V_k^t is embedded ;
25 $\min_{\forall r}\{pd_{isr}(t) \in PD(t)\}$;
26 embed $L_{jk}^v(t)$ on path F_{is}^r ;
27 Update the value of $N_{is}^a(t)$;
28 $totUnMappedVMs = totUnMappedVMs - 1$;
29 Update the resource availability of PS P_i and P_s ;
30 **else**
31 Repeat from the Step 18 ;
32 **end**
33 **end**

be derived as $\mathcal{O}(n_t * 2n_t^2) = \mathcal{O}(2n_t^3)$. The total running of iVNE algorithm can be concluded as follows.

$$\mathcal{O}(n_t) + \mathcal{O}(m \log m) + \mathcal{O}(4hn_t^2) + \mathcal{O}(2n_t^3)$$

Since, the number of paths between any two PSs is constant, the time complexity of the iVNE algorithm at time t can be rewritten as $\mathcal{O}(m \log m + 2n_t^3) = \mathcal{O}(m \log m + n_t^3)$. □

B. Dynamic VN Embedding

Following the initial placement or embedding of the VMs onto the physical network, we propose the DYVINE algorithm that remaps the VMs and the virtual links with the objective to improve the resource utilization.

The DYVINE algorithm is executed by calculating the LFV and GFV of all VMs and VNs, respectively, as given in Lines 1-2. $StopCond$ is used as a conditional variable that indicates when to stop the algorithm execution. In each iteration, $StopCond$ variable is set to $TRUE$ assuming that this is the last iteration where no further minimization of β is possible, as given in Line 5. However, due to migration of at least one valid VM, $StopCond$ variable is reset to be $FALSE$, as given in Line 19.

In each while loop as given in Line 4, each VM V_j^t is examined with all PSs as follows. At first, it is assumed that the VM is migrated to a PS p as presented in Line 8. If any other VM $tv \in N_t^v$ is already embedded onto the same PS p, the current VM and VM tv are treated as one VM, i.e., $tv1$. The new resource requirement $R_{tv1}^x(t)$ of $tv1$ is calculated as the sum of the resource requirements of the VM tv and V_j^t, which is given in Line 11. Mathematically,

$$R_{tv1}^x(t) = R_j^x(t) + R_{tv}^x(t) \tag{25}$$

Similarly, the network resource requirement $N_{tv1}^r(t)$ of VM $tv1$ is calculated by subtracting the network resource requirement of the virtual link from the sum of the total network resource requirement of both VMs as given in Line 11. Mathematically,

$$N_{tv1}^r(t) = \left(N_j^r(t) + N_{tv}^r(t)\right) - \left(2 * N_{(j\ tv1)}^r(t)\right) \tag{26}$$

Following the calculation of new resource demand, the LFV and GFV of the VMs and VNs are calculated as given in Line 12. If the newly calculated GFV value nb is less than the previous GFV value β, the current VM migration is considered as a valid and better embedding solution. However, if the newly calculated GFV value nb is greater than the previous GFV value β, the current VM migration is discarded or canceled. The situation may also arise that no VM from the set of VMs N_t^v is embedded onto the current PS p as given in the *else* case in Line 13. In such case, only the new LFV and GFV is calculated as given in Line 14.

Upon embedding the VMs and virtual links onto its optimized PS, the DYVINE algorithm waits for the current time slot to finish. The remaining required time slot Y is decremented by 1 unit at the beginning of each time slot as given in Line 23. It also checks if any VM or the virtual link is added to the current VN as given in Line 24. If a new VM is added, the set of VM N_t^v at current time t must have more number of VMs than the set of VMs at time $t-1$, N_{t-1}^v. Similarly, if a new virtual link is added to the existing VN at time t, the number of links in set E_t^v at time t must contain more number of links than in the set E_{t-1}^v. If the VN is changed by adding new VMs or virtual links, the VN graph G_t^v is reconstructed. The new VN graph is further forwarded to the iVNE function as presented in Algorithm 1. The new VMs and virtual links are only embedded onto the PSs using the BestFit and propagation delay matrix, as given in Line 25. The existing VMs and virtual links are not embedded again.

Algorithm 2: DYnamic VIrtual Network Embedding (DYVINE) Algorithm

Data: Dynamic VN :$G_t^V(N_t^V, E_t^V)$; Physical network:$G^P(N^P, E^P)$; Remaining required time slots Y

1 Calculate LFV of all VMs using Equation 7 ;
2 Calculate GFV of VN at time t, $\beta(t)$ using Equation 9 ;
3 $StopCond = FALSE$;
4 **while** $StopCond = FALSE$ **do**
5 $StopCond = TRUE$;
6 **foreach** VM $V_j^t \in N_t^v$
7 **foreach** PS p
8 Migrate V_j^t to PS p ;
9 **if** another VM $tv \in N_t^v$ is embedded to PS p
10 Treat tv and V_j^t as one VM $tv1$;
11 Calculate the new values of $R_{tv1}^x(t)$ and $N_{tv1}^r(t)$ using Equation 25 and 26, respectively ;
12 Calculate LFV of $tv1$, $\alpha_{tv1}(t)$ and GFV of VN, nb ;
13 **else**
14 Calculate LFV of tv, $\alpha_{tv}(t)$ and GFV of VN nb ;
15 **if** $\beta(t) < nb$
16 Undo the last VM migration in Line 8 ;
17 **else**
18 Consider current solution as valid. ;
19 $StopCond = FALSE$;
20 **end**
21 **while** $Y > 0$ **do**
22 Wait for current time slot to finish ;
23 $Y = Y - 1$;
24 **if** $|N_t^v| > |N_{t-1}^v|$ *OR* $|E_t^v| > |E_{t-1}^v|$ **then**
25 Reconstruct G_t^V and Invoke Algorithm 1 and embed only new VMs and virtual links ;
26 Repeat the steps from Line 1 ;
27 **end**
28 **if** $|N_t^v| < |N_{t-1}^v|$ *OR* $|E_t^v| < |E_{t-1}^v|$ **then**
29 Release the corresponding resources ;
30 **end**
31 **if** $\forall V_j^t, R_j^x(t) \neq R_j^x(t-1)$ *OR* $N_j^r(t) \neq N_j^r(t-1)$ **then**
32 Repeat the steps from Line 1 ;
33 **end**
34 **end**

The output of the iVNE algorithm is then used by the DYVINE algorithm and the steps given in Line 1 are repeated.

