Resource Allocation, and Survivability in Network Virtualization Environments

by

Muntasir Raihan Rahman

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, 2010

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.

Abstract

Network virtualization can offer more flexibility and better manageability for the future Internet by allowing multiple heterogeneous virtual networks (VN) to coexist on a shared infrastructure provider (InP) network. A major challenge in this respect is the VN embedding problem that deals with the efficient mapping of virtual resources on InP network resources. Previous research focused on heuristic algorithms for the VN embedding problem assuming that the InP network remains operational at all times. In this thesis, we remove that assumption by formulating the survivable virtual network embedding (SVNE) problem and developing baseline policy heuristics and an efficient hybrid policy heuristic to solve it. The hybrid policy is based on a fast re-routing strategy and utilizes a pre-reserved quota for backup on each physical link. Our evaluation results show that our proposed heuristic for SVNE outperforms baseline heuristics in terms of long term business profit for the InP, acceptance ratio, bandwidth efficiency, and response time.

Acknowledgements

I would like to express my deep gratitude to my supervisor, Prof. Raouf Boutaba, for his support, guidance and supervision. He has been very professional and helpful in his dealings with me throughout the duration of my mmath program. I have learned a-lot about academic research, and especially writing professional papers and searching for relevant research problems which has clearly made my research experience at Waterloo worthwhile. His patience with me and his constant encouragement has helped me to see the end of the tunnel in my attempt to finish my thesis.

My sincere thanks to Prof. Richard Trefler and Prof. Alex Lopez Ortiz for serving as readers for my thesis and providing valuable feedback and constructive criticism.

I am indebted to my parents and my sister for inspiring me throughout my life and advising me at every obstacle I have faced. I am a much better person because of them.

I express deep appreciation to my research collaborators in the Waterloo Network Virtualization research group, especially Mosharaf and Issam for jointly publishing papers with me. I would also like to thank Fida, Qi, Lu and Jin for constant support and great advice.

Finally I would like to thank the University of Waterloo and the Cheriton Scholarship Committee for considering me worthwhile to get adequate research funding throughout my mmath program and for providing excellent technical support as well as an inspiring and enjoyable academic environment.

Dedication

This thesis is dedicated to my parents and my sister.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Network Virtualization

The current Internet architecture has been supporting various distributed applications and heterogeneous network technologies quite successfully. However the immense popularity of the Internet has also turned out to be its biggest obstacle to seamless growth and innovation. The rigidity of the current Internet architecture has resulted in the so called Internet Ossification problem. Due to its multi-provider nature, adopting a new architecture or modifying an existing architecture requires consensus among multiple competing stakeholders. As a result, alterations to the current Internet are limited to incremental patches and deployment of new network applications have become increasingly difficult and error-prone.

Network virtualization has been proposed as a diversifying attribute of the future internetworking paradigm that can enable seamless integration of new features to the current Internet resulting in rapid evolution of the Internet architecture [4, 5, 9]. By allowing multiple heterogeneous network architectures to cohabit on a shared physical infrastructure, network virtualization promises better flexibility, security, manageability and decreased power consumption for the Internet. In a network virtualization environment (NVE), the traditional role of the Internet Service Provider (ISP) has been divided into two separate entities: (1) the infrastructure providers (InP) who are responsible for deploying and maintaining physical network resources (routers, links etc.) and the (2) service providers

(SP) who implement various network protocols and heterogeneous network architectures on virtual networks (VNs) composed from physical network resources leased from one or more infrastructure providers.

Virtual Network Embedding (VNE) is the central resource allocation problem in network virtualization. It deals with the efficient mapping of virtual networks onto physical network resources. More specifically, for each virtual network creation request, the VNE is responsible for mapping virtual nodes onto physical nodes and virtual edges onto one or more physical paths. The VNE problem, with constraints on virtual nodes and virtual links, can be reduced to the $\mathcal{NP}$-hard multi-way separator problem, even if the schedule of VN requests is known beforehand [3]. Even when all the virtual nodes are already mapped, the virtual link embedding problem remains $\mathcal{NP}$-hard. In order to reduce the hardness of the VN embedding problem and enable efficient heuristics, existing research has been restricting the problem space in different dimensions, e.g., considering the off-line version of the problem [24, 53], ignoring either node or link requirements [8, 24], assuming infinite capacity of the substrate nodes and links to obviate admission control [8, 24, 53], and focusing on specific virtual topologies [24]. Recently the authors in [6, 22] have proposed VNE heuristics that combine the node and link embedding phases. The authors in [17] have proposed a distributed algorithm that simultaneously maps virtual nodes and virtual links without any centralized controller. However, a limitation of all these heuristics is that they assume the substrate network to be operational at all times, which is not realistic. The existing heuristics are not capable of handling substrate node and link failures, which may lead to poor performance and increased frustration for the SP.

In this thesis, we formulate the survivable virtual network embedding (SVNE) problem to incorporate single substrate link failures in VNE and propose an efficient heuristic for solving it. To the best of our knowledge, this is the first work to consider survivability strategies in the network virtualization environment.

1.2 Contributions

Our main contributions in this thesis are as follows:

1. We add survivability mechanisms to the link embedding phase of virtual network embedding using efficient recovery and protection policies that can increase the long

term business profit of the InP. We formulate the survivable virtual network embedding (SVNE) problem and provide efficient heuristics to solve it. To the best of our knowledge, this is the first work to address survivability and failure awareness issues in network virtualization. In addition to that, our algorithms are also applicable in similar multi-layer network architectures, e. g. IP-over-WDM networks.

2. We add service level agreement (SLA) assurance to the embedding process by prioritizing the restoration of failed virtual links based on customer SLA constraints with the objective of minimizing the overall impact of failure and maximizing the business profit of the InP.

3. We propose a hybrid policy heuristic to solve SVNE. This solution is based on linear programming modules and has a number of configurable parameters. For example, the InP can control the percentage of resources dedicated for backup recovery and the number of paths allowed for primary and detour flows. This gives the InP greater control over its backup resource allocation policies and enables flexibility in determining the optimal allocation based on current failure patterns. The hybrid policy also exhibits better performance compared to baseline heuristics.

4. We introduce path-flow based optimization formulations for the different recovery and protection policies. Besides reducing number of constraints and variables there are other advantages in a path flow based formulation. The path formulation allows control over the characteristics of the paths selected for embedding and protection. For instance, we can directly control the total number of paths and number of hops per path for quality of service (QoS) purposes. This is not possible with a link-flow based formulation.

5. We add end-to-end Quality of Service (QoS) guarantees to the link embedding phase of virtual network embedding using our path indexed mixed integer programming formulations.

6. We also propose heuristics for incentive compatible virtual network embedding which can tolerate rational manipulation by different parties involved in the embedding process (Appendix A).

1.3 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 presents the required background on network virtualization, the virtual network embedding problem and survivability mechanisms. It also discusses the existing literature on the virtual network embedding problem in network virtualization and some relevant literature on survivability in related domains. Our proposed formulation for SVNE and solutions along with the necessary technical results are presented in Chapter 3. Chapter 4 presents performance evaluation results. Finally, we conclude in Chapter 5 by mentioning some possible future research directions. Appendix A presents our work on incentive compatible virtual network embedding.

Chapter 2

Background and Related Work

2.1 Network Virtualization Environment

In this section, we give a high level overview of the network virtualization environment (NVE), by describing the various actors involved and their relationships. Players in the network virtualization model differ from those in a traditional network environment. The main difference comes from the decoupling of the single role of the ISP into two different roles: infrastructure providers (InP) and service providers (SP). From a purely economic point of view, this decoupling amortizes the high cost of infrastructure management by sharing capital expenditure across multiple infrastructure providers. It should be noted that NVE business roles do not map one-to-one to distinct entities, meaning the same entity can assume two different roles at the same time.

2.1.1 Infrastructure Provider

Infrastructure providers (InP) deploy, and manage physical network resources in an NVE. They own the physical infrastructure and lease resources to service providers through well defined programmable interfaces and monetary contracts. There is usually no direct negotiation between InP's and end users, instead end users subscribe to services provided by the SP's. InP's communicate and collaborate among themselves to create the complete end-to-end underlying network based on mutual business relationships.

2.1.2 Service Provider

Service providers (SP) lease resources from one or more InP's to create virtual networks and deploy customized protocols and manage the network resource allocation to offer end-to-end services to the end users. A service provider can also create child virtual networks in a recursive manner, and lease its child networks to other SP's, creating a hierarchy of networks.

2.1.3 End User

End users in the network virtualization environment can choose to use the services offered by one or more SP. It can connect to multiple service providers for different services. An end user can simultaneously connect to multiple service providers for different services. Services are offered through service level agreements (SLA) negotiated between service providers and end users.

2.1.4 Broker

Brokers act as mediators in network virtualization environments among InP's, SP's and end users. Their presence simplifies the matching of service provider's requirements by aggregating and comparing offers from multiple infrastructure providers. On the other hand, brokers also allow the end users to choose desirable services from a variety of service providers offering similar services.

2.2 Network Virtualization Concepts

In this section we define some common terms and concepts pertinent to network virtualization.

2.2.1 Physical Topology

A weighted undirected graph $G_P = (V_P, E_P)$ usually represents the physical topology, where each node in the network is a vertex $v_P \in V_P$, with a set of attributes A_{v_P}. Each

physical link between two nodes is represented by an edge $e_P \in E_P$ with an attribute set A_{e_P}.

2.2.2 Virtual Topology

The virtual topology is similarly represented by another weighted graph $G_V = (V_V, E^V)$ with corresponding attribute sets. The virtual topology is also known as a logical topology.

2.2.3 Virtual Node

A virtual node can be a virtual host or a virtual router. A virtual host acts as a packet source or sink, whereas, a virtual router forwards packets according to the routing protocols specified for the virtual topology.

2.2.4 Virtual Link

A virtual link can span over multiple physical links, i. e. , it usually corresponds to a physical path. Often a single virtual link can be mapped to multiple physical paths in-order to satisfy some of the constraints of the virtual link, e. g. bandwidth constraints that cannot be satisfied using a single path.

2.2.5 Recursion

In a network virtualization architecture, sometimes it might be necessary to create and manage one or more virtual networks on top of another virtual network creating a hierarchy of virtual networks. This is known as recursion or nesting of virtual networks.

2.3 Virtual Network Embedding

Efficient usage of substrate network resources is dependent on effective techniques for virtual network embedding. The VN embedding problem is quite challenging, due to a number of practical challenges:

2.3.1 Node and link constraints

Each VN request is associated with resource constraints, like cpu resources for the virtual nodes and bandwidth resources for the virtual links that must be met by the embedding solution. For example, to run a controlled experiment, a researcher could require 1GHz of cpu power for each virtual node and 10mbps bandwidth for each virtual link. There may also be additional constraints like geographical location constraints for nodes and delay constraints for links. The complex combination of multiple node and link constraints make the embedding problem a computationally difficult multi-constrained optimization problem.

2.3.2 Admission Control

Finite substrate resources constrains the InP to reject or postpone some VN requests in-order to meet the resource guarantess for existing virtual networks.

2.3.3 On-line Requests

VN requests arrive in an on-line fashion resulting in sub-optimal performance compared to the hypothetical case where VN request arrival sequence is known beforehand.

These properties make the VN embedding problem very difficult. In-fact the problem remains computationally intractable even if some conditions are relaxed. Due to multiple constraints, the VNE problem is in general $NP-$hard, even in the off-line case. On the other hand, traditional techniques for solving on-line problems are not practical in this case, since the characteristics of the incoming VN requests are generally unpredictable and the search space is huge when the underlying substrate network is large.

2.4 Node and Link Stress Measures

In order to quantify the resource usage of the substrate network, we use the notion of *stress*. The substrate node stress, $S_N\left(n^S\right)$ is defined as the total amount of CPU capacity

allocated to different virtual nodes hosted on the substrate node $n^S \in N^S$.

$$S_N\left(n^S\right) = \sum_{\forall n^V \uparrow n^S} c(n^V) \tag{2.1}$$

where $x \uparrow y$ denotes that the virtual node x is hosted on the substrate node y.

Similarly, the substrate link stress, $S_E\left(e^S\right)$ is defined as the total amount of bandwidth reserved for the virtual links whose substrate paths passes through the substrate link $e^S \in E^S$.

$$S_E\left(e^S\right) = \sum_{\forall e^V \uparrow e^S} b(e^V) \tag{2.2}$$

where $x \uparrow y$ denotes that the substrate path of the virtual link x passes through the substrate link y.

The definitions of *node stress* and *link stress* are similar to that in [53] with the difference that instead of using the actual amount of CPU and bandwidth resources to measure stress, the authors in [53] represent stress as the number of virtual resources mapped on top of a physical resource.

2.5 Failures

There are many types of potential failures in a multi-layer network system like the network virtualization environment (NVE). Not all of them are equally important. A given failure may be common in one domain, and irrelevant in another. Next we discuss different types of potential failures in multi-layer networks.

2.5.1 Edge Failures vs. Node Failures

First of all, there are two basic elements that can fail in a network: edges and nodes. In practice, the failure of a node is equivalent to the failure of all adjacent edges.

2.5.2 Logical Failures vs. Physical Failures

Failures can occur at the logical or physical layer. Logical failures affect the logical layer only, and are transparent to the physical layer. In contrast, physical failures not only affect the physical layer, but also propagate to the logical layer, and if the network is recursive, any number of upper layers.

2.5.3 Single vs. Multiple Failures

Next we distinguish among single and multiple failures. The single failure case is more important and common, since the failure probability is often very small, making multiple simultaneous failures a rare event. On the other hand, considering multiple failures can facilitate the behavior of large scale systems under typical or high stress.

2.5.4 Errors vs. Attacks

Finally, we can also distinguish errors and attacks. Errors represent failures of randomly chosen components in the system, which is what we will be concerned in this thesis. On the other hand, attack scenarios assume the presence of some adversary that exploits a weak point of the system. A candidate weak point could be a highly stressed node or link, since removing such nodes and links will disconnect large number of upper layer networks.

2.6 Failure Protection Mechanisms

We now describe some general techniques to increase network robustness to failures.

2.6.1 Proactive vs. Reactive

Proactive failure protection mechanisms constantly keep the system prepared for failures, before they actually occur. Typically, this can be achieved by keeping a set of pre-computed backup paths, or by applying some redundant coding. In contrast, reactive mechanisms are activated only after a failure actually occurs and is detected. Usually, they trigger a

retransmission of lost data in case of short-lived failure, or they adaptively search for a new path for long-term failure. Proactive techniques are less resource efficient but fast, whereas reactive mechanisms more resource efficient but typically slower.

2.6.2 Physical Layer vs Logical Layer

Failure protection mechanisms can be provided at different layers. In case of a physical layer protection mechanism, a physical failure is detected directly at the physical layer. The affected paths can be changed to avoid the failed link by (i) replacing the precomputed paths (proactive technique) or (ii) rerouting on the fly (reactive technique). This is in general transparent to the logical layer.

