Virtual Network Embedding with Substrate Support for Parallelization

Sheng Zhang[†], Jie Wu[§], and Sanglu Lu[†]
[†]State Key Lab. for Novel Software Technology, Nanjing University, China
[§]Department of Computer and Information Sciences, Temple University, USA
[†]zhangsheng@dislab.nju.edu.cn, sanglu@nju.edu.cn, [§]jiewu@temple.edu

***Abstract*—*Network virtualization* has been the focus of intense research interest and is a promising approach to overcome the ossification of the Internet. A major challenge with network virtualization is *virtual network embedding*, which deals with the efficient embedding of virtual networks with resource constraints into a substrate network. Many research results have been reported regarding this problem. However, there hasn't been any focus on virtual network embedding with *substrate support for parallelization*, i.e., the substrate network supports parallel computation and allows a virtual node to be mapped into multiple substrate nodes. This paper is the first attempt at gaining a better understanding on how parallelization benefits embedding. We present a formal problem description and propose two algorithms that capitalize parallelism. Several extensions are developed to complement the proposed algorithms. From experimental results, the effectiveness and usefulness of the algorithms and extensions are confirmed.**

I. Introduction

From global commerce to communications, from national defense to entertainment–just as it seems, the Internet has been extremely successful and ubiquitous. However, the multi-provider nature of the Internet and the end-to-end design of Internet Protocol (IP) are now creating hurdles for its further evolution [1, 2]. Recently, *network virtualization* has received much attention as a promising approach to overcome the current ossification of the Internet [3, 4], and it has been investigated in a variety of prestigious projects, such as CABO [4], PlanetLab [5], and VINI [6].

In a network virtualization environment, an *infrastructure provider* (InP) maintains a *substrate network* (SN), which is composed of substrate nodes and links, while a *service provider* (SP) purchases slices of substrate resources (e.g., CPU, bandwidth, memory) from the InP and then creates his own customized *virtual network* (VN) to offer value-added service (e.g., Voice over IP, content distribution) to end users. In doing so, customized network protocols can be easily deployed without requiring universal agreements between competing stakeholders. Furthermore, this decoupling of traditional Internet service providers brings about a layered service architecture and provides great flexibility and diversity.

The fundamental challenge that network virtualization faces is efficiently utilizing substrate resources, that is, how to embed multiple virtual networks with resource constraints into a substrate network so as to efficiently utilize the substrate resource. Known as the *Virtual Network Embedding* (VNE) problem, it is proven to be NP-complete by reducing the *multiway separator problem* to this problem [7].

A significant amount of research [8–20] has investigated techniques for the VNE problem. Some work [9–11] restricted the problem space. Multi-path routing and dynamic migration support was envisioned in [12]. Attention was paid to opportunistic resource sharing in [19, 20]. Some other research [14, 16, 20] focused on topology-aware embedding.

However, there hasn't been any focus on virtual network embedding with substrate support for *parallelization*. More specifically, we make the substrate network more supportive of embedding, in the sense that the substrate network supports parallel computation and allows a virtual node to be mapped to *multiple* substrate nodes. Upon that, multiple substrate nodes can parallel accomplish the computation that the virtual node is dedicated to. Parallelization not only enables the substrate network to efficiently share its resources among virtual networks, but also makes virtual networks more reliable, as computation can quickly migrate to other substrate nodes in case a substrate node crashes. Since network virtualization is still in its infancy [12], we believe it is important to explore how we can design the SN to best serve its goals.

In this paper, we study the virtual network embedding problem with parallelization support and propose two algorithms, *ProactiveP* and *LazyP*, that intelligently perform efficient embedding. The contributions of this paper are threefold.

1) To the best of our knowledge, this is the first attempt that envisions substrate parallelization support in virtual network embedding. We present a formal description of the problem and propose two embedding algorithms that leverage parallelization.
2) We provide three extensions to further complement our algorithms. The first deals with the additional CPU and bandwidth consumption due to parallelization and communication; the second one considers situations where the maximum possible *speedup* [21] is specified by service providers, and the last one discusses expiration time.
3) Simulation results demonstrate that the proposed algorithms achieve a remarkably high acceptance ratio over time. In addition, the impacts of various extensions are also investigated in our simulations.

