ReViNE: Reallocation of Virtual Network Embedding to Eliminate Substrate Bottlenecks

Shihabur Rahman Chowdhury*, Reaz Ahmed*, Nashid Shahriar*, Aimal Khan*, Raouf Boutaba*,
Jeebak Mitra†, and Liu Liu‡
*David R. Cheriton School of Computer Science, University of Waterloo
{sr2chowdhury | r5ahmed | nshahria | a273khan | rboutaba}@uwaterloo.ca
†Huawei Technologies Canada Research Center
jeebak.mitra@huawei.com
‡Huawei Technologies
liuliu1@huawei.com

***Abstract*—Perceived as a key enabling technology for the future Internet, Network Virtualization (NV) allows an Infrastructure Provider (InP) to better utilize their Substrate Network (SN) by provisioning multiple Virtual Networks (VNs) from different Service Providers (SPs). A key challenge in NV is to efficiently map the VN requests from SPs on an SN, known as the Virtual Network Embedding (VNE) problem. VNE algorithms are typically online in nature. A VN embedding can become suboptimal over time due to the arrival and departure of other VNs as well as due to changes in SN such as failures. One way to mitigate the impact of such dynamism is to periodically reallocate resources for the existing VNs. VNE reallocation can increase an InP's revenue by decreasing bandwidth consumption and by increasing the possibility of accepting future VNs. In this paper, we study Reallocation of Virtual Network Embedding (*ReViNE*) problem to minimize the number of over utilized substrate links and total bandwidth cost on the SN. We propose an Integer Linear Programming formulation for the optimal solution (*ReViNE*-OPT) and a simulated annealing based heuristic (*ReViNE*-FAST) to solve larger problem instances. Simulation results show that on average our proposed heuristic performs within ∼19% of the optimal solution. Moreover, *ReViNE*-FAST generates more than 2.5× better solutions compared to the state-of-the-art simulated annealing based heuristic for VNE reallocation.**

I. Introduction

Network Virtualization (NV) [1] enables Infrastructure Providers (InPs) to provision Virtual Networks (VNs) from multiple Service Providers (SPs) on their Substrate Network (SN). Such sharing opens new revenue streams for the InPs and allows them to better utilize the substrate resources. However, the benefits from NV come at the cost of additional resource management challenges for the InPs. A key resource management challenge in NV is to efficiently map VN requests from SPs on an SN, known as the Virtual Network Embedding (VNE) problem [2]. Typical objectives of VNE include maximizing the number of mapped VNs [3] and minimizing the resource provisioning cost on the SN [4], [5].

Most of the VNE algorithms are online, *i.e.*, little or no future information about VN arrival is known. In contrast, NV is a dynamic environment where VNs can arrive and depart over time and the SN can change due to failures. Given such dynamism, the VN embedding that was optimal at a certain network state may become suboptimal with the progression of time. As a consequence, SN resource utilization can become skewed thus creating network bottlenecks. Such bottlenecks can be hardly avoided when no or little information is known about the future VNs. They can also lead to rejection of future VN requests, hence, adversely affecting an InP's revenue.

One way to mitigate the impact of such dynamic behavior in NV is to periodically reallocate the resources assigned to VNs, *i.e.*, migrate already embedded virtual nodes (VNodes) and virtual links (VLinks) to different substrate nodes (SNodes) and substrate paths (SPaths), respectively, to optimize resource usage in the SN. Such VN reconfiguration can be performed offline during an off-peak period. In this paper, we study **R**eallocation of **Vi**rtual **N**etwork **E**mbedding (*ReViNE*) problem to improve SN resource usage. Particularly, our objective is to reduce utilization skew by minimizing the number of bottleneck substrate links (SLinks), *i.e.*, SLinks with utilization over an operator defined threshold (*e.g.*, 80%) as well as the total bandwidth consumption of the embedded VNs.

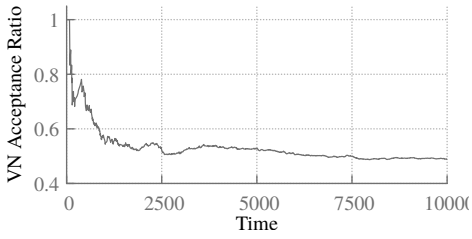
Minimizing the number of bottleneck SLinks may conflict with minimizing bandwidth consumption. This is because minimizing total bandwidth consumption requires embedding the VLinks on shorter SPaths. As a consequence, SLinks that are on shortest paths between most pair of SNodes have the higher probability of becoming bottlenecks. Minimizing the number of bottleneck SLinks thus requires embedding the VLinks on possibly longer SPaths, resulting in increased bandwidth consumption. Our formulation of *ReViNE* strikes a balance between these two conflicting objectives. Particularly, we have the following contributions in this paper:

- Integer Linear Programming (ILP) formulation for the optimal solution to *ReViNE* (*ReViNE*-OPT), which strikes a balance between minimizing the number of bottleneck SLinks and total bandwidth consumption on the SN.
- A heuristic algorithm (*ReViNE*-FAST) to tackle the computational complexity of *ReViNE*-OPT. Given the offline nature of *ReViNE*, we propose a simulated annealing [6] based heuristic.
- Performance comparison of *ReViNE*-FAST with that of the state-of-the-art simulated annealing based heuristic for VNE reallocation [7].

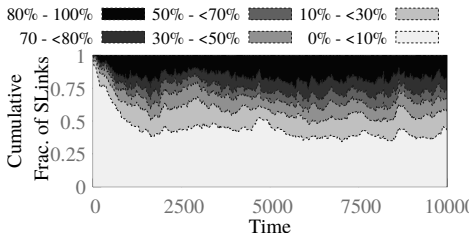
The rest of the paper is organized as follows. We start with motivating *ReViNE* through an experimental study in Section II. Then, we formally define *ReViNE* in Section III, followed by the ILP formulation for *ReViNE*-OPT in Section IV. A simulated annealing based heuristic to tackle the computational complexity of *ReViNE* is presented in Section V. Our evaluation of the proposed solutions are in Section VI, followed by a discussion on the related works from the literature in Section VII. Finally, we conclude with future research directions in Section VIII.

