Virtual Network Reconfiguration with Load Balancing and Migration Cost Considerations

Lingnan Gao[a], George N. Rouskas[a,b]
[a]Department of Computer Science, North Carolina State University, Raleigh, NC, USA
[b]Department of Computer Science, King Abdulaziz University, Jeddah, Saudi Arabia

***Abstract*—Efficient allocation of resources is an essential yet challenging problem in a virtual network environment, especially in an online setting whereby virtual network requests may arrive, depart, or be modified in real time. Virtual network reconfiguration may help to improve network performance by remapping a subset of virtual nodes or links to better align the allocation of resources to current network conditions. In this paper, we develop a virtual network reconfiguration scheme that aims to balance the load on the substrate network by dynamically reconfiguring the embedding of both virtual nodes and links. Our solution consists of decomposing the problem into two subproblems: i) virtual node selection, for which we present a linear programming-based fully polynomial time approximation scheme to select the virtual nodes to be migrated, and ii) virtual node remapping, for which we make use of random walk on a Markov chain to select new substrate nodes for the migrated virtual nodes.**

I. Introduction

Network virtualization [1] has drawn significant attention from both the academic and industry communities due to its potential to overcome what is referred to as Internet ossification [2], a situation where necessary or innovative transformations to the network architecture are difficult or impossible to achieve. In a network virtualization environment, the Internet Service Provider (ISP) is logically decoupled into two entities: the Service Provider (SP) and the Infrastructure Provider (InP). The InP is responsible for the operation and maintenance of the physical infrastructure, while the SP aggregates resources from InPs to create virtual networks (VNs) and provide end-to-end services to users [1]. Virtualization allows heterogeneous networks to co-exist on a shared infrastructure, and hence, it is widely considered as the most promising vehicle for research, experimentation, and deployment of innovative solutions for the next generation Internet.

In a network virtualization scenario, services are offered in the form of VNs, each VN consisting of a number of virtual nodes connected by virtual links. Therefore, for a VN to become fully functional, it has to be embedded onto the substrate network (SN) such that the requirements for the virtual nodes and links (e.g., CPU capacity and bandwidth, respectively) are satisfied. Efficient embedding of the VNs onto SNs gives rise to the virtual network embedding (VNE) problem [3], a challenging problem that remains NP-hard even when all the VN requests are known in advance.

In reality, the resource allocation problem in this context is online in nature: the lifetime of the VN requests may be finite and arbitrary; the arrival pattern of VN requests may be unpredictable; and the resource demands for existing VNs may change whenever users decide to scale up or down their requirements. If these dynamics are not taken into account and resource allocation is viewed as a purely static problem, the whole infrastructure may drift into an inefficient configuration that results in degraded performance for existing VNs and a higher rejection rate for subsequent VN requests [4].

One of the challenges in tackling online VNE problems has to do with modeling the dynamic behavior of VN requests, including the arrival process, the lifetime distribution, and the distribution of the time to the next scale up/down of resource requirements. An alternative approach is to reconfigure the VN embedding on a periodic basis by updating the mapping of virtual nodes and links to infrastructure resources in response to changes in VN requests. This is referred to as the virtual network reconfiguration (VNR) problem [11], and its objective is to improve resource utilization (for providers) and enhance performance (for users). Similar to the VNE problem, VNR is also concerned with the mapping of virtual nodes and links, but it also takes the existing configuration into account.

In our previous work [16], we presented a scheme to partition the virtual network requests and map them into different domains so that the traffic between different domains is minimized. In this work, we focus on algorithm design for the VNR problem so as to improve network performance by balancing the offered load across the substrate nodes and links. We first present a link-arc based mixed integer programming (MIP) formulation of the reconfiguration problem. The objective of the formulation is to minimize the maximum utilization of substrate nodes and links while bounding the number of virtual nodes that have to be migrated. This problem is NP-hard, since it reduces to the VNE problem if there is no bound on the number of virtual nodes to be migrated. Therefore, we decompose the problem into two phases, namely, virtual node selection and virtual node remapping. The first phase selects the virtual nodes to be migrated, while the second selects the new substrate nodes for the migrated virtual nodes. For the first phase, we solve the linear programming (LP) problem derived from the MIP. Specifically, we use a path-based formulation that achieves a near-optimal solution to the LP problem without intensive computation overhead. For the second phase, we use a Markov-chain based approach to filter candidate substrate nodes and solve a MIP problem to select the substrate node and complete link reconfiguration.

In Section II, we review previous work in this topic, and

978-1-5386-4128-6/18/$31.00 ©2018 IEEE

in Section III, we formally define the VNR problem and introduce the augmented graph model we use in developing the formulation. We present the algorithm we have developed for the VNR problem in Section IV, and we evaluate its performance in Section V. We conclude the paper in Section VI.

II. Related Work

The VNE problem has been studied extensively [3]. Although much research effort has been directed towards finding an efficient way of allocating substrate resources to virtual network requests, most studies consider the static problem with a focus on maximizing InP revenue or load balancing. To the best of our knowledge, only a few works consider a dynamic version of the problem where the objective is to reconfigure the VNs that are already mapped on the SNs; we review these works in the remainder of this section.

A method to reconfigure the VN requests to maximize the InP revenue has been proposed in [5]. In this study, emphasis is on the embedding of virtual links under the assumption that the traffic of a virtual link may be split and carried over several substrate paths. Consequently, the authors propose a reconfiguration scheme that dynamically adjusts the splitting ratio of a virtual link or sets up a new path in the substrate network. Although virtual link splitting and migration improves the acceptance ratio for VN requests, migration of virtual nodes is not considered in this work.

A proactive reconfiguration algorithm, referred to as global marking algorithm, was proposed in [4]. In the reconfiguration phase, the workload of the substrate nodes and links is examined. If a given substrate node or link is overloaded, then all virtual nodes and virtual links mapped on top of that particular substrate node or link, respectively, are marked for remapping. During the remapping phase, marked virtual nodes and links are re-assigned to the SN. This approach may incur unnecessary reconfigurations: often, only a fraction of the virtual nodes on a stressed substrate node need to be migrated, whereas this method tends to reconfigure all of them. Also, for congested substrate links, solely reconfiguring the mapping of the virtual links without migrating the virtual nodes causing the overload may not fully alleviate the congestion.