The remainder of this paper is organized as follows. Section II introduces the notations and problem formulation. Then, in Section III, we propose two algorithms. We also

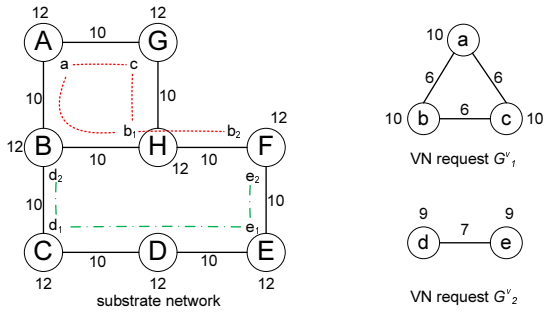


Fig. 1. An illustration of virtual network embedding with substrate support for parallelization. For virtual network G^v_1, the master mapping is $\{a \to A, c \to G, b \to H\}$, the slave mapping is $\{a \to \emptyset, c \to \emptyset, b \to \{F\}\}$, and the link mapping is $\{(ac) \to \{AG\}, (cb) \to \{GH\}, (ba) \to \{HB, BA\}\}$. $Ratio(b) = \{0.6, 0.4\}$. For virtual network G^v_2, the master mapping is $\{d \to C, e \to E\}$, the slave mapping is $\{d \to \{B\}, e \to \{F\}\}$, and the link mapping is $\{(de) \to \{CD, DE\}\}$. $Ratio(d) = \{2/3, 1/3\}$, and $Ratio(e) = \{2/3, 1/3\}$.

provide three extensions in Section IV. Simulation results are presented in Section V, where the performance of our algorithms are compared with respect to acceptance ratio and node/link utilization. We go over some existing related work in Section VI. Finally, Section VII concludes the paper.

II. Preliminary

In this section, we first introduce the notations used in this paper, and then we present the problem formulation.

A. Notations

The main constraints we consider in this paper are CPU and bandwidth, which is the typical case in almost all of the related literature on virtual network embedding so far. The notation system in this paper is similar to that in [10, 12, 20]. The principle behind these notations is that superscript "s" (or "v") indicates a substrate (or virtual) network. Commonly, we model a substrate (or virtual) network as a undirected weighted graph; vertices represent nodes and edges represent links. Each vertex is associated with a CPU capacity/constraint, while each edge is associated with a bandwidth capacity/constraint.

***Definition** 1: (Substrate Network).* A substrate network is $G^s = (N^s, E^s, C^s, B^s)$, where N^s and E^s are the sets of substrate nodes and links, respectively. C^s is the set of CPU attributes, and B^s is the set of bandwidth attributes.

Let $RC^s(n^s)$ and $RB^s(e^s)$ be the residual CPU of substrate node n^s and the residual bandwidth of substrate link e^s, respectively. Let $P^s(n^s_i, n^s_j)$ denote the set of loop-free paths between endpoints n^s_i and n^s_j in G^s. Similarly, we have:

***Definition** 2: (Virtual Network).* A virtual network is $G^v = (N^v, E^v, C^v, B^v)$, where N^v is the set of virtual nodes, and E^v is the set of virtual links. C^v is the set of CPU constraints, and B^v is the set of bandwidth constraints.

Fig. 1 shows an example of these notations. the corresponding number near each vertex (or edge) is the CPU (or bandwidth) capacity/constraint.

Traditional virtual network embedding is defined as a mapping of G^v to a subset of G^s such that each virtual node is mapped onto *exactly one* substrate node and each virtual link is mapped onto a substrate path between its corresponding endpoints. However, with substrate support for parallelization, one virtual node can be mapped onto *multiple* substrate nodes. It is then reasonable to assume that these multiple substrate nodes, which one virtual node is mapped onto, should be close to each other so as to mitigate the effect of network latency. In this paper, we represent this "closeness" by forcing these multiple substrate nodes to form a star topology, i.e., all *slave* nodes are one-hop away from the *master* node. To put it formally:

***Definition** 3: (Virtual Network Embedding, parallelization version).* Virtual network embedding from G^v to a subset of G^s is composed of three components: *master mapping* $\mathcal{M}_{ms}$, *slave mapping* $\mathcal{M}_{sl}$, and *link mapping* $\mathcal{M}_l$.

***Definition** 4: (Master Mapping and Slave Mapping).* The master mapping $\mathcal{M}_{ms}$ maps a virtual node n^v to a substrate node, denoted by $\mathcal{M}_{ms}(n^v)$, and the slave mapping $\mathcal{M}_{sl}$ maps a virtual node n^v to a subset of the neighbors of $\mathcal{M}_{ms}(n^v)$, denoted by $\mathcal{M}_{sl}(n^v)$, such that $\forall n^v, m^v \in N^v$: (i) $\mathcal{M}_{ms}(n^v) \cup \mathcal{M}_{sl}(n^v) = \mathcal{M}_{ms}(m^v) \cup \mathcal{M}_{sl}(m^v)$ iff. $m^v = n^v$, and (ii) $RC^s(\mathcal{M}_{ms}(n^v)) + RC^s(\mathcal{M}_{sl}(n^v)) \geq C^v(n^v)$.

