Rethinking Virtual Network Embedding: Substrate Support for Path Splitting and Migration

Minlan Yu[1], Yung Yi[2], Jennifer Rexford[1], Mung Chiang[2]
[1] Computer Science, Princeton University, Email: {minlanyu,jrex}@cs.princeton.edu
[2] Electrical Engineering, Princeton University, Email: {yyi,chiangm}@princeton.edu

***Abstract*—Network virtualization is a powerful way to run multiple architectures or experiments simultaneously on a shared infrastructure. However, making efficient use of the underlying resources requires effective techniques for *virtual network embedding*—mapping each virtual network to specific nodes and links in the substrate network. Since the general embedding problem is computationally intractable, past research has focused on two main approaches: (i) significantly restricting the problem space to allow efficient solutions or (ii) proposing heuristic algorithms that do not use the substrate resources efficiently. In this paper, we advocate a different approach: rethinking the design of the substrate network to enable simpler embedding algorithms and more efficient use of resources, without restricting the problem space. First, we allow the substrate network to split a virtual link over *multiple* substrate paths. Second, we employ *path migration* to periodically re-optimize the utilization of the substrate network to help accommodate new requests. In addition, we run node-mapping algorithms that are *customized* to common classes of virtual-network topologies. Our simulation experiments show that path splitting, path migration, and customized embedding algorithms enable a substrate network to satisfy a much larger mix of virtual-network requests in practice.**

I. Introduction

Network virtualization has emerged as a powerful way to diversify the future Internet by running multiple network services and experiments simultaneously on a shared substrate network [1], [2], [3]. Network virtualization may play an important role in supporting multiple architectures as a long-term solution for the future Internet [4]. Making efficient use of the underlying substrate network requires effective techniques for *virtual network (VN) embedding*—mapping a virtual network to specific links and nodes in the substrate network. However, the embedding problem is extremely difficult in theory and in practice, due to the following four properties:

1) ***Diverse topologies.*** The virtual networks have diverse topologies. Still, certain topological structures—such as a tree or hub-and-spoke—may be common in practice, depending on the service the virtual networks provide.
2) ***Resource constraints.*** Each VN request has resource constraints that the embedding must satisfy. Many VNs have both node (e.g., CPU) and link (e.g., bandwidth) constraints, making the embedding problem hard to solve.
3) ***Online requests.*** The VN requests arrive dynamically, and stay in the network for some period of time before departing. To be practical, the embedding algorithm must handle VN requests as they arrive.
4) ***Admission control.*** Since the substrate resources are limited, some VN requests must be rejected or postponed. That is, the embedding algorithm must perform admission control to reserve resources for accepted VNs and block requests when insufficient resources are available.

Several of these properties make the VN embedding problem different, and more difficult, than the Virtual Private Network (VPN) design problem [5], [6]. In particular, the VN embedding problem must deal with both node and link constraints for arbitrary topologies. In contrast, VPNs usually have a standard topology, such as full mesh and hub-and-spoke [7]. Moreover, the resource constraints in a VPN are typically just bandwidth requirements, specified by a traffic matrix (i.e., the traffic volume for each pair of nodes).

In fact, the VN embedding problem is computationally intractable, even if some of the four above-mentioned properties are ignored. The computational challenges of VN embedding have led researchers to focus on heuristic solutions [8], [9], [10]. In particular, the past work has typically restricted the problem space in one or more dimensions to enable efficient heuristics, at the expense of limiting the practical applicability of the solutions. In this paper, we advocate a different approach. Instead of restricting the problem space, we *make the substrate network more supportive of the VN embedding problem*. This allows us to create simpler embedding algorithms that make more efficient use of the substrate resources. We argue that our proposed enhancements to the substrate network are realizable in practice. The key contributions in this paper are summarized in the following:

1) ***Path splitting.*** We propose that the substrate network allow a virtual link to map to *multiple* underlying paths, with a flexible path-splitting ratio. Path splitting enables a polynomial-time algorithm for virtual-link embedding, makes more efficient use of the substrate bandwidth, and enhances robustness to link failures.
2) ***Path migration.*** For more efficient handling of online requests, we periodically re-optimize the embedding of existing virtual networks. We realize this by adapting the path-splitting percentages or selecting alternate substrate paths for a virtual link. More practical than node migration, path migration provides an efficient way of tackling online embedding problems, because we can find the optimal path migration solution in polynomial time.
3) ***Modularized algorithm.*** In addition to a general node-mapping algorithm, we also run customized algorithms for specific common topologies, like tree and hub-and-spoke. The modularized algorithm reduces computation

TABLE I
NOTATIONS OF VIRTUAL NETWORK EMBEDDING PROBLEM

G^s	Substrate network
N^s	Nodes of substrate network
L^s	Links of substrate network
A^s_N	Node attribute of substrate network
A^s_L	Link attribute of substrate network
$\mathcal{P}^s$	Paths on substrate network
G^v	Virtual network
N^v	Nodes of virtual network
L^v	Links of virtual network
C^v_N	Node constraint of substrate network
C^v_L	Link constraint of substrate network
R_N	Resources allocated for virtual network nodes
R_L	Resource allocated for virtual network links

time and uses substrate resources efficiently by capitalizing on the unique properties of the VN topology.

The remainder of this paper is organized as follows: In section II, we define the general VN embedding problem and discuss previous work. Section III describes our modularized VN embedding algorithm. Sections IV and V discuss the use of path splitting and path migration to make the substrate network flexible and supportive of the VN embedding problem. Section VI shows simulation results to validate our algorithm, and we conclude the paper in Section VII.

II. VIRTUAL NETWORK EMBEDDING PROBLEM

In this section, we first describe the general VN embedding problem with its various objectives, and explain why the VN embedding problem is very hard to solve. Then, we discuss how past work has focused primarily on restricting the problem space for efficient solutions.

A. General Virtual Network Embedding Problem

Substrate network. We denote the substrate network by an undirected graph $G^s = (N^s, L^s, A^s_N, A^s_L)$, where N^s and L^s refer to the set of nodes and links, respectively [1]. Substrate nodes and links are associated with their attributes, denoted by A^s_N and A^s_L, respectively. In this paper, we consider CPU capacity and location for node attributes, and bandwidth capacity for link attribute. We also denote by $\mathcal{P}^s$ the set of all the paths in the substrate network.

