FACILITATING THE PROVISION OF AUXILIARY SUPPORT SERVICES FOR OVERLAY NETWORKS

A Thesis
Presented to
The Academic Faculty

by

Mehmet Demirci

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Computer Science

Georgia Institute of Technology
August 2013

FACILITATING THE PROVISION OF AUXILIARY SUPPORT SERVICES FOR OVERLAY NETWORKS

Approved by:

Professor Mostafa Ammar, Advisor
School of Computer Science
Georgia Institute of Technology

Professor Umakishore Ramachandran
School of Computer Science
Georgia Institute of Technology

Professor Ehab Al-Shaer
College of Computing and Informatics
The University of North Carolina at Charlotte

Professor Ellen Zegura
School of Computer Science
Georgia Institute of Technology

Professor Nick Feamster
School of Computer Science
Georgia Institute of Technology

Date Approved: June 26, 2013

ACKNOWLEDGEMENTS

I would like to thank

my advisor Prof. Mostafa Ammar for his enduring patience, guidance, encouragement and support, and my other thesis committee members Prof. Ellen Zegura, Prof. Nick Feamster, Prof. Ehab Al-Shaer, and Prof. Umakishore Ramachandran for their valuable time and advice;

my colleagues and lab mates for their support and kind wishes, especially Samantha Lo, Dr. Srinivasan Seetharaman, and Fida Gillani, whom I have had the pleasure and honor of working with;

past and present members of the College of Computing staff who have helped me diligently with all kinds of registration, reimbursement and paperwork issues.

Lastly, I would like to thank my mother and my father, and my sister Zeynep, for their unwavering love and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Network virtualization and overlay networks have emerged as powerful tools for improving the flexibility of the Internet. Overlays are used to provide a wide range of useful services in today's networking environment, and they are also viewed as important building blocks for an agile and evolvable future Internet. Regardless of the specific service it provides, an overlay needs assistance in several areas in order to perform properly throughout its existence.

This dissertation focuses on the mechanisms underlying the provision of auxiliary support services that perform control and management functions for overlays, such as overlay assignment, resource allocation, overlay monitoring and diagnosis. The priorities and objectives in the design of such mechanisms depend on network conditions and the virtualization environment. We identify opportunities for improvements that can help provide auxiliary services more effectively at different overlay life stages and under varying assumptions.

The contributions of this dissertation are the following:

1. An overlay assignment algorithm designed to improve an overlay's diagnosability, which is defined as its property to allow accurate and low-cost fault diagnosis. The main idea is to increase meaningful sharing between overlay links in a controlled manner in order to help localize faults correctly with less effort.

2. A novel definition of bandwidth allocation fairness in the presence of multiple resource sharing overlays, and a routing optimization technique to improve fairness and the satisfaction of overlays. Evaluation analyzes the characteristics of different fair allocation algorithms, and suggests that eliminating bottlenecks via custom routing can be an effective way to improve fairness.

3. An optimization solution to minimize the total cost of monitoring an overlay by determining the optimal mix of overlay and native links to monitor, and an analysis of the effect of topological properties on monitoring cost and the composition of the optimal mix of monitored links. We call our approach *multi-layer monitoring* and show that it is a flexible approach producing minimal-cost solutions with low errors.

4. A study of virtual network embedding in software defined networks (SDNs), identifying the challenges and opportunities for embedding in the SDN environment, and presenting two VN embedding techniques and their evaluation. One objective is to balance the stress on substrate components, and the other is to minimize the delays between VN controllers and switches. Each technique optimizes embedding for one objective while keeping the other within bounds.

CHAPTER I

INTRODUCTION

The Internet has firmly established itself as the cornerstone of information exchange. As the world continues to use the Internet more heavily, the demands and expectations from it are steadily increasing. Beyond the more traditional uses such as web browsing and email, people nowadays are widely using the Internet for more demanding services like video streaming and online gaming. As a result, the Internet is facing an ever-growing challenge to provide sufficient bandwidth, high reliability, efficient troubleshooting and strong security. *Network virtualization*, and more specifically, *overlay networks* have become an important tool in the last decade to face this challenge. Overlay networks have been employed as a means to provide a variety of services in today's networking environment. These services range from reliable routing [6, 71] to quality of service [25, 55, 81] and multicast [19, 43].

An *overlay network* is a *virtual network* built on top of another network. The underlying network is called the *substrate network*. There are two main types of overlay technology: overlays such as peer-to-peer networks, where users join and leave an established network; and overlays within a *network virtualization environment*, where substrate network providers (*infrastructure providers*) are decoupled from overlay network providers (*service providers*) that deploy and operate the overlay networks [29, 86]. In this thesis, we focus on the second type of overlays built in a network virtualization environment.

This process of mapping an overlay network to a substrate is sometimes referred to as *overlay assignment*, *overlay placement*, or *virtual network embedding* [31]. In this thesis, we will use these terms interchangeably. In an overlay assignment, each

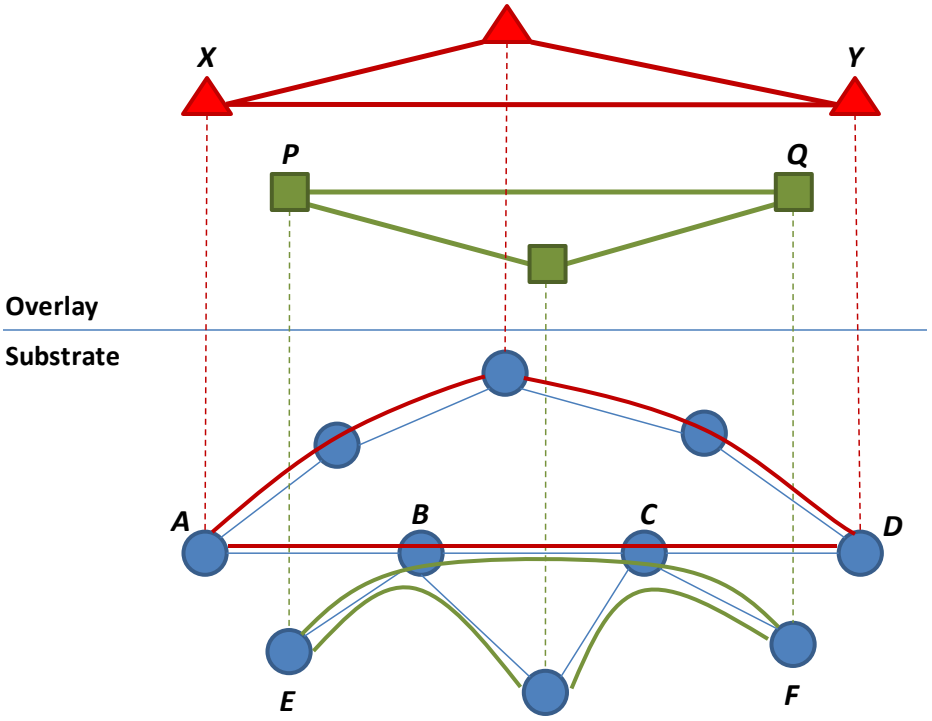


Figure 1: Two overlays mapped to a substrate

overlay node is mapped to a node in the substrate, and an *overlay link* corresponds to the substrate path between two overlay nodes. *Overlay routing* determines the path data packets will follow from a source to a destination in the overlay network. A portion of the physical resources of the substrate are assigned to an overlay network. Multiple overlay networks can coexist on the same substrate and share resources. For instance, in Figure 1, overlay links are mapped to the substrate using minimum-hop routing. The overlay link XY of the top overlay network is corresponds to the path $ABCD$ in the substrate. Similarly, the overlay link PQ of the bottom overlay network is mapped to $EBCF$ in the substrate. These two overlay links share the physical link BC.

1.1 Overlay Life Stages

Regardless of the specific service it provides, an overlay needs assistance in several areas in order to perform properly in every stage of its existence. We define four such life stages as follows:

1. **Conception**: The planning stage for the overlay. In addition to determining the goal of the overlay and the type of data to be transferred between nodes; the designer can also establish the overlay topology, pattern of communication and traffic demands at this stage. The result is an *overlay request* containing some or all of these parameters.

2. **Inception**: The *birth* of the overlay. The placement of the overlay onto the substrate, i.e., the mapping between the overlay topology and the native topology, takes place at this stage. Overlay nodes and links are mapped to substrate nodes and paths, and the necessary connections are established. An initial allocation of substrate resources may happen based on resource demands of the overlay and the available resources of the substrate.

3. **Activity**: The actual data transfer happens at this stage. During the activity phase, nodes may join or leave the overlay, faults and other performance problems may occur. To facilitate a fast response to such events, the overlay needs monitoring. The response may include *overlay reconfiguration* in some capacity.

4. **Departure**: The end of the overlay lifetime. The overlay gives the resources back to the substrate, connections are torn down, and the overlay ceases to exist on the substrate.

Throughout these stages of life, an overlay needs *auxiliary services* that help it function effectively. In this context, we define auxiliary services as those services that are not directly related to the main objective of data transfer between nodes, but

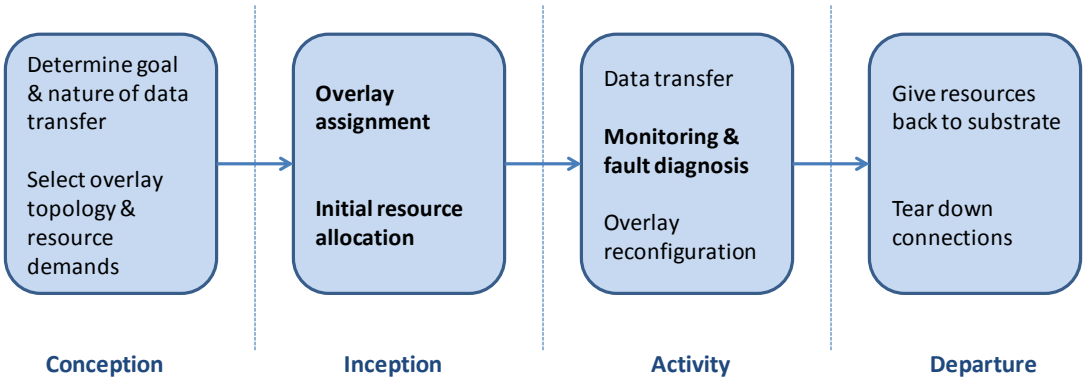


Figure 2: Life stages of an overlay

providing control and management functions. The auxiliary services we focus on in this thesis are overlay assignment, resource allocation, monitoring and diagnosis. We present a visual explanation of how these services fit into the overlay life stages in Figure 2.

The conception stage is outside the scope of this thesis. There has been a lot of work in *overlay topology design* with various objectives [20,44,54,56,88,94], which is an essential part of this stage. In the pieces of work that constitute this thesis, we assume that the overlay topology is given. The mapping of the overlay to the substrate, on the other hand, may be completely undetermined, partially determined, or given. We restate our assumptions in the problem statement section of each chapter. The departure stage is also not studied because the actions during the departure are not complicated enough to warrant a systematic study. Hence, the focus is on the inception and activity stages.

1.2 Thesis Statement and Contributions

The services written in bold in Figure 2 are within the scope of this thesis, and the objective is devising techniques to enable the provision of these auxiliary services in a fast, efficient and cost-effective manner. The *thesis statement* is as follows:

"Mechanisms enabling the provision of auxiliary services are important, their priorities must change depending on network properties, and an extensive arsenal of such mechanisms must be available to be deployed as necessary in a network virtualization environment."

The thesis consists of the pieces listed below. Each of these pieces focuses on facilitating one or more of the auxiliary services mentioned: overlay assignment, fault diagnosis, resource allocation and performance monitoring.

Overlay network placement for diagnosability

An important aspect affecting the performance of an overlay is *overlay assignment*, which describes the mapping between nodes and links at the overlay layer to those at the substrate layer. Algorithms that perform overlay assignment have a variety of objectives. We consider the objective of *diagnosability*, which we broadly define as an overlay's ability to allow accurate diagnosis with low overhead.

We study the question of how to design an overlay to be more accommodating to monitoring systems, i.e., to allow for easier diagnosis and to keep the overhead manageable. We define diagnosability formally in terms of diagnosis *accuracy* and efficiency. We examine certain properties of the topological mapping between the overlay and the substrate, particularly those related to the amount and nature of sharing between overlay components, and develop a placement algorithm to improve an overlay's diagnosability. We assume a passive end-to-end measurement structure and evaluate the placement algorithm by exploring how it affects the performance of this measurement system.

Fair resource allocation among overlay networks

Many overlay networks offering different services may be placed on the same substrate and compete for resources such as node CPU time or link bandwidth. As the number

of overlays and their resource demands increase, the substrate may not be able to meet all demands. Thus it is necessary to allocate substrate resources fairly and efficiently among competing overlays.

One important resource is substrate link bandwidth. The capacity of each substrate link must be divided among overlay flows sharing that link. The fairness of this division becomes especially important in case of resource scarcity and when overlay traffic constitutes the majority of the load on the substrate. In this work, we consider fairness in terms of the treatment toward whole overlays rather than individual flows. We explore different definitions of fairness, and formulate a fairness evaluation metric for allocations involving multiple sharing overlays. Moreover, we investigate routing optimization techniques to improve fairness and the satisfaction of overlays.

Cost-effective multi-layer monitoring of overlay networks

Monitoring all links in overlay networks is necessary to assess the performance observed by users and to detect anomalies. Monitoring must be sufficiently frequent for fast detection of problems and continue throughout the lifetime of the overlay. A simple strategy is to monitor all overlay links directly and individually. This approach can cause significant overhead, particularly when the monitoring is performed via active measurements. In this work, we adopt a more flexible approach that allows certain native link measurements in addition to end-to-end overlay measurements. Native link measurements can be used to infer desired metrics for overlay links by suitable combinations on native layer metrics. We focus on the *latency* metric although the work can be extended to certain other metrics such as loss rate.

Our goal is to minimize monitoring cost by determining the optimal mix of overlay and native links to monitor. We formulate an optimization problem to achieve this goal and implement it as an *integer linear program* (ILP). Through simulations, we determine that monitoring a combination of native and overlay links provides a

lower cost than overlay-only or native-only monitoring strategies. We also study how topological properties affect the monitoring cost and the compositions of the optimal mix. Lastly, we perform an experimental evaluation of *inference errors* that result from combining direct measurements to infer the unmeasured end-to-end metrics. We observe that while errors are manageable, in most cases a few inferred links produce high errors that increase the average error.

Mapping virtual networks to software defined network substrates

Software-defined networking has emerged in the last few years as a powerful approach to improve the customizability and flexibility of networks. In this work, we focus on network virtualization in the realm of software-defined networks (SDNs), and study how to embed virtual networks in this environment. Virtual network embedding is an important problem because intelligent embedding can lead to better performance and a more efficient allocation of network resources compared to random mapping. In SDNs, the presence of a central controller is a complicating factor, and customizable routing and differences in resource sharing present new opportunities and challenges.

We identify two aspects of virtual network (VN) embedding in SDNs: virtual node and link mapping, and controller placement. We tackle these problems together, developing techniques to perform embedding with two goals: balancing the load on the substrate network and minimizing controller-to-switch delays. We evaluate our techniques with simulation and Mininet emulation, and show that they are able to optimize for one of the above objectives while keeping the other within reasonable bounds.

NP-hardness and the use of heuristics: The overlay assignment problem with bandwidth constraints is proven to be NP-hard [7]. In our versions of the assignment problem, we assume there are *stress* (a measure of the load on a substrate component)

constraints on links, which can be mapped to bandwidth constraints for overlay flows. In addition, there are stress constraints on nodes as well. Hence, these problems are also NP-hard and necessitate the use of heuristics, which we develop end evaluate.

Target audience: The parties that can benefit from the ideas and methods in this thesis are substrate providers (infrastructure providers) and overlay providers (service providers). For example, an infrastructure provider can employ the overlay assignment techniques to balance the load on the substrate or make overlays more diagnosable depending on the priorities, offer multi-layer monitoring to overlays so that the cost to the network will be minimized, and use the fair resource allocation metric and algorithms to improve the satisfaction of the overlays. On the other hand, a service provider can be given the freedom to specify its priorities and choose an assignment technique from alternatives, or decide whether to take advantage of multi-layer monitoring after simulating costs for its overlays.

1.3 A Functional View

Figure 3 illustrates the organization of a general network virtualization environment (NVE) and how the pieces of work in this thesis fit into such an environment. We briefly describe each of these modules.

- *Service Provider (SP):* In an NVE, multiple SPs build virtual networks offering a variety of end-to-end services to end users who can opt-in to different SPs, i.e., users enter into agreements with SPs as they desire for the end-to-end services they would like to receive. The SP negotiates with the infrastructure provider (InP) for the resources needed to deploy the VN. Although only one SP is shown in Figure 3, multiple SPs can build and run their VNs on the infrastructure provided by one InP.

- *VN Assignment Module (VAM):* The VAM handles VN instantiation on the

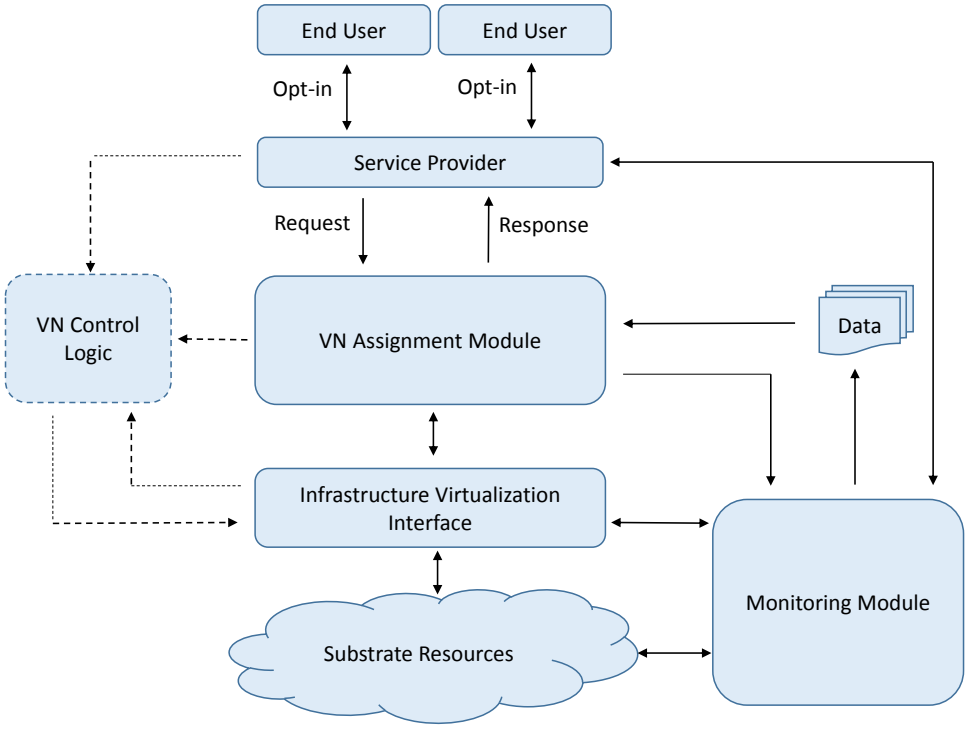


Figure 3: Functional architecture of a network virtualization environment

infrastructure. It provides an interface to the service provider (SP) and receives VN requests through this interface (e.g., NetFinder [97] provides such an interface for PlanetLab). It collects monitoring data from a repository or directly queries the monitoring module to learn about the state of the substrate. Based on the VN requests and the information about the substrate, it performs VN embedding, which may or may not be successful depending on constraints imposed by the SP or the infrastructure provider. The VAM can select among multiple VN embedding methods. Our work on overlay placement for diagnosability, fair resource allocation among overlays, and VN embedding in SDNs all fit into this module. It can be extended to include a reconfiguration component, which would inform the SP or the VN controller (defined below) about reconfiguration decisions.

- *Monitoring Module (MM):* This module constantly monitors the substrate for key metrics such as available bandwidth, CPU usage, end-to-end delays and loss rate. It also detects and localizes faults. Our work on multi-layer monitoring, which applies to delay and potentially loss rate monitoring, fits here. The MM can include a separate submodule for each VN and allow the SP to customize the type, frequency and intensity of the monitoring it desires. For instance, each SP can tell the MM whether it allows the use of multi-layer monitoring for its VN.

- *Infrastructure Virtualization Interface (IVI):* This module acts as a translation layer between the substrate and the virtual networks. A tool such as FlowVisor [76] is an example of such a layer in SDNs. It regulates the communication between the substrate and the VAM and VN controllers. In addition, it implements the VN in the substrate by providing isolation between virtual resources and scheduling. Problems regarding the IVI are outside the scope of this thesis.

- *VN Control Logic:* In the SDN environment, each VN is typically controlled by a separate controller. This controller is delegated as the central logic for the VN by the SP, and it communicates with the virtual nodes in the VN as necessary through the IVI. It also receives information about the state of the substrate from the MM via the IVI. We tackle the placement of these controllers in tandem with the VNs in our relevant work.

The three pieces VAM, MM, and IVI can be thought of as part of the infrastructure provider, while the VN controller is closely coupled with the service provider.

The Global Environment for Network Innovations (GENI) [33] project offers a virtual laboratory aiming to provide networking researchers with a platform that enables and supports a large number of simultaneous experiments on a shared infrastructure. GENI makes heavy use of virtualization to enable slice-based experimentation for

researchers. We present a comparison between our Figure 3 and an architectural diagram included in a document providing an overview of the GENI suite [64] to demonstrate that they are similar and our illustration is consistent with what happens in a real example of a virtualized infrastructure.

As shown in Figure 3 in [64], GENI users (experimenters) contact *aggregate managers* (AM, represented by FOAM in the figure) to request resources for their experiments. The AM then performs resource assignment and gives a slice of network resources to the user, who controls the slice via a NOX controller [66]. This controller communicates with the OpenFlow [63] switches through FlowVisor.

In our Figure 3, the service provider (SP) replaces the experimenter in GENI because the SP is responsible for constructing and deploying its virtual network (VN) with the intention of offering end-to-end services to end users. The VAM in our figure is analogous to the AM in the GENI figure. An instance of the IVI in our figure is represented in the GENI figure by the FlowVisor controller, which enables slicing the underlying substrate network. It provides a virtualization interface between the NOX controller (VN control logic in our Figure 3) and the OpenFlow switches that constitute the substrate nodes.

The GENI figure shows that an experimenter can receive resources from multiple networks to add to her slice. Similarly, our Figure 3 can be extended to include multiple SPs and multiple InPs, with each SP aggregating resources obtained from multiple InPs to construct its VN. In that case, each SP can talk to the VAMs from different InPs directly, or to a third party *broker* in charge of regulating the relationships between SPs and InPs. Virtual networks assembled from multiple InPs are outside the scope of this thesis.

1.4 Common Features and Design Priorities

A common feature of our approaches in this thesis is that they consist of *proactive* methods. We design methods for:

- Overlay placement so that subsequent fault diagnosis will be easier,
- VN embedding in SDNs so that subsequent communication delays will be lower and the load on the substrate will be more balanced,
- Fair resource allocation and routing modification so that subsequent network utilization and satisfaction of overlays will be higher,
- Selecting a set of native and overlay links for direct monitoring so that subsequent monitoring will have minimal cost.

Priorities in the design of these techniques depend largely on several factors such as the topological properties and resource availability of the substrate network, the number and sizes of overlay networks, their demands from the substrate and performance expectations. We briefly discuss some factors that influence design choices.

- **Substrate resources:** When substrate resources are scarce, it makes sense to prioritize fair allocation of those resources among competing overlays. When resources are abundant, other objective such as high diagnosability can be given priority.
- **Number and sizes of overlays:** We can combine these variables into the concept of *overlay density* which can be defined in several ways. For instance, we can say that overlay density is the ratio of the total number of overlay nodes to the number of substrate nodes. For networks with high density, a reasonable priority is load balancing.

Table 1: Relationship between priorities and methods

Priority \ Stage	Inception	Activity
Resource consumption	*Fair bandwidth allocation*	*Multi-layer monitoring*
Fast diagnosis	*Placement for diagnosability*	
Low latency	*Delay-minimizing VN embedding*	
Load balancing	*Stress-balancing VN embedding*	

- **Overlay demands:** Overlay demands and substrate resource availability affect priorities in opposite ways. High demands improve the importance of fair resource allocation and load balancing.

- **Other requirements:** Priorities may depend on the type of network and virtualization environment. For example, one current feature of network virtualization in SDN is the use of a separate controller per virtual network. It is important to ensure that the delays between the controller of a virtual network and its virtual nodes are low.

1.5 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 examines the previous work in the area in relation to this thesis. Chapter 3 discusses the placement of overlays for diagnosability. Chapter 4 presents a study on fair resource allocation among overlays. Chapter 5 explains our work on the multi-layer monitoring of overlay networks. Chapter 6 provides techniques for mapping VNs to SDN substrates. Chapter 7 summarizes the thesis and provides future work directions.

CHAPTER II

RELATED WORK

This chapter presents an overview of research on overlay networks, particularly in the topics of overlay assignment, resource allocation, monitoring and diagnosis. Network virtualization and overlay networks have been utilized for more than a decade to circumvent certain limitations of the Internet. Researchers have developed numerous overlays offering valuable functionalities that were otherwise too challenging to incorporate into the Internet, and virtualization technology continues to be helpful in confronting the impasse in terms of network innovation [8, 17].