Another reconfiguration scheme to maximize InP revenue was proposed in [13]. This approach takes into account the migration overhead so as to limit the service disruption caused by reconfiguring the virtual nodes. Reconfiguration is triggered whenever a VN request is blocked, and follows the solution of a MIP problem. Since this method relies on solving the MIP problem, it may not scale to large problem sizes or may not be appropriate whenever a low reconfiguration delay is required.

Based on the observation in [11] that VN request rejection is mainly caused by bandwidth shortage, a reactive VN reconfiguration algorithm was presented in [12]. The algorithm aims at improving the VN request acceptance rate while limiting the number of virtual nodes to be migrated, so as to minimize the overall reconfiguration cost. Virtual nodes are selected for migration based on the number of congested links along which they route their traffic. Selected virtual nodes are iteratively reconfigured until either the incoming VN request is successfully embedded or the number of virtual nodes reconfigured exceeds the given threshold. This work solely considers congested links and does not account for overloaded substrate nodes. Also, it does not take into consideration the magnitude of traffic demands originating/terminating at the virtual nodes. This may result in a situation whereby virtual nodes with little traffic (and hence, small impact on the SN) are selected for migration just because they happen to use congested paths. As an extension to the work in [11], a VN reconfiguration strategy was developed in [12] to handle the scale-up/down of VN requirements requested by users. This study used a genetic algorithm to minimize the combined cost of embedding and reconfiguration.

A scheme to reconfigure the VN within an evolving substrate network was proposed in [6]. In this work, reconfiguration is triggered in response to changes in the underlying SN, i.e., whenever substrate nodes and/or links are added or removed. The objective is to reconfigure the VN such that delay constraints are preserved while migration overhead is minimized. To this end, a heuristic algorithm was proposed to relocate virtual nodes that are affected by the changes or violate the delay constraints.

III. Network Model and Problem Formulation

A. Virtual Networks and Substrate Networks

We model the SN as a weighted, undirected graph $G^s = (V^s, E^s)$. The vertices of G^s stand for the substrate nodes, while the edges of G^s represent the substrate links. Each node A and edge (A,B) on G^s are weighted, and we let Cap_A denote the resource (e.g., CPU) capacity of the substrate node A, and Cap_{AB} denote the bandwidth capacity of substrate link (A,B).

Likewise, we model a VN as a weighted, undirected graph $G^v = (V^v, E^v)$, whose vertices and edges represent virtual nodes and links, respectively. The weight Req_a of vertex a reflects the resource (e.g., CPU) requirement of the virtual node, while the weight t_{ab} denotes the traffic demand of the virtual link (a, b)[1]. In addition, each virtual node may specify additional constraints, including the type of the substrate nodes they may be mapped onto, geographical constraints, etc., such that a virtual node may be placed only on a specific group of substrate nodes. We denote the set of substrate nodes that may support a virtual node a as $\Theta(a) \subseteq V^s$.

We assume that there exists a mapping of virtual nodes and links to the substrate network, $\mathcal{F} : G^v \rightarrow G^s$. We also assume that the resource requirements Req_a and traffic demands t_{ab} of the VNs evolve over time such that the current mapping $\mathcal{F}$ is not representative of the current state of the network. Although our work is agnostic with respect to how reconfiguration is triggered (e.g., whether it is performed periodically or is initiated as soon as a performance measure crosses a predefined threshold), our focus is on updating the mapping $\mathcal{F}$ to align it with current network conditions.

[1]We will use upper-case (respectively, lower-case) letters to denote substrate (respectively, virtual) nodes throughout this paper.

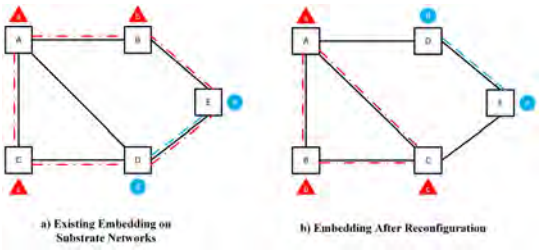

Fig. 1. Reconfiguration of virtual network requests

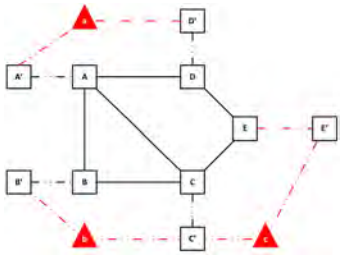

Fig. 2. Augmented graph

B. Reconfiguration Objectives

Our goal is to develop an online VN reconfiguration algorithm that remaps part of the VN requests so as to balance the load on the substrate nodes and links in a scenario where multipath routing could be supported [5]. Since the migration is more expensive than the reconfiguration of virtual links [4], we also aim to limit the number of virtual nodes that have to be migrated, so as to keep the reconfiguration cost low and minimize service disruption. A reconfiguration example is shown in Figure 1. The top part of the figure shows the two VN requests and the original embedding of the requests onto the substrate network, whereas the bottom part of the figure shows the new embedding of the two requests after reconfiguration has taken place.

We view this VNR problem as a multi-objective optimization problem, with three metrics to be minimized: substrate link utilization, substrate node workload, and the number of virtual nodes to be migrated. Specifically, we take link utilization as the primary objective and bound the other two, such that we formulate the goal in this form:

> *Minimize the maximum link utilization λ, under two constraints: resource utilization of each substrate node does not exceed $\rho\lambda$, and the total number of virtual nodes being migrated does not exceed a threshold M.*

M and ρ are parameters defined by the network operator. Parameter ρ may be used to adjust the tradeoff between minimizing the utilization of substrate links and nodes.

C. Augmented Graph

Our problem formulation makes use of the augmented graph presented in [15]. We start with the graph G^s that represents the substrate network and follow these steps to construct the augmented graph G^{aug} (refer also to Figure 2). For each substrate node A, we create a mirror node A', as well as an edge (A, A') with weight equal to the capacity of node A, Cap_A. For each virtual node a, we also create a corresponding node in the augmented graph. Recall that $\Theta(a)$ is the subset of substrate nodes to which virtual node a may be mapped. Therefore, for each substrate $A \in \Theta(a)$, we create an edge (a, A') between the virtual node a and the mirror node of A. For instance, in Figure 2 we assume that $\Theta(a) = \{A, D\}$, hence the augmented graph contains the edges (a, A') and (a, D'). The capacity of these edges is set to infinity.