Virtual network request. We denote by an undirected graph $G^v = (N^v, L^v, C^v_N, C^v_L)$ a virtual network request. The network topology (N^v, L^v) is a logical network in a sense it should be configured as a sub-network of the substrate network. A VN request typically has link/node constraints that are specified in terms of attributes of the substrate network. We denote by C^v_N and C^v_L such constraints for virtual nodes and links. An example virtual network request is: "connect two nodes $A, B \in N^V$ with constraints that node A and B should be in California and New Jersey, respectively, and we need 5 Mbps bandwidth on the virtual link between A and B."

[1]We use superscript to refer to substrate or virtual network, and use subscript to refer to nodes or links, unless otherwise specified.

VN embedding problem. A virtual network embedding problem is defined as a mapping $\mathcal{M}$ from G^v to a subset of G^s, such that the constraints in G^v are satisfied, i.e.,

$$\mathcal{M} : G^v \mapsto (N', \mathcal{P}', R_N, R_L),$$

where $N' \subset N^s$ and $\mathcal{P}' \subset \mathcal{P}^s$, and R_N and R_L are the node and link resources allocated for the VN requests. The VN network embedding can be naturally decomposed into node and link mapping:

$$\begin{array}{rl} \text{Node Mapping:} & \mathcal{M}^N : (N^v, C^v_N) \mapsto (N', R_N), \\ \text{Link Mapping:} & \mathcal{M}^L : (L^v, C^v_L) \mapsto (\mathcal{P}', R_L). \end{array}$$

Objectives. In addition to multiple constraints, the VN embedding problem may have multiple objectives. In this paper, our objective is to jointly maximize the revenue and minimize the cost from the perspective of the substrate network provider. Revenue and cost can be defined in various ways, depending on the substrate network provider's service strategy and the virtual networks' interests. A vector of two objectives can be scalarized into one objective by weighing revenue with cost. However, finding the right weight depends on the economic model of the corresponding virtual network service. Thus, we maintain a two-dimensional vector objective function.

Due to multiple objectives and multiple constraints, finding the optimal solution turns out to be a very difficult problem, i.e., NP-hard for node and/or link constraints of VN requests, even in the offline case. For example, assigning nodes to substrate network without violating bandwidth constraints can be reduced to the *multiway separator problem*, which is NP-hard [11]. Provided that the nodes are pre-selected, the link mapping problem for the requests with link constraints is still NP-hard (as discussed in more detail in section IV).

The online problem is even more difficult to solve than the offline one. Mathematically, the online embedding problem can be formulated and solved by dynamic programming. However, dynamic programming techniques are impractical here, mainly because *(i)* statistics of incoming VN requests are generally unpredictable, and *(ii)* the search space is prohibitively large with respect to the size of the substrate network.

B. Virtual Network Embedding Algorithms

Related work mainly addresses the hardness of the problem by *relaxing one or more of the key properties* of the problem (see table II). These key properties include (i) whether requests are processed online or not, (ii) whether the requests may have link constraints, node constraints, or both, (iii) whether admission control (AC) is performed to reject requests when insufficient resources are available, and (iv) what virtual topologies are supported.

Several of the previous studies focus on the *offline* problem, where all VN requests are known in advance. Zhu and Ammar [8] assume that the substrate network resources are unlimited, and aim at achieving load balancing in the substrate network, obviating the need for admission control. The VN-embedding problem for the requests with general topology is solved by subdividing the requests into multiple star topologies to allocate more substrate resource to the center of each decomposed

TABLE II
VIRTUAL NETWORK EMBEDDING ALGORITHMS

	on/offline	CPU/BW	AC	topology	objective
NetFinder[8]	offline	both	no	general	load balancing
[10]	offline (one req.)	both	no	backbone-star	cost
[9]	online	BW	no	traffic matrix	cost
Emulab [12]	online	BW	primitive	general	cost
This paper	online	both	yes	general	revenue & cost

star topology. Lu and Turner [10] also consider an offline problem for only a *single* virtual network with a backbone-star topology, where their goal is to minimize the cost. They assume that only bandwidth constraints are imposed, and the substrate network resources are unlimited with no admission control needed.

In regard to the online problem, Fan and Ammar [9] consider dynamic topology reconfiguration policies for virtual networks with dynamic communication requirements, but no consideration of the node constraints such as CPU. They also assume that substrate network resources are unlimited to accept all requests (i.e., no admission control) and try to find a strategy to minimize cost. Zhu and Ammar [8] also solve the online problem by recalculating the whole embedding solution periodically, which is computationally complex. The Emulab testbed [12] considers the online embedding problem with the bandwidth constraint. The node constraint in Emulab is provided as the exclusive use of nodes, i.e., different virtual networks cannot share a substrate node. Admission control is not explicitly addressed in [12], but we can conjecture an admission control that rejects request if the bandwidth/node resources are insufficient.

As we have described so far, past research on VN embedding algorithms tries to come up with an efficient solution by looking at only partial aspects of the problem addressed in Section I. This clearly limits the applicability of the algorithms. We highlight that we deal with the most general cases by considering all aspects of the four properties in the embedding algorithm design. In terms of the objective, our work is also more general, i.e., consideration of revenue and cost simultaneously. Rather than avoiding certain properties of the problem, we rethink the VN embedding problem by *making the substrate network more flexible and supportive to this problem, in conjunction with customization of the algorithms for different types of requests.* It turns out the flexibility of path splitting and migration is indeed very useful.

III. MODULARIZED ALGORITHM FOR DIVERSE REQUESTS

In this section, we first give an overview of our algorithm, which consists of (i) node mapping, (ii) link mapping, and (iii) migration. In the node mapping, we modularize the algorithm for the requests with different topological structures. In the link mapping, we treat the requests with/without path splitting separately. We devote the rest of this section to elaborate on the modularized node mapping and link mapping only for the requests without path splitting. We discuss the details on the link mapping for the request with path splitting and migration, which are our key contributions, in sections IV and V, respectively.