On the other hand, in case of reactive failure protection at the logical layer, a physical failure propagates to the logical layer where it is detected. The logical nodes recalculate the routes in the logical topology to avoid the failing logical link(s), which is transparent to the physical layer.

2.7 Network Survivability

Network survivability measures the ability of a network to support committed Quality of Services (QoS) in the presence of various failure scenarios. It is a generalization of the QoS guarantee that the network providers commit to their end users. In the case of failures, it becomes difficult to maintain such QoS requirements without properly pre-planning backup resources. Network survivability techniques include a set of tools to pre-plan and utilize backup resources to improve QoS in the presence of failures.

The causes of failures in networks are widespread and most failures are hard to forcast or eliminate, however it is still possible to minimize the impact of a set of specific failures by incorporating survivability strategies into the network design phase. Traditional network survivability techniques have two aspects, survivable network design and network restoration.

2.8 Existing Literature on Network Survivability

Survivable Virtual Network Embedding (SVNE) or virtual network embedding in the presence of arbitrary node and link failures is a research challenge that has yet to be addressed in the network virtualization literature. Node and link failure survivability problems have been investigated extensively for optical and multi-protocol label switched (MPLS) networks [41], and real time systems [52]. Two well known approaches for handling link failures in optical networks are protection and restoration. Protection is normally employed at the substrate network level during the design phase by provisioning backup light-paths. On the other hand restoration is done at the virtual network level by provisioning the network with additional capacity and is more reactive in nature. The key to efficient restoration mechanisms is survivable mapping in the presence of link failures. The authors in [43] mention three existing paradigms for survivable IP-over-WDM mapping algorithms based on (1) Integer Linear Programs (ILP), (2) Meta-heuristics like Genetic Algorithms (GA), Ant Colony Optimization (ACO), Tabu Search, and (3) Graph Theoretic algorithms. The most recent approach based on graph theoretic results called SMART [18, 19] is more efficient and scalable than ILP and heuristic local search approaches.

SMART repeatedly picks connected subgraphs of the logical topology and finds survivable mappings for them. It then reduces the logical topology by contracting the already mapped subgraph and continues the process. The authors in [43] continue working in this direction by exploiting duality between circuits and cuts due to the Max-flow min-cut theorem in Combinatorial Optimization [30]. They propose primal and dual algorithms that extend SMART (called CIRCUIT-SMART and CUTSET-SMART) and develop some heuristics to speed up their algorithms. Recently the authors in [20] extended the Max-flow min-cut theorem for multi-layer networks. They proposed new connectivity metrics suited for multi-layer networks and developed some heuristics for maximizing connectivity in the logical layer using ILP formulations and subsequent LP relaxations and rounding techniques.

Our work on survivable virtual network embedding (SVNE) differs in a number of aspects, due to unique challenges introduced by the network virtualization environment. First, the VNE problem is on-line in nature, whereas the survivable logical topology design problem in optical and multi-protocol label switched (MPLS) networks [19, 20, 43] is off-line. Secondly, in NVEs, we need to ensure that all virtual links are intact in the presence

of failures. This restriction is not present, for example, in optical networks where the goal is to only ensure that all nodes remain connected in the presence of failures, even if they are not connected via a direct overlay link. Our contribution also differs from existing work in terms of the objective formulation. Our aim is to develop a survivable virtual network embedding solution that simultaneously maximizes the long term revenue for the InP, and minimizes the long term penalty incurred by the InP due to service violations caused by failures. This dual nature of the objective function in the presence of failures is absent both in the existing research on optical and mpls networking domains and the existing VNE heuristics. Another novel aspect of our work is that we utilize path-flow based optimization formulations for solving the SVNE problem. The path formulation allows control over the characteristics of the paths selected for embedding and survivability against failures. For instance, we can directly control the total number of paths, number of hops per path, and impose delay constraints on virtual links for QoS purposes. This is not possible with a link-flow based formulation which has been used for the previous VNE heuristics [6, 22, 49, 53].

In optical networks, end-to-end connection requests arrive on-line and are processed as soon as they arrive. For a VN request, we have to guarantee survivability of all the VN links simultaneously, which makes the problem harder. We differentiate between *Weak* and *Strong* survivability in the context of SVNE. Weak survivability only ensures that the virtual nodes will stay connected in the presence of failures. Strong survivability guarantees that the original VN topology remains intact in the presence of failures. Failures in the underlying physical network can give rise to complex multi-layer failures in the network virtualization environment. Any such failure can effectively cause a cascading series of errors in the virtual networks directly hosted on those substrate network components, and possibly in many others that are recursively designed. In NVE, we require strong survivability since in the basic revenue model for NVE, the service provider pays an amount that is proportional to the resource (cpu for nodes, bandwidth for links) and VN topology requirements of the virtual network request. This means that provisioning of backup resources is essential in an NVE, since without backup provisioning we can only ensure weak survivability.

2.9 Existing Literature on Virtual Network Embedding

The VNE problem is similar to the previous works on embedding Virtual Private Networks (VPNs) in a shared provider topology and the network testbed mapping problem [14, 38]. However, a typical VPN request consists only of bandwidth requirements, specified in terms of a traffic matrix, without any constraint on its nodes. As a result, most VPN design algorithms come down to finding paths for source/destination pairs. On the other hand, the *Assign* algorithm [38] used in Emulab testbed considers bandwidth constraints alongside constraints on exclusive use of nodes, *i.e.* different VNs cannot share a substrate node. But in network virtualization, there are capacity and placement requirements on both virtual nodes and virtual links; in addition, substrate nodes and links can be shared by multiple VNs.

All the VNE algorithms come down to two basic phases: (i) assigning virtual nodes using some greedy heuristics, e.g., assign virtual nodes with higher processing requirements to substrate nodes with more available resources [49, 53]; (ii) embedding virtual links onto substrate paths using shortest path algorithms [53] in case of unsplittable flows, or using *multi-commodity flow problem* solvers in case of splittable flows [42,49]. The authors in [17] have proposed a distributed algorithm that simultaneously maps virtual nodes and virtual links without any centralized controller.

The VNE problem is is $\mathcal{NP}-$hard [3], so most research on this problem has focused on approximation algorithms or fast heuristics. In [53], the authors proposed simple greedy heuristic algorithms for the VNE problem and developed some additional heuristics to improve the performance of their algorithms. A different approach was considered in [49], where the authors proposed to modify the physical substrate network to allow heterogeneous virtual networks to be easily accommodated; this approach in fact shifts the computational burden from the virtual networks to the substrate networks.

In order to reduce the hardness of the VN assignment problem and to enable efficient heuristics, existing research has been restricting the problem space in different dimensions, which include:

1. Considering offline version of the problem (i.e., all the VN requests are known in advance) [24, 53];

2. Ignoring either node requirements or link requirements [8, 24];

3. Assuming infinite capacity of the substrate nodes and links to obviate admission control [8, 24, 53]; and

4. Focusing on specific VN topologies [24].

The authors in [49] consider all these issues, except for the location constraints on the virtual nodes, by envisioning support from the substrate network through node and link migration as well as multi-path routing.

Most of the existing algorithms can clearly be separated into two basic phases:

1. Assigning virtual nodes using some greedy heuristics, e.g., assign virtual nodes with higher processing requirements to substrate nodes with more available resources [49, 53]; and

2. Embedding virtual links onto substrate paths using shortest path algorithms [53] in case of unsplittable flows, or using *multi-commodity flow* algorithms in case of splittable flows [42, 49].

Recently the authors in [6, 22] have proposed VNE heuristics that combine the node and link embedding phases which lead to single phase solutions with increased revenue, lower cost and better acceptance ratio.

Chapter 3

SVNE Problem Formulation and Solutions

In this section, we provide a mathematical formulation of survivable virtual network embedding (SVNE) as an extension of the VNE problem. We then devise efficient heuristics to solve SVNE. Since we deal with substrate link failures in this thesis, our main focus is on the second phase of VNE, that is the link embedding phase. For node embedding, we use the existing heuristics proposed in the literature. As a result our approach to on-line SVNE for each incoming VN request is as follows:

- Node Embedding: Greedy [49, 53], Mixed Integer Programming [6].
- Link Embedding: Add survivability policies to handle arbitrary substrate link failures. [Thesis Contribution].

The existing node embedding heuristics and path selection mechanisms used in our SVNE solutions are described in subsequent sections of this chapter.

3.1 Substrate Network

We model the substrate network as a weighted graph $G^S(N^S, E^S)$, where N^S and E^S represent the set of substrate nodes and links respectively. Each substrate node $x \in N^S$

has an associated cpu capacity $cpu(x)$ and a geographical location value $loc(x)$. A substrate link $s = (s_x, s_y) \in E^S$ between substrate nodes $s_x, s_y \in N^S$ has a bandwidth capacity $b(s)$. From now on, we denote the endpoints of any substrate link s as s_x and s_y.

3.2 Virtual Network Request

A Virtual Network (VN) request $G^V(N^V, E^V)$ is also modeled as a weighted graph. VN requests are associated with constraints and QoS requirements embodied into service level agreements (SLA) [2]. A virtual node $y \in N^V$ has a cpu capacity requirement $cpu(y)$ and geographical location requirement $loc(y)$. A virtual link $v \in E^V$ is characterized by a bandwidth capacity requirement $b(v)$ and a delay constraint $d(v)$. $d(v)$ is used to preselect the set of admissible simple substrate paths[1] that can be used to embed v. An example of a typical substrate network and two virtual network topologies are shown in figure 3.1. The numerical values beside the substrate nodes and links represent cpu and bandwidth constraints of those nodes and links respectively.

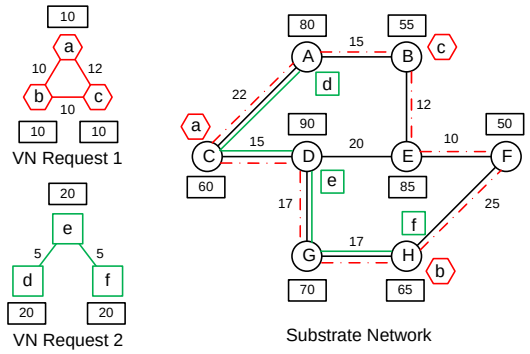

Figure 3.1: Mapping of VN requests onto a shared substrate network.

3.3 Resource Usage Metrics

We assume that substrate network resources are finite. As a result, the amount of residual substrate network resources diminishes as new VN requests are processed. We keep track

[1] A substrate path that repeats no substrate node.

of the residual substrate node and link capacities in order to make sure we don't accept a request unless there are adequate resources to serve it. The residual capacity of a substrate node $x \in N^S$ is defined as:

$$R_N(x) = cpu(x) - \sum_{y \in V(x)} cpu(y), \tag{3.1}$$

where $V(x)$ denotes the set of virtual nodes mapped onto x. Similarly the residual capacity of a substrate link $s \in E^S$ is defined as:

$$R_E(s) = b(s) - \sum_{\{v : \exists p \in \Gamma_E(v),\, s \in p\}} b(v), \tag{3.2}$$

where, $\Gamma_E(v)$ defines the set of paths in the InP that are used to embed the virtual link v (Section 3.4). The residual capacity values are updated after each new VN request has been successfully mapped on top of the substrate network as long as there remains adequate residual resources. The values are also updated after a VN departs and link failure occurrences and repairs.

In order to protect against single substrate link failures, we dedicate a certain percentage of bandwidth resources on each substrate link for backup purposes. For a substrate link s with total bandwidth $b(s)$, $\alpha(s)b(s)$ bandwidth is reserved for primary flows, whereas $\beta(s)b(s)$ is reserved for backup flows, where $\alpha(s) + \beta(s) = 1$. The residual bandwidth measure is accordingly decomposed into primary and backup residual bandwidth measures $\mathcal{R}_\alpha(s)$ and $\mathcal{R}_\beta(s)$ respectively. As a result, we need to keep track of these two residual bandwidth measures separately.

3.4 VN Embedding

The VN Embedding process refers to the mapping of the virtual network topology (logical topology) on top of the substrate network topology (physical topology) subject to certain constraints. The constraints are normally manifested in terms of the residual resource availability of the substrate network and the QoS parameters specified by the VN request.

An example of a VN embedding can be seen in figure 3.1. From graph theoretic standpoint, the VN embedding process can be separated into two separate stages:

1-Node Embedding Phase: Each virtual node from a VN request is mapped to a single distinct substrate node by a one-to-one mapping:

$$\Gamma_N : N^V \leftarrow N^S, \tag{3.3}$$

such that $\Gamma_N(x) = \Gamma_N(y), \ \ iff \ \ x = y \ \ \forall x, y \in N^V$, subject to the cpu capacity constraints: $cpu(x) \leq R_N(\Gamma_N(x)) \ \ \forall x \in N^V$.

2-Link Embedding Phase: Each virtual link is mapped to either an unsplittable substrate path or a splittable multi-commodity flow based set of paths between the substrate nodes corresponding to the endpoints of the virtual link. Mathematically, we have a mapping:

$$\Gamma_E : E^V \leftarrow \mathcal{P}^S, \tag{3.4}$$

such that $\forall v = (v_x, v_y) \in E^V$, and $\mathcal{P}^S$ is the set of simple paths of G^S. We have $\Gamma_E(v) \subseteq \mathcal{P}(\Gamma_N(v_x), \Gamma_N(v_y))$, subject to the bandwidth capacity constraints: $b(v) \leq R_E(p), \ \ \forall p \in \Gamma_E(e^V)$, where $\mathcal{P}(z, w)$ denotes the set of simple substrate paths between substrate nodes z and w, and $R_E(p) = \min_{s \in p} R_E(s)$. For any virtual link $v \in E^V$, we specify the set of QoS constrained substrate paths for v as $\mathcal{P}(v) = \{p \in \mathcal{P}^S | delay(p) \leq d(v)\}$.

3.5 An Example

To clarify the network model, we refer to the scenario in figure 3.1. The substrate network is G^S with $N^S = \{A, B, C, D, E, F, G, H\}$, $E^S = \{AB, AC, CD, DE, BE, EF, DG, GH, FH\}$. The virtual network VN Request 1 is G^V with $N^V = \{a, b, c\}$ and $E^V = \{ab, bc, ac\}$. We have the embedding of G^V on G^S as Γ with $\Gamma_N(a) = C, \Gamma_N(b) = H, \Gamma_N(c) = B$ and $\Gamma_E(ab) = \{CD, DG, GH\}, \Gamma_E(bc) = \{HF, FE, EB\}, \Gamma_E(ac) = \{CA, AB\}$.

