Topology-Awareness and Re-optimization Mechanism for Virtual Network Embedding

by

Nabeel Farooq Butt

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2009

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Nabeel Farooq Butt

Abstract

Embedding of virtual network (VN) requests on top of a shared physical network poses an intriguing combination of theoretical and practical challenges. Two major problems with the state-of-the-art VN embedding algorithms are their indifference to the underlying substrate topology and their lack of re-optimization mechanisms for already embedded VN requests. We argue that topology-aware embedding together with re-optimization mechanisms can improve the performance of the previous VN embedding algorithms in terms of acceptance ratio and load balancing. The major contributions of this thesis are twofold: (1) we present a mechanism to differentiate among resources based on their importance in the substrate topology, and (2) we propose a set of algorithms for re-optimizing and re-embedding initially-rejected VN requests after fixing their bottleneck requirements. Through extensive simulations, we show that not only our techniques improve the acceptance ratio, but they also provide the added benefit of balancing load better than previous proposals. The metrics we use to validate our techniques are improvement in acceptance ratio, revenue-cost ratio, incurred cost, and distribution of utilization.

Acknowledgements

I would like to thank Professor Raouf Boutaba for his guidance and support during the preparation of this thesis and the research presented in it. I would also like to acknowledge all other members of the Network Virtualization group with whom I have a useful discussion.

Dedication

This is dedicated to my family for their constant support and encouragement during all my studies.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

Development in the design of the current Internet infrastructure has become stagnant over the last several years. Internet has come to a halt, in the sense that it is almost impossible to introduce new technologies in hardware or software that better realize the needs of the enormously growing user base. This problem is often termed as the Internet Ossification problem[8]. Network virtualization is widely considered a potential candidate to provide the foundation for the future Internet architecture [19]. One major challenge in network virtualization environment (NVE) is the allocation of substrate network resources to online VN requests. The problem of allocating substrate resources for the virtual network (VN) is called VN embedding problem. The VN embedding problem reduces to the multi-way separator problem, which has been shown to be NP-hard [5]. From the point of view of an infrastructure provider (InP), a preferable allocation scheme should increase long-term revenue while reducing the cost of hosting individual VNs. Consequently, mechanisms that can increase acceptance ratio are of great interest to InPs, because acceptance ratio is directly proportional to their revenue.

While embedding a VN request, existing proposals [71, 69, 20, 48, 40] do not distinguish between different substrate nodes and links. However, in practice, some substrate nodes and links are more *critical* than others. For example, resource depletion in bridges and articulation points in the substrate topology are expected to have deeper impact on future embeddings than a random resource near the edge of the network. Choosing amongst feasible VN embeddings, the one that uses fewer critical resources can be a key to improving acceptance ratio in the long run. The

goal of the embedding process should be to minimize the chances of fragmenting resources by creating bottlenecks and at the same time maximize the chances for potential future revenue.

In this thesis, we investigate how differentiating between substrate resources can increase the acceptance ratios of the existing VN embedding algorithms [71, 69, 20, 48, 40]. In order to differentiate resources, we design a mechanism for assigning weights to substrate nodes and links based on their residual capacities and their importance in the substrate topology. These weights help us in prioritizing certain VN embeddings over others. In addition to network partitioning due to link and node depletion, attributes of the created partitions are also an important factor in weight assignment. In case of a substrate network partitioning, any virtual link from a VN request has to be turned down if its two ends can only be mapped in different partitions. We argue that distinguishing substrate resources is a key for better load balancing and for improving the acceptance ratio.

Over time, as new VNs are embedded and old ones expire, resources in the substrate network become fragmented. The obvious consequence is the rejection of many VN requests lowering the acceptance ratio and revenue. This can be amended if the fragmented resources are consolidated using re-optimization techniques. We propose algorithms to identify bottlenecks and techniques to re-embed VNs that cause VN request rejection. Our re-optimization mechanism consists of two stages: detection of bottleneck links and nodes, followed by relocation of virtual links and nodes to free up resources in the bottleneck areas. While consolidating resources for rejected VN requests, we have to make sure that all previous VN embeddings remain intact.

1.2 Problem Statement

Infrastructure Providers (InP) would want their embedding process to optimize their current revenue by effectively embedding VN requests, without adversely affecting the potential future revenue. Existing work on VN embedding falls short of ways to maximize the revenue for the InPs. Specifically, impartial treatment of substrate resources while allocating them to VNs and lack of thorough re-optimizing mechanisms for current VN embeddings are two problems that significantly narrow down the chances for future revenue. If these two problems are ignored *critical* resource assignments will need to increase to handle the cascading embeddings. Consequently, fragmenting the substrate network and hence rejecting a lot of VN

requests which could be mapped otherwise. As a result InPs would have to face lost revenue, decreased acceptance ratio, heavily-utilized clusters of resources, and increased average embedding cost. We shall prove this hypothesis under various scenarios using simulations.

1.3 Objectives

Our objective in this thesis, is to present a comprehensive solution for improving acceptance ratio of existing VN embedding algorithms and distributing load fairly among the critical resources of the substrate network. Our solution aims at improving the performance of previous VN algorithms while at the same time keeping the incurred cost to minimum. To this end, our approach consists in: first differentiating the *critical* resources so that the degree of *criticalness* of resources can be considered while embedding VN requests and secondly by providing a mechanism to re-embed initially-rejected VN requests by re-optimizing already embedded VNs. It is also our objective to make our techniques easy to incorporate with previous VN embedding proposals, i.e. with minimal modifications to previous algorithms.

1.4 Contributions

Our contributions are as follows:

- We present a mechanism for distinguishing resources of the substrate network based on the impact they might have on the substrate network, in case of resource depletion or failure.
- We presents algorithms for re-optimizing and re-embedding initially-rejected VN requests after fixing their bottlenecks requirements.
- Our approach improves acceptance ratio for existing VN embedding algorithms while keeping the cost low. It also provides the added benefit of distributing load fairly on the critical resources of the underlying network.
- Our mechanisms can be incorporated in existing algorithms with minimal changes.
- We evaluate our approach through extensive simulation and show the effectiveness of our techniques in terms of performance gain in existing VN embedding algorithms.

1.5 Thesis Organization

The rest of this thesis is organized as follows; Chapter 2 presents required background regarding VNs and graph theory. Chapter 3 discusses the state of the art in VN embedding algorithms and discusses the motivations for our work. Our proposals for distinguishing and re-optimizing critical resources are discussed in Chapter 4. Performance metrics, experimental setup, and evaluations are presented in Chapter 5. Finally, we conclude the thesis and present some limitations of our work as well as possible future work in Chapter 6.

Chapter 2

Background

In this chapter, we first provide an introduction of Network Virtualization. Next, we define the VN embedding problem. After that we provide some preliminaries from theories used in our algorithms. In particular, we provide some definitions and theorems from graph theory that we used. Multi-commodity flow problem is discussed next. In the end, in a tabular form we describe all the different variables used in this thesis.

2.1 Network Virtualization Environment

Fundamental technological changes are required to the Internet design to cope with the needs and demands of the Internet's user market. The huge success of the Internet is the main problem that hinders these required modifications and is continuously ossifying the Internet. Since its inception, Network Virtualization has gained the confidence of a wider research community to overcome the Internet's *ossification* problem. Network Virtualization provides a promising solution to overcome the Internet's Impasse and proceed with the Internet diversification proposal [64, 8]. The first important contribution of Network Virtualization is to support the coexistence of multiple heterogeneous network architectures onto a shared physical infrastructure. These coexisting networks are called virtual networks (VN). VN is a set of virtual links and virtual nodes used to provides some service. Different VNs can be entirely different in terms of topologies they have, the services they offer, the technologies they use, and the consumer market they focus on. This principal design goal of Network Virtualization broadens the market potential of Network Virtualization because it not only provides flexibility but

also facilitates testing and incremental deployment of new network technologies. The other important principal design goal of Network Virtualization is to introduce diversity by separating the role of an Internet Service Provider (ISP) into an *Infrastructure Provider (InP)* and a *Service Provider (SP)*[64, 8, 32].

2.1.1 History of Network Virtualization

The concepts of virtualization in the field of networking have been around for quite some time for solving different problems, for instance, VLANs, VPNs, active networks and overlay networks etc. In this section, we will briefly discuss some of these concepts. Although all of these systems were invented for different purposes, extending the functionality of the Internet is an important goal they all have in common.

Virtual Private Network (VPN), not to be confused with Virtual Networks, is a dedicated network for private communication between multiple geographically distributed sites and connected through tunnels over the shared Internet [34, 53, 52]. A VPN can be an Intranet, where all sites are owned by the same enterprise, or an extranet, involving multiple enterprises. In the VPN terminology the manager who provisions a VPN is called a Provider Provisioned VPN (PPVPN) [9] that is the service provider for the case of VPN. PPVPN technologies can be classified in to three different categories based on the layer they operate on i.e. Layer 3 VPN (IP,MPLS) [17, 15], Layer 2 VPN (Ethernet) [10, 12] and Layer 1 VPNs (SONET/SDH) [14, 59].

Programmable network research [16] attracted the research community because of its claim of on-demand services on the fly. The important requirement for programmable networks is a clear separation between the software and the communication hardware. Two different approaches were pursued to realize this goal. Open signaling approach [1] from the telecommunication point of view and the active network approach from the IP networks community [62, 61, 68, 50]. The former argues for modeling the hardware interfaces using open programmable interfaces, while the latter propounds the idea of customized computation based on the contents of the packets.

Overlay networks are based on the idea of creating a virtual topology over the existing physical topology, to provide user customized services. Deployment of overlays is not expensive because they are designed to be oblivious to the underlying network. Overlays are flexible and adaptive to the underlying network conditions.

Overlay networks have been used to tackle a plethora of problems faced by the users of the current Internet. Some of the major problems addressed by the overlay research community include performance guarantees [54], availability of the Internet routing [6], multicasting [29, 41, 21, 18], QoS guarantees [57], protection from Denial of Service attacks [43, 7], content distribution [45], file sharing [49], and storage systems [23]. Anderson et al. [8] argue against the viability of overlay networks, since they constitute narrow fixes instead of long term solutions. Indeed, most overlays are designed in the application layer and are oblivious to the underlying protocols and hence do not provide any fundamental change to the Internet architecture.

2.1.2 Business Model & Roles

To provide an on-demand instantiation of VNs by embedding them onto a collection of substrate resources (physical links and physical nodes), Network Virtualization extends current Internet's business model to include several new role players. In the Network Virtualization paradigm there are four important actors; they are the Infrastructure Provider (InP), the Service Provider (SP), the Broker, and the Users [19]. Decoupling the role of an ISP results in the first two new entities;

Infrastructure Provider

An InP, as its name suggests, is responsible for provisioning and managing the underlying network resources on which VNs can be created. It is also responsible for providing interfaces to different SPs to use and lease resources. Other responsibilities include maintenance and operations of the substrate resources. For VN instantiation, InP decides whether to accept or reject it, depending on the revenue and the cost of VN, and the availability of resources.

Multiple InPs can have different service level agreements (SLAs) and can collaborate to provide better services and wider networks. Such collaborations can lead to positive effects in capitalizing the revenue for themselves.

Service Provider

Focusing on the customer market, the Service provider creates a VN that will offer the type of services that users want. SPs achieve this by leasing the network re-

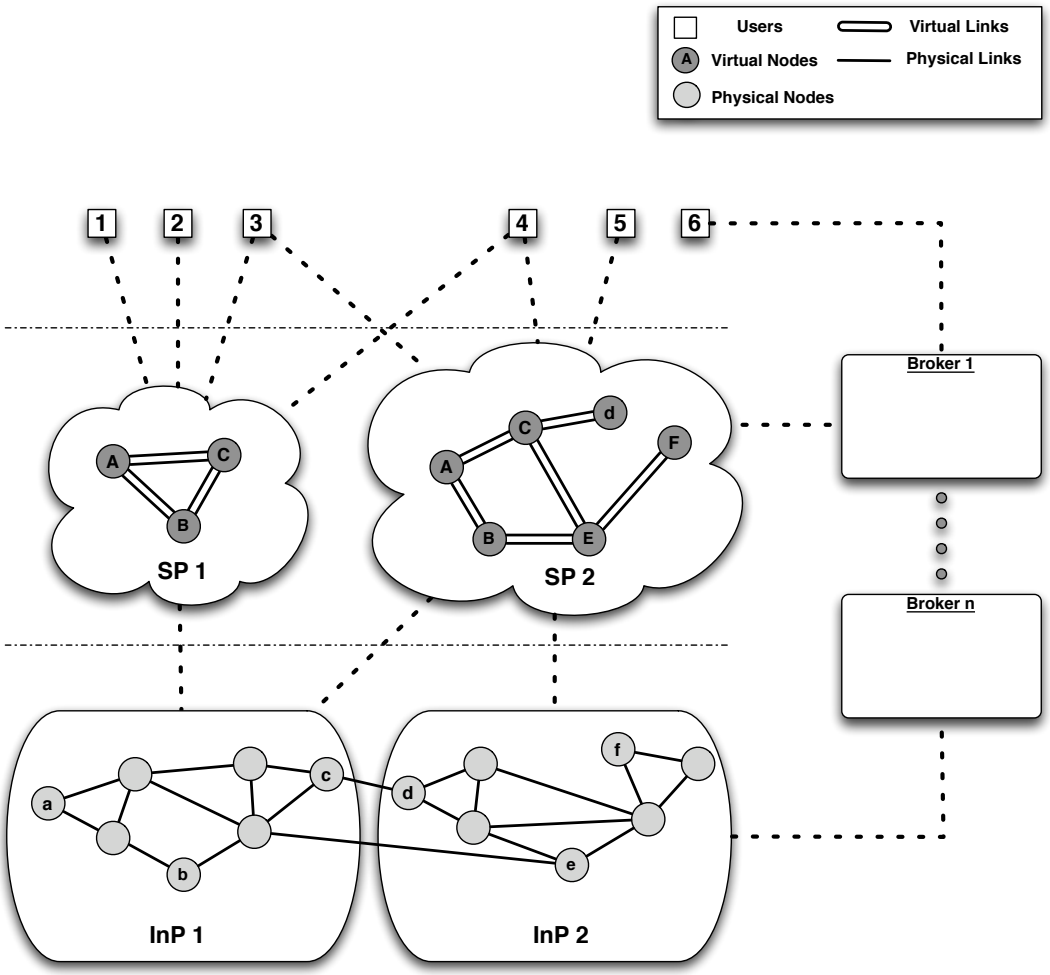

Figure 2.1: Virtual Network Environment

sources from InPs and deploying customized protocols on these allocated substrate resources.