In addition to the weight (capacity), we associate a cost with each edge in the augmented graph, as follows: the cost on all edges except the ones between virtual nodes and mirror substrate nodes is zero. If, in the current configuration, a virtual node a is mapped on, say, substrate node A, then the cost of edge (a, A') is also zero. Otherwise, the cost of the edge $(a, D'), A \neq D \in \Theta(a)$, is the reciprocal of the total traffic of virtual node a, i.e., $1/\sum_b t_{ab}$.

D. MIP Formulation

Using the augmented graph defined above, we formulate the VNR problem as the following mixed-integer programming (MIP) problem.

Decision Variables:

x_A^a: binary variable, indicating whether virtual node a is mapped onto substrate node A.

f_{AB}^{ab} : flow variable, indicating the fraction of traffic between virtual nodes a and b that is mapped onto substrate link (A, B).

MIP Formulation:

$$\min \ \lambda \tag{1}$$

$s.t.$

$$\sum_{A\in\Theta(a)} f_{aA'}^{ab} - \sum_{A\in\Theta(a)} f_{A'a}^{ab} = 1, \quad \forall a, b \in V^v, (a,b) \in E^v \tag{2}$$

$$\sum_{B\in\Theta(b)} f_{bB'}^{ab} - \sum_{B\in\Theta(b)} f_{B'b}^{ab} = -1, \quad \forall a, b \in V^v, (a,b) \in E^v \tag{3}$$

$$\sum_{A\in V^S} f_{AB}^{ab} - \sum_{A\in V^S} f_{BA}^{ab} = 0, \quad \forall a, b \in V^v, (a,b) \in E^v, B \in V^s \tag{4}$$

$$\sum_{(a,b)\in E^v} t_{ab} f_{AB}^{ab} \leq \lambda Cap_{AB}, \quad \forall A, B \in V^s \tag{5}$$

$$\sum_{a\in V^v} Req_a x_A^a \leq \rho\lambda Cap_A, \quad \forall A \in V^s \tag{6}$$

$$\sum_{b\in V^v} f_{aA'}^{ab} = \sum_{b\in V^v} x_A^a, \quad \forall a, b \in V^v, \ \forall A \in V^s \tag{7}$$

$$\sum_{a\in V^v} (1 - x_A^a) \leq M, \ \forall a \in V^v, A = Ext(a) \tag{8}$$

$$\sum_{A\in\Theta(a)} x_A^a = 1, \ \forall a \in V^v \tag{9}$$

$$0 \leq f_{AB}^{ab} \leq 1 \tag{10}$$

$$x_A^a \in \{0, 1\} \tag{11}$$

Remarks:

The objective of the MIP is to minimize the maximum utilization of any link in the SN.

Constraints (2)-(4) are the flow related constraints. Assume there exists a virtual link (a, b) between virtual nodes a and b, i.e., that the traffic between them $t_{ab} > 0$. Constraint (2) specifies that all traffic between a and b must pass pass through the mirror nodes that are connected to virtual node a in the augmenting graph. Similarly, constraint (3) ensures that all traffic will be routed to virtual node b via mirror nodes connected to that node in the augmented graph. Constraint (4) is the flow conservation constraint, and ensures that the net traffic in and out of a substrate node is zero.

Constraints (5) and (6) are the substrate link and node capacity constraints, respectively. Constraint (5) states that the total amount of traffic on a substrate link may not exceed λ times its capacity, while constraint (6) asserts that the resource requirement on a substrate node may not exceed $\rho\lambda$ times its capacity.

Constraint (7) maintains consistency between decision variables x and f, as it guarantees that, if virtual node a is mapped onto substrate node A, then all traffic associated with a will go through the augmented link between node a and the mirror node A'.

Constraint (8) ensures that the number of virtual nodes to be migrated does not exceed the threshold M. Here, $Ext(a)$ stands for the substrate node upon which virtual node a is mapped prior to reconfiguration. If, after reconfiguration, virtual node a is still placed upon substrate node $A = Ext(a)$, then the term $(1 - x_A^a)$ will be 0. Thus, the sum on the left hand side of the constraint equals the number of virtual nodes that are remapped due to reconfiguration.

Constraint (9) ensures that each virtual node is mapped to exactly one substrate node, while the last two constraints define the range of the decision variables.

IV. Reconfiguration Algorithm

A. General Procedure for Reconfiguration

Since VNR is an NP-hard problem, to tackle it efficiently we decompose it into two subproblems, namely, virtual node selection and virtual node remapping, described below.

Virtual Node Selection. Inspired by the work of [15], we use an LP relaxation based approach to select the virtual node to be migrated. First, we relax the integral constraints on variables x_A^a, i.e., constraints (11), and solve the resulting linear programming problem. Hence, virtual node selection is carried out by jointly considering the load on substrate links and nodes. Although LP problems are solvable in polynomial time, the computation time may be prohibitive for large-scale networks. To overcome this difficulty, we propose an approximation algorithm to obtain a near-optimal solution with small computation overhead. As a result, the delay to reach a reconfiguration decision may be kept low.

Once the LP problem is solved (optimally or using the approximation algorithm), we ascending sort variables x_A^a, where $A = Ext(a)$. Note that we can think of x_A^a as the likelihood that virtual node a is to remain in the same substrate node A. Consequently, we mark the M virtual nodes with the smallest value of x_A^a as the nodes to be migrated.