3.6 Penalty Function and Business Utility for InP

An SP negotiates a Service Level Agreement (SLA) with the InP for uninterrupted service throughout the lifetime of its requested VN. If the SLA contract is violated due to a substrate resource failure, then this results in frustration on part of the SP and subsequent penalty for the InP based on the level of frustration of the SP. Each SP owning a VN is characterized by a Service class which is represented by a function $\mathcal{S}_j(db)$, where $j \in \{1, 2, \ldots, C\}$ and C denotes the number of distinct service classes and db denotes the bandwidth differential, that is the difference between requested bandwidth and the bandwidth granted by the InP. We can model $\mathcal{S}_j(db)$ as an increasing function, however for simplicity, we assume that the function takes the shape of a step function, that is for $db < T_B$, $\mathcal{S}_j(db) = 0$, and for $t > T_B$, $\mathcal{S}_j(db) = P$. We call T_B the *frustration threshhold* for the service class j. Therefore the set of all SP is partitioned into equivalence classes based on their respective service class associations. We denote the mapping between VNs and service classes as $\varphi(.)$, where $\varphi(i) = j$ means that VN i is associated with service class j. Since we have reserved a percentage of bandwidth on each substrate link for backups, it cannot be ensured that all the SP will retain their complete VN topology when a failure occurs. In that case our objective will be to minimize the total penalty incurred due to SP frustration. For each $v \in E^V$ and service class j for a VN, we will denote $\mathcal{S}_j(v)$ as the penalty incurred due to service disruption.

Assume that a FAE event FAE(l) occurs where the substrate link $l \in E^S$ fails with Mean Time To Failure $MTTF(l)$ and Mean Time to Repair $MTTR(l)$. We define the availability of a substrate link l as

$$\mathcal{A}(l) = \frac{MTTF(l)}{MTTF(l) + MTTR(l)} \tag{3.5}$$

.

3.7 Network State Representation

In a generalized setting we can represent a network state as a tuple

$$(\{T_N(x)\}_{x \in V^S}, \{T_L(s)\}_{s \in E^S}, \mathcal{F}_N, \mathcal{F}_L\}), \tag{3.6}$$

where $T_N(x)$ denotes the number of virtual nodes embedded on the substrate node $x \in V^S$, $T_L(s)$ denotes the number of virtual links whose corresponding substrate paths pass through the substrate link $s \in E^S$. $\mathcal{F}_N$ is a $0-1$ vector denoting which substrate nodes are up and which have failed, that is $\mathcal{F}_N[x] = 0$ if $x \in V^S$ has failed and 1 otherwise. $\mathcal{F}_L$ is defined similarly. The network state changes due the occurrence of any of the following events:

- A new VN request arrives [VN Arrival Event (VNAE)].
- An existing VN request expires [VN Departure Event (VNDE)].
- A network component failure occurs [Failure Occurrence Event (FOE)].
- A failed network component recovers [Failure Repair Event (FRE)].

Since in this thesis we are restricted to single substrate link failures, the $\mathcal{F}_N$ vector has the form: $\mathcal{F}_N[x] = 1 \; \forall x \in V^S$, and the $\mathcal{F}_L$ vector always has a single 1 entry and 0 elsewhere.

3.8 Formulation of SVNE

We represent the input to SVNE as a tuple $< G^S, G^V, j, l, \{\alpha(s)\}_{s \in E^S} >$, where G^S and G^V represent the substrate and virtual networks respectively, j represents the service class of the SP owning G^V, $l \in E^S$ is the failed substrate link, and $\beta(s) = 1 - \alpha(s)$, such that $\beta(s)$ represents the percentage of bandwidth on each substrate link s reserved for backups. Let $\Pi(G^V)$ denote the revenue generated from G^V, where

$$\Pi(G^V) = T(G^V)[C_1 \sum_{v \in E^V} b\,(v) + C_2 \sum_{x \in N^V} cpu\,(x)] \tag{3.7}$$

C_1 and C_2 are weight factors which represent the relative importance of bandwidth and cpu to the generated revenue respectively. $T(G^V)$ represents the lifetime of the VN characterized by G^V. Each service class j is associated with a penalty function $\mathcal{S}_j(.)$, where $\mathcal{S}_j(v)$ represents the monetary penalty incurred if the bandwidth contract of virtual link v is violated.

Let $\mathcal{V}$ denote the set of all virtual links affected by the failure of l. Then the expected total penalty incurred by the InP to the corresponding SP is:

$$\mathcal{X}(G^V; l) = MTTR(l) \sum_{v \in \mathcal{V} \cap E^V} \mathcal{S}_j(v) \frac{db(v)}{b(v)} \tag{3.8}$$

$MTTR(l)$ is the mean time to repair for l. The difference between the bandwidth requested for v, and the actual bandwidth supplied by the InP is represented as $db(v)$. Let $G_1^V, G_2^V, G_3^V, \ldots$ be the sequence of VN requests, and $l_1, l_2, l_3, \ldots$ be the sequence of substrate link failure events. Then the objective of SVNE is to maximize long term business profit expressed as:

$$\Pi_\infty = \sum_{p=1}^{\infty} \sum_{q=1}^{\infty} [\Pi(G_q^V) - \mathcal{X}(G_q^V; l_p)] \tag{3.9}$$

3.9 Restoration and Protection Models

For fast protection against substrate link failures, we employ two types of restoration mechanisms in this thesis, namely *l*ocal (span) restoration and path (end-to-end) restoration. In the existing literature on survivable topology design, both of these mechanisms fall under the category of fast restoration mechanisms, due to their low restoration latency. The main objective in this thesis is to provide link embedding heuristics with fast restoration.

3.9.1 Link Restoration and Protection

For protecting a substrate path p corresponding to a virtual link against single link failures, we associate a primary path $W(p)$ and for each substrate link $e \in p$, a local backup detour $B_e(p)$. So for a substrate path with k substrate links, there will be k link detours for fast restoration against single link failures. From now one, when we refer to a path p in this model, it will consist of $\{W(p), B_e(p, \forall e \in p)\}$. It should also be mentioned that backup detours for different substrate paths can share bandwidth on their common substrate links. Link restoration is also known as local restoration due to the localized fault tolerance mechanism around each substrate link.

3.9.2 Path Restoration and Protection

In the simplest approach to path restoration, namely 1 : 1 protection, a connection p consists of a primary working path and a link disjoint backup path. Our approach to path restoration is more sophisticated in that we use a survivable version of the multi-commodity flow problem for path restoration in link embedding [44]. A survivable flow from a substrate node u to v consists of a primary flow of value f among the paths from u to v, and a distinct secondary flow of the same value f in such a way that both flows pass through link disjoint paths. Path restoration is also known as global restoration due to its end-to-end fault tolerance nature.

3.9.3 VN Recovery Policies

A VN *R*ecovery Policy is a set of rules that specify what action to take when a substrate link fails. We consider the following types of recovery policies in this thesis.

- Policy 1 [BLIND]: Whenever a substrate link l fails, recompute the entire link embedding phase considering the new substrate graph $G_l^S(V^S, E^S \setminus \{l\})$, which is the residual substrate graph obtained after removing the failed link.
- Policy 2 [PROACTIVE]: Pre-reserving additional backup bandwidth during VN embedding.
- Policy 3 [HYBRID]:
  - Proactive Computation of a set of candidate backup detours for each substrate link of the substrate graph using a path selection heuristic (BDC).
  - For each new VN arrival, perform a multi-commodity flow based link embedding (MFLE).
  - For each new failure event, perform the backup detour optimization (BDO).
  - Sub-policy 2.1: When a substrate failure occurs, only use the β percentage of substrate links for the backup detours.
  - Sub-policy 2.2: When a substrate failure occurs, use total residual capacity on substrate links for the backup detours.

The following sections provide mathematical formulations and heuristics for these different recovery and protection policies.

3.10 BLIND Policy Heuristic for SVNE

The BLIND policy is the simplest scheme among all the policies, hence the name. This policy is oblivious to any underlying structure of the problem space and the failure pattern. Whenever a substrate link fails, the BLIND policy simply recomputes a new link embedding for each VN affected by the substrate link failure. Although this policy seems simple, it has a high recovery complexity and reconfiguration cost, since even though the substrate link failure will only affect a localized portion of the embedding of a VN, it still recomputes the entire embedding.

The idea of the blind policy can be summarized in the following algorithm:

Algorithm 1 Blind Recovery Policy

1: **procedure** BRP($G^S, \mathcal{G}_l$)
2: **for all** VN's in $\mathcal{G}_l$ **do**
3: recompute link embedding on $G^S_l(V^S, E^S \setminus \{l\})$.
4: **end for**
5: **end procedure**

Here $\mathcal{G}_l$ refers to the set of VN's affected by the failure of l. We can also consider a slightly enhanced version of the blind policy, where the affected VN's are ordered in terms of potential for revenue generation and the total penalty that would be incurred if the VN is affected by the failure.

3.11 PROACTIVE Policy Heuristic for SVNE

The PROACTIVE policy protects each virtual link using a survivable version of the multi-commodity flow problem [44]. For each virtual link v, we send a primary flow of value $b(v)$ and also a secondary flow of value $b(v)$ among the QoS constrained paths allowed for v. To protect against single substrate link failures, we have to ensure that primary and

secondary flows are edge disjoint. We formulate the problem as a mixed integer program in the following manner:

PROACTIVE_MIP_LE

Minimize

-Objective Function

$$\sum_{v \in E^V} \mathcal{S}_j(v)[1 - \sum_{p \in \mathcal{P}(v)} \frac{b_2(p,v)}{b(v)}] + \sum_{v \in E^V, p \in \mathcal{P}(v)} b_1(p,v) + b_2(p,v) \tag{3.10}$$

Subject to

-Primary and Secondary Capacity Constraints

$$\sum_{v \in E^V, p \in \mathcal{P}(v)} \delta_s(p) b_1(p,v) \leq \mathcal{R}_\alpha(s) \;\; \forall s \in E^S \tag{3.11}$$

$$\sum_{v \in E^V, p \in \mathcal{P}(v)} \delta_s(p) b_2(p,v) \leq \mathcal{R}_\beta(s) \;\; \forall s \in E^S \tag{3.12}$$

-Primary and Secondary Bandwidth Constraints

$$\sum_{p \in \mathcal{P}(v)} b_1(p,v) = b(v), \;\; \forall v \in E^V \tag{3.13}$$

$$\sum_{p \in \mathcal{P}(v)} b_2(p,v) \leq b(v), \;\; \forall v \in E^V \tag{3.14}$$

-Disjoint Constraints

$$b_1(p,v) \leq b(v)\sigma_1(p,v), \ \ \forall v \in E^V, \forall p \in \mathcal{P}(v) \tag{3.15}$$

$$b_2(p,v) \leq b(v)\sigma_2(p,v), \ \ \forall v \in E^V, \forall p \in \mathcal{P}(v) \tag{3.16}$$

$$\delta_s(p)\delta_s(q)[\sigma_1(p,v) + \sigma_2(p,v)] \leq 1, \ \ \forall s \in E^S \tag{3.17}$$

-Variables

$$\sigma_1(p,v) \in \{0,1\}, \ \ \forall v \in E^V, \forall p \in \mathcal{P}(v) \tag{3.18}$$

$$\sigma_2(p,v) \in \{0,1\}, \ \ \forall v \in E^V, \forall p \in \mathcal{P}(v) \tag{3.19}$$

$$b_1(p,v) \geq 0, \ \ \forall v \in E^V, \forall p \in \mathcal{P}(v) \tag{3.20}$$

$$b_2(p,v) \geq 0, \ \ \forall v \in E^V, \forall p \in \mathcal{P}(v) \tag{3.21}$$

3.11.1 Remarks

- j denotes the service class associated with the VN. Subsequently $\mathcal{S}_j(v)$ denotes the penalty incurred for violating the bandwidth reservation for a virtual link v belonging to a VN of service type j.
- $\delta_s(p)$ is a link-path indicator variable, i. e. $\delta_s(p) = 1$ if $s \in p$, 0 otherwise.
- $b_1(p,v)$ and $b_2(p,v)$ represent the primary and backup flows on the simple path p for the virtual link v.
- The objective function 3.10 has two parts. The first part is for minimizing the total penalty incurred due to bandwidth violations, whereas the second part is concerned with minimizing the overall substrate network usage for primary and secondary flows.
- Constraints 3.11 are the primary and secondary capacity constraints and they specify that for each substrate link, the overall bandwidth used for primary and secondary flows must be within the primary and backup residual capacities of that substrate link respectively.

- Constraints 3.13 are bandwidth constraints for primary and secondary flows respectively.

- Constraints 3.15 represent the disjoint constraints. They are expressed in terms of the two integer variables σ_1 and σ_2. The third constraint in this set of constraints enforces that only one of them can take the value 1. If σ_1 is 0, then the first constraint forces b_1 to 0 also. However if σ_1 is 1, then the first constraint is trivially satisfied. The case for σ_2 is similar.

3.11.2 Solution Methodologies

PROACTIVE_MIP_LE is a mixed integer program, and hence $NP-$hard to solve. The usual approach is to relax the integer constraints and solve the relaxed Linear Program (LP) to obtain a fast heuristic. However the integrality of the MIP stems from the disjointness constraint 3.15 which forces the primary and backup flows to pass through link disjoint paths. It should also be noticed that we have a dedicated percentage of bandwidth resources for backups on each substrate link through the $\alpha(s), \beta(s)$ values for each substrate link $s \in E^S$. This *s*eparation property readily leads towards a fast simple heuristic using two sequential LP's as follows:

PROACTIVE_LP_LE_P

-Objective Function

Minimize

$$\sum_{v \in E^V, p \in \mathcal{P}(v)} b_1(p, v) \tag{3.22}$$

Subject to

$$\sum_{v \in E^V, p \in \mathcal{P}(v)} \delta_s(p) b_1(p, v) \leq \mathcal{R}_\alpha(s) \;\; \forall s \in E^S \tag{3.23}$$

$$\sum_{p \in \mathcal{P}(v)} b_1(p, v) = b(v), \;\; \forall v \in E^V \tag{3.24}$$

We define a boolean variable $\varphi(s), \forall s \in E^S$ which keeps track of the substrate links that have been used for sending primary flow. These values are then used in the second LP to avoid conflicts between primary and backup flows on the same substrate link.

PROACTIVE_LP_LE_B

-Objective Function

Minimize

$$\sum_{v \in E^V} \mathcal{S}_j(v)[1 - \sum_{p \in \mathcal{P}(v)} \frac{b_2(p,v)}{b(v)}] + \sum_{v \in E^V, p \in \mathcal{P}(v)} b_2(p,v) \tag{3.25}$$

Subject to

$$\sum_{v \in E^V, p \in \mathcal{P}(v)} \delta_s(p) b_2(p,v) \leq (1 - \varphi(s)) \mathcal{R}_\beta(s) \;\; \forall s \in E^S \tag{3.26}$$

$$\sum_{p \in \mathcal{P}(v)} b_2(p,v) \leq b(v), \;\; \forall v \in E^V \tag{3.27}$$

Remark

- It should be noted that we have multiplied the term $(1 - \varphi(s))$ to the right hand side of the first constraint in 3.26. If a substrate link s has been used for a primary flow, then $\varphi(s)$ will be 1, forcing the right hand side of that constraint to be 0. This ensures the disjointness of the primary and secondary flows.