***Definition** 5: (Link Mapping).* The link mapping $\mathcal{M}_l$ maps a virtual link $e^v = (n^v, m^v)$ to a substrate loop-free path, denoted by $\mathcal{M}_l(n^v, m^v)$, such that $\forall e^v \in E^v_i$: (i) $\mathcal{M}_l(n^v, m^v) \in P^S(\mathcal{M}_{ms}(n^v), \mathcal{M}_{ms}(m^v))$, and (ii) $RB^s(\mathcal{M}_l(n^v, m^v)) \geq B^v(e^v)$.

We also associate each virtual node with a vector $Ratio(n^v)$ to indicate how to distribute $C^v(n^v)$ among the master and slave nodes. Fig. 1 shows two embedding examples. "$Ratio(b) = \{0.6, 0.4\}$" indicates that node b will occupy 6 units of CPU in H and 4 units of CPU in F.

It is worth mentioning that parallelization brings about additional CPU and bandwidth costs. To concentrate on the main problem, we defer the discussion on this issue to Section IV-A.

B. Problem Formulation

Virtual network embedding requests that are submitted by SPs arrive and depart over time. When a virtual network is successfully embedded, its owner, i.e., an SP, has to pay rent proportional to his requested resources. Thus, the revenue, $\mathcal{R}(G^v)$, of embedding a virtual network can be defined as (following the definitions in previous work [10, 12, 20]):

$$\mathcal{R}(G^v) = \omega_c \cdot \sum_{n^v \in N^v} C^v(n^v) + \omega_b \cdot \sum_{e^v \in E^v} B^v(e^v) \qquad (1)$$

where ω_c and ω_b are the weights for CPU and bandwidth, respectively. From the InP's point of view, a natural objective is to increase his revenue. Since the amount of resources owned by the InP is limited in a relatively long period of time, the InP should optimize the embedding of virtual networks to maximize its revenue.

To this end, virtual networks should be properly and efficiently deployed on top of a substrate network, for which many algorithms have been proposed. However, little attention has been given to embedding with parallelization support. In the next section, we will present two algorithms, *ProactiveP* and *LazyP*, for the parallelization version of embedding.

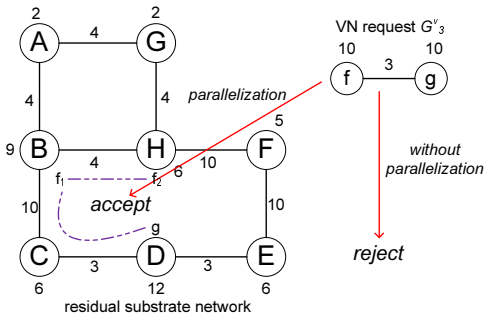


Fig. 2. Motivation of parallelization. The left part shows the residual substrate network of Fig. 1 after embedding two VN requests, G^v_1 and G^v_2.

III. Two algorithms: ProactiveP and LazyP

In this section, we first argue that the substrate network should support parallelization, and then we present two algorithms, *ProactiveP* and *LazyP*.

A. Motivation of Parallelization

To motivate substrate support for parallelization, consider the example shown in Fig. 2, where the left part shows the residual substrate network, which has already accepted two virtual network requests, G^v_1 and G^v_2. Now, suppose a new VN request G^v_3 arrives. If parallelization is not supported, this request would be rejected, as there is only one substrate node that has more than 10 units of available CPU resources. However, the request can be accepted in the presence of parallelization, as Fig. 2 shows.

Parallelization makes it possible for a substrate network to utilize every small piece of available resource, so as to accept more VN requests. Furthermore, it provides reliability and fast recovery. For example, when there is one master (or slave) node that crashes, the SN can direct the computation to other slave nodes simply by changing the *Ratio*.

Although the model used in this paper is simple, we wish to provide some insights on the design of better substrate infrastructures. In the following subsections, we introduce two heuristic embedding algorithms. One is the Proactive Parallelization (*ProactiveP*) algorithm, which divides computation among neighbors of the master node regardless of the residual resources of the master node, while the other is the Lazy Parallelization (*LazyP*) algorithm, which only distributes computation among the neighbors when there are not enough resources in the master node.