II. Motivation

We first perform an experimental study to motivate the requirement of reducing the number of bottleneck SLinks through reallocation. The experiment demonstrates that even with the optimal algorithm for VN embedding, substrate resource utilization can become skewed and can lead to rejection of VN requests. For this experiment, an in house discrete event simulator is used that simulates the arrival and departure of VNs on a large ISP network (AS-6461) from Rocketfuel ISP topology dataset [8]. Random VN topologies were generated with 50% probability of a link being present between two VNodes. We varied the VN size between 4 and 8. VN arrival rate and life time were random variable following a Poisson distribution with a mean of 10 arrivals per 100 time unit and exponential distribution with a mean of 1000 time units, respectively. These parameters were chosen in accordance with the values used in other research works from the literature [9], [10]. We modified the ILP formulation from [5] to discard the disjointedness constraint and optimally embed a VN request on an SN to minimize bandwidth provisioning cost.



(a) VN Acceptance Ratio



(b) SLink Utilization Distribution

Fig. 1. Results for Motivating Example

Fig. 1(a) shows VN acceptance ratio, *i.e.*, the fraction of accepted VNs over time. Fig. 1(b) shows the load distribution on SN. Fig. 1(a) shows that VN acceptance ratio drops close to 50% near time 2000 and remains almost the same throughout the simulation lifespan. One can think that such behavior is due to resource exhaustion in the SN, leading to rejection of VN requests. In reality, such behavior is due to the skewed load distribution on the SN as evident from Fig. 1(b). Fig. 1(b) shows that more than 40% of the SLinks have less than 10% utilization throughout the simulation timespan. At the same time, about 30% or more of the SLinks have a very high utilization, *i.e.*, more than 70%. Such skewness can be explained as follows. Although there is sufficient capacity available in part of the SN, subset of SLinks get more preference over the others by the embedding algorithm based on the objective. For example, an embedding algorithm that minimizes the cost of bandwidth allocation will prefer SLinks that are on shortest paths between SNodes. Thus these SLinks become saturated very quickly. If such SLinks also belong to a cut set of the SN, then the chances of a VN rejection is also increased by these SLinks becoming bottlenecks. As a result, VN requests are rejected since there are not sufficient paths in the SN that connect a VN's VNodes with the required capacity. A periodic reallocation of VNs can reduce bottlenecks by redistributing the load on the SN, leading to more accepted VNs.

III. System Model and Problem Statement

In this section, we first present a mathematical representation of the inputs, *i.e.*, the SN, the set of embedded VNs and location constraints in Section III-A. Then we formally define *ReViNE* in Section III-B.

A. System Model

1) Substrate Network: We represent the SN as an undirected graph, $G = (V, E)$, where V and E denote the set of SNodes and SLinks, respectively. The set of neighbors of an SNode $u \in V$ is denoted by $\mathcal{N}(u)$. We associate the following attributes with each SLink $(u, v) \in E$: (i) b_{uv} : total bandwidth capacity of the link (u, v), (ii) b^r_{uv} : residual bandwidth capacity of the link (u, v), and (iii) C_{uv} : cost of unit bandwidth on (u, v) for provisioning a VLink.

2) Set of Embedded VNs: We denote the set of embedded VNs on the SN with $\bar{G}$. Each VN $\bar{G}_i \in \bar{G}$ is represented as an undirected graph $\bar{G}_i = (\bar{V}_i, \bar{E}_i)$, where $\bar{V}_i$ and $\bar{E}_i$ are the set of VNodes and VLinks of $\bar{G}_i$, respectively. Each VLink $(\bar{u}, \bar{v}) \in \bar{E}_i$ has a bandwidth requirement $b_{i\bar{u}\bar{v}}$. We also have a set of location constraints for each VN $\bar{G}_i$, $L_i = \{L_i(\bar{u}) | L_i(\bar{u}) \subseteq V, \forall \bar{u} \in \bar{V}_i\}$, such that a VNode $\bar{u} \in \bar{V}_i$ can only be provisioned on an SNode $u \in L_i(\bar{u})$. We represent this location constraint with the following binary variable:

$$\ell_{i\bar{u}u} = \begin{cases} 1 & \textit{iff } \text{VNode } \bar{u} \in \bar{V}_i \text{ can be provisioned on } u \in V, \\ 0 & \text{otherwise.} \end{cases}$$

The input VNode and VLink embedding of a VN $\bar{G}_i \in \bar{G}$ on an SN G is represented with binary variables $y_{i\bar{u}u}$ and $x^{i\bar{u}\bar{v}}_{uv}$, respectively. These two variables are defined as follows:

$$y_{i\bar{u}u} = \begin{cases} 1 & \textit{iff } \text{VNode } \bar{u} \in \bar{V}_i \text{ is mapped to SNode } u \in V, \\ 0 & \text{otherwise.} \end{cases}$$

$$x^{i\bar{u}\bar{v}}_{uv} = \begin{cases} 1 & \textit{iff } \text{VLink } (\bar{u}, \bar{v}) \in \bar{E}_i \text{ is mapped to SLink}(u, v) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

B. ReViNE Problem Statement

Given an SN $G = (V, E)$, a set of embedded VNs $\bar{G}$, reallocate the current embedding of the VNs $\bar{G}_i \in \bar{G}$ by migrating VNodes and VLinks to different SNodes and SPaths in such a way that the re-optimized embedding achieves the following objectives:

a) Primary Objective: The number of bottleneck links in the SN is minimized. An SLink is considered bottleneck if its utilization is more than an input threshold $\theta\%$.

b) Secondary Objective: The total cost of allocating bandwidth for provisioning all the VNs are minimum. Mathematically, the secondary objective is to minimize the following cost function: $\sum_{\forall \bar{G}_i \in \bar{G}} \sum_{\forall (\bar{u},\bar{v}) \in \bar{E}_i} \sum_{\forall (u,v) \in P_{\bar{u}\bar{v}}} C_{uv} \times b_{i\bar{u}\bar{v}}$. Where, $P_{\bar{u}\bar{v}}$ is the SPath on which the VLink $(\bar{u}, \bar{v}) \in E_i$ is embedded after reallocation.

The reallocation is subject to the constraints that SLinks cannot be over-provisioned to accommodate the VLinks, and a VLink's demand cannot be routed along multiple SPaths.