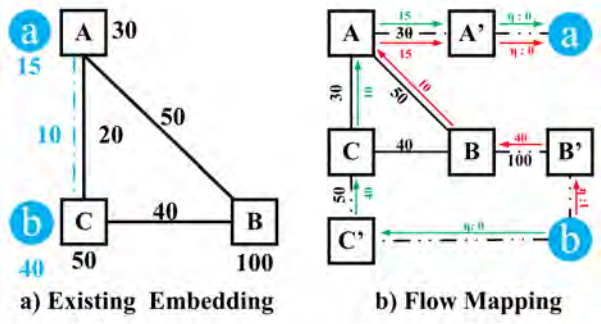

Fig. 3. An illustrative example to the path based formulation

Virtual Node Remapping: We have developed an algorithm based on a random walk on a Markov chain to filter substrate nodes for the virtual nodes selected in the solution to the previous subproblem, and use a MIP to map them back to the substrate network. This remapping phase takes into account load balancing across the substrate nodes and links.

B. Virtual Node Selection

As we mentioned above, solving a large-scale LP problem is computationally expensive, and may result in long reconfiguration delays that are not acceptable in an online scenario such as the one we are considering. Furthermore, we note that solving the LP problem exactly is *not* necessary: our focus is not on the exact values of variables x_A^a but rather on their relative values, since the goal is to rank virtual nodes based on their likelihood to be migrated. With this in mind, we build upon the work of [7] to design an approximation algorithm to obtain a near-optimal solution to the relaxed LP.

1) Path-based Formulation: In order to develop the approximation algorithm, we first present a path-based formulation that is equivalent to the link-arc MIP formulation (1)-(11), and relax the integral constraints. The path-based formulation is based on the observation that, by construction, the augmented graph imposes a connection between the mapping of a virtual node and routing. Specifically, if there is traffic flow between a virtual node a and a mirror node A', this implies a mapping of a onto substrate node A.

For simplicity, we denote the edges of the augmented graph between a substrate node A and its mirror node A' as e^A, and its capacity as $c(e^A)$ (recall that this is equal to the capacity of the corresponding substrate node). Similarly, we denote edges representing substrate links as e, and their capacity as $c(e)$. Let P_{ab} be the set of paths between virtual nodes a and b, and T_a denote the traffic originating at virtual node a, i.e., $T_a = \sum_{b \in V^v} t_{ab}$. We define the decision variables $x(p)$ as the amount of flow routed along path p. We also define r_p^a as a normalization factor which builds an association between the routing of the flow and the mapping of virtual nodes to substrate nodes: if routing flow along path p indicates

that virtual node a is mapped onto substrate node A, then $r_p^a = Req_a/T_a$. Note also that, since the flow on a path p indicates the substrate node to which the virtual nodes generating that flow are mapped, it also indicates whether or not these virtual nodes are to be migrated. Therefore, we introduce binary parameter η_p^a defined as follows: if path p indicates that virtual node a is to be migrated, then $\eta_p^a = 1$, otherwise $\eta_p^a = 0$.

An illustrative example to this is shown in Figure 3. The numerical value besides virtual network and substrate network stands for demands and capacities respectively. Two examples for the routing of the traffic are shown in Figure 3(b). If we route traffic along the green path, it implies that the virtual nodes a and b are placed on the substrate node A and C respectively. With the help of normalization factor r_p, the traffic we send on AA' and CC' will be the virtual node requests, namely, 15 and 40. Since this route does not imply virtual node migration, the penalty factor η will be set to 0 for this path. Likewise, if we send traffic along the red path, it will imply we are mapping virtual nodes a and b onto the substrate nodes A and B, and the amount of flow will be routed accordingly. The difference is that such this path would signify the migration of the virtual node b, and in return, the penalty factor η on this path will be set to 1.

In addition, we notice the problem of utilization minimization is equivalent to that of the throughput maximization. With a solution that will maximize the throughput, one can obtain a solution to the utilization minimization problem by scale down the traffic by the factor of the achieved throughput.

Based on these observations, we have the following path-based LP formulation of the relaxed version of the above MIP problem.

LP Formulation:

$$\max \ \lambda \tag{12}$$

$s.t.$

$$\sum_{p:e\in p} x(p) \le c(e), \quad \forall e \in E^s \tag{13}$$

$$\sum_{p:e^A\in p} r_p^a x(p) \le \rho c(A), \tag{14}$$

$$\forall a : (aA' \ or \ A'a) \in p, \forall A \in V^s$$

$$\sum_{p\in P_{ab}} x(p) \ge \lambda t_{ab}, \quad \forall a,b \in N^v \tag{15}$$

$$\sum_{p,a} \frac{\eta_p^a}{T_a} x(p) \le \lambda M \tag{16}$$

$$x(p) \ge 0, \quad \forall p \tag{17}$$

In this path based formulation, the objective function of the LP is to maximize the throughput for the traffic requirement. Constraint (13) enforces the capacity constraints for the substrate links, and is equivalent to constraint (5) in a throughput maximization scenario. Constraint (14) represents the capacity constraint for substrate nodes.

Based on the information from the augmenting graph, routing an amount $x(p)$ of traffic along this path determines the virtual nodes to be mapped to the substrate nodes at both ends of the path. Based on the definition of r_p^a, and combined with equation (18), one can verify that $\sum_{p:e^A\in p} r_p^a x(p)$ is the total amount of node resource requests placed on substrate node A from path p. By summing up over all paths p that contain this node A, constraint (14) is equivalent to constraint (6) in essence.

Constraint (15) ensures the traffic demand is satisfied. By maximizing the factor λ, we are sending as much flow as possible between each pair of the virtual nodes. As this constraint considers only the paths p between two specific virtual nodes in the augmented graph, and only non-negative flow is routed along each path p, we conclude that constraints (2) to (4), the flow continuity constraints, are satisfied.

Constraint (16) is equivalent to constraint (8), which ensures that the total number of virtual nodes to be migrated does not exceed the given threshold M. Since $\eta_p^a/T_a x(p) > 0$ only when path p indicates migration of virtual node a, the sum of all paths associated with virtual node a indicates the fraction of virtual node a to be migrated; consequently, the sum over all paths and all virtual nodes, $\sum_{p,a} \eta_p^a/T_a x(p)$, indicates the total (non-integer) number of virtual nodes to be migrated.

In addition, based on equation (18) below, one can verify that with an integer solution, each virtual node is mapped onto one and only one substrate node, satisfying constraint (9).