An SP can also perform the role of InP and by leasing other SPs its resources. Instantiating a VN requires a VN request to be submitted by the SP to the InP. After SP requests an InP for the creation of a VN, InP can deny the request if an agreement can not be reached among them or hosting the VN is not feasible for the InP. A third party may mediates this process to simplify the process for both the SP and the InP. These are called brokers.

User

Services are provided by the SPs to the users. Users in an Network Virtualization Environment may connect to multiple VNs to access different services but they do not interact directly with the InP. Terms and conditions might be the other issue that the NVE Users have to deal differently from the existing Internet users. A user can interact with a broker to find out the best SP providing the services user might be looking for. Once a deal is finalized between user and the SP, the user can continue to use the services provided by the SP until the expiration of the agreement.

Broker

A Broker is a middle man in almost all of the interaction between the rest of the three players. Brokers have a vital role to bring all the actors of the Network Virtualization paradigm together. Brokers help SPs lease resources from InPs and may eventually help users to purchase services from the SPs. They might also facilitate the collaboration between InPs to extend their network or to provide network connectivity in some new area.

2.1.3 Example of an NVE

In Figure 2.1, there are two InPs and there are two SPs. One of the SPs, SP 1, is entirely mapped to substrate resources from InP 1 and the other SP, SP 2, has some resources mapped from SP 1 and the rest from SP 2. Brokers are shown at the right side of the figure. You can see that brokers are in contact with multiple other players of the NVE. Broker 1 is providing service to user 6 and at the same time negotiating a deal with SP 2 on behalf of some actor. The networks shown inside the SPs are the virtual topologies of their respective VNs. The capital letter denotes a virtual node and also tells which substrate node it is mapped to, the substrate node with the same letter but in lower case. Note that a virtual node can only be mapped to a single substrate node whereas a virtual link can be mapped to multiple substrate paths. The edges shown in the virtual topology correspond to paths in the substrate topology.

Observe that users 3 and 4 are connected to both the service providers, SP 1 and SP 2, whereas all the other users are connected to one SP only. In NVE, the user may subscribe to multiple SPs to access the different services they offer. VNs can

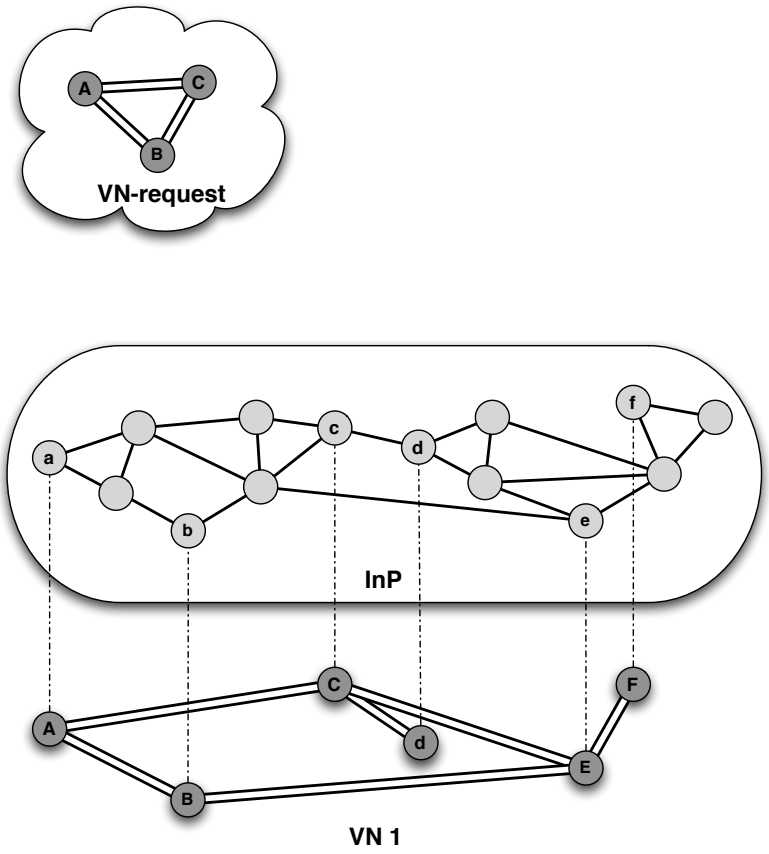

Figure 2.2: VN Embedding Example: a VN request arrives

have different lifetimes depending on the nature of the service provided to customer. One VN might be providing an email service and would want to live indefinitely in the system, whereas another might be just broadcasting a particular conference in some area and would be active only for a few hours. InPs can provide one or a combination of different types of infrastructure networks; a wired or wireless access network, and/or an optical backbone network, etc.

2.2 VN Embedding Problem

Provisioning resources for VN creation, known as VN embedding, is one of the major challenges faced by the research community. Embedding a VN on to the substrate network is equivalent to finding a mapping, of all the virtual nodes and the virtual links on to the substrate network. Even the offline version, where all VN requests are known in advance, can be reduced to the multi-way separator problem

[5], which is NP-hard. The objective of the VN embedding problem is to find an effective and efficient embedding for the VN request.

For a better understanding of this problem, an example is shown in figure 2.3. A VN, VN 1, is already mapped on to the substrate network. The virtual link mapping is not shown for VN 1, but there is a substrate path for every virtual edge. The VN request provides the desired virtual network topology of the VN it wants to create. Node and link constraints are also an important part of the VN request that directs the embedding process. As can be seen in figure 2.3, the nodes of the VN requests are mapped to the substrate nodes (the dotted lines). The virtual links are also mapped, and the mapping can be seen as the substrate paths of the same color as that of the corresponding virtual links.

While embedding the VN request, the InP might have a number of different objectives. It might want to maximize InP's own revenue and minimize the cost of hosting the VN. It might want to maintain the level of service up to a certain standard, and if mapping any more VNs can create problems it might reject any new VN requests. Ideally, all the current embeddings should be efficient all the time, to better utilize the resources of the underlying network but the online nature of VN request arrival makes it hard since embedding a new VN can cause previously embedded VNs to become inefficient. Optimizing all the VN embeddings, might be too expensive but on the other hand too many inefficient VNs can fragment the network and InPs might unnecessarily have to reject a lot of incoming VN requests resulting in loss of revenue. An InP has to decide whether to improve the embeddings of certain VNs or leave it to some later time.

Existing works propose different heuristics for VN embedding problem and provide approximate solutions. We will discuss these proposals in chapter 3. In the following subsections we provide theoretical formulation of the network model, node and link stress, and cost and revenue functions used by existing solutions to the VN embedding problem.

2.2.1 Network Model

Substrate Network

A substrate network is denoted by $G_S = (N_S, E_S)$, where N_S is the set of substrate nodes and associated with each node, n_s, in this set is a pair of attributes: CPU capacity, $c(n_S)$, and geographic location, $loc(n_S)$. The set E_S is a set of substrate

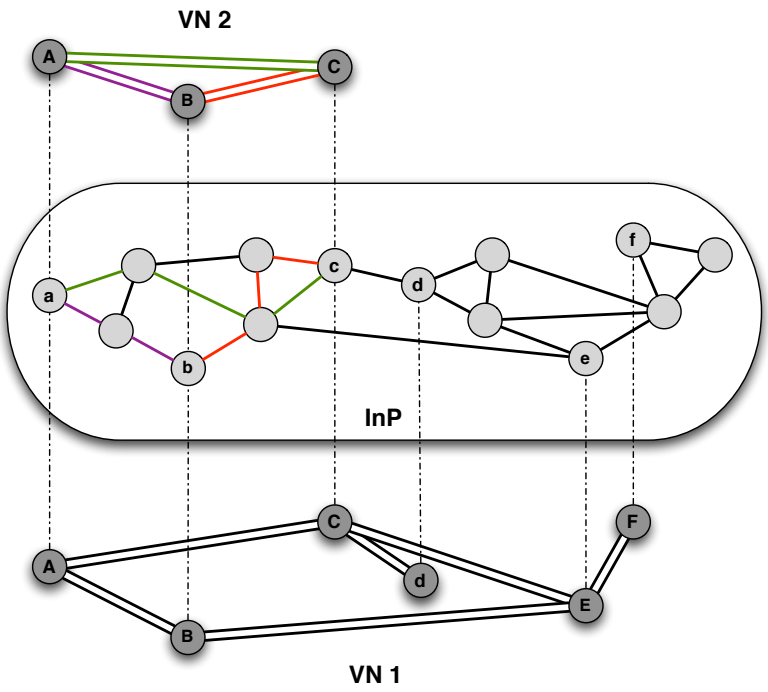

Figure 2.3: VN Embedding Example: VN request is mapped to the substrate network

links and each link, e_s, has an associated bandwidth capacity, $b(e_S)$, and the delay of that link, $d(e_S)$.

Virtual Network Request

A request for creating a virtual network consists of a desired topology and a set of constraints on that topology. In the literature, the VN request is modeled as a weighted undirected graph. In this thesis, we denote the virtual network graph as $G_V = (N_V, E_V)$. N_V is a set of virtual nodes and associated with each virtual node is a set of constraints $K(n_v)$, that the embedding should satisfy. This set of constraints can contain geographical location, required CPU capacity, memory etc. Similarly, E_V is a set of virtual links and associated with each virtual node is a set of constraints $K(e_v)$. These constraints should be satisfied by any feasible embedding. These constraints can be required bandwidth capacity, maximum delay, maximum loss rate etc.

Node & Link Stress

Node and link stresses are important factors in quantifying the resource utilization. These notions also appear in previous literature [71, 69, 20]. Node stress is defined as the fraction of all the CPU capacity of the substrate node that is mapped to some virtual node. We denote node stress by $S_{_N}$ and $n_{_V} \Lambda n_{_S}$ means that the virtual node $n_{_V}$ is mapped onto the substrate node $n_{_S}$

$$S_{_N}(n_{_S}) = \sum_{n_{_V} \diamond n_{_S}} c(n_{_V}) \tag{2.1}$$

Similarly, the link stress is defined as the fraction of all the bandwidth capacity of a substrate link that is mapped to some virtual link. It is denoted by $S_{_E}$.

$$S_{_E}(e_{_S}) = \sum_{e_{_V} \diamond e_{_S}} b(e_{_V}) \tag{2.2}$$

We define the residual capacity of a link or a node in terms of its stress. For a node, it is the total CPU capacity minus the node stress of that node and for a link its the total bandwidth capacity minus the link stress.

Node & Link Assignment

While embedding a virtual node, one more constraint in addition to all the others in $K(n_{_S})$ must be satisfied. This constraint is to map a virtual node to exactly one substrate node. In the case of a link, there are two cases depending on whether the flow is splittable or unsplittable. In the former case, the mapping should be a set of paths from the associated substrate nodes of the virtual link, and in the latter case, it is just a single path.

Revenue & Cost Functions

For optimizing the embedding, the notion of cost and revenue of a VN request are very important. In previous research [71, 69, 20] the notions of cost and revenue have been defined as follows.

Cost of a VN is defined as the total amount of substrate resources allocated to the VN. This include all the bandwidth capacities on all the substrate links and all the CPU capacities on all the substrate nodes.

Mathematically, the cost denoted by $C(G_{_V})$, is given by.

$$C(G_V) = \sum_{e_V \in E_V} \sum_{e_S \in E_S} f_{e_S}^{e_V} + \sum_{n_V \in N_V} c(n_V) \tag{2.3}$$

where $c(n_V)$ is the allocated CPU capacity for the virtual node n_V and $f_{e_S}^{e_V}$ is the allocated bandwidth capacity for the virtual link e_V on the substrate link e_S.

