Multi-Path Link Embedding for Survivability in Virtual Networks

by

Md Mashrur Alam Khan

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, 2015

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the 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.

Md Mashrur Alam Khan

Statement of Contributions

Most of the portions of this thesis appeared in the following publications:

- Md Mashrur Alam Khan, Nashid Shahriar, Reaz Ahmed, and Raouf Boutaba, "SiMPLE: Survivability in Multi-Path Link Embedding" in 11th Conference on Network and Service Management (CNSM), 9-13 November 2015 (To appear)
- Md Mashrur Alam Khan, Nashid Shahriar, Reaz Ahmed, and Raouf Boutaba, "SiMPLE: Survivability in Multi-Path Link Embedding". Programming of SAVI AGM & Workshop, 7 July 2015.

Abstract

Internet applications are deployed on the same network infrastructure, yet they have diverse performance and functional requirements. The Internet was not originally designed to support the diversity of current applications. Network Virtualization enables heterogeneous applications and network architectures to coexist without interference on the same infrastructure. Embedding a Virtual Network (VN) into a physical network is a fundamental problem in Network Virtualization. A VN Embedding that aims to survive physical (*e.g.,* link) failures is known as the Survivable Virtual Network Embedding (SVNE). A key challenge in the SVNE problem is to ensure VN survivability with minimal resource redundancy. To address this challenge, we propose SiMPLE. By exploiting path diversity in the physical network, SiMPLE provides guaranteed VN survivability against single link failure while incurring minimal resource redundancy. In case of multiple arbitrary link failures, SiMPLE provides maximal survivability to the VNs. We formulate this problem as an ILP and implement it using GNU Linear Programming Kit (GLPK). We propose a greedy proactive to solve larger instances of the problem in case of single link failures. In presence of more than one link failures, we propose a greedy reactive algorithm as an extension to the previous one, which opportunistically recovers the lost bandwidth in the VNs. Simulation results show that SiMPLE outperforms full backup and shared backup schemes for SVNE, and produces near-optimal results.

Acknowledgements

First of all, I would thank the almighty Allah, the most beneficent, the most merciful, for giving me a chance to pursue MMath program at the University of Waterloo.

I would like to express my deepest gratitude to my supervisor, Professor Raouf Boutaba, for his continuous support in my study and research, and for his patience, motivation, and knowledge. He always gave me freedom, and helped me all the time of my research and writing of this thesis. I would also like to thank the readers of this thesis, Prof. Reaz Ahmed and Prof. Martin Karsten, for providing their valuable feedback and comments on this work.

I am utterly grateful to my mother and brother for their unconditional love, support, and sacrifices. My mother has always been the source of strength and motivation for me. She was always beside me in my hard times, and constantly inspired me until I overcame the odds. My brother took over a lot of responsibilities in my family so that I can concentrate on my studies in Canada. It would not have been possible for me to make it this far without their support.

I would also thank my fellow coworkers, Dr. Reaz Ahmed and Mr. Nashid Shahriar, for their significant contributions in this work. They constantly helped me and guided me to make a successful research paper out of this work, which has been accepted in the 11th International Conference on Network and Service Management in November, 2015.

I would also acknowledge the contribution of the past and present members of the Network Virtualization research group for their feedback on this work, including Prof. Raouf Boutaba, Dr. Reaz Ahmed, Mr. Nashid Shahriar, Mr. Shihabur Rahman Chowdhury, Mr. Faizul Bari, Mr. Arup Raton Roy, Dr. Mohamed Faten Zhani, Mr. Md. Golam Rabbani, Mr. Zhihong Liu, Mr. Aimal Khan, Mr. Milad Ghaznavi, Mr. Min Feng, Mr. Leonardo Richter Bays and Mr. Aziz Lahlou.

Finally, I thank the University of Waterloo for providing me with adequate financial support, and for maintaining an excellent academic environment throughout my MMath program. This work was supported by the Natural Science and Engineering Council of Canada (NSERC) in part under its Discovery program, and in part under the Smart Applications on Virtual Infrastructure (SAVI) Research Network.

Dedication

This is dedicated to my family.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The Internet has to support a wide range of applications having diverse performance and functional requirements. For example, Audio/Video streaming requires dedicated bandwidth and bounded delay, online banking requires security guarantees, while web browsing and email applications are satisfied with best-effort delivery. Currently, these applications are deployed on the same network infrastructure, and rely on Internet's best-effort communication model that does not provision any guarantee. Network Virtualization (NV) [13] has been propounded as a promising solution for enabling heterogeneous applications and network architectures to coexist on the same physical infrastructure (or, substrate network, SN). NV involves two entities: Infrastructure Providers (InPs) and Service Providers (SPs). An InP owns and maintains the substrate, *e.g.,* data center networks. An SP, in contrast, requests network slices from one or more InP(s), and offers customized services to end users. An InP models a network slice as a Virtual Network (VN), and embeds the VN to the SN with proper isolation and guaranteed Quality of Service (QoS). In this way, NV enables multiple SPs to coexist on the same substrate without interference, and satisfies diverse application needs.

Efficient mapping of VNs onto an SN is known as the VN embedding (VNE) problem [18]. In its simplest form, the VNE problem is to map virtual nodes and links of a VN request onto substrate nodes and paths (sequence of physical links), respectively, while satisfying physical resource constraints. The VNE problem is $\mathcal{NP}$-hard and has been studied extensively in the literature [14], [33], [48]. However, one important aspect of the problem that received less attention is VN survivability. Finding a VN Embedding that can survive arbitrary substrate node or link failures is known as the Survivable Virtual Network Embedding (SVNE) problem [39]. A failure in the SN may cause multiple VNs to fail, which may significantly degrade service performance and availability. In many applications, a

service outage can incur high penalty in terms of revenue and customer satisfaction. For example, online businesses in North America lost 26.5 billion in revenue due to service downtime in 2010 [1]. Hence, VN survivability is crucial for both InPs and SPs.

Survivability has been thoroughly investigated in non-virtualized networks in the past [9], [30], [31], [42]. However, these solutions focus on ensuring network connectivity during failures, whereas in SVNE the focus is to preserve the virtual topology by using mutually exclusive substrate resources. Hence, existing solutions are not directly applicable to SVNE. Survivability of VNs is usually achieved through allocation of redundant (*i.e.,* backup) resources, which introduces additional challenges to the VNE problem. First, the failure characteristics and repair time are unpredictable [20], [34]. Reserving the full demand of a virtual link as backup is expensive, since backup resources remain idle when there are no failures [39]. To minimize resource wastage, shared backup schemes have been proposed in [22]. However, they do not guarantee the full requested bandwidth of a virtual link during failure. It is challenging to determine the minimum redundancy level for guaranteed survivability. Second, primary and backup resources need to be disjoint in the SN. Embedding each virtual link into multiple disjoint paths mitigates the impact of failures [35], [36], [46]. Although effective, this approach incurs path splitting overhead including packet redirection, increased routing table size, and packet reordering. In general, it is difficult to find the optimal trade-off between VN survivability, redundancy level, and path splitting overhead.

1.1 Problem Statement

To address the challenges mentioned in the previous section, we propose SiMPLE for ensuring **S**urvivability **i**n **M**ulti-**P**ath **L**ink **E**mbedding. SiMPLE presents a multi-path link embedding strategy by exploiting the path diversity in the SN. Link failures are more frequent than node failures, and node failures can be modeled as multiple link failures [41]. Hence, SiMPLE focuses on survivability against arbitrary substrate link failures. While embedding, SiMPLE assumes that the nodes in the VN request has already been mapped, and embeds each virtual link across a number of disjoint substrate paths. SiMPLE guarantees to preserve full demand of a VN request in presence of a single substrate link failure at a time instance. SiMPLE also provides a reactive mechanism to recover each virtual link affected by substrate link failures.

1.2 Motivation

The motivation of this thesis comes from the Infrastructure as a Service (IaaS) [8] business model, which facilitates provisioning virtualized resources over clouds, data centers, and ISP networks through the Internet. In addition, it decouples the role of the InPs and SPs. NV [13] provides an efficient and reliable way to deploy multiple heterogeneous VNs over the same SN. An InP deploys a large SN, and maintains its physical resources (*e.g.,* switches, links, CPU capacity, bandwidth). It also solves one (or more) instance(s) of the VNE problem to serve the arrived VN request(s). Each VN request comes from an SP, and has its own requirement (*e.g.,* topology, CPU demand, and bandwidth demand). While embedded, each VN facilitates its corresponding SP by hosting different application specific protocols and services for the end users. The VN requests are leased for a specific amount of time, after which an SP releases the SN resources and leaves. In this way, NV enables multiple heterogeneous VNs to coexist on the same SN while ensuring resource isolation, better manageability, security and privacy.

However, the SN is very likely to suffer from failures in its different components. Especially, the substrate link failures dominate in the SNs today, both in data center and ISP networks [20], [34]. The failure incidents show random behavior at their arrival rate and repair time. In a few cases, the links are failed on a planned manner, mostly due to maintenance reasons. However, as the authors in [34] have illustrated, approximately 80% of the link failures are unplanned, and they fail due to various unavoidable reasons (*e.g.,* misconfiguration, or fiber cut). In addition, the failed links may take a significant amount of time to repair. For example, the mean time to repair for a failed link in a data center may take as long as several hours [20], whereas, in ISP networks, it may take one or more days [34]. For these reasons, a VN embedding that survives substrate link failure(s) is often needed by both InPs and SPs. Hence, this work focuses on provisioning maximal survivability against substrate link failures on the embedded VNs.

1.3 Challenges

In this section, we present the major challenges for SiMPLE: *i.e.,* to design multi-path link embedding for provisioning virtual network survivability.

1.3.1 Maximal Survivability in Virtual Networks

We use the term *maximal survivability* to denote guaranteed survivability of the VN requests against arbitrary substrate component failures. To find an optimal algorithm for guaranteeing maximal survivability for VN requests is a challenging research area [18], [24]. This is because the substrate element failures are highly unpredictable [20], [34], and the exact amount of substrate resources that are needed for survivability is not known in advance [24]. For this reason, a number of research works consider only substrate link failures to achieve VN survivability [22], [35], [36], [39]. However, none of these works provide provable survivability guarantee, in presence substrate link failure(s).

1.3.2 Minimal Resource Redundancy

Survivability in VN request can be achieved through allocating redundant resources in the SN. This can be straightforward when we allocate a vast amount of redundant resources (*e.g.,* 100% or more redundancy). However, such an allocation strategy will be very costly, as well as it can leave a significant amount of resources to be idle when there are no failures present in the SN. In contrast, to determine the minimal amount of redundancy for survivability is a challenging problem. A number of factors, including the size of SN and VN, the unpredictability of failures, and the resource utilization in SN make it hard to determine both the amount and the allocation of redundant resources in embedding.

1.3.3 Minimal Path Splitting Overhead

We are assuming that the SN supports path splitting, *i.e.,* it can split a single data stream between the same source and destination across multiple paths. While embedding, path splitting can create additional overhead in the SN, *e.g.,* source and destination buffers, additional routing entries, and delays accumulated in multiple paths. If these overheads contribute too much, it will make VN embedding costly and infeasible. Furthermore, while minimizing path splitting, we may need to embed a virtual link with a huge demand only on one route. In the worst case, this route may not have enough residuals, and we may not have a feasible embedding. For this reason, minimizing the path splitting overhead is another key challenge in this thesis.

1.4 Contribution

The major contributions of this thesis are as follows:

- **Key concept.** We propose a novel concept to ensure maximal survivability while reserving only a fraction of the virtual link's demand as backup. To the best of our knowledge, SiMPLE is the only approach that provides provable survivability guarantee in presence of a single link failure without allocating full bandwidth of the virtual link's demand as backup.
- **Optimization model.** The design goal of SiMPLE is to find a trade-off between maximizing survivability and minimizing redundancy and path splitting overhead, which has not been considered in the previous studies. We formulate this joint optimization problem as an Integer Linear Program (ILP) to achieve the trade-off between maximizing survivability and minimizing both redundancy and path splitting overhead.
- **Proactive Approach.** We present an implementation of the ILP model in GLPK to find optimal solutions for small scale networks. For larger instances of the problem, we propose a greedy proactive algorithm that produces near-optimal solutions. We demonstrate SiMPLE's effectiveness through extensive simulations and compariosn with full backup and shared backup schemes for SVNE. Simulation results show that SiMPLE provides better survivability, requires lesser backup bandwidth, and generates more profit.
- **Reactive Approach.** While the proactive approach guarantees survivability against a single substrate link failure, it may not offer full protection for multiple substrate link failures. To mitigate the impact of such scenarios, we present a reactive approach in SiMPLE. This approach recovers the lost bandwidth in each virtual link affected by physical link failures. Simulation results show that, compared to the proactive approach, it improves the results by a significant margin in both minimizing failed VNs and obtaining higher profits for the InP.

1.5 Thesis Organization

The rest of the thesis is organized as follows.

- **Chapter 2** provides necessary background for this thesis. This chapter contains discussions on virtual network embedding, link failures, survivable virtual network embedding, and path splitting.
- **Chapter 3** presents the main concept and ILP model for SiMPLE. It also presents two greedy solutions – one proactive and one reactive – each representing one part of SiMPLE.
- **Chapter 4** presents our evaluation results. In this chapter, we compare SiMPLE with two existing SVNE approaches – Full Backup Scheme (FBS) and Shared Backup Scheme (SBS).
- **Chapter 5** concludes the thesis with an outline of possible future research directions.

Chapter 2

Background

In this chapter, we present the backgrounds studies that are required for understanding this thesis. In Section 2.1, we explain the virtual network embedding process, with each of its steps described. In Section 2.2, we present the definition and characteristics about substrate link failures, both in data center and ISP networks. In Section 2.3, we discuss how we can provide survivable virtual network embedding (SVNE) to handle link failures. Afterwards, we present path splitting – its basic concepts, enabling technologies, and applicability in VNE and SVNE in Section 2.4. Finally, we present a summary of this chapter in Section 2.5.

2.1 Virtual Network Embedding

In this section, we describe the representation of the substrate network (SN), the virtual network (VN), and discuss the basic approaches to perform virtual network embedding.

2.1.1 Substrate Network

We model the Substrate Network (SN) as an weighted graph $G^S(N^S, E^S)$. In this notation, N^S and E^S denote the sets of the Substrate Nodes (SNodes) and Substrate Links (SLinks), respectively. Each SNode $n^s \in N^S$ has a CPU capacity, $c(n^s)$, and each SLink $e^s \in E^S$ has a bandwidth capacity, $b(e^s)$. The residual CPU and bandwidth resources at n^s and e^s are represented by $r(n^s)$ and $r(e^s)$, respectively.

2.1.2 Virtual Network

Similar to the SN, we model the Virtual Network (VN) as an weighted graph $G^V(N^V, E^V)$. Here, N^V and E^V denote the sets of Virtual Nodes (VNodes) and Virtual Links (VLinks), respectively. The CPU demand of a VNode $n^v \in N^V$ and bandwidth demand of a VLink $e^v \in E^V$ are denoted by $c(n^v)$ and $b(e^v)$, respectively.

2.1.3 VN Embedding

The idea of the VN Embedding (VNE) problem is to embed the VN request onto the SN so that the VN demands are satisfied, and the SN capacities are not exceeded. In VNE, the key challenge is to use as few resources from the SN as possible, so that more VNs can be accommodated and a higher revenue is achieved by the InP. Generally, the VNE problem can be divided into two stages as follows.

Node Embedding

In this stage, each VNode $n^v \in N^V$ from a VN request is mapped to a single SNode by a node mapping function: $\xi_N : N^V \rightarrow N^S$, subject to CPU capacity constraints: $\forall n^v \in N^V : c(n^v) \leq r(\xi_N(n^v))$.

Link Embedding

In this stage, each VLink $e^v \in E^V$ is mapped to a substrate path $p^{e^v} \in P^{e^v}$ between ingress SNode $\xi_N(e^v_s)$ and egress SNode $\xi_N(e^v_d)$, where e^v_s and e^v_d respectively denote the source and destination VNodes of e^v. The link mapping function is $\xi_E : E^V \rightarrow P^{e^v}$, subject to bandwidth capacity constrainst: $\forall e^v \in E^V \wedge \forall p \in P^{e^v} : b(e^v) \leq r(p)$, where $r(p) = \min\limits_{e^s \in p} r(e^s)$.