However, the structure of the VN can be changed by removing the VMs or the virtual links. As presented in Line 28, it checks if any VM is removed by comparing the number of elements in the set N_t^v at time t and set N_{t-1}^v at time $t-1$. $N_t^v > N_{t-1}^v$ indicates the removal of at least one VM from the VN. Similarly, $E_t^v > E_{t-1}^v$ indicates the removal of at least one virtual link from the VN. Upon removal of a VM or virtual link, the corresponding computing resource and the network resource are released. The resource configuration of VN is checked as given in Line 31 by comparing the computing resource demand $R_j^x(t)$ at time t and $R_j^x(t-1)$ at time $t-1$. Similarly, the network resource demand is checked by comparing the value of $N_j^r(t)$ at time t and the value of $N_j^r(t-1)$ at time $t-1$. If the resource configure is changed, the DYVINE algorithm invokes itself with the new computing and network resource demands. This process is continued until the remaining required time slot Y is not equal to 0. In other words, the DYVINE algorithm runs for Y number of time slots. For the sake of better understanding, the DYVINE algorithm is explained with an example in Section IV-C.

Lemma 1: The total service time for any VN with Y number of required time slots can be calculated as

$$\gamma = Y * (\omega_2 + \Delta t) + \omega_1 \tag{27}$$

where, Δt is the duration of each time slot. ω_1 is the required duration to embed the VN upon its arrival and ω_2 is the duration to update the embedding solution in DYVINE algorithm due to change in the number of VNs.

Proof: DYVINE algorithm assumes that the duration of each time slot is uniform and upon arrival of any VN, the initial VNE requires less time than the duration of each time slot. Mathematically,

$$\omega_1 < \Delta t \tag{28}$$

In order to make the proposed VNE scheme feasible and cost effective, the following relation must hold

$$\omega_2 < \Delta t, \tag{29}$$

which indicates that DYVINE should ensure that the re-embedding the VN must require less time than the time length of each time slot, Δt. Considering the relation given in Equation 28 and 29, the total embedding time can be calculated as the $\omega_1 + (Y * \omega_2)$.

The total time length required for each VN to finish its execution can be calculated as $Y * \Delta t$. The service time, γ, represents the time required to embed the VN upon its arrival, the re-embedding time in each time slot and the total execution time. It is assumed that during re-embedding the VN in each time slot, the execution of the VN needs to be suspended. Under such circumstances, the total service time can be calculated as the sum of total execution time and the total embedding time. Mathematically,

$$\begin{aligned}\gamma &= (Y * \Delta t) + \omega_1 + (Y * \omega_2) \\ &= Y * (\omega_2 + \Delta t) + \omega_1\end{aligned} \tag{30}$$

□

Theorem 2: Maximizing the resource utilization maximizes the acceptance ratio.

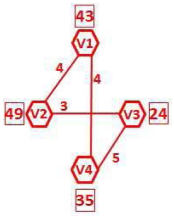
Proof: Acceptance ratio refers to as the ratio between the number of VN requests being accepted and the total number of VN requests received by the CSP. Here, resource utilization refers to as the utilization of available resources by creating heterogeneous VMs for the users. Creating heterogeneous VMs onto the PS leads to obvious resource fragmentation. Minimizing the amount of fragmented resource allows the CSP to create more number of VMs. For example, let's assume, CSP received a VM request with 7 units of resource requirement. The VM cannot be created if the 7 units of resources are distributed over multiple PSs. However, the VM can be created, if the same amount of resource is available on single PS. Inspire from such scenario, we are using BestFit matrix to find a suitable PS for each VM with the objective to minimize remaining resource availability. In other words, we are emphasizing on minimizing the fragmented resource, which enables the CSP to create more number of VMs. This infers the maximization of acceptance ratio. □

Theorem 3: Proposed VNE algorithm gives near optimal result for the VN due to optimal placement of each corresponding VM.

Proof: The objective of the proposed DYVINE algorithm is to maximize the resource utilization. It follows the greedy approach for embedding the VMs and virtual links. The embedding process starts with embedding each VM and adjacent virtual links. For each VM, DYVINE selects the optimal physical server in terms of resource utilization by considering the resource requirement of the VM and resource availability of the PS by using the BestFit matrix, where the remaining resource is minimum. The same procedure is followed to find the next best VM-PS pair. Further, DYVINE also makes locally optimal selection of physical links for each virtual links as given in Algorithm 1. With the intention to make the embedding solution more efficient in terms of the resource utilization, the resource availability of the selected PSs is further compared with the resource demand of the VMs as given in Algorithm 2. The intention of making locally optimal selection of PSs and physical links for each VM and virtual link is to find the global optimal solution for the entire virtual network. In doing so, DYVINE may not provide the optimal solution to the classical NP-hard VNE problem. However, this greedy algorithm can produce a near-optimal solution, which is established in the performance evaluation. □

Theorem 4: The time complexity of DYVINE algorithm with n_t number of VMs and m number of PSs at time t is $\mathcal{O}(n_t^3)$.

Proof: The DYVINE algorithm is executed by calculating the LFV of all VMs as given in Line 1, which requires the running time of $\mathcal{O}(n_t)$. Following this, the running time of calculation of GFV of the VN at time t is constant. Minimization of $\beta(t)$ has the time complexity of $\mathcal{O}(2n_t - 1)$, which continues for n_t number of times. Hence, the time complexity of *foreach* loop can be written as $\mathcal{O}(2n_t^2)$. After each movement of a VM, the steps from the Line 6 are repeated. Hence, the algorithm would stop its execution only when the LFVs of all the VMs are checked. This requires



(a) Example of VN graph

VMs	$R_i^x(0)$	$N_j^r(0)$
V1	43	8
V2	49	7
V3	24	8
V4	35	9

(b) VMs with resource demand

PSs	$A_i^x(0)$	$N_j^q(0)$
P5	25	20
P6	27	33
P7	27	34
P8	28	34
P15	50	23
P16	60	27
P17	75	36
P18	77	36

(c) Selected list of PSs

Fig. 3. Example of DYVINE: VN and list of PSs. (a) Example of VN graph (b) VMs with resource demand (c) Selected list of PSs.