One of the earliest instances of overlay networking was Mbone [27], an experimental backbone built as an overlay on top of the Internet for IP multicast traffic. Mbone utilized IP in IP tunneling [78] to connect multicast routers into an overlay and enable multicasting without requiring every router in the native network to have multicast capability. Although the number of multicast-capable routers in the Internet backbone has increased substantially over the years, IP multicast has not enjoyed widespread deployment due to its complexity. Researchers have proposed application level overlays [19, 43] for multicast as a low-cost and scalable alternative to IP multicast.

Other examples of services provided by overlays include content distribution [5,13, 52], peer-to-peer file sharing [12,32,60,80], quality of service [25,55,81], security [48,77, 79] and resilient routing [6, 71]. Recent efforts to enable innovation in the Internet, such as GENI [33] and Internet2 [39], make use of network virtualization to allow networking researchers to share the infrastructure simultaneously. One can envision that overlay networks are likely to be a significant component in the future networking

environment [21, 30, 68], where many overlays offering a variety of services may be placed onto a shared substrate. In fact, testbeds such as X-Bone [85], PlanetLab [69] and VINI [10] allow numerous virtual networks to coexist on the same substrate.

Multiple coexisting overlays on a shared substrate create a number of interesting research problems. For instance, overlay networks must be able to route around problematic nodes and links in the substrate for sustained operation. RON [6] exploits the existence of multiple substrate paths between two end nodes to avoid problems and offer resilient routing for overlays. Overlay networks may also prefer avoiding congested links and route their data over paths with high available bandwidth. To this end, Zhu et al. proposed a dynamic overlay routing scheme based on available bandwidth estimation [95]. Another interesting problem is the interaction between overlay routing and traffic engineering [57, 73]. Seetharaman et al. develop strategies to mitigate the instability resulting from conflicting objectives of protocols at different layers [74].

The above works concentrate on routing and traffic engineering to improve the quality of data transfer. Another dimension of overlay networking research can be classified as creating a healthy operating environment for overlays. There is a necessity for efficient monitoring without excessive overhead, so that problems can be discovered and addressed quickly. In addition, substrate resources must be shared in a fair manner among overlays, and overlay assignment algorithms must be developed to avoid inefficient performance. We group these efforts under the heading *auxiliary support services* for overlays, which are the focus of this thesis, and organize the rest of the related work by topic accordingly.

2.1 Overlay Assignment

Overlay network assignment (also known as overlay / virtual network mapping / embedding / placement) with node and link constraints is a complex problem and can

be reduced to the NP-hard multi way separator problem [7]. Much of the research in this area has focused on developing heuristics by shrinking the problem space through various assumptions [17].

Researchers have studied several aspects of the overlay network assignment problem and developed overlay assignment algorithms with a number of varying objectives [31]. Zhu and Ammar [96] proposed algorithms to place overlay networks onto the substrate to balance stress and thus avoid spots where resource contention becomes too high. Lu and Turner [59] concentrate on embedding a single overlay in a cost-efficient manner. Yu et al. [92] also study the problem of embedding virtual networks efficiently within resource bounds, but they allow for flexibility in the substrate with path splitting and path migration (or rerouting) in order to expand the problem space and find better solutions. Chowdhury et al. [18] used a mixed-integer program to perform coordinated node and link mapping resulting in increased acceptance ratio and revenue. Han et al. [35] advocated designing overlay networks to increase path independence between end nodes and developed overlay node placement heuristics to improve the robustness of the overlay. Our work on assignment for diagnosability extends this literature by considering diagnosability as an objective to overlay network placement.

Overlay design for diagnosability is influenced by the chosen diagnosis technique. We use the passive diagnosis scheme described in [82] to evaluate the diagnosability of our placements. A different diagnosis scheme, such as one based on passive measurements supported by selective active probing, may call for a modified approach to overlay placement. Such a system is described in the work by Tang et al. [83]. That system relies on passive measurements at the initial stage, and subsequently augments these measurements by systematically selected active probes to improve the accuracy of diagnosis.

2.2 Fair Resource Allocation

Allocating link bandwidth fairly among the flows within a single network has been widely studied. The *max-min fairness* definition in [11] is accepted as a good solution to this problem. Other fairness definitions related to max-min fairness have been proposed, such as *proportional fairness*, [62] which is an example of the *utility approach to fairness*.

Karbhari et al. [46] studied how these fairness definitions can be modified to deal with a number of multipoint-to-point sessions. We show that these definitions are also applicable in the multiple overlay network setting. The authors also worked on maximizing the throughput of end-to-end data carried on overlay paths using TCP [45], but they considered a single overlay network rather than multiple overlays competing for resources, as in the scenario in our resource allocation work.

A commonly used metric to evaluate the fairness of bandwidth allocation is Jain's fairness index [42], which defines uniform rates throughout the set of all flows as perfect fairness. In our work, we are concerned with the fairness of the allocation to entire overlays rather than single flows, and we calculate a network-wide value for each overlay representing its satisfaction with the allocation for evaluating fairness.

Allocation of resources to multiple overlay networks is studied in [22]. However, this work focused on a middleware solution that mediates resources according to a given proportion or fairness definition. Kleinberg et al. [50] considered tailoring network routing in order to optimize fairness. They prove that this problem is NP-complete, and develop approximation algorithms for this optimization, using the max-min fairness definition.

2.3 Overlay Monitoring and Diagnosis

Network monitoring and fault diagnosis are well-studied problems. Numerous monitoring techniques and measurement systems have been developed for overlay systems.

These techniques can be mainly grouped into two categories: active methods [1,15,93] and passive methods [9,38,67,82]. Active monitoring offers flexibility through probing the network at a desired frequency, but introduces additional overhead which may become crippling in the presence of multiple independent overlays conducting active probing. A shared underlay open to queries from overlays was proposed [65] as a solution to this problem.

Another approach to address the issue of probing overhead is to minimize the amount of probing traffic through intelligent design of the monitoring structure. An active monitoring system is presented in the work by Chen et al. [15], where a minimal *basis set* of overlay paths are monitored to infer the loss rate measurements of all overlay paths. iPlane [61] predicts end-to-end path performance from the measured performance of segments that compose the path. In our multi-layer monitoring work, we generalize these approaches and allow measuring both end-to-end paths and underlying native links. This extra degree of freedom makes it possible to monitor an even smaller set in most cases, where the information obtained from monitoring this set is then used to deduce the measurements of paths that are not directly monitored.

A passive monitoring technique that minimizes the number of monitored network paths in enterprise networks is presented in [2]. Tang and Al-Shaer [82] propose another passive diagnosis method, an evidence-based fault reasoning scheme based on belief evaluation that handles uncertainty using Dempster-Shafer theory [75]. We interface the technique in [82] with our overlay placement and use it as our framework for overlay diagnosis in our diagnosability work.

The problems of effective monitoring and optimizing monitor placement have been studied in the realm of optical networks as well. Researchers have utilized monitoring cycles [90] and paths [3,4] to improve failure localization. Designing monitoring trails (m-trails) [91] to minimize the total cost on the network was the subject of recent work [84]. Our work is distinct in two regards: First, our monitoring objective is

determining metrics (such as delay) for the set of all overlay links, not localizing single-link failures in the substrate network. Second, m-trails in optical networks contain non-simple cycles and paths, whereas overlay links are simple paths that allow less flexibility than m-trails.

2.4 Virtualization in Software Defined Networks

Software defined networking (SDN) separates the control plane from the physical switches comprising the data plane and allows tenants to run their own control logic on their own controllers. SDN is a suitable platform for network virtualization and researchers have been working on ways to provide virtualization to enable the coexistence of multiple virtual networks (VNs) in the SDN environment. FlowVisor [76] has been proposed as a virtualization solution in SDNs. In our work about VN embedding in SDNs, we assume the availability of a FlowVisor-like virtualization tool and analyze FlowVisor to identify the network resources to be shared in the presence of multiple VNs coexisting on an SDN substrate. Our approach is not necessarily dependent on the exact type of resource sharing or virtualization technology and it should be able to work with other SDN virtualization solutions, such as FlowN [24].

Various techniques for performing virtual network embedding in traditional networks have been proposed by researchers [31]. However, distinctions of the SDN environment (such as the centrality and the importance of the controller, and differences in the virtualization technology) necessitate a new approach. We utilize some definitions and ideas from the stress-balancing VN assignment algorithm presented in [96] and adapt them to our problem. We define stress in a modified way, and identify stress-balancing as one of two VN embedding objectives considered in this work.

The placement of controllers in SDNs has been studied [37] with the goal of minimizing the average and maximum delays from the controller to the switches. The

authors do not propose a specific method for this placement, but they evaluate how much controller placement affects said delays by finding the optimal placement via brute force. The location of the network is specified beforehand except for the placement of the controller, which can be moved around. In our work, we incorporate the objective of delay minimization into our embedding techniques. However, we consider a more general situation where there are many VNs whose controllers are either fixed at a predetermined location or free to be placed anywhere. Furthermore, the topology of each VN is given, but the mapping of that topology to the SDN substrate is yet to be determined.

CHAPTER III

OVERLAY NETWORK PLACEMENT FOR DIAGNOSABILITY

3.1 Introduction

Overlay networks must be constantly monitored for performance issues ranging from increased delay or congestion to node and link failures. It is important to diagnose network problems quickly and accurately, as well as to keep the overhead caused by the monitoring structure and incurred by the network to a tolerable amount. Efficient monitoring of substrate and overlay networks is a well studied problem and researchers have developed a myriad of monitoring techniques. In this work, we address the question of how to manipulate the placement of an overlay network to be more friendly to monitoring systems: to allow for easier diagnosis and to keep the overhead as low as possible. We specifically consider techniques that use *end-to-end network information* to diagnose problems in the network, both at end nodes and intermediate components. These end-to-end monitoring techniques can be active, passive or a combination of these two.

We concentrate on an overlay network's property to allow accurate diagnosis of potentially many faults with low overhead, and call this property *diagnosability*, which we define formally later in the chapter. Overlay network diagnosis is a difficult problem because overlays are built as virtual networks on top of a substrate. We explore how the placement of overlays on the substrate influences their diagnosability, and we develop an overlay placement algorithm where the main goal is maximizing diagnosability, but we also strive for placements that do not ignore and degrade other overlay performance metrics. As for the monitoring, we employ a passive end-to-end

measurement system called *collaborative diagnosis* based on *evidence based reasoning* [82]. We do not propose a new monitoring technique, but rather a method to map the overlay network onto the substrate so that monitoring performance can be improved.

Mapping overlays to network substrates, i.e., overlay network placement, has been the subject of several studies [31]. Researchers have identified and designed for various objectives, such as balancing the load on the substrate, maximizing the number of overlays that can be accommodated, or minimizing the total cost of carrying traffic. We identify diagnosability as a new objective. Regarding diagnosability as a priority makes sense if the substrate network is well-provisioned and/or the number of overlays and their resource demands do not put a significant load on the substrate. In such networks, fast and accurate fault diagnosis can become a primary design goal, and objectives such as efficient and balanced resource consumption can be relegated to a secondary status.

Why should we think about overlay placement optimized for diagnosability? A large part of diagnosability depends on the topology of the substrate network, nonetheless, the placement of the overlay upon the substrate also plays a significant role in determining the overlay's diagnosability. We present an example to illustrate this point: Figure 4 shows two placements of the same three-node overlay onto two different parts of a substrate. If the overlay is placed as ABC, a failure of node X could be observed on path AB through end-to-end monitoring of that path. However, the fault cannot be pinned down to a single component because there are two substrate links and a substrate node that can be the source of the fault. If the placement is ABC', the failure of node X would be observed on all three end-to-end paths and can be uniquely identified with a very high probability of being correct.

Other similar examples suggest that increased sharing of substrate components between overlay nodes and links can help localize faults more successfully, increasing

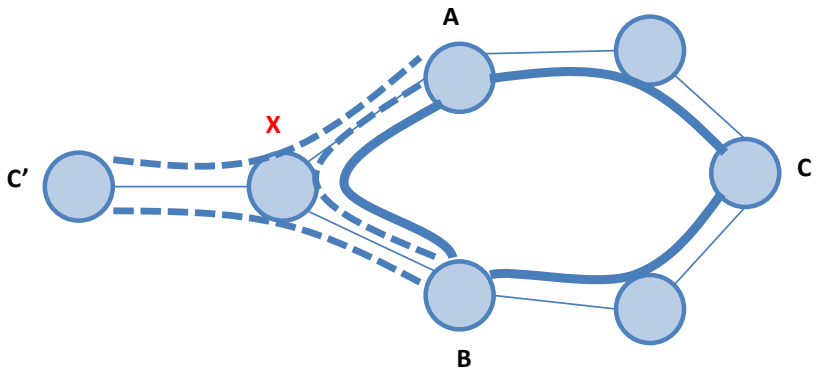


Figure 4: Sample topology with two possible placements of an overlay

diagnosability. However, there is usually a trade-off between diagnosability and the amount of *stress*[1] on substrate components, and higher diagnosability comes at the expense of stressing substrate components.

The rest of the chapter is structured as follows: We describe our model, define overlay placement and diagnosability formally, and discuss how efficiency and accuracy are two key components of diagnosability in Section 3.2. Section 3.3 presents ideas about overlay placement for diagnosability and an algorithm to perform the placement designed to improve diagnosability. We describe the passive monitoring-based diagnosis framework we utilized in Section 3.4. We evaluate the performance of our placement algorithm in Section 3.5. Finally, we provide a summary of the chapter in Section 3.6.

3.2 Model and Problem Statement

We model the substrate network as a directed graph $G = (V, E)$, where V is the set of vertices and E is the set of directed edges connecting these vertices. We model the overlay network as a directed graph $G' = (V', E')$, with $V' \subseteq V$ being the set of

[1]For a substrate node, stress is the number of overlay nodes mapped to it. The stress on a substrate link is the number of overlay links traversing it [96].

overlay nodes and E' being the set of overlay links. An *overlay node placement* is a mapping from V' to V which matches each overlay node to a substrate node. Each overlay link is then constructed as a series of substrate links starting with the source node and ending with the destination node. An overlay node placement along with the overlay links constitute an *overlay placement*.

We assume the existence of a diagnosis scheme detecting and localizing *persistent faults* in substrate components, which manifest themselves as end-to-end problems that are observable by different end users in a consistent way. Examples of such faults are component failures that result in loss of connectivity between end nodes. We assume such faults are rare events, so we do not consider more than two simultaneous faults. Fault diagnosis relies on end-to-end measurements only, and diagnosis software runs on all or some overlay nodes.

A fault at a substrate component (node or link) may be observed by one or more overlay paths depending on network topology and traffic patterns. The diagnosis technique produces a *suspect set* S, which contains all substrate components suspected to be the root cause of the observed end-to-end problem(s). We describe a specific diagnosis scheme from previous work in Section 3.4, and use it to test our placement technique. However, our placement method is designed to work with different diagnosis schemes, as long as they are based on end-to-end measurements.

Depending on the diagnosis scheme used, an overlay link may not be able to produce useful measurements (due to lack of traffic in a passive scheme or reachability issues in an active scheme). We define *covered overlay links* as those overlay links that are able to assist the diagnosis by producing meaningful measurements. If information as to which overlay links are covered and which are not is known beforehand, this information can be used in placing the overlay. More details about this are in Section 3.3.

We study methods to place overlays on the substrate in order to increase the ease

and efficiency of fault diagnosis. We address the following problem: *Given a substrate topology and an overlay topology, how can we place the overlay on the substrate so that the overall diagnosability of used substrate components will be maximized, without putting too much stress on substrate nodes and links?* A critical question is how to define diagnosability in a specific and quantifiable way.

Definition

Diagnosability is the ability to diagnose up to n faults in a network with *accuracy* and *efficiency*.[2] As the accuracy and the efficiency improve, so will the diagnosability. We now define these two concepts in a concrete way.

Accuracy

Let W be the set of faulty components. As long as $W \subseteq S$, there are no false negatives in the diagnosis as every faulty component is included in the suspect set S. However, if $W \not\subseteq S$, false negatives exist. We define *accuracy* below in terms of the probability of false negatives. As the probability of false negatives decreases, accuracy increases.

$$Accuracy = 1 - \mathrm{P}(\mathrm{falseNegatives}) = 1 - \mathrm{P}(W \not\subseteq S) \tag{1}$$

Efficiency

As the size of S grows, it becomes more difficult to pinpoint the source of the problem to a specific set of substrate components. If the diagnosis is accurate, $|W|$ has to be smaller than $|S|$; but if $|S| \gg |W|$, then there will be too many combinations of the elements in S that can comprise W. If $|S|$ is close to $|W|$, it will be easier to localize the source of the problems and determine W precisely. Thus a smaller suspect set signifies a higher efficiency in diagnosis.

[2] Accuracy and efficiency are similar to *completeness* and *resolution* in fault diagnosis, respectively [87].

Accuracy and efficiency are conflicting goals: If diagnosis focuses on accuracy, it can produce a large suspect set to avoid false negatives, in which case efficiency will suffer. If, on the other hand, diagnosis gives priority to efficiency, it risks a greater probability of false negatives. It is important to find a balance in this tradeoff for an effective diagnosis. We combine these two objectives into a single metric to evaluate diagnosability more easily. The idea is to consider the amount of work (represented by *diagnosis cost* defined below) left to localize the fault to a precise set of components after the diagnosis produces the suspect set S. If the diagnosis is accurate, the search space is limited to S. However, if the diagnosis is not accurate, i.e., there is at least one faulty component that is not in S, the worst case will require checking every substrate component in the overlay to determine W.

For a fault scenario where W is the set of faulty components in a network with χ substrate components and S_W is the suspect set produced by the diagnosis method, let us define a *binary variable* α_W to indicate whether the diagnosis is accurate or not in this particular fault scenario. The *diagnosis cost* for this fault scenario can be defined as follows:

$$cost(W) = \alpha_W |S_W| + (1 - \alpha_W)\chi. \tag{2}$$

If the diagnosis is accurate, $\alpha_W = 1$ and the cost is only the size of S_W; but it grows to χ in case of inaccuracy. Note that over many fault scenarios, total cost is calculated as an average of individual costs. Let us assume that we perform the diagnosis using a collection of fault scenarios, denoted by Φ.

$$AvgCost = \frac{\sum_{W \in \Phi} [\alpha_W |S_W| + (1 - \alpha_W)\chi]}{|\Phi|}. \tag{3}$$

We theorize that if accuracy is given priority, suspect sets are going to be larger and α values are going to be 1 most of the time. An extreme case is where $|S_W| = \chi$

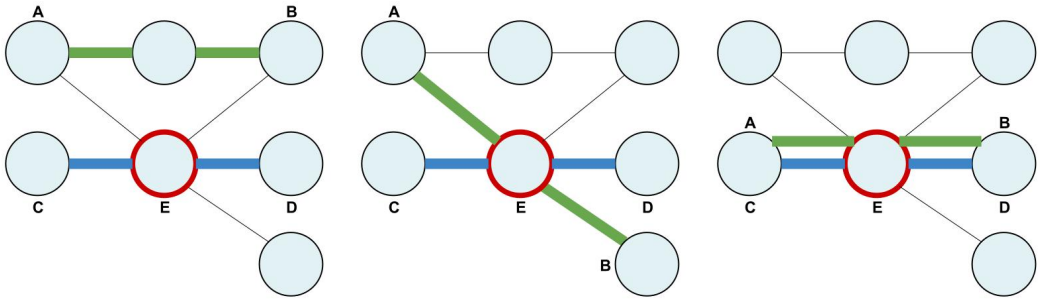
in all scenarios, which ensures perfect accuracy, but is just as bad as zero accuracy in terms of diagnosis cost. Similarly, an extreme priority placed on efficiency will in turn hurt accuracy and have the same detrimental effect on cost. Finding the balance between these two objectives is crucial to minimize cost, which should be the goal of the diagnosis scheme.

3.3 Overlay Placement for Diagnosability

In this section, we present our technique for performing overlay network placement to improve diagnosability. Overlay placement is a difficult problem, and finding optimal solutions to this problem is complex [59, 96]. Hence, we resort to the use of a heuristic approach. Below, we discuss the ideas behind our approach and describe our placement algorithm in detail.

The amount of sharing among overlay paths is a major variable affecting how precisely an observed fault can be localized. In general, increased sharing results in faults being observed by more end nodes and gives more information about the location of faults. With end-to-end diagnosis schemes, each measurement or observation provides information about a particular end-to-end path. An observed fault signifies that at least one component along the corresponding path is faulty. (If we assume faults are rare, we can say only one component is faulty.) If the fault is observed on multiple end-to-end paths, then we can narrow down the search for the faulty component(s) to the places where these paths overlap with each other. Hence, an overlay mapping with more sharing will provide more information about the states of substrate components. On the other hand, too much sharing (such as a single component appearing in every overlay path) can cause a single point of failure, which is undesirable, as well as performance issues such as congestion and ineffective bandwidth allocation.

Our overlay placement algorithm makes use of the above idea by making sure there is significant sharing among overlay paths. However, excessive sharing is not



(a) Placement 1 (b) Placement 2 (c) Placement 3

Figure 5: These placements illustrate how placement may affect diagnosability.

only potentially bad for performance but also it can lead to less efficient diagnosis. We present a simple example in Figure 5 to illustrate this. We assume there are only two overlay links in this overlay, AB and CD. As we see in Figure 5a, there is no sharing among the two paths in the first placement. The fault that occurs at node E can be observed only at overlay nodes C and D. We can deduce that at least one component along the path CD is faulty, but we have no way of knowing which exact component is faulty.

In the placement shown in Figure 5b, AB and CD share the intermediate node. When it fails, this event is observed at all four overlay nodes. Since E is the only place where AB and CD overlap, we can infer with high confidence that E is the faulty component. In this case, sharing among end-to-end paths helps the diagnosis.

Consider the placement shown in Figure 5c where A and C are placed on the same substrate node, and so are B and D. In this extreme case of sharing, AB and CD overlap completely. In the absence of additional end-to-end data, a fault in E does not give us any more information than in Figure 5a. This example shows that while a certain amount of sharing is good for diagnosability, total overlap between end-to-end paths is undesirable. Therefore, the main ideas behind our placement algorithm are the following:

i. *Increase sharing between overlay paths on average.* The more end-to-end paths

that share a substrate component, the higher the number of paths observing a fault at that component. This is helpful in localizing the fault.

ii. *Limit the maximum stress on a component.* A substrate node/link shared by too many overlay nodes/links has more risk of congestion and resource exhaustion. We need an upper limit on the stress on a substrate component.

iii. *Limit the number of overlay paths that overlap almost completely.* Two overlapping end-to-end paths observing a fault provide no more valuable information for diagnosis than one path. Sharing in this way will increase average sharing, but must be avoided if possible because it is not helpful to the diagnosis.

We want to combine these three ideas into a single metric that represents the suitability of a placement to our goals. Our heuristic approach to this starts by defining the *quality* of the placement, denoted by q, as a function of three individual parameters that represent the above three ideas.

i. *Average paths per link (p)*: The average number of overlay paths that share a substrate link. This can be calculated by adding together all overlay path lengths and then dividing the result by the number of distinct substrate links appearing in these paths.

ii. *Maximum stress on a component (m)*: The highest stress on a substrate node or link. We introduce a maximum threshold on m and denote it T. With no threshold, we may end up with placements that put an intolerable amount of stress on components and cause congestion or inefficient resource allocation. We discuss the selection of a suitable T in Section 3.5.

iii. *Number of overlay path pairs that differ from each other by no more than one substrate link (o)*: For instance, a pair of 3-hop paths that share at least 2 links would count toward this number. If they shared just 1 link, they would not count.

A high p is good for diagnosability and we want to maximize it while keeping m under the threshold T. Conversely, a high m and o are not desirable. We reflect these

ideas in the definition of q in terms of p, m, o and T in Equation 4.

$$q = p - (k_1 \cdot m + d \cdot \delta(m, T) + k_2 \cdot o) \tag{4}$$

$$\delta(m, T) = \begin{cases} 1, & \text{if } m > T, \\ 0, & \text{otherwise .} \end{cases} \tag{5}$$

k_1 and k_2 are coefficients that determine the amount of weight given to m and o in the assessment of placement quality. A suitable default value for k_1 and k_2 is 0.1. d is a *detractor*, a large number (e.g., 1000) to reduce the quality significantly if m exceeds the threshold T. When $m > T$, $\delta(m, T)$ is 1, and d is factored into the calculation of q.

We conducted experiments to understand the relationship between q and the diagnosis cost defined in Section 3.2. We placed 5 different overlays on 2 substrate topologies using both random placement and the optimized placement (described later in this section), thus ending up with 20 different overlay placements. For each of these 20 placements, we first calculated q, and then generated 5 random faults and averaged the diagnosis cost for these 5 faults. The relationship between q and average diagnosis cost for these 20 distinct networks showed that diagnosis cost decreases with increasing q, as intended, which suggests that maximizing q in order to reduce diagnosis cost is a suitable approach.

Diagnosability and robustness

Increased sharing between the paths of an overlay network may appear conflicting with the robustness of the system. We envision multi-overlay systems where there will have to be significant sharing between different overlays, and we are organizing the sharing *within* each overlay to ensure that it becomes more diagnosable. In many cases, our approach does not increase the number of paths (across all overlays) affected

by a failure in the substrate. We illustrate this point with an example in Section 3.5.