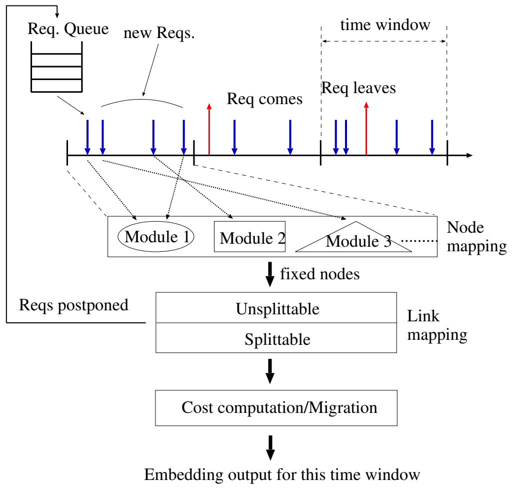


Fig. 1. Algorithm overview

A. Algorithm Overview

Our algorithm collects a group of incoming requests during a *time window* and then tries to allocate substrate resources to satisfy the constraints imposed by each request. Figure 1 depicts the overview of the algorithm for one time window.

Node and link mapping correspond to the initial resource-mapping. Our mapping algorithm is designed to accept as many requests as possible, resulting in revenue maximization. Some requests may not be accepted immediately, in which case they are stored in the *request queue*. The requests in the request queue are dropped if they do not have a chance to be served within some *delay*. This delay corresponds to the time that a request is willing to wait (see Section V for more discussions on the engineering issues on delay). The requests in the request queue are processed at the next time-window.

In both node and link mapping, we process the requests in the order of decreasing revenue. The revenue is defined based on the "agreement" between the virtual networks and the substrate providers. Thus, from the substrate providers' perspective, they can gain more revenue, if they serve the requests with large revenue first. This may lead to "starvation" of the requests with small revenue. The starved request should pay more to be in the system by increasing its revenue obtained by the substrate network provider.

Given the mapped nodes and links in the substrate network, migration plays a role in efficient resource usage with existing accepted requests by re-optimizing the substrate network resource utilization. Thus, this step can be interpreted as one that minimizes the cost, which is one of the objectives we discussed in II. Details on migration will be presented in Section V.

B. Customized Node Mapping

The first step is to find specific substrate nodes that the virtual nodes will be mapped to. We describe the node mapping algorithm for general requests and then discuss the customized versions for common kinds of VN topologies.

1) Node Mapping Algorithm for General Requests: For the general requests, we employ a "greedy" algorithm. This is because it is computationally hard to employ other strategies, such as iterative methods [10] and simulated annealing [9], [13]. The motivation of the greedy algorithm is to map the virtual nodes to the substrate nodes with maximum substrate resources so as to minimize the use of the resource at the bottleneck nodes/links. This is beneficial to future requests which should be mapped to nodes with scarce resources. Similar greedy algorithms are introduced in other related [8], [12] with difference objectives.

Greedy Node Mapping Algorithm for General Requests

1. Sort the requests according to their revenues.
2. Take one request with the largest revenue.
3. Find the subset S of substrate nodes that satisfy restrictions and available CPU capacity (larger than that specified by the request.)
4. For each virtual node, find the substrate node with the "maximum available resource" in S.
5. If fail in **3.** or **4.**, store this request at the request queue, and GOTO **2.**

In our algorithm, we collect all the requests in one time-window and those from the request queue, and then map all the virtual nodes in these requests to the substrate nodes. VN requests sometimes impose some *restrictions* on their nodes or links. The examples of node restrictions include geographical location and special functionality at the substrate node. These node restrictions are quite common in practice, e.g., servers near their customers in content-delivery service, programmable routers, and a node with Internet-2 network connectivity. Requests with restrictions give us some hints on how to map the virtual nodes. For example, location-specific requests usually tell us a geographical area that the substrate node should locate. This reduces the search space in the mapping process (**Step 3**).

Then, we keep track of the available node/link resources of the substrate network over time windows. The amount of available resource for a substrate node n^s is defined by:

$$CPU(n^s) \sum_{l^s \in L(n^s)} bw(l^s), \tag{1}$$

where $L(n^s)$ is the set of all adjacent substrate links of n^s, $CPU(n^s)$ is the remaining CPU resource of n^s, and $bw(l^s)$ is the unoccupied bandwidth resource for the substrate link l^s. With this definition, for a virtual node, we find the substrate node with the maximum available resource (**Step 4**). Note that we do not use $CPU(n^s)$ alone as the definition of available resource, since we have to ensure that the substrate node has enough adjacent bandwidth for the virtual network.

2) Customized Node Mapping Algorithm: In addition to the general node mapping algorithm, we provide a customized node mapping algorithm for requests with special topology. In many cases, the (logical) topologies of VN requests typically conform to some typical patterns. This is due to the fact that one of the popular applications of network virtualization is an overlay network that provides typical services, e.g., gaming and CDNs (Content Distribution Network). Popular topologies include tree, hub-and-spoke, and dual-hub-and-spoke. A tree topology is quite common in e.g., multicast distribution of IPTV, VoD, and video-casting, whereas hub-and-spoke topology is popular in enterprise networks and centralized databases/servers.

As a representative example, the following is the customized node mapping algorithm for requests with hub-and-spoke topology:

Customized Node Mapping Algorithm for Requests with Hub-and-spoke

Steps 1, 2, and 3: Same as in Greedy Node Mapping.
4. if the request has hub-and-spoke topology:
4.1 For each *hub* node, find the substrate node with the maximum available resource in S.
4.2 For each *spoke* node, find the shortest path between a substrate node in S and the substrate node mapped to the corresponding hub node.
else, apply **Step 4.** in Greedy Node Mapping Algo.
5. Same as in Greedy Node Mapping.