	V1	V2	V3	V4
P5	-18	-24	1	-10
P6	-16	-22	3	-8
P7	-16	-22	3	-8
P8	-15	-21	4	-7
P15	7	1	26	15
P16	17	11	36	25
P17	32	26	51	40
P18	34	28	53	42

	V1	V2	V3	V4
P5	∞	∞	1	∞
P6	∞	∞	3	∞
P7	∞	∞	3	∞
P8	∞	∞	4	∞
P15	7	1	26	15
P16	17	11	36	25
P17	32	26	51	40
P18	34	28	53	42

Fig. 4. Example of BestFit matrix.

another n_t number of calculations. As a result, the total time complexity of the DYVINE algorithm can be concluded as $\mathcal{O}(2n_t^2 * n_t) = \mathcal{O}(n_t^3)$. □

C. Example of DYVINE

For the sake of better understanding, let us take an example as given in Figures 3–5. As shown in Figure 3(a), a VN graph consists of $n_t = 4$ VMs, i.e., $V1, V2, V3$ and $V4$ with the initial resource requirement of 43 units, 49 units, 24 units, and 35 units, respectively. In Figure 3(b), the network resource requirement of VM $V1, V2, V3$, and $V4$ are 8 units, 7 units, 8 units, and 9 units, respectively, which are calculated using Equation 2. Similarly, let the physical network consists of hundreds of PSs. However, not all the PSs are eligible to host all the VMs. Hence, based on the Equations 20 – 22, the ineligible PSs and physical paths are removed. Further to reduce the embedding time, Equation 23 and 24 are applied onto the set of PS and only a maximum of $2 * n_t = 2 * 4 = 8$ number of PSs are selected as given in the Figure 3(c).

Each selected PS has enough network resource to satisfy the network resource demand of any VM. Based on the resource availability and resource requirement of all VMs, the BestFit matrix is constructed using the Equation 3. In the example given in 4, the first cell value is calculated by subtracting the resource demand of VM $V1$ from the resource availability of PS $P5$, which is -18. Similarly, the cell value that corresponds to the VM $V4$ and PS $P8$ is calculated as -7. The negative cell values are modified and are replaced with ∞. In the given example, the first cell value is changed to ∞ from -18. The ∞ value represents that the VM cannot be embedded onto the corresponding PS.

The propagation delay matrix is constructed using Equation 5 as presented in Figure 5(a). In this example, it is assumed that a maximum of three number of paths exist

	P5			P6			P7			P8			P15			P16			P17			P18		
	μ^0	μ^1	μ^2	μ^0	μ^1	μ^2	μ^0	μ^1	μ^2	μ^0	μ^1	μ^2	μ^0	μ^1	μ^2	μ^0	μ^1	μ^2	μ^0	μ^1	μ^2	μ^0	μ^1	μ^2
P5	0	0	0	0.73	0.8	0.26	0.35	0.73	0.46	0.96	0.91	0.61	0.95	0.09	0.38	0.87	0.91	0.26	0.49	0.65	0.67	0.29	0.75	0.79
P6	0.73	0.8	0.26	0	0	0	0.09	0.2	0.72	0.61	0.13	0.18	0.28	0.15	0.92	0.61	0.02	0.56	0.42	0.65	0.79	0.39	0.44	0.57
P7	0.35	0.73	0.46	0.09	0.2	0.72	0	0	0	0.59	0.13	0.28	0.91	0.27	0.96	0.02	0.9	0.86	0.84	0.6	0.11	0.95	0.55	0.58
P8	0.96	0.91	0.61	0.61	0.13	0.18	0.59	0.13	0.28	0	0	0	0.01	0.03	0.41	0.54	0.75	0.53	0.3	0.9	0.48	0.79	0.02	0.44
P15	0.95	0.09	0.38	0.28	0.15	0.92	0.91	0.27	0.96	0.01	0.03	0.41	0	0	0	0.2	0.18	0.15	0.71	0.28	0.71	0.29	0.88	0.07
P16	0.87	0.91	0.26	0.61	0.02	0.56	0.02	0.9	0.86	0.54	0.75	0.53	0.2	0.18	0.15	0	0	0	0.58	0.91	0.33	0.98	0.83	0.25
P17	0.49	0.65	0.67	0.42	0.65	0.79	0.84	0.6	0.11	0.3	0.9	0.48	0.71	0.28	0.71	0.58	0.91	0.33	0	0	0	0.98	0.41	0.38
P18	0.29	0.75	0.79	0.39	0.44	0.57	0.95	0.55	0.58	0.79	0.02	0.44	0.29	0.88	0.07	0.98	0.83	0.25	0.98	0.41	0.38	0	0	0

(a) Propagation delay matrix

	iVNE Output	DYVINE	
		Iteration 1	Iteration 2
P5	VM V3	V3 $\alpha_3 = 0.04$	
P6			
P7			
P8			
P15	VM V2	V2 $\alpha_2 = 0.02$	V2 $\alpha_2 = 0.02$
P16	VM V1	V1 $\alpha_1 = 0.28$	V1 $\alpha_1 = 0.28$
P17	VM V4	V4 $\alpha_4 = 0.53$	V3, V4 $\alpha_{34} = 0.21$
P18			
		Total $\beta = 3.61$	Total $\beta = 2.32$

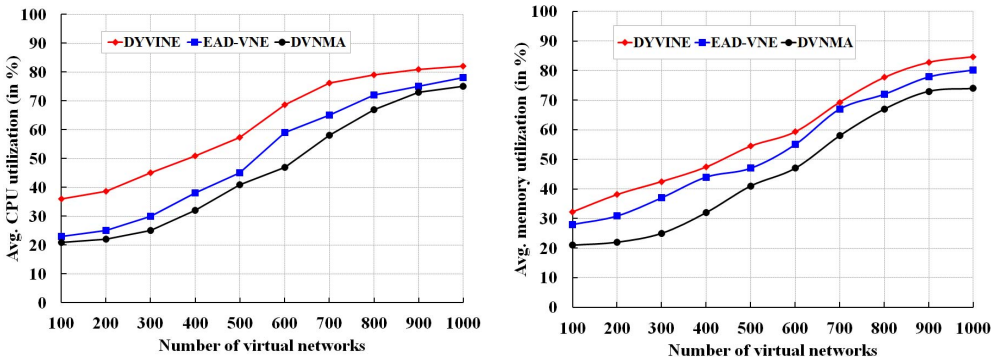
(b) Output of iVNE and DYVINE algorithm

Fig. 5. Example of DYVINE. (a) Propagation delay matrix (b) Output of iVNE and DYVINE algorithm.

between each pair of PSs. For clear understanding of the proposed algorithm, random propagation delay values are used in the given example. For example, the three paths between PS $P5$ and $P6$ have propagation delay of $0.73, 0.8$, and 0.26 units, respectively.