The goal of the placement algorithm is to maximize the placement quality, q. We start with a random placement of the overlay onto the substrate, and gradually improve q by taking small steps. At each step, we take one overlay path whose placement may be diminishing q and move its end nodes to other substrate nodes randomly. We describe the algorithm in Algorithm 1. Information about which overlay node pairs are able to provide useful measurements for diagnosis is considered in the calculation of q and the placement algorithm. Only covered overlay paths (as defined in Section 3.2) are taken into account in the effort to optimize the placement, other node pairs are ignored after the initial random placement.

Algorithm 1 Overlay Placement Algorithm

```
Do a random placement.
Calculate q_random for this placement.
q_max ← q_random
BestMapping ← the current mapping of the overlay onto the substrate.
while q has improved within the last stopCounter steps do
    Select a path from one of the three groups (described below) and move its end nodes randomly.
    Use the shortest path algorithm to update all overlay links containing these overlay nodes.
    Calculate q for this new placement.
    if q_current > q_max then
        q_max ← q_current
        BestMapping ← the current mapping of the overlay.
    end if
end while
Output q_max and BestTopo.
```

Doing a random placement has two stages. First, all overlay nodes are mapped one-to-one randomly to substrate nodes. Then the path between the end nodes in each overlay link is calculated using a shortest-path algorithm. We compute the initial quality q for this random placement, and mark this mapping as the best. The main step of the algorithm is moving paths that decrease q. The three groups of such paths mentioned in the algorithm description are explained below.

Group 1: Paths whose links have an average p below half the overall average p of the overlay. They reduce the average amount of sharing.

Group 2: Paths containing a component with stress $t > T$. These paths cause the component's stress to exceed T, resulting in unacceptable quality.

Group 3: Paths that share all but one of their substrate links with another path. These are placed inefficiently and may provide more valuable diagnosis information if placed elsewhere.

By moving paths in group 1, we aim to increase average sharing, while moving paths in groups 2 and 3 helps ensure that m stays below the threshold and o stays low. The combination of these moves improves q toward a maximum, which is our goal in modifying the placement. If q does not improve in *stopCounter* consecutive steps, we stop the algorithm. For *stopCounter*, we use a value of $10g$, where g is the total number of paths in all 3 groups. This ensures that each of these paths will have an average of 10 chances to move before the algorithm is allowed to stop for lack of improvement.

3.4 Collaborative Diagnosis Using Shared Observation

We now describe our method for generating the suspect set S defined in Section 3.2. There are many possible approaches to this problem, relying on active probing [14,93], passive monitoring [9, 26] or event-based diagnosis. We use the *user-driven collaborative diagnosis* technique presented in the work by Tang and El-Shaer [82]. Collaborative diagnosis is a lightweight approach that requires no monitoring sensors or active measurements. It is a passive approach relying only on end-to-end observations called *evidences* to diagnose network problems. End-user applications share their negative evidences (bad symptoms such as unreachability or high loss) to identify the problematic components along a path. In case of low user participation, insufficient

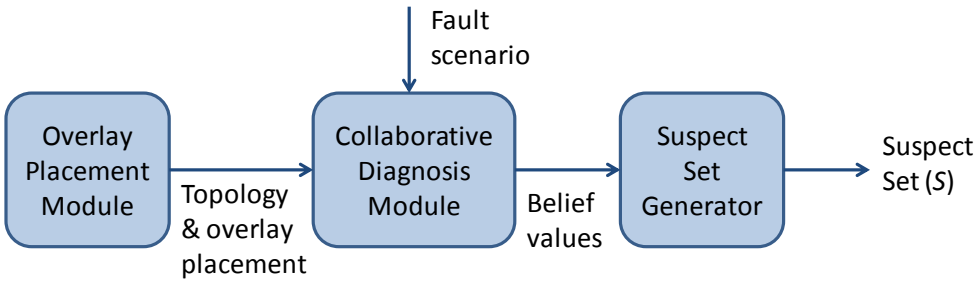


Figure 6: Interaction between collaborative diagnosis and suspect set generator

traffic or misbehaving sources, *uncertainty* in diagnosis may arise. To handle this issue, the collaborative diagnosis technique utilizes *evidential plausible reasoning* using Dempster-Shafer theory [75] to produce belief values. For details about collaborative diagnosis, readers may refer to the original work [82].

An end-to-end path needs to carry a certain amount of traffic to be able to produce evidences when faults occur along that path. A *covered* path is one that has sufficient traffic from its source to its destination to produce a negative evidence in case of the failure of a component on that path. The *coverage* of the overlay is the percentage of overlay-node pairs that have a covered path between their source and destination.

We provide the collaborative diagnosis system with the substrate topology and the placement of the overlay on the substrate. In case of faults in the network, overlay nodes that observe problems report evidences. The diagnosis system collects the evidences and utilizes them to produce *belief values*, one for each substrate component, indicating the belief that the component is faulty. From this set of values, we generate S, the set of components suspected to be the root causes of the negative end-to-end evidences. Figure 6 demonstrates how we use the diagnosis system to generate S.

Construction of suspect set S from beliefs

This is a critical step that allows a great deal of flexibility. A computationally inexpensive way of doing this is to set a *minimum belief threshold*, and include in S

only those components with beliefs equal to or higher than this threshold. We define this threshold with respect to the maximum belief produced by the diagnosis for any component. We define $\Gamma \in [0, 1]$ as an auxiliary parameter to determine the threshold. For example, if Γ is set at 0.5, only those components with a belief of at least half of the maximum belief are placed in S. More formally, if the maximum among all belief values is bel_{max}, then

$$S = \{c | belief(c) \geq \Gamma \times bel_{max}\}. \tag{6}$$

The choice of Γ has a significant effect on the amount of priority placed on the accuracy and the efficiency of diagnosis. At one end of the spectrum, a Γ infinitely close to 0 would put every component with a nonzero belief into S, signifying an extreme importance given to diagnosis accuracy. As we increase Γ, S gets smaller, increasing the efficiency of diagnosis, but on the other hand also increasing the probability of false negatives and the potential for inaccuracy.

Active probing supplement

One issue with purely passive fault diagnosis is that some end-to-end paths may not have sufficient traffic to indicate potential failures of components along them. More such paths would mean fewer evidences per faulty component, which could adversely affect diagnosability. In this case, limited active probing can be used in addition to passive measurements to reach a better level of diagnosability. We only consider adding active probes to paths that are not covered. The location and number of these active probes depend on the target level of diagnosability, i.e., an upper limit on the diagnosis cost as defined in Section 3.2. More details about active probe placement are discussed at the end of Section 3.5.

3.5 Evaluation of Overlay Placement Algorithms

In this section, we analyze the performance of the overlay placement algorithm presented in Section 3.3 and understand its effects on diagnosability. We consider diagnosability in terms of the cost, accuracy and efficiency definitions we gave in Section 3.2. For the diagnosis, we implement an emulator following the diagnosis framework given in Section 3.4. The emulator takes as input a substrate topology, an overlay topology, the mapping between the overlay and the substrate, and a fault structure. The fault structure tells the emulator to generate faults either randomly or at specific locations in the network. As output, the emulator produces a table containing the negative belief of every substrate component in that specific fault scenario. From this table, we generate the set of components suspected to be faulty, which we defined earlier as the suspect set S. This set is generated according to the procedure described in Section 3.4, by using Γ to tune the priority given to accuracy and efficiency in diagnosis.

3.5.1 Metrics

We use a combination of metrics to assess the diagnosability of a network after examining the results produced by the diagnosis emulator.

- *Accuracy:* Any occurrence where the faulty component set is not placed within S by the emulator is a false negative, which means at least one faulty component effectively escaped diagnosis. The accuracy is defined in Section 3.2 in terms of the probability of false negatives, and here we use false negative rate (i.e., fraction of cases where we come across false negatives) as the measure for accuracy.

- *Efficiency:* As $|S|$ grows, it becomes more difficult to localize the faults, so a smaller S is better for the efficiency of diagnosis. We use $|S|$ as a measure of efficiency.

- *Potential diagnosis cost:* The diagnosis cost metric, a measure of the effort required for fault localization, as defined in Section 3.2 is a combination of accuracy (i.e., lack of false negatives) and efficiency (lack of potential false positives).

3.5.2 Strategy

In the remainder of this section, we compare the performance of our overlay placement heuristic with random placement, and the *stress-balancing* placement presented in [96]. We experiment with two substrate topologies, both real autonomous system (AS) topologies taken from Rocketfuel data [70]. An overlay topology is constructed by first setting the number of overlay nodes at about 20% of the number of substrate nodes, and then connecting each pair of overlay nodes with a 50% probability. Paths between connected overlay nodes are calculated using the shortest-path algorithm. Overlay nodes are selected from the substrate randomly in the random placement algorithm, and we try 10 random placements for each experiment and average the results of these 10 placements. The quality-driven placement heuristic, on the other hand, produces a specific placement in each experiment.

Placements are fed into the diagnosis emulator along with the topology and a fault scenario to generate *fault belief* values for all substrate components. These values are then used to construct the suspect set S for this particular fault scenario. The metrics for evaluation are computed for each fault scenario, and then averaged over all fault scenarios to find the values for a certain placement. We follow the above strategy assuming full (100%) coverage at first. Later, we repeat the same steps with varying degrees of coverage and discuss the results at the end of this section.

3.5.3 Results

Choice of Γ

We run the first experiment to assess the effect of Γ (defined in Section 3.4) on the accuracy and the efficiency of diagnosis. We try out a number of different fault scenarios in 3 overlay topologies placed over each of the 2 substrate topologies. For this experiment, each fault scenario contains a single faulty component, and every substrate component is chosen as faulty once over the course of the experiment. The set of all valid fault scenarios contains only those where the faulty component lies within the overlay (since our goal is to maximize the diagnosability of the overlay, we do not concern ourselves with faults that may occur at other parts of the substrate). Each (substrate topology, overlay topology, fault scenario) combination is given as input to the emulator, which then produces results for different values of Γ, which is varied from 0.5 to 1 with 0.05 increments. Values of Γ below 0.5 are not considered because we have observed that 0.5 is as good a value for Γ as 0 in terms of avoiding false negatives, so exploring the lower half of the spectrum is superfluous.

Figure 7 presents the false negative rates in diagnosis and the size of the suspect set S ($|S|$) with varying Γ. The solid lines are random placement results, and the dotted lines are optimized placement results. False negatives do not occur until the 0.9 mark for the optimized placement, while $|S|$ diminishes consistently as Γ is increased. We observe that the optimized placement heuristic provides a reduction in both $|S|$ and the false negative rate compared to the random placement. As Γ gets closer to 1, this reduction increases percentagewise.

Figure 8 illustrates how the diagnosis cost is affected by Γ. The diagnosis cost displays a similar behavior to $|S|$ and the false negative rate in that the heuristic achieves a considerable reduction in cost, and the savings are more pronounced for larger Γ values. We also consider the performance of a stress-balancing overlay placement

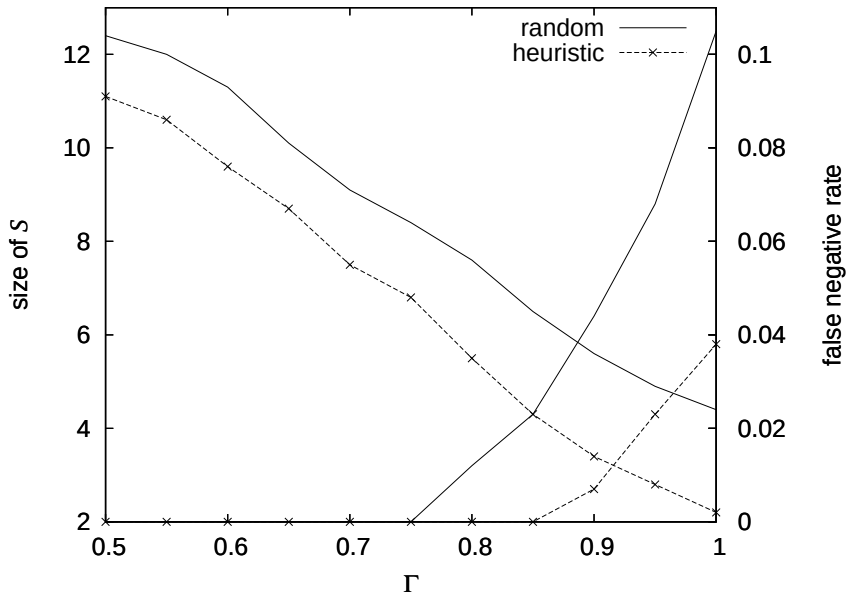


Figure 7: The effect of Γ on $|S|$ and false negatives

algorithm [96]: On average, it offers a slight improvement in cost compared to random placement, possibly thanks to its ability to avoid highly unbalanced placements where some paths have good diagnosability but others have terrible diagnosability. However, stress-balancing placement cannot approach the improvement provided by the optimized heuristic. For the purposes of this evaluation, we consider 0.9 as the optimal value for Γ because it provides the minimum cost among all the values we tried. Hence, instead of evaluating the algorithms with a range of Γ, we use a fixed Γ of 0.9 in the rest of this evaluation section.

Multiple faults

We present an experiment that considers the case where a fault scenario consists of two components. The emulator produces belief values for every two-component combination in the substrate. With Γ fixed at 0.9, we only include in S the pairs whose beliefs are at least 90% of the maximum observed belief. In Table 2 we list

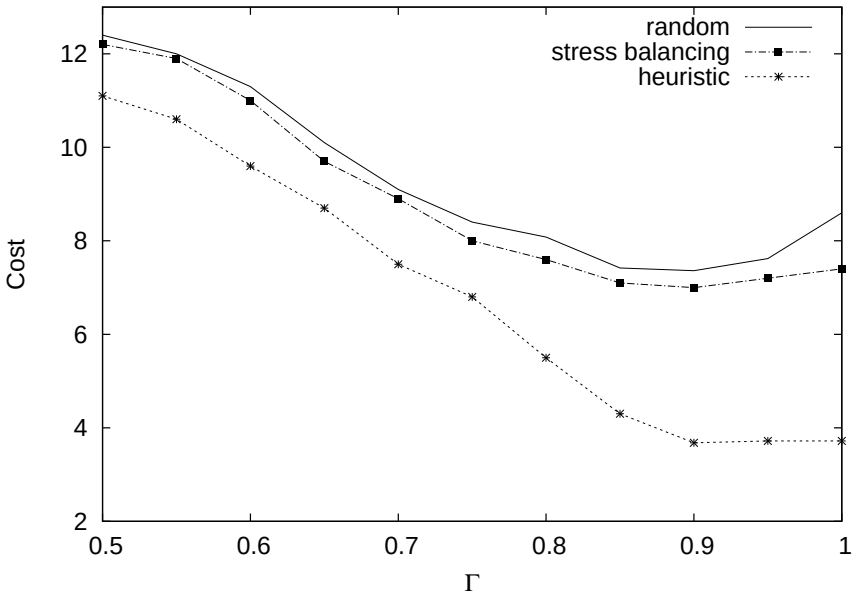


Figure 8: The effect of Γ on diagnosis cost

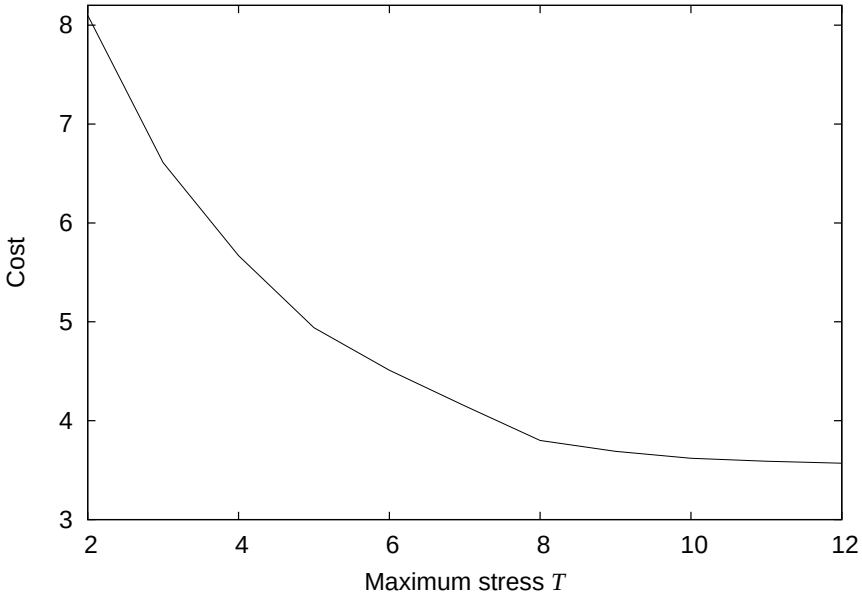


Figure 9: The effect of varying T on diagnosis cost

Table 2: Two faulty components

	Random placement	Placement for diagnosability
False neg.	0.056	0.013
$\|S\|$	6.42	4.08
Diag. Cost	9.1	4.7

the false negative rate, $|S|$ and Cost values for both the random placement and the heuristic. We observe the same kind of improvement in these metrics as we did with the single fault scenarios. One difference is that the false negative rate is slightly higher compared to the single fault case at a Γ value of 0.9. This may be due to a few extra cases where S failed to contain one of the two faulty components. As we have more simultaneous faults, false negatives may increase because S has to contain *all* faulty components to be considered accurate, and each additional fault introduces more risk of inaccuracy.

Choice of T

T is the upper limit for the number of paths that can share a component. It is a constraint introduced to control the risk of congestion or avoid inefficiency in throughput that can be achieved along overlay paths. Varying T in different topologies helps us assess its effect on diagnosability and the performance of the placement algorithm.

We run the algorithm with T values ranging from 2 to 12 and plot the diagnosis cost against T in Figure 9. We observe that as T increases, the cost decreases almost steadily until T hits about 8. This is because loosening a constraint extends the solution space for the algorithm and gives way to more *compact* overlay placements that have a higher amount of sharing, which is desirable for diagnosability. After this point, the decrease in cost slows down and the cost stays nearly constant after a certain point. While the initial reduction can be attributed to the algorithm acting more freely and finding more efficient solutions, the subsequent lack of improvement is interesting. As the control over sharing is removed gradually and the overlay

is allowed to become more *crammed* with p (paths per link) now being the most important variable in the placement, negative evidences increase not only for the faulty component, but also for non-faulty components along the same paths as the faulty one. This makes it harder to distinguish the faulty component from the others, as many other components are able to exceed the threshold set at 90% of the maximum belief, and $|S|$ cannot be decreased much further. With no or very few false negatives, $|S|$ is the determining factor for cost, which follows the behavior of $|S|$. Hence, it is important to find the ideal point for the value of T for efficient diagnosis, which is around 8 for this set of experiments. A good choice of T allows us to reduce the size of S and cost as much as possible without allowing unreasonably high and crippling sharing.

Effect of coverage and active probing

The final piece of the evaluation analyzes how varying coverage affects diagnosis cost, and how addition of active probes can improve diagnosability when coverage is low. Figure 10 shows how diagnosis cost decreases with increasing coverage. We see that complete coverage may not be essential for near-optimal diagnosis cost; however, the cost increases more rapidly after the coverage falls below roughly 60%.

As mentioned in Section 3.4, the deterioration of diagnosability due to decreasing coverage can be alleviated through the addition of active probes. It is possible to shrink the size of the suspect set S by placing active probes in a way that will eliminate elements from S. To this end, we find a non-covered overlay path that contains $|S|/2$ (or closest to it) components that are in S and add an active probe on that path. This either solidifies each component's position in S (if the active probe detects a fault) or removes it from S (if the probe goes through without a problem). This can be repeated until lowering the diagnosis cost to the desired level. We have observed that adding two active probes in this way yields a significant reduction (about 50%)

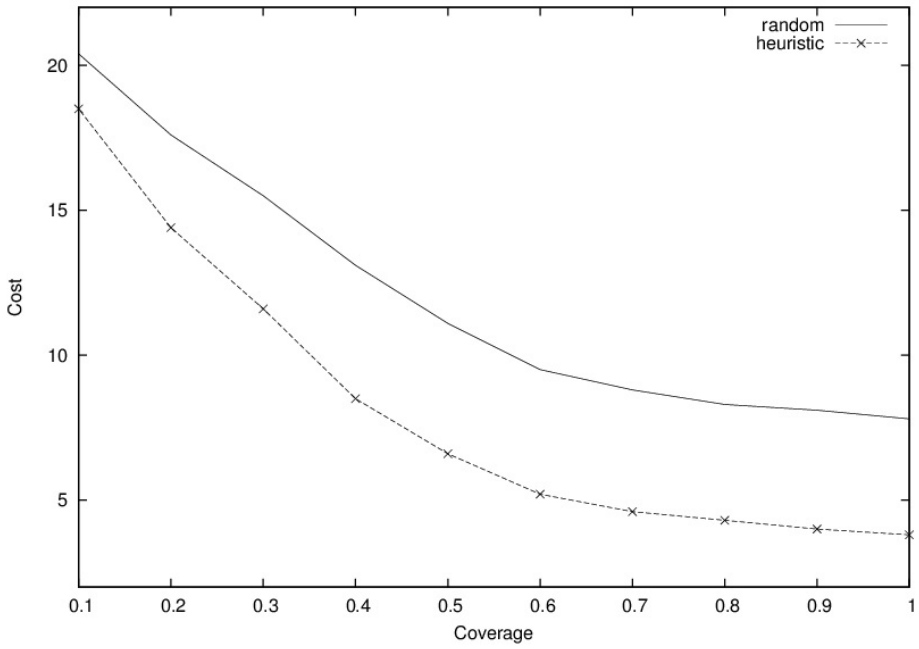


Figure 10: The effect of varying coverage on diagnosis cost

in diagnosis cost for low coverage (20%-30%), while more than three active probes offer negligible marginal utility.

Diagnosability vs. robustness

Finally, we look at the potential number of overlay paths affected by a fault in the substrate, when overlays are placed using either our placement heuristic and the stress-balancing algorithm in [96]. Table 3 shows the average number of overlay paths per substrate link for both algorithms in a Rocketfuel topology. When there are three overlays, the placement heuristic causes a higher number of paths per link, though the average is not very high. When the number of overlays is ten, the numbers come closer to each other, which shows that there is virtually no difference in the number of end-to-end paths affected in case of a potential fault for the two algorithms. This result illustrates that our placement heuristic does not cause any significant disadvantage in this possible measure of robustness.

Table 3: Paths affected by a potential fault

	Placement for diagnosability	Stress-balancing
3 overlays	2.56	1.92
10 overlays	5.68	5.42

3.6 Summary

In this work, we have leveraged the flexibility inherent in overlay network design into improved diagnosability. We identified diagnosability as a sensible design goal in well-provisioned networks, and gave a practical definition of diagnosability by separating it into accuracy and efficiency of diagnosis. We then presented a method to perform overlay placement with the goal of maximizing the diagnosability of the overlay by increasing meaningful sharing among overlay links in a controlled manner. Using a passive monitoring based fault diagnosis scheme, we show that our placement algorithm improves diagnosability compared to an overlay constructed without regard to its diagnosability. Furthermore, we studied how diagnosis cost is affected by the lack of useful end-to-end traffic, and argued that this problem can be alleviated through the use of limited selective active probing. Finally, we explored whether there is a significant tradeoff between robustness and placement for diagnosability. We concluded that while this tradeoff may exist for networks where overlay density is low, it becomes less significant as the number and/or sizes of overlays increase because highly shared substrate components become less avoidable.

The main takeaway from this chapter is that design for diagnosability is a feasible goal for overlays and solutions can be provided without significantly deteriorating the performance and robustness of the network, especially if the network is well-provisioned.

CHAPTER IV

FAIR ALLOCATION OF SUBSTRATE RESOURCES AMONG MULTIPLE OVERLAY NETWORKS

4.1 Introduction

Many overlay networks offering different services may be placed on the same underlying substrate network and compete for resources like node CPU time or link bandwidth. As the number of overlay networks placed on the substrate and their resource demands increase, the substrate network may not be able to meet the demands of all overlays. Hence the necessity arises to allocate limited substrate resources fairly and efficiently among all competing overlays.

One of the primary resources that must be distributed fairly is link bandwidth. We consider scenarios where the bandwidth demands of the overlay networks constitute a significant share of the load on the substrate, and substrate links are not able to fully meet the demands from all overlays simultaneously. This situation may occur if the amount of available bandwidth in the substrate is low, or if the number of overlays and/or their bandwidth demands are too high for the substrate to satisfy. Hence, in such a situation, fair allocation of available bandwidth among overlays must be treated as a priority in order to avoid starving some overlays while trying to satisfy others.

Each substrate link in the network has a certain available capacity for the overlay traffic it can carry, and this capacity must be divided in a fair and efficient manner among the overlay flows sharing that link. Furthermore, the allocation of bandwidth throughout the whole substrate network must be balanced among the overlays placed on it. Fair resource allocation to flows in a network is a classic research question that

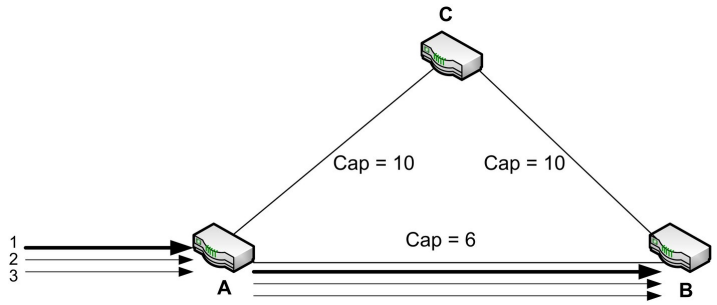
has been studied extensively in the past. The main distinction of this work is the consideration of a virtual network or overlay containing multiple flows as the unit subject to allocation.