Finally, to prove that this path-based LP formulation is equivalent to the MIP problem we discussed in Section III when we relax the integral constraint, we now show how to obtain the values of the decision variables of the MIP problem from the LP solution $x(p)$.

First note that x_A^a is no longer a binary variable but rather takes values in $[0,1]$ and represents the fraction of traffic from virtual node a that goes through substrate node A, i.e., it is routed along the edge (a, A') in the augmented graph. Therefore, we may obtain x_A^a as the ratio of the traffic on that path to the total traffic originating at a:

$$x_A^a = \sum_{p:(a,A')\in p} x(p)/\lambda T_a \tag{18}$$

Similarly, f_{AB}^{ab} may be obtained by taking the ratio of the amount of traffic that passes through an edge (A, B) over the total traffic from virtual node a to b, i.e.,

$$f_{AB}^{ab} = \sum_{p:(AB)\in p, p\in P_{ab}} x(p)/\lambda t_{ab} \tag{19}$$

2) Minimum Cost Multi-Commodity Flow Perspective: From our previous discussion, we know that the routing of the traffic implies the mapping of virtual nodes. This helps to build a connection between the mapping of the virtual nodes and multi-commodity problem. Indeed, if we see the migration penalty factor in constraint (16) as the cost for the flow along path p, bounded by the cost λ, then the problem can be seen as a Minimum Cost Multi-Commodity Flow (MCMCF) problem.

In [7], a fully polynomial time approximation scheme (FPTAS) has been proposed to solve an LP-based MCMCF approximately. Specifically, the FPTAS obtains a solution that

is within a factor of $(1+\omega)$ of the optimal solution and runs in time $\widetilde{O}(\omega^{-2}|E|^2)$, where E is the number of edges in the augmenting graph.

Our MCMCF formulation above is similar to the MCMCF problem in [7], except for the following two differences:

1) There exists two set of capacity constraints, namely, constraint (13) and constraint (14).
2) In (16), the cost is not bounded by a constant value.

This means that the FPTAS of [7] may not be directly applied to solve our problem. As the first step we take to accommodate the differences, we replace λ by a constant factor σ (how to determine σ shall be addressed later). Which converts constraints (16) into:

$$\sum_{p,a} \frac{\eta_p^a}{T_a} x(p) \leq \sigma M \tag{20}$$

The cost will then be agnostic of the throughput λ. The obstacle to apply FTPAS [7] is the existence of capacity constraints. In the remainder of this section, we show how to extend the FPTAS of [7] to solve for our problem.

3) Dual Problem: We start by stating the dual form of the MCMCF problem.

Decision Variables:

$l(e)$: Associated with constraint (13), interpreted as the length of substrate edge on augmented graph

$l(e^A)$: Associated with constraint (14), interpreted as length of the edge between a substrate and mirror substrate node

$z(ab)$: Associated with constraint (15), interpreted as the shortest path between a and b

ϕ: Associated with constraint (20), interpreted as the penalty factor for migration.

Dual Problem

$$\min_{l,\phi} D(l,\phi) = \sum_e c(e)l(e) + \rho \sum_{e^A} c(e^A)l(e^A) + \sigma M \phi \tag{21}$$

$s.t.$

$$\sum_{e\in p} l(e) + r_p^a l(e^A) + r_p^b l(e^B) + (\frac{\eta_p^a}{T_a} + \frac{\eta_p^b}{T_b})\phi \geq z(ab),\ \forall p \in G^{aug},\ \forall a,b \tag{22}$$

$$\sum_{ab} t_{ab} z(ab) \geq 1 \tag{23}$$

We denote the dual objective as $D(l,\phi)$ in (21), which consists of two parts: total length of the augmenting graph $\sum c(e)l(e) + \rho \sum c(e^A)l(e^A)$ weighted by the edge capacity, and weighted migration penalty $\sigma M\phi$. Therefore, the dual problem can then be seen as an assignment of length functions $l(e)$, $l(e^A)$ to the edges of the augmented graph, and penalty factor ϕ such that the weighted sum $D(l,\phi)$ is minimized.

We define the length of a path between virtual node a and b under the length function as follows:

$$\sum_{e\in p} l(e) + r_p^a l(e^A) + r_p^b l(e^B) + (\frac{\eta_p^a}{T_a} + \frac{\eta_p^b}{T_b})\phi \tag{24}$$

More specifically, if path p implies no migration for either a or b, then the distance between a and b will be measured by the length function; otherwise, for each virtual node to be migrated, an additional penalty of ϕ/T is incurred. When we have a feasible solution to the dual problem, we interpret $z(ab)$ as the shortest path between virtual nodes a and b under the length function and penalty factor.

We now design a primal-dual approximation algorithm to solve the MCMCF problem.

4) FPTAS Algorithm to Solve the MCMCF: The FPTAS for the MCMCF problem is shown as Algorithm 1, and is based on the FPTAS in [7]. At a high level, the algorithm operates as follows. We assign an initial length to all edges, and then we iteratively route flow along the shortest path. We determine δ and precision factor ϵ in the same way as in [7].

The length of a path p is defined in (24). Each time we send flow along path p, the length of each edge along this path increases, thus reducing the likelihood that subsequent flow will follow this edge. This helps to spread traffic evenly among all edges and increases the overall throughput.

The algorithm proceeds in phases, each phase having $|V^v|$ iterations. During the a^{th} iteration, we route the flow along the shortest path between virtual node a and all its neighbors in the VN under the current length function and penalty factor. For example, if virtual nodes a and b are neighbors in the VN with traffic demand t_{ab}, then in the iteration corresponding to node a, we route t_{ab} amount of flow between virtual nodes a and b.

The a^{th} iteration of each phase of the algorithm considers traffic from virtual node a and terminates when all traffic from that node has been routed. At each step of the a^{th} iteration, we build a Dijkstra shortest path tree rooted on vertex a of the augmented graph. Flow is routed along the shortest path, and the amount of flow routed between each pair of virtual nodes is equivalent to the minimum remaining edge capacity along this shortest path. In particular, the flow we put on the augmented edge that represents the substrate node is r_p^a times the flow we put elsewhere. In terms of minimum capacity along the path, capacity of the augmenting edge is factorized by $1/r_p^a$ to ensure that the substrate node is not overloaded.