IV. ILP Formulation

The primary objective of *ReViNE* is to minimize the number of bottleneck SLinks. However, solely minimizing the number of SLinks can lead to re-embedding VLinks on longer SPaths, resulting in high bandwidth usage in the SN. Therefore, we also minimize the total bandwidth cost for provisioning the VLinks as a secondary objective. As per the problem statement, we do not consider any node mapping cost and consider only link mapping cost. In what follows, we formulate the optimal solution of *ReViNE* as an ILP. We first introduce our decision variables in Section IV-A. Then, we present the constraints in Section IV-B followed by the objective function in Section IV-C.

A. Decision Variables

A VLink from a VN $\bar{G}_i \in \bar{G}$ should be mapped to a non-empty SPath. We introduce the following decision variable to indicate the mapping between a VLink $(\bar{u}, \bar{v}) \in \bar{E}_i$ and an SLink $(u, v) \in E$.

$$X_{uv}^{i\bar{u}\bar{v}} = \begin{cases} 1 & \text{if VLink } (\bar{u}, \bar{v}) \in \bar{E}_i \text{ is mapped to } (u, v) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

The following decision variable denotes VNode mapping:

$$Y_{i\bar{u}u} = \begin{cases} 1 & \text{if VNode } \bar{u} \in \bar{V}_i \text{ is mapped to } u \in V, \\ 0 & \text{otherwise.} \end{cases}$$

Note that $x_{uv}^{i\bar{u}\bar{v}}$ and $y_{i\bar{u}u}$ represent the given embedding (input), while $X_{uv}^{i\bar{u}\bar{v}}$ and $Y_{i\bar{u}u}$ represent the re-optimized embedding (output).

B. Constraints

1) Node Mapping Constraints: (1) ensures that VNodes are provisioned according to the location constraint. Then, (2) restricts a VNode to be provisioned on exactly one SNode. Finally, (3) ensures that multiple VNodes from the same VN are not mapped to the same SNode.

$$\forall \bar{G}_i \in \bar{G}, \forall \bar{u} \in \bar{V}_i, \forall u \in V : Y_{i\bar{u}u} \leq \ell_{i\bar{u}u} \quad (1)$$

$$\forall \bar{G}_i \in \bar{G}, \forall \bar{u} \in \bar{V}_i : \sum_{u \in V} Y_{i\bar{u}u} = 1 \quad (2)$$

$$\forall \bar{G}_i \in \bar{G}, \forall u \in V : \sum_{\bar{u} \in \bar{V}_i} Y_{i\bar{u}u} \leq 1 \quad (3)$$

Note that, VNode embedding follows from VLink embedding since there is no cost associated with VNode embedding.

2) Link Mapping Constraints: (4) ensures that every VLink is mapped to a non-empty set of SLinks and no VLink is left unmapped. Then, SLink resources over-provisioning is constrained by (5). Finally, (6) presents flow constraint that ensures VLinks are mapped to a continuous SPath.

$$\forall \bar{G}_i \in \bar{G}, \forall (\bar{u}, \bar{v}) \in \bar{E}_i : \sum_{\forall (u,v) \in E} X_{uv}^{i\bar{u}\bar{v}} \geq 1 \quad (4)$$

$$\forall \bar{G}_i \in \bar{G}, \forall (u, v) \in E : \sum_{\forall (\bar{u},\bar{v}) \in \bar{E}_i} X_{uv}^{i\bar{u}\bar{v}} \times b_{i\bar{u}\bar{v}} \leq b_{uv} \quad (5)$$

$$\forall \bar{G}_i \in \bar{G}, \forall \bar{u}, \bar{v} \in \bar{V}_i, \forall u \in V : \sum_{\forall v \in \mathcal{N}(u)} (X_{uv}^{i\bar{u}\bar{v}} - X_{vu}^{i\bar{u}\bar{v}}) = Y_{i\bar{u}u} - Y_{i\bar{v}u} \quad (6)$$

C. Objective Function

Our objective in *ReViNE* is to minimize the number of bottleneck SLinks while minimizing the cost of provisioning bandwidth on the SLinks. Therefore, we have a multi-objective optimization with two objectives. In the following, we mathematically formulate each of the objectives:

1) Bottleneck SLink cost: We consider an SLink to be a bottleneck link if its utilization is more than $\theta\%$. We use the following derived variable (dependent on θ) to denote if an SLink $(u, v) \in E$ is a bottleneck:

$$\Psi_{uv}(\theta) = \begin{cases} 1 & \textit{iff} \sum_{\forall \bar{G}_i \in \bar{G}} \sum_{(\bar{u},\bar{v}) \in E_i} X_{uv}^{i\bar{u}\bar{v}} \times b_{i\bar{u}\bar{v}} > \dfrac{\theta b_{uv}}{100} \\ 0 & \text{otherwise.} \end{cases} \quad (7)$$

Assume that C_{uv}^b is the cost of having a bottleneck SLink (u, v). The total cost incurred for having bottleneck SLinks is:

$$C_{bottleneck} = \sum_{\forall (u,v) \in E} \Psi_{uv} \times C_{uv}^b \quad (8)$$

2) Bandwidth cost: The total cost of allocating bandwidth on the SLinks to accommodate the VLinks can be represented by the following equation:

$$C_{bw} = \sum_{\forall \bar{G}_i \in \bar{G}} \sum_{\forall (\bar{u},\bar{v}) \in \bar{E}_i} \sum_{\forall (u,v) \in E} X_{uv}^{i\bar{u}\bar{v}} \times C_{uv} \times b_{i\bar{u}\bar{v}} \quad (9)$$

3) Final Objective Function: Our final objective function is to minimize the following weighted sum of (9) and (8):

$$\text{minimize } (\alpha C_{bottleneck} + (1 - \alpha) C_{bw}) \quad (10)$$

Where, $0 \leq \alpha \leq 1$ determines the relative weight of the cost components. For instance, setting α to 0.5 gives equal weight to both objectives, whereas, a value higher than 0.5 prioritizes minimizing the number of bottleneck SLinks over minimizing bandwidth consumption in the SN.