A critical question is what constitutes a fair allocation of bandwidth among multiple overlays. A common fairness definition is *max-min fairness*. Another simple definition is Jain's fairness index [42] which defines uniform rates throughout the set of all flows as perfect fairness. We explore other possible fairness definitions in Section 4.4.

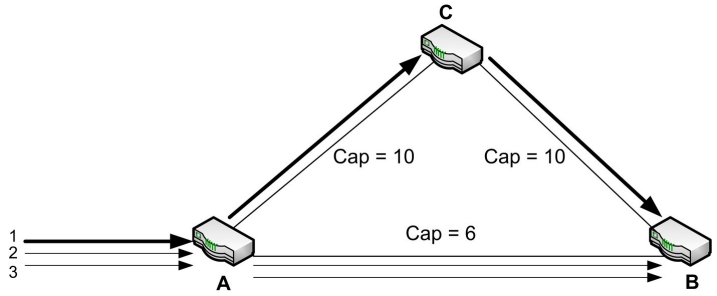
Another important question is how routing in the substrate network affects fair allocation among overlay networks. Deviating from simple minimum-hop routing for overlay networks can facilitate achieving higher rates as well as better fairness in bandwidth allocation. We also explore this question and describe a heuristic method to fine-tune routing in Section 4.5.2.

We present a small example to illustrate the effects of the fairness definition and the routing on rate allocation. Consider the scenario in Figure 11 where three flows belonging to two different networks arrive at Router A and they are all destined for Router B. Assume that all flows have a demand of 10 units per second and rate allocation is performed using max-min fair sharing. Link capacities are 10 units per second for AC and CB, and 6 units per second for AB.

Flow 1 belongs to overlay network 1, flows 2 and 3 belong to overlay network 2. If all three flows are routed on AB as seen in Figure 11a, each of them will receive 2 units of bandwidth. This allocation is perfectly fair when we consider the flows individually. However, if we want to define perfect fairness as equal total rates for all sharing *overlays*, then a fairer allocation would be to give 3 units to flow 1, and 1.5 units each to flows 2 and 3. How do we decide which allocation is better? In Section 4.3 we formulate a fairness metric for allocations involving multiple bandwidth-sharing overlay networks.



(a) Three flows from two overlays sharing AB.



(b) Flow 1 is moved to the path AC-CB.

Figure 11: Three flows from two different overlays are competing for link bandwidth. Thick lines indicate a flow belonging to Overlay 1, thin lines are flows from Overlay 2.

In Figure 11a, the total rate allocated to the three flows is 6 units. However, with a small routing change a better rate can be achieved. Consider the case shown in Figure 11b where we route flow 1 over AC-CB instead of AB. This increases the path length from one hop to two hops, but flow 1 is now able to receive 10 units while flows 2 and 3 both receive 3 units, and the total bandwidth allocated to the flows becomes 16 units. We see that deviating from shortest-path routing in the substrate can lead to a higher utilization of substrate link bandwidth and a more satisfactory allocation for the overlays competing for it.

The contributions of this work are:

- A novel fairness definition for scenarios that involve sharing between multiple overlay networks,
- Recognizing the applicability of multipoint-to-point session fairness definitions

and algorithms given in [46] to the multiple overlay setting,

- Substrate routing optimization techniques to improve the satisfaction of overlays.

The rest of this chapter is organized as follows: We explain the network model and problem statement in Section 4.2, and propose a metric to evaluate the fairness of allocations in the presence of multiple overlays in Section 4.3. Section 4.4 gives fairness definitions adapted from the literature and algorithms to achieve the properties specified by these definitions. We examine the issue of routing in relation to fair resource allocation and propose methods to improve substrate utilization in Section 4.5. We evaluate our methods in Section 4.6. We summarize the chapter in Section 4.7.

4.2 Model and Problem Statement

We consider n overlay networks placed on a substrate network. We model the substrate network as a directed graph $G = (V, E)$, where V is the set of vertices and E is the set of directed edges connecting these vertices. We model the i^{th} overlay network as a directed graph $G^i = (V^i, E^i)$, with $V^i \subseteq V$ being the set of overlay nodes and E^i being the set of overlay links. Let $l = \sum_{i=1}^{n} |E^i|$. That is, l is the total number of overlay links from all overlay networks sharing the substrate network.

We assume an environment where multiple overlay networks are competing for substrate network resources. We are focusing on bandwidth as the contested resource. In our model, each overlay link represents a traffic flow from its source to its destination, and each flow has a certain bandwidth demand from the underlying substrate network. Let F denote the set of all flows from all overlays placed on the substrate. For a substrate link e, let F_e be the set of overlay flows that demand bandwidth from e. Note that $|F_e| \leq l$.

The following information is given:

- The traffic demand matrix M_1, M_2, ..., M_n for each of the n overlay networks
- The capacity of each substrate link in the network
- The connectivity information for the network

In addition to the above, one of the following two may be given:

1. Routing information, i.e., the substrate path for all overlay links
2. The algorithm for assigning rates to the overlay flows

If routing is given: In this case the goal is to decide on an allocation algorithm that will yield a fair assignment of rates to the overlay flows in the system. The allocation algorithm will then calculate how much bandwidth should be given to each overlay flow on each of the $|E|$ substrate links in the network. More formally, for each $e \in E$ with capacity Cap_e, we must decide how to divide Cap_e among the overlay links in F_e. Ultimately, the allocation can be represented in a $|E| \times l$ matrix where each row represents the bandwidth allocation for the corresponding substrate link. Note that the row for substrate link e would contain zero values under columns representing overlay links that are not in F_e.

In this scenario, the sum of all rates in a feasible bandwidth allocation has an upper limit that depends on the capacity of the substrate links appearing in the routes. An allocation is *feasible* if no link's capacity is exceeded when it is applied. The upper limit can be found using the following optimization.

$$\begin{aligned} &\text{maximize} \sum_{f \in F} rate_f \\ \text{subject to} \quad & \\ &\forall e \in E \sum_{f \in F_e} rate_f \leq Cap_e \end{aligned} \tag{7}$$

In the above optimization, $rate_f$ denotes the rate assigned to flow f. Since there is no fairness constraint on this optimization, we do not expect the resulting allocation to

be fair. This optimization may tend to assign high rates to a few flows and completely starve the others in an effort to maximize the aggregate rate achieved.

To achieve fairness, we must first have a clear definition of what fair sharing means in the described setting. The method of fair resource allocation will depend on this definition. Various fairness definitions are discussed in Section 4.4.

If the allocation algorithm is given: Given the ability to select the routes that overlay flows will follow from their sources to their destinations, and depending on the network topology and substrate link capacities, we may be able to increase assigned rates and improve fairness compared to a given routing. In this scenario, our goal is to select the substrate routes for overlay flows in such a way that will maximize the total bandwidth allocation. Details of this problem are discussed in Section 4.5.

4.3 Evaluating Fairness

In this section, we propose a new metric for evaluating the fairness of a bandwidth allocation in a multi-overlay setting. Other fairness metrics have been proposed before, one commonly used metric is Jain's fairness index [42]. Jain's index is an alternative to metrics like standard deviation or min-to-max ratio for evaluating the amount of variation among a set of values. In the context of bandwidth allocation, each value is the ratio of the actual rate of a traffic flow to its *fair* rate. Knowledge of the values that represent perfect fairness is assumed. We do not assume knowledge of a universally fair allocation of rates across all overlays, but we evaluate each overlay in comparison to other overlays overlapping with it. Furthermore, we do not use individual flow rates in the evaluation, instead we calculate a network-wide value for each overlay representing its satisfaction with the allocation.

The basic idea behind our metric is that each overlay network in the system would be entitled to a certain amount of bandwidth if it was the sole requester of bandwidth, and it may receive less than this amount in the presence of sharing. In an ideally fair

allocation, sharing would cause the total rate of every overlay to be decreased by the same ratio.

We compute the rate allocated to each flow in the presence of multiple sharing overlays, and compare these rates to the rates that would be achieved if each overlay was alone in the system. We then compare the average rate reduction for each overlay's flows with the reduction experienced by other overlays' flows that are sharing with that overlay. The result of this comparison gives us an idea about how the overlay is being treated. A more formal and detailed description is given below.

Let N^i denote the i^{th} network in the system and F^i denote the set of flows belonging to N^i. For $f \in F^i$, we denote the rate allocated to f by r_f. We define the *isolated fair rate* for f as the rate it would receive *if* the network it belongs to (in this case N^i) were the only overlay in the system. The isolated fair rate for $f \in F^i$ is denoted by h_f and calculated using max-min fair rate allocation assuming N^i is the only overlay placed on the substrate. Hence, h_f is the ideal rate for f that is achieved when f is sharing bandwidth only with flows in its own overlay (N^i) and it is used as a benchmark to evaluate the satisfaction of f in the presence of inter-overlay sharing.

The ratio of r_f to h_f can constitute a measure of satisfaction for this particular *flow* in the presence of sharing with flows from other overlays. A more meaningful measure for the *overlay* N^i is the *average isolated satisfaction* (S_i) of all its flows.

$$S_i = \frac{\sum_{f \in F^i} min(\frac{r_f}{h_f}, 1)}{|F^i|} \tag{8}$$

The above metric by itself cannot describe an overlay network's satisfaction by a certain allocation when the overlay is sharing bandwidth with other overlays. We have to take into account the rates allocated to the flows that belong to other overlays sharing with this overlay. For N^i, we define the *unified flow set* ($\widehat{F}^i$) as the union of F^i, and the set of flows that

- belong to overlay networks other than N^i,
- share one or more links with at least one flow in F^i,
- are bottlenecked at a link that is shared between the two overlays OR not bottlenecked at all.

More formally,

$\widehat{F}^i = F^i \cup \{g : g \in F^j$ s.t. $i \neq j$, and $\exists f \in F^i$ s.t. g shares at least one link with f AND if g has a bottleneck link e, f also traverses e.$\}$

We calculate the *average unified satisfaction* ($\widehat{S}_i$) for N^i as follows:

$$\widehat{S}_i = \frac{\sum_{f \in \widehat{F}^i} min(\frac{r_f}{h_f}, 1)}{|\widehat{F}^i|} \tag{9}$$

For an overlay network, we define the *network satisfaction* ($NetSat_i$) for N^i as the ratio of its average isolated satisfaction to its average unified satisfaction.

$$NetSat_i = \frac{S_i}{\widehat{S}_i} \tag{10}$$

The above metric can give us an idea about how an overlay is being treated in terms of bandwidth allocation. In a system where all flows in all overlays are able to receive their isolated fair rates, each overlay would have a NetSat of 1. In the presence of bottlenecks, a higher NetSat would indicate a more favorable treatment toward the overlay. However, the NetSat value cannot fully assess the fairness of a rate allocation. We need a metric to evaluate the fairness of an allocation across all the overlays in the system. NetSat is a rough measure of how an individual overlay is being treated, and we would like to compare all the NetSat values to evaluate the amount of variation among the satisfactions of the overlays. To this end, we define

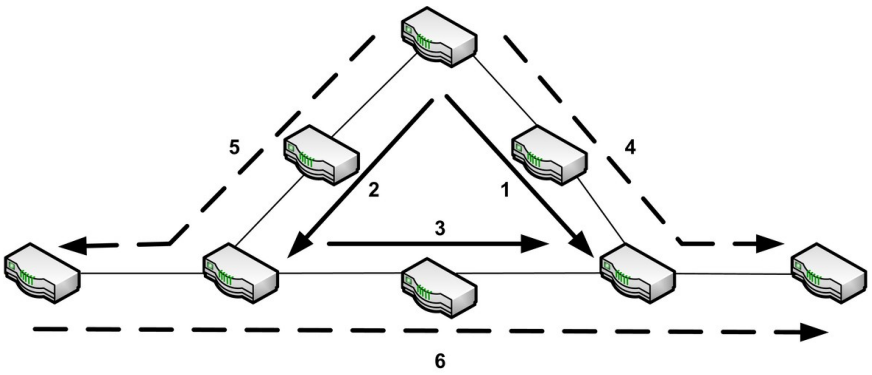


Figure 12: Two overlays sharing

the *global fairness index* (GFI) of a system as the *population standard deviation* of the NetSat values of the n overlays in it.

$$\mu = \frac{1}{n}\sum_{i=1}^{n} NetSat_i \tag{11}$$

$$\text{GFI} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(NetSat_i - \mu)^2} \tag{12}$$

A smaller number, i.e., a number closer to zero, for the global fairness index would indicate a fairer allocation.

We present a numerical example to illustrate the calculation of the metric. Figure 12 shows two overlay networks placed on the substrate: flows 1, 2, and 3 (solid lines) belong to overlay A; and flows 4, 5, and 6 (dashed lines) belong to overlay B. Assume each substrate link is bidirectional, and all flow demands are also 1 unit. The leftmost and rightmost links have a capacity of 2 units each, so they do not play a role in the allocation as all flows are bottlenecked at the middle links.

Suppose overlay A was the only overlay on this substrate network. In that case, all of its flows would receive 1 unit. Similarly, all the flows in overlay B would receive 1 unit or bandwidth if B was alone on the substrate. Hence, the isolated fair rates for all flows are 1.

We consider two sample allocations. Allocation 1 gives all flows a rate of 0.5 units. Allocation 2 gives 0.4 units to flows in overlay A, and 0.6 units to flows in overlay B. Below, we calculate the average isolated satisfaction for each overlay.

In allocation 1:

$$S_A = \frac{min(0.5/1, 1) + min(0.5/1, 1) + min(0.5/1, 1)}{3} = 0.5.$$

$$S_B = \frac{min(0.5/1, 1) + min(0.5/1, 1) + min(0.5/1, 1)}{3} = 0.5.$$

In allocation 2:

$$S_A = \frac{min(0.4/1, 1) + min(0.4/1, 1) + min(0.4/1, 1)}{3} = 0.4.$$

$$S_B = \frac{min(0.6/1, 1) + min(0.6/1, 1) + min(0.6/1, 1)}{3} = 0.6.$$

Next, we calculate the average unified satisfaction of the overlays. For both networks, the unified flow set contains all six flows since all flows are bottlenecked at shared links.

In allocation 1:

$$U_A = \frac{min(0.5/1, 1) \times 6}{6} = 0.5.$$

$$U_B = \frac{min(0.5/1, 1) \times 6}{6} = 0.5.$$

In allocation 2:

$$U_A = \frac{min(0.4/1, 1) \times 3 + min(0.6/1, 1) \times 3}{6} = 0.5.$$

$$U_B = \frac{min(0.6/1, 1) \times 3 + min(0.4/1, 1) \times 3}{6} = 0.5.$$

We next compute the network satisfaction indexes as follows:

In allocation 1:

Table 4: Calculations to obtain the global fairness index for two example allocations

	Allocation 1	Allocation 2
S_A	0.5	0.4
S_B	0.5	0.6
U_A	0.5	0.5
U_B	0.5	0.5
$NetSat_A$	1	0.8
$NetSat_B$	1	1.2
GFI	0	0.2

$NetSat_A = S_A/U_A = 0.5/0.5 = 1$

$NetSat_B = S_B/U_B = 0.5/0.5 = 1$

In allocation 2:

$NetSat_A = S_A/U_A = 0.4/0.5 = 0.8$

$NetSat_B = S_B/U_B = 0.6/0.5 = 1.2$

Finally, we calculate the global fairness index for the entire system.

$\mu_1 = (1+1)/2 = 1$, $\mu_2 = (1.2+0.8)/2 = 1$.

In allocation 1: GFI $= \sqrt{((1-1)^2 + (1-1)^2)/2} = 0$

In allocation 2: GFI $= \sqrt{((0.8-1)^2 + (1.2-1)^2)/2} = 0.2$

We summarize the above calculations in Table 4. The GFI values suggest that allocation 1 is perfectly fair, and they also capture the slight unfairness in allocation 2.

4.4 Fairness Definitions and Algorithms

In this section, we give fairness definitions and algorithms to achieve the fairness criteria associated with these definitions. We start with our max-min fairness definition in the multiple overlay network context, and then present the fairness definitions from [46] in Section 4.4.2 and Section refsection-plnf for completeness. Table 7 demonstrates the notation we used.

Table 5: Notations used

E	Set of substrate links
E^i	Links in the i^{th} overlay network
A	Substrate links that are not saturated
Cap_a	Capacity of link a
$Cap_{a,x}$	Capacity assigned to overlay x on link a
U_a	Used capacity of link a
$U_{a,x}$	Capacity used by overlay x on link a
N	Set of all overlay networks
N^i	i^{th} overlay network
F	Set of all overlay flows
F^i	Flows in the i^{th} overlay
P	Set of flows that are not physically bottlenecked
P_a	Flows in P that span link a
$P_{a,x}$	Flows on link a in P belonging to overlay x
Y	Set of flows that are not virtually bottlenecked
$demand_f$	The bandwidth demand of flow f
$rate_f$	The rate of flow f
$rate'_f$	The rate of flow f in alternate allocation
$totalWeight_i$	The total weight given to the i_{th} overlay network
$weight_f$	The weight assigned to flow f

The definitions and algorithms in [46] can be mapped to our multi-overlay setting: Multipoint-to-point sessions correspond to the overlay networks, and the connections in these sessions correspond to the overlay flows. There is one difference between the two sets of definitions: There is no concept of traffic demand in the multipoint-to-point fairness algorithms. We assume in the multi-overlay algorithms that each overlay flow has a traffic demand associated with it and the algorithms ensure that the assigned rate does not exceed this demand.

4.4.1 Max-Min Fairness

Definition: An allocation of rates is max-min fair if it is feasible and for any other feasible allocation where $rate'_f > rate_f$, there is another flow f' such that $rate_f \geq rate_{f'} > rate'_{f'}$.

That is, every flow has a *bottleneck* link where it has the highest rate and its rate cannot be increased any further without decreasing the rate of some other flow.

The algorithm that achieves max-min fairness is given below. In this algorithm, flows from different overlay networks are not distinguished and all flows are treated as if they belonged to one network.

Algorithm 2 Max-min Fairness Algorithm

$P \leftarrow F$
$A \leftarrow E$
while $P \neq \emptyset$ **do**
  $n_a \leftarrow$ number of non-bottlenecked flows that span link a
  $incr \leftarrow \min(\min_{a \in A}(\frac{Cap_a - U_a}{n_a}), \min_{f \in P}(demand_f - given_f))$
  $\forall f \in P \;\; rate_f += incr.$
  recalculate U_a for all $a \in A$
  $A \leftarrow \{a | Cap_a - U_a > 0\}$
  $P \leftarrow \{f | f$ does not span any saturated link and $given_f < demand_f\}$
end while

4.4.2 Normalized Rate Network Fairness

The max-min fairness algorithm does not address the issue of fairness among different overlay networks. The normalized rate network fairness (NRNF) definition tries to increase the balance in bandwidth allocation at the overlay network level. We give the formal definition below.

Definition: The *normalized rate* of a flow is its actual rate divided by the weight assigned to it. That is, $nrate_f = \frac{rate_f}{weight_f}$.

Definition: An allocation of rates is normalized rate network fair if it is feasible and for any other feasible allocation where $nrate'_f > nrate_f$, there is another flow f' such that $nrate_f \geq nrate_{f'} > nrate'_{f'}$.

In other words, every flow has a *bottleneck* link where it has the highest normalized rate and this rate cannot be increased any further without decreasing the normalized rate of some other flow.

The NRNF algorithm defined below achieves balance between overlay networks by assigning weights to all flows so that the sum of weights of flows belonging to each overlay network will be equal. Each flow receives a rate proportional to its

weight at the substrate links. When each overlay has an equal total weight, overlays with more flows will end up with a lower weight per flow. Hence, flows belonging to smaller overlays will be favored during allocation at the substrate links. This ensures that *network* rates, i.e., the sum of the rates assigned to an overlay's flows, stay more balanced among different overlays. Note that when every overlay has the same number of flows and the same total weight, this allocation is identical to the max-min fair allocation.

The total weight of an overlay can be assigned to its flows in different ways, the simplest being assigning equal weight to all flows within an overlay. That is, if the overlay network N^i has $|F^i|$ flows in it and it is given a total weight of w, each of these flows would be given $w/|F^i|$ as its weight.

Algorithm 3 Normalized Rate Network Fairness Algorithm

$P \leftarrow F$
$A \leftarrow E$
$\forall f \in P \;\; weight_f \leftarrow \frac{totalWeight_i}{|F^i|}$ where f belongs to the i^{th} overlay.
while $P \neq \emptyset$ **do**
  for all $a \in A$ **do**
    $P_a \leftarrow \{f | f \in P$ and f spans substrate link $a\}$
  end for
  $incr \leftarrow \min(\min_{a \in A}(\frac{Cap_a - U_a}{\sum_{f \in P_a} weight_f}), \min_{f \in P}(\frac{demand_f - given_f}{weight_f}))$
  $\forall f \in P \;\; rate_f += incr \times weight_f$.
  recalculate U_a for all $a \in A$
  $A \leftarrow \{a | Cap_a - U_a > 0\}$
  $P \leftarrow \{f | f$ does not span any saturated link and $given_f < demand_f\}$
end while

In short, this algorithm makes sure that the rate assigned to each flow at its bottleneck link will be proportional to its weight. If the sum of flow weights are all equal for different overlay networks, we expect the total bandwidth given to overlays to be balanced. However, the weight assignment can be manipulated in order to treat the overlay networks differently. Assigning a higher weight to an overlay will cause its flows to receive higher rates. We can control the allocation of rates to an extent by varying the weight assignment. We illustrate the use of weight manipulation in

Section 4.6.1.

4.4.3 Per-Link Network Fairness

The idea behind the per-link network fairness definition is to achieve network-level fairness at every substrate link by dividing the link capacity equally among all the networks traversing that link. The capacity assigned to each overlay at a link is then divided equally among the flows belonging to the overlay and traversing the link.

Definition: An allocation of rates is per-link network fair if it is feasible and for any other feasible allocation where $rate'_f > rate_f$ at a certain link e and f belongs to an overlay network N^1, there is another network N^2 which has a flow f' such that f' traverses e and $rate_f \geq rate_{f'} > rate'_{f'}$.

This algorithm divides the capacity of the substrate link equally among all the overlay networks that go over it. A network may not be able to fill up its capacity on this link because its flows may be bottlenecked elsewhere. In that case, other networks are allowed to share the remaining capacity. A flow is said to be *virtually bottlenecked* if the initial capacity assigned to its network is fully consumed. A flow is *physically bottlenecked* when its rate can no longer be increased without decreasing the rate of some other flow that has a lower rate. The algorithm continues until all flows are physically bottlenecked.

4.5 Optimizing Substrate Routing for Improved Rate Allocation

In our original setting, routing information for the overlay networks was given and we were trying to provide fair bandwidth allocation with predetermined paths for all overlay links in the system. But suppose that the overlay links are specified only with a pair of end nodes and we are free to select the paths between these pairs.

Selecting the substrate paths corresponding to the overlay links in the system brings the possibility of achieving higher total rates or better fairness in the allocation

Algorithm 4 Per Link Network Fairness Algorithm

$P \leftarrow F$
$A \leftarrow E$
while $P \neq \emptyset$ **do**
  $Y \leftarrow F$
  $\forall a \in A$:
  $P_a \leftarrow \{f | f \in P$ and f spans $a\}$
  $nf_a \leftarrow |P_a|$
  $N_a \leftarrow \{x | x \in N$ and $\exists f \in P_a$ s.t. f belongs to $x\}$
  $nn_a \leftarrow |N_a|$
  $\forall x \in N_a \ \ Cap_{a,x} \leftarrow \frac{Cap_a - U_a}{nn_a}, P_{a,x} \leftarrow \{f | f \in P_a$ and f belongs to $x\}$
  while $Y \neq \emptyset$ **do**
    $incr \leftarrow \min(\min_{a \in A, x \in N_a}(\frac{Cap_{a,x} - U_{a,x}}{|P_{a,x}|}), \min_{f \in P}(demand_f - given_f))$
    $\forall f \in P \ \ rate_f += incr.$
    $\forall a \in A$:
    1. $\forall x \in N_a$ recalculate $U_{a,x}$.
    2. recalculate U_a.
    $A \leftarrow \{a | Cap_a - U_a > 0\}$
    $P \leftarrow \{f | f$ does not span any saturated link and $given_f < demand_f\}$
    $Y \leftarrow \{r | r \in P$ and the network that r belongs to has not used up its capacity on any link$\}$
  end while
end while

of rates, or both, compared to settling for shortest-path routes. By expanding the set of possible routes for each overlay link, we may be able to avoid situations where multiple flows are bottlenecked at a low bandwidth link.

In this section, we assume that the rate allocation algorithm is given, and we would like to optimize the routing in the substrate to satisfy the demands of as many flows as possible and maximize the total utilization of the substrate bandwidth.

We revisit the concept of isolated fair rates from Section 4.3: Instead of using the rate achieved in the presence of shortest-path routing, we look all the possible routings for the flows in a network and select the routing scenario where the highest total rate for the network is achieved with max-min-fair rate allocation. The rates in this scenario are the isolated fair rates for the flows in the network.

We consider two ways of finding the optimal routing. One is a brute-force method that tries to evaluate as many different routings as possible and selects the best one.