Each time we route flow along a particular edge, the remaining capacity of that edge is updated accordingly, along with the length function $l(e)$ and ϕ. If the remaining flow is less than the capacity of the edge, we route all remaining flow along the shortest path. The algorithm will terminate when the weighted sum in (21) $D(l,\phi) \geq 1$.

Note that the total amount of flow we route will violate the capacity constraints. To obtain a feasible solution, we determine the most congested links and we also compute the total number of migrated nodes from $x(p)$. Let Δ be the maximum ratio by which any of the constraints are violated. Then, we scale down the all the flows by a factor of Δ to obtain a feasible solution.

Using proof techniques similar to the ones in [7], we can show that this algorithm obtains a solution that is within $1+\omega$

Algorithm 1 FPTAS for MCMCF

Input:
G^{aug}: augmented graph, $c(e)$: edge capacity on G^{aug}
t_{ab}: traffic request, r_p^a: normalizing factor
T_a: total traffic from a, ϵ: precision factor
ρ: balancing factor, σ: estimated throughput
Output: λ: maximum utilization of substrate link
$x(p)$: assignment of flow to the substrate network

Initialize $l(e) = \delta/c(e),\ l(A) = \delta/\rho c(e^A), \phi = \delta/M$;
while $D(l,\phi) < 1$ **do** ▷ Phase
  for $a = 1, 2, \cdots, |V^v|$ **do** ▷ Iteration
    $t^r_{ab} = t_{ab}, b = 1, ..., |V^v|$ ▷ Remain Traffic
    $c^r(e) := c(e)$ ▷ Remain Capacity
    while $D(l,\phi) < 1$ *and* $t'_{ab} > 0, \exists\ k$ **do** ▷ Step
      $SPT_a :=$ shortest path tree rooted on node a
      for all b with $t^r_{ab} > 0$ **do**
        ▷ Find min remaining capacity on path p_{ab}
        $c := \min\{c(e), \frac{1}{r^a_p}\rho c(e^A), \frac{1}{r^b_p}\rho c(e^B)\}$,
          $p_{ab} \in SPT_a,\ e, e^A, e^B \in p_{ab}$;
        Route flow along p_{ab}
          ▷ Update edge length, capacity and cost
        $c^r(e) := c^r(e) - c,\ e \in p_{ab};\ t^r_{ab} = t^r_{ab} - c$
        $l(e) = (1 + \epsilon c/c(e))l(e)$
        $l(e^A) = (1 + \epsilon r^a_p c/\rho c(e^A))l(e^A)$
        $l(e^B) = (1 + \epsilon r^b_p c/\rho c(e^B))l(e^B)$
        $\phi := (1 + \epsilon c\eta^a_p/T_a M + \epsilon c\eta^b_p/T_b M)\phi$
      end for
    end while
  end for
end while
Scale down the flow to obtain feasible solution.

of the optimal one, and its runtime is $\tilde{O}(\omega^{-2}|E|^2)$. Due to page constraints, the proof is omitted.

5) Bisection Search for σ: Recall that in the MCMCF formulation, we must set parameter $\sigma = \lambda^\star$ in order to solve the problem and obtain the optimal assignment of routes. Since we do not know $\lambda^\star$ in advance, we first establish lower and upper bounds for σ and then carry out a binary search to find a value for σ that yields a solution close to $\lambda^\star$.

As reconfiguration shall yield a solution no-worse than the current configurations, its throughput shall serve as lower-bound for σ. The upper-bound of the throughput is when the load on substrate links or nodes is completely balanced, in another word, we take the minimum ratio of $\{link\ resource/link\ request, node\ resource/node\ request\}$ as the given upper bound for the throughput value.

With the upper and lower bound established, we can then carry out the bisection search for a suitable σ. We denote the upper and lower-bound for σ as σ^U and σ^L respectively. We use the mid-point of the interval, $(\sigma^L + \sigma^U)/2$ as estimated σ^E. The bisection will terminate if the throughput achievable via FPTAS, $\lambda(\sigma^E)$, is close to σ^E, otherwise, we shall update σ^L and σ^U accordingly and iteratively repeat the process until this we find σ.

Algorithm 2 Overall Algorithm for MCMCF

Input:
σ^L: σ upper bound σ^U: σ lower bound
Output:
Virtual nodes to be migrated.

Construct the augmented graph G^a
Find lower and upper-bound, σ^L and σ^U
while $\sigma \neq \lambda^E$ **do**
  setting $\sigma^E = (\sigma^L + \sigma^U)/2$
  $x(p), \lambda(\sigma^E) \leftarrow$ solve MCMCF by FPTAS
if $\sigma^E > \lambda(\sigma^E)$: $\sigma^U = \lambda(\sigma^E)$, **else**: $\sigma^L = \lambda(\sigma^E)$
end while
Obtain the migration indicator from $x(p)$ from (18)
Select virtual nodes with largest migration indicator

6) Complete Algorithm: The algorithm for virtual nodes selection is shown as Algorithm 2. We first construct augmented graph that includes the SN and VNs, and then we iteratively solve the MCMCF problem with a given parameter σ. We terminate this process once we identify an appropriate value for σ.

After normalization, we obtain the migration x^a_A based on the assignment of flow in the primal problem according to equation (18), and we use this value of the indicator for virtual node migration. Then, we select M virtual nodes with the largest migration indicator, mark them for reconfiguration, and pass them to the second subproblem that we discuss next.

C. Remapping of Selected Virtual Nodes

Inspired by the work of [8] and [9], we propose an algorithm to map the virtual nodes selected by the previous subproblem onto new substrate nodes, based on random walk on a Markov chain. For each virtual node a to be migrated, the algorithm assigns a numerical value to each substrate node A that represents the fitness of mapping a to A. We use this fitness value to reduce the search space for placing a virtual node. Specifically, for each virtual node, we select the N_s substrate nodes with the highest fitness value, and then search for the most suitable substrate node using an MIP.