V. Simulated Annealing Heuristic

A. Motivation for a Heuristic

Theoretically, *ReViNE* is at least as hard as NP-hard multi-commodity unsplittable flow problem [11], [12]. However, unlike many other resource allocation problems in NV such as different variants of VNE [2], [13], *ReViNE* is an offline problem and is expected to be scheduled less frequently, *e.g.*, once every few hours or days. Given that *ReViNE* is not time critical, one can think of executing *ReViNE*-OPT every time a reallocation is triggered. However, running the optimal solution every time is not practical for the following reasons:

Expensive Computation: First, *ReViNE*-OPT is expensive in terms of both computation and memory resources. We explored the practicality of running *ReViNE*-OPT. We implemented *ReViNE*-OPT from Section IV using IBM ILOG CPLEX libraries and ran it on a machine with 80 CPU cores and 1TB of memory. For small problem instances (50 – 100 node SNs with less than 60 VNs), the CPLEX implementation took several hours in some cases and tens of gigabytes of memory. The solution could not even run for large problem instances (1000 node SNs) due to memory limitations.

Impractical Solutions: Second, although *ReViNE*-OPT gives us the optimal state of the network, the optimal final state might not be achievable in reality. The primary reason is that *ReViNE*-OPT gives us the final optimal embedding, whereas for a real life implementation we need a sequence of VNode and VLink migrations that will not disrupt the VN connectivity. The ILP formulation, *i.e.*, *ReViNE*-OPT does not take into account that a virtual resource has to be provisioned first before it can be migrated from the old embedding. Such dependency is hard to formulate in an ILP, hence, *ReViNE*-OPT might yield a state that is not achievable in practice.

These reasons have motivated us to design a heuristic that can find a near optimal and practical state from the current embedding in a reasonable time. A number of proposals in the literature re-optimize VN embedding using a greedy heuristic [14]–[16]. However, given the offline nature of the problem and the slower timescale at which it is solved, we have more computational cycles available to solve the problem at our disposal. Therefore, we propose to use simulated annealing [6] instead of designing a greedy one step solution. Simulated annealing is an effective meta-heuristic for finding locally optimal solutions from a large search space of integral variables.

B. Heuristic Design

Simulated annealing explores the neighborhood of an initial solution and keeps improving the solution. It also probabilistically accepts worse solutions from the search neighborhood. Typically, a temperature parameter (T) and energy function controls the probability of accepting a worse solution. Parameter T is set to a higher value during the initial iterations of the search, resulting in a higher probability of accepting a worse solution. A cooling schedule attenuates the temperature and eventually decreases the probability of accepting a worse solution towards the end of the search. By accepting worse solutions, simulated annealing tries to avoid being stuck at a local minima. We run multiple iterations of the neighborhood exploration procedure for each temperature to cover a wider search space. In order to design a simulated annealing heuristic, we need to define the following:

- A suitable data structure to represent solution from the solution space.
- A neighborhood generation function, which, generates a new solution from a given solution.
- An energy function, which determines the fitness of a solution. It regulates the probability of accepting or rejecting a solution during an iteration of simulated annealing.
- A cooling schedule, which determines how the temperature is attenuated during the search process.

In what follows, we describe how we have addressed the aforementioned issues and also provide pseudo-codes where necessary.

1) Solution Representation: We use the tuple $\langle G, \bar{G}, L, nmap, emap, \alpha, \theta, \xi \rangle$ to represent a solution in the solution space. $G, \bar{G}$, and L represent the SN, set of VNs and location constraints, respectively, as defined in Section III. $nmap_i \in nmap$ and $emap_i \in emap$ represents the VNode and VLink embedding of a VN $\bar{G}_i \in \bar{G}$, respectively. α is the weight from (10). θ is the utilization threshold from (7) that determines if an SLink is a bottleneck or not. Finally, ξ represents the set of bottleneck SLinks.

2) Neighborhood Generation: We randomly perform one of the following three operations on a solution to generate a neighborhood solution:

a) Bottleneck SLink Reconfiguration:: We randomly select a bottleneck SLink and try to reduce the selected SLink's utilization by reallocating VLinks embedded on it. Alg. 1 presents the pseudo-code for bottleneck SLink reconfiguration process. Alg. 1 first selects a VLink embedded on the selected SLink with maximum number of bottleneck SLinks on the VLink's embedded SPath (line 4). The rationale for this criteria is to reallocate a VLink that affects more bottleneck SLinks. Then we compute a new embedding SPath for the VLink while excluding the selected bottleneck SLink from path computation (line 7 – 8). Then, we reallocate the VLink and update the utilization of the SLinks on the old and new SPaths. This process is continued until the selected SLink is no longer a bottleneck or no VLink can be migrated.

b) VNode Migration:: We randomly choose a VNode from a randomly selected VN and re-embed that VNode on a different SNode. Pseudo-code for this process is described in Alg. 2. Alg. 2 goes through the selected VNode's location constraint set and determines the cost change for migrating to that SNode. The SNode that yields maximum cost reduction is then chosen for migration.

c) Virtual Link Migration:: We randomly choose a VLink from a randomly selected VN and re-embed the VLink. In order to re-embed the VLink, we compute an SPath between the SNodes corresponding to the node embedding of the endpoints of the VLink.

Note that in these algorithms we have used a procedure named MCP (Minimum Cost Path) to compute embedding SPaths. MCP computes a path in the SN between a pair of SNodes, which avoids a set of forbidden SLinks. MCP assigns weight to each SLink that is proportional to the already allocated bandwidth on that SLink. Such weight function allows MCP to find an SPath avoiding the already highly utilized SLinks.

Algorithm 1: Bottleneck Reconfiguration

1 **function** *ReconfigureBottleneck(solution)*
2 $(u, v) \leftarrow$ Random bottleneck SLink from $solution.\xi$
3 **while** (u, v) *is a bottleneck* **do**
4 $(m, n) \leftarrow$ VLink mapped to (u, v) with maximum number of bottleneck SLinks on its embedded SPath
5 $i \leftarrow k | \bar{G}_k \in solution.\bar{G}$ and $(m, n) \in \bar{E}_k$
6 $(u', v') \leftarrow (solution.nmap_i^m, solution.nmap_i^n)$
7 $forbidden \leftarrow \{(u, v)\}$
8 $P_{mn} \leftarrow$MCP$(solution.G, u', v', forbidden)$
9 **if** $P_{mn} = \phi$ **then break**
10 $solution.emap_i^{mn} \leftarrow P_{mn}$
11 Update utilization and residual bandwidth
12 **return** *solution*