The main difference of the customized node mapping from the greedy node mapping is that the maximum available resource is allocated *only for the hub nodes* (**Step 4.1**). For the spoke nodes, we choose the substrate node that has the *shortest path* to the substrate node mapped to the hub node (**Step 4.2**). The motivation is due to the traffic volume difference between the hub and the spokes. By Step 4.2 we can also significantly *reduce the cost* for the embedding, since cost is generally proportional to the distance (i.e., number of hops), whereas the greedy algorithm allocates large substrate resource to "unimportant nodes", i.e., the spokes. This waste of resource will have the negative effect that the future requests may not find available resources for the greedy algorithm. We also note that the greedy algorithm does not allow us to perform a similar task, i.e., prioritization in node mapping depending on "importance" of the virtual node. This is because it is hard to know to which virtual node requires more resources for a general topology and to find the special pattern of link connectivity. The algorithm for hub-and-spoke topology is representative, so that similar techniques can be readily applied to popular, standard topologies such as tree, double hub-and-spoke, and backbone-star [10].

C. Link Mapping for Requests without Path Splitting

When the substrate nodes are selected for mapping, we map the virtual links to specific substrate links by customizing the algorithms for the request with or without path splitting. The algorithm for the request without path splitting is given by:

Link Mapping Algorithm for Requests without Path-splitting

1. Sort the requests by their revenues.
2. Take one request with the largest revenue.
3. For each virtual link of the request, we search the K-shortest paths for increasing K, and stop the search if we can find one.
4. If fail in **3.** for some virtual link, then reject this request, and store it in the request queue. GOTO **2.**

To find the optimal solution for requests with unsplttability is known to be NP-hard (see more discussions on computational complexity on link mapping in later section). Therefore, we use the *k-shortest path algorithm* to find link mapping for unsplittable flow(s) as an approximation approach. This algorithm can be solved in $O(N \log N + kN)$ time in a graph with N nodes [14]. We search K-shortest paths for increasing values of K, until we find a path which has enough bandwidth to map the corresponding virtual link.

IV. Link Mapping based on Path Splitting

In this section, we present our link-mapping algorithm for the virtual network requests that allow path splitting. First, we explain how path splitting simplifies the embedding problem and enables more efficient use of the substrate resources. Next, we describe how the substrate network can perform path splitting without disrupting the characteristics of the virtual link, and then finally, present our algorithm.

A. Path Splitting for Fast Computation and Greater Efficiency

The conventional approach to link mapping is to assign each virtual link to a *single* path in the substrate network. However, finding an optimal mapping from a virtual link to a single substrate path reduces to the Unsplittable Flow Problem (UFP), which is NP-hard [15], [16]. In addition, even the best single-path embedding of a virtual link may not make efficient use of substrate resources. To illustrate this, see Figure 2, after VN req. 1 has been accepted, a new VN req. 2 (with just a single link with a bandwidth requirement of 30) arrives. Since no path in the substrate has 30 units of available bandwidth, the request must be rejected, even though, collectively, the substrate has sufficient resources. In particular, the paths (D,E) and (D,G,H,I,E) have enough bandwidth, together, to support the new VN request.

If we make the substrate network *more supportive* by allowing *path splitting*, the link mapping problem becomes computationally tractable, and better resource utilization can be achieved. Splitting a virtual link l with capacity constraint C_l means that we map a virtual link into multiple paths in the substrate network, such that the sum of reserved end-to-end bandwidth along the multiple paths is equal to C_l. We refer to the percentage of traffic on each path as the *splitting ratio*. The link mapping problem with path splitting and flexible splitting ratio relaxes the constraints, and can be reduced to the multicommodity flow problem (MFP) [17], which is polynomial-time solvable. Moreover, path splitting enables better resource utilization by harnessing the small pieces of available bandwidths, allowing the substrate to accept more VN requests. For example, in Figure 2, we can accept the VN req. 2 by splitting virtual link (d, e) with the splitting ratios 2/3 and 1/3 on (D,E) and (D,G,H,I,E), respectively.

The benefits of having multiple paths have been established in other contexts, e.g., reliability and load balancing. In fact, even having just *two* paths, load balancing can significantly reduce the maximum load, compared to that with only a single path [18], [19]. Path splitting also helps to overcome failure. In case of node or link failures, we can quickly switch the traffic to the other paths simply by changing the splitting ratios. In contrast, in a single-path setting, a failure would require the substrate network to establish a completely new path for the virtual network, resulting in severe service disruption.

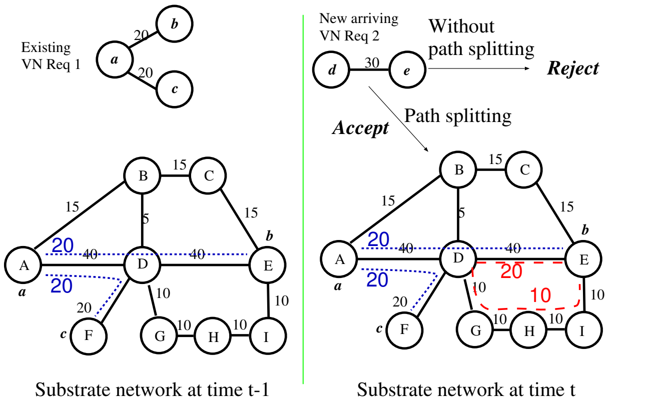


Fig. 2. Example of benefits of path splitting

B. Realizing Path Splitting in the Substrate Network

Path splitting can be easily implemented in the substrate network without significant overhead. When the virtual node directs a packet over the virtual link, the substrate sends the packet over one of the paths based on the target splitting ratio. The virtual network is largely oblivious to the splitting of the traffic, as long as care is taken to prevent out-of-order packet delivery. The substrate can employ a variety of techniques to prevent performance disruptions:

1) *Hash-based splitting.* Out-of-order delivery is primarily a concern for packets in the same *flow*—a group of packets between the same end hosts. Hash-based splitting prevents out-of-order delivery by directing all packets from the same flow to the same path. In its implementation, we first divide the hash space into weighted partitions that each correspond to one substrate path. Then, we apply hashing to the packets based on their header bits and forward the packets to the corresponding substrate path. We can also use consistent hashing to minimize disruptions when adapting the splitting ratios [20]. This hash-based scheme is efficient and, in fact, is widely used in IP networks to split traffic evenly over equal-cost multipath [21], [22].
2) *Adding artificial delay.* Another solution is to equalize the delays along the multiple paths. This is possible because all substrate nodes and links belong to a single party—the substrate provider. The substrate provider can add small artificial delay to overcome variable propagation delay, e.g., by using Dummynet [23]. Moreover, we do not need to be concerned about congestion-related delay, since unlike in a conventional best-effort network (e.g., the Internet), these virtual networks are allocated bandwidth resources in advance.
3) *Tagging the packets.* Since the substrate network is under the control of a single party, each packet can be tagged with a sequence number or timestamp. Then the remote end-point of the virtual link can reorder the packets based

on the tags before delivering the packets to the receiving virtual node.