We now have the following polynomial time LP based heuristic:

Algorithm 2 LP Based Heuristic for Proactive Recovery Policy

```
1: procedure LPHPP(G^S, G^V)
2:     Solve PROACTIVE_LP_LE_P
3:     for all s ∈ E^S do
4:         φ(s) = 0
5:     end for
6:     for all v ∈ E^V do
7:         for all p ∈ P(v) do
8:             if b_1(p, v) > 0 then
9:                 φ(s) = 1
10:            end if
11:        end for
12:    end for
13:    Solve PROACTIVE_LP_LE_B
14: end procedure
```

3.12 HYBRID Policy Heuristic for SVNE

We propose a *hybrid policy* heuristic for solving SVNE. The heuristic consists of three separate phases. In the first phase, before any VN request arrives, the InP pro-actively computes a set of possible backup detours for each substrate link using a path selection algorithm. Therefore, for each substrate link l, we have a set D_l of candidate backup detours. The InP is free to utilize any path selection algorithm that suits its purposes, e. g. k-shortest path algorithm [45], column generation or primal dual methods [1]. The second phase is invoked when a VN request arrives. In this phase, the InP performs a node embedding using existing heuristics [6, 53] and a multi-commodity flow based link embedding, that we denote as HYBRID_LP_LE. Finally, in the event of a substrate link failure, a reactive backup detour optimization solution HYBRID_LP_BDO is invoked which reroutes the affected bandwidth along candidate backup detours selected in the first phase. The pseudo-code for the hybrid policy is shown in the following algorithm.

Algorithm 3 Hybrid Policy Heuristic

procedure HRP($G^S(N^S, E^S)$)
  for all $s \in E^S$ **do**
    pre-compute candidate detour set $\mathcal{D}_s$.
  end for
  for all event arrivals **do**
    if event type == VN arrival **then**
      compute node embedding for VN $G^V(N^V, E^V)$.
      solve **HYBRID_LP_LE**.
      update $\mathcal{R}_\alpha(s), \forall s$ involved in HYBRID_LP_LE.
    end if
    if event type == Failure arrival **then**
      solve **HYBRID_LP_BDO**.
      update $\mathcal{R}_\beta(s), \forall s$ involved in HYBRID_LP_BDO.
    end if
  end for
end procedure

We now show the formulations of HYBRID_LP_LE and HYBRID_LP_BDO.

3.12.1 Formulation of HYBRID_LP_LE

In this phase we use a path based multi-commodity flow formulation to embed all the virtual links simultaneously. For each pair $(x, y) \in V^S \times V^S$, we have a set of preselected end-to-end paths $\mathcal{P}(x, y)$. For a virtual link $v \in E^V$, we denote $\mathcal{P}(v) = \mathcal{P}(v_x, v_y)$ as the set of pre-selected QoS constrained simple paths for embedding v, where v_x and v_y are the end-points of v. Since the node embedding phase precedes the link embedding phase, we already know which virtual node is mapped to which substrate node. For any virtual link $v = (x', y') \in E^V$, we denote this as $x' \rightarrow \Gamma_N(x') = x$ and $y' \rightarrow \Gamma_N(y') = y$. HYBRID_LP_LE can be expressed as the following linear program:

HYBRID_LP_LE

-Objective Function

$$\text{Minimize} \sum_{v \in E^V} \sum_{p \in \mathcal{P}(v)} b(p, v) \tag{3.28}$$

Subject to

-Primary Capacity Constraint

$$\sum_{v \in E^V} \sum_{p \in \mathcal{P}(v)} \delta_s(p) b(p,v) \leq \mathcal{R}_\alpha(s), \ \ \forall s \in E^S. \tag{3.29}$$

-Primary Bandwidth Constraint

$$\sum_{p \in \mathcal{P}(v)} b(p,v) = b(v), \ \ \forall v \in E^V \tag{3.30}$$

Remarks

1. $\delta_s(p)$ is the link-path indicator variable, that is, $\delta_s(p) = 1$ if $s \in p$, 0 otherwise.
2. The objective function 3.28 corresponds to the revenue function Π for the VN.
3. $b(p,v)$ is the amount of bandwidth allocated on path p for virtual link v. A strictly positive value for $b(p,v)$ will indicate that p is a substrate path used for v. The values of $b(p,v)$ are stored and later used in the subsequent phase of the heuristic.
4. Constraint 3.29 is the primary capacity constraint which states that the total primary bandwidth allocated for all virtual links must be within the primary residual capacity of each substrate link.
5. Constraint 3.30 is the primary bandwidth constraint which specifies that the total bandwidth requirement of each virtual link must be distributed among all the QoS constrained paths allowed for that virtual link.

3.12.2 Formulation of HYBRID_LP_BDO

HYBRID_LP_BDO can be expressed as the following linear program.

HYBRID_LP_BDO

-Objective Function

$$\text{Minimize} \sum_{v \in E^V} \mathcal{S}_j(v) \sum_{p \in \mathcal{P}(v)} \delta_l(p) \lceil b(p,v) \rceil [1 - \sum_{d \in \mathcal{P}_l} \frac{b(d,p,v)}{b(p,v)}] \tag{3.31}$$

Subject to

-Backup Capacity Constraint

$$\sum_{v \in E^V, p \in \mathcal{P}(v), d \in \mathcal{D}_l} \lceil b(p,v) \rceil \ \delta_s(d) \ b(d,p,v) \delta_l(p) \leq \mathcal{R}_\beta(s) \forall s \in E^S \tag{3.32}$$

-Recovery Constraint

$$\sum_{d \in \mathcal{D}_l, v \in E^V, p \in \mathcal{P}(v)} \delta_l(p) \ \lceil b(p,v) \rceil \ \delta_s(d) \ b(d,p,v) \leq \sum_{d \in \mathcal{D}_l, v \in E^V, p \in \mathcal{P}(v)} \delta_l(p) \ b(p,v) \tag{3.33}$$

Remarks

1. j represents the service class associated with the VN. Subsequently $\mathcal{S}_j(v)$ denotes the penalty incurred for violating the bandwidth reservation for a virtual link v belonging to a VN of service type j.

2. $\lceil x \rceil$ denotes the ceiling of x, that is $\lceil x \rceil = 1$ iff $x > 0$. So $\lceil b(p,v) \rceil = 1$ indicates that p is a path used for the embedding of v. Note that the $b(p,v)$ values are calculated and stored in the HYBRID_LP_LE phase.

3. For the failed substrate link l, we have the set of candidate backup detours, $\mathcal{D}_l = \mathcal{P}(l_x, l_y) \setminus \{l\}$.

4. $b(d,p,v)$ denotes the amount of rerouted bandwidth on detour $d \in \mathcal{D}_l$ for $b(p,v)$, that is for the primary path p allocated for virtual link v.

5. The objective (equation 3.31) refers to the penalty function formulated in equation 3.8.

6. Constraint 3.32 is the backup capacity constraint which states that the total backup flow on all the detours passing through a substrate link must be within the backup residual capacity of that substrate link.

7. Constraint 3.33 is the recovery constraint and it signifies that the total disrupted primary bandwidth must be allocated along the precomputed set of detours. The objective function ensures that the virtual links that have higher penalty values will be given priority during the recovery.

Discussion

Both HYBRID_LP_LE and HYBRID_LP_BDO are linear programs, as a result our proposed HYBRID policy is a polynomial time heuristic for SVNE. Another important feature of HYBRID is that it decouples primary and backup bandwidth provisioning. As a result, we don't need complex disjoint constraints in our solution which would have resulted in a hard mixed integer program. The objective functions of HYBRID_LP_LE and HYBRID_LP_BDO jointly solve the long term objective of SVNE as expressed in equation 3.9.

3.13 Heuristics for Node Embedding

3.13.1 Greedy Node Embedding

The main advantage of a greedy node embedding heuristic is that it is simple and cost efficient, in contrast to iterative methods or meta-optimization techniques, e. g. simulated annealing. The greedy algorithm maps virtual nodes to substrate nodes with maximum residual substrate resources in order to minimize the use of resources at bottleneck nodes and links [49, 53]. The metric quantifying available substrate resources is as follows:

$$H(x) = R_N(x) \sum_{l \in L(x)} b(l), \tag{3.34}$$

where $L(x)$ is the set of links adjacent to x, and $R_N(x)$ is the residual cpu capacity of x. This metric leads to the following greedy node embedding algorithm, which assumes batch processing, i. e. the InP collects VN requests at the end of a fixed time interval, allocates them simultaneously. The algorithm can be easily converted to a pure online algorithm.

Algorithm 4 Greedy Node Embedding algorithm

1: **procedure** GREEDY NODE EMBEDDING($G^V = (N^V, E^V)$)
2: Sort VN requests according to revenue.
3: exit if no requests left.
4: Take the request with largest revenue.
5: Find the set of substrate nodes S that satisfy restrictions and available cpu capacity. if $S = \{\}$, then exit.
6: For each virtual node $n \in G^V$, find the substrate node $x = argmax_x H(x)$. Map n to x.
7: **end procedure**

3.13.2 D-ViNE Algorithm

In this section, we describe a node embedding heuristic based on a mixed integer programming formulation that maximizes correlation between node and link embedding phases in order to increase revenue and minimize cost [6]. The basic idea is to augment the substrate graph and simultaneously map the virtual nodes and links using a mixed integer programming formulation. Since mixed integer programs are computationally intractable, the authors used relaxation and rounding to develop polynomial time heuristics. For details, we refer to [6]. We use these existing node embedding algorithms to implement the node embedding phase of our SVNE solutions.

3.13.3 Deterministic Node Embedding Algorithm

D-ViNE [6] takes online VN requests as inputs and maps them onto the substrate network one at a time. It takes decisions based only on the past VN requests that it has already seen, *i.e.* , D-ViNE uses no look-ahead. Since the integer domain constraints on the x variables used in the formulation have already been relaxed, we no longer get integer values for the x variables. Instead, D-ViNE employs deterministic rounding technique to get integer values for x. We introduce $\varphi : N^S \rightarrow \{0, 1\}$, which is initially set to zero for all $n^S \in N^S$ signifying that all the substrate nodes are initially unused. Whenever a virtual node is mapped to a particular physical node n^S, we set $\varphi(n^S)$ to 1 to ensure that no substrate node is used twice for the same VN request.

Algorithm 5 Deterministic VNE

1: **procedure** D-ViNE($G^V = (N^V, E^V)$)
2: Create augmented substrate graph $G^{S'} = \left(N^{S'}, E^{S'}\right)$
3: Solve **VNE_LP_RELAX**
4: **for all** $n^S \in N^S$ **do**
5: $\varphi(n^S) \leftarrow 0$
6: **end for**
7: **for all** $n \in N^V$ **do**
8: **if** $\Omega(n) \cap \{n^S \in N^S | \varphi(n^S) = 1\} = \emptyset$ **then**
9: VN request cannot be satisfied
10: **return**
11: **end if**
12: **for all** $z \in \Omega(n)$ **do**
13: $p_z \leftarrow (\sum_i f^i_{\mu(n)z} + f^i_{z\mu(n)}) x_{\mu(n)z}$
14: **end for**
15: Let $z_{max} = \arg\max_{z \in \Omega(n)} \{p_z | \varphi(z) = 0\}$
16: set $\mathcal{M}_N(n) \leftarrow z_{max}$
17: $\varphi(z_{max}) \leftarrow 1$
18: **end for**
19: Update residual capacities of the network resources.
20: **end procedure**

Description and Discussion

The procedure begins by creating an augmented substrate graph, $G^{S'} = \left(N^{S'}, E^{S'}\right)$ for the VN request $G^V = \left(N^V, E^V\right)$ using the augmentation method described [6]. Next it solves a relaxed linear program to get a fractional solution which is at least as good as the integer solution. For each virtual node, D-ViNE first checks whether there are any unmapped substrate nodes within its feasible region (the substrate nodes in the virtual nodes Ω set). If the corresponding Ω set is empty, D-ViNE stops the embedding process immediately and rejects the VN request. Otherwise the *deterministic rounding procedure* is initiated in line 12.

For each virtual node n, D-ViNE calculates a value p_z for each substrate node $z \in \Omega(n)$ in its cluster. p_z is calculated as the product of the value $x_{\mu(n)z}$ and the total flow passing through the meta-edge $\mu(n)z$ in both directions. The reason behind using this multiplication instead of just $x_{\mu(n)z}$ is as follows. In the MIP solution x_{uv} is set to binary

values based on the presence of flows in either direction in the edge (u, v). When the binary constraint x is relaxed, one might expect that the fractional values of x_{uv} would also be proportional to the total flow in the edge (u, v). But during the LP relaxation process, the correlation between the flow variable f and the binary variable x is lost. It is because a linear program tries to optimize the objective function without violating the constraints; it does not care about the values as long as they are within their permitted domains. As a result, in the relaxed linear program, it is possible that the f values are very high and the corresponding x values are very low or vice versa. Multiplying the f and x values thwarts the possibility of selecting a substrate node based on high x value but very low f value on its corresponding meta-edge and vice versa. The ones that have better values for both the variables f and x are more likely to be in the solution of the MIP than others. D-ViNE maps the virtual node n onto the unmapped substrate node z (*i.e.* , $\varphi(z) = 0$) with the highest p_z value, breaking ties arbitrarily.

3.14 Path Selection Mechanisms

The effectiveness of the proposed heuristics depend on efficient path selection mechanisms. Especially the proposed hybrid policy heuristic can adopt any path selection algorithm that suits its purpose. In this section we delineate various path selection mechanisms that can be utilized in our solutions.

3.14.1 Static Path Selection Heuristics

The k-shortest path algorithm is the simplest heuristic that can be employed in our solutions. It is a static algorithm, in the sense that the path set for each virtual link remains constant throughout the duration of the virtual network. In our experiments, we used a k-shortest path algorithm adapted for efficient path computation in communication networks [45].

3.14.2 Dynamic Path Selection Heuristics

The first dynamic path selection approach is to use a primal dual formulation, where we first find the dual LP associated with the relaxed primal LP by swapping variables and

constraints. The primal dual approach leads to an iterative algorithm which raises another issue of fast convergence. Normally theoretical convergence guarantees of iterative primal dual algorithms do not always work well in practice.

The second approach for handling dynamic path selection is to employ a column generation approach. Here instead of solving the LP with all the flow variables at once, only a subset of path variables is considered at each step. After solving the LP at each step with an active set of paths, the path set is updated by adding improved paths and removing unused paths. This is also an iterative algorithm, however the convergence test is usually simpler and more practical than the primal dual approach. But an additional issue with this approach is updating the active path set at each step using an efficient path selection heuristic.