Solving the VNE problem is $\mathcal{NP}$-hard, as it is related to the multi-way separator problem [33]. Even with a given VNode mapping, the problem of optimally allocating the VLinks to substrate paths reduces to the unsplittable flow problem [27], and is thus $\mathcal{NP}$-hard as well.

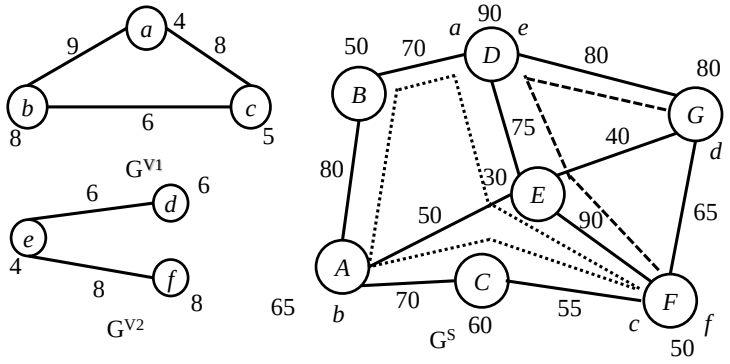


Figure 2.1: Embedding two VN requests onto an SN

VNE: An Illustrative Example

Fig. 2.1 depicts the mapping of the two VN requests, G^{V1} and G^{V2} (on the left) on an SN, G^S (on the right). Here, the SNodes and VNodes are labeled with letters inside the corresponding node. Node mapping for G^{V1} is $\xi_N^1(a) = D$, $\xi_N^1(b) = A$, $\xi_N^1(c) = F$, and link mapping is $\xi_E^1(ab) = DBA$, $\xi_E^1(ac) = DEF$, $\xi_E^1(bc) = ACF$; while G^{V2} has node mapping $\xi_N^2(e) = D$, $\xi_N^2(d) = G$, $\xi_N^2(f) = F$, and link mapping $\xi_E^2(ed) = DG$, $\xi_E^2(ef) = DEF$.

2.1.4 VNE Literature Survey

The VN Embedding literature can be classified based on the coordination between its two sub-problems – node embedding and link embedding [24]. The first category, *i.e.,* the uncoordinated VNE, does not take into account the possibility of coordination between these two sub-problems. One possible example of this category is presented in [46], where the link embedding is performed after a greedy node embedding. In the link embedding step, the multi-commodity flow problem formulation is used to find a set of paths between the substrate hosts for each virtual link. However, this lack of coordination might result into embedding adjacent virtual nodes towards distant physical nodes, which increases resource consumption. The second category, *i.e.,* the one-stage coordinated VNE [12], embeds the virtual links at the same time as the virtual nodes. In this category, at first, a greedy algorithm (*e.g.,* PageRank [37]) is used to rank the virtual nodes. Then, a virtual node pair and their intermediate virtual link are embedded. Afterwards, the remaining virtual nodes and virtual links connecting them are embedded one by one. The third category, *i.e.,* the two-stage coordinated VNE, embeds the virtual network into the substrate network

into two distinct steps [15]. The first step is to represent a set of preferred substrate nodes for each virtual node as a meta-node, and perform a relaxed Mixed Integer Programming (MIP) to calculate the physical hosts of the virtual nodes in the corresponding meta-nodes. The second step is to find the physical paths corresponding to the virtual links, and this is solved by the multi-commodity flow problem formulation. In the one-stage and two-stage coordinated VNE approaches, the coordination between node mapping and link mapping stages are strong, and the substrate nodes chosen to map the virtual nodes are likely suited to provide virtual link mappings with low embedding cost.

The VNE approaches described so far are applicable when the virtual network is embedded into only one substrate network. However, in real-world scenario, a large virtual network may be embedded in a distributed fashion across several autonomous substrate networks. In the latter case, the virtual network is split into several virtual networks, each of which are embedded in different substrate networks. The connectivity between the smaller virtual networks is maintained by the external links among different SN. A good example of such distributed VNE algorithm is presented in [16].

2.2 Link Failures

An SLink Failure is defined as the event when the connection between two adjacent nodes (routers or switches) is down in a network. SLink failures are very common in both data center and in ISP networks, as identified by the authors in [20] and [34], respectively. According to these studies, a single SLink failure is defined as the event on which only one SLink is down at a certain time. For example, an SLink e^f can be down at time t_1, and it can be repaired at time t_2, where $t_2 > t_1$. If no other SLink is down during the time interval $[t_1, t_2]$, then this incident is called single SLink failure event. The time interval $(t_2 - t_1)$ is defined as time-to-repair (TTR). In contrast to the single SLink failure event, there can be multiple SLink failures in the network, where more than one SLink experience down events simultaneously. As an instance of multiple SLink failure, consider a router failure, which can trigger down events for all the connected SLinks. Furthermore, multiple single SLink failures can have overlapping failure time, which generates another type of multiple SLink failures. Multiple SLink failures can further be categorized into two separate classes – simultaneous and concurrent failures. In simultaneous failure scenario, multiple SLinks fail at the same time, and typically due to the same incident (*e.g.*, router failure). In concurrent failure scenario, multiple SLinks fail independently, and have an overlapping TTR.

In Sections 2.2.1 and 2.2.2, we will present the characteristics of SLink failures in data center and ISP networks, respectively.

2.2.1 Link Failures in Data Center Networks

Authors in [20] present some interesting characteristics of failures in data center networks. In today's data center networks, a number of factors may contribute to SLink failures, mostly dominated by hardware (*e.g.,* power supply or fan failure) and software (*e.g.,* BIOS upgrade, OS reboot) issues. Notably, SLink failures happen about ten times more than SNode failures per day. Failure events arrive randomly at data center networks, and they show high variability in their rate of occurrence. However, half of the time SLink failures are single failure occurrences. The TTR for these failures are also variable in nature. Statistically, almost 60% of the SLink failures are repaired within five minutes. However, depending on the type of SLink, the TTR can also be significantly long (*e.g.,* several hours). In these cases, the SN may experience another random failure, resulting into the multiple failure scenario. Furthermore, although the data center networks provide redundancies to prevent packet loss in failure scenario, these mechanisms are only able to reduce the impact of failures by no more than 40%.

2.2.2 Link Failures in ISP Networks

The characteristics of failures in ISP networks are presented in [34]. An SLink in an ISP network may not operate properly all the time due to various reasons, such as fiber cut, maintenance, mis-configuration etc. There are some planned failures for scheduled maintenance, and these activities are performed periodically by all Infrastructure Providers (InPs). However, 80% of total SLink failures are unplanned (*e.g.,* not scheduled due to maintenance) in ISP networks [34]. Note that 70% of these failures are single SLink failures, whereas the other 30% generate multiple failure events. The TTR in ISP networks can be significantly higher than that of the data center networks. Most of the failures in ISP networks can be repaired within an hour. However, depending on the failure and the affected SLink, the TTR can even be a day or more. In these cases, multiple SLink failures are very likely to occur.

2.3 Survivable Virtual Network Embedding

Survivable Virtual Network Embedding (SVNE) deals with keeping the VNs intact, even after substrate failures (*e.g.,* SNodes or SLinks). SNode failures are very rare and results into multiple SLink failures [20], [34], [41]. Hence, majority of the SVNE literature focuses on SLink failures. Throughout the rest of this thesis, we restrict ourselves on SLink failures. Two most prominent solutions are common in the SVNE literature [18], [24], namely, protection, and recovery. We describe each one in context of IP networks in Sections 2.3.1 and 2.3.2, respectively. Finally, in Section 2.3.3, we will present survivability mechanisms in other domains (*e.g.,* optical, MPLS, and cloud environments).

2.3.1 Protection

In SVNE literature, protection (also known as, *proactive allocation*) refers to allocating redundant resources per each VLink while embedding, *i.e.,* before any SLink failure occurs. To survive against SLink failures, backup resource can be allocated in two ways [24], namely, SLink protection and path protection. In SLink protection, a primary path p is associated to each VLink, and each SLink $e^s \in p$ is protected by a detour. Upon an SLink failure, traffic on that SLink is locally rerouted through its detour. In Fig. 2.1, SLink DE can be associated with two detours DGE and $DBAE$ for the VLinks ac and ef, respectively. In case of the path protection, each end-to-end primary path p is protected by an SLink disjoint backup path from source to destination. The source activates the backup path when it is notified about the failure of an SLink along path p. In Fig. 2.1, the bandwidth demanded by VLinks ac and ef can be reserved in the backup paths DGF and $DBACF$, respectively, which are SLink disjoint to the primary path DEF. Hence, redundant bandwidth has to be allocated in SN for each backup path. However, to minimize redundancy multiple backup paths can share the same redundant bandwidth in SLinks.

A number of research works contribute to the VLink protection problem. Several research works, including [10] and [39], formulated two separate LP models for VLink embedding, which allocate full demand of each VLink along a primary path and a disjoint backup path. These full backup schemes result into poor bandwidth utilization. Shared backup schemes, on the other hand, allow multiple VLinks to share backup resources allocated to each end-to-end path [11] or SLink [22]. However, primary paths are dedicated to each VN and cannot be shared with other VNs. Since the same backup resources are shared among multiple VLinks, these approaches do not offer bandwidth guarantee, even in presence of a single SLink failure. The authors in [22] propose two shared backup schemes

to protect against any potential single SLink failure: Shared On-Demand approach and Shared Pre-Allocation approach. In the first approach, bandwidth resources are allocated to the primary flows and backup flows upon the arrival of each VN request. Backup resources can be reused by other VNs to make room for accepting more incoming VN requests. However, primary flows are dedicated to each VLink and are not allowed to be shared with other VLinks. In the second approach, backup bandwidth for each SLink is pre-allocated during the initial phase, *i.e.,* before any VN request arrives. Since the bandwidth pre-allocation only needs to be done once and not for every VN request, it requires less processing during the VN embedding phase. The disadvantage of these approaches is that they can not guarantee recovery of full bandwidth even in the case of single SLink failure. For example, if multiple VLinks are embedded on the failed SLink, bandwidth recovery of these VLinks can be compromised due to sharing.

2.3.2 Recovery

The recovery (or, reactive) techniques provide VN survivability without allocating any backup bandwidth at the beginning. In practice, they react after an SLink fails, and start the path restoration mechanism. B. Lu [32] proposes such a reactive mechanism where either a substitute path is searched, or the corresponding VN is remapped after a link failure. In a highly saturated SN, there may not be enough resources left for finding the substitute path or remapping the VN.

We discuss two most prominent recovery approaches in SVNE. Rahman et al. [38] [39] proposes a three-phase hybrid mechanism where a set of possible backup detours for each SLink is computed before any VN request arrives. Then, node embedding is done for the arriving request with an existing embedding algorithm [14], [48], followed by a multi-commodity based link embedding. Finally, in the event of a SLink failure, a reactive online optimization mechanism reroutes the affected flows along candidate backup detours selected in the first phase. This approach may demand a long convergence time, leaving VNs unavailable during such periods. This may cause data loss. Furthermore, since the substrate resources are not fragmented to serve multiple virtual requests, all proactive and reactive approaches may suffer from improper load balancing and link underutilization.

2.3.3 Survivability in Other Domains

Network survivability in Optical and Multi-Protocol Label Switched (MPLS) networks is usually considered during the network design. The solutions in these domains, *e.g.,* [30],

[31] assume that traffic demands are known in advance (*i.e.,* offline). In contrast, SVNE is online; it needs to provide survivability for unpredictable VN request arrivals and demand patterns. Furthermore, SVNE solutions have to ensure the intactness of all VLinks in presence of failures. This restriction is not present in Optical/MPLS networks, where the goal is to ensure connectivity in the network. Hence, these solutions are not suitable for the NV environment.

Due to the importance of providing high service availability in Cloud environments, recently there is a trend towards designing survivable resource allocation schemes for bandwidth constrained data centers [4], [7], [45], [47]. These works aim at improving fault tolerance by spreading out network elements across multiple failure domains. Xu et al. [43] proposes a resource allocation scheme for provisioning virtual data centers with backup virtual machines and links. However, this work do not consider the survivability of physical machines and links. Bodik et al. [9] proposes an optimization framework for improving survivability while reducing the total bandwidth consumption. However, this approach does not consider the heterogeneous failure rates of the underlying physical equipment, and mitigates the bandwidth bottleneck only in the core of the data center. Zhang et al. [47] proposes a framework for reliable virtual data center embedding in clouds by considering heterogeneous failure rates. SiMPLE differs from these works in its objective of simultaneously optimizing VN survivability, bandwidth usage, and path splitting overhead.

2.4 Path Splitting

Path splitting is a routing strategy to allow one data stream to be splitted across multiple paths. Path splitting enables an increased resource utilization, failure-tolerance, and better QoS. Various path splitting techniques, such as, Equal-cost Multipath Routing (ECMP) and Multipath TCP (MPTCP), have been used in the IP and TCP layers, respectively. We discuss each of them in Section 2.4.1 and 2.4.2, respectively. Finally, in Section 2.4.3, we elaborate how path splitting can be utilized in VNE or SVNE context.

2.4.1 Equal Cost Multi Path Routing

Equal Cost Multi Path (ECMP) routing [25] is an extension of the shortest path routing [5], which allows multiple paths between a source and a destination to forward packets. At the beginning, just like the shortest path routing protocol, it calculates the shortest paths between the source and the destination. In contrast to the shortest path routing protocol

that randomly selects one of the next hops in case of a tie, ECMP saves all next hops in the current hop. Afterwards, ECMP forwards packets having the same flow ID (source and destionation IP addresses, and port numbers) towards the same path. In this process, ECMP distributes multiple flows across multiple paths to achieve better load balancing. Since ECMP does not actually split one flow across multiple paths, it has no jitter (*i.e.,* delay incurred in different paths carrying the same flow). However, the working principle of SiMPLE differs from ECMP, because SiMPLE works under the assumption that the SN can split each flow across multiple paths.

2.4.2 Multipath TCP

The goal of the Multipath TCP (MPTCP) [3], [19], as proposed by the Internet Engineering Task Force (IETF) is to utilize the network resources by splitting a single data stream across multiple, potentially disjoint paths. For this purpose, MPTCP assumes the presence of multiple IP addresses at a host. The presence of multiple network addresses at one host are used to find multiple paths. At the beginning, MPTCP operates by establishing a regular TCP connection (*i.e.,* flow) between the source-destination pair. Afterwards, it makes opportunistic discoveries for additional source-destination paths. If such path(s) are found, MPTCP divides the flow traffic among the discovered paths (*i.e.,* subflows). The MPTCP protocol operates between the IP layer and the application layer in the network stack. For the non-MPTCP-aware applications and/or hosts, MPTCP acts just like the traditional TCP. The underlying assumption of SiMPLE, *i.e.,* to split each flow across multiple paths, is very similar to MPTCP. However, in contrast to MPTCP, SiMPLE provisions guaranteed survivability by ensuring the disjointness constraint among the working paths.

2.4.3 Path Splitting in VNE/SVNE

The notion of path splitting can be used in both VNE and SVNE contexts. The key idea to use path splitting in VNE or SVNE is to allow multiple paths, instead of just one, for embedding each VLink. Each path provisions a portion of the demand requested by the corresponding VLink. In this process, the whole requested demand of the VLink is obtained by the summation of the resources provided by these paths. In this concept, the major advantage is that, even if one path is affected by an SLink failure, the other remaining paths can serve a significant portion of the VLink demand. As an instance of path splitting in VNE, let us again consider the example presented in Section 2.1.3. In Fig. 2.1, we can embed VLinks ac and ef onto multiple paths such as $\xi_E^1(ac) = \{DEF, DGF\}$

and $\xi_E^2(ef) = \{DEF, DBACF\}$. Here, upon the failure of the SLink DE, both VLinks ac and ef can partially survive through the alternate paths DGF and $DBACF$, respectively.