Revenue is defined as the total resources requested as part of the VN request. It is denoted by $R(G_V)$ and computed as follows:

$$R(G_V) = \sum_{e_V \in E_V} b(e_V) + \sum_{n_V \in N_V} c(n_V) \tag{2.4}$$

The set of constraints together with the cost and the revenue associated with a virtual network dictates the actual embedding of the virtual links and nodes of this virtual network.

2.3 Graph Theory Preliminaries

To improve the embedding of a VN request, it is important to consider the structure of the underlying substrate network. This section provides some graph theory background that we later use in the formulation of the problem, and in the algorithms proposed in this thesis.

2.3.1 Cut vertices and Cut edges:

A cut vertex is a vertex whose removal disconnects the graph. Similarly, a cut edge is an edge whose removal makes the graph disconnected. Other names for cut vertices and cut edges are articulation points and bridges. An important thing to note is that removal of a cut node can increase the number of components of the graph up to the degree of that node. While removing an edge can increase the number of components at most by 1. Efficient algorithms exist for checking whether an edge or a node is a cut edge or a cut node. Algorithms to enumerate all cut nodes and cut edges are given in the Appendix.

2.3.2 Maximum Flow and Minimum Cut:

A theorem from graph theory used in this work is the Max-Flow Min-Cut theorem [24]. The maximal flow minimum cut theorem establishes that the maximum

amount of flow from source 's' to sink 't' is equal to the minimum capacity of the s-t cut. A cut is a set of edges that partitions the graph such that source is in one partition and sink is in the other. The capacity of the cut is the capacity of the edges in the cut. In other words, this theorem states that the maximum amount of flow from the source to the sink is equal to the minimum capacity that needs to be removed from the network so that no flow can pass from the source to the sink. There exist polynomial time algorithms for computing maximum flow and minimum cut of a graph [35, 36, 42, 26].

Edmonds and Karp[26] presented an algorithm which is an implementation of the Ford Fulkerson method [35] for computing the maximum flow in a network. Dinic[25] also presented algorithms for computing maximum flow. The running time of this algorithm is $O(VE^2)$. A more efficient algorithms for computing max flow was also proposed and is called the push-relabel algorithm [37]. Its running time in the general case is $O(V^2E)$ and with the dynamic tree structure of Sleator and Tarjan [56] it runs in $O(VE\log(V^2/E))$.

2.3.3 Multi-Commodity Flow Problem:

Multi-commodity flow (MCF) problem is a network flow problem in which multiple commodities need to be transported from some source to some sink. Every commodity has two nodes associated that will carry traffic. In the case of integer variables, Muti-Commodity flow is NP-complete even for only two commodities [30]. In the fractional version of the problem, a solution is found in polynomial time. Lets assume $G(V, E)$ to be a flow network, $c(u, v)$ to be the capacity of edge (u, v). There are k commodities, denoted by K and for each commodity k_i we have the source s_i, the sink t_i and the demand d_i. The flow along the edge (u, v) is $f_i(u, v)$. The constraints and objective function for this problem are given below.

Capacity Constraints:

$$\sum_{i=1}^{k} f_i(u, v) \leq c(u, v) \tag{2.5}$$

The capacity constraints stipulate that the flow on any edge should be less than or equal to the capacity of that edge.

Flow Conservation:

$$\sum_{w \in V}^{k} f_i(u, v) = 0 \text{ when } u \neq s_i, t_i \forall v, u, f_i(u, v) = -f_i(v, u) \tag{2.6}$$

Flow conservation constraints make sure that all the traffic is generated at the source node and consumed at the sink node. For all the other nodes, the flow into a node is equal to the flow out of that node.

Demand Satisfaction Constraints:

$$\sum_{w \in V}^{k} f_i(s_i, w) = \sum_{w \in V}^{k} f_i(w, t_i) = d_i \tag{2.7}$$

Finally, constrains on all edges ensure that all bandwidth requirements are satisfied.

Now, for minimum cost multi-commodity flow, there is a cost associated with every edge, $cost(u, v)$, and the objective is to minimize it. The solution of the following linear program gives the minimum cost flow between all the commodities.

Objective: minimize

$$\sum_{w \in V} cost(u, v) \sum_{w \in V}^{k} f_i(s_i, w) \tag{2.8}$$

2.4 Terminology

Table 2.1 provides the terminology the terminology used for the substrate and virtual resources used throughout the remainder of the thesis.

Variables	**Description**
c_1	Fraction of nodes contained in connected component 1
c_2	Fraction of nodes contained in connected component 2
C	Set of connected components when a cut-vertex is removed
ϕ	Ratio of introduced to total cut-vertices when a link/node is removed
ψ	Ratio of introduced to total cut-edges when a link/node is removed
δ	A very very small constant
$\zeta(x)$	Critical Index of x
$\rho(x)$	Popularity Index of x
$R_N(x)$	Residual or Available capacity of a node x
$R_E(x)$	Residual or Available capacity of an edge x
f_{uv}^i	Denotes flow from u to v for virtual edge i
$\aleph(x)$	Scaling factor for resources x
α	A variable to assigning different weights to CI
β	A variable to assigning different weights to PI
ω	A variable to further scale the SF

Table 2.1: Description of Variables used in the thesis

Chapter 3

Related Work

In this chapter, we present an overview of the existing work addressing the VN embedding problem and the problem of re-optimizing already embedded VNs. We begin with categorizing previous proposals based on the type of techniques they use to solve the VN embedding problem. We explain the assumptions they make and their impact. Finally, we describe the previous work done in the area of re-optimizing the VN embeddings.

3.1 Static VN Embedding Approaches

The primary incentive for the InP in embedding a VN request on its substrate network is to increase its revenue while keeping the incurred cost minimum. Other than that, keeping the load distributed among the substrate resources increases the chances of provisioning resources later on and hence adds the potential revenue. We have classified the static VN embedding algorithms from the existing literature in various categories as described below.

3.1.1 Two Stage Solutions

By two stage solutions, we mean that the nodes and the links are assigned in a sequential order. Even the offline version of the VN embedding problem is NP-hard as it can be reduced to multi-way separator problem [5]. The unsplittable flow version of the problem is still NP-hard even if the nodes are already mapped. Consequently, existing research [31, 48, 71, 69, 20] has relied on heuristic based solutions for the edge mapping after they have already determined the mappings

of the nodes in a greedy fashion. Mapping the nodes beforehand simplifies the problem. Subsequently the links can be mapped either using k-shortest paths or solving the requirement specific LP programs for Multi Commodity Flow (MCF). Zhu and Ammar [71] and Lu and Turner [48] have focused on the offline version of the problem. Lu and Turner [48] developed algorithms for specific topologies such as star and hub-and-spoke topologies. Their experimentation considered a single VN. Precluding admission control complexity by assuming infinite capacities of substrate nodes and links characterizes the works in [31, 71, 48]. The authors in [69] used k-shortest paths [27, 28] and MCF [4] for embedding virtual links once they have greedily mapped all the nodes. They also support the case of customized node mapping and they present a customized node mapping algorithm for hub-and-spoke topology. Although customized node mapping is useful tool, it is not applicable to arbitrary topologies. One important reason, greedy approaches have been extensively used for node mapping is that other approaches like simulated annealing [48] and iterative approaches [44, 31] are more expensive. Chowdhury et al. [20] recently proposed a solution to the VN embedding problem by coordinating the node and the link mapping phases using a mixed integer programming formulation [55]. Although, their algorithms coordinate the node and link mapping phases using the notion of meta nodes, they still fall in the category of two stage solutions.

3.1.2 Single Stage Solutions

There are few algorithms that map the nodes and the links at the same time. Lischka and Karl [47] introduced a backtracking VN embedding algorithm based on subgraph isomorphism detection. Their approach is based on the graph matching techniques of Cordella et al. [22]. The advantage of such algorithms is that whenever a bad mapping is discovered they can backtrack to previous feasible embedding while the two stage algorithms have to calculate all the node and link mappings again. Through simulation Lischka and Karl [47] show that their algorithms are faster and better than those of Yu et al. [69]. On the other hand, they did not show how efficiently they can search for alternate embeddings and the effects of backtracking (in terms of network stability and existing VN embeddings disruption).

Distributed algorithms for VN embedding have also been proposed such as [40]. They use distributed algorithms based on the Multi-Agent Systems [63] to embed the links and the nodes simultaneously. However, to simplify the problem of efficiently embedding the VN request in a distributed manner, they assumed unlimited

resources, so that all the VN requests can be accepted. Distributed approaches, as compared to centralized approaches, have certain benefits like reduced communication cost, no single point of failure, parallel processing of multiple VN requests, and decreased time to repair partial mapping failures. On the other hand, they perform rather poorly as compared to their centralized counterparts [20].

As mentioned by Lu and Turner [48], embedding techniques that try to map node and links in the same phase are victim of one of the two problems: expensive re-computation of the current modified network conditions; and a potentially huge number of local fixes to choose from.

3.1.3 Links Only Embedding

Some existing works focused on VN embedding which considers only the link constraints. Allocating resources for Virtual Private Networks [38, 51] is the earliest, but simpler, incarnation of the VN embedding problem. The fact that they consider only the bandwidth constraints on virtual links makes it easier that the VN embedding problem. Since constraints on virtual nodes are ignored, this problem often simplifies to selecting paths that satisfy the given constraints. In [51], the only restriction on nodes assumed by the authors was that no two VNs can be mapped to the same node. They have used simulated annealing to map VNs. Szeto et al. [58] have presented algorithms for embedding only the virtual links, with an assumption that the virtual nodes have already been mapped.

3.2 Dynamic VN Re-optimization Approaches

Haider et al. [39] categorized the approaches for resource assignment in two categories: static and dynamic. In static approaches, the VN embeddings do not change during their lifetime, while in the dynamic approaches, the VN embeddings are reconfigured based on some notion of current performance and resources stress. Static approaches typically result in under utilization of the physical resources. Arriving and departing VNs also affect the utilization of the underlying network infrastructure, since some previously and mapped VNs might not be resource efficient anymore. This may cause fragmentation among substrate resources and considerably lowers the chances of supporting future VN requests. To overcome this problem, different proposals have been made in the literature. Zhu and

Ammar[71] argue that selective reconfiguration should be done periodically to optimize the overloaded segments of the substrate network. Yu et al. [69] present a case for path splitting and migration to periodically rebalance the mappings of VNs so as to make efficient use of the substrate network and consequently increase the chance of accepting future VN requests. We will discuss these proposals in more detail below.

3.2.1 Selective Reconfiguration

Based on prior work on rerouting in circuit-switched networks [66], Zhu and Ammar [71] studied the problem of dynamic resource assignment. Since the reconfiguration of VN embedding can cause a significant amount of relocation of resources, its impact can be worse than that of the flow re-routing problem. Zhu and Ammar [71] used the weighted sum of reconfiguration rate, node switching rate and link switching rate to quantify the cost of reconfiguration. Their selective reconfiguration mechanism first identifies the nodes and links whose stress ratios are above a given threshold and then marks all the VNs containing these over stressed resources for reconfiguration. Reconfiguration only takes place in the next reconfiguration interval. If a VN finds itself marked, during a reconfiguration interval, it reallocates substrate resources. They call this step *per VN reconfiguration*.

Our work differs from that of Zhu and Ammar [71], in that we quantify the cost of reconfiguration as the difference between the initial cost and the final cost after reconfiguring. Zhu and Ammar [71] define node stress ratio and link stress ratio as the average node and link stress to maximum node and link stress, where stress is the number of virtual nodes or links mapped to a substrate node or link. They used these notions for periodic reconfiguration of selected critical VNs. They control the cost of reconfiguration by controlling the periodic interval. To the contrary, we use a reactive reconfiguration rather than a periodic one by reconfiguring only when a VN request gets rejected. Another drawback of selective reconfiguration is that it selects over stressed resources and reconfigures all the VNs containing these resources. We argue that the reconfiguration of all these nodes to reduce the stress is not necessary and an improvement over this would be to reconfigure only as many low priority VNs as required to reduce the stress of these resources. Finally, another shortcoming of the selective reconfiguration approach is that the location of the resource is really important since a highly stressed resource at the edge of the network might not be as problematic as an average stressed resource in the core of the network because of potential network partitioning.

3.2.2 Path Splitting & Migration

Contrary to Haider et al. [39], we classify the work of Yu et al. [69] on path splitting and migration as a dynamic approach to VN embedding. The reason being that the embedding of the VNs change during their lifetime. Classification of this work as static or dynamic also depends on whether path splitting and migration are implemented at the substrate level and embedding process is oblivious to that or not. Yu et al. [69] propose greedy node mapping algorithms. The objective of their algorithms is to maximize revenue. For embedding the virtual links they use the k-shortest path algorithm [27, 28] and an MCF based LP formulation. They also use path splitting and path migration for optimizing the resource utilization at periodic intervals. One benefit of using a periodic interval is that it allows them to consider the online VN embedding problem as if it was offline. Again, the cost associated with periodic optimization might still be too expensive. Path migration can be implemented in terms of path splitting, by adding new paths to old ones and slightly changing the splitting ratio. Substrate routers can implement consistent hashing to realize path migration [13]. Node migration can also be achieved using the work of Wood et al. [67]. To avoid out-of-order packet delivery they use hash-based splitting. Hash-based schemes have been used in IP networks to evenly split traffic between paths of the same costs [11, 33].