B. ProactiveP

ProactiveP employs a greedy approach to deal with master mapping, which plays the leading role in VN embedding. The slave nodes are chosen from the neighbors of each master node, and the link mapping utilizes Dijkstra [22] to find the shortest path that meets the requirements. *ProactiveP* is shown in Alg. 1; it consists of four phases.

In the initialization phase (lines 3-4), every substrate node is set to be unused, and every node updates $RC^s_{nei}(n^s)$, denoting the summation of the residual resources of the neighbors (including n^s itself) of n^s. In the master mapping phase (lines 5-12), all virtual nodes are sorted and placed in a queue in the decreasing order of $C^v(n^v)$; we then map each virtual node to the unused substrate node with the largest $RC^s_{nei}(n^s)$. If there is not enough resources (line 9-10), the embedding request is rejected; otherwise, this substrate node is marked to be used. In the slave mapping phase (lines 13-20), no matter how many residual units of CPU that the master node has, *ProactiveP* divides the CPU requirement into pieces that are proportional to the residual units of CPU in neighbors of the master node. In the link mapping phase (lines 22-28), each virtual link is mapped to the shortest substrate path that satisfies the bandwidth requirement between the corresponding endpoints. If there is no such path, this request is rejected.

Algorithm 1 Proactive Parallelization (*ProactiveP*)

1: **while** true **do**
2: //initialization phase
3: $\forall n^s \in N^s$, $unused(n^s) \leftarrow 1$ and update $RC^s_{nei}(n^s)$
4: wait until a VN request, say G^v, arrives
5: //master mapping phase
6: $Q \leftarrow$ sorted virtual nodes in decreasing $C^v(n^v)$
7: **for** $i = 1$ to $Q^s.length$ **do**
8: $\mathcal{M}_{ms}(Q[i]) \leftarrow argmax(RC^s_{nei}(n^s) \cdot unused(n^s))$
9: **if** $RC^s_{nei}(\mathcal{M}_{ms}(Q[i])) < C^v(Q[i])$ **then**
10: Rejection occurs
11: **end if**
12: $unused(\mathcal{M}_{ms}(Q[i])) \leftarrow 0$
13: //slave mapping phase
14: $Ratio(Q[i])[\mathcal{M}_{ms}(Q[i])] \leftarrow \frac{RC^s(\mathcal{M}_{ms}(Q[i]))}{RC^s_{nei}(\mathcal{M}_{ms}(Q[i]))}$
15: **for all** m^s is a neighbor of $\mathcal{M}_{ms}(Q[i])$ **do**
16: $unused(m^s) \leftarrow 0$
17: $\mathcal{M}_{sl}(Q[i]) \leftarrow \mathcal{M}_{sl}(Q[i]) \cup \{m^s\}$
18: $Ratio(Q[i])[m^s] \leftarrow \frac{RC^s(m^s)}{RC^s_{nei}(\mathcal{M}_{ms}(Q[i]))}$
19: **end for**
20: **end for**
21: //link mapping phase
22: **for all** $e^v = (n^v, m^v) \in E^v$ **do**
23: $P^{s\prime} \leftarrow \{p | p \in P^s(\mathcal{M}_{ms}(n^v), \mathcal{M}_{ms}(m^v)), RB^s(p) \geq B^v(e^v)\}$
24: **if** $P^{s\prime} == \emptyset$ **then**
25: Rejection occurs
26: **end if**
27: $\mathcal{M}_l(e^v) \leftarrow argmin_{p \in P^{s\prime}}(hop(p))$ (Dijkstra)
28: **end for**
29: **end while**

C. LazyP

LazyP shares most parts with *ProactiveP*, except the slave mapping phase, shown in Alg. 2. As the algorithm name indicates, *LazyP* applies parallelization only when the residual units of CPU in the master node for a virtual node is not sufficient. More specifically, when there is a need for parallelization (lines 3-9), *LazyP* iteratively chooses the unused node that has the most residual units of CPU among the

neighbors of the master node for virtual node $Q[i]$ (line 4), and tries to satisfy the CPU demand of $Q[i]$ (line 7) until $Q[i]$ is successfully embedded.