A number of works in both VNE and SVNE literatures use path splitting, and we summarize the most prominent ones as follows. The authors in [46] introduced path splitting in VNE to embed a VLink over multiple substrate paths. For this purpose, they use multi commodity flow (MCF) based link embedding. In an MCF based solution, any intermediate SNode between the end hosts can split the flow. However, in SiMPLE, we assume that only the source and destination SNodes can split the flow, hence MCF cannot be used in this context. R. Oliviera et al. [35], [36] proposed an embedding strategy that provides opportunistic recovery for each VLink via path splitting. In these cases, the authors use heuristics to find a set of paths between the source and destination SNodes. Upon failure of each SLink, their proposed algorithm uses any spare capacity in the other existing paths in SN to recover the lost bandwidth. In this process, this algorithm mitigates the impact of failure by switching the affected traffic on the failed SLink to alternative paths. In the worst case, it can salvage a fraction of the VLink's bandwidth during an SLink failure. However, these approaches do not guarantee VN survivability in presence of a single SLink failure.

2.5 Summary

In this chapter, we have presented the necessary background materials for this thesis. We have introduced the basic concepts of VNE, followed by an illustrative VNE example and VNE literature survey in Section 2.1. In Section 2.2, we have presented the definition and characteristics of SLink failures, in context of data center and ISP networks. Afterwards, we have presented the basic concepts, example, classification, and literature survey on SVNE in Section 2.3. Finally, in Section 2.4, we have introduced the notion of path splitting, along with its prominent enabling technologies, and its usage on VNE and SVNE literatures.

Chapter 3

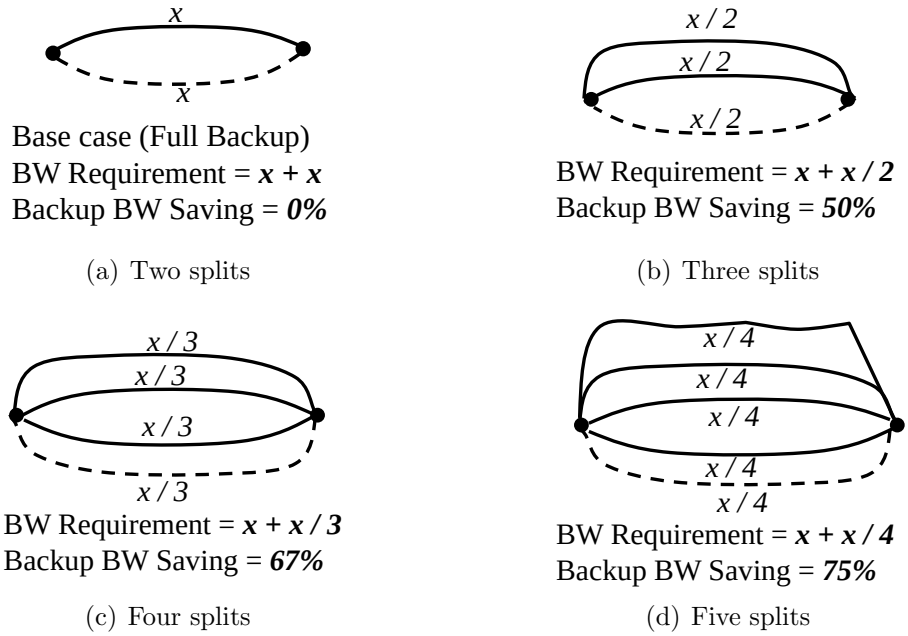
SiMPLE

In the previous chapter, we presented the initial concepts on SVNE. We also discussed that the existing SVNE approaches do not provide guaranteed protection for a single SLink failure. In addition, we have seen that SLink failure(s) are a very common phenomena, both in data center and ISP networks. Motivated by these facts, we propose SiMPLE (**S**urvivability **i**n **M**ulti **P**ath **L**ink **E**mbedding). SiMPLE consists of two parts – one proactive allocation strategy to provision guaranteed survivability against a single SLink failure, and one reactive recovery strategy to opportunistically recover a VLink affected by an SLink failure.

The remainder of this chapter is organized as follows. In Section 3.1, we describe the main embedding concept of SiMPLE, and explain its two parts. In Section 3.2, we present the ILP formulation of SiMPLE. In Section 3.3, we present two solutions for each part in SiMPLE – `SiMPLE-PR` and `SiMPLE-RE`, with illustrative examples. Finally, we provide a summary of this chapter in Section 3.4.

3.1 The Embedding Concept in SiMPLE

The main concept of SiMPLE embedding consists of two parts – proactive allocation, and reactive recovery. In this section, we describe the main concepts of these parts.



(a) Two splits

(b) Three splits

(c) Four splits

(d) Five splits

Figure 3.1: Proactive Embedding in SiMPLE

3.1.1 Proactive Allocation

The proactive embedding concept of SiMPLE is illustrated in Fig. 3.1. A basic proactive approach for SVNE, the Full Backup Scheme (FBS), is illustrated in Fig. 3.1(a). In this case, a VLink with demand x is embedded onto two disjoint paths with sufficient residual capacity. One of the paths acts as the primary (denoted with solid line), whereas the other is reserved for the backup (dashed line). When an SLink in the primary path fails, the backup path serves the VLink traffic. When the failed SLink recovers, the primary path starts serving the VLink again. However, such technique provisions twice the demand of each VLink. As a result, the number of accepted VNs and SLink utilization decreases significantly.

SiMPLE operates according to Fig. 3.1(b) – Fig. 3.1(d). In Fig. 3.1(b), the VLink is split into three disjoint substrate paths, and $x/2$ bandwidth is allocated to each of them. In this case, two paths are used to carry the primary flow, whereas the third path is used as backup. Since these paths are disjoint, at most one of them can be affected by a single SLink failure. If an SLink fails, the two unaffected paths deliver the requested bandwidth x. Note that only half of the requested bandwidth is allocated in the backup path, or, in other words, 50% backup bandwidth is saved in contrast to FBS. We can extend this idea to

a higher number of splits, say k. Fig. 3.1(c) and Fig. 3.1(d) present the VLink embedding scenario for $k = 4$ and 5, respectively. As highlighted in these figures, 67% and 75% backup bandwidth is saved in these two cases, respectively. In addition, the splitting of each VLink into multiple substrate paths improves the possibility of VN request acceptance; even if the full requested bandwidth is not available in any of the SLinks, a VLink can be embedded by splitting the required bandwidth over multiple paths. In other words, it utilizes the SLinks more efficiently than FBS, and increases the number of accepted VNs. However, increasing the number of splits introduces additional overhead, which must be taken into consideration.

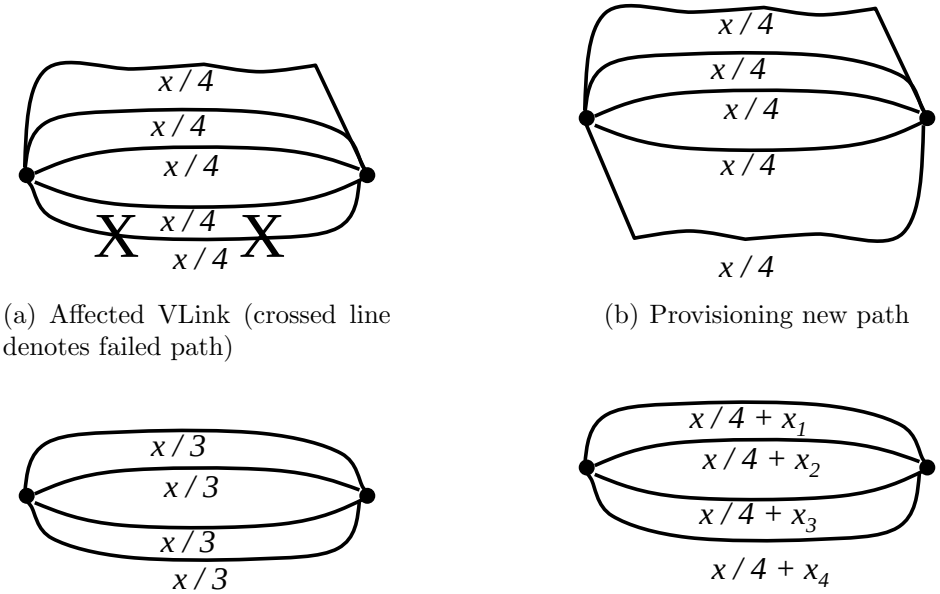
There is a trade-off between the number of splits, and VN embedding overhead. Indeed, each path splitting has a cost in terms of routing entry updates, source and destination buffers, and additional SLink delays. We formulate these costs mathematically in Section 3.2. If we increase the number of splits too much, these costs may result into infeasible VN embeddings. In Section 4.1.1, based on our experimental results, we show that an optimal embedding should not exceed more than five splits, for different workloads.

Theorem 3.1.1. *SiMPLE proactive embedding guarantees to preserve the full demand of every embedded VLink in case of a single SLink failure.*

Proof. We prove this Theorem by contradiction. Assume that a VLink $\tilde{e}^v \in E^V$ is not supported with its full demand. According to the SiMPLE working principle, at least two paths $p_1^{\tilde{e}^v}$ and $p_2^{\tilde{e}^v}$ in $\tilde{e}^v$ are impacted by a single SLink failure. By definition, $p_1^{\tilde{e}^v}$ and $p_2^{\tilde{e}^v}$ are disjoint, (*i.e.,* they have no common SLink), and this leads to a contradiction. □

Theorem 3.1.2. *While embedding VNs with same characteristics (*e.g., *size, demand, and arrival rate), SiMPLE proactive embedding outperforms FBS in the number of accepted VNs by a factor of* $2\left(\frac{k-1}{k}\right)$*, where* k *is the average number of splits in embedding a VLink.*

Proof. Assume that FBS and SiMPLE are evaluated for time T, and accepted a series of n VNs. Each VN has ν VLinks, and each VLink demands x bandwidth. At time T, substrate bandwidth consumptions of FBS and SiMPLE are given by $\mathcal{B}_F^T = 2n\bar{l}\nu x$ and $\mathcal{B}_S^T = kn\bar{l}\nu\frac{x}{k-1}$, respectively, where $\bar{l}$ is the average length of the substrate paths used in embedding. The additional substrate bandwidth used in FBS is given by $\mathcal{B}_F^T - \mathcal{B}_S^T = n\bar{l}\nu x\frac{k-2}{k-1}$. Before SiMPLE consumes bandwidth $\mathcal{B}_F^T$, an additional $\hat{n}$ VNs with same characteristics would occupy $\hat{\mathcal{B}}_S^T$ bandwidth, where $\hat{\mathcal{B}}_S^T = k\hat{n}\bar{l}\nu\frac{x}{k-1}$. Since $\hat{\mathcal{B}}_S^T = \mathcal{B}_F^T - \mathcal{B}_S^T$, we obtain, $\hat{n} = n\frac{k-2}{k}$. Hence, the number of accepted VNs in SiMPLE is $n + \hat{n} = 2\frac{k-1}{k}n$. □



(a) Affected VLink (crossed line denotes failed path)

(b) Provisioning new path

(c) Fixed allocation

(d) Variable allocation

Figure 3.2: Reactive Failure Recovery in SiMPLE

3.1.2 Reactive Recovery

The proactive approach described in the previous section guarantees survivability of each VLink in case of a single SLink failure. However, this approach can be vulnerable in presence of multiple SLink failures, especially when the failures occur within a very short time window. This is because multiple failures with overlapping MTTR can affect different paths used in the embedding of a single VLink.

Theorem 3.1.3. *While embedding VNs with same characteristics (*e.g., *size, demand, and arrival rate), if two SLink failures are present in an SN, they affect the same VLink with a probability in* $[\mathcal{C}, \alpha\mathcal{C}]$*, where* α *is the expected number of VLinks embedded onto an SLink, and* $\mathcal{C}$ *is a constant depending on the SN and VN.*

Proof. Assume that k and l denote the average number of splits and average path length while embedding a VLink, respectively. The first SLink failure will affect α VLinks, each of which have $(k-1)l$ critical SLinks. Note that if any of these critical SLinks fail, the corresponding VLink will be served with less than 100% of its demand. In one extreme, these $\alpha(k-1)l$ SLinks may be shared among $(k-1)l$ SLinks. In the other, they may be

Table 3.1: Approximate values for Theorem 3.1.3

Term	Value
Number of Splits, k	$4-5$
Average Path length, l	$4-5$
Average Number of VLink per SLink, α	$3-4$
Number of Substrate Links, L	500

totally disjoint. The probability of a second failure in one of these SLinks can be between $\mathcal{C}$ and $\alpha\mathcal{C}$, where L is the total number of SLinks, and $\mathcal{C} = \frac{(k-1)l}{L}$. □

Observation 3.1.1. *We conducted experiments to determine approximate values for the terms presented in Theorem 3.1.3, which are given in Table 3.1. By substituting these values, we estimate that the probability of two SLink failures affecting the same VLink lies within* $2-16\%$ *on an SN with* 500 *SLinks.*

To handle the impact of multiple failures, we propose a reactive approach in SiMPLE. This approach, as presented in Fig. 3.2, works for each VLink that are affected by an SLink failure. In essence, this approach considers each possibility to recover an affected VLink (a VLink with a failed path, see Fig. 3.2(a)), and selects the most feasible one. The first possibility, as shown in Fig. 3.2(b), is to provision another link-disjoint path for the affected VLink. The second and third possibilities, as shown in Fig. 3.2(b) and 3.2(c) respectively, consider increasing the lost bandwidth by a fixed or variable amount among the other working paths that had already been used in embedding. Among these possible alternatives, SiMPLE considers a set of certain criteria (*e.g.,* amount of physical resources used, load balancing) to evaluate their goodness, or *cost*. A detailed description of the cost function can be found in Section 3.2. The embedding contributing to the lowest cost is chosen by SiMPLE.

Note that, in the worst case, none of these possibilities will work. This implies that another link disjoint path does not exist, and the existing paths do not have enough residual capacity to support the bandwidth lost due to the failure(s). In this case, the VLink will be served with less than 100% of its demand. Nonetheless, we argue that this case can only arise when a highly congested network is subject to a massive number of failures in a very short time, which is often less frequent.

3.2 ILP Formulation

We use the following notations to represent different aspects of embedding, which are also summarized in Table 3.2. The set P^{e^v} represents a set of disjoint paths $\{p_1^{e^v}, p_2^{e^v}, \ldots, p_k^{e^v}\}$ in SN where a VLink e^v is embedded. Note that the number of paths in P^{e^v} will be equal to the number of splits for $e^v \in E^V$, *i.e.,* $|P^{e^v}| = k^{e^v}$. Two boolean variables are defined as follows.

$$X(n^v, n^s) = \begin{cases} 1, & \text{if } n^v \text{ is embedded to } n^s \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

$$Y(p_i^{e^v}, e^s) = \begin{cases} 1, & \text{if the path } \ p_i^{e^v} \text{ contains } e^s \\ 0, & \text{otherwise} \end{cases} \tag{3.2}$$

We formulate SiMPLE as an ILP Model. In this model, we optimize both the number of splits and the set of substrate paths for each VLink of a VN such that the overall embedding cost is minimized. Afterwards, the corresponding VLinks are mapped to optimal sets of paths. The VN embedding cost has the following three components.