The algorithms presented by Yu et al. [69] use path migration of already embedded virtual links, keeping the node mappings fixed. VN requests can also be rejected if a virtual node mapping fails, in that case path splitting and migration would not be of any use. In our re-optimization mechanism, we also consider node migration. While migrating node, however we also have to migrate all the virtual links connected to them.

Previous approaches strive to improve VN embedding at various degrees, none of them foresee the possibility of narrowing down the chances for accepting future VN requests, while embedding. They treat all the substrate nodes and links equally, which is not the case in practice. Their indifference to the underlying topology may lead to non optimal decisions while mapping resources on 'critical' substrate links and nodes that can negatively affect the acceptance ratio in the future. A complementary approach for improving the acceptance ratio, is the effective use of re-optimization mechanisms. By detecting and relocating bottleneck link and node embeddings to better alternates, we can achieve improvements. Ideally, we want to keep all the VNs optimized at all times (e.g., through periodic updating [71]). But this may result in the relocation of a lot of virtual links and virtual nodes and hence

incur significant overhead. A better way to deal with this problem is to take action only when it becomes inevitable [69]. However, unlike [69] which only considers link migrations, our algorithms takes into account relocation of both virtual links and nodes.

Chapter 4

Topology Awareness and Re-optimization

Indifference to the attributes of the substrate network resources and lack of re-optimization mechanisms are the two major candidates for improvement in the existing VN embedding algorithms [20, 69, 71, 48, 47]. Significant performance improvement can be achieved, once these problems have been effectively resolved. Impartial treatment of all the links and nodes of the substrate network, not only narrows down the chances of accepting future VN requests, load balancing is also affected by it. In this work, we treat each node and link differently by assigning each of them different weight functions. The weight function reflects how severe the impact if this resource was reserved. Embedding of certain VNs can drift toward in-efficiency with the arrival and departure of other VNs. Previous literature on VN embedding other than Zhu and Ammar [71] and Yu et al. [69], did not study the performance gain that can be achieved by re-optimizing and re-embedding of the VNs. In the same time and as discussed in chapter 3, the studies in [71, 69] are limited in several ways. Zhu and Ammar [71] proposed a periodic reconfiguration of selected critical VNs, while the selection is based on the notion of link and node stress, and where stress is the number of virtual nodes or links mapped to a substrate node or link. Since the resources required to satisfy a VN request are not known in advance, we cannot correlate the selection process (for periodic reconfiguration) with the impact it might have in the future. Yu et al. [69] use path migration for migrating already embedded virtual links, keeping the node mappings fixed. However, it is possible that path migration can still leave the problem unresolved. In our work, we use reactive re-optimization, which is performed whenever a VN request gets rejected, and we consider both node and link re-optimization.

Our approach also differs in the way we select potential VNs to relocate. In the following sections, we present our techniques to incorporate topology-awareness and re-optimization of VN embeddings.

4.1 Topology-Aware Embedding

We differentiate between substrate network resources by introducing scaling factors to their costs. The *scaling factor (SF)* of a resource refers to its likelihood of becoming a bottleneck. It is calculated as a combination of two weight factors that are explained later in this section.

We aim to incorporate topology-awareness in existing algorithms with minimal changes. In our solution, proposals using MCF only need to update their objective functions (constraints require no change) and proposals using shortest path algorithms should select paths with minimum total scaled costs to upgrade themselves. By scaling the cost of all the resources by SF, we prioritize the embeddings that affect the substrate network the least.

4.1.1 Critical Index

The first weight function we define is the *critical index (CI)* of a substrate resource. The CI of a resource measures the likelihood of a residual substrate network partition[1] due to its unavailability. To understand the impact of such partitioning, consider the following scenario: suppose that a substrate network is partitioned into two, almost equal-sized, components. The probability of both the end nodes of any new virtual link to be embedded in different partitions is 0.5. In this case, we might have to reject almost 50% of the VN requests.

We denote CI by ζ ($\zeta : x \rightarrow [0, 1)$), where x is either a substrate link or a node. Higher value of $\zeta(x)$ means its highly likely that unavailability of x will partition the substrate graph and vice versa. The mathematical definition of CI is given in equations 4.1 and 4.2 for links and nodes respectively.

[1]We refer to residual substrate network while discussing partitioning. A residual substrate network is composed of residual capacities of substrate resources.

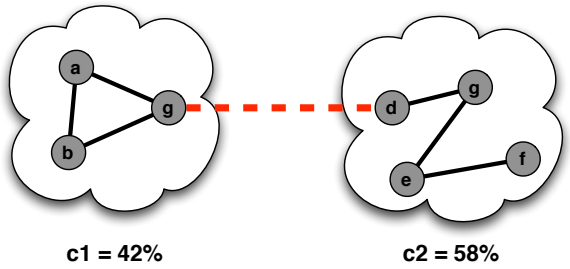

Figure 4.1: Edge (g,d) is a bridge

Figure 4.2: Bridges and cut-nodes introduced when removing (g,d)

$$\zeta(e_s) = \begin{cases} \left(\frac{1-|c_1-c_2|}{2}\right) + \frac{1}{2} & e_s \in \text{cut-edges} \\ \frac{\phi+\psi}{4} & e_s \notin \text{cut-edges} \end{cases} \tag{4.1}$$

$$\zeta(n_s) = \begin{cases} \frac{1}{2}\left(\prod_{c\in C} P\{c\}\right)^{\frac{1}{|C|}} + \frac{1}{2} & n_s \in \text{cut-vertices} \\ \frac{\phi+\psi}{4} & n_s \notin \text{cut-vertices} \end{cases} \tag{4.2}$$

In equation 4.1, if e_s is a bridge (cut-edge) then removing it partitions the graph. c_1 and c_2 are the fractions of substrate nodes in each of these partitions. In Fig. 4.1, the edge (g,d) is a bridge and removing it disconnects the underlying network. The first and the second component have 42% and 58% of the nodes respectively. So, while solving equation 4.1 for edge (g,d), values for $c_1 = 0.42$ and $c_2 = 0.52$ respectively. When e_s is not a bridge, then after removing it we get ϕ and ψ, where ϕ (ψ) is the fractions of new cut-nodes (cut-edges) to the total cut-nodes (cut-edges) respectively. For example, in Fig. 4.2, for calculating the CI for the edge (c,f), values of ϕ and ψ are 0.25 and 0.75 respectively. To understand this, have a look at Fig. 4.2, initially yellow nodes (i.e. node b, f, and g) are cut-nodes and the

edge (f,g) is the only cut-edge. After removing (f,g), three new cut-edges (i.e. (i,f), (b,i), and (b,c)) are introduced and only one new node becomes a cut-node (node i). The values of ϕ and ψ are then 0.25 and 0.75 respectively and $\zeta(e_s) = 0.25$.

In order to compute CI for nodes, we might have to deal with more than two components, as removing a cut-node can partition the network in more than two components whereas a cut-edge can only increase the total number of components by just one. In equation 4.2, C is the set of components we get when we remove n_s. $P\{c\}$ here is the fraction of the total nodes present in component c. If n_s is not a cut-node, the CI is calculated similar to the case when e_s is not a cut-edge. For understanding the other case, consider Figure. 4.3 where the node u is a cut-node and removing it partitions the graph into three components. $P\{c\}$ is shown in the figure and $|C| = 3$ and hence $\zeta(n_s) = 0.64$. To distingish between a cut-resource and normal ones, we make sure that a cut-resource has its CI value ≥ 0.5. Computing ζ is required only once in the beginning and it is updated whenever a substrate network is extended. Efficient algorithms exist for computing articulation points [60] and for computing the connected components and are used in calculating ζ. The time complexity for the bi-connected components and articulation points algorithms is $O(V + E)$. Computing ζ is not computationally intensive and the algorithm we used for calclating ζ is given in the Appendix.

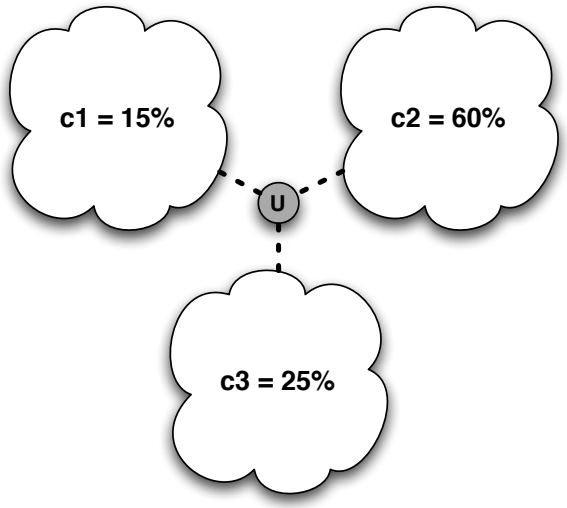

Figure 4.3: Calculating Critical Index for a cut-node u

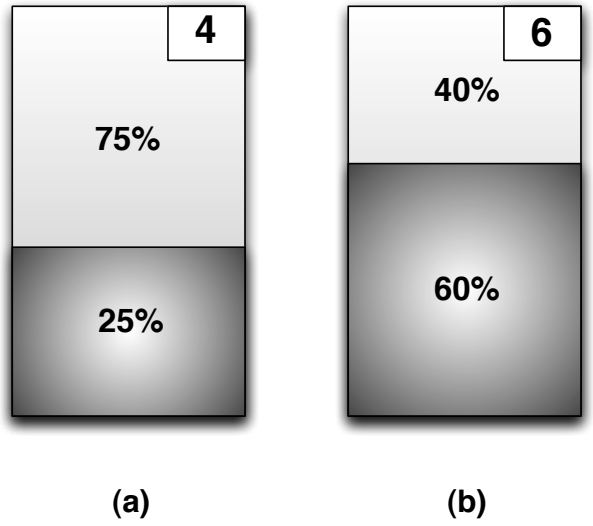

Figure 4.4: Calculating Popularity Index for a resource

4.1.2 Popularity Index

Resource saturation is the other major factor that can cause substrate network partition. We call this weight function *Popularity Index (PI)*, denoted by ρ and $\rho : x \rightarrow [0, 1)$. PI measures "how severe an impact a resource's unavailability can have on the underlying network?". To be more specific, PI is the time weighted average of used resources on a particular link or a node. PI also depends on the number of VNs that are mapped onto that resource. In equation 4.3, $R_{E_{i-1}}$ is the percent of reserved bandwidth on link x, where the index $i-1$ means the previous value of R_E; similarly, $R_{N_{i-1}}$ is the percent of reserved CPU capacity at the node x. The variables a and b $(a+b=1)$ are used to assign different weights to current and previous values. In equation 4.3, ν is the number of VNs mapped on top of x. The higher the value of ρ the higher the probability that mapping onto this link or node will saturate it and create a bottleneck.

$$\rho(x) = \begin{cases} \left(aR_{E_{i-1}} + bR_{E_i}\right)^{\frac{1}{\nu}} & x \in E_S \\ \\ \left(aR_{N_{i-1}} + bR_{N_i}\right)^{\frac{1}{\nu}} & x \in N_S \end{cases} \tag{4.3}$$

For illustration see Fig. 4.4. For any resource, (whether a link or a node) lets assume that at some time i its used capacity (bandwidth capacity for links and CPU

capacity for nodes) is 60% and the numbers of VNs mapped onto it are 6. Calculate PI involves calculating the weighted average, which depends on two different weight values a and b, of the fraction of resources currently in use and then the fraction of resources previously used. To account for the number of VNs that can be affected, we raise that value by the reciprocal of the number of VNs currently mapped on that resource.

The purpose of PI is to prioritize resources that are not depleted. Not only does PI provide a way to fend off underlying network partitioning due to resource depletion, it also provides load balancing as a byproduct. Load balancing is achieved by favoring resources based on the degree of criticialness associated with them. The reason is that utilization is an important factor in calculating the PI.

4.1.3 Scaling Factor

CI and PI values are both crucial when mapping a particular link or node. Although, both measure the chances of partitioning the substrate network either because of resources outage or resource depletion, they are quite different in nature. The former focuses more on the topological attributes of the underlying network and the latter aims at utilized resources and their impacts on the substrate network. We unify them in a single parameter, called *Scaling Factor (SF)* and denote it by $\aleph$. In equation 4.4, we have multiplied both CI and PI by weights α and β respectively. We also use a parameter ω to further scale the cost. It would be interesting to see what are the effects of different settings of α, β, and ω. SF is used to scale the cost of resources while embedding a VN request.

$$\aleph(x) = 1 + \omega\big(\alpha\zeta(x) + \beta\rho(x)\big) \tag{4.4}$$

$$\text{minimize:} \sum_{e_V \in E_V} \sum_{e_S} \aleph(e_S) f_{e_V}^{e_S} \tag{4.5}$$

The objective functions of the LP formulations of previous proposals [20, 69] can be updated by scaling the cost of resources by $\aleph$. Objective function of a basic LP-formulation for the MCF problem with topology-aware feature added would look something like the equation 4.5.