Algorithm 2 The slave mapping phase of *LazyP*

1: $Ratio(Q[i])[\mathcal{M}_{ms}(Q[i])] \leftarrow \frac{RC^s(\mathcal{M}_{ms}(Q[i]))<C^v(Q[i])?RC^s(\mathcal{M}_{ms}(Q[i])):C^v(Q[i])}{C^v(Q[i])}$
2: $remainder \leftarrow C^v(Q[i]) - RC^s(\mathcal{M}_{ms}(Q[i]))$
3: **while** $remainder > 0$ **do**
4: $g^s \leftarrow argmax(RC^s(m^s) \cdot unused(m^s))$, where m^s is a neighbor of $\mathcal{M}_{ms}(Q[i])$
5: $unused(g^s) \leftarrow 0$
6: $\mathcal{M}_{sl}(Q[i]) \leftarrow \mathcal{M}_{sl}(Q[i]) \cup \{g^s\}$
7: $Ratio(Q[i])[g^s] \leftarrow \frac{RC^s(g^s)<remainder?RC^s(g^s):remainder}{C^v(Q[i])}$
8: $remainder \leftarrow remainder - RC^s(\mathcal{M}_{ms}(Q[i]))$
9: **end while**

IV. Extensions

This section presents three extensions to complement the proposed embedding algorithms.

A. Additional CPU and Bandwidth Consumption

Ideally, parallelization causes no additional CPU and bandwidth consumption but meanwhile achieves the same effect as the scenario without parallelization. For example, in Fig. 1, virtual node b in G_1^v requires 10 units of CPU, and it is mapped to H and F. In an ideal case, the sum of units of CPU allocated in H and F is still 10, and there is no additional bandwidth consumption along HF. However, in fact, splitting computation, merging results, exchanging messages, and other issues will cost additional CPU and bandwidth resources. To capture this, we propose the following techniques.

For additional CPU consumption, we define the *penalty factor* pf: we allocate pf times as many CPU units as a virtual node demands. In the aforementioned example, if $pf = 1.2$, then 12 units of CPU should be allocated to b in H and F. For additional bandwidth consumption, we assume that it is a constant, say Z. That is, an additional Z units of bandwidth should be allocated between each pair of slave and master nodes. By using the simulations in Section V, we will show how pf and Z affect the performance of *ProactiveP* and *LazyP*, respectively.

B. The Maximum Possible Speedup

We would regret not mentioning that, it is the very computation determined by a virtual network, not our proposed embedding algorithms, that is parallelized. The *speedup* [21] refers to how much a parallel algorithm is faster than a sequential algorithm. Suppose that the maximum possible speedup of the computation in a virtual network is 5, that is, even if we allocate more than five substrate nodes for a virtual node in this virtual network, we still cannot achieve a shorter computation time than just allocating five nodes for that virtual node, which means that it is wasteful, with regard to substrate resources, to allocate more than five substrate nodes for any virtual node in this virtual network. In the proposed algorithms, if a service provider specifies the speedup to be $speedup$, then we just restrict the allowable number of slave nodes to $speedup-1$. Later, we will see the impact of $speedup$.

C. Expiration Time

VN requests arrive and leave over time. If our embedding algorithms immediately reject a VN request due to the shortage of available resources, then the InP definitely loses a customer. Therefore, to further improve the embedding performance and provide flexibility, we adopt a similar approach to [12]: for each VN request, its owner can specify an *expiration time* to indicate the amount of time that the embedding request is willing to wait. The embedding algorithm will try to process requests that do not expire at regular intervals.

V. Simulation

In this section, we evaluate the performance of our algorithms and see the impacts of the extensions.

A. Simulation Setup

As network virtualization is still an open field, the settings in this paper are similar to [10, 12–14, 20]. In order to see the impact of the substrate network topology on the performance of the proposed algorithms, we use two graph models.

ArpaNet [23] graph: it contains 20 nodes and 32 links (we insert additional links to increase its connectivity).

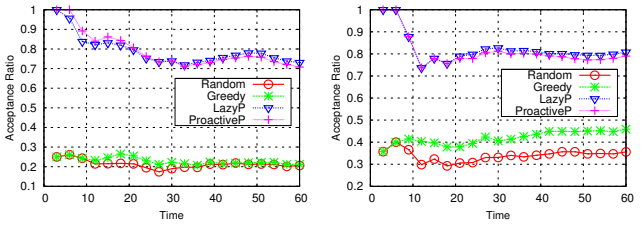
Erdős-Rényi model $G(n, p)$: it contains n nodes, and each pair of nodes is connected with probability p. In our simulations, we set $n = 20$ and $p = 0.4$ to be compatible with $ArpaNet$.