Next, we will describe our algorithm that maps the virtual links with splittability into the substrate network.

C. Link Mapping Algorithm with Path Splitting

Link Mapping Algorithm for Requests with Path-splitting

MFP Calculation:

1. For all requests with splittability, construct linear constraints on the commodities for each virtual link.
2. Solve MFP (Multicommodity Flow problem).

Node Movement:

3. If infeasible, find the bottleneck link.
4. Map the virtual node at one end of the bottleneck link to another randomly-chosen substrate node, then GOTO **2.** with new linear constraints.
5. If infeasible for T times, eliminate the request making MPF infeasible, then construct linear constraints with the remaining requests, and GOTO **2.**

For simplicity, consider a request with only one link l^v with the capacity constraint C, where two end nodes of l^v are denoted by n_1^v and n_2^v. Denote by $\mathcal{M}^N(n_1^v) = n_1^s$ and $\mathcal{M}^N(n_2^v) = n_2^s$ the substrate nodes mapped to n_1^v and n_2^v, respectively, which are determined by the node mapping algorithm in section III. Finding multiple substrate paths is equivalent to finding a flow from the source n_1^s to the destination n_2^s in the substrate network with available capacity resource on the substrate links.

At the link mapping stage, all the virtual nodes have already been mapped to the substrate nodes. Therefore, we select those requests that allow path splitting and solve the MFP problem (**Steps 1 and 2**). Two end-nodes of a virtual link are mapped to two substrate nodes, which form a single commodity. Then, a group of requests considered in this time-window generates a group of, say r, commodities. The algorithm tries to find all the paths for r commodities based on the linear constraints:

$$\forall l^s \in L^s \sum_{i=1..r} f(c_i, l^s) \leq bw(l^s) \tag{2}$$

where $f(c_i, l^s)$ is the bandwidth on the substrate link l^s that we allocate to commodity c_i (or its corresponding virtual link).

Steps 3, 4, and 5 correspond to the node movement stage. Note that we cannot guarantee that a feasible solution is always found in MFP calculation stage, since there might be some "congestion" by a large number of virtual links on some substrate links with scarce bandwidth resources. To overcome such congestion, we pick one bottleneck link and try to get some virtual requests away from it by changing their node mapping. We are able to find the bottleneck link by investigating which linear constraints (for the commodities) are violated. Then, we randomly choose one end node of this bottleneck link, and map it to the substrate node with maximum remaining resources (defined in (1)). If such a dynamic node movement is not beneficial to solve the MFP for a pre-defined number of time windows, we reject and store the corresponding requests, and then try to solve the MFP with the remaining requests.

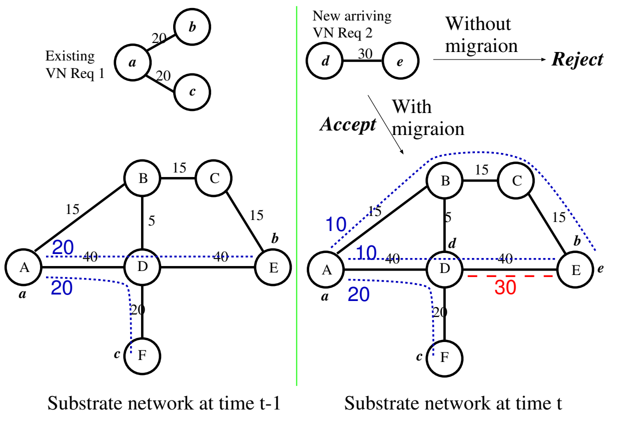


Fig. 3. Example of Benefits of Migration

Some virtual network might be sensitive even to the minor disruptions of flow splitting and thus do not allow path splitting. They have to use the link mapping algorithm for requests without path splitting in section III-C, resulting in inefficient substrate resource utilization.

V. Solving the Online Problem: Migration

In this section, to deal with the online VN embedding problem, we introduce the idea of path migration, i.e., changing the route or splitting ratio of a virtual link.

A. Benefits and Realization of Path Migration

Since VN requests dynamically arrive and depart over time, the "optimal" online embedding algorithm should be able to predict the future. One may want to use dynamic programming to solve this mathematically. However, developing an algorithm relying on dynamic programming is not practical in VN embedding, as we have discussed in Section II. Then, sub-optimal, heuristic algorithms are possible to let the system drifts into inefficient resource usage. This motivates us to find ways to "re-balance" the mapping of virtual networks to make more efficient use of the substrate resources to maximize the chance of accepting future requests. We also need a way to handle link or node failures in the substrate network, as well as dynamic changes of link capacity constraints.

We discuss the technical issues on migration by categorizing it into path migration and node migration.

Path migration. Path migration is composed of two kinds of methods: (i) tuning the splitting ratio (which is applied only for requests that allow path splitting), and (ii) migrating paths entirely or partially. By this path migration, we leave more bandwidth to accept new more requests. Path migration will not cause significant service disruptions for two reasons: In (i) we need just a slight change of flow splitting ratio for the already-existing paths; in (ii) migrating paths is similar to changing routes in the Internet. We can create the new path in advance before moving the traffic to avoid service disruption. Therefore, path splitting will not influence the application significantly.

Note that path migration is closely coupled with path splitting, especially in its method (i). Path splitting is an important component not only to reduce complexity for offline requests, but also to give more flexibility to the substrate network, leading to more efficient online embedding in migration. For example, in Figure 3, the new VN req. 2 comes with bandwidth requirement 30. Without migration, the VN req. 2 should be rejected due to scarcity of the network resource. However, with migration and if the VN req. 1 being served on the substrate allows path splitting, we can split the virtual link (a, b), and migrate its partial traffic to another path (A,B,C,E) to make enough bandwidth on (A,D,E) for the VN req. 2.