To illustrate the concept of differentiating by scaling the cost, have a look at Fig. 4.5. We are given a VN request that has two virtual nodes and a virtual

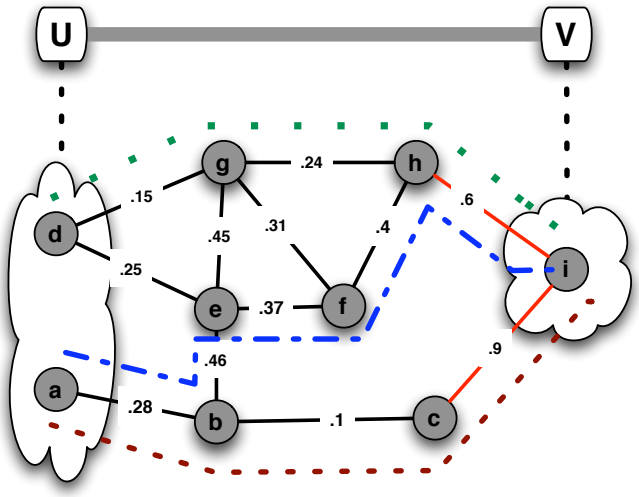

Figure 4.5: Differentiation embedding with the help of SF

link between them. Virtual node U can be mapped to substrate nodes a or d, where as virtual node V can only be mapped onto substrate node i. Given this node mapping, for the virtual link (U,V), lets say, there are only three possible embeddings (actually path here), which are shown in different colors. It can be seen that the *green* path (d-g-h-i) has the lowest SFs along the path, which makes it a preferred candidate for embedding (U,V).

We use a slightly different cost function than that of the previous literature. In the modified formula, we are scaling up the cost by the scaling factor to avoid embedding onto critical resources whenever possible. In the formula below, n_S is the substrate node on which a virtual node n_V is mapped.

$$C(G_V) = \sum_{e_V \in E_V} \sum_{e_S \in E_S} \aleph(e_S) f_{e_S}^{e_V} + \sum_{n_V \in N_V} \aleph(n_S) c(n_V) \tag{4.6}$$

4.2 Re-optimizing Bottleneck Embeddings

In this section, we describe our re-optimizaton mechanism. Once embedded, a virtual network remains operational for its lifetime, which can be arbitrary and can only be known at the time of VN request arrival. Expiration of a VN can change the network condition and as a result, some of the previously mapped VN embeddings can become inefficient or non-optimal. Embedding is inefficient or non-optimal in the sense that there exists another embedding that is better, in

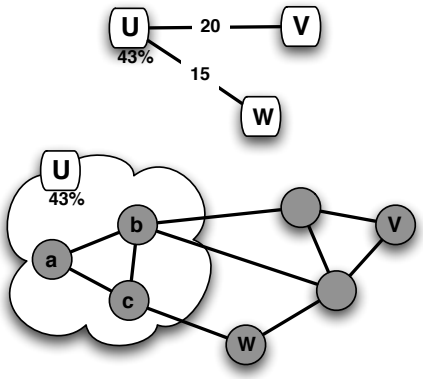

Figure 4.6: Unneccessarily refused VN link

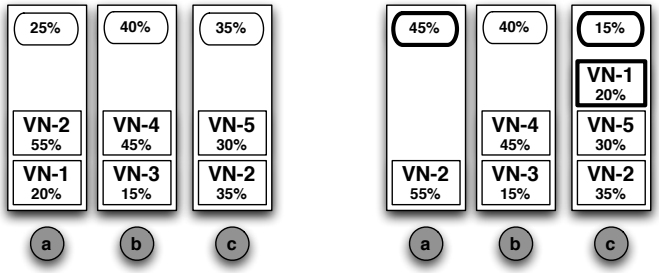

Figure 4.7: Relocating a virtual node of VN-1 to make room for virtual node U

terms of some specific objective. The objective in our case is to reduce the number of bottlenecks. Inefficient embedding may still satisfy all the constraints of the VN, but it might limit the resources at critical nodes and links. Such embeddings can be reoptimized by relocating virtual nodes and reassigning virtual links to better alternate embeddings. Although efficient algorithms exist for embedding of VNs onto a substrate network, but they cannot forsee the VN embeddings drifting toward inefficiency. The goal of most of the previous work is on efficient first time embedding [20, 48, 40]. Only Zhu and Ammar [71] and Yu et al. [69], did somework on improving the inefficient embeddings. Zhu and Ammar [71] have proposed reconfiguration at periodic intervals based on their notion of node and link stress. Yu et al. [69], uses path migration and only take action when it is inevitable.

From a practical point of view, it has been shown how to migrate virtual nodes (virtual routers) from one physical node to another [65]. Virtual router migration is achieved by migrating the control plane to the new location. It supports asyn-

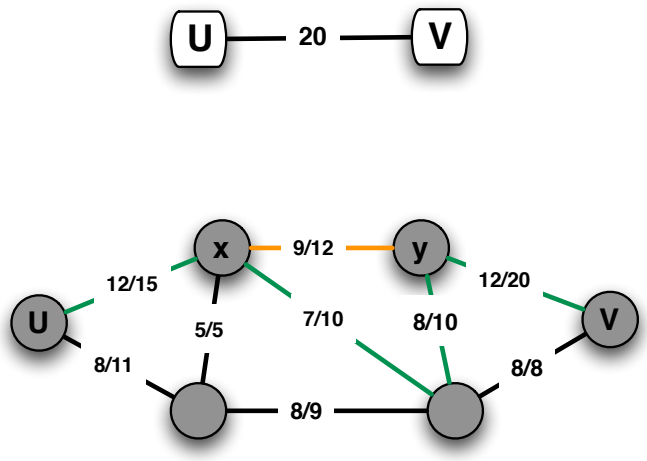

Figure 4.8: Unneccessarily refused VN link

chrounous migration of links by cloning the data plane at the new location during migration and temporarily forwarding packets in both data planes. Link migration can also be achieved by dynamically setting up and tearing down physical links; it has been already achieved using programmable transport networks [2].

Our re-optimization mechanism is a two-stage reactive solution. In the first stage, we detect the bottleneck virtual links and nodes that caused the rejection of a VN request. Next stage deals with relocating/reassigning these nodes/links to less critical regions of the substrate network so that the rejected VN request may be accepted. As compared to the periodic reconfiguration of Zhu and Ammar [71], our reactive re-optimization mechanism is much less costly and has a little overhead. Periodic reconfiguration marks the highly stressed nodes and links and in the next periodic interval some of the VNs containing these nodes re-embeds themselves. An argument against this strategy is that, an average utilized resource in the core (backbone) might be more crucial a bottleneck than a higly stressed node/link at the edge of the network. On the other hand, we take a more rigorous mathematical approach to detect bottleneck links and nodes rather than selective reconfiguration based on nodes and links stresses. The incurred cost, the cost of re-embedding plus the cost of relocation of resources, is also much lesser in the reactive approach as compared to the periodic one.

We illustrate the basic idea behind our relocation and reassignment mechanism using the example in Fig. 4.6, 4.7, 4.8, 4.9, and 4.10. In Fig. 4.6, a VN request is being rejected because one of its virtual nodes cannot be mapped, i.e., node U which requires 43% of CPU capacity. There are three possible embeddings for this

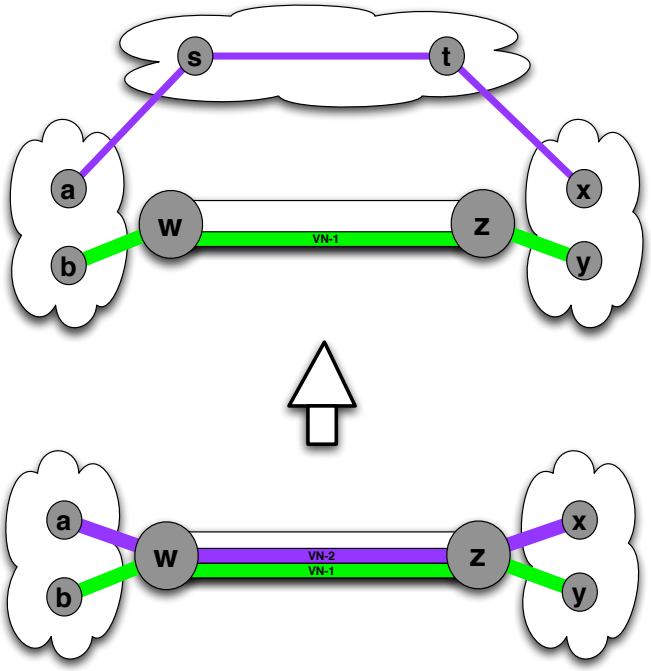

Figure 4.9: Reassigning links to make room for virtual link

node i.e. substrate nodes a, b, and c (shown inside a cloud), but none have the required capacity to accomodate virtual node U. Fig. 4.7 shows the currently embedded virtual nodes on the substrate nodes a, b, and c. We can see in Fig. 4.7, that relocating the VN-1 node from node a to node c makes enough room to accommodate U. Note that all the associated virtual links of VN-1 should also be reassigned accordingly.

Now, suppose that a VN request has just one virtual link i.e. (U,V). This virtual link is turned down because of the bottleneck substrate link (w,z), as shown in Fig. 4.8. Our technique to find the bottleneck nodes and links will be discussed later. As can be seen in Fig. 4.9, VN-2 can be reassigned to free up some bandwidth on link (w,z) and finally virtual link (U,V) can be embedded. Bottleneck links are selected from the minimum cut of the virtual link (U,V), as shown in Fig. 4.10.

4.2.1 Detecting Bottleneck Nodes and Links

For bottleneck detection, we deal with VN requests that cannot be embedded due to lack of residual capacities in substrate resources. Contrary to the periodic reconfiguration scheme presented in [71], we have taken a reactive approach in detecting

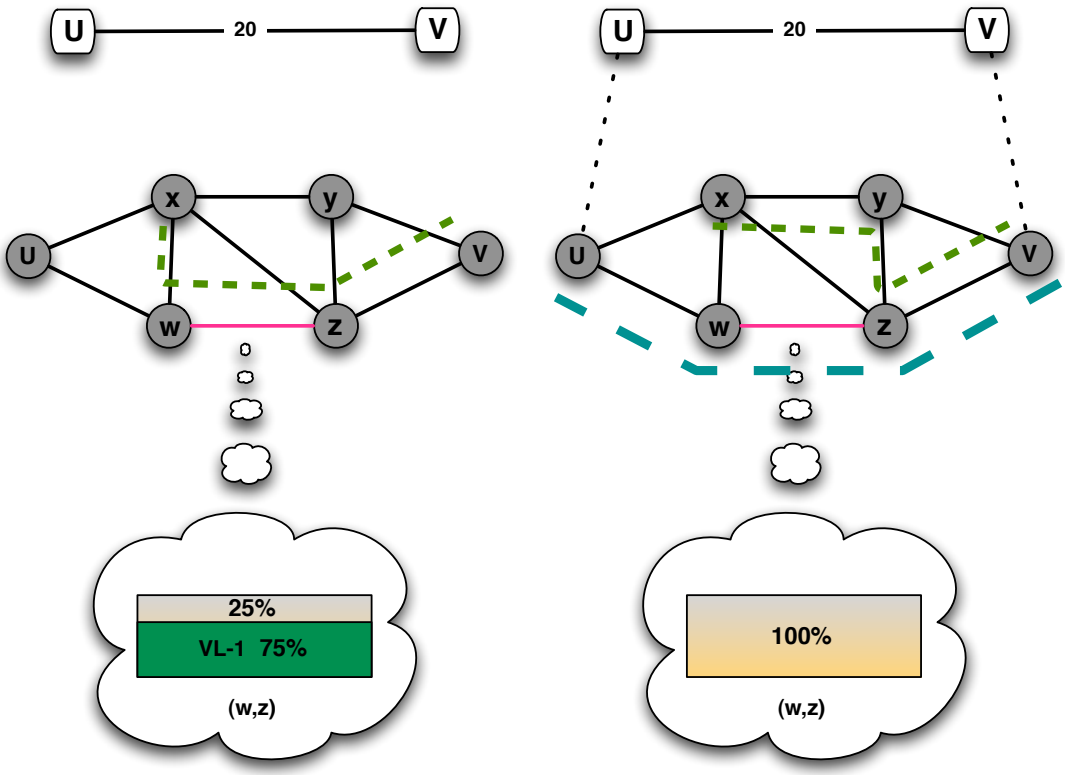

Figure 4.10: Reassigning virtual link VL-1 to make room for some other virtual link

Algorithm 1 Detecting V-Nodes that cannot be mapped

1: $N_{bad} \leftarrow$ v-nodes that cannot be mapped
2: **for all** $v_n \in N_{bad}$ **do**
3: $C_{N_S}(v_n) \leftarrow$ candidate nodes
4: **for all** $n \in C_{N_S}(v_n)$ **do**
5: $L_{low} = \{x : x$ has low priority than $VN_{rej}\}$
6: **if** capacity(L_{low}) > required capacity **then**
7: Save (n, L_{low}), for relocating later
8: **end if**
9: **end for**
10: fails if v_n cannot be fixed
11: **end for**

bottlenecks – we reoptimize whenever a VN request gets rejected.