In each graph model, the CPU capacity at substrate nodes and the bandwidth capacity at substrate links are generated randomly from the range $[50, 100]$. The number of virtual nodes in each virtual network follows a uniform distribution between 2 and 10. Each pair of virtual nodes is connected with probability 0.5. The lifetime of each virtual network follows an exponential distribution with an average of 10 minutes. The arrivals of embedding requests are modeled as a Poisson process with an average rate of five requests per minute.

We use the following performance metrics for comparison. (i) *acceptance ratio*, which is the ratio of the number of accepted virtual network requests to all requests; (ii) *node utilization*, which is the ratio of the amount of occupied CPU resources to overall CPU resources in the substrate network; (iii) *link utilization*, which is the ratio of the amount of occupied bandwidth resources to overall bandwidth resources in the substrate network.

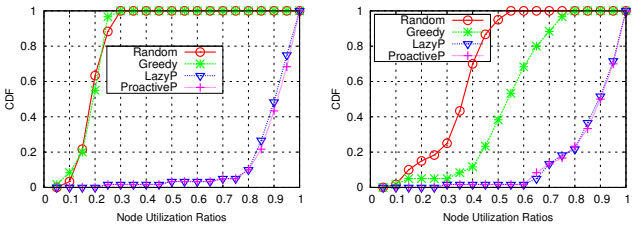
B. Simulation Results: Comparison of Algorithms

In this subsection, we do not consider the extensions in Section IV and focus on the comparison of the following algorithms. (i) ***Random*** (without parallelization): it randomly chooses a substrate node for each virtual node, and then employs Dijkstra to find a substrate path for each virtual link; (ii) ***Greedy*** (without parallelization): it greedily chooses an



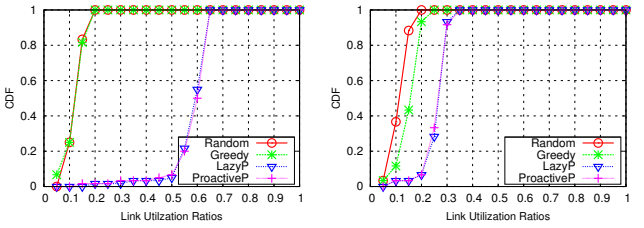
(a) On *ArpaNet* graph (b) On *Erdős-Rényi model* $G(20, 0.4)$

Fig. 3. Acceptance Ratio



(a) On *ArpaNet* graph (b) On *Erdős-Rényi model* $G(20, 0.4)$

Fig. 4. CDF of node utilization ratio



(a) On *ArpaNet* graph (b) On *Erdős-Rényi model* $G(20, 0.4)$

Fig. 5. CDF of link utilization ratio

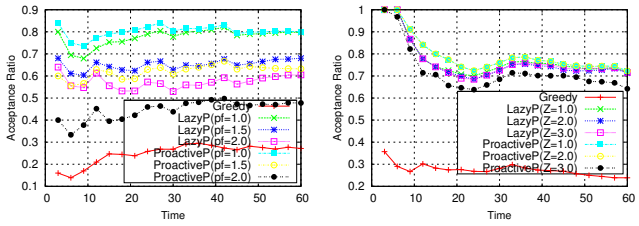
unused substrate node with the most available units of CPU for each virtual node in a sorted queue, and then does the same as *Random*; (iii) ***LazyP***; and (iv) ***ProactiveP***.

Fig. 3 shows the comparison of acceptance ratio between them. In general, *LazyP* and *ProactiveP* achieve a much higher acceptance ratio than *Random* and *Greedy* on both of the two graph models. We notice that every algorithm performs better on $G(20, 0.4)$ than on *ArpaNet*. The main reason is that $G(20, 0.4)$ has more links than *ArpaNet* on average.

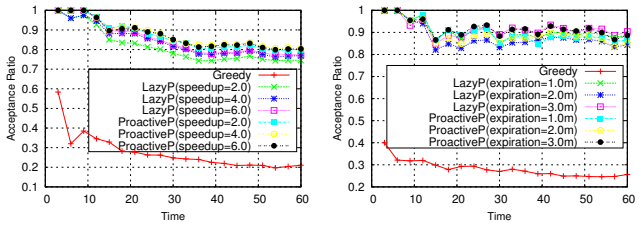
Figs. 4 and 5 illustrate the comparison of cumulative distribution function (CDF) of the node and link utilization ratios, respectively. We note that, (i) *LazyP* and *ProactiveP* allocate more resources than the others, which is in accordance with Fig. 3; (ii) since $G(20, 0.4)$ has more resources than *ArpaNet*, the node/link utilization ratio of the former is smaller than that of the latter on average; and (iii) the link utilization ratio is smaller than node on average. The reason could be that there are relatively abundant links connecting nodes.