According to Algorithm 1, the first minimum value is searched in BestFit matrix. The cell value that corresponds to the VM $V3$ and PS $P5$ is the minimum value. Hence, VM $V3$ is embedded first onto the PS $P5$ as the remaining corresponding value is minimum. The minimum value indicates that the embedding of VM $V3$ onto the PS $P5$ will leave the minimum left-over resource. After making the decision to embed the VM $V3$ onto the PS $P5$, the next minimum value is searched. However, for further search of the next minimum value, the same PS is not considered. Hence, the values in the row that corresponds to the PS $P5$ are discarded. The VM $V2$ is now embedded onto the PS $P15$ as the corresponding cell value is minimum. After embedding two VMs, it is now observed that one virtual link with both adjacent already-embedded VMs needs to be embedded. In the given example, the virtual link between the VM $V3$ and $V4$ needs to be embedded. In order to embed the virtual link, the propagation delay matrix is used to find the physical path with minimum propagation delay value. The propagation delay of the path μ^1 between the PS $P5$ and $P15$ is 0.09 unit, which is the minimum value among other two paths. Hence, the path μ^1 between the PS $P5$ and $P15$ is used to embed the virtual link between the VM $V3$ and $V2$. The process of searching the next minimum value in the BestFit matrix is repeated and the VM $V1$ is embedded onto the PS $P16$. Upon embedding the VM $V1$, the virtual link between $V1$ and $V2$ is embedded. The same procedures are followed to embed other VMs and virtual links.

The output of iVNE algorithm is shown in Figure 5(b). This output is now used as the input to the DYVINE algorithm. The DYVINE algorithm is executed with the calculation of the value of LFV (α_j) of each VM using Equation 7. Here, the LFV of VM $V1$ that is embedded onto the PS $P16$ is $\alpha_1 = 0.28$, which is calculated by $\frac{60-43}{60}$. Similarly, LFVs of VM $V2$, $V3$, and $V4$ are calculated as $\alpha_2 = 0.02, \alpha_3 = 0.04$, and $\alpha_4 = 0.53$, respectively. Following the calculation of LFV of each VMs, the GFV of the VN (β) is calculated using Equation 9. In the given example as shown in Figure 5(b), the value of β is calculated as 3.61. The objective of the Algorithm 2 is to minimize the value of β. For this, every



(a) CPU resource utilization (b) Memory resource utilization

Fig. 6. Comparison in terms of computing resource utilization. (a) CPU resource utilization (b) Memory resource utilization.

VM is examined. For example, the VM $V3$ is migrated to the PS $P17$. By such migration, both VMs can be combined logically and is treated as one VM, $V34$. As both VMs are on one PS, the network bandwidth requirement between the VM $V3$ and $V4$ can now be ignored. Hence, the new network bandwidth requirement of the VM $V3$ is $N_3^r = 3$ units and VM $V4$ is $N_4^r = 4$ units after combining both VMs. The new LFV of VM $V34$ is calculated as $\frac{75-(24+35)}{75} = 0.21$. Considering the value of LFV of each VMs and new network resource demand of VMs, the new β value of the entire VN is calculated as 2.60. The new β value is smaller than the previous β value of 3.61. Hence, the current embedding solution is considered as efficient one. The same procedure of migrating the VM is followed in order to further minimize the value of β. However, in this example, no further movement is possible, which can minimize the value of β and the current solution is considered as the final one at current time instance.

V. Performance Evaluation

In this section, the performance evaluation of the proposed VNE algorithm is presented. EAD-VNE [10] and DVNMA [11] VNE algorithms are considered to compare with our work. The EAD-VNE algorithm presents a VN embedding mechanism considering the fluctuation in the resource demand of the VMs and virtual links with the objective to optimize the energy consumption leading to maximization of the revenue generation. The DVNMA algorithm addresses the issue of embedding the dynamic VNs with the objective to minimize the embedding cost.

A. Simulation Setup

The cloud environment is equipped with 128 numbers of heterogeneous physical servers. The resource availability at each PS is distributed randomly in the range between 15 GB and $40GB$ of memory and the number of CPUs ranges between 8 units and 32 units of CPUs. The Fat-Tree network topology is used for the communication purpose among the PSs. The available bandwidth resource of each physical path is randomly distributed from $50Mbps$ through 1 $Gbps$. The propagation delay of each physical path is represented as milliseconds (ms), which ranges from $0.01ms$ through 5 ms.

On constructing the VN request, the number of VMs ranges from 2 through 10 VMs. The arrival and lifetime of the VNs follow the Poisson and exponential distribution, respectively. The mean arrival rate of the VNs is 5 per 100 time units. The lifetime of a VN (Y) is referred to as the total number of time slots required to finish its execution, which is 100 time units. The virtual links are created among the VMs with the connectivity probability of 0.7. The link connectivity probability can be used to calculate the total number of virtual links present in the VN. For each virtual link, the propagation delay ranges from $0.1ms$ and 2.5 ms. The random distribution of the bandwidth ranges between 100 $Kbps$ through 500 $Kbps$, which is used as the unit of network bandwidth resource of the VNs. The computing resource requirement of the VMs is randomly distributed ranging from $500MB$ through 4 GB of memory and 1 unit through 4 units of CPUs.

B. Simulation Results

The proposed algorithm is compared with EAD-VNE [10] and DVNMA [11] algorithms in order to evaluate its performance. With a constant of 8 number of VMs in each VN, the average resource utilization using all the three algorithms is compared as shown in Figure 6. Resource utilization is calculated in percentage, which refers to as the percentage of the total resource allocated to the VNs. The average CPU resource utilization of all the active PSs is presented in Figure 6(a). It is observed that the proposed DYVINE algorithm outperforms over other two VNE algorithms in terms of CPU resource utilization. The average CPU utilization is increased to more than 80% from 36%, when the number of VNs are increased to 1000 from 100. The CPU resource utilization in case of EAD-VNE and DVNMA algorithm is less than 80%, when the number of VNs is 1000. The major reason behind such efficient CPU utilization is due to its ability to embed the VMs in a compact manner using the BestFit approach. It exploits the opportunity to embed multiple VMs onto the PS, where the remaining CPU resource can be minimized. As a result, the CPU utilization is minimized. Similarly, as presented in Figure 6(b), the memory utilization of the proposed DYVINE algorithm is 32% and 85%, when the number of VNs is 100 and 1000, respectively. By observing both Figures 6(a) and 6(b), it is concluded that the proposed

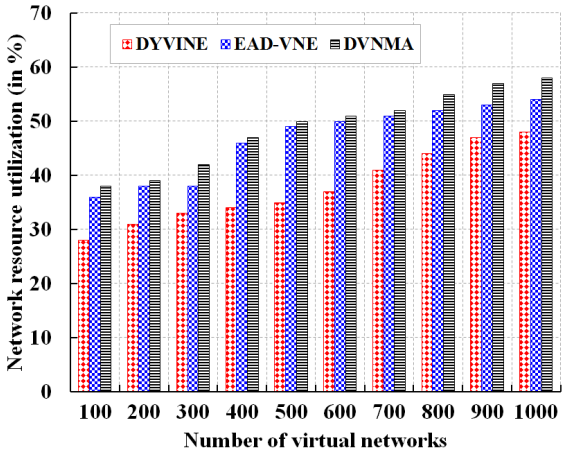


Fig. 7. Comparison in terms of network resource utilization.