Algorithm 1, finds a list of virtual nodes that cannot be mapped, and for each one of them it stores corresponding bottleneck substrate nodes. After detecting the

Algorithm 2 Detecting V-Links that cannot be mapped

1: L_{bad} = { v : MaxFlow(v) < ReqCap(v) }
2: **for all** $v \in L_{bad}$ **do**
3: **repeat**
4: **for all** $e \in MinCut(v)$ **do**
5: $L_{low} = \{x : x$ has low priority than $VN_{rej}\}$
6: Save (e, L_{low})
7: **end for**
8: **until** $Capacity(MinCut(v)) < ReqCapacity(v)$
9: fails if v cannot be fixed
10: **end for**

candidate substrate nodes for an unmapped virtual node, it determines whether enough capacity can be salvaged on any of the candidate nodes or actually on L_{low}. L_{low} is the set of virtual nodes of other VN requests with less priority than the rejected VN request. If a single virtual node cannot be mapped we do not proceed any further. Finally L_{low} and associated substrate node n are stored for later use in the relocation phase. The worst case running time of algorithm 1 is $O(n^2)$, but on the average it would be much faster than that.

Next, we use algorithm 2 to determine the unmapped virtual links, L_{bad}, and corresponding bottleneck links. A virtual link is rejected if there is not enough capacity between its two end nodes. The maximum flow from the source to the sink[2] and the minimum s-t cut can be computed in $O(V^3)$ using the Edmond-Karp maximum flow algorithm [24]. Algorithm 2 iterates through L_{bad} and for every virtual link it finds the minimum cut[3]. Re-assigning some virtual links on the minimum cut can increase the maximum flow between source and sink. Note that even if we made enough room in the first minimum cut it might still not guarantee the required flow between the source and the sink. Since the minimum cut can change over time, we have to iteratively compute the minimum cut. To keep the running time of algorithm 2 small, we repeat this only a constant number of times. Next we re-assign links in L_{low} to free up some capacity on these links. If we fail to map only a single virtual link; we can just reject this request immediately.

4.2.2 Nodes & Links Selection and Placement

[2]Source and Sink are the end nodes of the rejected virtual link

[3]for information on Max-Flow Min-Cut theorem, readers are referred to [24]

Algorithm 3 Relocating a Virtual Node

```
1: Q_vn = v-nodes that are adjacent to vn
2: C_vn = candidate s-nodes for vn
3: status = false
4: for all n ∈ C_sn do
5:    status = true
6:    for all v ∈ Q_vn do
7:       Map-Link(Substrate(v), vn)
8:       if map fails then
9:          status = false
10:         break
11:      end if
12:   end for
13:   if status then
14:      Relocate vn to n
15:   end if
16: end for
```

Algorithm 4 Reassigning a Virtual Link

```
1: R_E(e_S) = 0
2: (u, v) be the virtual nodes of e_V
3: c = MaxFlow(u, v)
4: if c = 0 then
5:    return false
6: end if
7: if ReqCap(e_V) <= c then
8:    Re-embed virtual link e_V
9: else
10:   Free c amount of flow from e_S
11:   Augment c amount of flow elsewhere
12: end if
13: Restore and Adjust R_E(e_S)
14: return true
```

Algorithm 3, relocates a virtual node vn which is currently mapped on a substrate node sn to some other substrate node, provided all the virtual links incident on vn can also be reassigned. Q_{vn} is a set of virtual nodes adjacent to vn and C_{vn} is a list of substrate nodes which are potential hosts for vn. Note that C_{vn} is

sorted according to the overhead cost. The algorithm iterates on C_{vn} and for every substrate node n in C_{vn}, it checks whether it can map all the virtual links after vn is relocated to n. If it can map all these links then vn can be relocated to the substrate node used to map these links; finally, it relocates vn.

For reassigning a virtual link, algorithm 4 takes the virtual link e_V which is mapped to substrate link e_S. First it sets the available bandwidth on e_S to zero, so that the freed capacity on e_S is not allocated to e_V, when e_V is reassigned. It then finds the maximum flow between the end nodes of e_V and if it is zero than nothing can be done. If the maximum flow is greater than the required bandwidth capacity of the link e_V then it just frees the old mapping of e_V and remaps it again. If the maximum flow is less than the required capacity then we can free c amounts of capacity from e_S and map c units of capacity elsewhere by using the computed flow. Finally, we restore and adjust the freed bandwidth capacity on e_S (which was set to zero at the beginning). These link mappings should be done using topology-aware LP formulation to keep the overhead cost of reassigning e_V to a minimum.

Chapter 5

Evaluation

Our evaluation focuses primarily on quantifying the effectiveness of our techniques when applied to the VN embedding algorithms in the existing literature. We classify the previous VN embedding algorithms into several categories based on how they map nodes and links, as done in [20]. We consider the following four algorithms in our evaluation: D-ViNE and R-ViNE presented in [20] map nodes deterministically and randomly, respectively; in both the algorithms, links are embedded using a modified MCF. Zhu and Ammar [71] and Yu et al. [69] both use greedy node mapping; for link embedding, the former uses shortest path algorithm, whereas the latter uses multi-commodity flow. Hence, we use the term G-SP for algorithms of Zhu and Ammar [71] and S-MCF for algorithms of Yu et al. [69], where G stands for geedy, SP stands for shortest path, and MCF stands for multi-commodity flow.

5.1 Experimental Setup and Performance Metrics

We have extended the simulator of [20] to include our proposed techniques and algorithms. We have used GT-ITM [70] for generating topologies for the underlying substrate networks. Substrate networks in our experiments have 100 nodes and around 400 links, on average. Node CPU capacities and link bandwidth capacities are randomly chosen between 50 to 100. For VN requests, we used a similar setup as previously used in [20, 71, 69]. VN requests arrive in a Poisson process with an inter-arrival time of 20 time units. Lifetimes of the VN requests follow an exponential distribution with mean 1000 time units. Virtual nodes for VN requests are chosen uniformly between 5 to 10. CPU and bandwidth requirements are distributed

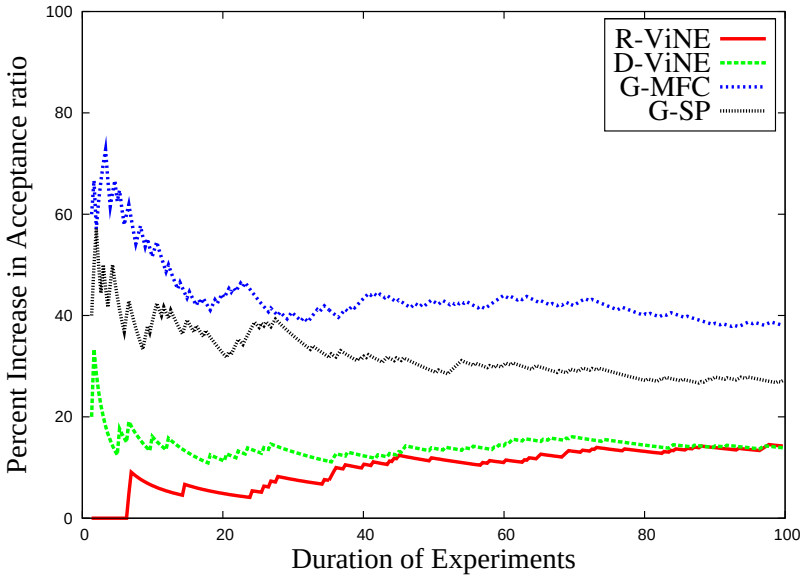

Figure 5.1: Improvement in Acceptance Ratio for Different Algorithms

uniformly between 0 to 20 and 0 to 50 respectively. The metrics we use to measure performance in these experiments are increase in acceptance ratio, revenue-cost ratio, incurred cost, and distribution of utilization of resources.

5.2 Evaluation Results

5.2.1 Improvement in Acceptance Ratio

In our first set of experiments, we compare the increase in acceptance ratios of all the algorithms. As shown in Fig. 5.1, in steady state, our mechanisms improve acceptance ratios by almost 40% for G-MCF and just below 35% for G-SP. For D-ViNE and R-ViNE, the improvements are smaller but still a sizeable 17%. We believe that the lower increase for D-ViNE and R-ViNE is due to the fact that they already have much higher acceptance ratio than G-SP and G-MCF without the proposed improvements [20].

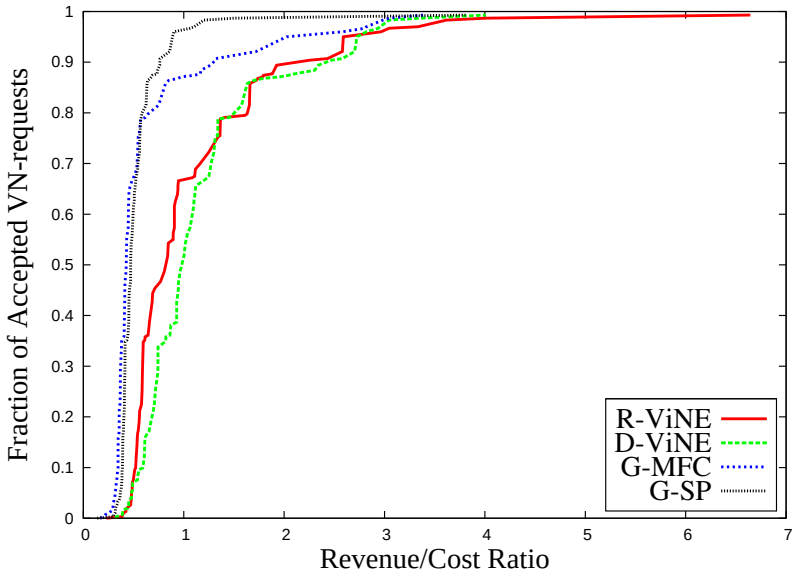

Figure 5.2: Revenue-Cost Ratio for Different Algorithms

5.2.2 Revenue/Cost Ratio

Next, in Fig. 5.2, we present the per request revenue/cost ratio for the compared algorithms. We see here that quite a few observations are close to zero which is an indication that those VN requests are being mapped to one or more very critical links or nodes. On the average, the revenue/cost ratio of D-ViNE and R-ViNE is slightly above 2 for 90% of the accepted VN requests, which is significantly higher than that of G-SP and G-MCF. This shows that our techniques not only improve the acceptance ratios (Fig. 5.1), but they also keep the revenue/cost ratio within acceptable range (specially for D-ViNE and R-ViNE).

5.2.3 Acceptance Ratio vs Incurred Cost

An important factor in evaluating the effectiveness of our techniques is the cost incurred by increasing the acceptance ratio. In Fig. 5.3, we have plotted the percentage increase in acceptance ratio against the percentage increase in cost for making room for these VN requests. Here the incurred cost is the sum of the cost of moving already embedded VN requests and accepting VN requests rejected earlier. As shown in Fig. 5.3, the maximum increase in cost is close to 250%, but the

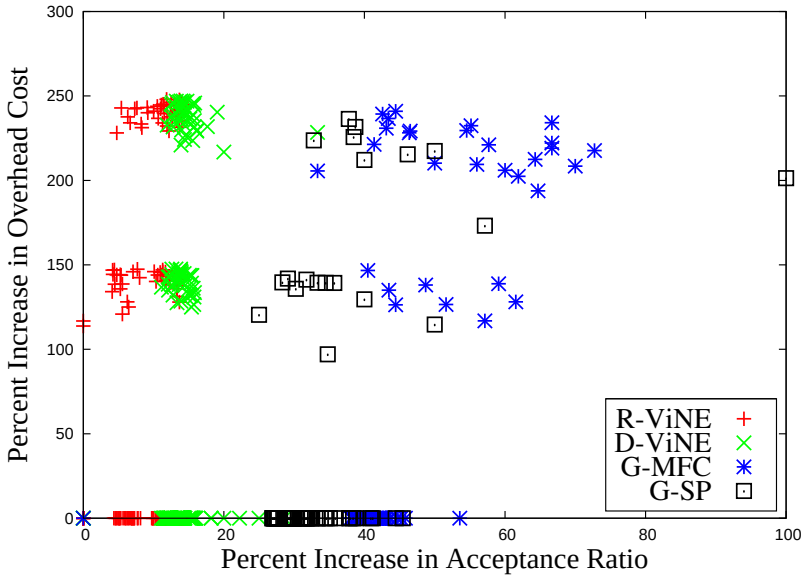

Figure 5.3: Acceptance Ratio vs Incurred Cost

range of acceptance ratio improvement varies from $7 - 77\%$ for various algorithms. For all the algorithms there are quite a lot of observations, where the incurred cost is very low or even zero. This is because the VNs required to move to make room for rejected VN requests have a lot of alternative embeddings of equal cost, and hence incurred no additional cost.