The algorithm to assign a fitness value to the substrate nodes is presented as Algorithm 3. First, we compute the remaining link capacity $c^r(e)$ after removing the virtual nodes to be migrated. For each virtual node to be migrated, we compute the traffic between this node and all the substrate nodes. For example, if node a is selected for migration, and it communicates with nodes b and d, both of whom are mapped on substrate node A, the amount of traffic between a and A is $t_{ab} + t_{ad}$. Based on this value, we generate a probability vector for virtual node a that represents the initial probability of mapping a to each substrate node A:

$$\pi^{(0)}_{a\to A} = \sum_{b:b\to A} t_{ab} / \sum_b t_{ab} \tag{25}$$

Algorithm 3 Selection of Candidate Substrate Nodes

Input: $c^r(e)$: remaining substrate link capacity
t_{ab}: traffic request; T_M: iteration numbers
N_s: number of candidate substrate nodes to be selected
Output:
Set of candidate substrate nodes for each virtual node
1: Construct initial distribution according to equation (25)
2: Construct transition probability with equation (26)
3: **for** $j = 1, 2, \cdots, T_M$ **do**
4: Update probability distribution according to (27)
5: **end for**
6: For each a, select N_s substrate node with largest π

We also generate the transition probability matrix by letting the transition probability from A to B as:

$$tp_{AB} = c^r(e_{AB}) / \sum_D c^r(e_{AD}) \tag{26}$$

At each step, the probability $\pi_{a\to A}^{(t+1)}$ is updated by the follows expression:

$$\pi_{a\to A}^{(t+1)} = \mu\pi_{a\to A}^{(t)} + (1-\mu)\sum_B \gamma_{AB}^t tp_{BA}\pi_{a\to B}^{(t)} \tag{27}$$

where μ is a smoothing factor which controls the rate of transition. The term γ_{AB}^t is defined as

$$\gamma_{a,B}^t = \pi_{a\to B}^{(t)} / \sum_a \pi_{a\to B}^{(t)} \tag{28}$$

and captures the interference between the placement of different virtual nodes on the substrate link.

The algorithm terminates after a T_M steps, at which time, for each virtual node a to be migrated, we select the N_S substrate nodes with the largest π value as the candidate substrate nodes for migration.

The operation of this algorithm can be interpreted as follows. The initial probability values are determined by the intensity of traffic between a virtual node and a substrate node, such that a substrate node that receives more traffic from a virtual node will have higher weight in the remapping process. With this initial probability distribution, we start a random walk on the Markov chain. During the t^{th} transition step, aside from $\pi^{(t-1)}$, the probability distribution from the previous step, two additional factors affect the transition rate: the remaining link capacity, and parameter μ. Specifically, the transition rate is proportional to the remaining link capacity, meaning that virtual nodes are more likely to be placed onto substrate nodes with a larger amount of available link capacity towards the destination. Also, the transition rate is affected by the value for other virtual nodes, i.e., if a link is likely to be shared among several virtual links, the available network resource decreases accordingly, and the probability of sharing also decreases. We note that we have the algorithm terminate after a fixed number of steps, instead of when the probability vector converges. This helps with increasing locality in the mapping, as it tends to place a virtual node upon a substrate node close to other substrate nodes it communicates with, which in turn helps reduce the overall traffic.

In summary, by constructing a Markov chain in the above manner, and carrying out a random walk for a fixed number of iterations, we obtain a probability vector that represents the likelihood of placing the selected virtual nodes onto substrate nodes. The probability reflects the fitness of placement by taking into account available bandwidth, interference with other nodes, and locality of communication. For each virtual node, we select N_s substrate nodes as candidates.

As a final step, given the above probability vector, we determine a new mapping of the selected virtual nodes onto the substrate network by solving an MIP which aims at minimizing jointly the node utilization, link utilization, and cost of flow. This MIP can be efficiently solved by setting N_s to a small number. The formulation, in essence, is similar to MIP formulation in Section III. Due to page constraints, the MIP formulation is omitted.

V. Evaluation

We consider three different schemes in our evaluation study: (1) no reconfiguration (no-rcnfg), (2) reconfiguration of virtual links only (l-rcnfg) using the algorithm in [5], and (3) reconfiguration of both virtual links and nodes (ln-rcnfg(M)) with our proposed algorithm, where M stands for the maximum number of nodes that may be migrated during each iteration; we use $M = 2, 4, 6$ in this study. We compare the three schemes on three performance metrics: maximum (substrate) link utilization, maximum (substrate) node utilization, and InP revenue, defined as the sum, over all requests, of the bandwidth and node resource request of a request times the request's lifetime.

For the experiments, we developed an embedding testbed similar to the one in [5]. The topology of the substrate network and the virtual network requests are generated by the GT-ITM modeling tool [14] The substrate network consists of 50 nodes. The capacity of the substrate links and nodes follow a uniform distribution in $[50, 100]$. The number of virtual nodes in each request is an integer uniformly distributed in $[2, 10]$, while the requirements of virtual nodes and links are also uniformly distributed in $[20, 40]$.

The arrival and departure intervals of the virtual requests follow an exponential distribution, with a mean inter-arrival time of 10 and a mean inter-departure time of 100. For each simulation, we generate 1000 virtual requests, and embed each arriving virtual using the same virtual network embedding algorithm of [5]. We assume that reconfiguration takes place periodically at constant intervals, and we vary the interval between reconfiguration events. And for each VM to be migrated, we select $N_s = 10$ substrate nodes as candidate substrate nodes.