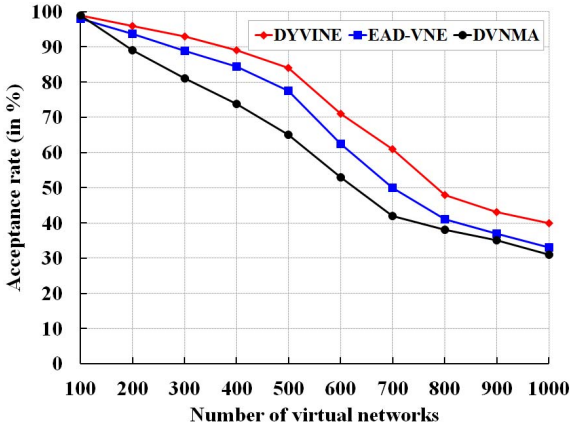


Fig. 8. Acceptance rate (in %).

DYVINE algorithm gives efficient result as compared to the EAD-VNE and DVNMA algorithms in terms of resource utilization.

The performance of the proposed algorithm is compared with other two algorithms in terms of network resource utilization, as shown in Figure 7. The network resource utilization refers to as the percentage of the total amount of network resource is being allocated to the VNs. In case of DYVINE, the amount of network resource utilization increases from approximately less than 30% to nearly 50%, when the number of VNs increases from 100 to 1000 with an average of 8 VMs per VN. The link connectivity probability in each VN is kept constant at 0.7. For 100 VNs, the network utilization of EAD-VNE and DVNME algorithm lie in the range of 35% and 40%. However, the utilization increases to more than 50%, when the number of VNs increases to 1000. The lower network resource utilization in case of DYVINE algorithm is due to the placement of multiple VMs of same VN onto the single PS, due to which the resource demands of the corresponding virtual links are ignored.

The acceptance rate of the CSP has direct impact on the generated revenue. Higher acceptance rate refers to as the maximum number of VNs being accepted by the CSP with constant number of total PSs. In Figure 8, the acceptance rate is more than 99%, when the number of VNs is 100. However, the acceptance rate decreases when the number of VNs increases. It is observed that the acceptance rate in case of proposed DYVINE algorithm is approximately 40%, when the number of VNs increases to 1000, whereas the acceptance rate is below 34% and 32% in case of EAD-VNE and DVNMA algorithms, respectively for 1000 number of VNs.

In order to verify the efficiency in terms of network resource usage of the proposed VNE algorithm, the data regarding the network resource demand are obtained and the amount of average network resource allocated to the VNs is simulated as shown in Figure 11. For 100 VNs with a mean of 3964 $Kbps$ of network resource demand per VN, an average of $3600Kbps$ of actual network resource is allocated to each VN, which saves approximately 10% of the network resource. This efficiency is achieved due to the embedding of multiple VMs from the same VN onto the single PS. As a result, the demands of the corresponding virtual links are ignored and are not embedded onto any physical path. The similar trend is observed even when the number of VNs is increased to 1000, where the average network resource demand is 5147 $Kbps$ and an average of 4800 $Kbps$ of network resource is allocated. This saves approximately 7% of the total network resource. This efficiency plays a major role in increasing the acceptance rate.

As discussed in Section V-A, the embedding time refers to as the total time taken by the VNE algorithm to embed the current VNs. In this work, the embedding time includes the time taken by the VNE algorithm to examine the changes made in the structure and configuration of the VNs, re-embedding the existing VMs and virtual links according to the BestFit approach, embedding the new VMs and virtual links etc. The embedding time in case of all three algorithms are depicted in Figure 9. It is observed that all three VNE algorithms take approximately $10 - 20ms$ to embed 200 number of VNs. However, the efficiency of the proposed algorithm is observed when the number of VNs increases to 1000. To embed 1000 number of VNs with a constant of 8 number of VMs in each VN, the proposed DYVINE algorithm takes approximately $50ms$, which is $85ms$ and $78ms$ in case of EAD-VNE and DVNMA algorithms, respectively. The major reason behind such less embedding time is due to the involvement of very less number of PSs. In the proposed algorithm, only maximum of $2n_t$ numbers of PSs are chosen to embed the current VN at any time instance t. As a result, DYVINE algorithm gives the desired result in very less time.

Figure 10 represents the relationship between the size of each VN, the number of PSs that are used in the set B in Algorithm 1, and the resource utilization. Here, the impact of VN size and length of set B or number of PSs onto the utilization of CPU and memory resources are studied as depicted in Figures 10(a) and 10(b), respectively. As discussed earlier, the resource utilization is referred to as the percentage of total amount of resource allocated to all the VNs. The number of VNs is kept constant at 500. The VN size n_t at any time instance t varies from 2 VMs through 10 VMs. The number of PSs varies from $2n_t$ through $8n_t$ PSs. The resource utilization slightly decreases when more number of PSs is considered as given in Line 10 of Algorithm 1. The

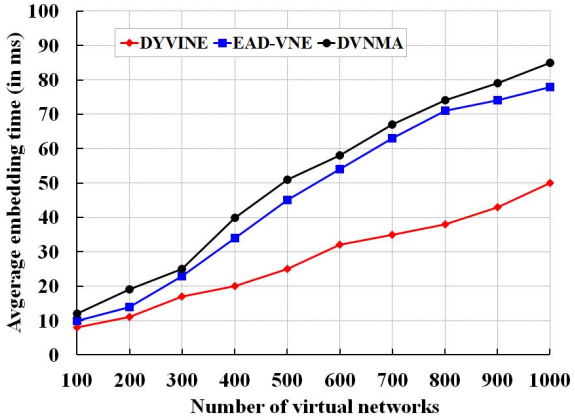
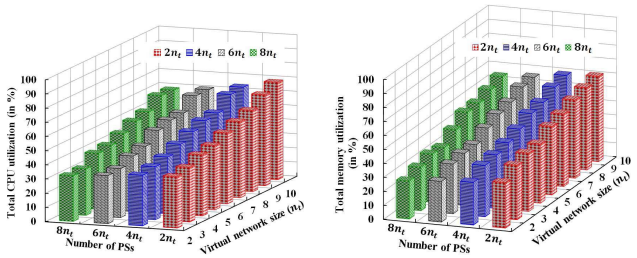


Fig. 9. Average embedding time (in milliseconds).



(a) CPU resource utilization (b) Memory resource utilization

Fig. 10. Computing resource utilization with variable VN size and number of PSs. (a) CPU resource utilization (b) Memory resource utilization.