5.2.4 Differentiating Resources

In Fig. 5.4, we have drawn the CDF of the scaling factors of all links and nodes for R-ViNE [20] and R-ViNE incorporated with topology-aware embedding (hereinafter referred to as "TA-R-ViNE"). The other algorithms, compared with their extended versions, also show similar characteristics. SF is a good measure because it gives an indication of potential bottlenecks; an SF value close to 1 means that a resource is more likely to become a bottleneck. We can see here that TA-R-ViNE has significantly fewer resources being indicated as bottlenecks than R-ViNE. With TA-R-ViNE, only 15% resources have SF values above 0.4; whereas with R-ViNE, it is almost 35%. This graph shows that topology-awareness plays a significant role in identifying bottlenecks.

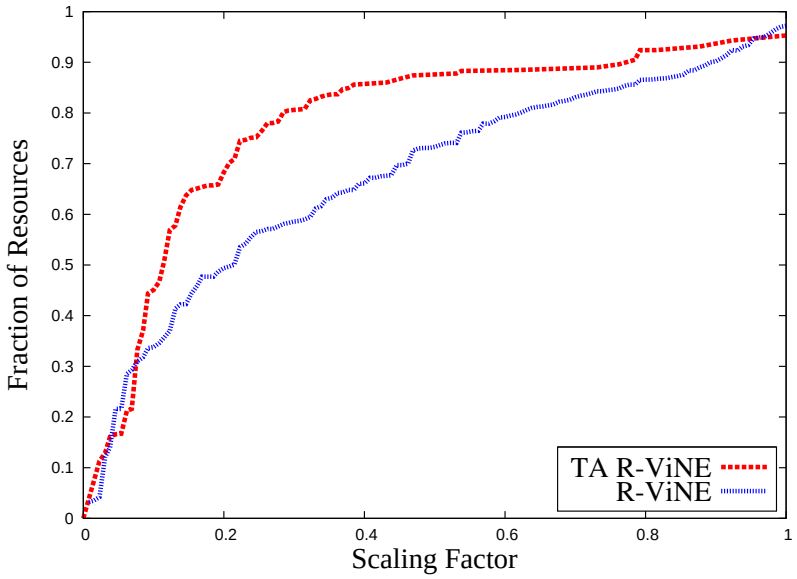

Figure 5.4: CDF of SFs of all links and nodes

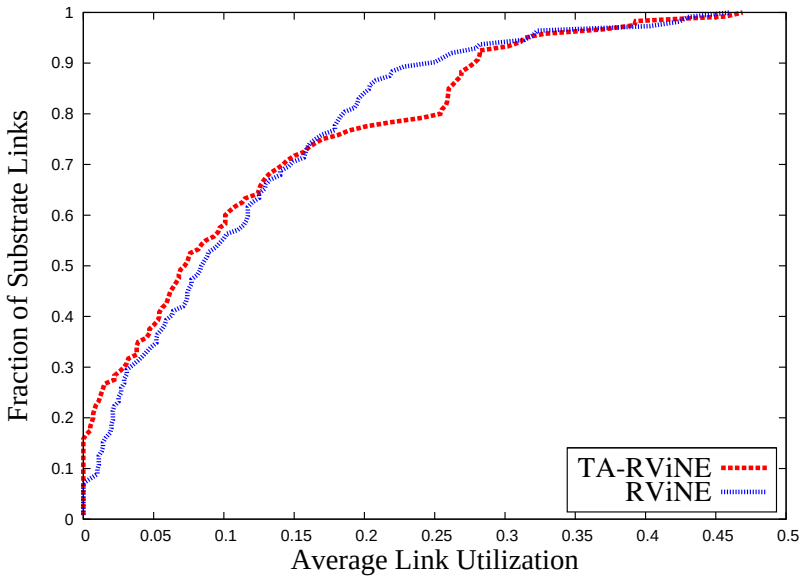

Figure 5.5: Average link utilization

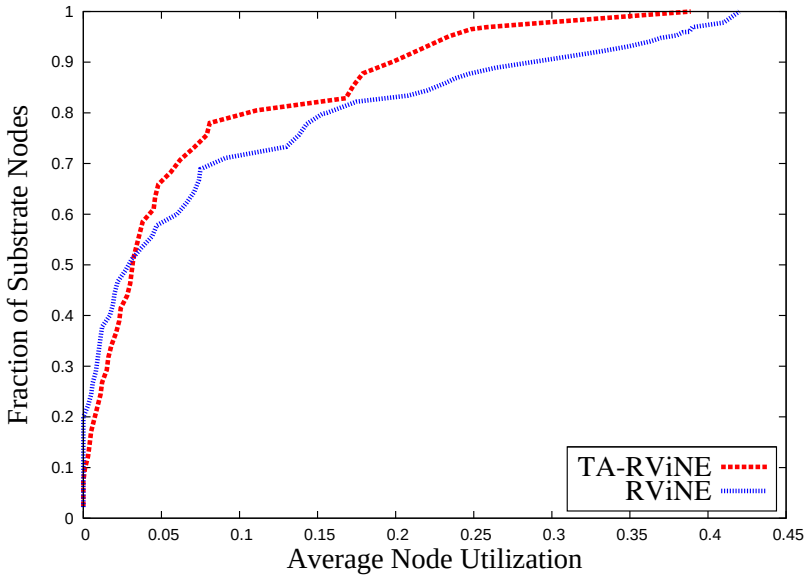

Figure 5.6: Average node utilization

5.2.5 Link and Node Utilization

A comparison between link utilization of TA-R-ViNE and R-ViNE is shown in Fig. 5.5. It presents an average of multiple runs of carefully designed experiments that ensured that both techniques could map the same VN requests. We can see that for R-ViNE, almost 90% of the links are utilizing less than 25% of bandwidth, whereas for TA-R-ViNE it is almost 80%. An explanation would be that these links are alternative links to bottlenecks; hence they are used more often. We can also see that TA-R-ViNE used almost 8% fewer substrate links than R-ViNE. In Fig. 5.6, almost 50% of the nodes have less utilization than 50% of the nodes mapped by TA-R-ViNE, but the other 50% have much higher utilization. This figure shows that to avoid bottlenecks TA-R-ViNE has distributed load among nodes.

Chapter 6

Conclusions and Future Work

6.1 Contributions

While still in its infancy, Network Virtualization offers a lot of exciting challenges that spread across multiple research areas. One of the major challenges, is to design effective and efficient online VN embedding algorithms for network virtualization environment. In this thesis, we have presented mechanisms for differentiating various resources (links and nodes) according to their impact on the connectivity of the substrate network, and their utilization coupled with the number of VNs utilizing them. We have also proposed algorithms for re-optimizing the already embedded VN requests in order to improve the overall acceptance ratio. Our techniques can be easily integrated with existing VN embedding algorithms to improve their performance.

Through simulation we have shown that our techniques significantly improve the performance of existing algorithms in terms of increased acceptance ratio, revenue-cost ratio, cost incurred by these techniques and load balancing based on *criticalness* of resources. From a practical as well as theoretical point of view, we show that our techniques are sound, practical, and effective. Experimental results endorse the fact that our techniques have enough potential to be seriously considered as an integral part of any future VN embedding algorithm design. The performance metrics we used to evaluate our techniques include improvement in acceptance ratio, revenue-cost ratio, incurred cost, and distribution of utilization. Some experimental results when our proposed techniques are incorporated are: (1) $17 - 40\%$ improvement in acceptance ratio for various VN embedding algorithms, (2) Revenue/Cost ratio of 90% of accommodated VN requests is slightly above 2, and (3)

the maximum cost for accommodation is 250%.

Despite our efforts there are still some challenges that need to be addressed before these techniques can be deployed in real world scenario. The following section present some issues open for future research.

6.2 Limitations and Future Work

6.2.1 Stability Analysis

We currently assume that whenever a bottleneck node or a bottleneck link is fixed by relocation, it will not result in a need for further relocations, since we are relocating greedily. Before choosing the target virtual resources to relocate we calculate the relocation cost of all candidate targets and we chose the one with the lowest cost. To address this problem a threshold can be used to decide whether to proceed with the relocation or just drop the request.

Furthermore, a more comprehensive approach to this problem would be to design a relocation strategy which can ensure that equilibrium is reached with minimum disruption. In this perspective a game theoretic approach can be used to model the VNs currently as players whose objective is to optimize their respective embedding by relocating their virtual resources to under utilized parts of the substrate network. This however requires further investigation.

6.2.2 Graph Isomorphic Re-optimization

The idea of graph isomorphism and re-optimization of VNs seem to be naturally related. Achieving optimization using graph isomorphism, for the sake of optimizing the overall embedding of the VNs on a substrate network, is obviously an idea worthy of a thorough analysis and comprehensive evaluation. On top of that, if all the bottleneck VN collaborate and make as less disruption (relocation) as possible then a scheme like this can ameliorate the performance and utilization of the substrate network significantly. The major problem in pursuing this idea would be the NP-hardness of the graph isomorphism problem. We can address this problem using approximate algorithms. We can also rely on heuristics to limit the search space of the problem and then try to solve it. However, decreasing the search space might involve introducing more constraints on the virtual nodes and

virtual links. An obvious problem with introducing more constraints is that quite a number of VN requests will get rejected resulting in a decrease in revenue.

6.2.3 Multi-Commodity Max-flow Min-cut Techniques

Bottleneck link detection was performed in our work using the seminal work of Ford and Fulkerson [35]. They showed that for 1-commodity flow problems, the max-flow and the min-cut are always equal. Leighton and Rao [46] present approximate max-flow min-cut theorem for uniform multi-commodity flow problem. In a uniform multi-commodity flow problem, every pair of nodes is a commodity and all the commodities have the same demand. They showed that in uniform multi-commodity flow problems the max-flow is within a $\Theta(\log n)$-factor of the min-cut. They also presented applications of the flow results regarding various network problems such as path routing problem, network reconfiguration, and communication cost in distributed networks. These ideas can be helpful in the context of VN embedding. As a future work, we would like to design more sophisticated bottleneck detection and re-optimization techniques based on the theoretical results of Leighton and Rao [46].

6.2.4 Distributed Topology-Aware VN embedding

An important direction for future research is to study how topology-awareness can be incorporated in the distributed design of the VN embedding problem. Distributed approaches with their obvious benefits (i.e., reduced communication cost, no single point of failure, parallel processing of multiple VN requests etc.) as compared to centralized approaches are potential candidates for reconfiguration and re-optimization of existing VN embeddings. A major obstacle in this regard, would be the dynamic identification of bridges (cut edges) and articulation nodes (cut nodes) in the substrate network. Ahuja and Zhu [3] presented an efficient algorithm for cut nodes, cut edges, and biconnected components in asynchronous networks. In the context of topology-awareness, what complicates the problem is the calculation of scaling factors for all these nodes and edges. Recall that calculating scaling factor (SF) also require a fraction of nodes in all created components. This may require a distributed algorithm to keep track of the cut resources and all the components they can form and the fraction of nodes these components can have. Unlike the centralized version we studied in this thesis, this may add to the complexity of the problem.

Another area of interest would be to study topology-awareness in wireless or ad hoc settings. An important factor that SF should also depend on, in this setting, would be the rate at which a particular resource goes down. It might be the case that a highly critical resource might go down, over burdening a lot of other critical resources. To prevent this scenario, the formulation of SF should be updated accordingly and the SF should be directly proportional to the rate at which a resources connects/disconnects.

6.3 Concluding Note

We have presented in this thesis a comprehensive approach to improve the performance of the existing virtual network embedding algorithms by differentiating substrate network resources and incorporating re-optimization mechanism. Our mechanisms of topology-awareness and re-optimizing requires little modification to previous virtual network embedding algorithms. Although there are still several issues that need to be addressed, to the best of our knowledge we present a first comprehensive approach to improving virtual network embedding by capitalizing on topology-aware embedding and re-optimization mechanism. We envisage our approach being implemented as an integral part of the virtual network embedding design principals. In such a scenario, this will help Infrastructure providers to increase their revenue while keeping the cost of embedding low.

APPENDICES

Appendix A

Enumerating all cut nodes

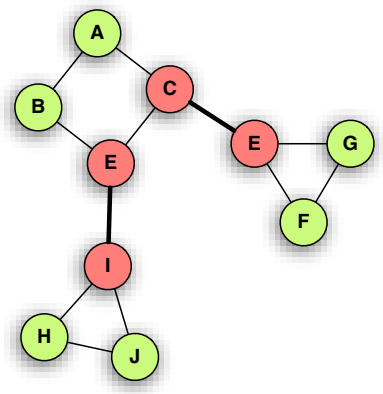

Figure A.1: Enumerating all the cut-nodes

Let G_π be any depth-first search tree (DFST) of the given graph G. When a DFS is done on a graph, a DFS tree is obtained with the starting node as the root of the tree. Every node can be labeled as the depth first number (DFN), which is the order in which the DFS first visited this node. All the edges that are also in the DFST are called *tree edges* and all the rest of the edges are called *back edges*. Every root node of the sub-DFST, which is formed by a biconnected component will be called the root of that component.

Some of the basic facts required to understand the algorithm are as follow.