3.2.1 Split and Join Cost

The first cost is the split and join cost at the source and destination SNodes for a VLink e^v. In SiMPLE, we assume that the SN supports path splitting, and this assumption relies on the substrate switches. This is because each data stream is *split* at the ingress switch, and subsequently *joined* at the egress switch. Without loss of generality and for simplifying the formulation, we do not place any cap on the number of splits and joins per SNode. Let $d_1(n^s, k)$ and $d_2(n^s, k)$ be the splitting and joining costs into k branches at $n^s \in N^S$. The total split and join cost at n^s is denoted by $D(n^s, k) = d_1(n^s, k) + d_2(n^s, k)$. We can represent the total split and join cost as follows in (3.3).

$$\ddot{I}(e^v, P^{e^v}, k^{e^v}) = \left(D(\xi_N(e_s^v), k^{e^v}) + D(\xi_N(e_d^v), k^{e^v})\right) \tag{3.3}$$

Table 3.2: Notation Table

$G^S(N^S, E^S)$	Substrate Network (SN)
$n^s \in N^S$	Substrate Node (SNode)
$e^s \in E^S$	Substrate Link (SLink)
$c(n^s)$	Total CPU capacity of $n^s \in N^S$
$b(e^s)$	Total Bandwidth capacity of $e^s \in E^S$
$r(n^s)$	Residual CPU capacity of $n^s \in N^S$
$r(e^s)$	Residual Bandwidth capacity of $e^s \in E^S$
$d_1(n^s, k)$	Split cost into k branches for $n^s \in N^S$
$d_2(n^s, k)$	Join cost into k branches for $n^s \in N^S$
$D(n^s, k)$	Split and Join cost into k branches for n^s $D(n^s, k) = d_1(n^s, k) + d_2(n^s, k)$
$\ddot{I}(e^v, P^{e^v}, k^{e^v})$	Split and join cost for e^v
$\beta(n^s)$	Switching cost of $n^s \in N^S$
$\ddot{S}(e^v, p_i^{e^v})$	Switching cost for e^v
$\delta(e^s)$	Link delay of $e^s \in E^S$
$\ddot{L}(e^v, p_i^{e^v}, k^{e^v})$	SLink cost for e^v
$G^V(N^V, E^V)$	Virtual Network (VN) Request
$n^v \in N^V$	Virtual Node (VNode)
$e^v \in E^V$	Virtual Link (VLink)
$c(n^v)$	Total CPU demand of $n^v \in N^V$
$b(e^v)$	Total Bandwidth demand of $e^v \in E^V$
$T(G^V)$	Lifetime of G^V
ξ_N	Node Mapping function
ξ_E	Link Mapping function
w^N	Relative weight of the nodes
w^L	Relative weight of the links
P^{e^v}	The disjoint path set $\{p_1^{e^v}, p_2^{e^v}, \ldots, p_k^{e^v}\}$ where $e^v \in E^V$ is embedded; $\lvert P^{e^v} \rvert = k^{e^v}$
e_s^v	Source VNode of e^v
e_d^v	Destination VNode of e^v
$X(n^v, n^s)$	1, if n^v is embedded to n^s, 0 otherwise
$Y(p^{e^v}, e^s)$	1, if path p^{e^v} uses e^s, 0 otherwise

3.2.2 Switching Cost

The second cost is the packet switching cost, and it is presented in (3.4) as $\ddot{S}(e^v, p_i^{e^v})$. This cost is associated with each mapped path of e^v due to forwarding the fragmented data stream between the source and destination SNodes. For such a path $p_i^{e^v} \in P^{e^v}$, all intermediate SNodes forward each flow to the next appropriate SNode. Without loss of generality and for simplifying the formulation, we do not place any cap on the number of switchings per SNode. The switching cost at $n^s \in N^S$ is denoted by $\beta(n^s)$.

$$\ddot{S}(e^v, p_i^{e^v}) = \sum_{n^s \in p_i^{e^v}} \left(\frac{c(n^s)}{r(n^s)} \beta(n^s) \right) \tag{3.4}$$

3.2.3 SLink Cost

The third and final cost component, SLink cost, is given by $\ddot{L}(e^v, p_i^{e^v})$ in (3.5). This cost represents the sum of allocated substrate bandwidth cost and accumulated delays along the SLinks on $p_i^{e^v}$. This cost is also defined for each mapped path $p_i^{e^v} \in P^{e^v}$ for e^v. In (3.5), the term w^E represents the relative weight of the SLink delay ($\delta(e^s)$) compared to the allocated bandwidth cost (in bandwidth units). In today's data center networks, the link delay is usually very small [21]. For this reason, we suggest that w^E should take a fractional value less than one.

$$\ddot{L}(e^v, p_i^{e^v}, k^{e^v}) = \sum_{e^s \in p^{e^v}} \left(\frac{b(e^s)}{r(e^s)} \frac{b(e^v)}{k^{e^v} - 1} + w^E \delta(e^s) \right) \tag{3.5}$$

A goal in our ILP model is to ensure proper load balancing across SNodes and SLinks. To this end, each SNode and SLink is associated with a non-linear weight function that produces low values for under-utilized links, while weight function value increases rapidly as an SNode's or SLink's utilization approaches saturation. The fractions $\frac{c(n^s)}{r(n^s)}$ and $\frac{b(e^s)}{r(e^s)}$ give higher privilege to less loaded SNodes and SLinks, respectively, over the saturated ones. Therefore, in (3.4) and (3.5), these two fractions are chosen as the *load balancing factors* for SNodes and SLinks, respectively. The possible alternates, *e.g.,* $(1 - \frac{r(n^s)}{c(n^s)})$ and

$(1 - \frac{r(e^s)}{b(e^s)})$, have a linear relation between utilization and demand, and so cannot be used for our purpose.

Now we introduce the SiMPLE objective function. The goal is to minimize the cost presented in (3.6). In this equation, $\ddot{I}$ and $\ddot{S}$ have units in MIPS (for split, join, switching costs involving CPU resources), whereas $\ddot{L}$ has Mbps unit. To unify these different units, we multiply the split, join, and switching costs with a weight, w^N. Furthermore, in comparison with the bandwidth resources, the CPU resources are cheaper and more available. Therefore, we propose that w^N should be a fraction. In this process, we prioritize bandwidth in the cost function above other resources.

`SiMPLE_ILP` :

$$minimize \left[\sum_{e^v \in E^V} \left(\begin{array}{c} \ddot{I}(e^v, P^{e^v}, k^{e^v}) w^N + \\ \sum\limits_{p_i^{e^v} \in P^{e^v}} \left(\begin{array}{c} \ddot{S}(e^v, p_i^{e^v}) w^N + \\ \ddot{L}(e^v, p_i^{e^v}, k^{e^v}) \end{array} \right) \end{array} \right) \right] \tag{3.6}$$

The constraints for `SiMPLE_ILP` are presented in (3.7) - (3.13). Constraint (3.7) assures that the SNode capacities are not violated. Constraint (3.8) states that the SLink capacities are always satisfied while embedding each VLink. For accepted VNs, the VNode demand satisfaction constraint is presented in Constraint (3.9). In our problem, we need that each VNode must be embedded to exactly one SNode, and this contraint is given in Constraint (3.10). Constraint (3.11) denotes that each substrate path required to embed one VLink must be disjoint, which comes from the SiMPLE embedding concept as presented in Section 3.1. Constraint (3.12) ensures that a total of k^{e^v} paths are found while embedding a VLink e^v, whereas Constraint (3.13) ensures that k^{e^v} is an integer between 2 and 5.

$$\forall n^s \in N^S : \sum_{n^v \in N^V} c(n^v) \times X(n^v, n^s) \leq c(n^s) \tag{3.7}$$

$$\forall e^s \in E^S : \sum_{e^v \in E^V} \frac{b(e^v)}{k^{e^v} - 1} \times Y(p^{e^v}, e^s) \leq b(e^s) \tag{3.8}$$

$$\forall e^v \in E^V : Y(P^{e^v}, e^s) \times \frac{b(e^v)}{k^{e^v} - 1} \leq r(e^s) \tag{3.9}$$

$$\forall n^v \in N^V : \sum_{n^s \in N^S} X(n^v, n^s) = 1 \tag{3.10}$$

$$\forall e^v \in E^V : \sum_{p_i^{e^v} \in P^{e^v}} Y(p_i^{e^v}, e^s) \leq 1 \tag{3.11}$$

$$\forall e^v \in E^V : \sum_{p_i^{e^v} \in P^{e^v}} \sum_{e^s \in p_i^{e^v}} \frac{1}{|p|} \times Y(p_i^{e^v}, e^s) = k^{e^v} \tag{3.12}$$

$$\forall e^v \in E^V : (k^{e^v} \in \mathbb{N}) \wedge (2 \leq k^{e^v} \leq 5) \tag{3.13}$$

3.3 Proposed Solutions

The ILP model presented in Section 3.2 can find optimal solution for small instances of the multi-path embedding problem, but it will not scale with SN and VN size. In this section, we propose two scalable greedy algorithms for each part in SiMPLE, as introduced in Section 3.1. The first one is a survivable multi-path proactive embedding approach, and the second one is a recovery mechanism that operates after each SLink failure. Both solutions operate under the assumption that the node mapping has already been done, possibly using one of the greedy approaches (*e.g.,* First Fit or Best Fit approach [26]).

3.3.1 `SiMPLE-PR`

The proactive solution, `SiMPLE-PR`, embeds each VLink of a VN request as it arrives. The algorithm iteratively computes a set of disjoint paths for each VLink, and returns the result of embedding, or ϕ if none exists.

Algorithm 1 SiMPLE Proactive Allocation, `SiMPLE-PR`

1: **function** `SiMPLE-PR`(G^S, G^V, ξ_N)
2: **for all** $e^v \in E^V$ **do**
3: $\forall k \in \{2,3,4,5\} : P^k \leftarrow \phi \wedge Cost(P^k) \leftarrow \infty$
4: **for** $k \in \{2,3,4,5\}$ **do**
5: $\mathbb{E}^S \leftarrow E^S$
6: **for** $j \leftarrow 1, k$ **do**
7: $Q \leftarrow$ `Dijkstra` $(N^S, \mathbb{E}^S, \xi_N(e^v_s), \xi_N(e^v_d), \frac{b(e^v)}{k-1})$
8: $P^k \leftarrow P^k \cup Q$
9: $\mathbb{E}^S \leftarrow \mathbb{E}^S - P^k$
10: **end for**
11: **end for**
12: $P^* \leftarrow \min(P^2, P^3, P^4, P^5)$
13: **if** $Cost(P^*) = \infty$ **then**
14: **return** ϕ
15: **end if**
16: $\xi_E(e^v) \leftarrow P^*$
17: **end for**
18: $\forall e^s \in E^S \cap P^* :$ `update` $r(e^s)$
19: **return** ξ_E
20: **end function**

The input to `SiMPLE-PR`, as presented in Algorithm 1, is an SN G^S, a VN G^V, and its node mapping function, ξ_N. In `SiMPLE-PR`, we split each VLink into no more than five paths. This is because, as illustrated experimentally in Section 4.1.1, a higher number of splits will cause a very high splitting, joining, routing and delay overheads, which will eventually make the embedding expensive and infeasible. `SiMPLE-PR` iteratively works on each VLink of a newly arrived G^V (Lines $2-19$). The set P^k (initially empty) denotes the set of candidate paths selected for split k, where $2 \leq k \leq 5$ and $k \in N$ (Line 3). At each iteration of k (Lines $4-11$), `SiMPLE-PR` runs the Dijkstra's weighted shortest path algorithm to select a candidate path with the sufficient residuals ($b(e^v)$ / $(k-1)$) between the source and destination SNodes of the corresponding VLink (Line 7). This path is added to P^k (Line 8). To maintain the disjointness constraint, the SLinks of the path are temporarily removed from G^S (Line 9). At the end of this loop, the discarded SLinks are restored (Line 5), and the set of paths with the minimal cost, P^*, is calculated (Line 12). If no such set is found (*i.e.,* cost of P^* is ∞), `SiMPLE-PR` finds no feasible mapping for this VLink (and hence G^V) and returns ϕ (Lines $13-15$). Otherwise, it updates the link

mapping function ξ_E(Line 16), and moves on to process the next VLink. If the mapping of all the VLinks are found in this process, SiMPLE-PR returns ξ_E (Line 19). In this case, G^V is embedded onto G^S, and the residual capacities in the corresponding SLinks are updated (Line 18).

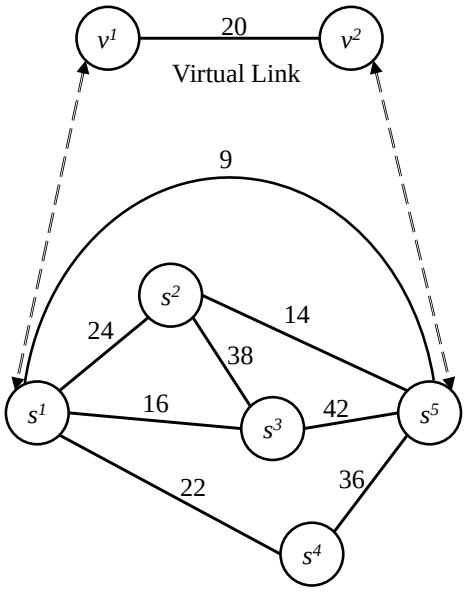
Theorem 3.3.1. *The running time of SiMPLE-PR is* $O(|E^V| \times (|N^S| \cdot log|N^S| + |E^S|))$.

Proof. As mentioned in the previous paragraph, SiMPLE-PR iteratively embeds each VLink of a VN. For each VLink, it runs Dijkstra's algorithm $O(1)$ times. Since, for i splits, Dijkstra's algorithm is invoked i times, where $2 \leq i \leq 5$. In total, Dijkstra's algorithm is called at most $2 + 3 + 4 + 5 = 14$ times. The running time of Dijkstra's algorithm is $O(|N^S| \cdot log|N^S| + |E^S|)$. The later operations, *e.g.,* temporarily discarding the SLinks in the selected path, take $O(|E^S|)$ running time. At the end of the iteration, updating the residual capacities of the SLinks in the optimal path P^* also takes $O(|E^S|)$ running time. Therefore, a single VLink is embedded in a running time $O((|N^S| \cdot log|N^S| + |E^S|))$. In total, SiMPLE-PR embeds a VN in a running time of $O(|E^V| \times (|N^S| \cdot log|N^S| + |E^S|))$. □

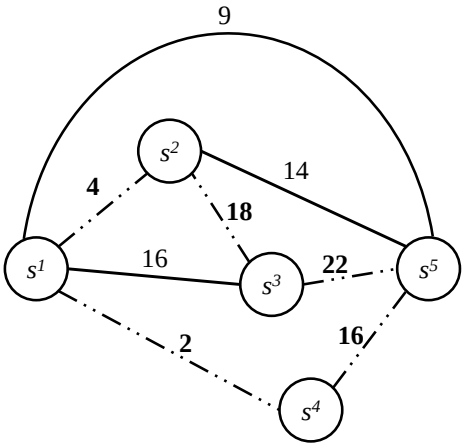
SiMPLE-PR in Action

Here we explain SiMPLE-PR as presented in Algorithm 1 with an example. Fig. 3.3(a) depicts an SN of five SNodes and eight SLinks, and a VN with two VNodes and one VLink. In this example, we consider small size VN for simplicity, and this same procedure is applicable for the larger VNs. The numbers beside each VLink and SLink denote their demand and residual capacities, respectively. The VNodes v^1 and v^2 are mapped to SNodes s^1 and s^5, respectively. Our goal is to embed the VN to the SN. For the sake of simplicity, we assume that the node capacity and demand constraints are satisfied in the SN. Furthermore, we assume that each splitting and joining activity has a cost of 3, and each SNode switch and SLink delay has a cost of 1. We also assume that the node and link weights, w^N and w^E, have value 1.

For two splits case (Fig. 3.3(b)), SiMPLE-PR finds two disjoint paths between s^1 and s^5 with the requested demand, 20. To achieve this goal, it calculates the weighted shortest path between these two SNodes. After the calculation of the weighted shortest path (s^1, s^2, s^3, s^5), its SLinks are discarded from E^S. Then, SiMPLE-PR runs a similar procedure to find the second weighted shortest path, (s^1, s^4, s^5). These selected paths, represented by the path set P^2, is highlighted with dotted lines in Fig. 3.3(b). In this case, the split and join cost at s^1 and s^5 is $3+3+3+3 = 12$. The switching cost is given by $\sum_{i=1}^{7} 1 = 7$, and the bandwidth cost is $\sum_{i=1}^{5}(20 + 1) = 105$. The embedding cost, as returned by the cost function in (3.6), is 124.