The other is a heuristic method that approximates the brute-force method. We now describe these two methods.

4.5.1 Brute Force Method

The brute force method starts by enumerating all paths between every overlay node pair. It then applies the fair bandwidth allocation algorithm of choice to all the possible path combinations to find the set of paths that would enable the highest total rate. Assume there are n overlay networks sharing the substrate network. E^i is the set of overlay links that belong to the i^{th} overlay network. Each element of E^i is expressed as a source-destination pair (src, dst). The j^{th} element in E^i can be written as $e^i_j = (src^i_j, dst^i_j)$. Let λ^i_j be the number of all the possible paths from src^i_j to dst^i_j.

Algorithm 5 Brute force routing optimization

for $i = 1$ to n **do**
  for $j = 1$ to $|E^i|$ **do**
    Find all possible paths from src^i_j to dst^i_j.
  end for
end for
Enumerate all routing possibilities for the system. There will be $numChoice = \prod_{i=1}^{n} \prod_{j=1}^{|E^i|} \lambda^i_j$ combinations.
for $i = 1$ to $numChoice$ **do**
  Apply allocation algorithm and evaluate the total rate of the resulting allocation.
  Update the best choice if this is the largest total rate so far.
end for

This technique is clearly not scalable because there may be a large number of possible paths between each source-destination pair, and even for tens of flows the number of routing possibilities will be too high to evaluate. We can make this technique more scalable by limiting the number routing possibilities to consider.

4.5.1.1 k shortest paths

Instead of enumerating all paths between a source and a destination, we can put a limit of k on the number of different paths to examine per source-destination

pair. This method also has exponential ($O(k^u)$, where u is the number of all source-destination pairs) complexity, but the number of combinations to consider can be reduced, depending on the choice of k. This technique also ensures that we do not go too far sacrificing path length for marginal gains in total rate because it only considers the k shortest paths for each source-destination pair.

4.5.2 Heuristic Approach

The brute force approach described in the previous section would give the optimal routing for total rate, but it quickly becomes infeasible as the number and sizes of overlay networks grow.

Another approach is to start with minimum hop routing for all overlay networks, run the fair rate allocation algorithm, and then try to improve the total rate step by step by making changes to the routes that overlay links follow. The heuristic method is given in the following algorithm definition.

The main idea behind this heuristic is that substrate links that bottleneck a lot of flows have the biggest effect on the inefficiency of a rate allocation. We find these links and try moving some flows away from them in order to increase the possibility of meeting more flow demands.

In the algorithm definition, D represents the set of *bottlenecked* flows that are reroutable at any point. (We do not consider rerouting the flows whose demands are met.) In the beginning, all flows are reroutable. When a flow is rerouted once, it is removed from D. The reason for is to avoid continuing to reroute a flow many times in different combinations and to make sure that we zero in on a solution without wasting too much time. The algorithm stops when no reroutable flow remains, or when no link bottlenecks any reroutable flow.

In practice, evaluating $k^{|B_e|}$ routings at each round of the algorithm may become too expensive if a lot of flows are being bottlenecked at certain links. For this reason,

Algorithm 6 Heuristic for route selection

Do the rate allocation with shortest path routing.
$D \leftarrow$ all overlay flows that are bottlenecked
repeat
1. Find the substrate link e that bottlenecks the most overlay flows in D. Let B_e be the set of flows bottlenecked at e.
2. Using the k shortest paths for each flow in B_e, evaluate $k^{|B_e|}$ routing scenarios. Remove all flows in B_e from D.
3. Select the routing that results in the highest total rate.

until There are no flows left in D.

we implement an upper limit on the number of routings to consider at each iteration. If $k^{|B_e|}$ exceeds that limit, we sample from those $k^{|B_e|}$ routings at a frequency that will place the number of evaluated routing possibilities roughly at our limit. More specifically, if the limit is lim, we evaluate one routing every $\frac{k^{|B_e|}}{lim}$ possibilities.

We compared the brute force method with the heuristic method using some examples involving a small number of overlays and flows, and verified that for these examples the heuristic produced nearly identical results to the brute force method. As the brute force method quickly becomes infeasible, we use only the heuristic method for larger systems.

4.6 Evaluation

This evaluation section consists of two parts: In Section 4.6.1, we present example allocations to demonstrate some properties of the algorithms given is Section 4.4. In this part, we assume shortest path routing in the substrate. Section 4.6.2 considers the scenario where the allocation algorithm is given, and evaluates the route selection heuristic we described in Section 4.5.2.

4.6.1 Example Allocations with Different Algorithms

The algorithms in Section 4.4 achieve fairness according to different definitions. The max-min fairness algorithm results in a balanced allocation of rates among individual flows, while the other two algorithms yield allocations whereby the total rates assigned

to different overlay networks are more balanced.

We present example rate allocations to illustrate three different scenarios. In the first example, there are multiple overlay networks with varying numbers of flows, and we compare the properties of the max-min fair allocation and the normalized rate network fair (NRNF) allocation algorithms.. In the second example, the number of flows in each overlay is the same, but we would like to treat each overlay differently so we assign them varying total weights using the NRNF algorithm and compare the results with max-min fair allocations. The third example illustrates how the global fairness index values change between allocations that use different weight structures..

Scenario 1

In this example, we place 12 overlay networks on top of the substrate network seen in Figure 20a. The first 4 of these overlays have 6 flows, the second 4 overlays have 12 flows, and the third 4 overlays have 20 flows each. We then perform the rate allocation using both the max-min fair allocation algorithm and the normalized-rate network-fair (NRNF) algorithm. The capacity of each substrate link is 10 units, and the bandwidth demand of each flow is 1 unit. The total weights for all overlays are set to 1 in the NRNF algorithm.

Table 6 presents the results of the example allocations. The first three rows of data contain the average total rate for each 4-overlay group: small (6-flow) overlays, medium (12-flow) overlays, and large (20-flow) overlays. We observe that with the max-min fair allocation, the total rate for networks grow proportionally to the number of flows in them. On the other hand, the NRNF algorithm keeps aggregate rates closer to each other and the total rate of an overlay does not increase proportionally to the number of flows in it.

The fourth and fifth rows of data show the minimum and maximum total rate among all 12 overlays in the system. The NRNF algorithm yields a higher minimum

Table 6: Results of the example allocation to overlays with varying number of flows

	Max-min fair	NRNF
Avg total rate for 6-flow overlays	4.78	6
Avg total rate for 12-flow overlays	9.65	10.60
Avg total rate for 20-flow overlays	15.20	12.87
Minimum total rate	4.10	6
Maximum total rate	17.64	16.47

and a lower maximum than the max-min fair allocation algorithm when total overlay rates are considered.

Scenario 2

We now consider a scenario where the number of flows in each overlay is equal. As we mentioned in Section 4.4.2, when all overlays have an equal number of flows, setting all overlay weights to 1 in the NRNF allocation will result in an allocation identical to that given by the max-min fair allocation. However, in this example we would like to show the effect of assigning different weights to different overlays.

We place 10 overlay networks with 12 flows each on the substrate. We keep the link capacities at 10 units each, but we increase the demand of each flow to 5 units in order to form bottlenecks and observe the variations in the allocations more easily. We perform the weight assignment such that for the i^{th} overlay, the weight is i. Thus we have 10 different weights, 1 through 10, assigned to the overlays. We then allocate rates to these overlays using the NRNF algorithm. For comparison, we also do an allocation using equal weights for each overlay, which is effectively a max-min fair allocation. The next step is to take the ratio of the total rate given to each overlay in the NRNF allocation to the total rate given to the same overlay in the max-min fair allocation. We repeat this procedure with 10 different sets of 10 overlay networks and average the ratios over these 10 sets.

Figure 13 demonstrates how the ratios of the NRNF rates to the max-min fair rates change with the weight assignment. We observe that overlays that have low weights

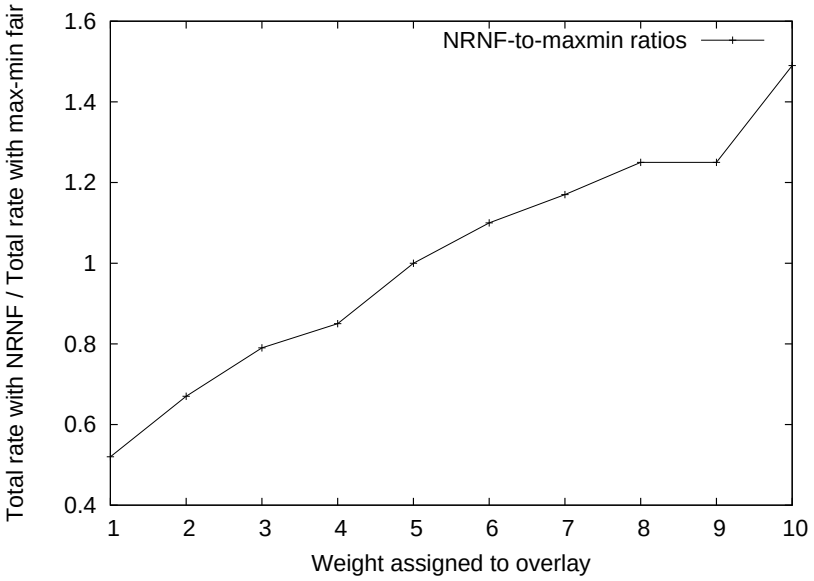


Figure 13: Average ratios of rates from the weighted allocation to max-min fair rates

are treated unfavorably compared to the max-min fair allocation. Conversely, overlays with the highest weights are receiving favorable treatment and achieving higher total rates than they would under max-min fair allocation.

Scenario 3

Finally, we would like to demonstrate how different treatment of overlays is reflected by our global fairness index (GFI) metric from Section 4.3. We use the same substrate and a set of 16 overlay networks, each with 12 flows demanding 1 unit of bandwidth each. First, we assign a total weight of 1 to each overlay and use the NRNF algorithm to perform the rate assignment. Then, at each step, we change the weight of one overlay from 1 to 2, thus favoring it in terms of rate allocation. We plot the number of favored overlays against the GFI values resulting from the corresponding allocation. Figure 14 shows that as we give preferential treatment to more overlays, allocations become less fair, as illustrated by rising GFI values (Low GFI values are better).

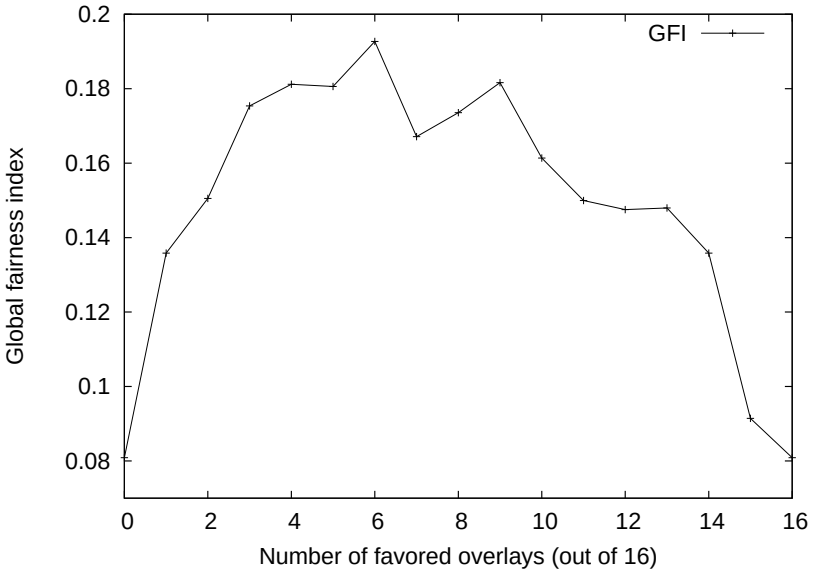


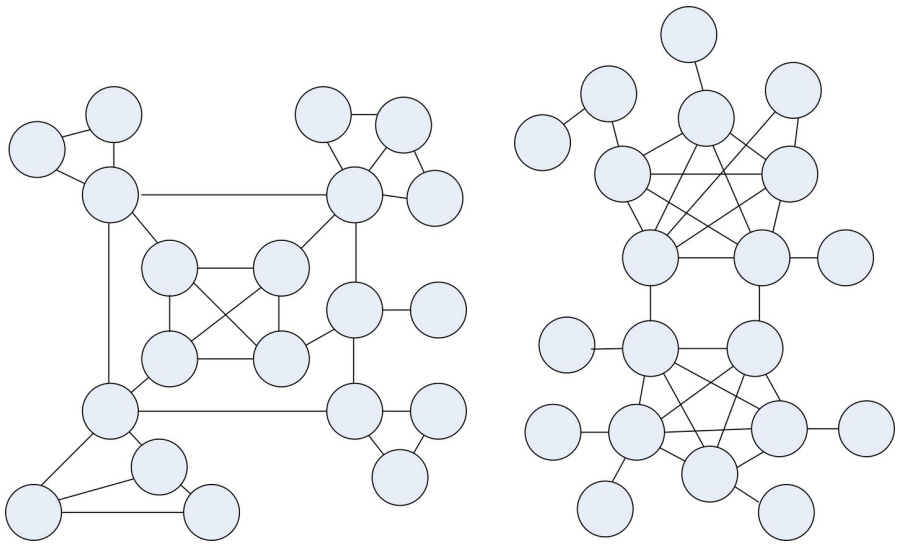
Figure 14: Global fairness index changing with the number of favored overlays

This trend continues until we have doubled the weights of a considerable percentage (30-40%) of the overlays. Doubling the weights of more than half of the overlays naturally reduces the advantage gained by a higher weight and diminishes the effect of favorable treatment for an overlay. Hence, the GFI values start dropping to indicate fairer allocations after the halfway point.

4.6.2 Evaluating the Route Selection Technique

In this section, we evaluate the performance of our route selection heuristic from Section 4.5.2 and compare it to default minimum-hop routing in terms of the total rate achieved and the fairness properties.

We use two different topologies for our evaluation: Topology 1 is a general single-domain topology, and Topology 2 represents two node clusters connected by two inter-cluster links. Figure 15 shows the two network topologies we used. We assume that each substrate link has a capacity of 10 units.



(a) Topology 1 (b) Topology 2

Figure 15: Substrate network topologies

We place n overlay networks on these topologies, where n ranges from 2 to 10. For each value of n, the simulation is repeated three times using different overlays. The number of overlay nodes is set to roughly 20% of the substrate nodes for every overlay. These nodes are connected into a full mesh: every possible pair of overlay nodes represents an overlay link.

We define a traffic flow with a traffic demand of 1 unit from the source to the destination of every overlay node pair. Hence, for an overlay with z nodes, there are $z(z-1)$ flows, each with a demand of 1 unit. For the topologies we used, each overlay network has 4 nodes and 12 flows, one between each node pair.

We first assign a route to each node pair using shortest path routing and perform the rate assignment using the max-min fair algorithm from Section 4.4.1. (Note that the NRNF algorithm with equal overlay weights would yield identical results since every overlay has an equal number of flows.) Then we execute the route improvement heuristic from Section 4.5.2 to find the routing that maximizes the total rate assigned to all flows.

We demonstrate the improvement in the total rate in Figure 16 and Figure 17

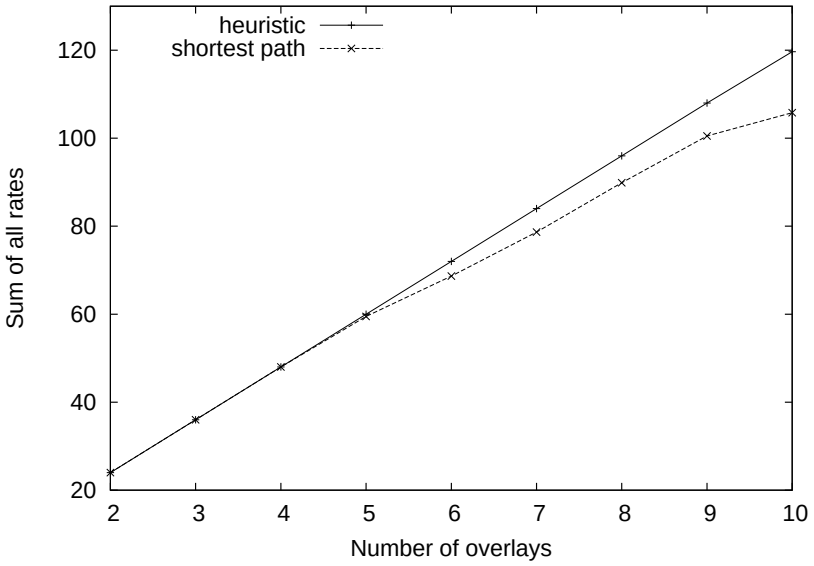


Figure 16: Total rates for Topology 1

for the two topologies. As the number of networks is small, substrate links are not saturated by overlay flows and all flows fully receive what they demand. As we add more overlays, bottlenecks start to occur and some flows have to settle for rates lower than their demands. In these cases, using the route selection heuristic to move some flows to paths different from those suggested by shortest-path routing provides an improvement, and we are able to accommodate a larger number of overlays on the substrate without bottlenecking their flows.

The simplistic scenario where every link has equal bandwidth is rarely the case in real networks. For example, the links that connect the two clusters in Topology 2 may have different capacities. We examine this case by increasing the capacity of one of these links from 10 units to 20 units. As we see in Figure 18, our heuristic is able to achieve a more significant improvement in this case compared to when all capacities are equal. As the variation in link capacities increases, the route selection heuristic may achieve more improvement over shortest path routing, especially if the

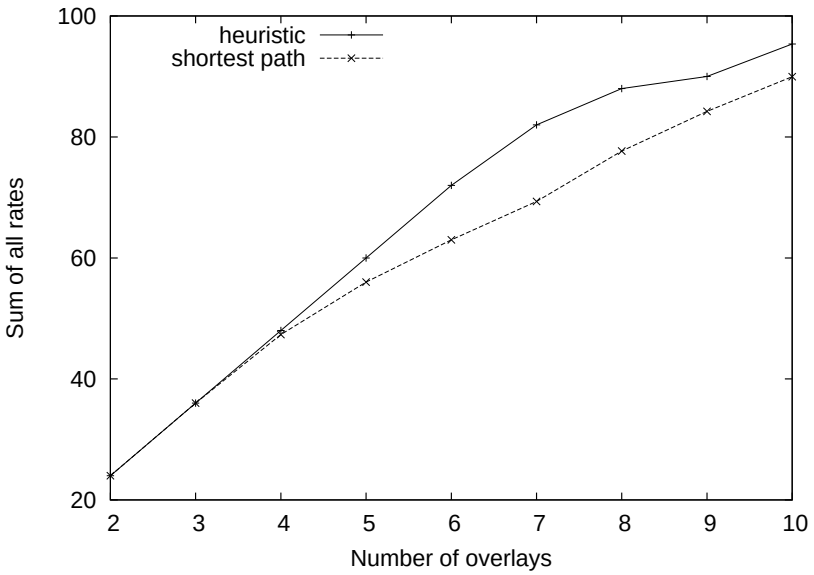


Figure 17: Total rates for Topology 2

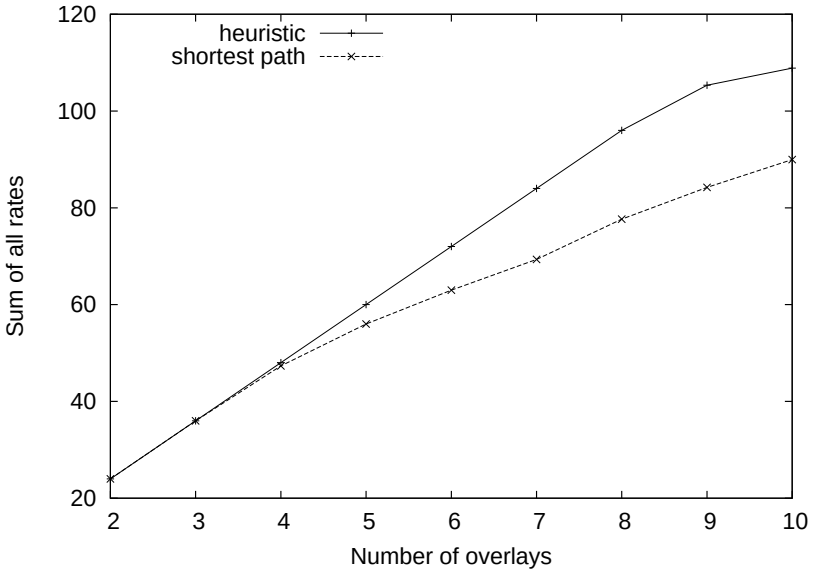


Figure 18: Total rates with uneven capacities (Topology 2)

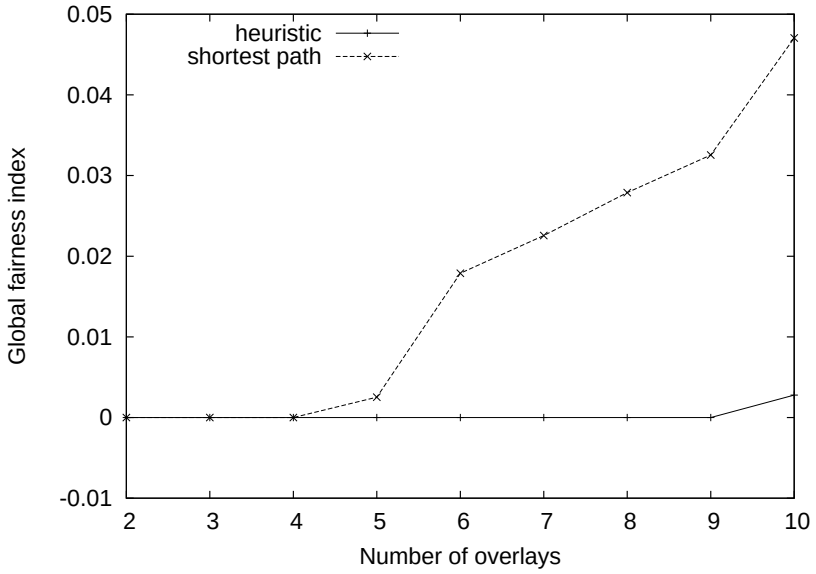


Figure 19: Using the route selection heuristic can also improve fairness. A global fairness index closer to zero is more desirable.

links excluded from shortest path routes have higher capacities than the links in those routes.

Moving some flows away from shortest-path routes can help us assign higher rates to them, but does this modification also have an effect on the fairness of our allocations? To answer this question, we evaluate each allocation using our fairness metric from Section 4.3. Figure 19 shows global fairness index values for the allocations over Topology 1. Recall that a global fairness index (GFI) closer to zero is considered better. When all overlays are receiving their demands in full, the sharing is perfectly fair and all GFIs are zero. Fairness issues may start to occur as more overlays are added, and using the routing heuristic can help more overlays to achieve rates closer to their isolated fair rates, thus also improving the fairness as evaluated and reflected by our metric.

4.7 Summary

In this work, we have examined the issue of fair bandwidth allocation among overlay networks competing for the resources of the same substrate network. We have observed that some existing fairness definitions and algorithms for achieving multipoint-to-point session fairness are applicable to the multiple overlay network scenario. We described a novel method to evaluate the fairness of an allocation in the presence of multiple sharing overlays. We also explored the effect of routing decisions on fair bandwidth allocation, and proposed brute force and heuristic methods to facilitate finding the routing which enables the optimal bandwidth allocation to flows from multiple overlay networks. We showed in the evaluation that our global fairness index metric successfully captures different treatment toward overlays. We also characterized the improvement in rates and fairness achieved by our routing modification heuristic.

The main takeaways from this chapter are the following:

1. When evaluating the fairness of a rate allocation among multiple overlay networks, a simple comparison of bandwidth demands and assigned rates is not sufficient. Significant differences between rates achieved (relative to demands) by different overlays do not necessarily stem from unfair treatment by the rate allocation, but are possibly due to the placement of some overlays in problematic regions of the substrate where link capacities are low and/or sharing is excessive. This should be taken into account when evaluating the fairness of rate allocations. Our fairness metrics does this by comparing rates given to overlay flows against what could be achieved if the same overlay were to exist in the same place without company. This approach provides the opportunity to evaluate rate allocations using a practical benchmark.

2. No allocation algorithm can provide perfect fairness in every setting involving

complex sharing among multiple overlays. Even though fair bandwidth allocation algorithms are necessary and useful in situations with high resource contention, eliminating as many bottleneck situations as possible through routing adjustments (if possible) is a better way to improve overall fairness in bandwidth allocation.

CHAPTER V

MULTI-LAYER MONITORING OF OVERLAY NETWORKS

5.1 Introduction

Monitoring all links in infrastructure overlay networks with persistent nodes is necessary to assess the overall performance of the users and to detect anomalies. Since an *overlay link* is in reality an end-to-end native path spanning one or more native links, this full monitoring operation can constitute a significant overhead (in terms of bandwidth and processing) for large overlays, especially if the monitoring is performed by active measurements.

In this work, we alleviate the overlay network monitoring problem by adopting a more flexible approach that allows certain native link measurements in addition to end-to-end measurements. These native link measurements can be used to infer desired metrics for overlay links by suitable combinations of native layer metrics. We call this approach *multi-layer monitoring*. This framework allows for four different options:

1. **Monitor all overlay links:** With this strategy, all overlay links are monitored directly and individually.