Node migration. In general, we avoid node migration since it may cause significant service disruption of ongoing virtual networks. However, node migration is also feasible as long as we plan it well, for the following reasons: First, long-running services usually have their own maintenance windows, where they drain traffic off a server to upgrade the software. These windows can be used for migration. Second, with ample warning and prior planning, we can minimize the negative effect of migration on ongoing service. Node migration can be done quite quickly in practice, e.g. within a few seconds [24].

B. Migration Algorithm

We mainly focus on VN requests with long duration in our migration algorithm. This duration can be specified by the requests when they arrive at the system, or inferred if a request has already been served for some time. Migrating short-duration requests is intuitively not cost effective, because it takes more migration efforts than the resource utilization benefits we gain. However, for the long-duration requests, during the time they are running, other requests arrive and depart. Therefore resource utilization can be inefficient. The strategy of giving priority to long requests in the embedding problem can supported by the research on job scheduling, e.g., [25], where migrating a long job away from a busy host helps not only the long job, but also the many short jobs that are expected to arrive at the host in the future.

In practice, VN requests are possibly quite diverse in their durations, ranging from a few months to several hours. As an example, content distribution networks such as Akamai [26] keep providing service once it starts, whereas the impromptu conference and gaming stay in the system for relatively short time.

In the migration algorithm, we fix the node mapping of both the requests in the current time window and those in previous time windows, which have been mapped to the substrate network. *We perform path migration by rerunning the link mapping step* in Section IV. For a pre-defined threshold time T, we only consider the requests which allow path splitting and whose duration is longer than T. Then we recalculate the MFP (Multicommodity Flow Problem) as described in our link mapping algorithm. If we have benefits for cost reduction by migration, then path migration is performed.

C. Time Window and Delay

As we have mentioned in Section III, we collect our requests within a *time window,* and then find the embedding solution for them. The size of this time window is not necessarily fixed, but can be flexible. It is determined by the number of waiting requests and the amount of available substrate resources. If time window is large, we can handle more requests together, leading to a better efficient solution. However, significantly long time window may cause the incoming requests to wait. For this reason, we allow *delay*, the time that each request is willing to wait. In our paper, we mainly focus on the time window with fixed length. Developing an algorithm which dynamically determines time window length together with investigating the relationships between migration, time window and delay is left as a future work.

VI. Performance Evaluation

A. Evaluation Environment and Performance Metric

We implemented a VN embedding simulator (publicly available at [27]) to evaluate our algorithm in terms of path splitting, migration, and customized node mapping.

Substrate network. We use GT-ITM tool [28] to generate the substrate network topology. The GT-ITM tool has been popularly used in the research which requires the practical network topology generation. The substrate network is configured to have 100 nodes, whose scale corresponds to a medium-scale ISP. From this setting, at every experiment, about 500 links are generated on average. The CPU resources at nodes and the link bandwidths at links follow a uniform distribution ranging from 0 to 100.

Virtual network request. In one VN request, the number of VN nodes is randomly determined following a uniform distribution between 2 and 10. We comment that the similar simulation environment is also adopted in the related work considering online problem [9]. Each pair of virtual nodes are randomly connected with probability 0.5. This means that for a n-node virtual network, we have $n(n-1)/4$ links. The arrivals of VN requests are modeled by a Poisson process with mean 5 reqs. per time window. The duration of the request follows an exponential distribution with 10 time windows on average. We run all of our simulations for 500 time windows, which corresponds to about 2500 requests on average in one instance of simulation. This is sufficient to believe our data is from the steady-state regime, since as presented in the next subsection, within 10 time windows, the system enters the steady state.

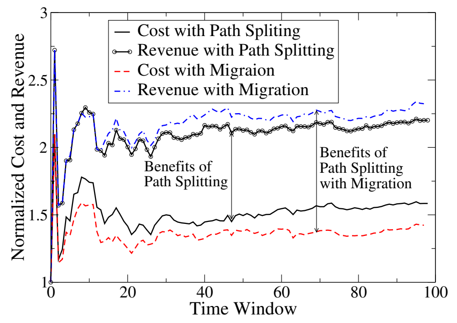
The parameters and their symbols that we vary in all our simulations are:

- E[cpu]: average CPU requirement on a virtual node,
- E[bw]: average bandwidth requirement on a virtual link,
- RPS-%: percentage of the requests with path splitting.
- DELAY: delay of a request.

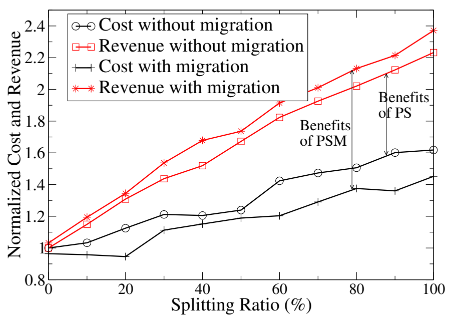
Performance metric. As we have mentioned, our algorithm is designed to jointly minimize cost and maximize revenue. Thus we plot both measures together in all our simulations. We use the following definition of cost (C) and revenue (R): for a VN request $G^v = (N^v, L^v)$,

$$R = \sum_{l \in L^v} \mathrm{bw}^v(l), \quad C = \sum_{p \in \mathcal{P}(G^v)} \mathrm{hops}(p) \times \mathrm{bw}^s(p, G^v), \tag{3}$$

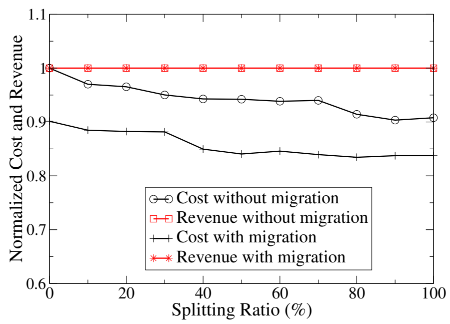
where $\mathrm{bw}^v(l)$ is the bandwidth requirement of the virtual link l, $\mathcal{P}(G^v)$ is the entire set of paths allocated for virtual links in G^v, $\mathrm{hops}(p)$ is the number of hops in a path p, and finally $bw^s(p, G^v)$ is the reserved bandwidth over a path p w.r.t the