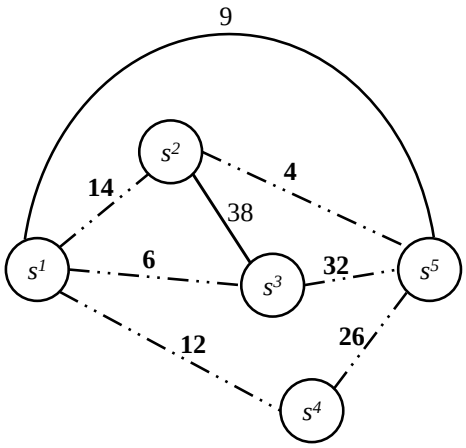


(a) The SN and VN Request



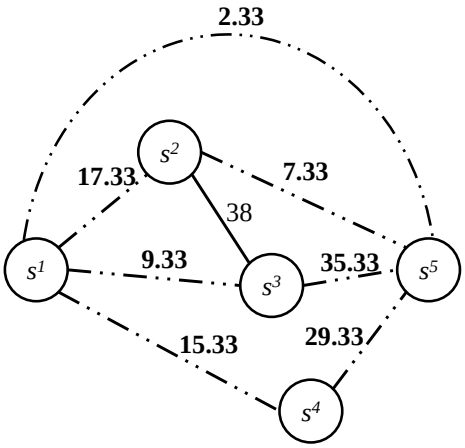
$P^2 = \{(s^1, s^2, s^3, s^5), (s^1, s^4, s^5)\}$
Cost = *6 + 6 + (4 + 3) + 5 * 20 + 5 =* ***124***

(b) Two splits embedding



$P^3 = \{(s^1, s^2, s^5), (s^1, s^3, s^5), (s^1, s^4, s^5)\}$
Cost = *9 + 9 + (3 + 3 + 3) + 7 * 10 + 6 =* ***103***

(c) Three splits embedding



$P^4 = \{(s1, s5), (s^1, s^2, s^5), (s^1, s^3, s^5), (s^1, s^4, s^5)\}$
Cost = 12 + 12 + *(2 + 3 + 3 + 3) + 6.67 * 7 + 7 =* ***88.69***

(d) Four splits embedding

Figure 3.3: `SiMPLE-PR` Embedding Example

For three (Fig. 3.3(c)) and four splits (Fig. 3.3(d)) cases, `SiMPLE-PR` operates in a way similar to the two splits scenario, but finds three and four shortest paths, respectively. The set of paths for three and four splits are given by $P^3 = \{(s^1, s^2, s^5), (s^1, s^3, s^5), (s^1, s^4, s^5)\}$ and $P^4 = \{(s^1, s^5), (s^1, s^2, s^5), (s^1, s^3, s^5), (s^1, s^4, s^5)\}$, respectively. The embedding costs for three and four splits are 103 and 88.69, respectively. Note that there is no set of five disjoint paths between s^1 and s^5. In this case, the embedding cost is infinite. In this example, four splits yields the lowest cost, and it is chosen by `SiMPLE-PR`.

3.3.2 `SiMPLE-RE`

The reactive recovery mechanism, `SiMPLE-RE`, performs on each SLink failure as they arrive. As briefly discussed in Section 3.1.2, `SiMPLE-RE` recovers a failed substrate path in an affected VLink by exploring different recovery strategies. In the end, it returns the recovery embedding with lowest cost, which minimizes the physical resource consumption while considering load balancing.

Algorithm 2 SiMPLE Reactive Recovery, `SiMPLE-RE`

function `SiMPLE-RE`(G^S, e^v, ξ_N)
  $P^{e^v} \leftarrow P^{e^v} - \{p_f^{e^v}\}$
  $P^1 \leftarrow$ `FindNewPath`(G^S, e^v)
  $P^2 \leftarrow \forall p^{e^v} \in (P^{e^v} - p_f^{e^v})$: `alloc` $(p^{e^v}, \frac{b(e^v)}{k^{e^v}-1})$
  $P^3 \leftarrow \forall p^{e^v} \in (P^{e^v} - p_f^{e^v})$: `add` $(p^{e^v}, b(p_f^{e^v}) \cdot \frac{r(p^{e^v})}{\sum r(p^{e^v})})$
  $P^* \leftarrow \min(P^1, P^2, P^3)$
  $\xi_E(e^v) \leftarrow P^*$
  $k^{e^v} \leftarrow |P^*|$
  $\forall e^s \in E^S \cap P^*$: `update` $r(e^s)$
  return ξ_E
end function
function `FindNewPath`(G^S, e^v)
  $\mathbb{E}^S \leftarrow E^S - P_{new}^{e^v}$
  $P^{e^v} \leftarrow P^{e^v} \cup$ Dijkstra $(N^S, \mathbb{E}^S, \xi_N(e_s^v), \xi_N(e_d^v))$
  return P^{e^v}
end function

`SiMPLE-RE` is briefly presented in Algorithm 2. The input to this algorithm are the substrate network, G^S, and the affected virtual link, e^v, the node mapping function, ξ_N.

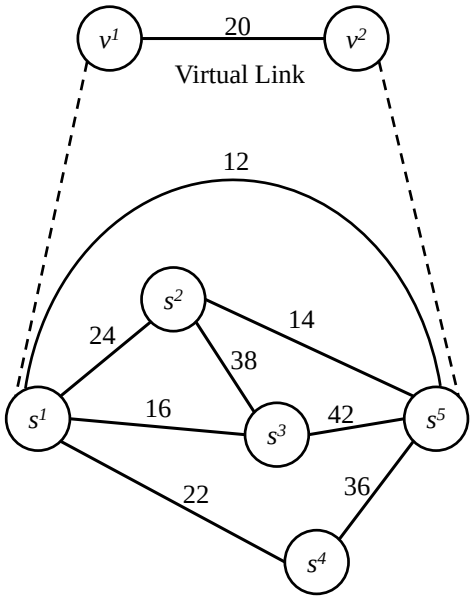
At first, this algorithm discards the failed path, $p_f^{e^v}$, from the existing paths in e^v, P^{e^v} (Line 2). Then, it calls the `FindNewPath` method (Line 3), which finds another link-disjoint path, adds it to P^{e^v} (Lines $11-14$), and returns this set into P^1 (Line 3). Next, `SiMPLE-RE` allocates the fixed amount of bandwidth in the existing paths, and stores the result in P^2 (Line 4). Afterwards, `SiMPLE-RE` recovers the lost bandwidth, $b(p_f^{e^v})$, among the existing paths, P^{e^v}. For this purpose, it splits $b(p_f^{e^v})$ in proportion to residual bandwidth of the paths in P^{e^v}, adds this split bandwidth to these paths, and saves this set into P^3 (Line 5). After that, `SiMPLE-RE` chooses the embedding with the lowest cost, P^*, among P^1, P^2 and P^3 (Line 6). In the next step, the new embedding of e^v is changed to P^* (Line 7), and the number of splits, k^{e^v} is updated accordingly (Line 8). Finally, the residual capacities of the SN is updated according to the changed embedding (Line 9), and the resultant embedding is returned (Line 10).

Theorem 3.3.2. *The running time of `SiMPLE-RE` is* $O(|N^S|log|N^S| + |E^S|)$.

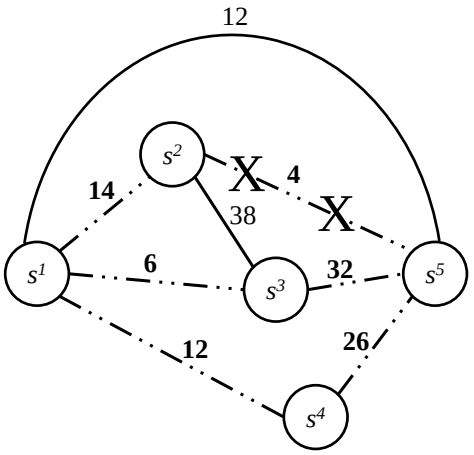
Proof. Note that the steps presented in Lines $3-5$ in `SiMPLE-RE` are independent of each other, and can be executed in parallel. Provisioning a new path, as presented in the `FindNewPath` method (Lines $11-14$) invokes Dijkstra's shortest path algorithm, and this can be run in $O(|N^S|log|N^S| + |E^S|)$ time. The fixed and variable bandwidth allocations, as presented in Lines $3-4$, are linear operations in the number of physical links, $|E^S|$. Line 6 finds the minimum among the three possible alternative embeddings, which is a constant-time operation. Lines $7-8$ are also linear in $|E^S|$, since they involve updating the VLinks and SLinks, respectively. As a result, the running time of `SiMPLE-RE` is determined by the `FindNewPath` method call in Line 3, which is $O(|N^S|log|N^S| + |E^S|)$. □

`SiMPLE-RE` in Action

The working principle of `SiMPLE-RE` is explained with an illustrative example in Fig. 3.4. The notations and different cost values that we use in Fig. 3.4 are the same as the ones introduced in Fig. 3.3 in Section 3.3.1. Fig. 3.4(a) shows a VN request with VNodes embedded to an SN, with the VN demand and SN capacities depicted as levels. Fig. 3.4(b) shows a potential embedding chosen by `SiMPLE-PR`, and it also shows a failed SLink (s^2, s^5). At this stage, `SiMPLE-RE` operates, and recovers the lost path (s^1, s^2, s^5) as follows. First, it finds a new path (s^1, s^5), and calculates the new embedding cost, which is 81. This step is illustrated in Fig. 3.4(c). Second, it considers fixed allocation strategy. However, it fails to allocate the required bandwidth in the existing paths since (s^1, s^4) do not have enough residual bandwidth. In this step, the cost of embedding is ∞. Third, `SiMPLE-RE` considers variable allocation strategy as illustrated in Fig. 3.4(d). The lost bandwidth in

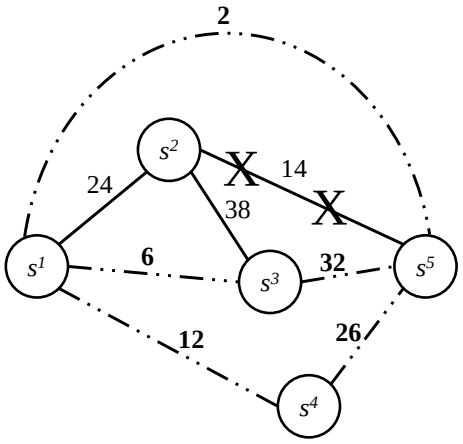


(a) The SN and VN request



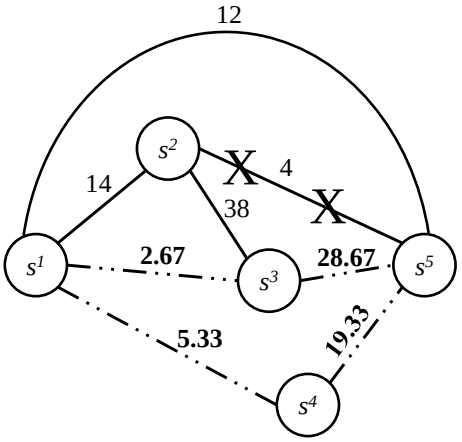
$P = \{(s^1, s^2, s^5), (s^1, s^3, s^5), (s^1, s^4, s^5)\}$

(b) A potential embedding by SiMPLE-PR



$P^1 = \{(s1, s5), (s^1, s^3, s^5), (s^1, s^4, s^5)\}$
Cost = 9 + 9 + *(2 + 3 + 3) + 10 * 5 + 5 = 81*

(c) Provisioning new path



$P^3 = \{(s^1, s^3, s^5), (s^1, s^4, s^5)\}$
Cost = 6 + 6 + *(3 + 3) + 10 * 4 + 16 * 3.33 / 6 + 42 * 3.33 / 32 + 22 * 6.67 / 12 + 36 * 6.67 / 26 + 4 = 96.71*

(d) Variable allocation

Figure 3.4: SiMPLE-RE Recovery Example

(s^1, s^2, s^5) is 10, and it is redistributed in (s^1, s^3, s^5) and (s^1, s^4, s^5) as 3.33 and 6.67 units, respectively. The embedding cost calculated in this process is 96.71. Finally, `SiMPLE-RE` chooses the new path strategy as illustrated in Fig. 3.4(c), since it has the lowest cost.

3.4 Summary

In this chapter, we have explained the main working principle of SiMPLE. SiMPLE expects that the node embedding as input, and performs link embedding. It works in two stages – proactive allocation, and reactive recovery. In the first step, SiMPLE allocates a number of disjoint paths between the source and destination SNodes, and divides the VLink demand across these paths. In the second step, after an SLink failure has occurred, SiMPLE recovers each affected VLink by either provisioning a new path, or redistributing the lost bandwidth in the other previously embedded paths. We have proposed an ILP formulation for SiMPLE, and proposed two heuristic algorithms, `SiMPLE-PR` and `SiMPLE-RE`, for these two steps.

Chapter 4

Evaluation

In this chapter, we evaluate different aspects of SiMPLE through simulations. We begin the discussion with a description of the evaluation environment, then present our main evaluation results. We begin the discussion by describing the simulation setup in Section 4.1. The baseline approaches that we use to evaluate SiMPLE are discussed in Section 4.2. Our terminologies and evaluation metrics are then introduced in Sections 4.3 and 4.4, respectively. The evaluation metrics have been chosen to quantify the performance of SiMPLE in terms of business profit for the InP. The embedding performance evaluation results for `SiMPLE-PR` are presented in Section 4.5. The survivability analysis for `SiMPLE-PR` and `SiMPLE-RE` are then presented in Sections 4.6 and 4.7, respectively. Finally, we summarize this chapter in Section 4.8.

4.1 Simulation Setup

We consider an online version of the SVNE problem, where each VN request is embedded as it arrives. We use Fat tree [6] and synthetically generated topologies as SN to evaluate the behavior of SiMPLE in data center networks and ISP networks, respectively. The Synthetic topology connects each SNode at a low probability (≤ 0.1), which represents an arbitrary ISP network. To demonstrate the SiMPLE scalability, we present the results on VN embedding performance at small scale experiments, and VN survivability at large scale experiments. In small scale experiments, we evaluate both `SiMPLE-PR` and the optimal solution, `SiMPLE-OP`, where the later is an implementation of the ILP model (presented in Section 3.2) using GLPK. This ILP model finds an optimal embedding for all

Table 4.1: Evaluation Environment

Characteristics	Small Scale	Large Scale
Number of SNodes	125 [1] 100 [3]	500 [2] 500 [3]
SNode Capacity	$[50, 150]$	$[10, 50]$
SNode Switching Cost	$[2, 7]$	$[2, 7]$
Number of SLinks	500 [1] 574 [3]	4000 [2] 4029 [3]
SLink Capacity	$[70, 80]$	$[70, 80]$
SLink Delay	$[3, 15]$	$[3, 15]$
Splitting Cost	10 per split	10 per split
Joining Cost	10 per join	10 per join
VNodes per VN	$[2, 6]$	$[2, 10]$
VNode Capacity	$[5, 20]$	$[5, 20]$
VLink Conn. Prob.	0.5	0.5
VLink Demand	$\alpha\%$ of $[70, 80]$	$[10, 20]$
Total Number of VNs	300	300
Total Simulation Time	15000	15000
VN Arrival Rate, λ_V	`Pois`$\{0.05\}$	`Pois`$\{0.05\}$
VN Lifeime	`Geo`$\{1000\}$	`Geo`$\{1000\}$
Failure Arrival Rate, λ_F	N/A	`Pois` $\{0.05 \times \gamma\}$
Failure Repair Time	N/A	`Geo`$\{7000\}$

VLinks of a VN request. To reduce the solution space, the GLPK implementation considers the first 200-shortest loop-less paths between a pair of SNodes, computed using Yen's Algorithm [44]. However, for the large instances, GLPK exceeds memory limits and is unable to find any solution. Also note that we evaluate `SiMPLE-RE` only for the survivability experiments, since its major focus is failure recovery.

For all experiments, VN requests are generated by varying their size randomly. We model VN arrival and SLink failure events as Poisson processes. The lifetime of a VN request is modeled using a Geometric distribution. It is worth noting that, our simulation setup and choice of different simulation parameters are similar to the previous works [11], [39] on the SVNE problem. The simulation parameters are summarized in Ta-

[1] 10-ary Fat tree topology
[2] 20-ary Fat tree topology
[3] Synthetic topology

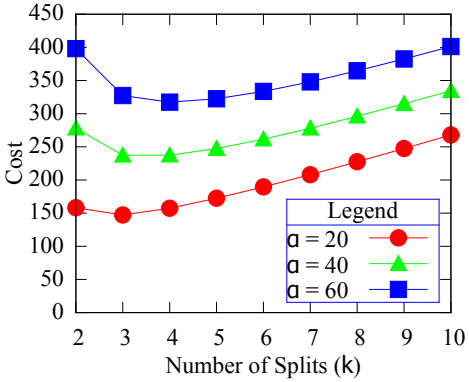


Figure 4.1: Impact of different number of splits for one VLink

ble 4.1. In this table, $[x_{min}, x_{max}]$ denotes a uniform distribution between x_{min} and x_{max}. $Pois\{p\}$ and $Geo\{g\}$ stand for the Poisson and Geometric distributions with mean p and g, respectively. For our experiments, we use random node mapping, which is less informed and thus makes the VLink embedding more challenging than the systematic node mapping approaches (*e.g.,* First Fit [26]).