Figure 4 plots the maximum link utilization under three different cases. Compared to the case of no reconfiguration (non-rcnfg), our algorithm (ln-rcnfg) reduces link utilization by at least 28%, and further as the maximum number M of nodes to be migrated increases. Compared to reconfiguration

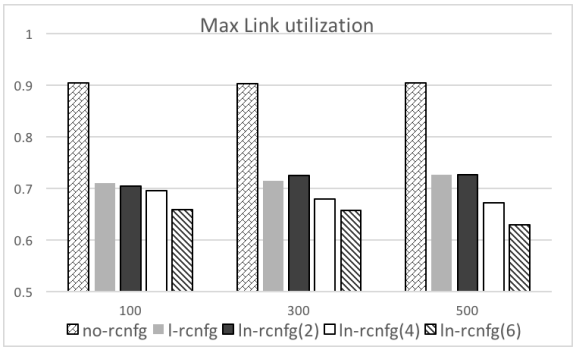

Fig. 4. Maximum link utilization vs. number of reconfiguration events

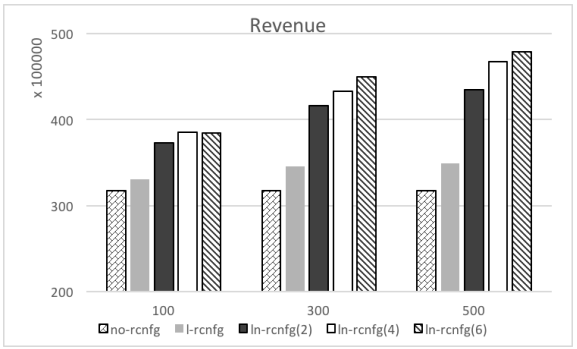

Fig. 6. InP revenue vs. number of reconfiguration events

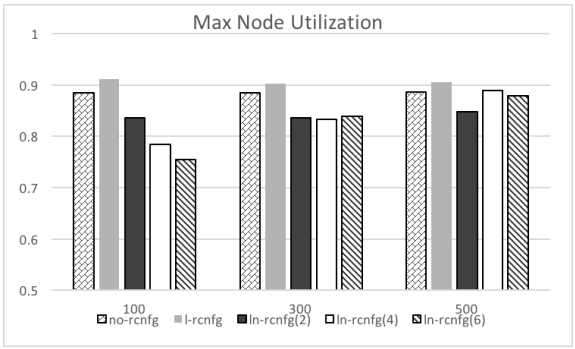

Fig. 5. Maximum node utilization vs. number of reconfiguration events

of virtual links only (l-rcnfg), our scheme decreases the maximum link utilization up to 15% when $M = 6$. Figure 5 plots the maximum node utilization. Again, our algorithm improves upon the no- or link only-reconfiguration by up to 22%, but the results show that the improvement is sensitive to the value of M, which must be selected appropriately.

Finally, the revenue of the infrastructure provider is shown in Figure 6. As we can see, link only reconfiguration results in higher revenue than no reconfiguration, as expected, as it may accommodate additional requests. Our algorithm provides further increase in revenue, between 12-36% compared to link-only reconfiguration, and from 17-50% compared to no reconfiguration.

We note that revenue and resource utilization are two seemingly conflicting goals, since an increase in revenue often implies higher resource utilization as more demands are mapped onto the physical infrastructure. An important insight that we gain from the above set of results, is that a tradeoff between utilization and revenue does not necessarily exist. Migrating a small fraction of virtual nodes will not only help accommodate additional virtual network requests, but will also drive down resource utilization. In other words, by using intelligent algorithms such as the ones we presented in this work, the InP may benefit from an increase in revenue while also providing customer with better service.

VI. Concluding Remarks

Resource reallocation is essential to the performance in a virtual network environment. We designed a reconfiguration scheme that aims at load balancing while minimize the cost of reconfiguration by limiting the number of virtual nodes to be migrated. We model this problem as a network flow problem, and develop two algorithms to select both the virtual nodes to be migrated and the substrate node where to place those virtual nodes. Simulation results show that even when a small number of nodes are migrated, our algorithm balances the load and helps maximize InP revenue, reconciling two seemingly conflicting goals.

References

[1] A. Wang, M. Iyer, R. Dutta, G.N. Rouskas, and I. Baldin. Network virtualization: Technologies, perspectives, and frontiers. *Journal of Lightwave Technology*, 31(4):523–537, 2013.
[2] N.M. Mosharaf, K. Chowdhury and R. Boutaba. A survey of network virtualization. *Computer Networks*, 54(5):862–876, 2010.
[3] A. Fischer, J.F. Botero, M.T. Beck, H. De Meer, and X. Hesselbach. Virtual network embedding: A survey. *IEEE Communications Surveys & Tutorials*, 15(4):1888-1906, 2013.
[4] Y. Zhu and M.H. Ammar. Algorithms for assigning substrate network resources to virtual network components. Proc. *IEEE INFOCOM*, 2006.
[5] M. Yu, Y. Yi, J. Rexford, and M. Chiang. Rethinking virtual network embedding: substrate support for path splitting and migration. *ACM SIGCOMM Comp. Comm. Rev.*, 38(2):17-29, 2008.
[6] Z. Cai, F. Liu, N. Xiao, Q. Liu, and Z. Wang. Virtual network embedding for evolving networks. Proc. *IEEE GLOBECOM,* 2010.
[7] G. Karakostas. Faster approximation schemes for fractional multicommodity flow problems. *ACM Tran. Algorithms (TALG)*, 4(1):13, 2008.
[8] L. Gong, *et al.* Toward profit-seeking virtual network embedding algorithm via global resource capacity. Proc. *IEEE INFOCOM*, 2014.
[9] X. Cheng, *et al.* Virtual network embedding through topology-aware node ranking. *ACM SIGCOMM Comp. Comm. Rev.*, 41(2):38–47, 2011.
[10] A.V. Goldberg, *et al.* An implementation of a combinatorial approximation algorithm for minimum-cost multicommodity flow. Proc. *IPCO*, pp. 338-352. Springer, 1998.
[11] I. Fajjari, *et al.* Vnr algorithm: A greedy approach for virtual networks reconfigurations. Proc. *IEEE GLOBECOM*, 2011.
[12] B. Dab, *et al.* Vnr-ga: Elastic virtual network reconfiguration algorithm based on genetic metaheuristic. Proc. *IEEE GLOBECOM*, 2013.
[13] P.N. Tran and A. Timm-Giel. Reconfiguration of virtual network mapping considering service disruption. Proc. *IEEE ICC*, 2013.
[14] GT-ITM. http://aiweb.techfak.uni-bielefeld.de/content/bworld-robot-control-software/.
[15] N.M. Mosharaf, *et al.* Virtual network embedding with coordinated node and link mapping. Proc. *IEEE INFOCOM*, pp. 783-791, 2009.
[16] L. Gao and G. Rouskas. Network-aware virtual request partitioning based on spectral clustering. Proc. *IEEE ICCCN*, 2016.