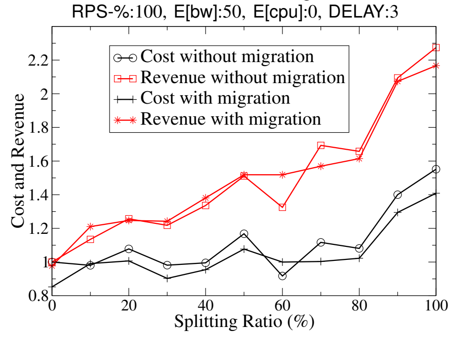
(a) Normalized cost/revenue changes with time. RPS-%:100, E[bw]:50, E[cpu]:0, DELAY:3



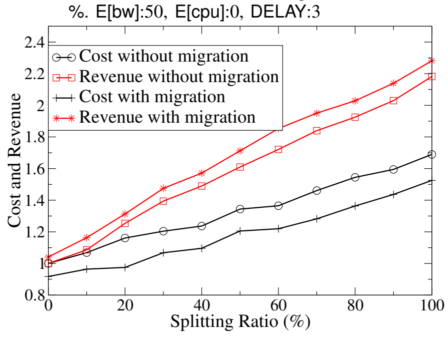
(b) Normalized cost/revenue changes with RPS-%. E[bw]:50, E[cpu]:0, DELAY:3



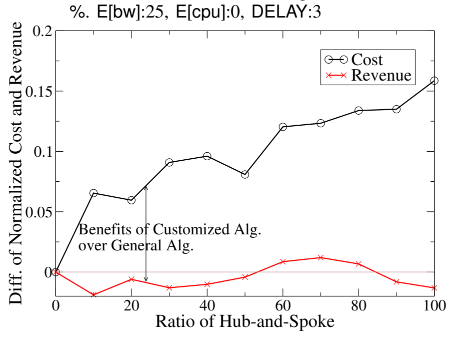
(c) Normalized cost/revenue changes with RPS-%. E[bw]:25, E[cpu]:0, DELAY:3



(d) Normalized cost/revenue changes with RPS-%. E[bw]:50, E[cpu]:25, DELAY:3



(e) Normalized cost/revenue changes with RPS-%. E[bw]:50, E[cpu]:0, DELAY:6



(f) Effect of modularized algorithm for hup-and-spoke topology, RPS-%:50, E[bw]:50, E[cpu]:0, DELAY:3

Fig. 4. Simulation results to show the benefits of Path-splitting, Migration, and Customization

request G^v. Note that the cardinality of $\mathcal{P}(G^v)$ may be larger than that of L^v due to path splitting.

Several remarks are in order: (i) Our definitions of cost and revenue do not reflect CPU resources, since bandwidth resources are in general more scarce than CPU resources; (ii) We do not consider the number of hops in the revenue definition, since it is not clear how this virtual network is mapped in the substrate network; (iii) In the cost definition, hops can be substituted by other metrics, e.g., physical distance.

In most of our plots, we compare costs and revenues of the substrate network with (i) No Path Splitting (NPS), (ii) only Path Splitting (PS), and (iii) both Path Splitting and Migration (PSM). In order to validate benefits of (ii) and (iii), *we plot the normalized costs and revenues* of PS and PSM by those of NPS, i.e.,

$$R_{norm}(PS) = \frac{R(PS)}{R(NPS)}, \quad C_{norm}(PS) = \frac{C(PS)}{C(NPS)}, \tag{4}$$

where the same normalization is applied to PSM. This normalization allows us to equalize the dimensions of cost and revenue with respect to those of NPS. Thus, as a benefit-measure, we define *the normalized pay-off* to be the gap between $R_{norm}(PS)$ and $C_{norm}(PS)$ (resp. $R_{norm}(PSM)$ and $C_{norm}(PSM)$) (see Figure 4 for the pictorial interpretation of these gaps). Then, the increasing normalized pay-off of either PS or PSM means the increasing benefits of PS or PSM over NPS.

B. Evaluation of Path-splitting and Migration

Figure 4 shows the entire set of simulation results, which show the benefits of path splitting, migration, and customized node mapping for the requests with hub-and-spoke topology. We interpret the simulation results by providing the following main observations:

(1) The benefits of path splitting increase with more requests allowing path-splitting, and the "benefit pattern" depends on the bandwidth requirement. All of the Figures 4(a), (b), (c), (d), and (e) uniformly show the increasing benefits of path splitting as the percentage of the requests allowing path splitting increases. Figure 4(a) shows the traces of normalized cost and revenue changes over time for a particular set of parameters: E[cpu]= 0, E[bw]= 50, RPS-%= 100, and DELAY= 3. Figures 4(b), (c), (d), and (e) shows the normalized cost and revenue (which is time-averaged) with increasing values of the percentage of the requests with path splitting.

One interesting fact that we note here is that the pattern of getting benefits depends on the stringency of link bandwidth requirement. Consider the experiments in Figures 4(b) and (c) with the different link bandwidth requirements. In Figure 4(c) with lower bandwidth requirements, most of the requests are accepted, therefore the revenues do not change significantly with RPS-%, and cost difference mainly contributes to the normalized pay-offs. Note that the normalized costs for PS and PSM are *smaller than* 1, which is in contrast to those in (b). This is because there are enough available bandwidth resources due to low aggregate bandwidth requirements. This leads to the fact that path splitting enables a virtual link to be mapped to multiple shorter paths, which is the main source of cost reduction.

(2) The benefits of migration also increase for more requests with path splitting, but, for stringent CPU requirements, mi-

gration does not have significant extra benefits compared with PS. Figures 4(a), (b), (c) and (e) show that the significant benefits of migration as $\mathsf{RPS}-\%$ increases. However, in Figure 4(d) with stringent CPU requirements, the benefits of migration over those of path splitting is modest. This is because we only employ path migration, which does not take effect when the node CPU resource is the bottleneck.

(3) Benefits of path splitting and migration are not significantly affected by delay. Figures 4(b) and (e) shows that our benefits of path splitting and migration are not influenced by delays, since when in the steady-state, the average number of requests that are served and depart is "stable."