Algorithm 2: VNode Migration

1 **function** *ReallocateVNode(solution)*
2 $i \leftarrow$ Index of a randomly selected VN
3 $m \leftarrow$ A random VNode from VN, $solutions.\bar{G}_i$
4 $\Delta_{max} \leftarrow 0, best_u \leftarrow NIL$
5 **for** $u \in solution.L_i(m)$ **do**
6 $\Delta \leftarrow$ Cost improvement for reallocating m to u
7 **if** $\Delta > \Delta_{max}$ **then** $\Delta_{max} \leftarrow \Delta$, $best_u \leftarrow u$
8 **if** $best_u \neq solution.nmap_i^m$ ***and*** $best_u \neq NIL$ **then**
9 **for** $\forall n \in \mathcal{N}(m)$ **do**
10 $P_{mn} \leftarrow$ MCP$(solution.G, best_u, solution.nmap_i^n, \phi)$
11 $solution.emap_i^{mn} \leftarrow P_{mn}$
12 Update utilization and residual bandwidth
13 $solution.nmap_i^m \leftarrow best_u$
14 **return** *solution*

3) Energy Function and Cooling Schedule: We use cost function (10) as the energy of a solution. During iteration k of a simulated annealing search, the probability of accepting a solution is a function of energy and temperature. We define this probability as follows:

$$\mathcal{P}(Energy, T_k) = \begin{cases} 1 & \text{if } \Delta_{Energy} < 0 \\ e^{-\Delta_{Energy}/T_k} & \text{otherwise.} \end{cases}$$

Where, T_k is the temperature at the k-th iteration and $\Delta_{Energy} = Energy(neighbor) - Energy(current)$. Here, $Energy(neighbor)$ is the energy of the neighbor solution generated by randomly applying one of the above three neighborhood generation techniques on the current solution with energy, $Energy(current)$. We adapt the following linear function from [7] as our cooling schedule and determine the temperature, T at iteration $k+1$: $T_{k+1} = \rho * T_k$, where $0 < \rho < 1$ is the cooling rate.

The combined algorithm is presented as pseudo-code in Alg. 3. Alg. 3 takes as input the initial solution, *i.e.*, initial set of VN embedding ($solution$), maximum number of iterations to perform (it_{max}), the number of iterations to perform per temperature value (it_{temp}), an initial temperature (T_0) and the cooling rate (ρ). During each iteration, a neighboring solution is generated by invoking GenerateNeighbor procedure (line 7). Based on the energy of the neighboring solution and the current temperature, this neighboring solution is retained or rejected (line 11 – 14). Finally, the best solution generated during all the iterations is returned.

C. Parallel Implementation of Heuristic

Solution space exploration can be further improved by spawning multiple simulated annealing searches from different starting points. We generate a number of neighbors from the initial solution using our neighborhood generation procedure GenerateNeighbor. These solutions are then used as a starting point for independent simulated annealing searches. Each of these independent searches can be performed on a separate thread without any synchronization. When the execution of all threads finishes, the one yielding the best solution among all other threads is chosen as the global best.

Algorithm 3: *ReViNE*-FAST

1 **function** *ReallocateVN(solution, it_{max}, it_{temp}, T_0, ρ)*
2 $iterations \leftarrow 0$, $T \leftarrow T_0$
3 $current \leftarrow best \leftarrow solution$
4 **while** $iterations < it_{max}$ **do**
5 **while** $iterations_t < it_{temp}$ **do**
6 $current\_energy \leftarrow Cost(current)$
7 $next \leftarrow$ GenerateNeighbor$(current)$
8 $next\_energy \leftarrow Cost(next)$
9 $\Delta_{energy} = next\_energy - current\_energy$
10 $p \leftarrow$ rand(0, 1)
11 **if** $\Delta_{energy} < 0$ **then** $current \leftarrow next$
12 **else if** $p < e^{-\Delta_{energy}/T}$ **then**
13 $current \leftarrow next$
14 **if** $Cost(best) > Cost(current)$ **then**
15 $best \leftarrow current$
16 Increment $iterations_t$
17 $T = \rho * T$, Increment $iterations$
18 **return** $best$

VI. Evaluation

In this section, we first describe the simulation setup in Section VI-A followed by a description of the evaluation metrics in Section VI-B. Then, we present the findings focusing on the following aspects: (i) performance comparison of *ReViNE*-FAST with *ReViNE*-OPT in Section VI-C, (ii) the impact of reallocation on VN acceptance ratio in Section VI-D, (iii) performance comparison of *ReViNE*-FAST with the state-of-the-art simulated annealing based heuristic for VNE reallocation from the literature [7] (named as SA-realloc in this section) in Section VI-E, and (iv) the convergence behavior of *ReViNE*-FAST compared to [7] for larger problem instances in Section VI-F.

A. Simulation Setup

1) Testbed: We have implemented *ReViNE*-OPT using IBM ILOG CPLEX 12.5 C++ libraries. We have also implemented *ReViNE*-FAST in C++. Our implementation of simulated annealing leverages multiple CPU cores and performs multiple searches in parallel. The CPLEX implementation was run on a machine with 8×10 core Intel Xeon E7-8879 CPU and 1TB of memory. The heuristic was run on a 4×8 core Intel E5-4640 CPU and 512GB of memory. If not specified, *ReViNE*-FAST performed 32 parallel searches, each search running on a separate thread pinned on a dedicated CPU. We set the initial temperature to 0.95 and the cooling rate to 0.99. We set the unit costs C^b_{uv} and C_{uv} from (8) and (9), respectively, in a way such that the cost components in (10) are comparable. The weight factor α was set to prioritize bottleneck link cost over bandwidth cost for reducing utilization skew in the SN.

2) Small Test Cases: We have used two randomly generated SNs for the small test cases. The first SN contains 50 nodes and 100 links, and the second SN contains 100 nodes and 187 links. SLink capacity of these SNs was set to 40000 units. We used randomly generated graphs with different sizes and bandwidth requirements for the VNs. We varied the mean SLink utilization between 60% to 80% by varying the per VLink bandwidth requirement and the number of VNs. The number of VNs was varied between 20 and 75 depending on the SN. We set a random value between 70% and 90% as the utilization threshold for determining bottleneck SLinks.

3) Large Test Cases: For the larger test cases, much like the smaller test cases, the SN and VNs are randomly generated. The SN consists of 1000 nodes and 2000 links. In each test case, there are about 300 VNs, each with 7 VNodes on average. We set the utilization threshold for determining bottleneck SLinks at a random value between 70% and 80%. The number of bottleneck SLinks in these cases have been varied from 25% to 65% of the number of SLinks. The CPLEX solver was unable to solve the large test cases with the available compute resources, hence, the large cases were tested with *ReViNE*-FAST and SA-Realloc only.