3.15 Alternative Formulation for SVNE

In this section we present a mixed integer programming formulation for link embedding with fast substrate link restoration. The goal is to find a feasible multi-commodity flow for the working paths and local backup detours for each link on the working paths subject to the flow constraints and bandwidth constraints. The objective is to minimize the total amount of backup resources required, that is the number of times a substrate link is included in a backup detour subject to the constraints. A more realistic objective would be concerned with maximizing shared protection among the backup detours, but we leave it as a future research direction for now. Since we are concerned with the link embedding phase of virtual network embedding, we already know the substrate nodes that have been selected for the endpoints of each virtual link. As a result we can treat each virtual link as a commodity in the link restorable multi-commodity flow formulation. We employ a path indexed formulation, since it allows us to handle QoS parameters specified with the VN request. The components of the MIP are shown below:

LP_LE_LR

-Objective Function

Minimize

$$\sum_{v\in E^V}\sum_{p\in\mathcal{P}(v)}\sum_{f\in E^S}\sum_{d\in\mathcal{P}(f)\setminus\{f\}} b_2(p,v,f,d) + \sum_{v\in E^V}\frac{db(v)}{b(v)} \tag{3.35}$$

subject to

-Link Capacity Constraint

$$\sum_{v\in E^V}\sum_{p\in\mathcal{P}(v)}\delta_s(p)b_1(v,p) + \sum_{v\in E^V}\sum_{p\in\mathcal{P}(v)}\delta_s(p)\sum_{f\in E^S\setminus\{s\}}\sum_{d\in\mathcal{P}(f)\setminus\{f\}} b_2(v,p,f,d) \le \mathcal{R}(s),\ \ \forall s\in E^S \tag{3.36}$$

-Primary Flow Constraint

$$\sum_{p\in\mathcal{P}(v)} b_1(v,p) = b(v),\ \ \forall v\in E^V \tag{3.37}$$

-Detour Flow Constraint

$$\sum_{d\in\mathcal{P}(f)\setminus\{f\}} \delta_f(d)b_2(v,p,f,d) \le b_1(v,p),\ \ \forall v\in E^V, \forall p\in\mathcal{P}(v), \forall f\in p \tag{3.38}$$

-Domain Constraints

$$b_2(p,v,f,d)\ge 0,\ \ \forall v\in E^V, \forall p\in\mathcal{P}(v), \forall f\in p, d\in\mathcal{P}(f)\setminus\{f\} \tag{3.39}$$

$$b_1(v,p)\ge 0,\ \ \forall v\in E^V, \forall p\in\mathcal{P}(v) \tag{3.40}$$

3.15.1 Remarks

- $db(v) = b(v) - \sum_{f\in p}\sum_{d\in\mathcal{P}(f)\setminus\{f\}} b_2(p,v,f,d)$
- The objective (3.35) is to minimize the total amount of backup resources and also minimize the total bandwidth disruption caused due to failures.

- The primary and detour flow constraints (3.37, 3.38) specify that for each virtual link commodity $v \in E^V$, the total flow among all QoS constrained paths (working and backup detours) must be bounded by $b(v)$, the bandwidth requirement of the virtual link.
- Next we have the substrate link Capacity Constraint (3.36) which limits the total amount of working and backup flow for all the virtual links passing through a substrate link s to the residual capacity of s, $\mathcal{R}(s)$.
- Finally the Domain Constraints (3.40, 3.39) specify the range of values for the flow variables.
- Since LP_LE_LR is a linear program, we can solve it in polynomial time.

Chapter 4

Performance Evaluation

In this chapter, we first describe our simulation environment, then present evaluation results. Our evaluation is aimed at quantifying the performance of the proposed HYBRID policy approach to SVNE in terms of long term business profit for the InP by maximizing revenue earned from VN's and minimizing the penalty incurred due to substrate link failures.

4.1 Simulation Environment

We implemented a discrete event simulator for SVNE adapted from our ViNE-Yard simulator [6]. Since network virtualization is still not widely deployed, the characteristics of VN's and failure are not well understood yet. Specifically there are no analytical or experimental results on the substrate and virtual network topology characteristics, VN arrival dynamics or link failure dynamics in network virtualization. As a result, we use synthetic network topologies, and poisson arrival processes for VN's and link failures in our simulations. However our choice of substrate and virtual topologies and VN arrival process parameters are chosen in accordance with previous work on this problem [6, 49]. We used the GNU linear programming toolkit to solve all the linear programs in our formulations.

The substrate network topologies in our experiments are randomly generated with 50 nodes using the GT-ITM tool [18] in 25 x 25 grids. Each pair of substrate nodes is randomly connected with probability 0.5. The cpu and bandwidth resources of the substrate nodes

Notation	**Description**
Simulation Parameter: α	Percentage of bandwidth of a substrate link for primary flow.
Simulation Parameter: $\gamma = \frac{\lambda_F}{\lambda_V}$	Ratio of failure and VN arrival rate.
Simulation Parameter: n_V	Number of VN nodes.
Simulation Parameter: k	Number of paths allowed for link embedding and detours.
Performance Metric: π	Average business profit in the long run.
Performance Metric: ar	Average acceptance ratio in the long run.
Performance Metric: bru	Average backup resource usage percentage in the long run.
Performance Metric: t	Average response time to a failure.

Table 4.1: Simulation Parameters and Performance Metrics

and links are real numbers uniformly distributed between 50 and 100. We assume that both VN requests and substrate link failure events follow a Poisson process with arrival rates λ_V and λ_F. The ratio $\gamma = \frac{\lambda_F}{\lambda_V}$ is a parameter that we vary in our simulations. We use realistic values for $MTTR(l)$ based on failure characteristics of real ISP networks [25] which represent InP networks in a NVE. In each VN request, the number of virtual nodes is a uniform variable between 2 and 20. The average VN connectivity is fixed at 50%. The bandwidth requirement of a virtual link is a uniform variable between 0 and 50, and the penalty value $\mathcal{S}_j(v)$ for a virtual link v is set to a uniform random variable between 2 and 15 monetary units. In our simulations, we set $\alpha(s) = \alpha, \forall s$ belonging to the substrate network and vary α, where $0 \leq \alpha \leq 1$. For each set of experiments conducted, we plotted the average of 5 values for the performance metrics. The simulation parameters and output performance metrics are shown in Table 4.1.

Notation	Algorithm Description
SVNE-Greedy-Hybrid	Greedy Node Embedding with Hybrid Policy
SVNE-DViNE-Hybrid	DViNE Node Embedding with Hybrid Policy
SVNE-Greedy-Proactive	Greedy Node Embedding with Proactive Policy
SVNE-DViNE-Proactive	DViNE Node Embedding with Proactive Policy
SVNE-Greedy-Blind	DViNE Node Embedding with Blind Policy
SVNE-DViNE-Blind	DViNE Node Embedding with Blind Policy

Table 4.2: Compared Algorithms

4.2 Comparison Method

Comparing our heuristics with previous work is difficult since the earlier heuristics do not consider substrate resource failures. As a result we evaluate our proposed *hybrid* policy against two base-line policies. The first one (we call it a *blind* policy) recomputes a new embedding for each VN affected by the substrate link failure. The second one is a *proactive* policy which pre-reserves both primary and backup bandwidth for each virtual link on link disjoint substrate paths. We omit details of these baseline policies due to space limitation. For node embedding, we use greedy [53] and DViNE heuristics [6]. In our evaluation, we have compared six algorithms that combine different node embedding strategies [6,53] with our proposed survivable link embedding strategies, namely, SVNE-Greedy-Hybrid, SVNE-DViNE-Hybrid, SVNE-Greedy-Proactive, SVNE-DViNE-Proactive, SVNE-Greedy-Blind, and SVNE-DViNE-Blind, as shown in the following Table .

4.3 Evaluation Results

We use several performance metrics for evaluation purposes in our experiments. We measure the long term average profit earned by the InP by hosting VN's. The profit function

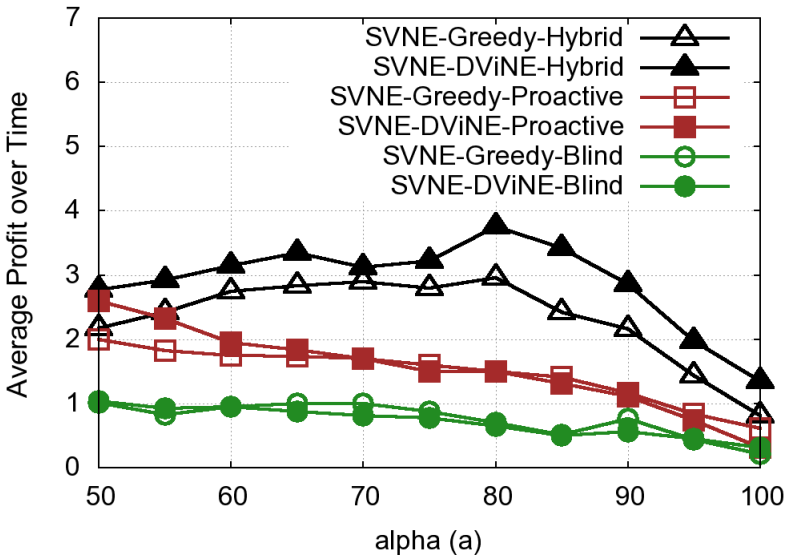

Figure 4.1: Business profit against α

depends on both the revenue earned from VN's by leasing resources and penalties incurred due to service disruption caused by substrate link failures. The penalty depends on both the amount of bandwidth violated due to a failure and the time it takes to recover from a failure as expressed in equations 3.8 and 3.9. We also measure the long term average acceptance ratio, percentage of backup bandwidth usage and response time to failures. We present our evaluation results by summarizing the key observations in the following subsections.

4.3.1 Acceptance ratio and Business profit:

The hybrid policy leads to higher acceptance ratio and increased business profit in the presence of failures. Figures 4.1 shows the long term business profit against the percentage α of substrate link bandwidth for primary flows, while Figure 4.2 does it against the ratio of failure and VN rate γ. We notice that over the range of values for α and γ, the hybrid policy outperforms both the blind and proactive policies. Also the hybrid policy generates the highest profit at $\alpha = 80\%$, whereas the proactive and blind policies generate lesser profit with increased values of α. As α increases, there is less bandwidth available for

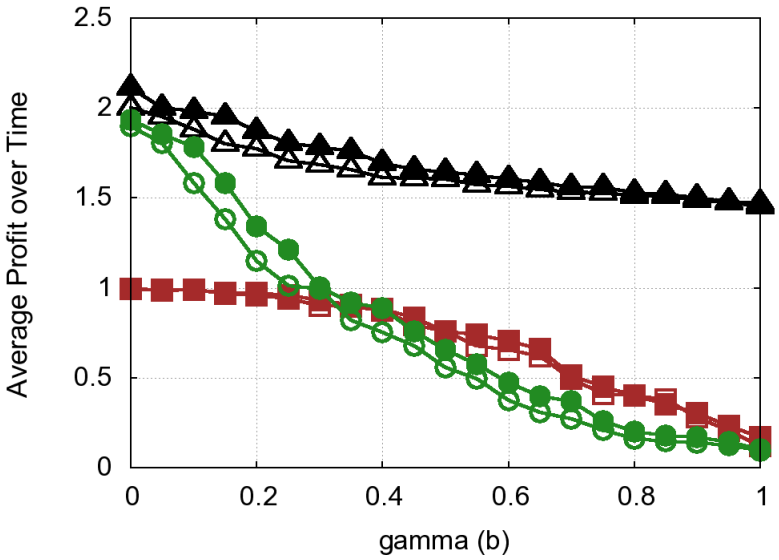

Figure 4.2: Business profit against γ

backups on the substrate link and this hinders the performance of these policies. This also affects the hybrid policy, but it still has better performance due to its reactive nature. The profit and acceptance ratio for the blind policy drops more rapidly than the hybrid policy against increase in γ as shown in Figures 4.2 and 4.4. Although, the profit for the proactive policy increases with γ, it is still outperformed by the hybrid policy for the range of the simulation parameters.

4.3.2 Responsiveness to Failures:

The hybrid policy has faster reaction time to failures than its counterparts. In Figure 4.6, we notice that the hybrid policy reacts faster than the blind policy when a failure occurs. When a substrate link fails, the blind policy recomputes the entire embedding, which is time consuming. The hybrid policy, on the other hand, only re-routes the bandwidth of the affected virtual links which results in faster response time and ultimately lower penalty values for the InP.

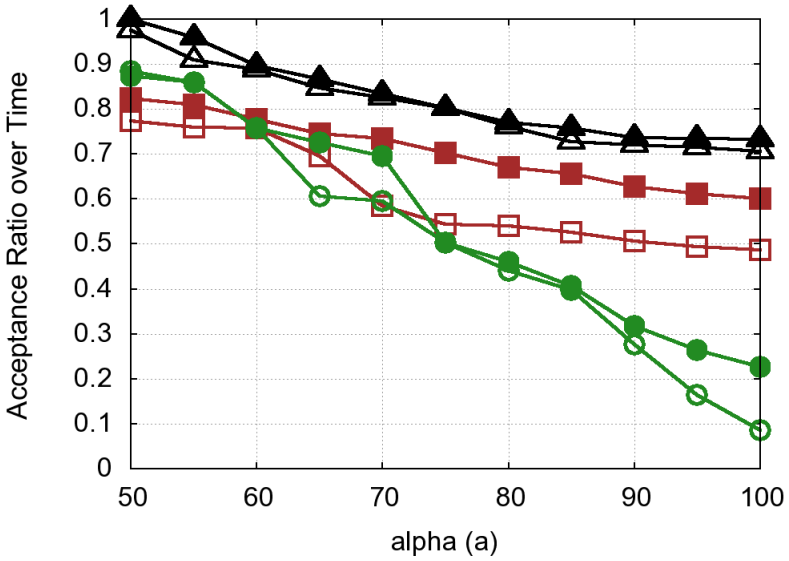

Figure 4.3: Acceptance ratio against α

4.3.3 Bandwidth Efficiency:

The hybrid policy is bandwidth efficient. The proactive policy pre-reserves additional bandwidth for each virtual link during the instantiation phase. On the other hand, the hybrid policy does not pre-reserve any backup bandwidth during the link embedding phase. It pre-selects the candidate paths for re-routing and allocates backup bandwidth only when an actual failure occurs. As a result, the average bandwidth usage increases less rapidly with γ compared to the blind policy. This is shown in Figure 4.5.

4.3.4 Performance on Specific VN Topologies

Up until now we have focused on arbitrary VN request topologies in our evaluations. However, some classes of topologies are naturally expected to be more prevalent than others due to their use in popular applications. For example, hub-and-spoke topologies are commonly used to connect distributed sites to a centralized server e.g.in content distribution networks. In this section, we compare the performance of the proposed policies on two different classes of topologies: hub-and-spoke and mesh.

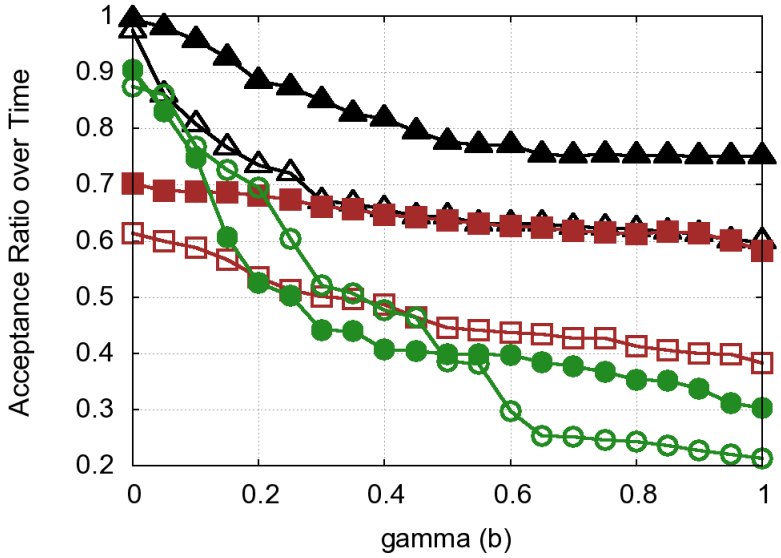

Figure 4.4: Acceptance ratio against γ

Hub-and-Spoke Topologies

We have used similar simulation settings for this set of experiments while ensuring hub-and-spoke topologies in the VN requests instead of random graphs. Table 4.3.4 summarizes the results of the compared algorithms for the five performance metrics. The results presented here are for an arrival rate of 4 VNs per 100 time units, and we present all values after standard deviations of their successive samples become negligible. The algorithms used for this experiment are the exact same ones without any topology-specific modifications.