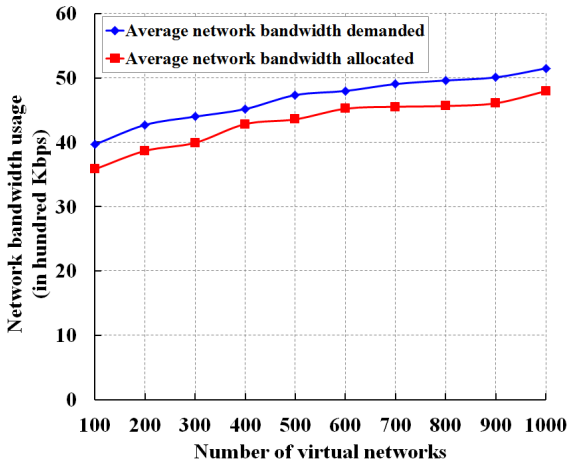


Fig. 11. Network resource usage.

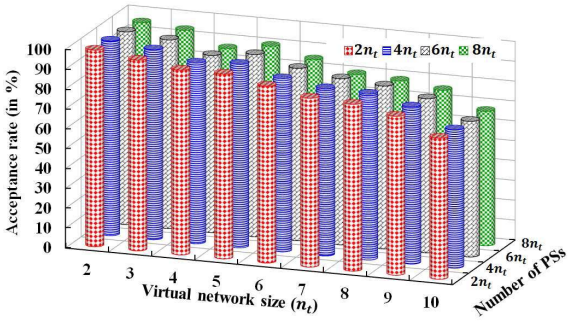


Fig. 12. Acceptance rate with variable VN size and number of PSs.

CPU utilization is approximately 36%, when the VN size is 2 VMs and only $2n_t = 4$ number of PSs are considered. With constant VN size, the resource utilization slightly decreases to 32%, when the number of PSs increases to $8n_t = 16$. Similarly, the CPU utilization decreases from approximately 68% to 60%, when the number of PSs increases from $2n_t = 4$ to $8n_t = 16$ with constant VN size of 10 VMs.

The similar trend is also observed in case of memory utilization as shown in Figure 10(b). The memory resource utilization is approximately 32%, when the VN size is 2VMs and $2n_t = 4$ number of PSs are considered. However, the utilization of memory decreases to 27%, when more number of PSs are considered in embedding the process with same VN size of 2 VMs. On the other hand, it is found that the memory resource utilization increases from 27% to 54% with constant $8n_t$ number of PSs, when the VN size increases from 2 to 10 VMs, as presented in Figure 10(b). The similar trend is also observed in case of CPU utilization, as shown in Figure 10(a).

The impact of VN size and number of PSs parameters are also analyzed on the acceptance rate of the proposed DYVINE algorithm, as shown in Figure 12. The acceptance rate is 99%, when the VN size n_t is 2 VMs and the number of PSs is $2n_t = 4$. As more numbers of PS are considered in the embedding process, a VN is more likely to be accepted by the DYVINE algorithm. Acceptance rate decreases in two scenarios, when the VN size increases or less number of PSs are chosen in the DYVINE algorithm.

From the above simulation results, it is observed that increasing the number of PSs gives more efficient result in terms of resource utilization and acceptance rate. However, this may give inefficient result in terms of embedding time as presented in Figure 13. The number of VNs is kept constant at 500. The proposed DYVINE algorithm takes approximately $13ms$ to embed 500 VNs with VN size of 2 VMs, when the number of PSs is $8n_t = 16$. However, the embedding time can be reduced to approximately only $8ms$ by reducing the number of PSs to $2n_t = 4$. This improvement can be realized when the VN size increases from 2 to 10 VMs. The time taken to embed 500 VNs with $n_t = 10$ VMs is $32ms$, when the number of PSs is only $2n_t = 20$. This embedding time increases to $43ms$, when the number of PSs increases to $8n_t = 80$. From the above figure, it is observed that increase in the number of PSs, i.e., the length of set B as given in Line 10 of Algorithm 1 gives better result in terms of resource utilization and acceptance with the cost of higher embedding time.

Figure 14 shows the comparison of the proposed DYVINE algorithm with the optimal solution in terms of CPU, memory and network resource utilization. The comparison is done to demonstrate how close the proposed DYVINE algorithm to the optimal solution is. The optimal results are obtained from a small scale simulation environment, where the number of physical servers is 25 and the number VMs per VN ranges between 2 and 6. For each VN, the minimum link connectivity probability is set to be 0.7. The simulation results are obtained for a minimum of 10 VNs and a maximum of 100 VNs. For each VN, all possible embedding solutions are compared to find the optimal solution.

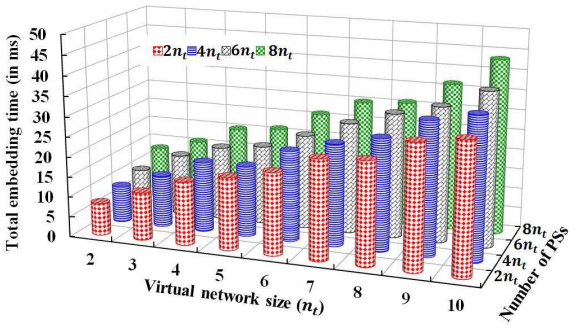
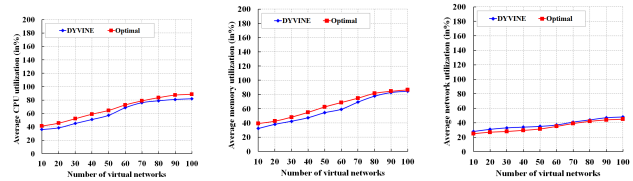


Fig. 13. Embedding time with variable VN size and number of PSs.



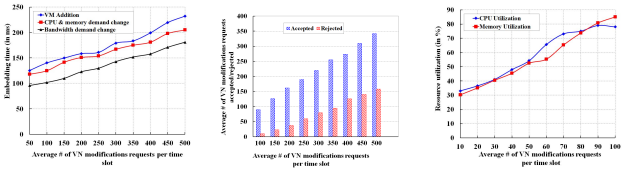
(a) CPU resource utilization. (b) Memory resource utilization. (c) Network resource utilization.

Fig. 14. Comparison of DYVINE with the optimal resource utilization. (a) CPU resource utilization. (b) Memory resource utilization. (c) Network resource utilization.