C. Simulation Results: Effects of Extensions

We are also interested in evaluating the impacts of the penalty factor pf, additional bandwidth consumption Z, maximum



(a) The impact of *penalty factor* (b) The impact of Z



(c) The impact of *speedup* (d) The impact of *expiration time*

Fig. 6. Impact evaluation on *ArpaNet* graph

possible *speedup*, and *expiration* time. The default setting for them is $pf = 1, Z = 1, speedup = 20, expiration = 0$. For each of the four parameters, we ran experiments with one parameter of varying values while keeping the other three parameters to their default values. Figs. 6 and 7 show the numerical results on two substrate models, respectively. Some key observations are summarized as follows.

(1) The penalty factor has a greater influence on ProactiveP than LazyP. In Figs. 6a and 7a, the curves related to *LazyP* are closer to each other than those of *ProactiveP*. The main reason may be that *ProactiveP* trends to divide the CPU requirement into as many pieces as possible and makes substrate nodes more load-balanced. This is obviously not beneficial to embedding requests with large penalty factors.

(2) A larger Z makes algorithms perform worse in G(20,0.4) compared to in ArpaNet. This can be noted in Figs. 6b and 7b. As $G(20, 0.4)$ has more links than *ArpaNet*, a larger Z consumes more bandwidth resources.

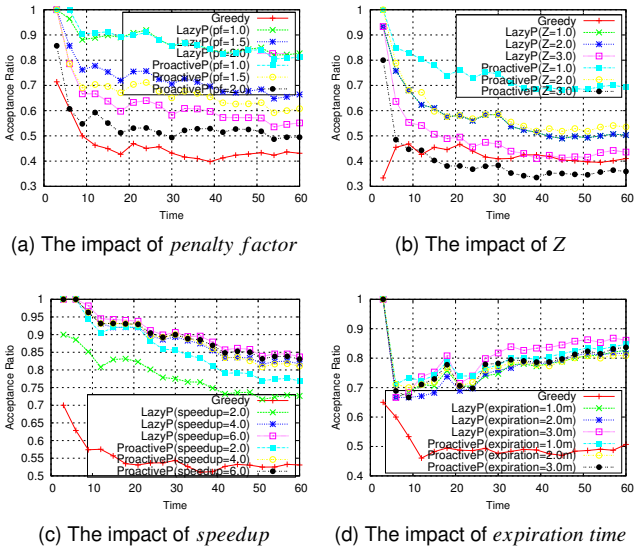
(3) Parameter speedup doesn't significantly affect LazyP and ProactiveP on both graph models. Figs. 6c and 7c show the results of $speedup = 2, 4$, and 6. Even when *speedup* is restricted to 2, *LazyP* and *ProactiveP* still perform better than the other algorithms to a great extent.

(4) Retry does improve the acceptance ratio. Figs. 6d and 7d show that the proposed algorithms with retry opportunities enable the substrate to accept more embedding requests.

VI. Related Work

A large number of mapping algorithms for virtual networks have been proposed in the past [8–20]. These algorithms gave good inspiration to the design of our algorithm.

Simulated annealing was introduced to cope with the NP-completeness of the VNE problem in [8, 18]. There, a candidate mapping solution is generated randomly, and the algorithms improve the candidate solution through iterative



(a) The impact of *penalty factor* (b) The impact of Z

(c) The impact of *speedup* (d) The impact of *expiration time*

Fig. 7. Impact evaluation on *Erdős-Rényi model* $G(20, 0.4)$

adjustments. In [8], only the bandwidth constraint was considered, and a substrate node was not allowed to be mapped to more than one virtual node. The tradeoff between results accuracy and running time was investigated in [18].

The study of embedding with the assumption of unlimited substrate resources was performed in [9] and [11]. The former focused on load balancing, while the latter attempted to minimize the embedding cost of a single virtual network with a backbone-star topology.

Substrate support for path splitting was envisioned in [12], where the authors also discussed migration for online requests. The authors in [10] added additional location constraints and employed linear programming and deterministic/randomized rounding techniques to achieve better coordination between node and link mapping. A subgraph isomorphism detection-based embedding algorithm was proposed in [13]. The scenario of mapping virtual networks between multiple substrate networks, i.e., inter-domain mapping, was examined in [15].

Topology was incorporated into embedding in research [14, 16, 20]. Two metrics, *Critical Index* and *Popularity Index*, were introduced to differentiate between substrate nodes and links in [16]. Inspired by PageRank [24], Markov chain-based algorithms were developed in [14, 20] to compute the ranking of substrate nodes, which further facilitates VN embedding.