As seen in Table 4.3.4, relative performance of the compared algorithms are unchanged for hub-and-spoke topologies. Careful readers will notice that related observations for random graph requests also hold true in this case.

Mesh Topologies

Mesh topologies can be considered to be at the opposite end of the spectrum of specific topologies. In this case, we again use similar experimental settings and make sure that the VN requests form full mesh topologies. Simulation results in steady states are summarized

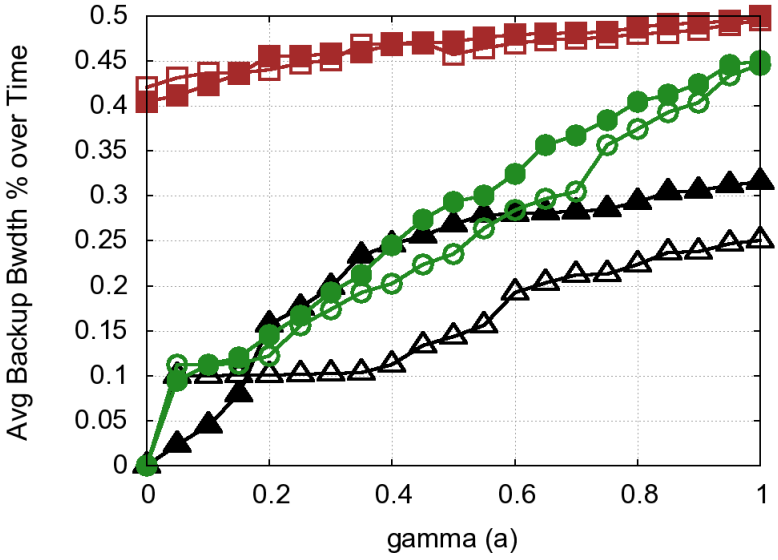

Figure 4.5: Backup resource usage against γ

in Table 4.3.4 for similar experimental conditions. The algorithms used for this experiment are also without any topology-specific modifications.

The most noticeable change in this case is overall performance reductions across the board, a large portion of which can be attributed to the natural dense formation of mesh topologies that call for more resources than substrate can provide. However, relative performance of the compared algorithms are mostly unchanged.

4.3.5 Effect of k: Number of Paths Allowed

We also evaluate our performance metrics against k (Figures 4.7, 4.8, 4.9), which specifies the size of the path-sets for primary and detour flows in our path-flow based formulations. The results indicate the superior performance of the hybrid policy against the baseline policies. However there is some variability among the performance metrics for different values of k, which could suggest that a SVNE solution that continuously updates k in order to improve performance, might be better than a static solution that always uses the same value for k. This might also point towards iterative approaches using primal dual or column generation approaches.

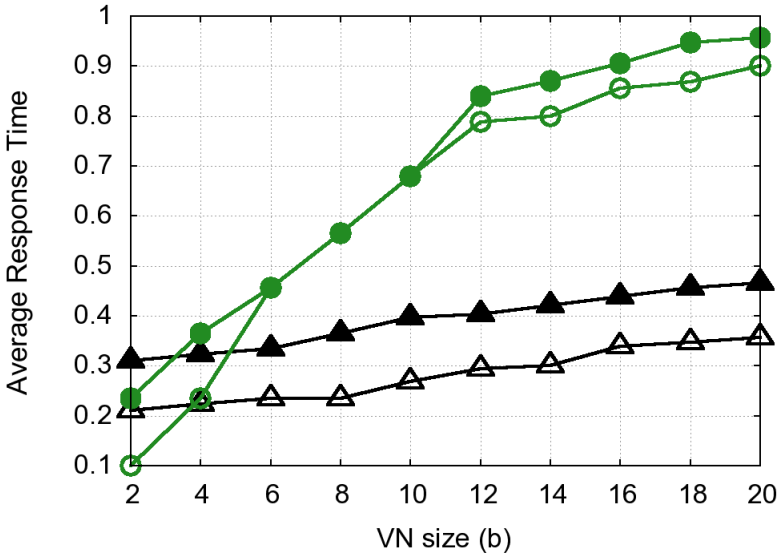

Figure 4.6: Response time against VN size

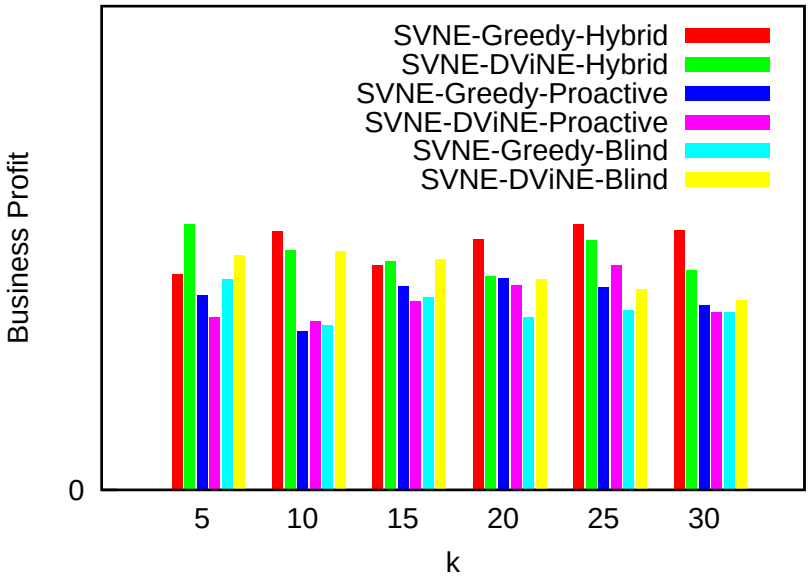

Figure 4.7: Business Profit against k

In this set of experiments, we vary the value of k between 5, 10, 15, 20, 25, and 30. We

	Business Profit	**Acceptance Ratio**	**Backup Usage**
SVNE-Greedy-Hybrid	0.756	0.642	0.459
SVNE-DViNE-Hybrid	0.813	0.717	0.395
SVNE-Greedy-Proactive	0.662	0.576	0.835
SVNE-DViNE-Proactive	0.525	0.496	0.887
SVNE-Greedy-Blind	0.372	0.412	0.695
SVNE-DViNE-Blind	0.441	0.324	0.760

Table 4.3: Comparative Performance on Hub-and-Spoke Topologies

	Business Profit	**Acceptance Ratio**	**Backup Usage**
SVNE-Greedy-Hybrid	0.743	0.675	0.489
SVNE-DViNE-Hybrid	0.876	0.777	0.391
SVNE-Greedy-Proactive	0.608	0.580	0.809
SVNE-DViNE-Proactive	0.598	0.501	0.882
SVNE-Greedy-Blind	0.333	0.417	0.699
SVNE-DViNE-Blind	0.448	0.321	0.708

Table 4.4: Comparative Performance on Mesh Topologies

observe a similar trend in performance against k, since the hybrid policy exhibits better performance compared to the baseline policies. For business profit, we observe that the hybrid policy with DViNE has highest profit for value $k = 5$, whereas, for the other values of k, the hybrid policy with greedy node embedding has maximum profit. For the previous set of experiments, we did not observe any significant variation in performance due to the selected node embedding heuristic. This implies that the performance metrics against k are affected by the selected node embedding mechanism. However the acceptance ratio against k experiments do not exhibit any variation due to the selected node embedding heuristic.

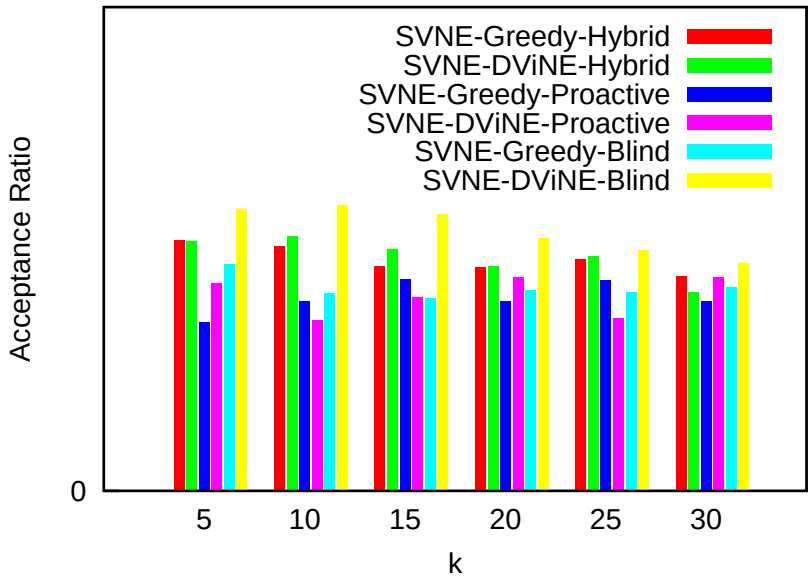

Figure 4.8: Acceptance Ratio against k

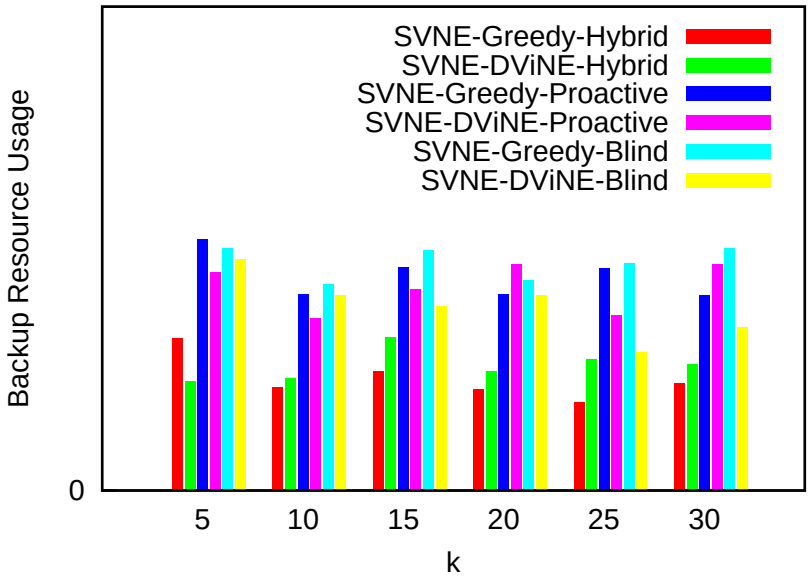

Figure 4.9: Backup Resource Usage against k

Chapter 5

Conclusions and Future Works

5.1 Summary of Contributions

Network virtualization stands at a unique point in the virtualization design space by supporting dual purposes. On one hand, it provides a powerful means to run multiple custom networks on top of a single shared substrate thereby enabling the de-ossification of the current Internet architecture and leading towards a long term solution for the future Internet. On the other hand, it also plays a vital role in running multiple network experiments in a shared experimental facility (e. g. GENI, VINI), and allowing networking researchers access to real testbeds instead of simulators to test novel algorithms and protocols.

Resource allocation and survivability in network virtualization offer a large number of research challenges that span multiple research areas in theory, and systems. In this thesis, we have addressed the important aspect of adding survivability to network virtualization in-order to ensure seamless operation of the virtual networks embedded on top of an InP in the presence of failures. In this regard, we have formulated the SVNE problem to incorporate substrate failures in the virtual network embedding problem. We have also proposed baseline policy solutions and an efficient hybrid policy heuristic to solve SVNE. To the best of our knowledge this is the first attempt to add survivability to virtual network embedding algorithms along with support for business profit oriented optimization. Moreover, our proposed heuristic can be extended to deal with multiple link failures, and subsequently combined with a node migration strategy [48] to solve the single substrate node

failure problem. We have shown detailed formulations of our proposed SVNE policies and derived efficient heuristics using optimization techniques. We also performed evaluations to demonstrate the validity and importance of our contributions.

5.2 Future Research Directions

There are many possible research directions that can be directly pursued from our current work. We know briefly mention some of the important extensions and directions:

5.2.1 Survivability in Multi-domain NVE

Survivability in a multi-domain NVE could raise further challenges since it involves both intra and inter domain link failures. Multiple simultaneous inter-domain and intra-domain failures could lead to complex scenarios requiring more sophisticated recovery and protection mechanisms than just intra-domain survivability mechanisms. However inter-domain survivability mechanisms are predicated on complete end-to-end multi-domain virtual network embedding solutions, which is still an unresolved research problem.

5.2.2 Resource Allocation and Survivability in Recursive NVE

It would also be interesting to extend survivability to recursive NVE, where the first level VNs can act as InPs to a second level of VNs. Resource allocation, protection, and restoration issues in such recursive environments could be investigated under cross layer optimization or network utility maximization (NUM) frameworks. However a global optimization formulation for resource allocation and survivability in recursive NVE would assume the existence of a single party that controls all the layers. Since this scenario is not practical, the solution to such a global optimization problem would further require distributed solutions using dual decomposition techniques.

5.2.3 Adaptive SVNE

Our proposed solutions to SVNE are static in the sense that at any given solution instance we have fixed values for the $\alpha : \beta$ proportions and k. However the revenue of the InP might

depend on complex combinations of these parameters which points towards dynamic solutions to SVNE using control theory or statistical machine learning techniques. Feedback control mechanisms can be utilized to continuously monitor the performance metrics and use the feedback to adapt the simulation parameters in-order to further improve the performance of the mechanisms.

5.2.4 Service Differentiation Aware SVNE

In our SVNE formulation, we specify a fixed partition among primary and backup substrate resources. We can introduce service differentiation into the model by further subdividing the backup resource percentage into multiple service classes, where a higher service class will have a larger portion of the backup percentage, but will also have to pay more to the InP for better survivability guarantee. Service differentiation mechanisms like Paris metro pricing schemes have been shown to increase revenue for ISP's. Similar mechanisms can be utilized in service differentiation aware SVNE to further improve performance.