We run our experiments under different levels of workload, α, defined as the percentage of the average VLink demand to the average SLink capacity. To observe the impact of different workloads, α is varied from 10% to 60%. Furthermore, since our focus is to mitigate SLink failures, we measure SiMPLE's ability to survive different failure levels, expressed as γ – the ratio of the failure rate to the VN arrival rate. In large scale experiments, we stress the SN with a lot of failures, even at a rate higher than the VN arrival rate. For this reason, γ is varied from 1 to 6. In addition, the mean time to repair (MTTR) is significantly higher than the mean VN lifetime (Table 4.1) to magnify the impact of failures.

4.1.1 Determining Maximum Number of Splits

In this experiment, we measure the impact of the number of splits (k) on VN embedding cost (given in (3.6)). For this experiment, we have varied the number of splits from 2 to 10 on a dense SN topology, which facilitates having at least 10 paths between its end hosts. Furthermore, we have varied the load on VN demands to see the impact of the number of splits. Since we are interested to see the impacts of the number of splits on each VLink, in this experiment, the VN requests had only two VNodes and one VLink. The experiment was run on a 12-ary Fat tree [6] topology for α = 20, 40, and 60. The results from this

experiment are shown in Fig. 4.1.

Fig. 4.1 shows that, for all α, the embedding cost increases with an increasing number of splits. When the number of paths increases beyond 5, this behavior becomes more prominent. For this reason, we restricted the value of k between 2 and 5 for the experiments in Section 4.

4.2 Baseline Approaches

We compare SiMPLE to two proactive approaches, *Full Backup Scheme (FBS)* and *Shared Backup Scheme (SBS)*.

4.2.1 Full Backup Scheme

In FBS, the full demand of each VLink is mapped to two disjoint substrate paths, which are computed using Dijkstra's weighted shortest path algorithm. The shorter of these two paths act as primary, and the other path is reserved as backup.

4.2.2 Shared Backup Scheme

Among the alternatives for proactive shared backup, we choose [22] as it generalizes the main concept of SBS. The primary and backup path allocation in SBS is similar to that in FBS. In contrast to FBS, multiple VLinks can share the same resources for their backup paths. When a failure occurs, the affected VLinks try to recover their full demand from the backup path. In case of multiple VLinks trying to claim the bandwidth from the same backup link simultaneously, the fair sharing policy is adopted.

4.3 Terminology

We use the following terms to analyze failure impacts.

4.3.1 Path Failure

A *path failure* event is defined as the failure of one (or, more) SLink(s) belonging to a spacific path. At this state, the corresponding path cannot carry the flow from the source to the destination SNode.

4.3.2 Affected VLink

A VLink is affected by a SLink failure *if and only if* one (or, more) of its substrate paths fail(s). An affected VLink may still retain its full demand depending on the severity of failure. For example, both FBS and SiMPLE retain their full demand in presence of a single SLink failure.

4.3.3 Failed VLink and Failed VN

A VLink is failed *if and only if* all of its mapped substrate paths fail (*i.e.,* when it meets 0% of its demand). A VN fails *if and only if* one (or, more) of its VLinks fail(s). For example, a VLink failure in Full Backup Scheme implies two different substrate paths failures, whereas, in SiMPLE, it implies a maximum of five different substrate paths failures.

4.4 Performance Metrics

Unless otherwise specified, the symbols used in this Section have their usual meanings as described in Section 3.2. The performance metrics are discussed as follows, and summarized in Table 4.2.

4.4.1 Profit, Ψ

We first define the revenue, $\Pi(G^V)$, for a VN as $\Pi(G^V) = c_1 \sum_{e^v \in E^V} b(e^v) + c_2 \sum_{n^v \in N^V} c(n^v)$. Here, c_1 and c_2 are application-specific constants that represent the relative importance of bandwidth and CPU. The profit of G^V is defined by $\Psi(G^V) = T(G^V) \times \big(\Pi(G^V) - Cost(G^V)\big)$. Here, $T(G^V)$ is the lifetime of G^V, and $Cost(G^V)$ represents the total substrate cost for G^V, as represented in (3.6). The overall profit is given by, $\Psi = \sum_{\forall G^V} \Psi(G^V)$.

Table 4.2: Evaluation Metrics

α	Workload
λ_V	VN arrival rate
λ_F	Failure arrival rate
γ	Failure level
$\Pi(G^V)$	Revenue earned from G^V
$\Psi(G^V)$	Profit earned from G^V
Ψ	Total profit for all VNs
$\hat{\mathbb{B}}$	Average fraction of backup substrate resources
$\mathbb{AR}$	The acceptance ratio
$\hat{\mathbb{F}}$	Average fraction of survived bandwidth
$\hat{\mathbb{S}}$	Average splitting overhead
$Prob(\rho^i)$	Probability of i simultaneous VN failures

4.4.2 Acceptance Ratio, $\mathbb{AR}$

It is the ratio of the number of accepted VNs in the system ($|\mathbb{Z}^{\mathbb{A}}|$) to the total number of arrived VN requests ($|\mathbb{Z}^{\mathbb{T}}|$). Formally, $\mathbb{AR} = |\mathbb{Z}^{\mathbb{A}}|/|\mathbb{Z}^{\mathbb{T}}|$, where $\mathbb{Z}^{\mathbb{A}} \subseteq \mathbb{Z}^{\mathbb{T}}$.

4.4.3 Average Fraction of Backup Bandwidth, $\hat{\mathbb{B}}$

For $e^v \in E^V$, $\mathbb{B}^{e^v}$ is the ratio of its backup bandwidth allocation to its total bandwidth allocation, *i.e.,* $\mathbb{B}^{e^v} = |p_b^{e^v}|/\sum_{p_i^{e^v} \in P^{e^v}} |p_i^{e^v}|$. Here, $|p_b^{e^v}|$ is the bandwidth consumption for backup path $p_b^{e^v}$. The average fraction of backup bandwidth is, $\hat{\mathbb{B}} = \underset{\forall e^v \in E^V}{Avg} \left(\mathbb{B}^{e^v}\right)$.

4.4.4 Average Splitting Overhead, $\hat{\mathbb{S}}$

The average splitting overhead is given by the average of the total split, join, and switch cost for all VLinks, *i.e.,* $\hat{\mathbb{S}} = \underset{e^v \in E^V}{Avg} \left(\ddot{I}(e^v, P^{e^v}, k^{e^v}) + \sum_{p_i^{e^v} \in P^{e^v}} \ddot{S}(e^v, p_i^{e^v})\right)$

4.4.5 Average Fraction of Survived Bandwidth, $\hat{\mathbb{F}}$

Let $\tilde{E}^V \subseteq E^V$ denote the set of affected VLinks. For an *affected VLink* $\tilde{e}^v \in \tilde{E}^V$, $\mathbb{F}^{\tilde{e}^v}$ represents the ratio of the available bandwidth to its total demand. The average fraction of survived bandwidth ($\hat{\mathbb{F}}$) of the affected VLinks is given by, $\hat{\mathbb{F}} = \underset{\forall \tilde{e}^v \in \tilde{E}^V}{Avg} \left(\mathbb{F}^{\tilde{e}^v}\right)$.

4.4.6 Probability of Simultaneous VN Failures, $Prob(\rho^i)$

Let ρ^i denote the event of i simultaneous VN failures, and τ_i be the duration of time for ρ_i. $Prob(\rho^i)$ is denoted as the ratio of its lifetime τ_i to total simulation time τ, *i.e.,* $Prob(\rho^i) = \tau_i/\tau$.

4.4.7 Nine Availability

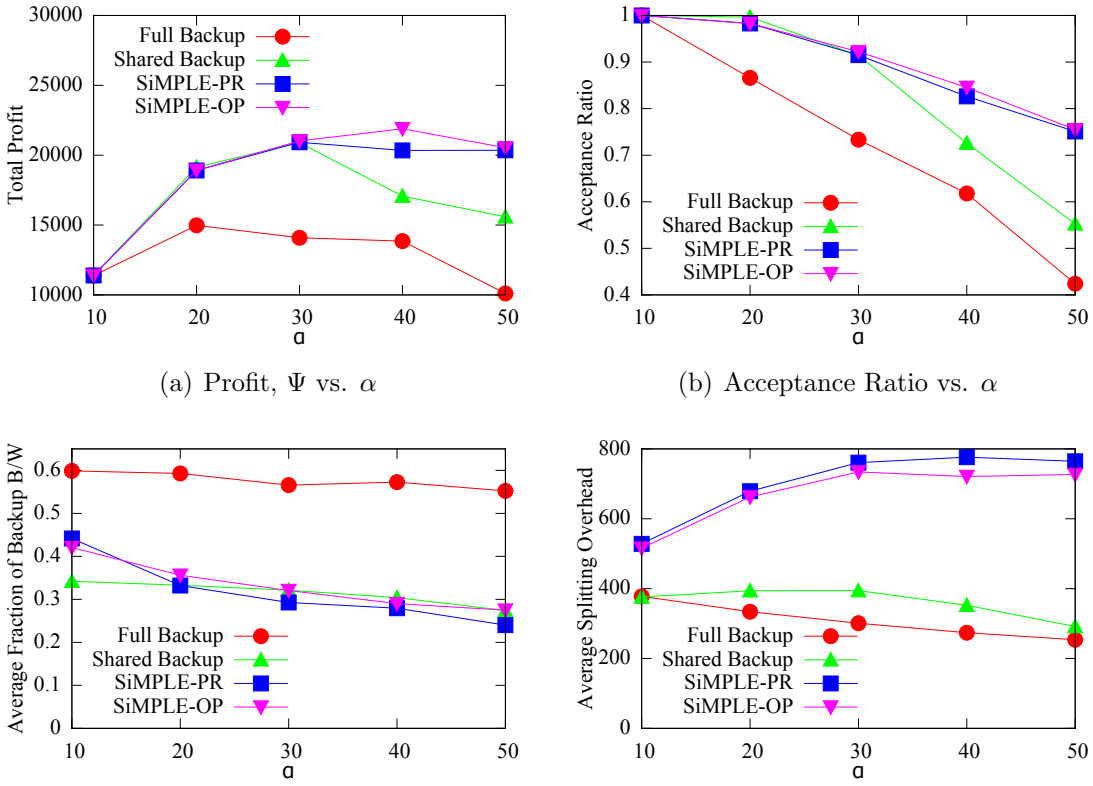
The availability of a system is often represented by the number of nines in its uptime probability; *e.g.,* 1 or 2 nines imply that the probability of the system being available is 0.9 or 0.99, respectively [17]. We compute the nine availability of a failed VN, G^V, as $\left(-\log_{10}\omega(G^V)\right)$, where $\omega(G^V)$ is the ratio of time G^V is in failed state to its lifetime.

4.5 Performance Evaluation Results

We evaluate the VN embedding performances in the following four schemes for Fat tree and Synthetic topologies.

4.5.1 Profit

In terms of Profit, `SiMPLE-PR` outperforms both FBS and SBS approaches, and is very close to the optimal result (`SiMPLE-OP`). Fig. 4.2(a) and Fig. 4.3(a) show the profits for different load (or, α) in the Fat tree and Synthetic topologies, respectively. As shown in these two figures, all approaches achieve similar profits for small load ($\alpha \leq 20$). However, at increased loads, the profits decrease for FBS and SBS. `SiMPLE-PR` achieves approximately $100 - 300\%$ and $50 - 120\%$ more profit than FBS and SBS, respectively.



(a) Profit, Ψ vs. α

(b) Acceptance Ratio vs. α

(c) Backup B/W, $\hat{\mathbb{B}}$ vs. α

(d) Avg. Split Overhead vs. α

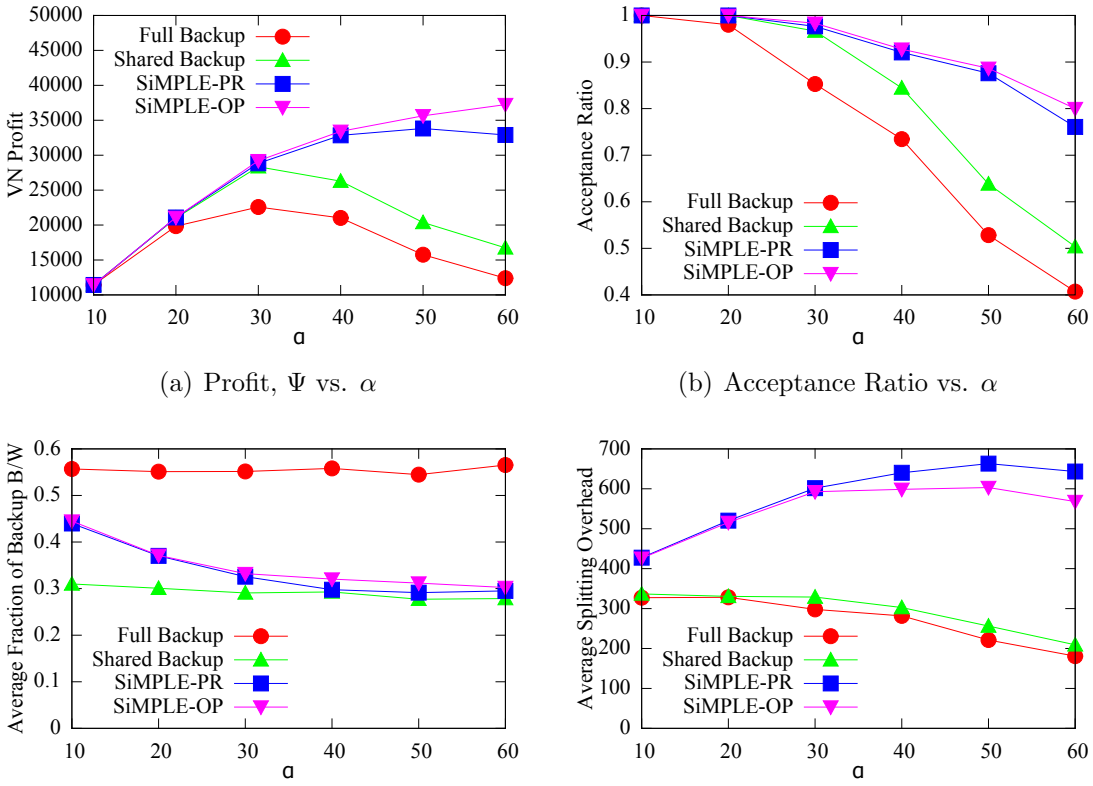
Figure 4.2: Performance Analysis for Fat tree topology

4.5.2 Acceptance Ratio

Results for the $\mathbb{AR}$ at different α are given in Fig. 4.2(b) and Fig. 4.3(b), respectively. According to these results, `SiMPLE-PR` performs as good as FBS and SBS for small loads ($\alpha \leq 20$). However, at larger loads, $\mathbb{AR}$ of `SiMPLE-PR` exceeds the baseline approaches by roughly $20 - 100\%$, and lies very close to `SiMPLE-OP`.

4.5.3 Overhead

The overhead of the considered approaches are evaluated from two perspectives – backup bandwidth allocation and splitting overhead. `SiMPLE-PR` uses a very small fraction of the total allocated bandwidth resource as backup. Fig. 4.2(c) and Fig. 4.3(c) show $\hat{\mathbb{B}}$ for



(a) Profit, Ψ vs. α

(b) Acceptance Ratio vs. α

(c) Backup B/W, $\hat{\mathbb{B}}$ vs. α

(d) Avg. Split Overhead vs. α

Figure 4.3: Performance Analysis for Synthetic topology

different α for Fat tree and Synthetic topologies, respectively. These two figures show that FBS uses more than half of its resources for backup, regardless of α and substrate topology. On the contrary, $\hat{\mathbb{B}}$ is relatively smaller for both `SiMPLE-PR` and `SiMPLE-OP`. The value $\hat{\mathbb{B}}$ for SBS is always small for all α, since SBS allows sharing the same backup resource between multiple VLinks, and thus does not guarantee survivability unlike SiMPLE. However, for heavier loads, SiMPLE uses approximately $40 - 50\%$ less backup bandwidth than FBS, and performs very close to SBS. The splitting overhead, $\hat{\mathbb{S}}$, of these approaches are shown in Fig. 4.2(d) and Fig. 4.3(d). According to these results, $\hat{\mathbb{S}}$ in `SiMPLE-PR` or `SiMPLE-OP` is roughly two to three times higher than that in FBS or SBS. But this increase in splitting overhead comes with the benefits of survivability guarantee and reduced backup overhead. Moreover, with the built-in path splitting capability, modern switches are expected to mitigate this impact.