Figure 14(a) shows the comparison of CPU utilization of the optimal solution and the solution obtained from the DYVINE algorithm. With the given small scale simulation environment, the CPU utilization is 36%, when the number of VN is 10. The utilization increases to 82%, when the number of VNs increases to 100. On the other hand, the utilization increases from 42% to 89%, when the number of VNs increases from 10 to 100. The utilization is more in case of optimal solution as the VMs are placed in an optimal manner and more number of VNs are accepted for the resource allocation. Similar to the CPU utilization, Figure 14(b) shows the comparison of optimal solution and the proposed algorithm in terms of average memory utilization. The memory utilization increases from 40% to 87%, when the number of VNs increases from 10 to 100. On the other hand, the memory utilization is 32% in case of DYVINE, when the number of VNs is 10. However, the utilization value increases to 85%, when the number of VNs increases to 100.

The proposed algorithm is compared with the optimal solution in terms of network resource utilization as shown in Figure 14(c). In the proposed algorithm, multiple VMs from the same VN share same physical server. As a result, the allocated amount of network resource is less than the required amount of network resource. However, it is observed from the comparison results as shown in Figure 14(c) that network utilization can further be minimized. The network resource utilization increases from 28% to 48% and from 25% to 44.5%, when the number of VNs increases from 10 to 100 with minimum virtual link connectivity probability of 0.7 in case of DYVINE and optimal solution, respectively.

The main aspect of the proposed algorithm is its ability to handle the dynamic resource demand of the virtual networks. Figure 15 shows the relation of average number of VN modification requests per unit time with different parameters such



(a) Embedding time (in milliseconds) (b) Number of VN modification requests accepted or rejected. (c) Resource utilization.

Fig. 15. Effect of dynamic VN modification requests on other parameters. (a) Embedding time (in milliseconds) (b) Number of VN modification requests accepted or rejected. (c) Resource utilization.

as embedding time, number of requests accepted or rejected and resource utilization. In Figure 15(a), X-axis represents the average number of requests coming to the CSP to change the existing VNs. For this, the number of existing VNs is kept constant at 1000. In other words, 1000 number of VNs are currently running in the cloud platform. For those currently running VNs, CSP may expect a large number of requests to modify the VN structure or the resource configuration of the VMs and virtual links. The request can be divided into three categories as (a) request to add new VM, (b) request to modify the resource demand of the existing VMs, and (c) request to modify the network bandwidth demand of the virtual links.

Figure 15(a) shows the simulation results of embedding time of certain number of VN modification requests of the above mentioned three categories. The embedding time in Y-axis is calculated in milliseconds, whereas the average number of the requests per unit time ranges from 50 and 500. It is observed that the proposed algorithm requires approximately 125 milliseconds to process 50 number of VN modification requests in order to add new VM onto the existing VNs. Adding new VM onto the existing VN also includes the addition of new virtual link(s). As a result, the embedding time to add a new VM is expensive as compared to the processing of other types of VN modification requests. The time required to process the requests to update the VM resource demand and virtual link network resource demand is approximately 115 and 95 milliseconds, respectively. However, the embedding time increases to approximately 230 milliseconds, when the average number of requests to add new VM increases to 500. On the other hand, the embedding time increases to approximately 200 and 180 milliseconds, when the number of requests to update the resource demand of the VMs and virtual links increases to 500 each, respectively as shown in Figure 15(a).

Similarly, Figure 15(b) shows the number of requests to modify the VN being accepted or rejected. The X-axis indicates the average number of VN modification requests per unit time. The VN modification requests could be the request to add new VNs or modify the resource demand of VMs and the virtual links. Y-axis represents the average number of such requests being accepted or rejected. The average value in Y-axis is obtained by simulating the algorithm for multiple times. It is observed that an average of 90 number of VN modification requests from a total of 100 requests are accepted and 10 number of requests are rejected. However, when the average number of VN modification requests increases to 500,

an average of 340 number of different types of VN modification requests are accepted and are processed. The requests to modify the VN would be rejected only if no enough resource is found. The effect of VN modification request on resource utilization is also shown in Figure 15(c). This observation can be compared with the CPU and memory utilization as shown in Figure 14(a) and 14(b), respectively. It is observed that the resource utilization slightly decreases when the number of VN modification requests increases. The CPU utilization is 33.4%, when an average of 10 VN modification requests arrive per unit time, which is approximately 3% less in normal scenario as shown in Figure 14(a). Similarly, the memory utilization is 85%, less than the resource utilization under normal circumstances as shown in Figure 14(b), when the number of VN modification requests is 100 per unit time.

VI. Conclusions

The problem of embedding virtual network, whose structure and resource demand or configuration can change over time is studied in this work. An efficient DYnamic VIrtual Network Embedding (DYVINE) algorithm is proposed taking the dynamic nature of the virtual networks into account. In this work, dynamic nature refers to as the change in resource configuration or demand of the VMs and virtual links and changes in the network topology or the structure of the VNs. The VMs and virtual links are embedded based on the BestFit approach. Two fitness values LFV and GFV of the VMs and entire VNs, respectively are formulated to embed the VMs. Each VM mapping is followed by the mapping of the adjacent virtual links. The propagation delay parameter is considered while embedding the virtual links onto the physical paths. Furthermore, the proposed algorithm exploits the opportunity to embed the VMs in a more compact manner with the objective to minimize the fitness values. As a result, the resource utilization and acceptance rate are maximized. Besides, we have evaluated the proposed algorithm by comparing with the existing approaches. As the part of our future work, we would like to focus on extending the current work by taking other parameters into account, which can improve the QoS and maximize the revenue for the CSP. The proposed algorithm does not consider the requests generated from the edge computing platforms as such requests are associated with the parameters and constraints that are different from the associated parameters and constraints of the cloud computing. However, we will focus on extending the proposed DYVINE algorithm in future considering the computing environment, which can integrate both edge and cloud computing.

References

[1] H. Shen and L. Chen, "Distributed autonomous virtual resource management in datacenters using finite-Markov decision process," *IEEE/ACM Trans. Netw.*, vol. 25, no. 6, pp. 3836–3849, Dec. 2017.

[2] C. Beşiktaş, D. Gözüpek, A. Ulaş, and E. Lokman, "Secure virtual network embedding with flexible bandwidth-based revenue maximization," *Comput. Netw.*, vol. 121, pp. 89–99, Jul. 2017.

[3] I. Fajjari, N. Aitsaadi, B. Dab, and G. Pujolle, "Novel adaptive virtual network embedding algorithm for cloud's private backbone network," *Comput. Commun.*, vol. 84, pp. 12–24, Jun. 2016.

[4] W. Xia, P. Zhao, Y. Wen, and H. Xie, "A survey on data center networking (DCN): Infrastructure and operations," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 1, pp. 640–656, 1st Quart., 2016.