B. Evaluation Metrics

a) Cost Ratio: Cost ratio is the ratio of costs obtained by two different solution approaches, *e.g.*, between *ReViNE*-FAST and *ReViNE*-OPT.

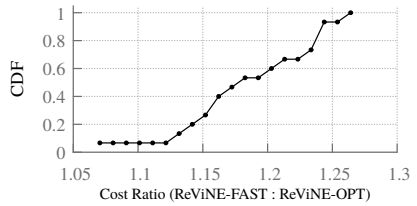
b) Reduction Ratio: Reduction ratio is the percentage of reduction of a given metric (*e.g.*, cost, number of bottleneck SLinks) by an algorithm compared to the initial value of the metric before optimization.

c) Performance Gap: Performance gap is the relative performance difference between a heuristic and the optimal solution. We compute performance gap for a metric $\mathcal{M}$ (*e.g.*, cost or bottleneck SLink reduction) using the following:

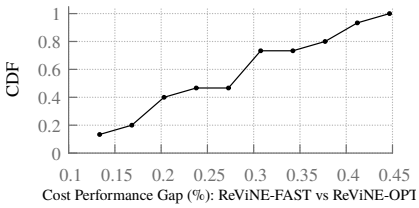
$$\frac{(\mathcal{M}_{previous} - \mathcal{M}_{optimal}) - (\mathcal{M}_{previous} - \mathcal{M}_{heuristic})}{\mathcal{M}_{previous} - \mathcal{M}_{optimal}}$$

In the above equation, $\mathcal{M}_{previous}$, $\mathcal{M}_{optimal}$, and $\mathcal{M}_{heuristic}$ represent the value of a metric before optimization, the optimal value and the value obtained using a heuristic, respectively. Performance gap shows the relative gap between the improvement obtained by a heuristic compared to the optimal solution. Cost and reduction ratio fail to capture the essence of relative performance improvement by the heuristic and optimal solution, hence, we also use performance gap metric to compare the performance of heuristic with that of the optimal.

C. Comparison between ReViNE-OPT and ReViNE-FAST



(a) CDF of cost ratio



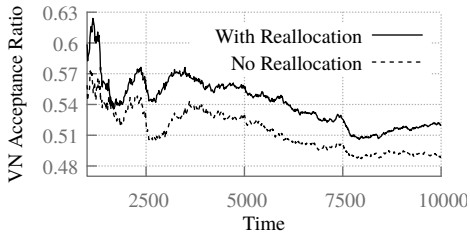
(b) CDF of Cost Performance Gap

Fig. 2. Comparison between *ReViNE*-OPT and *ReViNE*-FAST

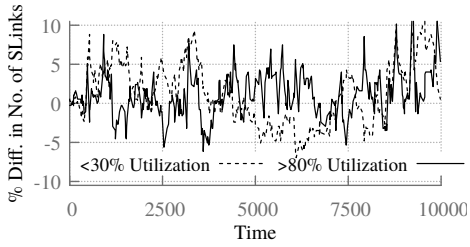
We ran both *ReViNE*-FAST and *ReViNE*-OPT on the small test cases to compare *ReViNE*-FAST's performance with *ReViNE*-OPT. In our simulations, both *ReViNE*-FAST and *ReViNE*-OPT were able to reduce the number of bottleneck SLinks down to zero. Therefore, the cost function, after reallocation in this case, directly represents the bandwidth cost of the solutions. For small scale scenario, the ratio of costs obtained by the solutions gives the extent of extra bandwidth used by *ReViNE*-FAST. It is worth mentioning that even with smaller problem instances, *ReViNE*-OPT took several hours to finish. On the other hand, we performed 32 parallel instances of simulated annealing, each performing 60,000 iterations in less than 20 seconds.

We plot the CDF of *ReViNE*-FAST to *ReViNE*-OPT's cost ratio in Fig. 2(a). In Fig. 2(a), the 90th percentile of cost ratio is within ∼1.24, *i.e.*, in 90% cases the heuristic uses up to ∼24% extra bandwidth compared to the optimal solution. On average, we found *ReViNE*-FAST to allocate ∼19% extra bandwidth compared to *ReViNE*-OPT. To demonstrate the performance of *ReViNE*-FAST more clearly, we also plot the CDF of performance gap for cost metric between *ReViNE*-FAST and *ReViNE*-OPT in Fig. 2(b). Performance gap demonstrates the relative performance improvement obtained by the discussed approaches. We observe that the relative performance gap between *ReViNE*-FAST and *ReViNE*-OPT is within ∼0.45%, with an average of ∼0.3% over all test cases. The significance of this metric is that the total cost reduction by the heuristic is within ∼0.3% of the total cost reduction obtained by the optimal solution on average for these test cases.

D. Impact of Reallocation



(a) Impact on VN Acceptance Ratio



(b) Impact on Load Distribution

Fig. 3. Impact of VN Reallocation

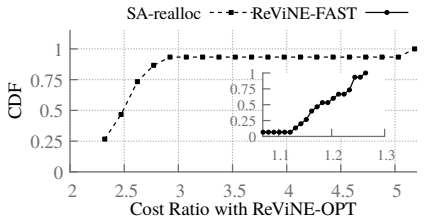
In this section, we present quantitative results showing the impact of reallocation on VN acceptance ratio and load distribution over SN. For this study, we repeat the same experiment from Section II with the exception of executing *ReViNE*-FAST every 450 time units. Such setting triggers about 24 reallocations during the simulation runtime, simulating a scenario with hourly reallocation. We performed a total of 60,000 iterations of simulated annealing search. Utilization threshold for determining bottleneck links was set to 75%.

Fig. 3(a) shows the impact of reallocation on VN acceptance ratio. In order to clearly present the results, we excluded the data from warm up period, *i.e.*, first 1000 time units in this case. The advantages of reallocation is clearly evident from Fig. 3(a). The results show about 5% increase in VN acceptance ratio. It is worth mentioning that this result represents a lower bound of the improvement since VNs are being embedded using the optimal algorithm. In reality, VNs are embedded using heuristic [2] and we foresee more improvement in acceptance ratio for those cases.