4.5.4 Execution Time

The average execution time for embedding a VN request in `SiMPLE-OP` and `SiMPLE-PR` are presented in Table 4.3, which shows that `SiMPLE-PR` is $50-60$ times faster than `SiMPLE-OP`. A significant portion of the execution time of `SiMPLE-OP` is consumed by GLPK in finding an optimal solution. However, the performance of `SiMPLE-PR` lies very close to `SiMPLE-OP` in all cases, as evident in Fig. 4.2 and Fig. 4.3. For large scale instances with 500 SNodes, GLPK exceeds memory limits, hence cannot find a solution.

Table 4.3: Average Execution Time (sec)

Topology	`SiMPLE-OP`	`SiMPLE-PR`
Fat tree	61.72	0.95
Synthetic	51.04	0.96

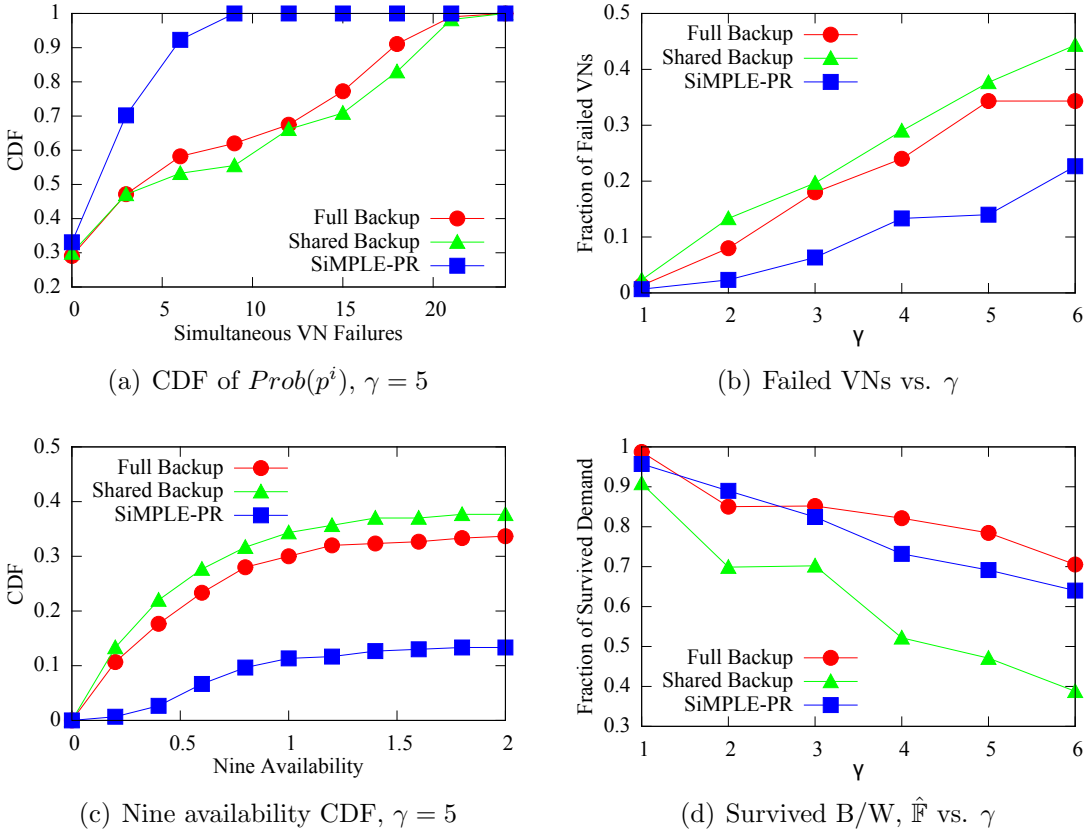
4.5.5 Discussion

Profit vs. $\mathbb{AR}$

From Fig. 4.2(a) and Fig. 4.3(a), we observe that both FBS and SBS suffer from decreasing profit with increasing α. This reduction in profit is due to the lower $\mathbb{AR}$, as presented in Fig. 4.2(b) and Fig. 4.3(b). To embed the VLinks, SiMPLE relies on *path splitting*. Since SiMPLE spreads the VLink demand across multiple paths, it utilizes the substrate resources more efficiently, and achieves a higher $\mathbb{AR}$. On the contrary, FBS and SBS do not rely on path splitting, and fail to achieve satisfactory $\mathbb{AR}$ due to resource fragmentation. SBS utilizes resources more efficiently than FBS because of backup resource sharing, and achieves slightly better performance.

Profit vs. Overhead

In addition to providing a higher profit as shown in Fig. 4.2(a) and Fig. 4.3(a), SiMPLE requires a lower fraction of backup bandwidth. This behavior is depicted in Fig. 4.2(c) and Fig. 4.3(c). SBS has the lowest backup bandwidth over all workloads α, which is mostly due to the backup resources fair sharing policy. On the contrary, because it is often not cost effective to split smaller demands, SiMPLE has a slightly higher backup bandwidth requirement than SBS at lower α. However, with increasing α, the number of splits at



(a) CDF of $Prob(p^i)$, $\gamma = 5$

(b) Failed VNs vs. γ

(c) Nine availability CDF, $\gamma = 5$

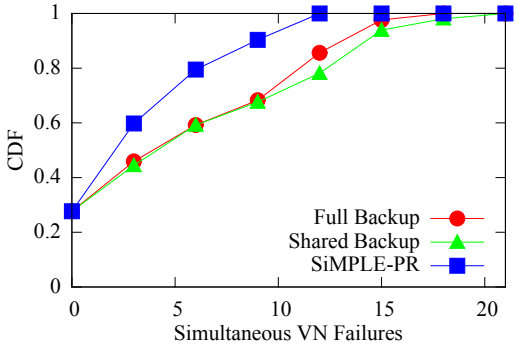
(d) Survived B/W, $\hat{\mathbb{F}}$ vs. γ

Figure 4.4: Survivability Analysis for Fat tree topology

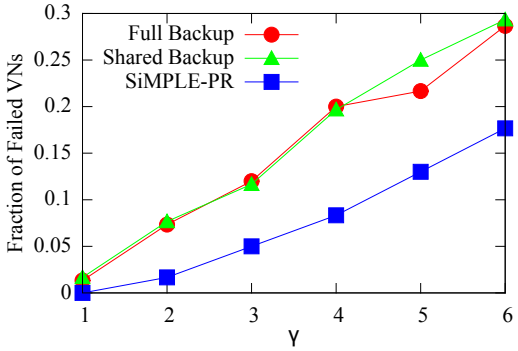
each VLink increases. Therefore, in SiMPLE, $\hat{\mathbb{B}}$ decreases, and becomes similar to SBS. At the same time, path splitting allows SiMPLE to achieve a higher profit than FBS and SBS. However, path splitting brings additional overhead (presented in Fig. 4.2(d) and Fig. 4.3(d)) to SiMPLE. Nonetheless, this overhead is compensated by larger profit, better acceptance ratio, and lower backup bandwidth requirement.

4.6 Survivability Evaluation Results for `SiMPLE-PR`

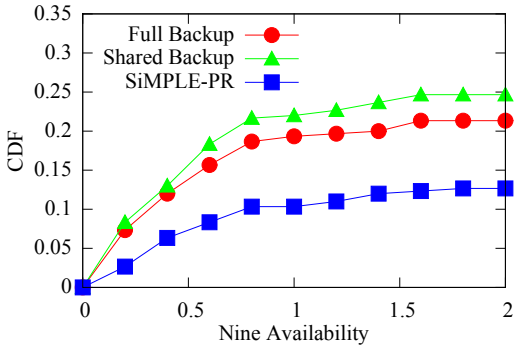
We conducted experiments to evaluate survivability of `SiMPLE-PR`, FBS, and SBS in the event of failures for Fat tree and Synthetic topologies.



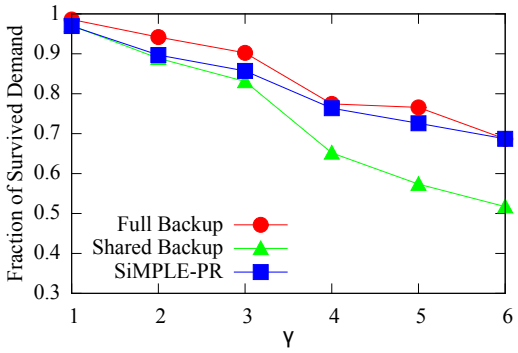
(a) CDF of $Prob(p^i)$, $\gamma = 5$



(b) Failed VNs vs. γ



(c) Nine availability CDF, $\gamma = 5$



(d) Survived B/W, $\hat{\mathbb{F}}$ vs. γ

Figure 4.5: Survivability Analysis for Synthetic topology

4.6.1 Impact of Failures

The impact of failures is evaluated from two perspectives. First, we present the Cumulative Distribution Function (CDF) for $Prob(\rho^i)$ – the probability of i simultaneous VN failures, for $i = 0, 1, 2, \ldots, i_{max}$, where i_{max} denotes the maximum number of simultaneous VN failures. For $\gamma = 5$, the CDF for Fat tree and Synthetic topologies are shown in Fig. 4.4(a) and Fig. 4.5(a), respectively. Second, we measure the fraction of failed VNs to total accepted VNs in SN. For different γ, these results are shown in Fig. 4.4(b) and Fig. 4.5(b) for Fat tree and Synthetic topologies, respectively. These figures show that, both simultaneous and total VN failures are less likely to occur in `SiMPLE-PR`. In contrast, these quantities are higher in FBS, and the highest in SBS. For larger γ, the number of failed VNs is approximately $50 - 100\%$ higher in FBS and SBS than that in `SiMPLE-PR`. These

results reveal that SiMPLE-PR provides the best resilience to failures, whereas SBS performs the worst among these schemes.

4.6.2 Availability

The CDF of nine availability of the failed VNs for $\gamma = 5$ are depicted in Fig. 4.4(c) and Fig. 4.5(c), for Fat tree and Synthetic topologies, respectively. We see that a small fraction of VNs have low nine availability in SiMPLE-PR. In contrast, this fraction is much higher in case of FBS and SBS. Therefore, compared to these two schemes, SiMPLE-PR provides high availability to a higher number of embedded VNs. For example, the number of VNs with only 68% or less availability (0.5 nines) in FBS and SBS are roughly twice (Synthetic topology) or four times (Fat tree topology) than that in SiMPLE-PR.

4.6.3 Failure Tolerance

To evaluate the failure tolerance of each of the considered approaches, we measure the average fraction of survived bandwidth for affected VLinks, $\hat{\mathbb{F}}$. Fig. 4.4(d) and Fig. 4.5(d) present the changes in $\hat{\mathbb{F}}$ for different values of γ for Fat tree and Synthetic topologies, respectively. In these figures, we see that the $\hat{\mathbb{F}}$ obtained in SiMPLE-PR is within $5 - 10\%$ of that in FBS for all values of γ. However, $\hat{\mathbb{F}}$ provided by SBS is lower than the other two schemes. For larger values of γ, $\hat{\mathbb{F}}$ obtained in SBS is approximately $50 - 70\%$ less than that in SiMPLE-PR, which demonstrates a poor performance of SBS in presence of frequent failures.

4.6.4 Discussion

Impact of Failures vs. Availability

We see that SiMPLE outperforms FBS and SBS in both minimizing failure impact (Fig. 4.4(a), Fig. 4.4(b), Fig. 4.5(a), Fig. 4.5(b)) and achieving better availability (Fig. 4.4(c), Fig. 4.5(c)). The superiority of SiMPLE is achieved due to embedding VLinks over multiple disjoint paths. Since SiMPLE associates more SLinks to each VLink e^v, the minimum number of SLink failures required for e^v to fail also increases. In contrast, the number of associated SLinks to e^v in FBS and SBS are lower, because they do not embed VLinks into multiple paths. Hence, SLink failures are more likely to cause VLink (or, VN) failures in these two approaches. For SBS, the SLinks in the backup path of a VLink e_1^v may

already be used by another VLink e_2^v suffering from SLink failure, which makes e_1^v more vulnerable. For these reasons, SBS suffers from failures more than FBS, while SiMPLE outperforms both of these approaches. Again, the number of associated SLinks to each VLink e^v is higher in SiMPLE than that in FBS and SBS. Hence, repairing an SLink is more likely to restore one of the failed paths associated to e^v in SiMPLE. This will make e^v operational by salvaging a fraction of its demand through the restored path. In contrast, the probability of restoring one of the failed paths in FBS or SBS is lower than SiMPLE. Therefore, SiMPLE achieves higher availability than these two strategies.

Impact of Failures vs. Fault Tolerance

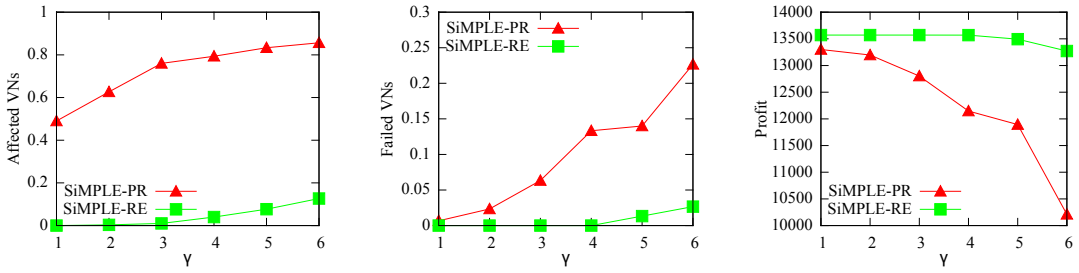
The correlation between low impact of failures (Fig. 4.4(a), Fig. 4.4(b), Fig. 4.5(a), Fig. 4.5(b)) and high fault tolerance (Fig. 4.4(d), Fig. 4.5(d)) in SiMPLE is also part of its main concept, *i.e.,* path splitting with minimal backup. In previous paragraph, we have seen how path splitting increases survivability by associating multiple SLinks to each VLink. In addition, we notice that the number of operational SLinks in an affected VLink e^v is also high. These fully operational SLinks facilitate e^v to retain its full demand (for a single SLink failure), or a high fraction of it (for multiple SLink failures). In SiMPLE, $\hat{\mathbb{F}}$ is very close to that of FBS. However, FBS needs dedicated backup path with full demand unlike SiMPLE. The backup path sharing in SBS makes it vulnerable to multiple and frequent failures. Therefore, for the affected VLinks, SiMPLE performs identically to FBS, and outperforms SBS.

4.7 Survivability Evaluation Results for `SiMPLE-RE`

In this section, we present the simulation results on survivability experiments for `SiMPLE-RE`. For Fat tree and Synthetic topologies, these results are shown in Fig. 4.6 and Fig. 4.7, respectively.

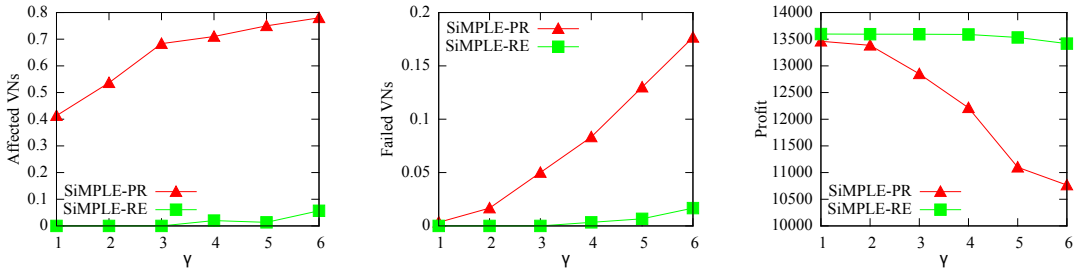
4.7.1 Impact of Failures

The impact of failures is investigated from two perspectives. First, we measure the fraction of affected VNs to the total number of embedded VNs in the SN, for different values of γ. These fractions for `SiMPLE-PR` and `SiMPLE-RE` are illustrated in Fig. 4.6(a) (for Fat tree topology) and in Fig. 4.7(a) (for Synthetic topology). These graphs demonstrate that



(a) Affected VNs vs. γ (b) Failed VNs vs. γ (c) Increase in Profit vs. γ

Figure 4.6: Recovery Analysis for Fat tree topology



(a) Affected VNs vs. γ (b) Failed VNs vs. γ (c) Increase in Profit vs. γ

Figure 4.7: Recovery Analysis for Synthetic topology

when `SiMPLE-RE` is adopted, the total number of affected VNs are always decreased. For smaller values of γ, `SiMPLE-RE` successfully recovers all affected VNs in the substrate. The decrease in the affected VNs is less when γ is large. However, `SiMPLE-RE` still decreases the number of affected VNs by a factor of 85% (for Fat tree topology) or 90% (for Synthetic topology).