APPENDICES

Appendix A

Incentive Compatible Virtual Network Embedding

In this appendix, we attempt to initiate a systematic study of network virtualization from a game theoretic and mechanism design perspective. In an Internet scale network virtualization environment, it is not economically feasible to pool all network resources with one infrastructure provider that will be solely responsible for resource allocation. As a result the physical resources will be distributed across a number of infrastructure providers. Because of conflicting goals and economic interests, the InP's may act strategically to increase their own utility which may conflict with the global goal of social welfare maximization. If a virtual network request requires using network resources that are distributed across multiple InP's, then a participating InP might lease resources that locally optimize its own resource usage. It can do that by lying about its local topology or choosing paths that are less costly locally but can result in higher global cost. In other words, since the internal topology and resources owned by an InP are local and private information, it can choose to misrepresent its resources and local topology in-order to maximize its utility. So assuming that the strategic InPs will act in their own interest, the resources (nodes and links) selected for mapping the virtual network request might result in a sub-optimal resource allocation, which might not coincide with the virtual network embedding that would result if the InPs were truthful about their internal resources and local topology. This strategic setting suggests the use of mechanism design [26, 29, 31] to force the InPs to align their local interests with the global goal of social welfare maximization, where a social welfare

maximizing outcome will be a virtual network embedding that minimizes the global cost of the selected resources. Here the system wide goal is to solve a global resource allocation problem, where each InP (agent) controls a certain part of the resource pool.

We now mathematically define the virtual network embedding problem [6,49,53] modified from a mechanism design perspective. The components and assumptions of the network model are described below:

- The topology of the substrate network is modeled as a weighted graph $G_S(V_S, E_S, c(.))$, where V_S is the set of substrate nodes, E_S is the set of substrate links and $c : E_S \rightarrow \mathcal{R}$ is the edge cost function. Let $n_S = |V_S|$ and $m_S = |E_S|$. We assume that the substrate nodes have zero cost.

- The topology of the virtual network is modeled as another graph $G_V(V_V, E_V)$, where V_V, E_V, n_V, m_V are defined similarly.

- The goal of the virtual network embedding problem is to compute an embedding that maps each virtual node to a substrate node and maps each virtual link to a substrate path. Mathematically, we have to compute an embedding $\Gamma(\Gamma_V, \Gamma_E)$, where $\Gamma_V : V_V \rightarrow V_S$ and $\Gamma_E : E_V \rightarrow 2^{E_S}$.

- We assume that distinct virtual links will be mapped to edge-disjoint substrate paths, that is, for $e_V^1, e_V^2 \in E_V$ and $e_V^1 \neq e_V^2$, we have $\Gamma_E(e_V^1) \neq \Gamma_E(e_V^2)$. We call this the *edge-disjoint property.*

- We define the cost of an embedding as the sum of the edges selected for the embedding as

$$C(\Gamma) = \sum_{e_V \in E_V} \sum_{e_S \in \Gamma_E(e_V)} c(e_S) \tag{A.1}$$

  Our main objective is to compute the embedding that minimizes the cost function defined above.

A.1 Virtual Network Embedding and Mechanism Design

A.1.1 Economic Model

The main objective of this work is to study the virtual network embedding problem from a mechanism design perspective. To this end, we require a suitable economic model that captures the strategic interactions among self interested parties in a network virtualization environment. We choose to model this strategic interaction such that each substrate link on the substrate network topology is owned by a self interested strategic agent. There are a number of reasons for adopting this simple economic model. First of all, we believe that this model could be a reasonable starting point for more complicated economic models. And secondly, a similar model was adopted by Nisan and Ronen [28] in their seminal paper *Algorithmic Mechanism Design* to study the strategic aspects of Network Routing. Although in a realistic setting, various subnetworks of the Internet topology are owned by separate Internet Service Providers (ISP) or Infrastructure Providers (InP), we make these assumptions to keep the analysis of the algorithms and the mechanism tractable, while maintaining practical viability. Our results in this model can provide insights and guidelines for strategy proof pricing schemes for virtual network embedding in a more complex setting where each InP owns a subnetwork instead of just a physical link of the global topology. Our goal is to obtain a minimum cost embedding in the presence of self interested agents owning the substrate links. Mechanism design [27, 29, 31] is concerned with the aggregation of privately known preferences of self-interested rational agents into a social choice. The main goal of mechanism design is to discover protocols for dealing with strategic agents that achieve the stated goal of the designer. The protocol has to be designed in such a way that the actions of the selfish agents are properly aligned with the system wide goal of the central designer. In this appendix work, we assume the existence of a central authority responsible for implementing the mechanism for the embedding problem, the case where the computational task of the mechanism is distributed among the rational agents, called Distributed Algorithmic Mechanism Design (DAMD) [10, 11], which is a more realistic model for an Internet scale system, will be discussed later. The most famous result in the mechanism design literature is the Vickrey-Clarke-Groves (VCG) mechanism [7, 13, 46]. In a VCG mechanism, each agent submits its preferences directly

in a single stage to a central mechanism design authority who then computes a payment function for each agent. The power of the VCG mechanism arises from the fact that it is a truthful mechanism, which means that agents do not have any incentive to lie about their preferences. So the VCG mechanism successfully aligns all the agent's private preferences with the system designers global design objective.

We now formally apply the VCG mechanism in the context of the virtual network embedding problem. Since we are applying the VCG mechanism, we can assume that the agents will be truthful. In that case the bulk of the work shifts to designing appropriate payment functions in an efficient manner. Our goal here is to compute the embedding that will minimize the cost function defined in A.1. Since each substrate link (substrate edge) e is controlled by a strategic agent, we compute the VCG payment for each edge e as follows:

$$p^e = \begin{cases} C(\Gamma(\Gamma_V, \Gamma_E; G_S - e)) - C(\Gamma(\Gamma_V, \Gamma_E; G_S|_{e=0})) & \text{if } e \text{ belongs to the embedding} \\ 0 & \text{otherwise} \end{cases}$$

Here $C(\Gamma(\Gamma_V, \Gamma_E; G_S - e))$ corresponds to the cost of the embedding without taking the edge e into account, whereas the term $C(\Gamma(\Gamma_V, \Gamma_E; G_S|_{e=0}))$ is the cost of the embedding after subtracting the cost of the edge e. So we can see from the payment function, that if an edge e is not a part of the embedding, then it receives a zero payment. Otherwise its payment is the difference between the cost of the embedding without e and the cost of the embedding assuming e is free.

A.1.2 Components of the Proposed Mechanism

The main goal is to use VCG type mechanisms for the Virtual Network Embedding problem. Here we assume the existence of a central authority responsible for computing the mechanism. The components of the proposed mechanism are described as follows:

- **Players**: Each substrate link of the substrate graph is owned (controlled) by a strategic agent. So there are $n = |E_S|$ players in the mechanism.
- **Outcomes**: All possible embeddings of the virtual network on top of the substrate network form the set of outcomes $\mathcal{O}$. It should be noted that the number of possible

outcomes will be exponential.

- **Bids**: Each agent bids the cost of its substrate link c_i. He may lie about the true cost in order to avoid being part of the selected embedding, that is c_i is not necessarily equal to the actual cost of the link $c(e_S)$, where $e_S \in E_S$.

- **Objective Function**: The objective function is to compute the minimum cost embedding from the set $\mathcal{O}$ of all possible embeddings. The output function is defined as $f : \mathcal{R}^n \rightarrow \mathcal{O}$.

- **Payment Functions**: We use VCG type payment functions, $p_i : \mathcal{R}^n \rightarrow \mathcal{R}$. The payment for an agent owning link e is as follows:

$$p^e = \begin{cases} C(\Gamma(\Gamma_V, \Gamma_E; G_S - e)) - C(\Gamma(\Gamma_V, \Gamma_E; G_S|_{e=0})) & \text{if } e \in \Gamma \\ 0 & \text{otherwise} \end{cases}$$

A.1.3 Efficient Computation of the Payment Functions

We now shift our focus towards the algorithmic aspects of the problem. Without VCG payments the algorithmic problem only consists of computing the cost minimizing embedding only once. However in the presence of VCG payments, we need to compute the embedding $n+1$ times (we assume n denotes the number of edges in the selected embedding): once for the original network G_S and n times for the substrate network with each edge e removed, that is for each $G - e$. We will start with simple brute force algorithms and then gradually tweak the algorithm to reduce its time complexity. The idea of our first algorithm is very simple, however we believe it is a good starting point for developing strategy proof embedding algorithms for network virtualization. The virtual network embedding problem actually asks for a minimum cost subgraph of the substrate network onto which the virtual network will be mapped. An optimal way to do it will be to check all possible ways to map the virtual nodes on the physical nodes, and then for each possible node mapping, find the shortest path between the physical nodes corresponding to the end point virtual nodes of each virtual link. This computation must be performed $n + 1$ times. The pseudo-code of the algorithm is given below.

Algorithm 6 VCG-ViNE: VCG Computation for Virtual Network Embedding

procedure VCG-ViNE(G_S, G_V)
  Let C_{min} be the cost of the lowest cost embedding
  $C_{min} \leftarrow \infty$
  Let p_{vcg} denote the VCG payment vector corresponding to the lowest cost embedding
  for all $\binom{n_S}{n_V}$ possible node embeddings **do**
    Let Γ_V be the currently selected node embedding
    Let p_c be the current VCG payment vector
    $C \leftarrow 0$
    for all $e_V = (n_V^1, n_V^2) \in E_V$ **do**
      Let $n_S^1 = \Gamma_V(n_V^1)$ and $n_S^2 = \Gamma_V(n_V^2)$
      Find the shortest path P between n_S^1 and n_S^2 on G_S
      Let $d(P)$ be the cost of P
      $C \leftarrow C + d(P)$
      for all $e \in P$ **do**
        Compute shortest path $d(P - e)$ when e is removed from G_S
        $p_c[e] \leftarrow (d(P - e)) - (d(P)|_{e=0})$
      end for
    end for
    if $C < C_{min}$ **then**
      $C_{min} \leftarrow C$
      $p_{vcg}[e] \leftarrow p_c[e], \forall e \in E_S$
    end if
  end for
  Return C_{min} and p_{vcg}
end procedure

The correctness of the algorithm depends on the following lemma. The lemma holds because of the edge-disjoint property that we imposed on our network model.

Lemma 1 *Let $G_S(V_S, E_S, c(.))$ be the substrate network and $G_V(V_V, E_V)$ be the virtual network. Let the lowest cost embedding of G_V onto G_S be $\Gamma(\Gamma_V, \Gamma_E)$. Consider a substrate link $e \in E_S$ that has been selected by the embedding Γ. Let p_e^Γ denote the VCG price for e with respect to the embedding Γ. There exists a path $\mathcal{P} \in \Gamma_E(E_V)$ such that[1] $e \in \mathcal{P}$. Then $p_e^\Gamma = p_e^\mathcal{P}$, where $p_e^\mathcal{P}$ denotes the VCG price for e with respect to the path $\mathcal{P}$.*

[1] $\Gamma_E(E_V)$ is the set of all paths in G_S that correspond to the virtual links in G_V.

Description of the Algorithm

We now describe some of the details of the algorithm. The for loop in line 5 runs over all possible node embeddings. The second for loop in line 9 goes through each virtual link with respect to a fixed node embedding. For each virtual link, we compute the shortest path between the substrate nodes corresponding to the end point virtual nodes of that virtual link. Once a shortest path has been computed, we compute the payment for each substrate link on that shortest path by removing that substrate link from the topology and recomputing the shortest path. We keep track of the node embedding that corresponds to the lowest cost embedding and finally return the cost of the minimum embedding and the VCG payment vector.

We now discuss the time complexity of the proposed algorithm. There are $\binom{n_S}{n_V}$ ways to map the virtual nodes to physical nodes. For each possible node mapping, we have to compute $O(n_S)$ shortest paths[2] for each of the $O(n_V^2)$ virtual links. One shortest path computation can be done in $O(n_S \log n_S + n_S^2)$ time using standard Dijkstra's algorithm. So the total running time of the proposed algorithm is $O(\binom{n_S}{n_V}.n_S.n_V^2.(n_S \log n_S + n_S^2))$. The authors in [16] showed that VCG prices for each edge on a shortest path can be computed in time bounded by the time complexity of just one shortest path computation. Using this result, the brute force approach can be improved to $O(\binom{n_S}{n_V}.n_V^2.(n_S \log n_S + n_S^2))$ time.

A.2 More Realistic Models

In this section, we discuss extensions to more realistic network scenarios. Specifically we mention some ways to extend the basic network and economic model used in this work.

A.2.1 More Realistic Network Models

We mentioned earlier that our adopted network model had some limitations. For example, we have been concerned with the embedding of a single virtual network. A natural extension is to consider more than one virtual network request arriving over time. In that case, we can apply the VCG mechanism and the algorithm VCG-ViNE separately for each

[2]The length of any path in a graph containing n nodes is atmost $n-1$

arriving virtual network request. Since it is possible to share the bandwidth of one substrate link among multiple virtual networks, we would have to keep track of the remaining available bandwidth for each substrate link. However, since the remaining available bandwidth for each substrate link is also private information only known to the agent owning that link, this might lead to another level of strategic behavior on behalf of the agents. Moreover, it is also possible to hypothesize strategic interaction at a different granularity in the presence of incoming online virtual network requests. We can assume that each separate virtual network request is performed by strategic agents (the service providers) who want to maximize their utility from using the substrate network resources as much as possible. Normally a virtual network request will also contain a *time duration* parameter indicating how long the virtual network wants to utilize the substrate resources. An agent in this case might lie about the duration in-order to maximize utility. As an example, consider a service provider that requires a virtual network to serve end users every alternate day for one month. So it requires substrate network resources for 15 days, but in-order to avoid the virtual network creation costs and release costs, the service provider agent might register the required time duration parameter as one month, which is not its true time duration preferences. Also a service provider might want to keep the virtual network for more time than required in anticipation of future resource needs. This will block other service providers from using the substrate network resources and will decrease network resource utilization in the long run. So we can see that service provider agents can not only lie about their preferences for substrate network resources, but they can also misrepresent their time duration by reporting false start times and end times. This scenario is very close in nature to the *online mechanism design* setting [12, 32, 34, 35], where a dynamically changing set of agents interact with the mechanism over an extended period of time. An online mechanism is an extension of the classical mechanism design framework, which introduces the notion of time dependency and the agents can arrive and depart at discrete points of time. As a result the mechanism must make decisions at each time step. One of our future research directions would be to investigate the online mechanism design aspect of dynamic network virtualization environments.

A.2.2 More Realistic Economic Models

In our basic economic model, we assumed that each substrate link was owned and controlled by a strategic agent. This is not very realistic and does not correspond to the way the real Internet topology is organized. If we consider the Internet to be a massive scale graph, then different disjoint subgraphs would be owned by separate *autonomous systems* (AS). The AS's themselves are connected by inter-domain links. An example of this is shown in figure A.1.

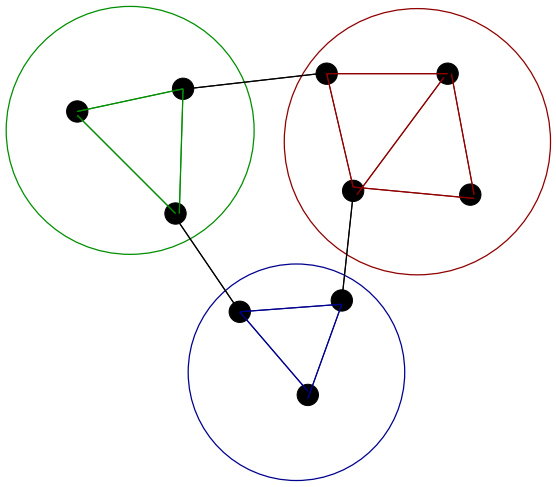

Figure A.1: More Realistic Economic Models.