2. **Monitor a *basis set* of overlay links:** The work in [16] introduces a method to select and monitor a minimal subset of overlay links called the *basis set*. The characteristics of the remaining overlay links are inferred from the measurements for the basis set.

3. **Monitor all native links:** Another option is to monitor all the underlying

native links in the network. Afterwards observed native layer metrics are combined to produce the results for all the overlay links.

4. **Monitor a mix of native links and overlay links (Multi-layer Monitoring):** In this option proposed in this work, we monitor some native links and a subset of the overlay links. We then infer the remaining overlay links by combining these observations.

Note that while options 2-4 have the potential to reduce the monitoring cost, they are also prone to *inference errors* when an overlay link measurement is inferred from measurements on native and/or other overlay links.

The multi-layer monitoring strategy (option 4) is the most general one and subsumes all others. It also affords significant flexibility in monitoring overlays. Our objective in this work is to minimize monitoring cost by determining the optimal mix between overlay and native layer monitoring. To this end, we formulate this as an optimization problem and discuss some features of its solution.

Previous work has considered overlay network monitoring and developed various approaches for it. Chen et al. [16] propose an algebraic approach to efficiently monitor the end-to-end loss rates in an overlay network. They use linear algebraic techniques to find a minimal *basis set* of overlay links to monitor and then infer the loss rates of the remaining ones. iPlane [61] predicts end-to-end path performance from the measured performance of segments that compose the path. We generalize these techniques and allow measuring both end-to-end paths and underlying segments. Our approach in this work requires a deep collaboration between the overlay network operator and the native network, similar to the design goals of the overlay-friendly native network [72].

The remainder of this chapter is organized as follows: We describe the multi-layer monitoring problem in Section 5.2. Section 5.3 presents our linear program based

solution. We present details from simulating the multi-layer monitoring framework in general topologies in Section 5.4. Section 5.5 describes PlanetLab experiments that we conducted to characterize the inference errors that can result from this multi-layer monitoring solution. We summarize the chapter in Section 5.6.

5.2 The Multi-Layer Monitoring Problem

We model the native network as a directed graph $G = (V, E)$, where V is the set of vertices and E is the set of directed edges connecting these vertices. Next, we model the overlay network as a directed graph $G' = (V', E')$, with $V' \subseteq V$ being the set of overlay nodes and E' being the set of overlay links. In a multi-layer network, each overlay link spans one or more native links. Thus, the following relation holds: $e' \in E'$ is a set $\{e_{1e'}, e_{2e'}, ..., e_{ne'}\}$, where $e_i \in E$ and $e_{ke'}$ denotes the k^{th} native edge in e'.

Link monitoring incurs a certain cost, typically in the form of resource overhead (e.g., processor utilization, bandwidth), at each layer. We use $C(e)$ and $C'(e')$ as the cost of monitoring a native link and an overlay link respectively. Since $C(e)$ and $C'(e')$ are variables, the cost structure is flexible and can accommodate various scenarios. For instance, if it is not possible to monitor certain native links directly, the cost variables for those links can be set to infinity.

Let $\mathcal{M} = \{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_{\mathrm{N}}\}$ represent the desired set of monitoring operations we would like to get results for, which in our case is the set of desired overlay link measurements. Let $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_{\mathrm{Q}}\}$ represent the set of monitoring operations that are actually performed. This set can contain a mixture of native and overlay link measurements. Let *composition rule* $\mathcal{F}(\mathcal{P}, \mathcal{M}_{\mathrm{i}})$ represent a function that combines the results from available native and overlay link measurements to infer the desired measurement of the overlay link $\mathcal{M}_{\mathrm{i}}$. In this work, we use the composition rule of the latency metric.

Table 7: Notations used

E	Edges in the native layer
E'	Edges in the overlay layer
$C(e)$	Cost to monitor native link e
$C'(e')$	Cost to monitor overlay link e'
$X_m(e)$	1 if native link e is monitored, 0 otherwise*
$X_i(e)$	1 if native link e is inferred, 0 otherwise*
$Y_m(e')$	1 if overlay link e' is monitored, 0 otherwise**
$Y_i(e')$	1 if overlay link e' is inferred, 0 otherwise**
$f(e, e')$	1 if overlay link e' is routed over native link e, 0 otherwise
$x_i(e, e')$	1 if native link e is inferred from overlay link e', 0 otherwise
$l_i(e)$	Integer representing the inference dependency between native links to resolve inference loops

* A native link can be monitored or inferred but never both. Some are neither monitored nor inferred if they are not needed in inferring overlay link measurements.

** An overlay link is either monitored or inferred, but never both.

We say that a certain $\mathcal{M}$ is *feasible with respect to* $\mathcal{P}$, if all values in $\mathcal{M}$ can be computed from $\mathcal{P}$. Clearly, if $\mathcal{M} \subseteq \mathcal{P}$, then the monitoring problem is *feasible*. In cases when $\mathcal{M} \nsubseteq \mathcal{P}$, feasibility is not always assured.

The optimization problem can thus be stated as, "Given a monitoring objective $\mathcal{M}$, find the $\mathcal{P}$ such that $\mathcal{M}$ is feasible with respect to $\mathcal{P}$ and $cost(\mathcal{P}) = \sum_{i=1}^{Q} cost(\mathcal{P}_i)$ is minimal."

Assumptions and Limitations

In this work, we assume that the best-effort routing at the native layer treats measurement probes in the same manner as other data packets, so as to obtain an accurate estimate of the user experience. We restrict our work to the metric of latency, although it has been shown that the logarithm of link loss rates are additive metrics that can be composed in a manner similar to link latencies [16]. Furthermore, the linear programming formulation in the subsequent section cannot be applied for multi-path routing at the native layer: The overlay link latency composition rule needs revision for handling multi-path routing. We reserve these extensions to the model for future

study.

5.3 Linear Programming Formulation

Using the notation presented in Table 7, we formulate the optimization problem as the following Integer Linear Program (ILP):

$$\textbf{minimize Total Cost} = \sum_{e \in E} X_m(e) \cdot C(e) + \sum_{e' \in E'} Y_m(e') \cdot C'(e') \tag{13}$$

subject to the following constraints

$$\forall\ e' \in E', e \in e' : \quad X_m(e) + X_i(e) = 1, \text{ if } (Y_m(e') + Y_i(e')) = 0\ . \tag{14}$$

$$\forall\ e' \in E', e \in e', d \in (e' - e) : \quad x_i(e, e') \leq (X_m(d) + X_i(d))\ . \tag{15}$$

$$\forall\ e' \in E' : \quad \sum_{e \in e'} x_i(e, e') \leq (Y_m(e') + Y_i(e'))\ . \tag{16}$$

$$\forall\ e \in E : \quad X_i(e) \leq \sum_{e' \in E'} x_i(e, e') \leq 1\ . \tag{17}$$

$$\forall\ e' \in E', e \in e', d \in (e' - e) : \quad x_i(e, e') = \begin{cases} 1, & \text{if } l_i(e) > l_i(d), \\ 0, & \text{otherwise}\ . \end{cases} \tag{18}$$

$$\forall\ e' \in E' : \quad Y_i(e') = 1, \text{if } e' \text{ can be inferred from other overlay links in } \mathcal{P}\ . \tag{19}$$

$$\begin{aligned} \forall\ e \in E, e' \in E' : \quad & X_m(e) \in \{0, 1\}, X_i(e) \in \{0, 1\}, x_i(e, e') \in \{0, 1\}, \\ & Y_m(e) \in \{0, 1\}, Y_i(e) \in \{0, 1\}\ . \end{aligned} \tag{20}$$

Constraints (14) to (20) assure the feasibility of the solution. These constraints can be explained as follows:

(2) This constraint, applied to all overlay links, determines the exact layer at which each overlay link is to be monitored. If the overlay link is not already monitored or inferred, then monitor, or infer, all native links it spans. Furthermore, this

constraint will ensure that we only monitor or infer, and never both. This condition also prevents an overlay link from being monitored, if all its constituent native link measurements are already known.

(3) We enforce the constraint that a native link e is inferred from an overlay link e' only if all other native links in that overlay link are already monitored or inferred. This insures that the inferred native link can be appropriately calculated from other link measurements.

(4) This constraint insures that a native link e is inferred from an overlay link e' only if the overlay link latency is already monitored, or inferred, at the overlay layer (i.e., $Y_m(e') + Y_i(e') = 1$) . Furthermore, we place the constraint that no more than 1 native link can be inferred from each overlay link. This is typically achieved in an ILP by setting the sum of individual variables $x_i(e, e')$ to be less than or equal to 1.

(5) This is a complex constraint which achieves three sub-goals: (a) Mark a native link as *inferred* if it is inferred on any of the overlay links that span it, (b) Mark a native link as *not inferred* if it is not inferred on any of the overlay links that span it, and (c) Insure that a native link is inferred only from 1 overlay link, so as to reduce wasting resources on performing multiple inferences. These three constraints ensure that we accurately mark a native link as *inferred*.

(6) This constraint is crucial to remove any *circular inference*, which can happen if we infer one native link measurement through an arithmetic operation on the measurement of another. We achieve this by assigning integer inference levels (denoted by variable l_i), such that a native link must be inferred only from other native links that have a lower inference level.

(7) We use this constraint to implement the basis set computation and infer some

overlay link measurements from other known overlay link measurements.

(8) Lastly, we specify the binary constraints for all variables used. This constraint makes the problem hard.

We apply the described ILP to any given topology and solve it using the GNU linear programming kit [34], which uses the branch-and-bound approximation technique. The optimal solution for a given topology identifies the overlay links that can be inferred from other native and overlay links, and describes how these inferences should be done. Using this information, we infer the latency of all overlay links ($\mathcal{M}$) from available measurements ($\mathcal{P}$) in our database.

5.4 Examples Using Multi-Layer Monitoring

In this section, we present various simulation experiments to demonstrate the types of results obtainable from our optimization approach and how it is affected by various network features. Although we only simulate intra-domain topologies, our model and ILP are equally applicable to multi-domain topologies.

Random Placement

In the first experiment we consider five native link topologies derived from Rocketfuel [70] data. For each network we generate an overlay network using approximately 20% the number of nodes in the native topology as overlay nodes. These nodes are placed randomly among the native nodes and fully-connected to form the overlay network. In this case, we define the cost of monitoring as the total number of native and overlay measurements needed. We consider the following four monitoring strategies:

- *Monitoring all overlay links:* The total cost is the cost of monitoring all $|V'| \cdot (|V'| - 1)$ overlay links, where V' is the set of overlay nodes.

- *Monitoring all native links:* The total cost is the number of distinct native links spanned by all the overlay links.

Table 8: The lowest cost for each strategy when *unitNativeCost* is the same as *unitOverlayCost*

AS #	Number of overlay nodes	All overlay	All native	Basis set	Combination (n: native, o: overlay)
1221	21	420	102	198	98 (66 n, 32 o)
1755	17	272	112	98	92 (42 n, 50 o)
3257	32	992	240	500	222 (142 n, 80 o)
3967	15	210	98	138	78 (46 n, 32 o)
6461	28	756	224	394	210 (146 n, 64 o)

- *Monitoring a basis set of overlay links:* To obtain this solution, we set the cost of monitoring a native link very high in our ILP so that the solution selects only overlay links for monitoring.

- *Monitoring a combination of native and overlay links:* We set the cost of monitoring a native link equal to the cost of monitoring an overlay link in the ILP. (From here on, we refer to these costs as *unitNativeCost* and *unitOverlayCost*, respectively.) The ILP then produces a solution that minimizes the total cost, which is the same as minimizing the number of measurements in this case.

Table 8 demonstrates the lowest total monitoring cost that can be achieved by the above monitoring strategies for each topology. In addition, the cost that results from monitoring native links and the cost that results from monitoring overlay links are reported separately for the multi-layer combination strategy in the last column. In all topologies, monitoring a combination of native and overlay links provides the lowest-cost option. On average, this lowest cost is 71% lower than the cost for the naive all-overlay approach and 11% lower than the all-native solution. This represents significant saving, while being flexible enough to accommodate other constraints.

Amount of link-level overlap

In this section, we study the effect of overlap between overlay links over the optimal monitoring solution. As a measure, we use the average number of overlay links that

Table 9: Effect of link-level overlap on the lowest total monitoring cost

AS	Overlap coefficient	Lowest total cost	# of overlay links	Cost per link
3967	8.59	78	210	0.37
1755	9.21	92	272	0.34
6461	12.80	210	756	0.28
3257	17.08	222	992	0.22
1221	18.33	98	420	0.23

span a native link in the network. We call this value the *overlap coefficient*. For this analysis we use the results from the first experiment.

Table 9 demonstrates how the lowest cost solution, as given by our ILP, varies with the amount of link-level overlap. In the table, *Cost per overlay link* represents the total monitoring cost divided by the number of overlay links. The rows are sorted by increasing overlap coefficient. We observe that in general, the monitoring cost per overlay link decreases as overlap increases. However, the cost per link value for AS 1221 is slightly higher than that of AS 3257 although the former has a higher overlap coefficient. This may suggest that increasing overlap can only decrease the cost per link by a limited amount.

Percentage of overlay nodes

In this experiment, we vary the fraction of overlay nodes among all nodes in the network. We call this fraction *overlay node density*. We examine two Rocketfuel topologies using five different density values from 0.1 to 0.5, and random overlay node placement. Our ILP gives the results in Table 10 when $unitNativeCost = unitOverlayCost$. This result is consistent with the effect of link-level overlap. As the overlay node density increases, link-level overlap also increases, and the cost per overlay link decreases.

Table 10: Effect of overlay node density on the optimal monitoring solution

Overlay node density	Cost per link for AS 1755	Cost per link for AS 3967
0.1	0.75	0.71
0.2	0.34	0.37
0.3	0.25	0.28
0.4	0.15	0.17
0.5	0.12	0.13

5.5 Experimental Evaluation of Inference Errors

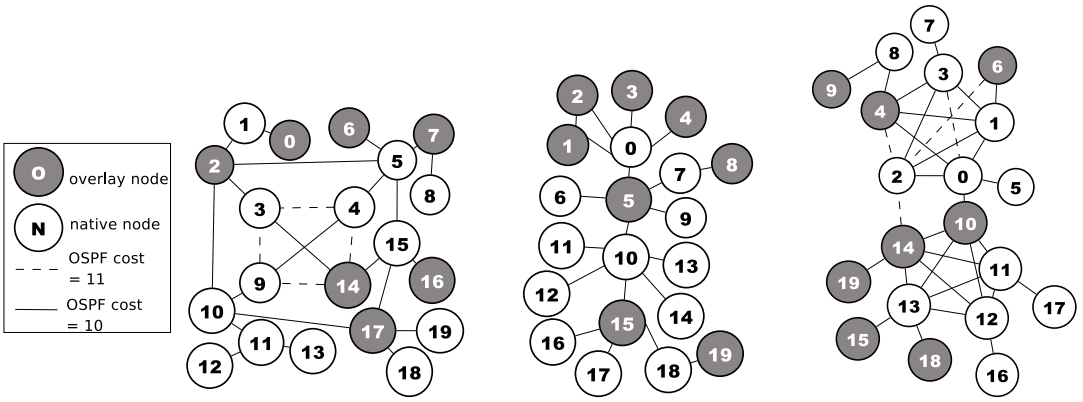
Composing an end-to-end measurement from other measurements can introduce an error in the result. We refer to this as *inference error*. One source of error may be packets traversing different sequences of router functions. For example, an end-to-end latency measurement probe may be forwarded along the fast path of a router, while probes that measure the latency of native links may be forwarded along the slow path. This makes the latter probe packets susceptible to processor delays, thereby introducing additional latency. Furthermore, some native link measurements may be inferred from overlay link measurements using arithmetic operations. This too introduces estimation error.

We represent the inference error for overlay links by computing the *absolute relative estimation error*. We compute this error value as a percentage:

$$\text{Abs. Rel. Est. Error Percentage}(e') = \frac{|\tilde{\rho}(e') - \rho(e')|}{\rho(e')} \times 100 \tag{21}$$

where $\rho(e')$ is the actual measurement result for e' and $\tilde{\rho}(e')$ is the inferred result obtained through combining a different set of measurements.

To assess the extent of inference errors, we conducted experiments on PlanetLab [69] using three different overlay topologies shown in Figure 20. We implemented these topologies as virtual networks on PlanetLab using PL-VINI, the VINI [10] prototype running on PlanetLab. In each experiment, we picked 20 PlanetLab nodes



(a) Topology 1 (b) Topology 2 (c) Topology 3

Figure 20: Three PlanetLab topologies we use. (a) represents a general AS topology. (b) has a tree-like structure which can be found on some campus-wide networks such as [23]. (b) can be interpreted as a graph of two interconnected ASes. Native links are assigned with different OSPF costs to avoid multiple shortest paths.

from different ASes as our native network and ran OSPF on this network with PL-VINI. Note that we cannot control the inter AS routing of these PlanetLab nodes. We treated the edges between these nodes on the PL-VINI network as native links. We picked 8 nodes out of the 20 as our overlay nodes, and assumed that these 8 nodes are fully connected to form an overlay network.

For each topology, we ran 4 rounds of measurements at different times. In each round, we measured the delay on all native and all overlay links by simultaneously running 100 pings on every link at a frequency of 1 per second. We calculated the delay from node a to node b as the average round-trip time over all ping results for native or overlay link $a - b$.

In order to find the optimal combination of links to monitor for these topologies, we ran our ILP on each of them with the objective of minimizing the total number of measurements. The output of the ILP gave us a set of overlay and native links to monitor. Using this output and the measurement results for the corresponding topology, we first inferred the measurements of the links that are not monitored, and then calculated the errors in these inferences using Equation 21. The errors for all-native and basis set solutions are calculated in a similar manner.

Table 11: Costs and inference errors for different monitoring strategies

(a) Topology 1

	Cost	Mn_i	Mn_a	*Max*
All-overlay	56	0	0	0
All-native	34	5.01	5.01	21.18
Basis set	38	2.68	0.86	20.29
Combination	26	3.43	2.70	20.12

(b) Topology 2

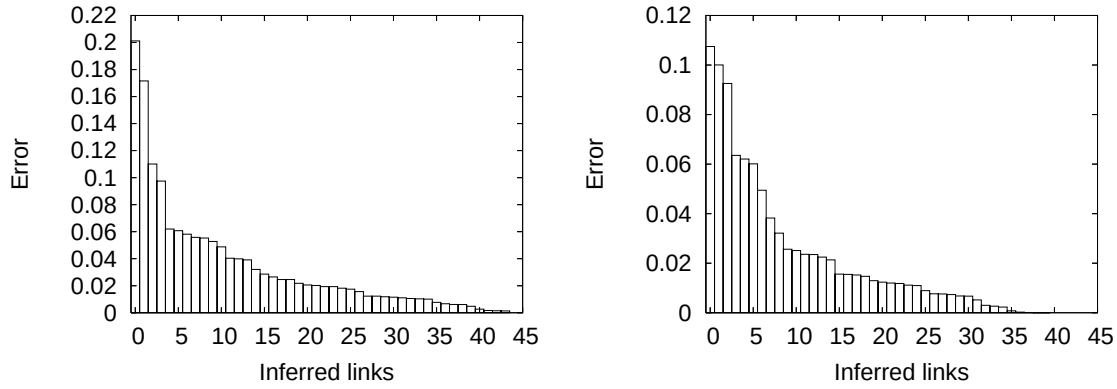
	Cost	Mn_i	Mn_a	*Max*
All-overlay	56	0	0	0
All-native	24	1.43	1.43	4.30
Basis set	26	0.96	0.51	2.79
Combination	18	1.58	1.35	3.17

(c) Topology 3

	Cost	Mn_i	Mn_a	*Max*
All-overlay	56	0	0	0
All-native	30	3.54	3.54	10.75
Basis set	26	1.13	0.61	4.95
Combination	24	2.35	1.68	10.75

Table 11 summarizes the results for all three topologies. The *Cost* column represents the lowest possible monitoring cost that can be achieved by each strategy. *Max* is the largest inference error observed in a certain strategy. Mn_i is the inference error averaged over all *inferred* overlay links, while Mn_a is the error averaged over *all* the overlay links in the network, with the difference being that direct overlay link measurements have no errors. Averaging over all overlay links does not reduce the error in the case of all-native monitoring because in this case all overlay links are inferred and none are measured directly. However, $Mn_a < Mn_i$ in the basis set and lowest-cost combination strategies because some overlay links are directly measured and these zero errors bring down Mn_a.

Among the last three strategies, monitoring a combination of native and overlay links achieved the lowest cost, and monitoring a basis set of overlay links resulted in the smallest error. However, we should note that if we use a different cost definition, such as the total number of native links carrying probe traffic, these results may change significantly. For instance, in Topology 3, the last strategy uses a combination of 8 native and 16 overlay links, spanning a total of 42 native links, while the all-native solution spans 30 links and the basis set solution spans 52 native links. Our



(a) Topology 1 (b) Topology 3

Figure 21: Error rates of inferred overlay links

insight from these experiments suggests that in general, all-native solutions minimize bandwidth consumption, basis overlay set solutions minimize error, and using a combination of native and overlay links allows reducing the total number of measurements with comparable errors.

For the two topologies whose maximum errors are above 10%, we examine the error distribution among the inferred overlay links as shown in Figure 21. We sort the inference errors from high to low and place them on the graphs from left to right. It can be seen that in both cases a few inferred links produce high errors that dominate the rest, increasing the mean error. If the ILP is aware of the overlay links that incur a high error when they are inferred, it can choose to monitor them directly and avoid these errors. Thus, adding certain error constraints to the ILP is a plausible step to improve its performance.

5.6 Summary

In this work we have proposed *multi-layer monitoring* as a flexible approach for overlay network measurement. We focused on the specific issue of determining the optimal mix of native and overlay link monitoring. We show that the overall cost of monitoring the network is the least when we allow native link measurements, as well

as end-to-end measurements. We present a novel ILP formulation that when solved minimizes the cost of network monitoring with the appropriate combination of end-to-end and native network measurements. Through simulation studies, we observe that the optimal monitoring solution, i.e., the set of native and overlay links that minimizes the total monitoring cost while supplying sufficient information, depends on unit monitoring costs as well as the selection and placement of overlay nodes. We also find that the average monitoring cost per overlay link is lower for topologies where there is a high overlap between overlay links. Furthermore, we evaluate our approach through PlanetLab experiments with a focus on the question of *inference errors*.

The main takeaways from this chapter are the following:

1. If the monitoring framework allows measurements at both the native layer and the overlay layer, the lowest-cost monitoring solution will usually utilize a balanced mix of direct measurements from both layers. This shows that there is good value in the multi-layer monitoring approach.

2. Inference of end-to-end metrics from other direct measurements causes a tolerable amount of error. The amount of error varies for different paths, and a few paths with high errors increase the average error. Once these paths are determined, a hard constraint requiring the monitoring system to monitor them directly can be introduced and the optimal monitoring mix can be updated as necessary.

CHAPTER VI

DESIGN AND ANALYSIS OF TECHNIQUES FOR MAPPING VIRTUAL NETWORKS TO SOFTWARE-DEFINED NETWORK SUBSTRATES

6.1 Introduction

Over the past few years, software-defined networking (SDN) has emerged as a promising technology. By separating the control plane from the data plane, SDN facilitates network experimentation and allows for optimizing switching/routing policies. The forwarding planes in SDNs are managed by remote processes called *controllers*. OpenFlow [63] was proposed and has been extensively used as a uniform interface between the control and data planes in SDNs.

Multiple tenants can share the underlying SDN infrastructure through virtualization. One method of virtualization in SDNs is FlowVisor [76], a special purpose OpenFlow controller that can create *slices* of network resources and place each slice under the control of a different OpenFlow controller. FlowVisor slices the network along multiple dimensions: topology, bandwidth, switch CPU, and the set of traffic under the control of the slice, i.e., its *flowspace*. Each slice has a *slice policy* defining its resources and the controller associated with it.

FlowVisor is interjected between multiple OpenFlow slice controllers and OpenFlow switches acting as the data plane. It intercepts messages between the control and data planes in both directions, and rewrites them in compliance with the slice policy of the corresponding slice. FlowVisor uses a variety of mechanisms (details are in [76]) for different network resources to ensure isolation between slices. Switch CPU is identified as the most critical resource as it is the most easily exhausted, but

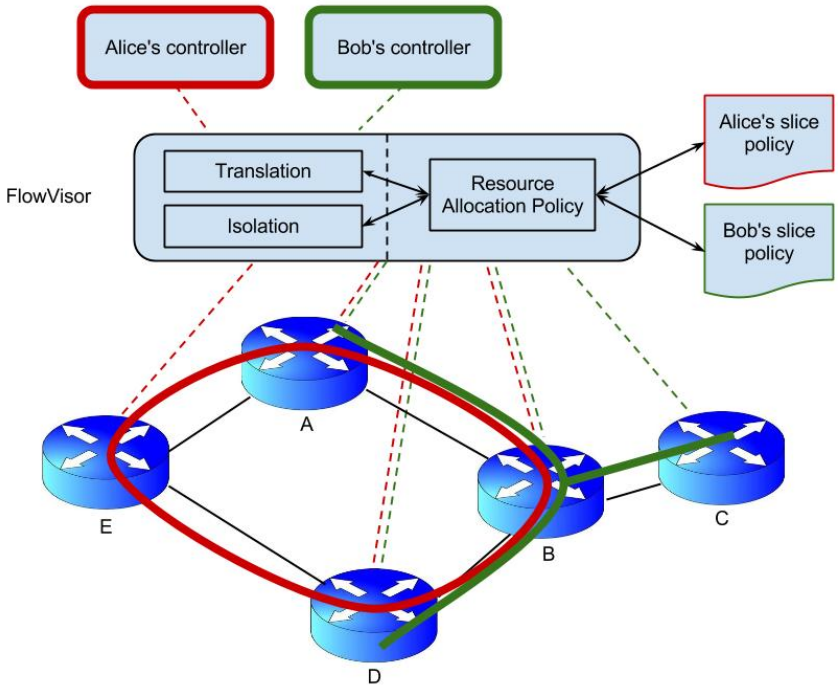


Figure 22: Two FlowVisor slices sharing a substrate of OpenFlow-enabled switches

there are methods to divide and isolate link bandwidth and flow entry space as well.