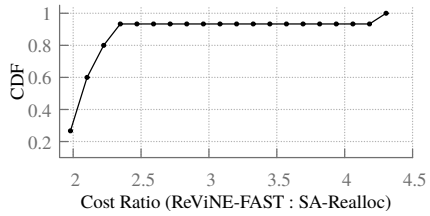
In Fig. 3(b), we present the difference between percentage of SLinks having different utilizations with and without reallocation. To show the results more clearly, we present only two classes of utilization: lightly loaded links, *i.e.*, links with less than 30% utilization, and highly loaded links, *i.e.*, links having more than 80% utilization. A positive value in difference indicates more of that type of link present without reallocation. We observe a larger area above the positive line for the highly loaded links in Fig. 3(b), indicating an overall reduction in bottleneck SLinks. We also observe a larger area over the positive line for the lightly loaded SLinks, indicating higher utilization of the network. Therefore, the overall network utilization has been increased while bottlenecks have been decreased by performing periodic reallocations.

E. Comparison with Related Work [7]

We have implemented the state-of-the-art simulated annealing heuristic for VNE reallocation from [7] and compare SA-realloc's performance with our solutions. SA-realloc generates neighboring solutions by randomly migrating a VNode to the first feasible SNode. However, SA-realloc does not provide any comparison with the optimal solution. We compare SA-realloc with both *ReViNE*-OPT and *ReViNE*-FAST and present the results in Fig. 4.



(a) Cost Ratio Comparison



(b) Cost Ratio of SA-realloc [7] to *ReViNE*-FAST

Fig. 4. Performance comparison with [7]

SA-realloc was also able to reduce the number of bottleneck SLinks down to zero for the small test cases. However, as we can see from Fig. 4(a), solutions obtained using [7] use 2.8× extra bandwidth compared to *ReViNE*-OPT on average. In contrast, *ReViNE*-FAST uses only ∼19% extra bandwidth on average compared to *ReViNE*-OPT. We perform a head-to-head comparison of [7] and *ReViNE*-FAST in Fig. 4(b) by plotting the CDF of cost ratio between SA-realloc and *ReViNE*-FAST. The results show that *ReViNE*-FAST always performs better than SA-realloc and uses 3.15× less bandwidth on average.

F. Large Scale Results

As stated earlier, *ReViNE*-OPT was not able to solve the larger test cases. Therefore, we only compare *ReViNE*-FAST with SA-realloc from [7] using the large test cases. For this scenario, *ReViNE*-FAST spawned 4 parallel searches in 4 separate threads pinned on dedicated CPUs, while SA-realloc was run with a single thread as per the description in [7]. We first recorded the cost function value after terminating both the heuristics after 5 minutes of execution. We also compare the convergence behavior of both heuristics by running them for extended period of time, *i.e.*, more than 5 minutes.

Fig. 5 shows the results for 5 minute execution duration. We plot the CDF of SA-realloc to *ReViNE*-FAST's cost ratio in Fig. 5. Even for the large test cases, *ReViNE*-FAST always performs better than SA-realloc. Solutions generated

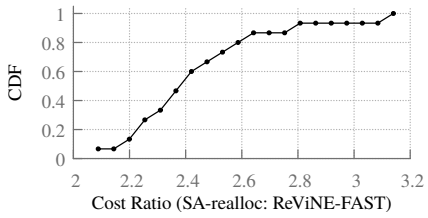


Fig. 5. Large inputs run for $\leq$ 5 minute

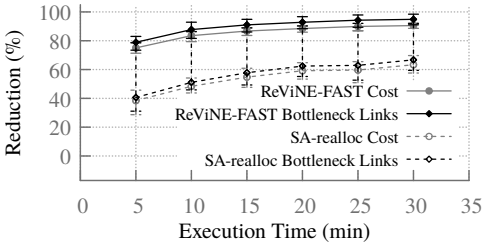


Fig. 6. Convergence behavior comparison

by *ReViNE*-FAST have $\sim$2.5$\times$ lesser value of cost function on average compared to those obtained by SA-realloc.

Results for running both *ReViNE*-FAST and SA-realloc for longer duration ranging from 5–30 minutes in 5 minute increments are presented in Fig. 6. Fig. 6 gives us some insight into the quality of results obtained during the first 5 minute of execution compared to longer runs. We report the mean percentage reduction in cost function and the number of bottleneck SLinks against increasing execution time in Fig. 6. Our first observation is that *ReViNE*-FAST performs significantly better compared to SA-realloc for similar execution time. For instance, *ReViNE*-FAST reduces both the cost function and the number of bottleneck SLinks by $\sim$80% within the first 5 minutes, compared to about $\sim$40% reduction by SA-realloc. Even after 30 minutes of execution, *ReViNE*-FAST maintains similar performance gap with that of SA-realloc. Finally, the improvement obtained within the first 5 minutes of execution is more significant compared to the improvement obtained when *ReViNE*-FAST runs longer. Although the same is true for SA-realloc, however, the solution quality obtained by *ReViNE*-FAST in first 5 minutes is closer to the optimal.

VII. Related Works

Different variants of VNE has received a significant attention from the research community over the past years [2]. However, VNE algorithms from these works are mostly online and can cause substrate resource fragmentation and skewed utilization over time. This motivated research proposals to reallocate VNE to improve SN performance. In this section, we discuss the state-of-the-art in VNE reallocation and contrast our approach with them.

Some of the earlier work in VNE also proposed to perform periodic migration of virtual resources to increase VN acceptance ratio [17] or remove bottlenecks from the SN [18]. [17] proposes to migrate both VNode and VLinks, whereas, [18] proposes to migrate only VLinks. These works assume that VLinks can be embedded on multiple SPaths, which is not valid in our case. Moreover, [17] and [18] do not provide optimal solutions to the problem and the performance gap of their heuristics with optimal solution is unknown.

A few research works have proposed to perform reallocation of VNs when a new VN request cannot be embedded on the SN [14], [19], [20]. [19] reallocates the VNodes and VLinks to remove bottlenecks in the SN that lead to rejection of a VN request. On the other hand, [14] migrates star-subgraphs of VNs to make room for the rejected VN in the SN. [20] addresses similar problem as [19] and [14] with the addition of minimizing VNode and VLink migration to reduce service disruption. These approaches are reactive, *i.e.*, they trigger a reallocation at certain events such as failure to embed a VN, and propose one shot heuristics for VNE reallocation. In contrast, we propose to use a local search meta-heuristic to explore more of the solution space. Our solution can be used both pro-actively or reactively. A few other works have addressed VNE reallocation problem to better balance the load across the SN and also reduce SLink utilization [15], [21]. However, they do not provide both optimal and heuristic solutions that strike a balance between minimizing the number of bottleneck SLinks and the total bandwidth cost.