An easy way to extend our results would be allow each AS to bid separately for each of its substrate links. However this works only if we ignore substitutability and complementarity effects[3]. If such effects cannot be ignored, then we can resort to the known results in case of *single minded bidders* [21]. A single minded bidder only cares about one particular set of items. In our case, the agent will only be concerned with the set of substrate links that fall within its designated AS. We can then apply the greedy payment scheme from [21] which preserves truthfulness. Notice that this payment function is a non VCG payment scheme and only works for a restricted subclass of single minded bidders. It should also be noted that the inter-domain links are not controlled by any agent. It might be possible

[3]Complementarity: valuation for a set of substrate links can be more than sum of valuations for the individual substrate links, Substitutability: valuation for a set of substrate links can be less than sum of valuations for the individual substrate links.

that these links are controlled by the central mechanism designer or jointly by the ASs that are interconnected by a private inter-domain link.

A.2.3 Distributed Computation of VCG Payments

In the traditional mechanism design (MD) and algorithmic mechanism design (AMD) setting [29], it is assumed that a trusted center exists that implements the required economic mechanisms. However in an Internet scale network virtualization environment, it is no longer feasible to transmit all relevant private information to a central trusted authority. There are a number of reasons for this. First of all, the InPs will most likely form a Peer-to-Peer (P2P) network with peering relationships. As a result no one InP might agree to take on the role of the trusted center. Secondly, the communication overhead of transmitting information to a center (if one exists) might be prohibitive at such a large scale. This calls for a decentralized approach that not only distributes incentives but also the computation of mechanism among the InPs and leads us to the realm of Distributed algorithmic mechanism design (DAMD) [10, 11]. DAMD differs from AMD in a number of ways, for example the complexity measures for DAMD include *network complexity* which measures the number of messages passed over the network, message size, local computation and local storage required for the messages used to implement the mechanism. Another important difference is that the task of computing the mechanism is now distributed across the strategic agents. As a result in DAMD agents can not only manipulate their private valuations, but also the results of the local computations required for mechanism. These issues suggest that DAMD is the most suitable economic model for studying problems in the domain of networking. The DAMD approach has been recently used to study networking problems like web-caching, P2P file sharing, and overlay network construction.

We now discuss some approaches towards distributed computation of VCG payments [33] for our proposed mechanism. A distributed algorithm will delegate the task of computing the outcome and payment functions of the mechanism among the agents themselves, in the absence of any central authority. It can be easily seen that distribution leads to further scope of manipulation since the agents can change the computational results that they have control over. As an example, consider a set of agents organized in a *ring topology* and the agents are performing a distributed algorithm for the second price auction by passing around the two top most bids (tokens) along the ring. As a result an agent can

manipulate the second highest bid when the token comes to that agent and if no other agent further changes the token, then that manipulative agent can get the item at a lower price. One approach to prevent computational manipulations is to use *replication* [33]. In this technique, the set of agents are divided into two groups and each group computes its own version of the outcome and payment functions. If the results from the two groups match, then the outcome and payments are enforced. If they don't match, then at least one agent in one group cheated and a severe penalty is enforced on all agents[4]. It should be noted that in these distributed settings, truthfulness can no longer be achieved in dominant strategies, rather we have to resort to the slightly weaker solution concept of ex-post Nash equilibrium [40]. Another issue that should be addressed is that the payment function for an agent i should be computed by any subset of agents excluding i to avoid manipulation. An approach that can be adapted to our economic model is to utilize a *distributed hash table* (DHT) type system, where the payment for each substrate link agent will be performed by a subset of neighboring[5] substrate link agents. This type of scheme has been applied in the context of incentive mechanisms for promoting cooperation in P2P networks [15, 47].

A.3 Related Work

Traditionally computer networks has been a very rich application area for game theory and mechanism design. However to the best of our knowledge our work is the first to formally apply mechanism design in the context of the virtual network embedding problem in network virtualization. There has been a number of research projects that studied the stability and co-existence of multiple overlay networks on top of a native IP network which is quite similar to the network virtualization setting [23, 37]. However these results are concerned with post mapping phases, that is after the overlay networks have been constructed, whereas our results are more concerned with the strategic aspects of the virtual network formation phase. Also these works are more related to the classic results on *selfish routing* by Roughgarden and Tardos [39]. Recently Yuen and Li applied mechanism design to study the dynamic multicast tree formation problem in overlay networks [50]. The same authors also investigated applications of mechanism design for dynamic topology

[4]In this case we need a central authority to enforce penalties, so the approach is not completely distributed.

[5]Two substrate links are neighbors if they share a substrate node.

formation in autonomous networks [51].

A.4 Conclusion

The presence of multiple types of stake-holders at different levels of granularity (end-users, service providers and infrastructure providers) in a network virtualization environment naturally leads to large scale strategic interactions among the different parties. We believe that mechanism design is an appropriate tool for smoothing out the conflicting interests and preferences of this diverse set of agents. In this appendix work we have initiated a systematic study of network virtualization from a mechanism design perspective. We have specifically focused on the virtual network embedding problem in the presence of multiple infrastructure providers controlling disjoint parts of the shared substrate infrastructure. Our main contribution in this appendix work is the application of the classic VCG mechanism to solve the virtual network embedding problem in the presence of strategic agents. We have developed a simple algorithm for computing the VCG payments and discussed ways to extend the results to more realistic scenarios. In the future we will perform experiments using simulation tools and PlanetLab test-beds [36] to test the scalability and performance of our initial results. We will also apply tools from game theory and mechanism design to address some other important problems in network virtualization, e. g. efficient market mechanisms for dynamic network virtualization environments where service providers and infrastructure providers can buy and sell substrate network resources in the presence of brokers who monitor the market and determine market clearing prices. We can also examine recursive network virtualization environments where the service providers themselves can resell the network resources they purchased from the infrastructure providers and add infinitum. These hierarchical and recursive environments and interactions would ultimately lead to a very complex and multi-level network virtualization marketplace that can be analyzed and controlled using tools from both micro-economics and macro-economics.

References

[1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, February 1993.

[2] Issam Aib and Raouf Boutaba. On leveraging policy-based management for maximizing business profit. *IEEE Transactions on Network and Service Management (TNSM)*, 4(2):14, 2007.

[3] D.G. Andersen. Theoretical approaches to node assignment. Unpublished Manuscript, `http://www.cs.cmu.edu/~dga/papers/andersen-assign.ps`, 2002.

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

[5] N.M.M.K. Chowdhury and R. Boutaba. Network virtualization: state of the art and research challenges [topics in network and service management]. *Communications Magazine, IEEE*, 47(7):20–26, July 2009.

[6] N.M.M.K. Chowdhury, M.R. Rahman, and R. Boutaba. Virtual network embedding with coordinated node and link mapping. In *IEEE INFOCOM*, pages 783–791, April 2009.

[7] Edward H. Clarke. Multipart pricing of public goods. *Public Choice*, 11(1), September 1971.

[8] J. Fan and M. Ammar. Dynamic topology configuration in service overlay networks - a study of reconfiguration policies. In *IEEE INFOCOM*, 2006.

[9] Nick Feamster, Lixin Gao, and Jennifer Rexford. How to lease the internet in your spare time. *SIGCOMM Comput. Commun. Rev.*, 37(1):61–64, 2007.

[10] Joan Feigenbaum, Christos Papadimitriou, Rahul Sami, and Scott Shenker. A bgp-based mechanism for lowest-cost routing. *Distrib. Comput.*, 18(1):61–72, 2005.

[11] Joan Feigenbaum and Scott Shenker. Distributed algorithmic mechanism design: recent results and future directions. In *DIALM '02: Proceedings of the 6th international workshop on Discrete algorithms and methods for mobile computing and communications*, pages 1–13, New York, NY, USA, 2002. ACM.

[12] Eric Friedman and David C. Parkes. Pricing WiFi at Starbucks– Issues in online mechanism design. In *Fourth ACM Conf. on Electronic Commerce (EC'03)*, pages 240–241, 2003.

[13] T. Groves. Incentives in teams. *Econometrica*, 41:617–631, 1973.

[14] Anupam Gupta, Jon M. Kleinberg, Amit Kumar, Rajeev Rastogi, and Bulent Yener. Provisioning a virtual private network: A network design problem for multicommodity flow. In *Proceedings of ACM STOC*, pages 389–398, 2001.

[15] David Hausheer. *PeerMart: Secure Decentralized Pricing and Accounting for Peer-to-Peer Systems*. PhD thesis, ETH Zurich, Aachen, Germany, March 2006.

[16] J. Hershberger and S. Suri. Vickrey prices and shortest paths: What is an edge worth? In *FOCS '01: Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, page 252, Washington, DC, USA, 2001. IEEE Computer Society.

[17] Ines Houidi, Wajdi Louati, and Djamal Zeghlache. A distributed virtual network mapping algorithm. In *Proceedings of IEEE ICC*, pages 5634–5640, 2008.

[18] Maciej Kurant and Patrick Thiran. Survivable MApping Algorithm by Ring Trimming (SMART) for large IP-over-WDM networks. In *BroadNets*, 2004.

[19] Maciej Kurant and Patrick Thiran. Survivable Routing of Mesh Topologies in IP-over-WDM Networks by Recursive Graph Contraction. *IEEE Journal on Selected Areas in Communications*, 25(5):922 – 933, 2007.

[20] K. Lee and E. Modiano. Cross-layer survivability in wdm-based networks. In *INFOCOM 2009, IEEE*, pages 1017–1025, April 2009.

[21] Daniel Lehmann, Liadan Ita Oćallaghan, and Yoav Shoham. Truth revelation in approximately efficient combinatorial auctions. *J. ACM*, 49(5):577–602, 2002.

[22] Jens Lischka and Holger Karl. A virtual network mapping algorithm based on subgraph isomorphism detection. In *Proceedings of ACM SIGCOMM VISA*, 2009.

[23] Y. Liu, H. Zhang, W. Gong, and D. Towsley. On the interaction between overlay routing and underlay routing. *IEEE INFOCOM*, 4:2543–2553 vol. 4, March 2005.

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

[25] Athina Markopoulou, Gianluca Iannaccone, Supratik Bhattacharyya, Chen-Nee Chuah, Yashar Ganjali, and Christophe Diot. Characterization of failures in an operational ip backbone network. *IEEE/ACM Trans. Netw.*, 16(4):749–762, 2008.

[26] A. Mas-Colell, M.D. Whinston, and J.R. Green. *Microeconomic theory*. Oxford University Press New York, 1995.

[27] A. Mas-Colell, M.D. Whinston, and J.R. Green. *Microeconomic theory*. Oxford University Press New York, 1995.

[28] Noam Nisan and Amir Ronen. Algorithmic mechanism design. In *Games and Economic Behavior*, pages 129–140, 1999.

[29] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, New York, NY, USA, 2007.

[30] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, second edition, 1998.

[31] David C. Parkes. Classic mechanism design. In *Iterative Combinatorial Auctions: Achieving Economic and Computational Efficiency*, chapter 2. PhD Thesis, University of Pennsylvania, 2001.

[32] David C. Parkes. Online mechanisms. In Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay Vazirani, editors, *Algorithmic Game Theory*, chapter 16. Cambridge University Press, 2007.

[33] David C. Parkes and Jeffrey Shneidman. Distributed implementations of Vickrey-Clarke-Groves mechanisms. In *Proc. 3rd Int. Joint Conf. on Autonomous Agents and Multi Agent Systems*, pages 261–268, 2004.

[34] David C. Parkes and Satinder Singh. An MDP-Based approach to Online Mechanism Design. In *Proc. 17th Annual Conf. on Neural Information Processing Systems (NIPS'03)*, 2003.

[35] David C. Parkes, Satinder Singh, and Dimah Yanovsky. Approximately efficient online mechanism design. In *Proc. 18th Annual Conf. on Neural Information Processing Systems (NIPS'04)*, 2004.

[36] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. In *Proceedings of HotNets–I*, Princeton, New Jersey, October 2002.

[37] Lili Qiu, Yang Richard Yang, Yin Zhang, and Scott Shenker. On selfish routing in internet-like environments. *IEEE/ACM Trans. Netw.*, 14(4):725–738, 2006.

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

[39] Tim Roughgarden and Éva Tardos. How bad is selfish routing? *J. ACM*, 49(2):236–259, 2002.

[40] Jeffrey Shneidman and David C. Parkes. Specification faithfulness in networks with rational nodes. In *Proc. 23rd ACM Symp. on Principles of Distributed Computing (PODC'04)*, pages 88–97, St. John's, Canada, 2004.

[41] Thomas E. Stern and Krishna Bala. *Multiwavelength Optical Networks: A Layered Approach*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

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

[43] Krishnaiyan Thulasiraman, Muhammad S. Javed, and Guoliang (Larry) Xue. Circuits/cutsets duality and a unified algorithmic framework for survivable logical topology design in ip-over-wdm optical networks. *IEEE INFOCOM*, 2009.

[44] Todimala and B. Ramamurthy. Approximation algorithms for survivable multi-commodity flow problems with applications to network design. In *Proc. of IEEE INFOCOM '06*, April 2006.

[45] Donald M Topkis. A k shortest path algorithm for adaptive routing in communications networks. *IEEE Transactions On Communications*, 36(7), july 1988.

[46] William Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *The Journal of Finance*, 16(1):8–37, 1961.

[47] Vivek Vishnumurthy, Sangeeth Chandrakumar, and Emin Gun Sirer. Karma: A secure economic framework for p2p resource sharing. In *Workshop on the Economics of Peer-to-Peer Systems*, Berkeley, California, 2003.

[48] Yi Wang, Eric Keller, Brian Biskeborn, Jacobus van der Merwe, and Jennifer Rexford. Virtual routers on the move: live router migration as a network-management primitive. In *SIGCOMM*, pages 231–242, New York, NY, USA, 2008. ACM.

[49] Minlan Yu, Yung Yi, Jennifer Rexford, and Mung Chiang. Rethinking virtual network embedding: Substrate support for path splitting and migration. *ACM SIGCOMM CCR*, 38(2):17–29, April 2008.

[50] Selwyn Yuen and Baochun Li. Strategyproof mechanisms for dynamic tree formation in overlay networks. In *INFOCOM*, pages 2135–2146, 2005.

[51] Selwyn Yuen and Baochun Li. Strategyproof mechanisms towards dynamic topology formation in autonomous networks. *MONET*, 10(6):961–970, 2005.

[52] Qin Zheng and Kang G. Shin. Fault-tolerant real-time communication in distributed computing systems. *IEEE Trans. Parallel Distrib. Syst.*, 9(5):470–480, 1998.

[53] Y. Zhu and M. Ammar. Algorithms for assigning substrate network resources to virtual network components. In *Proceedings of IEEE INFOCOM*, 2006.