Figure 22 illustrates two slices sharing the same network substrate. Each slice has its own view of the topology, which we call the *virtual topology* (VT) of the slice, depending on the switches it contains: Alice's VT is ABDE, and Bob's VT is ABCD. Switches that are added into both slices (like A, B, and D) need to allocate CPU power and flow entries between these slices. Similarly, if a link is being shared between multiple slices (like AB and BD), its bandwidth must be divided to accommodate the flows belonging to each slice. Note that the dashed lines showing the controller-to-switch communications constitute a conceptual representation. The exact paths for control traffic will be determined by the physical locations of the controllers.

Slices of the network can be used for many different purposes depending on what the owners are trying to accomplish. Services or experiments that run on these slices may display a wide variety of resource requirements. We can think of a slice along with its controller as a virtual network (VN) in SDN. As VNs get bigger and the

number of VNs increases, resource contention may become a problem.

VN embedding[1] within a *network virtualization environment*, where substrate network providers are decoupled from VN providers that deploy and operate the VNs [29, 86], is a well-known problem. It is an important NP-hard problem because suboptimal mappings can cause bad performance and/or higher operating cost. Solutions that strive for well-balanced and resource-efficient VN mappings have been proposed by researchers [59, 92, 96].

In this work, we tackle the problem of designing embedding techniques for VNs in the SDN environment. While some of the previous VN embedding solutions may be applicable to SDN virtualization, some differences inherent in SDN call for different definitions and approaches. Firstly, each VN on an SDN-based substrate possesses its own controller, and there are requirements and considerations that come with the existence of this controller. The controller is responsible for organizing the operation of the VN by sending routing updates, traffic engineering policies etc. and needs to react quickly to faults in the network. Therefore, it must be able to communicate effectively to all the switches that are part of the VN, so any VN embedding effort needs to make sure all controller-to-switch channels avoid congestion and high delays. Furthermore, the ease of customizing packet routes in SDNs presents an additional degree of freedom in VN embedding. For these reasons, studying VN embedding in SDNs presents different challenges from other network virtualization environments:

1. The placement of the controller is important for the above stated reasons, so it should be treated as a special node by the VN embedding method.

2. We have more control over the routes in the substrate, so VN embedding can take advantage of this flexibility to fine-tune solutions.

3. As we will explain in Section 6.2, differences in SDN resource sharing from that

[1] *VN embedding* and *VN mapping* are being used interchangeably throughout this chapter.

in traditional networks call for modifications to previous VN embedding efforts.

The systems that we investigate in this work are those with many tenants running a number of VNs with varying sizes, functions and requirements. The mapping of the virtual components to substrate components and the placement of the controllers are both important variables influencing the performance of the VNs in such a system. Efficient utilization of networks resources, low risk of congestion, reliability and fast response are all desirable properties. To this end, we identify two simple goals that guide our design: balancing the load on substrate nodes and links, and maintaining low delay between controllers and switches in all VNs.

The contributions of this work can be summarized as follows.

- We identify virtualization properties unique to the SDN environment and discuss how these properties should affect VN embedding efforts. Specifically, we argue that the placement of the controller for each VN should be a part of embedding due to the critical mission of the controller and the necessity for the controller to have a fast communication channel to the switches that it controls.

- We developed two embedding techniques with different priorities: The first method focuses on balancing the stress on substrate components while keeping the delays between controllers and switches within bounds. The second method strives to minimize controller-to-switch delays while limiting the stress on substrate components.

- Our stress-balancing embedding technique is derived from previous work [96], but we modify the definitions of node and link stress to account for the presence of control traffic, and make adjustments to treat the controller as a special node.

- We evaluated the impact of both techniques on the objective metrics (stress and controller-to-switch delays) as well as other metrics such as end-to-end

delays and throughput. We also analyzed the effects of changes in the tunable parameters of our methods and definitions.

The remainder of the chapter is organized as follows. In Section 6.2, we describe resource sharing in SDNs and formally define the VN embedding problem. Section 6.3 presents our approach and two variations of VN embedding techniques with different priorities. Section 6.4 contains results from evaluation efforts conducted via simulation and Mininet [53] emulation, and Section 6.5 summarizes the chapter.

6.2 Model and Problem Statement

In this section, we first briefly discuss the nature of resource sharing in SDNs in the presence of multiple virtual networks, and then describe the VN embedding problem in the SDN environment.

6.2.1 Identifying SDN Resources

Virtualization in SDNs requires a mechanism to share network resources among multiple slices. A classification of these resources was presented in the FlowVisor work [76]. We summarize these as follows.

Switch CPU: Sharing computational power between slices at switches is orchestrated considering two main tasks performed for each slice: generating new flow messages to be sent to the corresponding controller, and handling controller requests regarding the slice. CPU power demands at a switch increase with the number of virtual nodes and links at mapped to it / traversing it.

Bandwidth: Isolation of bandwidth is handled via per-slice queues at each port on OpenFlow switches. Each slice has its own queue at each port, and these queues are serviced according to the resource allocation policy. In particular, FlowVisor implementation uses minimum bandwidth queues, whereby a slice is guaranteed a

minimum of $X\%$ of the bandwidth on a link, and possibly more if the link is not fully utilized.

In our design, we take into account both the number of virtual links that share a substrate link and the number of controller-to-switch connections that must go over this link. Balancing these numbers across the substrate is useful to avoid potential hot spots and reduce the possibility of congestion.

Flow entry space: The number of flow entries that can be used by each slice is also limited at each switch. When a controller is over its limit, it is not allowed to insert any new rules at the switch. This is another reason to consider the number of virtual links traversing a substrate node as part of the load on that node.

6.2.2 VN Embedding Problem

Network model: We take the underlying SDN infrastructure as the network substrate and model it as a graph (V, E), where V is the set of substrate nodes and E is the set of substrate links.

We can think of VN-embedding in SDN as a combination of two sub-problems:

1. Deciding which switches to include in a slice,

2. How to configure the routing within the slice.

This problem becomes interesting only when there are many VNs sharing the underlying infrastructure. So we are focusing on the problem of embedding multiple VNs on a substrate of OpenFlow enabled switches using FlowVisor.

We consider controller placement and VN embedding as a joint problem and look at two flavors: *VN embedding with fixed-location controller* and *VN embedding with adjustable-location controller*, depending on whether the location of the controller for a VN has been predetermined or can be selected freely. With multiple VNs, the location of the controller for each VN can be either fixed (as some users may

desire the ability to select precise locations for the central controllers of their VNs) or adjustable. Our objective for VN embedding is twofold: balancing the load on switches and links while trying to minimize the average controller-to-switch delay for each VN. As a measure of load, we concentrate on *stress* from previous work [96] in the VN embedding literature.

We modify the definition of stress given in [96], where it was defined for a substrate node as the number of virtual nodes mapped to it, and for a substrate link as the number of virtual links traversing it. We add two things to these definitions: For node stress, we take into account the additional CPU power and flow entry space load that comes from virtual links that traverse a node; and for link stress, we add a component for the total load due to all the controller-to-switch communications going over a particular link. The definition of stress in this work is as follows.

- For a substrate node $v \in V$, stress is a weighted combination of the number of virtual nodes assigned to it ($n_N(v)$), and the number of virtual links traversing it ($n_L(v)$). More formally, if $S_N(v)$ denotes the stress of substrate node v, then

$$S_N(v) = \alpha \cdot n_N(v) + \beta \cdot n_L(v), \forall v \in V \tag{22}$$

  Modifying the positive parameters α and β in the above definition will change the importance given to each component of node stress. If hosting a virtual node is considered to cause more stress for a substrate node than being part of a virtual link, α will be assigned a bigger value relative to β, and vice versa. In a special case, if $(\alpha, \beta) = (1, 0)$, the definition reduces to the simpler form given in [96].

- For a substrate link $e \in E$, total link stress is the sum of data traffic stress and control traffic stress. Data traffic stress is placed on e by virtual links traversing it ($tr(e)$ is the number of such virtual links), and each virtual link contributes

an amount equal to the ratio of its bandwidth demand and the capacity of e. Control traffic stress is a weighted contribution for each controller-to-switch path going over this link. For controller C_i, each controller-to-switch connection contributes γ_i to the stress of each link on the path, where $0 \leq \gamma_i \leq 1$. Let $c_i(e)$ denote the number of controller-to-switch connections traversing link e in the i^{th} VN out of $numV$ total VNs in the system. More formally, if $S_L(e)$ denotes the stress of substrate link e, then

$$S_L(e) = \sum_{i=1}^{tr(e)} \frac{demand_i}{cap(e)} + \sum_{i=1}^{numV} \gamma_i \cdot c_i(e), \forall e \in E \tag{23}$$

In calculating the link stress, the demand of a virtual link is capped at the lowest substrate link capacity in the corresponding path since that is an implicit limit on the highest rate the virtual link can achieve. If bandwidth demands are not known in advance, we assume they are all equal. γ_i can be different for each VN depending on the intensity of its control traffic. A higher γ_i would signify that the load caused by control traffic is more important for this VN than that in a VN with a lower γ_i value.

The formal problem definition for VN embedding in SDNs is given below.

Let $Slice_i$ be the i^{th} slice sharing a network of switches. The virtual topology of this slice, VT_i, defines the nodes and links included in this slice. VT_i is described as a graph (V_i', E_i'), where V_i' is the set of virtual nodes, and E_i' is the set of virtual links. This slice, along with its controller C_i, constitute virtual network N_i. Our problem can then be states as follows:

Given the virtual topology of all numV VNs, and information as to whether the location of each C_i is fixed or adjustable, organize all VNs in a way that will 1) minimize the maximum stress on a switch or link while keeping all controller-to-switch delays under a threshold R, or 2) minimize the average controller-to-switch delay for

each N_i while keeping the maximum stress under a threshold T.

6.3 VN Embedding Techniques

We consider two aspects of the VN embedding problem and address them together: balancing the load and resource usage in the network in the presence of multiple VNs, and controlling the delay in the communication between controllers and switches in all VNs. We describe two heuristics to tackle this version of the NP-hard VN embedding problem, one balancing the load as much as possible while keeping controller-to-switch delays within certain bounds, the other minimizing these delays while keeping the maximum stress under a threshold.

6.3.1 Stress-Balancing Embedding (SBE)

The goal of the first heuristic is to produce low and balanced stress on substrate nodes and links while making sure that the worst controller-to-switch delay does not exceed a certain threshold. Reducing stress on nodes is important for decreasing the possibility that resources such as CPU processing power and flow tables will get exhausted. Keeping stress low on links helps avoid congestion and may result in a lower percentage of the link's bandwidth being consumed. For delays, we use a threshold R to limit the maximum controller-to-switch delay.

We borrow the concept of *neighborhood resource availability* (NR) for substrate nodes from [96]:

$$\mathrm{NR}(v) = [S_{Nmax} - S_N(v)] \cdot \sum_{e \in L_v} [S_{Lmax} - S_L(e)] \tag{24}$$

$S_{Nmax} = \max_{v \in V} S_N(v)$ is the maximum node stress, and $S_{Lmax} = \max_{e \in E} S_L(e)$ is the maximum link stress in the substrate. L_v in the set of substrate links adjacent to v.

A high NR signifies that the node and the links connected to it are lightly loaded.

We make use of NR to inform our node selection process. The main idea in this technique, similar to [96], is to map high-degree virtual nodes to substrate nodes with high NR, since high-degree virtual nodes are going to set up more connections and need more resources. We perform this mapping in a controlled way to ensure that the delay threshold R is not exceeded. The details of the heuristic are shown in Algorithm 7, and an explanation of the steps of the heuristic is given below.

Step 1.0. Precomputation of pairwise delays: Given the substrate network topology (V, E) and the delays between neighboring nodes, find the shortest path between all $|V| \cdot |V - 1|$ node pairs and populate the pairwise delay matrix for all these pairs.

Step 1.1. Mapping VNs with fixed-location controllers: Assume the heuristic is given $numV$ VNs, $numF$ of which having fixed controller locations. For each of these $numF$ VNs, it finds all substrate nodes whose distances from the controller are at most R to form the set of potential hosts, H, for virtual nodes. Then, it performs a one-to-one mapping between virtual and substrate nodes such that virtual nodes with higher degrees are mapped to substrate nodes with higher NRs *within the set* H. After that, it maps each virtual link to the shortest path between its source and destination nodes.

Step 1.2. Mapping VNs with adjustable-location controllers: After the above step, the heuristic moves on to the VNs with adjustable-location controllers. It orders the VNs according to their sizes, i.e., VNs with the most virtual links are mapped first. The controller for the biggest VN is attached to the available substrate node with the highest NR value, and then, the heuristic maps its nodes one by one, according to the rules in Step 1.1, and repeats this until all VNs are mapped.

Picking all nodes from H ensures that we never go over the delay threshold R for any VN. The rest is fixing all controllers to high-NR nodes and performing the mappings according to the "high degree to high NR" principle advocated in [96] to produce low and balanced stress.

Algorithm 7 Stress-balancing embedding heuristic (SBE)

Input: Substrate network (V, E), $numF$ VNs with fixed-location controllers (N_1, ... , N_{numF}), $numV - numF$ VNs with adjustable-location controllers (N_{numF+1}, ... , N_{numV}) where $numF \leq numV$, and the maximum delay threshold R,
Compute the shortest path for all node pairs (x, y) s.t. $x, y \in V$.
Calculate the delay for all such node pairs.
for $i = 1 \rightarrow numF$ **do**
    Find the set H of nodes such that the delay from C_i to the node is at most R.
    Calculate the NR for all members of H.
    Sort all virtual nodes in V_i in the order of decreasing node degree in the virtual topology.
    for $j = 1 \rightarrow |V_i'|$ **do**
        Map v_j' to $h \in H$ s.t. NR$(h) = max_{v \in H} NR(v)$.
        Remove h from H.
    end for
    for $j = 1 \rightarrow |E_i'|$ **do**
        Set the path for virtual link e_i' to the shortest path between its source and destination.
    end for
end for
Order remaining VNs in terms of decreasing number of virtual links.
for $i = numF + 1 \rightarrow numV$ **do**
    Calculate NR for all substrate nodes.
    Attach C_i to the node with the highest NR value.
    Find the set H of nodes such that the delay from C_i to the node is at most R.
    Sort all virtual nodes in V_i in the order of decreasing node degree in the virtual topology.
    for $j = 1 \rightarrow |V_i'|$ **do**
        Map v_j' to $h \in H$ s.t. NR$(h) = max_{v \in H} NR(v)$.
        Remove h from H.
    end for
    for $j = 1 \rightarrow |E_i'|$ **do**
        Set the path for virtual link e_i' to the shortest path between its source and destination.
    end for
end for
Return the final mapping and quit.

6.3.2 Delay-Minimizing Embedding (DME)

The second heuristic primarily aims to minimize average delays between VN controllers and switches used by the VN. Keeping these delays low is crucial for the timely response of the controller to switch events that require intervention. The maximum stress is not minimized but it is controlled by an upper limit T. We define T as a pair (T_N, T_L) to be able to impose limits on both maximum node stress and maximum link stress. These limits can be high or low relative to each other depending on what we want to prioritize. The main idea is to select nodes that are near their corresponding controllers, and then rerouting to avoid exceeding stress thresholds. The technique is shown in algorithmic form in Algorithm 8, and the operation is described in detail in the following steps:

Step 2.0. Precomputation of pairwise delays: Given the substrate network topology (V, E) and the delays between neighboring nodes, find the shortest path between all $|V| \cdot |V-1|$ node pairs and populate the pairwise delay matrix for all these pairs.

Step 2.1. Mapping VNs with fixed-location controllers: Given $numV$ VNs, $numF$ of which having fixed-location controllers, the heuristic starts by ranking the switches in the substrate by the delay from the controller of each of these $numF$ VNs. For each of these $numF$ VNs, it then performs a one-to-one mapping between all virtual nodes and the substrate nodes that are closest to the controller, such that the stress on any of the substrate nodes will not exceed T_N after this stage. It determines the shortest path between the pair of nodes in each virtual link and selects that as the route for the virtual link.

Step 2.2. Mapping VNs with adjustable-location controllers: The heuristic orders the VNs according to their sizes, i.e., VNs with the highest number of virtual links are mapped first. For each VN to be mapped, it finds three nodes with the lowest average distance to all the other substrate nodes and attaches the controller of the VN to the node with the highest NR out of that three. Then it performs the above

described fixed-controller VN mapping procedure.

Step 2.3. Virtual link rerouting: After the initial mapping of all VNs is completed, the next step is ensuring that the maximum stress threshold T is not exceeded. To this end, the heuristic takes advantage of the rerouting possibilities presented by the underlying SDN substrate to move virtual links away from highly stressed substrate nodes and links that are causing T to be exceeded. It calculates the stress on every substrate node and link, and then the maximum node stress, $S_{Nmax} = \max_{v \in V} S_N(v)$, and the maximum link stress, $S_{Lmax} = \max_{e \in E} S_L(e)$. If either S_{Nmax} exceeds T_N or S_{Lmax} exceeds T_L, then the heuristic performs rerouting as described at the end of Algorithm 8 until both $S_{Nmax} \leq T_N$ and $S_{Lmax} \leq T_L$.

The delay-minimizing embedding results in low controller-to-switch delays because it greedily selects switches that have the lowest delay to the controller. Attaching the central controller to the node with the largest NR helps situate the VN in a relatively empty area in the substrate, and routing flexibility is exploited to limit maximum stress.

6.4 Evaluation

In this section, we evaluate our VN embedding techniques with simulation and Mininet [53] emulation.

6.4.1 Metrics

Below are the metrics that we use to evaluate our VN embedding techniques:

- *Controller-to-switch delay:* average and maximum (for each VN)
- *Node stress:* average and maximum (across an entire substrate)
- *Link stress:* average and maximum (across an entire substrate)
- *End-to-end delay:* average (for each VN)

Algorithm 8 Delay-minimizing embedding heuristic (DME)

Input: Substrate network (V, E), $numF$ VNs with fixed-location controllers (N_1, ... , N_{numF}), $numV - numF$ VNs with adjustable-location controllers (N_{numF+1}, ... , N_{numV}) where $numF \leq numV$, and the maximum stress threshold $T = (T_N, T_L)$,
Compute the shortest path for all node pairs (x, y) s.t. $x, y \in V$.
Calculate the delay for all such node pairs.
for $i = 1 \rightarrow numF$ **do**
    Sort all nodes in N_i according to delay from C_i.
    Available nodes $\leftarrow V$
    for $j = 1 \rightarrow |V_i'|$ **do**
        Map v_j' to the available node v_j such that v_j is the closest to C_i and $S_N(v_j) + \alpha \leq T_N$. ($\alpha$ is from Equation 22.)
        Remove v_j from the list of available nodes for N_i.
    end for
    for $j = 1 \rightarrow |E_i'|$ **do**
        Set the path for virtual link e_i' to the shortest path between its source and destination.
    end for
end for
Order remaining VNs in terms of decreasing number of virtual links.
for $i = numF + 1 \rightarrow numV$ **do**
    Calculate NR for all substrate nodes.
    Find the 3 substrate nodes with the lowest average distance to other nodes.
    Attach C_i to the one with the highest NR among this 3.
    Available nodes $\leftarrow V$
    for $j = 1 \rightarrow |V_i'|$ **do**
        Map v_j' to the available node v_j such that v_j is the closest to C_i and $S_N(v_j) + \alpha \leq T_N$.
        Remove v_j from the list of available nodes for N_i.
    end for
    for $j = 1 \rightarrow |E_i'|$ **do**
        Set the path for virtual link e_i' to the shortest path between its source and destination.
    end for
end for
Calculate stress for each node in V and each link in E.
$S_{Nmax} \leftarrow \max_{v \in V} S_N(v)$, $S_{Lmax} \leftarrow \max_{e \in E} S_L(e)$
$v_{max} \leftarrow v \in V$ s.t. $S_N(v) = S_{Nmax}$
$e_{max} \leftarrow e \in E$ s.t. $S_L(e) = S_{Lmax}$
if $S_{Nmax} > T_N$ **or** $S_{Lmax} > T_L$ **then**
    Reroute virtual links away from v_{max} and e_{max} until $S_N(v_{max}) \leq T_N$ and $S_L(e_{max}) \leq T_L$.
else
    Return the final mapping and quit.
end if
Go to 29.

- *Throughput:* average (for each VN)

As controller-to-switch delay and stress on substrate components are two things that we are optimizing for, we study how they are affected by different types of embedding through simulation. In addition, we want to gain insights as to how these goals are influencing VN performance metrics, such as end-to-end delays and throughput. To this end, we emulate scenarios on Mininet to understand how end-to-end delays and throughput are changing with different embedding methods.

6.4.2 Strategy

As substrate networks, we utilize 10 different topologies from the Internet Topology Zoo [51]. These topologies vary in size and shape, allowing us to evaluate our techniques with different network properties. We construct 20 different virtual topologies, ranging from 5 nodes to 20 nodes. For each of these virtual topologies, we randomly select a coefficient in the range [0.3,0.7] as the probability of connection between any two virtual node pairs. This gives us some variation in graph density. We randomly assign fixed-location controllers to 5 of these VNs, the rest have adjustable-location controllers (except for the part of the evaluation where we vary the percentage of fixed-location controllers, in Section 6.4.3.5).

The parameters in the node stress formula from Section 6.2 are selected as (α, β) = (1,1), and we assume all virtual link demands and substrate link capacities are equal. For γ_i, we pick a random value from the range [0,0.5] for each VN separately. This provides a variation in the contribution of control traffic toward link stress for different VNs. We do not let γ_i go over 0.5 in this experiment because that could cause control traffic to dominate link stress and lower the number of viable locations for controllers, thus reducing the flexibility in controller placement (more on this at the end of this section, in Section 6.4.3.6).

We compare SBE (Algorithm 7) and DME (Algorithm 8) against two simpler variations: pure stress-balancing mapping and naive delay-minimizing mapping. Pure stress-balancing mapping executes SBE without regard to delays (i.e., with an extremely high delay threshold R). Naive delay-minimizing mapping places the adjustable-location controllers randomly, and then maps virtual nodes to the closest possible substrate nodes to minimize controller-to-switch delays, similar to DME yet without any attention to stress. We also consider purely random mapping, which performs virtual node mapping and controller localization completely randomly over the entire substrate, but it performs badly enough to disrupt graph scales, so we do not show it in all graphs and we declare it infeasible for practical use compared to the other options.

We run evaluations with the following objectives:

- Comparing average and maximum controller-to-switch delays for SBE, DME, and two simpler variations of these techniques,
- Comparing average and maximum stress for the same four techniques,
- Comparing average end-to-end delays and throughput using Mininet emulation,
- Varying the percentage of VNs with fixed-location controllers to see how this affects metrics.
- Varying stress threshold T and the contribution of control traffic to link stress, γ_i, to measure their effects.

To produce results, we map all 20 VNs together to each of the substrate topologies using all of the different techniques to be evaluated. For each VN, we average the resulting metrics over the 10 substrate topologies. SBE uses a controller-to-switch delay threshold R of 50ms, which is identified as a common target for maximum latency in [37] because it is the target restoration time of a SONET ring. DME uses

a variable stress threshold $T = (T_N, T_L)$ depending on the stress values achieved by SBE for the same scenario. By default, T_N and T_L are set to be 50% more than the S_{Nmax} and S_{Lmax} procuded by SBE, respectively, for a given substrate and set of VNs. However, we evaluate the effect of different T values in Section 6.4.3.6.

6.4.3 Results

6.4.3.1 DME with a small example

Before moving on to larger topologies, let us demonstrate the properties of one of our heuristics: DME. We consider placing a 3-node VN on a 5-node substrate topology with link delays ranging from 5ms to 20ms. There are $C(5, 3) = 10$ possible virtual node mappings, and 5 possible controller locations, resulting in a total of 50 VN mappings. Average controller-to-switch delays for these 50 mappings range from 5ms to almost 17ms. DME produces the mapping with the average controller-to-switch delay of 5.33ms, the second lowest value among the 50.

The simple 3-node VN topology does not allow much variation in node stress, but link stress values are more variable. We set γ (control traffic coefficient) to 0.5. If the link stress threshold T_L is set to 2.5, DME does not need to perform any rerouting. However, if T_L is reduced to 1.5, DME manages to get under the threshold by rerouting a virtual link and increasing its expected end-to-end delay from 16ms to 30ms in the process. Therefore, we conclude that DME is able to achieve optimal or near-optimal controller-to-switch delays. It may or may not need to perform rerouting depending on stress thresholds. If rerouting is required, DME can reduce the maximum stress under the threshold even in a small topology with very few route options. Controller-to-switch delays are not affected by rerouting since there is no node remapping, but end-to-end delays for rerouted virtual links may increase significantly. However, unless stress thresholds are unreasonably low, DME can get away with rerouting only a small fraction of virtual links.