C. Evaluation of Customized Node Mapping

To evaluate the performance of modularization, we compare greedy node mapping and our customized node mapping with the requests of hub-and-spoke topologies in Figure 4(f). In the plot, we first compute the revenues and the costs of both greedy and customized node mapping for different percentages of the requests with hub-and-spoke topology. Then, we normalize each result by that of the case of only general topology (This is similar to normalization by the case with no path-splitting in earlier figures). Through this procedure, we obtain the normalized revenues and costs for both algorithm, say, $R_{norm}^{greedy}, C_{norm}^{greedy}$, and $R_{norm}^{custom}, C_{norm}^{custom}$, and plots the data of $R_{norm}^{greedy} - R_{norm}^{custom}$ and $C_{norm}^{greedy} - C_{norm}^{custom}$ in the graph. The vertical gap between two points at each x-axis represents the extra normalized pay-offs of the customized node mapping over the general mapping. Therefore, in Figure 4(f), the increasing gap between the revenue and the cost tell us the increasing benefits of our customized algorithm as we have more requests with hub-and-spoke topology.

VII. Conclusion

A key problem in the current study of network virtualization, the VN embedding problem, has various constraints and objectives that make it computationally intractable. In this paper, rather than significantly restrict the problem space to make the problem tractable, or propose heuristic algorithms, we rethink the VN embedding problem by proposing a more flexible substrate network to better support virtual network embedding. This flexibility includes path splitting and migration. From both the theoretical and engineering perspective, we show that allowing substrate path splitting and migration would help us to attain better resource utilization. We also propose a modularized algorithm framework with customized node mapping algorithms dealing with common classes of virtual-network topologies. Through our publicly available simulator, we demonstrate the benefits of these approaches in making the embedding problem computationally easier, and the resulting embeddings more efficient. We are currently investigating how to combine the strategies of migration, time window and delay to further improve the handling of online requests.

References

[1] T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the Internet impasse through virtualization," *IEEE Computer Magazine*, vol. 38, no. 4, pp. 34–41, 2005.

[2] "GENI," http://www.geni.net/.

[3] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, "In VINI veritas: Realistic and controlled network experimentation," in *Proc. ACM SIGCOMM*, Pisa, Italy, September 2006.

[4] N. Feamster, L. Gao, and J. Rexford, "How to lease the Internet in your spare time," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 1, pp. 61–64, 2007.

[5] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan, and J. E. van der Merwe, "Resource management with hoses: Point-to-cloud services for virtual private networks," *IEEE/ACM Trans. Netw.*, vol. 10, no. 5, pp. 679–692, 2002.

[6] A. Gupta, J. Kleinberg, A. Kumar, R. Rastogi, and B. Yener, "Provisioning a virtual private network: A network design problem for multicommodity flow," in *Proc. ACM Symposium on Theory of Computing*. New York, NY, USA: ACM Press, 2001, pp. 389–398.

[7] S. Raghunath, K. K. Ramakrishnan, S. Kalyanaraman, and C. Chase, "Measurement based characterization and provisioning of IP VPNs," in *Proc. Internet Measurement Conference*. New York, NY, USA: ACM Press, 2004, pp. 342–355.

[8] Y. Zhu and M. Ammar, "Algorithms for assigning substrate network resources to virtual network components," in *Proc. IEEE INFOCOM*, 2006.

[9] J. Fan and M. Ammar, "Dynamic topology configuration in service overlay networks: A study of reconfiguration policies," in *Proc. IEEE INFOCOM*, 2006.

[10] J. Lu and J. Turner, "Efficient mapping of virtual networks onto a shared substrate," Washington University, Technical Report WUCSE-2006-35, 2006.

[11] D. G. Andersen, "Theoretical approaches to node assignment," Unpublished Manuscript, 2002.

[12] R. Ricci, C. Alfeld, and J. Lepreau, "A solver for the network testbed mapping problem," *ACM Computer Communication Review*, vol. 33, no. 2, pp. 65–81, 2003.

[13] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science, Number 4598, 13 May 1983*, vol. 220, 4598, pp. 671–680, 1983.

[14] D. Eppstein, "Finding the k shortest paths," in *Proc. IEEE Symposium on Foundations of Computer Science*, 1994, pp. 154–165.

[15] J. Kleinberg, "Approximation algorithms for disjoint paths problems," Ph.D. dissertation, MIT, 1996.

[16] S. G. Kolliopoulos and C. Stein, "Improved approximation algorithms for unsplittable flow problems," in *IEEE Symposium on Foundations of Computer Science*, 1997, pp. 426–435.

[17] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[18] M. Mitzenmacher, "The power of two choices in randomized load balancing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 10, pp. 1094–1104, 2001.

[19] P. Key, L. Massoulie, and D. Towsley, "Path selection and multipath congestion control," in *Proc. IEEE INFOCOM*, 2007.

[20] I. Avramopoulos, D. Syrivelis, J. Rexford, and S. Lalis, "Secure availability monitoring using stealth probes," Princeton University, Technical Report TR-769-06, October 2006.

[21] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, and J. Rexford, "Netscope: traffic engineering for IP networks," *IEEE Network Magazine*, vol. 14, no. 2, pp. 11–19, March 2000.

[22] B. Augustin, X. Cuvellier, B. Orgogozo, F. Viger, T. Friedman, M. Latapy, C. Magnien, and R. Teixeira, "Avoiding traceroute anomalies with paris traceroute," in *Proc. Internet Measurement Conference*. New York, NY, USA: ACM Press, 2006, pp. 153–158.

[23] L. Rizzo, "Dummynet: a simple approach to the evaluation of network protocols," *ACM Computer Communication Review*, vol. 27, no. 1, pp. 31–41, 1997.

[24] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and gray-box strategies for virtual machine migration," in *Proc. Networked Systems Design and Implementation*, Cambridge, MA, April 2007.

[25] M. Harchol-Balter and A. B. Downey, "Exploiting process lifetime distributions for dynamic load balancing," *ACM Transactions on Computer Systems*, vol. 15, no. 3, pp. 253–285, 1997.

[26] "Akamai content distribution network," http://www.akamai.com/.

[27] "Virtual Network Embedding Simulator," http://www.cs.princeton.edu/~minlanyu/embed.tar.gz.

[28] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *Proc. IEEE INFOCOM*, 1996.