Due to workload's fluctuation, resources purchased by service providers are not adequately used for most of the time. Thus, the authors in [19, 20] proposed opportunistic resource sharing-based mapping algorithms, where substrate resources are shared among multiple virtual networks opportunistically.

VII. Conclusions

This paper considers the virtual network embedding problem and envisions substrate support for parallelization, which provides reliability, fast recovery, and efficient resource utilization. We present a formal problem description and develop two algorithms, *ProactiveP* and *LazyP*. Extensions that deal with additional resource consumption, speedup, and expiration time are proposed. Simulation results confirmed the performance of the proposed algorithms. In future work, we intend to look in detail into parallelization-based embedding schemes, and we will attempt to combine path splitting with parallelization.

Acknowledgments

This work is supported in part by the National NSF of China under Grant No. 61073028 and No. 61021062; Key Project of Jiangsu Research Program under Grant No.BE2010179; Jiangsu Natural Science Foundation under Grant No. BK2011510; the National 973 Basic Research Program of China under Grant No. 2009CB320705 and No. 2011CB302800; College graduate research and innovation project of Jiangsu under Grant No. CXZZ12_0055; US NSF grants ECCS 1128209, CNS 1065444, CCF 1028167, CNS 0948184, and CCF 0830289.

References

[1] J. Turner and D. Taylor, "Diversifying the Internet," in *IEEE GLOBECOM 2005*.
[2] N. Chowdhury and R. Boutaba, "A survey of network virtualization," *Computer Networks*, vol. 54, no. 5, pp. 862–876, 2010.
[3] T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the Internet impasse through virtualization," *Computer*, vol. 38, 2005.
[4] N. Feamster, L.-X. Gao, and J. Rexford, "How to lease the Internet in your spare time," *ACM SIGCOMM CCR*, vol. 37, 2007.
[5] PlanetLab, "http://www.planet-lab.org/."
[6] VINI, "http://www.vini-veritas.net/."
[7] D. G. Andersen, "Theoretical approaches to node assignment," Dec. 2002, unpublished Manuscript.
[8] R. Ricci, C. Alfeld, and J. Lepreau, "A solver for the network testbed mapping problem," *ACM SIGCOMM CCR*, vol. 33, no. 2, 2003.
[9] Y. Zhu and M. Ammar, "Algorithms for assigning substrate network resources to virtual network components," in *IEEE INFOCOM 2006*.
[10] N. Chowdhury, M. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," in *INFOCOM 2009*.
[11] J. Lu and J. Turner, "Efficient mapping of virtual networks onto a shared substrate," *Washington University, Tech. Rep. WUCSE-2006-35*, 2006.
[12] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: substrate support for path splitting and migration," *ACM SIGCOMM CCR*, vol. 38.
[13] J. Lischka and H. Karl, "A virtual network mapping algorithm based on subgraph isomorphism detection," in *VISA 2009*.
[14] X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo, and J. Wang, "Virtual network embedding through topology-aware node ranking," *SIGCOMM CCR*, vol. 41, pp. 38–47, April 2011.
[15] M. Chowdhury, F. Samuel, and R. Boutaba, "Polyvine: policy-based virtual network embedding across multiple domains," in *VISA 2010*.
[16] N. Butt, N. Chowdhury, and R. Boutaba, "Topology-awareness and reoptimization mechanism for virtual network embedding," in *IFIP Networking 2010*.
[17] I. Houidi, W. Louati, D. Zeghlache, P. Papadimitriou, and L. Mathy, "Adaptive virtual network provisioning," in *ACM SIGCOMM VISA 2010*.
[18] S. Zhang, Z. Qian, S. Guo, and S. Lu, "FELL: A flexible virtual network embedding algorithm with guaranteed load balancing," in *ICC 2011*.
[19] S. Zhang, Z. Qian, B. Tang, J. Wu, and S. Lu, "Opportunistic bandwidth sharing for virtual network mapping," in *IEEE Globecom 2011*.
[20] S. Zhang, Z. Qian, J. Wu, and S. Lu, "An opportunistic resource sharing and topology-aware mapping framework for virtual networks," in *IEEE INFOCOM 2012*.
[21] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *AFIPS 1967 (Spring)*.
[22] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. The MIT Press.
[23] ArpaNet, "http://en.wikipedia.org/wiki/arpanet."
[24] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Technical Report 1999-66, November 1999.