[5] N. Shahriar *et al.*, "Virtual network survivability through joint spare capacity allocation and embedding," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 502–518, Mar. 2018.

[6] P. K. Sahoo, C. K. Dehury, and B. Veeravalli, "LVRM: On the design of efficient link based virtual resource management algorithm for cloud platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 4, pp. 887–900, Apr. 2018.

[7] O. Soualah, N. Aitsaadi, and I. Fajjari, "A novel reactive survivable virtual network embedding scheme based on game theory," *IEEE Trans. Netw. Service Manage.*, vol. 14, no. 3, pp. 569–585, Sep. 2017.

[8] H. Cao, Y. Zhu, L. Yang, and G. Zheng, "A efficient mapping algorithm with novel node-ranking approach for embedding virtual networks," *IEEE Access*, vol. 5, pp. 22054–22066, 2017.

[9] H. Cao, L. Yang, and H. Zhu, "Novel node-ranking approach and multiple topology attributes-based embedding algorithm for single-domain virtual network embedding," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 108–120, Feb. 2018.

[10] Z. Zhang, S. Su, J. Zhang, K. Shuang, and P. Xu, "Energy aware virtual network embedding with dynamic demands: Online and offline," *Comput. Netw.*, vol. 93, pp. 448–459, Dec. 2015.

[11] G. Sun, H. Yu, V. Anand, and L. Li, "A cost efficient framework and algorithm for embedding dynamic virtual network requests," *Future Gener. Comput. Syst.*, vol. 29, no. 5, pp. 1265–1277, Jul. 2013.

[12] F. Yan, T. T. Lee, and W. Hu, "Congestion-aware embedding of heterogeneous bandwidth virtual data centers with hose model abstraction," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 806–819, Apr. 2017.

[13] T. M. Nam, N. H. Thanh, H. T. Hieu, N. T. Manh, N. Van Huynh, and H. D. Tuan, "Joint network embedding and server consolidation for energy–efficient dynamic data center virtualization," *Comput. Netw.*, vol. 125, pp. 76–89, Oct. 2017.

[14] T. Wang and M. Hamdi, "Presto: Towards efficient online virtual network embedding in virtualized cloud data centers," *Comput. Netw.*, vol. 106, pp. 196–208, Sep. 2016.

[15] N. Ogino, T. Kitahara, S. Arakawa, and M. Murata, "Virtual network embedding with multiple priority classes sharing substrate resources," *Comput. Netw.*, vol. 112, pp. 52–66, Jan. 2017.

[16] C. Yu, Q. Lian, D. Zhang, and C. Wu, "PAME: Evolutionary membrane computing for virtual network embedding," *J. Parallel Distrib. Computing*, vol. 111, pp. 136–151, 2018.

[17] C. K. Pyoung and S. J. Baek, "Joint load balancing and energy saving algorithm for virtual network embedding in infrastructure providers," *Comput. Commun.*, vol. 121, pp. 1–18, May 2018.

[18] Z. Xu, W. Liang, and Q. Xia, "Efficient embedding of virtual networks to distributed clouds via exploring periodic resource demands," *IEEE Trans. Cloud Comput.*, vol. 6, no. 3, pp. 694–707, Jul./Sep. 2018.

[19] L. Yu, H. Shen, Z. Cai, L. Liu, and C. Pu, "Towards bandwidth guarantee for virtual clusters under demand uncertainty in multi-tenant clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 2, pp. 450–465, Feb. 2018.

[20] Y. Liang and S. Zhang, "Embedding parallelizable virtual networks," *Comput. Commun.*, vol. 102, pp. 47–57, Apr. 2017.

[21] S. Haeri and L. Trajković, "Virtual network embedding via monte Carlo tree search," *IEEE Trans. Cybern.*, vol. 48, no. 2, pp. 510–521, Feb. 2018.

[22] X. Hesselbach, J. R. Amazonas, S. Villanueva, and J. F. Botero, "Coordinated node and link mapping VNE using a new paths algebra strategy," *J. Netw. Comput. Appl.*, vol. 69, pp. 14–26, Jul. 2016.

[23] A. Aral and T. Ovatman, "Network-aware embedding of virtual machine clusters onto federated cloud infrastructure," *J. Syst. Softw.*, vol. 120, pp. 89–104, Oct. 2016.

[24] A. Alsarhan, A. Itradat, A. Y. Al-Dubai, A. Y. Zomaya, and G. Min, "Adaptive resource allocation and provisioning in multi-service cloud environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 1, pp. 31–42, Jan. 2018.

[25] H. Yao, X. Chen, M. Li, P. Zhang, and L. Wang, "A novel reinforcement learning algorithm for virtual network embedding," *Neurocomputing*, vol. 284, pp. 1–9, Apr. 2018.

[26] L. Gong, H. Jiang, Y. Wang, and Z. Zhu, "Novel location-constrained virtual network embedding LC-VNE algorithms towards integrated node and link mapping," *IEEE/ACM Trans. Netw.*, vol. 24, no. 6, pp. 3648–3661, Dec. 2016.

[27] P. Zhang, H. Yao, and Y. Liu, "Virtual network embedding based on the degree and clustering coefficient information," *IEEE Access*, vol. 4, pp. 8572–8580, 2016.

[28] L. Yin, Z. Chen, L. Qiu, and Y. Wen, "Interference based virtual network embedding," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2016, pp. 1–6.

Chinmaya Kumar Dehury received the B.C.A. degree from Sambalpur University, India, in 2009, the M.C.A. degree from Biju Patnaik University, India, in 2013, and the Ph.D. degree in computer science and information engineering from Chang Gung University, Taiwan, in 2019. His research interests are in scheduling, resource management, and fault tolerance problems of cloud computing.

Prasan Kumar Sahoo (SM'16) received the B.Sc. degree (Hons.) in physics and the M.Sc. degree in mathematics from Utkal University, India, in 1987 and 1994, respectively, the M.Tech. degree in computer science from IIT Kharagpur, India, in 2000, the Ph.D. degree in mathematics from Utkal University in 2002, and the Ph.D. degree in computer science and information engineering from National Central University, Taiwan, in 2009. He has been an Adjunct Researcher with the Division of Colon and Rectal Surgery, Chang Gung Memorial Hospital, Taiwan, since 2018. He is currently a Professor with the Department of Computer Science and Information Engineering, Chang Gung University, Taiwan. His current research interests include artificial intelligence, big data analytic, cloud computing, and the Internet of Things (IoT). He is an Editorial Board Member of the *International Journal of Vehicle Information and Communication Systems* (IJVIC). He was the Program Committee Member of several IEEE and ACM conferences.