Second, we measure the fraction of failed VNs to the total number of embedded VNs in SN. For Fat tree and synthetic topologies, these numbers are shown in Fig. 4.6(b) and in Fig. 4.7(b), respectively. These figures state that, for smaller values of γ, the number of failed VNs are decreased completely, for both Fat tree and Synthetic topologies. For larger values of γ, this decreasing factor is still close to 90%.

4.7.2 Profit

The profit for both SiMPLE-PR and SiMPLE-RE are shown for Fat tree and Synthetic topologies, in Fig. 4.6(c) and Fig. 4.7(c), respectively. From these two figures, we see that SiMPLE-RE provides a higher profit than SiMPLE-PR, for all values of γ. Especially, for higher values of γ, this increase in profit can reach as much as 35%.

4.7.3 Discussion

In comparison with SiMPLE-PR, SiMPLE-RE reduces both the number of affected and failed VNs, as well as a corresponding increase in profit. For small values of γ, the impact of failures is relatively low, and SiMPLE-RE recovers each failure successfully. For this reason, it decreases the number of affected VNs and failed VNs by 100%. For higher values of γ, the impact of failures are also higher, and, as discussed in Section 3.1.2, there can be cases when SiMPLE-RE fails to recover. For this reason, this affected or failed VN decreasing factor is not 100% for higher values of γ. As seen in Fig. 4.6(a), Fig. 4.6(b), Fig. 4.7(a) and Fig. 4.7(b), the total number of affected VNs are always higher than the total number of failed VNs. This behavior can be explained from the definitions in Section 4.3, *i.e.,* it requires all the paths used in a VLink embedding to fail to create an instance of a failed VN. In contrast, it requires only one path used in a VLink embedding to fail to create an instance of an affected VN. Therefore, by intuition, the total number of affected VNs will always be higher than the total number of failed VNs.

Since SiMPLE-RE recovers the affected and failed VNs reactively, it also increases the profit Ψ. This behavior is more prominent for large values of γ. For a small γ, the number of affected or failed VNs are small in SiMPLE-PR, and their recoveries do not increase Ψ more than 15%. However, for a large γ, the number of affected or failed VNs are high, resulting to a higher profit for SiMPLE-RE. In these cases, even though SiMPLE-RE cannot recover all the affected or failed VNs, it recovers most of them. As a result, the increase in Ψ is also larger.

4.8 Summary

In this chapter, we have presented our evaluation results. We have shown that SiMPLE can be beneficial both in the context of data center and ISP networks. It outperforms both FBS and SBS in context of VN embedding performance and VN survivability metrics.

While SiMPLE guarantees to survive single SLink failure scenario, we demonstrated that SiMPLE achieves very high survivability against multiple, frequent and arbitrary SLink failures.

Chapter 5

Conclusion

Network Virtualization facilitates multiple heterogeneous virtual networks by allowing their coexistence in a shared substrate network while providing performance isolation and guaranteeing QoS. Since both data center and ISP networks are very likely to suffer from link failures, they can jeopardize the soundness of the VNs deployed on them. To ensure VN survivability, one promising approach is to allocate redundant resources (*e.g.,* bandwidth) while embedding the VNs, which may create additional overheads. This leads to the trade-off between provisioning maximal VN survivability while minimizing both the redundancy level and the overheads. However, improving VN survivability can decrease the SLA violation and increase the profit earned in context of the InP.

In this thesis, we propose SiMPLE, which achieves VN **S**urvivability **i**n **M**ulti-**P**ath **L**ink **E**mbedding. In Section 5.1, we summarize the contributions of this thesis. In Section 5.2, we provide a list of interesting research directions that can be pursued based on these contributions.

5.1 Summary of Contribution

In this thesis, we have presented SiMPLE which exploits the substrate network's path splitting capability for survivable embedding of virtual network requests. SiMPLE's design goal is to reconcile the conflicting objectives of achieving maximal survivability and minimizing both redundancy and overhead. Compared to existing approaches, SiMPLE reserves less backup bandwidth, yet guarantees virtual link survivability in presence of a single substrate link failure. In case of multiple link failures, the survived bandwidth of the

affected virtual link(s) is better than that of FBS and SBS. Simulation results demonstrated that SiMPLE reduces the failure percentage by at least 50% over those two schemes, and provides better availability of VNs. In addition, backup bandwidth overhead in SiMPLE is $40-50\%$ less than that of FBS, and lies very close to SBS. The SiMPLE reactive recovery approach improves profit generated by the SiMPLE proactive allocation by approximately 40%, as well as decreases the number of failed VNs by approximately 90%. Finally, the path splitting overhead incurred by SiMPLE is compensated by guaranteed survivability, increased profit, better acceptance ratio and lower backup bandwidth requirement.

5.2 Future Works

- **Prototype Implementation.** As a future extension of this work, we intend to evaluate the performance of SiMPLE through a prototype implementation. For this purpose, we plan to use Software Defined Network (SDN) [29], which proposes a logically centralized controller to control the switches in data plane remotely. As introduced in Section 2.4.2, Multipath TCP can be an enabling technology to support path splitting in the substrate. The whole prototype can be deployed on a designated SDN testbed, like Distributed OpenFlow Testbed (DOT) [2], [40], or Mininet [23].

- **Coordinated Node and Link Mapping.** Currently, SiMPLE assumes that a node embedding is given to it as input, and performs link embedding accordingly. However, instead of providing the node embedding result, the service providers would tend to provide a preferred location for each virtual node. In this case, the challenge is to find both node and link embedding in a coordinated manner. M. Chowdhury [15] provides some insights to this problem. We also would like to extend SiMPLE towards a coordinated node and link mapping strategy.

- **Nested Network Virtualization.** Nested NV means the scenario when a VN is deployed on top of another VN [28]. In this case, multiple levels of VNs are formed, where the VNs on level i act as the SN for the VNs on level $i+1$. It would be interesting to extend the idea of SiMPLE to the context of nested NV, with at least two levels of VNs. Survivability in such a multi-layer NV environment could raise further challenges since it involves cross layer optimization.

References

[1] Available at: http://www.informationweek.com/it-downtime-costs-$265-billion-in-lost-revenue/d/d-id/1097919.

[2] Dot: A distributed openflow testbed. dothub.org.

[3] Multipath tcp - linux kernel implementation. multipath-tcp.org.

[4] Sharad Agarwal, John Dunagan, Navendu Jain, Stefan Saroiu, Alec Wolman, and Harbinder Bhogan. Volley: Automated data placement for geo-distributed cloud services. In *NSDI*, pages 17–32, 2010.

[5] Chang Wook Ahn and R. S. Ramakrishna. A genetic algorithm for shortest path routing problem and the sizing of populations. *IEEE Transactions on Evolutionary Computation*, 6(6):566–579, 2002.

[6] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. 38:63–74, Oct 2008.

[7] Nikhil Bansal, Ranjita Bhagwan, Navendu Jain, Yoonho Park, Deepak Turaga, and Chitra Venkatramani. Towards optimal resource allocation in partial fault-tolerant applications. In *The 27th Conference on Computer Communications IEEE INFOCOM*, 2008.

[8] Sushil Bhardwaj, Leena Jain, and Sandeep Jain. Cloud computing: A study of infrastructure as a service (iaas). *International Journal of engineering and information Technology*, 2(1):60–63, 2010.

[9] Peter Bodík, Ishai Menache, Mosharaf Chowdhury, Pradeepkumar Mani, David A. Maltz, and Ion Stoica. Surviving failures in bandwidth-constrained datacenters. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 431–442, 2012.

[10] Qingyun Chen, Ying Wan, Xuesong Qiu, Wenjing Li, and Ailing Xiao. A survivable virtual network embedding scheme based on load balancing and reconfiguration. In *IEEE NOMS*, pages 1–7, Poland, May 2014.

[11] Yang Chen, Jianxin Li, Tianyu Wo, Chunming Hu, and Wantao Liu. Resilient virtual network service provision in network virtualization environments. In *IEEE ICPADS*, pages 51–58, Shanghai, China, Dec 2010.

[12] Xiang Cheng, Sen Su, Zhongbao Zhang, Hanchi Wang, Fangchun Yang, Yan Luo, and Jie Wang. Virtual network embedding through topology-aware node ranking. *ACM SIGCOMM Comput. Commun. Rev.*, 41(2):38–47, Apr 2011.

[13] N. M. Mosharaf Kabir Chowdhury and Raouf Boutaba. A survey of network virtualization. *Computer Networks*, 54:862–876, Apr 2010.

[14] N. M. Mosharaf Kabir Chowdhury, Muntasir Raihan Rahman, and Raouf Boutaba. Virtual network embedding with coordinated node and link mapping. In *INFOCOM 2009, IEEE*, pages 783–791. IEEE, 2009.

[15] N. M. Mosharaf Kabir Chowdhury, Muntasir Raihan Rahman, and Raouf Boutaba. Virtual network embedding with coordinated node and link mapping. In *INFOCOM 2009, IEEE*, pages 783–791. IEEE, 2009.

[16] N. M. Mosharaf Kabir Chowdhury, Fady Samuel, and Raouf Boutaba. Polyvine: policy-based virtual network embedding across multiple domains. In *Proceedings of the second ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures*, pages 49–56. ACM, 2010.

[17] John R. Douceur. Is remote host availability governed by a universal law? *ACM SIGMETRICS Performance Evaluation Review*, 31:25–29, 2003.

[18] Andreas Fischer, Juan Felipe Botero, Michael Till Beck, Hermann de Meer, and Xavier Hesselbach. Virtual network embedding: A survey. *IEEE Communications Surveys and Tutorials*, 15:1888–1906, Feb 2013.

[19] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure. Tcp extensions for multipath operation with multiple addresses, Jan 2013.

[20] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. Understanding network failures in data centers: Measurement, analysis, and implications. In *ACM SIGCOMM*, volume 41, pages 350–361, Aug 2011.

[21] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: a scalable and flexible data center network. In *ACM SIGCOMM*, pages 51–62, 2009.

[22] Tao Guo, Ning Wang, Klaus Moessner, and Rahim Tafazolli. Shared backup network provision for virtual network embedding. In *IEEE ICC*, pages 1–5, Kyoto, Japan, Jun 2011.

[23] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz, and Nick McKeown. Reproducible network experiments using container-based emulation. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 253–264. ACM, 2012.

[24] Sandra Herker, Ashiq Khan, and Xueli An. Survey on survivable virtual network embedding problem and solutions. In *ICNS*, pages 99–104, Portugal, 2013.

[25] C. E Hopps. Analysis of an equal-cost multi-path algorithm, Nov 2000.

[26] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM J. Comput*, 3:299–325, 1974.

[27] Stavros G. Kolliopoulos and Clifford Stein. Improved approximation algorithms for unsplittable flow problems. In *IEEE SFCS*, pages 426–436, Oct 1997.

[28] Teemu Koponen, Keith Amidon, Peter Balland, Martn Casado, Anupam Chanda, Bryan Fulton, Igor Ganichev, Jesse Gross, Natasha Gude, Paul Ingram, et al. Network virtualization in multi-tenant datacenters. In *USENIX NSDI*, 2014.

[29] Diego Kreutz, Fernando M. V. Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey, Oct 2014.

[30] William Lau and Sanjay Jha. Failure-oriented path restoration algorithm for survivable networks. *IEEE Transactions on Network and Service Management*, 1:11–20, Apr 2004.

[31] Kayi Lee, Eytan Modiano, and Hyang-Won Lee. Cross-layer survivability in wdm-based networks. *IEEE/ACM Transactions on Networking (TON)*, 19(4):1000–1013, 2011.

[32] Bo Lu, Tao Huang, Xiao chuan Sun, Jian ya Chen, and Yun jie Liu. Dynamic recovery for survivable virtual network embedding. *The Journal of China Universities of Posts and Telecommunications*, 21:77–84, Jun 2014.

[33] Jing Lu and Jonathan Turner. Efficient mapping of virtual networks onto a shared substrate. *Washington University (WUCSE) Tech. Rep*, 2006.

[34] Athina Markopoulou, Gianluca Iannaccone, Supratik Bhattacharyya, Chen-Nee Chuah, and Christophe Diot. Characterization of failures in an ip backbone. In *INFOCOM, IEEE*, volume 4, pages 123–133, Mar 2004.

[35] Rodrigo R. Oliveira, Leonardo R. Bays, Daniel S. Marcon, Miguel C. Neves, Luciana S. Buriol, Luciano P. Gaspary, and Marinho P. Barcellos. Dos-resilient virtual networks through multipath embedding and opportunistic recovery. In *SAC*, pages 597–602, Coimbra, Portugal, Mar 2013.

[36] Rodrigo R. Oliveira, Daniel S. Marcon, Leonardo R. Bays, Miguel C. Neves, Luciana S. Buriol, Luciano P. Gaspary, and Marinho P. Barcellos. No more backups: Toward efficient embedding of survivable virtual networks. In *IEEE ICC*, pages 2128–2132, Budapest, Hungary, Jun 2013.

[37] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, November 1999.

[38] Muntasir Raihan Rahman, Issam Aib, and Raouf Boutaba. Survivable virtual network embedding. In *NETWORKING*, pages 40–52, May 2010.

[39] Muntasir Raihan Rahman and Raouf Boutaba. Svne: Survivable virtual network embedding algorithms for network virtualization. *IEEE Transactions on Network and Service Management*, 10:105–118, Feb 2013.

[40] Arup Raton Roy, Md. Faizul Bari, Mohamed Faten Zhani, Reaz Ahmed, and Raouf Boutaba. Design and management of dot: A distributed openflow testbed. In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, Krakow, Poland, May 2014.

[41] Ajay Todimala and Byrav Ramamurthy. A scalable approach for survivable virtual topology routing in optical wdm networks. *IEEE Journal on Selected Areas in Communications*, 25(6):63–69, August 2007.

[42] Y. Xiong and L. G. Mason. Restoration strategies and spare capacity requirements in self-healing atm networks. *IEEE/ACM Transactions on Networking (TON)*, 7(1):98–110, 1999.

[43] Jielong Xu, Jian Tang, Kevin Kwiat, Weiyi Zhang, and Guoliang Xue. Survivable virtual infrastructure mapping in virtualized data centers. In *2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*, pages 196–203, 2012.

[44] Jin Y. Yen. An algorithm for finding shortest routes from all source nodes to a given destination in general networks. *The Quarterly Journal of Pure and Applied Mathematics*, 27:526–530, 1970.

[45] Wai-Leong Yeow, Cedric Westphal, and Ulas C. Kozat. Designing and embedding reliable virtual infrastructures. *ACM SIGCOMM CCR*, 41(2):57–64, 2011.

[46] Minlan Yu, Yung Yi, Jennifer Rexford, and Mung Chiang. Rethinking virtual network embedding: Substrate support for path splitting and migration. In *ACM SIGCOMM CCR*, volume 38, pages 17–29, Apr 2008.

[47] Qi Zhang, Mohamed Faten Zhani, Maissa Jabri, and Raouf Boutaba. Venice: Reliable virtual data center embedding in clouds. In *INFOCOM IEEE*, pages 289–297, April 2014.

[48] Yong Zhu and Mostafa H. Ammar. Algorithms for assigning substrate network resources to virtual network components. In *INFOCOM, IEEE*, pages 1–12, 2006.