- The root of G_π is an articulation point of G *iff* it has more than one child.
- A non-root vertex v in G_π is an articulation point of G *iff* v has a child w in G_π such that no vertex in subtree(w)[1] is connected to a proper ancestor of v

[1]subtree(w) denotes the subtree rooted at w in G_π

Algorithm 5 Enumerating all articulation nodes

Art-Node(v,u)

```
1: low[v] ← DFN[v] ← time ← time + 1
2: for all w ≠ u s.t. (v, w) ∈ E do
3:    if DFN[w] = 0 then
4:       call Art-Node(w,v)
5:       low[v] ← min{low[v], low[w]}
6:       if DFN[v] = 1 and DFN[v] ≠ 2 then
7:          print v is an articulation point
8:       end if
9:       if DFN[v] ≠ 1 and low[w] ≥ DFN[v] then
10:         print v is an articulation point
11:      end if
12:   else
13:      low[v] ← min{low[v], DFN[w]}
14:   end if
15: end for
```

by a back edge.

- $low[w] = min\{DFN[w], DFN[w]\}$, where x has a back edge to some n s.t. n *in* subtree(w)
- A non-root vertex v in G_π is an articulation point of G *iff* v has a child w such that $low[w] \geq DFN[v]$

Appendix B

Enumerating all cut edges

A bridge is defined as an edge whose removal separates the graph into two disconnected components. This problem can be solved using the algorithm presented by Tarjan [60]. Main observation is that an edge cannot be a bridge if it is contained in a cycle. Its converse is also true, i.e. an edge that is not a part of a cycle is a bridge. Bridges can be detected by the presence of the *back edges*. An edge, (u, v), is a bridge *iff* neither v nor any of its children has a back edge to either u or any of u's ancestors. This problem can be solved in a single depth first traversal of the graph. The running time of the algorithm given below is $O(VE + V^2)$.

Algorithm 6 Enumerating all bridges

Detect-Bridges(u)

1: $num \leftarrow 0$
2: $low[u] \leftarrow DFN[u] \leftarrow time \leftarrow num + 1$
3: **for all** $u \in N$ **do**
4: **if** $(u, v) \in E$ and $DFN[u] == -1$ **then**
5: call Detect-Bridges(v)
6: $low[v] \leftarrow min\{low[v], low[w]\}$
7: **if** $low[v] > DFN[u]$ **then**
8: print (u, v) is a bridge
9: **end if**
10: $low[u] \leftarrow min\{low[u], low[v]\}$
11: **else**
12: $low[u] \leftarrow min\{low[u], DFN[v]\}$
13: **end if**
14: **end for**

References

[1] Open signaling working group. 6

[2] M. Agrawal, S. R. Bailey, A. Greenberg, J. Pastor, P. Sebos, S. Seshan, K. van der Merwe, and J. Yates. Routerfarm: towards a dynamic, manageable network edge. In *ACM SIGCOMM INM Workshop*, pages 5–10, 2006. 32

[3] M. Ahuja and Y. Zhu. An efficient distributed algorithm for finding articulation points, bridges, and biconnected components in asynchronous networks. In *Proceedings of the Ninth Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 99–108, London, UK, 1989. Springer-Verlag. 46

[4] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, February 1993. 19

[5] D. Andersen. Theoretical approaches to node assignment. 1, 11, 18

[6] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient overlay networks. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 131–145, New York, NY, USA, 2001. ACM. 7

[7] D. G. Andersen. Mayday: Distributed Filtering for Internet Services. In *4th Usenix Symposium on Internet Technologies and Systems*, Seattle, WA, March 2003. 7

[8] T. Anderson, L. Peterson, S. Shenker, and J. Turner. Overcoming the internet impasse through virtualization. *Computer*, 38(4):34–41, April 2005. 1, 5, 6, 7

[9] L. Andersson and T. Madsen. Provider provisioned virtual private network (vpn) terminology. *RFC 4026*, 2005. 6

[10] L. Andersson and E. Rosen. Framework for Layer 2 Virtual Private Networks (L2VPNs). *RFC 4664*, 2006. 6

[11] B. Augustin, X. Cuvellier, B. Orgogozo, F. Viger, T. Friedman, M. Latapy, C. Magnien, and R. Teixeira. Avoiding traceroute anomalies with paris traceroute. In *Internet Measurement Conference*, pages 153–158, 2006. 22

[12] W. Augustyn and Y. Serbest. Service requirements for layer 2 provider-provisioned virtual private networks. *RFC 4665*, 2006. 6

[13] I. Avramopoulos, D. Syrivelis, J. Rexford, and Spyros L. Secure availability monitoring using stealth probes. Technical report, Princeton University Technical Report TR-769-06, 2006. 22

[14] D. Benhaddou and W. Alanqar. Layer 1 virtual private networks in multidomain next-generation networks. *Communications Magazine, IEEE*, 45(4):52–58, April 2007. 6

[15] R. Callon and M. Suzuki. A framework for layer 3 provider-provisioned virtual private networks (ppvpns). *RFC 4110*, 2005. 6

[16] A. T. Campbell, H. G. De Meer, M. E. Kounavis, K. Miki, J. B. Vicente, and D. Villela. A survey of programmable networks. *SIGCOMM Comput. Commun. Rev.*, 29(2):7–23, 1999. 6

[17] M. Carugi and D. McDysan. Service requirements for layer 3 provider-provisioned virtual private networks (ppvpns). *RFC 4031*, 2005. 6

[18] Y. Chawathe, S. McCanne, and E. A. Brewer. Rmx: Reliable multicast for heterogeneous networks. In *IN PROC. IEEE INFOCOM*, pages 795–804, 2000. 7

[19] N. M. Mosharaf Kabir Chowdhury and Raouf Boutaba. A survey of network virtualization. Computer Networks (In press), 2010. 1, 7

[20] N. M. Mosharaf Kabir Chowdhury, Muntasir Raihan Rahman, and Raouf Boutaba. Virtual network embedding with coordinated node and link mapping. In *IEEE INFOCOM*, 2009. 1, 2, 13, 18, 19, 20, 24, 29, 31, 38, 39, 41

[21] Y. Chu, S. G. Rao, S. Seshan, and H. Zhang. A case for end system multicast. In *in Proceedings of ACM Sigmetrics*, pages 1–12, 2000. 7

[22] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. An improved algorithm for matching large graphs. In *In: 3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition, Cuen*, pages 149–159, 2001. 19

[23] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, Chateau Lake Louise, Banff, Canada, October 2001. 7

[24] R. Diestel. *Graph theory*. Springer-Verlag, New York, 1997. 14, 35

[25] E. A. Dinic. Algorithm for solution of a problem of maximum flow in a network with power estimation. *Soviet Math. Doklady*, 11(1277-1280), 1970. 15

[26] J. Edmonds and R. M. Karp. Theoretical improvements in the algorithmic efficiency for network flow problems. *Journal of the ACM*, 19:248–264, 1972. 15

[27] D. Eppstein. Finding the k shortest paths. In *Proc. 35th Symp. Foundations of Computer Science*, pages 154–165. IEEE, November 1994. 19, 22

[28] D. Eppstein. Finding the k shortest paths. *SIAM J. Computing*, 28(2):652–673, 1998. 19, 22

[29] H. Eriksson. Mbone: the multicast backbone. *Commun. ACM*, 37(8):54–60, 1994. 7

[30] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multi-commodity flow problems. *SIAM Journal on Computing*, 5(4):691–703, 1976. 15

[31] J. Fan and M. Ammar. Dynamic topology configuration in service overlay networks: A study of reconfiguration policies. In *IEEE INFOCOM*, 2006. 18, 19

[32] N. Feamster, L. Gao, and J. Rexford. How to lease the Internet in your spare time. *ACM SIGCOMM Computer Communication Review*, 37(1):61–64, 2007. 6

[33] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, and J. Rexford. Netscope: traffic engineering for ip networks. *Network, IEEE*, 14(2):11–19, Mar/Apr 2000. 22

[34] P. Ferguson and G. Huston. What is a vpn? Technical report, Cisco Inc., 1999. 6

[35] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956. 15, 46

[36] A. V. Goldberg. A new max-flow algorithm. Technical report, MIT Tehnical report MIT/LCS/TM-291, 1985. 15

[37] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. *Journal of the ACM*, 35:921–940, 1988. 15

[38] A. Gupta, J. Kleinberg, A. Kumar, R. Rastogi, and B. Yener. Provisioning a virtual private network: A network design problem for multicommodity flow. In *ACM STOC*, pages 389–398, 2001. 20

[39] A. Haider, R. Potter, and A. Nakao. Challenges in resource allocation in network virtualization. *20th ITC Specialist Seminar*, 2009. 20, 22

[40] I. Houidi, W. Louati, and D. Zeghlache. A distributed virtual network mapping algorithm. In *IEEE ICC*, pages 5634–5640, 2008. 1, 2, 19, 31

[41] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, J. W. O'Toole, Jr., M. Frans, and K. James. Overcast: Reliable multicasting with an overlay network. pages 197–212, 2000. 7

[42] A. V. Karzanov. Determining the maximal flow in a network by the method of preflows. *Sov. Math. Dokl*, 1974. 15

[43] A. D. Keromytis, V. Misra, and D. Rubenstein. Sos: Secure overlay services. In *In Proceedings of ACM SIGCOMM*, pages 61–72, 2002. 7

[44] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983. 19

[45] B. Krishnamurthy, C. Wills, and Y. Zhang. On the use and performance of content distribution networks. In *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 169–182, New York, NY, USA, 2001. ACM. 7

[46] F. T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, 1999. 46

[47] J. Lischka and H. Karl. A virtual network mapping algorithm based on subgraph isomorphism detection. In *ACM SIGCOMM VISA Workshop*, pages 81–88, 2009. 19, 24

[48] J. Lu and J. Turner. Efficient mapping of virtual networks onto a shared substrate. Technical Report WUCSE-2006-35, Washington University, 2006. 1, 2, 18, 19, 20, 24, 31

[49] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *Communications Surveys & Tutorials, IEEE*, 7(2):72–93, March 2006. 7

[50] D. Niculescu. Survey of active network research. 6

[51] R. Ricci, C. Alfeld, and J. Lepreau. A solver for the network testbed mapping problem. *ACM SIGCOMM CCR*, 33(2):65–81, April 2003. 20

[52] E. Rosen. BGP/MPLS IP Virtual Private Networks (VPNs). *RFC 4365*, 2006. 6

[53] E. Rosen and Y. Rekhter. BGP/MPLS VPNs. *RFC 2547*, 1999. 6

[54] S. Savage, A. Aggarawl T. Anderson, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan. Detour: a case for informed internet routing and transport. *IEEE Micro*, 19:50–59, 1999. 7

[55] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley Sons, Inc. New York, NY, USA., 1998. 19

[56] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. In *STOC '81: Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pages 114–122, New York, NY, USA, 1981. ACM. 15

[57] L. Subramanian, I. Stoica, H. Balakrishnan, and R. H. Katz. Overqos: offering internet qos using overlays. *SIGCOMM Comput. Commun. Rev.*, 33(1):11–16, 2003. 7

[58] W. Szeto, Y. Iraqi, and R. Boutaba. A multi-commodity flow based approach to virtual network resource allocation. In *IEEE GLOBECOM*, pages 3004–3008, 2003. 20

[59] T. Takeda. Framework and requirements for layer 1 virtual private networks. *RFC 4847*, 2007. 6

[60] R. E. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. 27, 51

[61] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden. A survey of active network research. *IEEE Communications Magazine*, 35:80–86, 1997. 6

[62] D. L. Tennenhouse and D. J. Wetherall. Towards an active network architecture. *Computer Communication Review*, 26:5–18, 1996. 6

[63] Gerald Tesauro, David M. Chess, William E. Walsh, Rajarshi Das, Alla Segal, Ian Whalley, Jeffrey O. Kephart, and Steve R. White. A multi-agent systems approach to autonomic computing. *Autonomous Agents and Multiagent Systems, International Joint Conference on*, 1:464–471, 2004. 19

[64] J.S. Turner and D.E. Taylor. Diversifying the Internet. In *IEEE GLOBECOM*, volume 2, 2005. 5, 6

[65] Y. Wang et al. Virtual routers on the move: Live router migration as a network-management primitive. *ACM SIGCOMM*, (231–242), 2008. 31

[66] E.W.M. Wong, A.K.M. Chan, and T.-S.P. Yum. A taxonomy of rerouting in circuit-switched networks. *Communications Magazine, IEEE*, 37(11):116–122, Nov 1999. 21

[67] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif. Black-box and gray-box strategies for virtual machine migration. In *USENIX NSDI*, 2007. 22

[68] A. A. Youssef. A survey of active networks. Technical report, University of Maryland, Computer Science Department; CS-TR-4422, 2002. 6

[69] M. Yu, Y. Yi, J. Rexford, and M. Chiang. Rethinking virtual network embedding: substrate support for path splitting and migration. *SIGCOMM CCR*, 38(2):17–29, 2008. 1, 2, 13, 18, 19, 21, 22, 23, 24, 29, 31, 38

[70] E. Zegura. How to model an Internet. In *IEEE INFOCOM*, pages 594–602, 1996. 38

[71] Y. Zhu and M. Ammar. Algorithms for assigning substrate network resources to virtual network components. In *IEEE INFOCOM*, pages 1–12, April 2006. 1, 2, 13, 18, 19, 21, 22, 24, 31, 32, 33, 38