Finally, some research works have proposed to use meta-heuristics for VNE reallocation [7], [22]. Unlike single step algorithms, meta-heuristic algorithms explore more of the solution space through systematic search methods. [7] proposes to use simulated annealing for VNE reallocation. It generates the neighboring solution by randomly migrating a VNode from a randomly selected VN. In contrast, we perform multiple operations to generate a neighboring solution. Moreover, we parallelized our simulated annealing search, which is not done in [7]. More recently, Yuan *et al.*, [22] proposed to use particle swarm optimization (PSO) for VNE reallocation to minimize peak SNode and SLink utilization. Unlike simulated annealing, PSO is a population based meta-heuristic, *i.e.*, at a given time the search frontier is expanded from a population of solutions. It would be interesting to quantitatively compare our solution with that of [22] and we leave it as a future work.

VIII. Conclusion and Future Work

In this paper, we have presented **Re**allocation of **Vi**rtual **N**etwork **E**mbedding (*ReViNE*), which strikes a balance between minimizing the number of bottleneck SLinks and the total bandwidth cost. We have proposed both optimal and heuristic solutions to *ReViNE*. Simulation results show that our proposed heuristic performs within $\sim$19% of the optimal solution and generates more than 2.5$\times$ better solutions compared to the state-of-the-art simulated annealing based VNE reallocation. In the future, we plan to extend and apply our work in the context of Dense Wavelength Division Multiplexed (DWDM) optical networks. Fragmentation and skewed utilization can affect DWDM networks more than IP networks since the wavelengths have to be assigned in discrete fashion and wavelength continuity constraints have to be maintained for VLink provisioning. Another interesting extension of this work would be to explore population based meta-heuristics such as Artificial Bee Colony optimization [23] for solving *ReViNE*.

REFERENCES

[1] N. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," *Computer Networks*, vol. 54, no. 5, pp. 862–876, 2010.

[2] A. Fischer, J. F. Botero, M. T. Beck, H. De Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.

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

[4] A. Razzaq and M. S. Rathore, "An approach towards resource efficient virtual network embedding," in *Evolving Internet (INTERNET), 2010 Second International Conference on*. IEEE, 2010, pp. 68–73.

[5] S. R. Chowdhury, R. Ahmed, M. M. A. Khan, N. Shahriar, R. Boutaba, J. Mitra, and F. Zeng, "Protecting virtual networks with drone," in *Proc. of IEEE/IFIP NOMS*, 2016.

[6] E. Aarts and J. Korst, *Simulated annealing and boltzmann machines*. New York, NY; John Wiley and Sons Inc., Jan 1988.

[7] S. B. Masti and S. V. Raghavan, "Simulated annealing algorithm for virtual network reconfiguration," in *Next Generation Internet (NGI), 2012 8th EURO-NGI Conference on*. IEEE, 2012, pp. 95–102.

[8] N. Spring, R. Mahajan, and D. Wetherall, "Measuring isp topologies with rocketfuel," *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4, pp. 133–145, 2002.

[9] M. Chowdhury, M. R. Rahman, and R. Boutaba, "Vineyard: Virtual network embedding algorithms with coordinated node and link mapping," *IEEE/ACM Transactions on Networking (TON)*, vol. 20, no. 1, pp. 206–219, 2012.

[10] M. R. Rahman and R. Boutaba, "Svne: Survivable virtual network embedding algorithms for network virtualization," *IEEE Transactions on Network and Service Management*, vol. 10, no. 2, pp. 105–118, 2013.

[11] Y. Dinitz, N. Garg, and M. X. Goemans, "On the single-source unsplittable flow problem," *Combinatorica*, vol. 19, no. 1, pp. 17–41, 1999.

[12] C. Chekuri, S. Khanna, and F. B. Shepherd, "The all-or-nothing multicommodity flow problem," in *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*. ACM, 2004, pp. 156–165.

[13] S. Herker, A. Khan, and X. An, "Survey on survivable virtual network embedding problem and solutions," in *International Conference on Networking and Services, ICNS*, 2013.

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

[15] B. Wanis, N. Samaan, and A. Karmouch, "Substrate network house cleaning via live virtual network migration," in *Communications (ICC), 2013 IEEE International Conference on*. IEEE, 2013, pp. 2256–2261.

[16] P. N. Tran and A. Timm-Giel, "Reconfiguration of virtual network mapping considering service disruption," in *Communications (ICC), 2013 IEEE International Conference on*. IEEE, 2013, pp. 3487–3492.

[17] Y. Zhu and M. H. Ammar, "Algorithms for assigning substrate network resources to virtual network components." in *INFOCOM*, vol. 12, 2006.

[18] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: substrate support for path splitting and migration," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 17–29, 2008.

[19] N. F. Butt, M. Chowdhury, and R. Boutaba, "Topology-awareness and reoptimization mechanism for virtual network embedding," in *International Conference on Research in Networking*. Springer, 2010, pp. 27–39.

[20] P. N. Tran, L. Casucci, and A. Timm-Giel, "Optimal mapping of virtual networks considering reactive reconfiguration," in *Cloud Networking (CLOUDNET), 2012 IEEE 1st International Conference on*. IEEE, 2012, pp. 35–40.

[21] R. Mijumbi, J. Serrat, J. Rubio-Loyola, N. Bouten, F. De Turck, and S. Latré, "Dynamic resource management in sdn-based virtualized networks," in *10th International Conference on Network and Service Management (CNSM) and Workshop*. IEEE, 2014, pp. 412–417.

[22] Y. Yuan, C. Wang, C. Wang, S. Zhu, and S. Zhao, "Discrete particle swarm optimization algorithm for virtual network reconfiguration," in *International Conference in Swarm Intelligence*. Springer, 2013, pp. 250–257.

[23] D. Karaboga and B. Basturk, "A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm," *Journal of global optimization*, vol. 39, no. 3, pp. 459–471, 2007.