6.4.3.2 Controller-to-switch delays

We look at how different embedding techniques affect the delays between the VN controller and the switches in the VN. We consider both average and maximum controller-to-switch delays for each VN: The average is calculated over all of the switches included in a given VN, and the maximum is the delay from the controller to the farthest switch in the VN.

Average and maximum controller-to-switch delays for all VNs are shown in Figure 23 and Figure 24 respectively, for SBE, DME, naive delay-minimizing mapping, and pure stress-balancing mapping. Each data point is these graphs represents the value for one of 20 VNs, averaged over mappings to 10 substrate topologies. All 20 values produced by the same technique are sorted in increasing order, and then plotted. The X-axis is labeled "Fraction of VNs" because the x-coordinate corresponding to a y-value signifies the fraction of all VNs that have a delay less than or equal to that y-value. For example, a data point (0.2, 10) would suggest that 20% of all VNs in this experiment (i.e., 4 out of 20) have delays of at most 10ms, for the particular technique that the data point belongs to. We use this X-axis in the other graphs in the remainder of this evaluation section as well.

In Figure 23, we see that DME produces very close results to (in some case better than) the naive delay-minimizing mapping, even though the former is also trying to limit stress. We infer that the stress thresholds T_N and T_L do not significantly affect the mapping in most cases, and when they do, the effects are minimal, possibly because DME utilizes minimally invasive rerouting in Step B.3. instead of remapping nodes in cases where stress thresholds are exceeded after the previous steps. SBE and the pure stress-balancing heuristic produce higher delays, but SBE is able to perform a bit better than pure stress-balancing mostly because it forces itself to keep controller-to-switch delays under the threshold R as it is doing the mapping.

Maximum delays shown in Figure 24 exhibit a similar pattern: Maximum delays

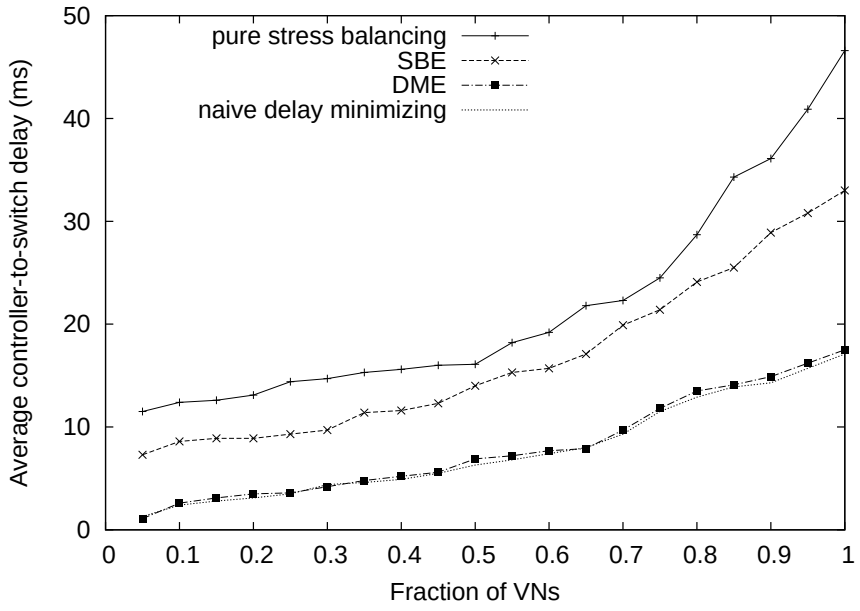


Figure 23: Average controller-to-switch delays for all VNs: SBE, DME, naive delay minimizing mapping, and pure stress balancing mapping.

for DME and naive delay-minimizing mapping are close to each other, and maximum delays for the other two techniques are higher as expected. We see several instances where the worst-case delay for SBE is close to the threshold of 50ms, while the worst-case delays for pure stress-balancing exceed 50ms in those cases. This suggests that SBE has made an active effort to keep the maximum delay under 50ms.

6.4.3.3 Node and link stress

We now analyze the impact of the heuristics on node and link stresses. For this experiment, we use the results collected from the simulation in the previous subsection (Section 6.4.3.2). But instead of reporting stress values per VN, we focus on the overall average across the entire substrate in all 10 topologies.

Average and maximum node stress values from all runs are shown in Table 12, and average and maximum link stress values are shown in Table 13 for all four techniques

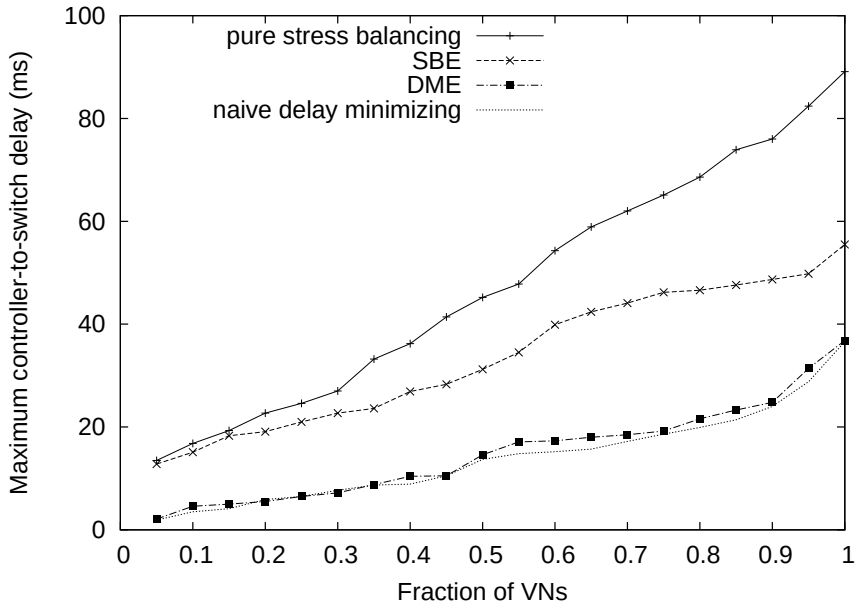


Figure 24: Maximum controller-to-switch delays for all VNs: SBE, DME, naive delay minimizing mapping, and pure stress balancing mapping.

considered. While these values do not tell much on their own, a comparison among them yields the following: SBE performs comparably to pure stress balancing, and it may be preferable since it provides significant savings in maximum controller-to-switch delays. The naive delay-minimizing heuristic is about twice as bad for stress as optimal heuristics, and DME performs somewhere in between these ends due to its stress balancing components.

6.4.3.4 End-to-end delays and throughput

For this part of the evaluation, we use Mininet 2.0 to emulate the operation of an SDN with multiple VNs placed on it. 10 different VNs are embedded on 3 substrate networks sliced using FlowVisor. Bandwidth for all substrate links is set to 100Mbps. We use ping to measure end-to-end delays, and execute file transfer between every pair of nodes via TCP with Iperf [40].

Table 12: Average and maximum node stress

	Avg node stress	Max node stress
Pure stress balancing	11.4	21.0
SBE	12.1	23.5
DME	16.8	32.4
Naive delay minimizing	19.2	41.1

Table 13: Average and maximum link stress

	Avg link stress	Max link stress
Pure stress balancing	6.5	9.8
SBE	6.8	10.9
DME	10.3	16.1
Naive delay minimizing	12.7	24.3

Figure 25 demonstrates average end-to-end delays from SBE, DME, random mapping, and the naive delay-minimizing mapping. DME is performing considerably better than SBE and random mapping, offering an average of 45% reduction compared to SBE and a 65% reduction compared to random mapping. DME tends to cluster the virtual nodes of a VN as closely as possible without exceeding stress bounds, so it is good at minimizing both controller-to-switch delays and end-to-end delays. The naive delay-minimizing mapping heuristic performs closely to DME, and even better in some situations due to forced rerouting increasing end-to-end delays in DME. SBE does better than random mapping, probably because it tries avoid distant low-degree nodes.

Figure 26 shows average throughput results collected over 20 emulations. DME achieves slightly higher throughput than random mapping since it actively avoids nodes with unacceptably high stress, but they are still comparable in terms of averages. The naive delay-minimizing heuristic performs the worst since it maps the VN in a confined area without any regard for node and link stress. The stress-balancing SBE performs considerably better than the other methods. Also, there is less variation in achieved throughput among different VNs (the highest average is 43% more

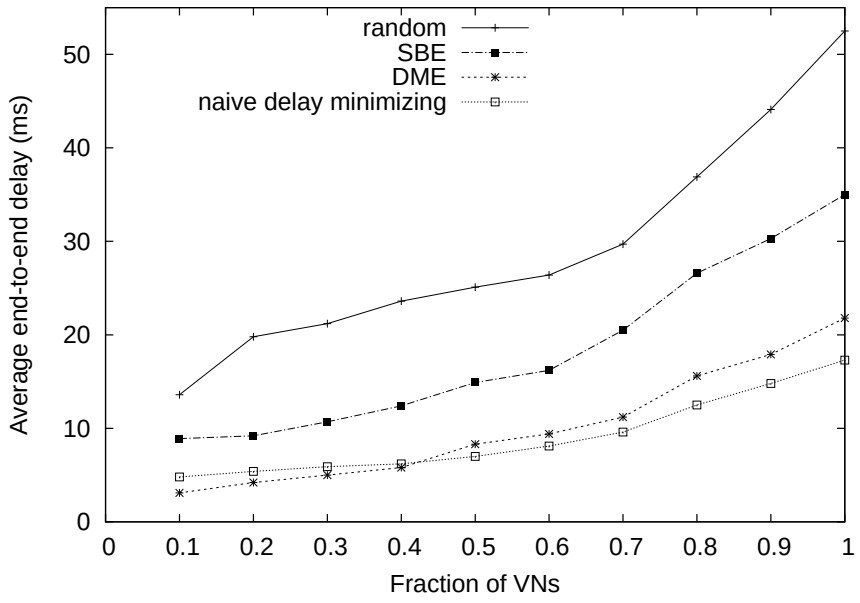


Figure 25: Average end-to-end delays for all VNs: SBE, DME, random mapping, and the naive delay minimizing heuristic.

than the lowest, as opposed to 57% for DME and 67% for random mapping), indicating a more balanced network. These results suggest that our techniques translate into improved performance in a realistic network emulation environment.

6.4.3.5 Varying the percentage of fixed-location controllers

In our model, we provide the option of fixing the location of the controller because a user may want to place the controller for her VN at a certain location, such as her home PC or a server at a specific data center. Assume there are $numV$ VNs to be placed, $numF$ with fixed-location controllers and the remaining $numV - numF$ with adjustable-location controllers. We run simulations with 5 different $numF/numV$ ratios: 0, 0.25, 0.5, 0.75 and 1. We use the same 20 VNs we have been using throughout this section and 3 substrate topologies for this experiment. For each topology, we run the experiment 10 times and average the values for each VN from all mappings.

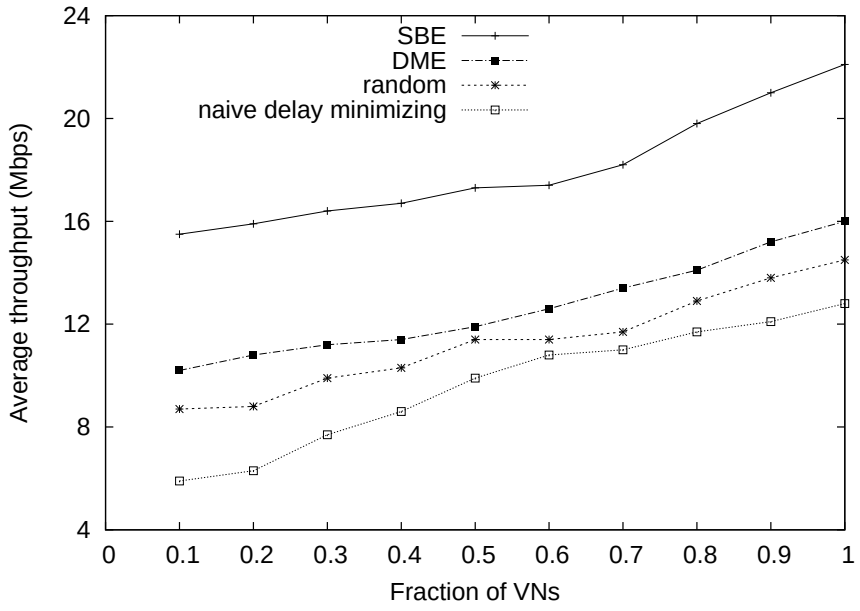


Figure 26: Average throughput for all VNs: SBE, DME, random mapping, and the naive delay minimizing heuristic.

We evaluate SBE and DME, both at their strong suits: minimizing average stress for SBE, and minimizing controller-to-switch delays for DME. We present results regarding four metrics: average and maximum node stress for SBE, and average and maximum controller-to-switch delay for DME. We do not give results for link stress separately because node stress results are sufficient in demonstrating the relevant trends.

Table 14 displays average and maximum node stress across the substrate achieved by SBE at 5 different values of $numF/numV$. We observe approximately 30% increase in average node stress and 85% increase in maximum node stress as the $numF/numV$ ratio goes from 0 to 1. An increase in stress is expected since some of the fixed-location controllers can be attached to nodes with low degrees and NR values that are suboptimal for stress minimization. The maximum stress is affected more because as the percentage of fixed-location controllers increases, so does the probability of an

unlucky combination of fixed controller locations that puts multiple controllers close to each other, thus making the occurrence of a high-stress node more likely. This impact becomes lessened for high $numF/numV$ ratios because the damage is already done by then.

Figure 27 and Figure 28 show average and maximum controller-to-switch delays for all VNs achieved by DME at 5 different values of $numF/numV$. For both metrics, a higher percentage of fixed-location controllers leads to higher delays due to decreased flexibility and suboptimal controller locations. However, this effect is more pronounced with the maximum delay values, and for smaller percentages (as seen from percentagewise bigger differences toward the bottom in Figure 28). A possible explanation of this situation is that with the large number of VNs on the substrates, fixing more of the controllers has a smaller effect on the average since some of the fixed controller locations may actually end up being suitable by chance. On the other hand, fixing the location of another controller increases the probability of hitting a bad controller location (i.e., one that is not suitable for delay minimization), so maximum delay values rise faster for low $numF/numV$ ratios. But as the ratios get larger, the probability of having already hit the worst controller location also increases, so maximum delays do not rise as fast for high $numF/numV$ ratios.

Therefore, the takeaway from this exercise is that

1. The variation in the percentage of VNs with fixed controller locations has a bigger effect on maximum values than averages, particularly maximum controller-to-switch delays,

2. For low percentages, the negative effects that come with fixed-location controllers are mild.

3. For high percentages (75% and up), the flexibility that comes from adjustable-location controllers has negligible positive impact.

Table 14: Average and maximum node stress with varying $numF/numV$ ratios for SBE

	Avg node stress	Max node stress
$numF/numV = 0$	10.8	17.1
$numF/numV = 0.25$	11.9	21.2
$numF/numV = 0.5$	12.8	26.7
$numF/numV = 0.75$	13.6	30.8
$numF/numV = 1$	14.3	31.6

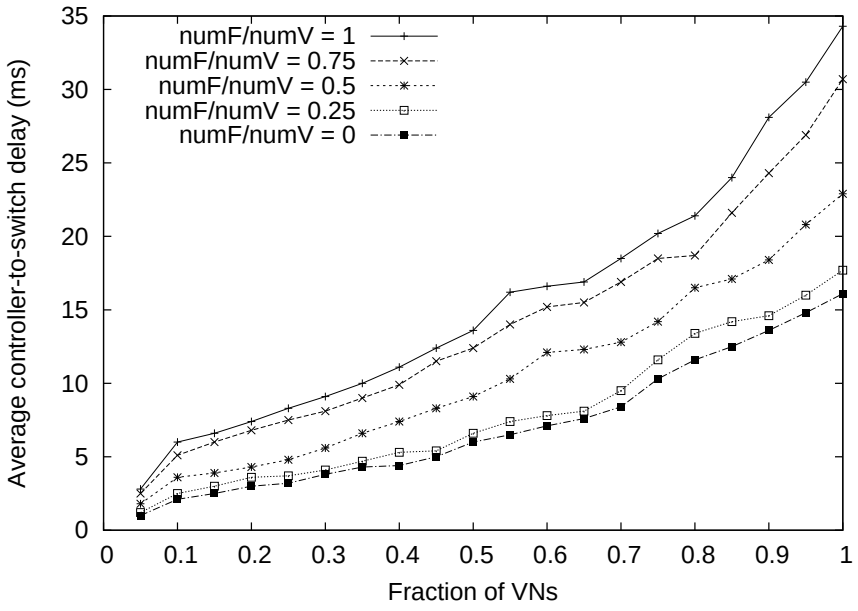


Figure 27: Average controller-to-switch delays for all VNs: DME with varying $numF/numV$ ratios

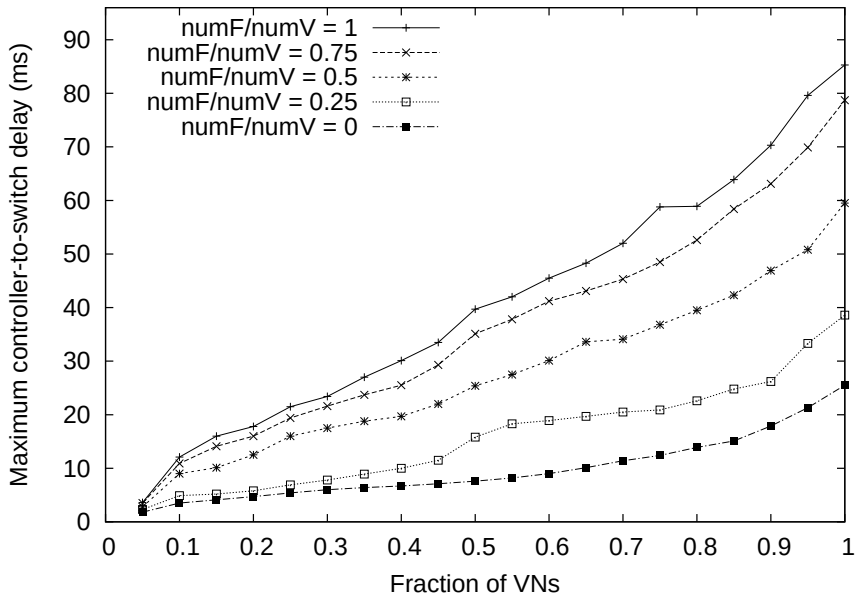


Figure 28: Maximum controller-to-switch delays for all VNs: DME with varying $numF/numV$ ratios

6.4.3.6 Varying T and γ

In order to gain insight into how the stress threshold T affects the results produced by DME, we look at different levels of T. For our previous experiments, we had set the two components of T, T_N and T_L, to 50% more than S_{Nmax} and S_{Lmax} produced by SBE. Let us define a *stress allowance multiplier*, denoted by σ, such that $T_N = \sigma \cdot S_{Nmax}$ and $T_L = \sigma \cdot S_{Lmax}$. So far, we have used $\sigma = 1.5$, and we now evaluate a few other values for σ.

Table 15 summarizes our findings. We only look at average and maximum controller-to-switch delays because stress values are directly dependent on σ so it is not very interesting to study stress in this case. Values are averaged over 20 VNs placed over 3 substrates, so the maximum delays here are average-maximum delays. When $\sigma = 1$, there is a tight control over stress, which allows less freedom in the effort to minimize

delays, which rise by about 50% on average compared to the case when $\sigma = 1.5$. Beyond this, the gains are marginal. For $\sigma = 2$ and $\sigma = 3$, there is no significant reduction in delays, except for a few cases where maximum delays can be reduced thanks to the extra degree of freedom. Therefore, $\sigma = 1.5$ appears to be a reasonable choice for DME, and we use that for every other experiment in this evaluation.

Another parameter that we experiment with is γ in Equation 23 defining link stress in Section 6.2. The definition of link stress affects both mapping techniques and potentially all metrics, but we are presenting the effect we found the most interesting. For this simulation, we assume γ is equal for all VNs, and evaluate 5 different γ values. Table 16 presents the controller-to-switch delay results, again averaged for 20 VNs placed over 3 substrates. We see that the delays increase with increasing γ, but a bit more slowly after $\gamma = 0.5$. This effect is probably due to the fact that when γ increases, control traffic and thus the locations of the controllers (and the links around it) become more critical in determining link stress. This takes away some of the ability to move the controllers freely to minimize delays because if a controller is moved to a low-NR node, it will be difficult to keep link stress within bounds. Hence, stress becomes the dominant factor in controller placement, causing delays to rise. But they do not rise too much because even with less flexible controller locations, DME is still a delay minimizing heuristic and it still has flexibility in virtual node placement to keep delays as low as possible.

6.5 Summary

In this work, we have studied techniques to perform VN mapping on an SDN substrate. The goals of our techniques were load balancing between switches as well as delay minimization between controllers and switches. Each technique focuses on optimizing for one of these goals while keeping the other one in check. Both techniques

Table 15: Delays with varying T for DME

	Avg delay (ms)	Max delay (ms)
$\sigma = 1$	12.37	24.44
$\sigma = 1.5$	8.06	15.16
$\sigma = 2$	7.72	13.89
$\sigma = 3$	7.65	13.89

Table 16: Delays with varying γ for DME

	Avg delay (ms)	Max delay (ms)
$\gamma = 0$	7.49	12.62
$\gamma = 0.25$	8.95	16.04
$\gamma = 0.5$	10.76	23.12
$\gamma = 0.75$	11.82	25.01
$\gamma = 1$	11.91	25.47

couple VN controller placement with virtual node and link assignment. Various input parameters in our definitions and techniques determine the extent of flexibility in embedding efforts and the range of improvements in evaluation metrics.

The main takeaways from this chapter are the following:

1. There is a tradeoff between minimizing controller-to-switch delays and balancing the load on substrate nodes and links, however, these two goals are not in direct conflict with each other. While it may not be possible to optimize for both goals at the same time, it is quite possible to achieve near-optimal results for one metric while keeping the other metric within reasonable bounds.

2. The presence of the central controllers in SDNs as critical nodes has a significant impact on VN embedding objectives and the locations of controllers must be selected carefully in order to better optimize for performance goals.

CHAPTER VII

CONCLUSIONS AND FUTURE WORK

7.1 Research Summary and Contributions

This thesis tackles the problem of facilitating the provision of auxiliary support services for overlay networks. The focus is on overlay assignment, resource allocation, monitoring and fault diagnosis. Overlays need these services to continue healthy operation, and careful design choices need to be made to ensure that they are provided efficiently and without excessive overhead. The concerns and priorities governing these design choices change depending on the environment and circumstances. For example, a substrate with scarce resources populated with resource-hungry overlays necessitates effective techniques for fair resource allocation and load balancing, whereas a well-provisioned network occupied by a few overlays may justify priority being given to diagnosability.

The contributions of this thesis can be summarized as follows:

Methods for overlay assignment in order to improve the ease and accuracy of fault diagnosis

Fast and accurate fault diagnosis is an important requirement for networks. The diagnosis method, the network topology, the likelihood of faults and the amount of traffic are all factors in determining how diagnosable a network is. For overlays, the placement of the overlay on the substrate also affects its diagnosability. Thus, improving diagnosability can be a placement goal, especially if the substrate is rich in resources and other goals such as load balancing are not of paramount importance.

We define diagnosability in terms of two properties of diagnosis: accuracy, representing the success rate in identifying the faulty component(s), and efficiency, representing the ability to identify faulty component(s) with minimal effort. These two properties are combined to define *diagnosis cost*: A lower diagnosis cost signifies higher diagnosability. We then develop a method for overlay placement to improve diagnosability by increasing meaningful sharing between overlay links in a controlled manner. Sharing helps diagnosability because in case of a fault, the number of end-to-end paths observing the fault is important in determining the exact location of the fault with high confidence. Our method increases this number through more sharing, but does this in a controlled fashion: 1) It enforces a maximum stress threshold to make sure that it balances the increase in sharing across the overlay and does not keep piling on a few substrate components, which can cause performance and reliability problems, and 2) it avoids meaningless sharing (such as near-complete overlap) which does not produce any additional valuable information for fault localization.

We utilize an existing passive diagnosis method based on aggregating end-user observations and evaluate the parameters of our design. We also analyze situations where overlay traffic is insufficient to produce useful observations on some links, and show that additional selective active probing on a few paths can help diagnosis considerably. Furthermore, we investigate the potential effect of our placement method on the robustness of the overlay. While the method causes more end-to-end paths within an overlay to be affected by a fault in the substrate, this may not be a disadvantage when there are many overlays on the substrate. In this case, our method does not cause a significant increase in the stress of substrate components. In addition, since it tends to make overlays more compact and separate them from each other, a fault may affect more paths within a single overlay but fewer overlays in total.

A novel framework to evaluate the fairness of resource sharing in the presence of multiple competing overlays

When substrate resources are unable to satisfy all overlay demands, fair allocation of resources becomes critical. An important resource that often causes bottlenecks is substrate link bandwidth. Simplistic fairness metrics such as Jain's fairness index [42] can be sufficient for evaluating the fairness of rate allocation between a few flows, however, evaluating an allocation among multiple overlay networks with varying sizes and resource demands requires a more refined approach. To this end, we develop a new fairness metric based on the idea that bandwidth sharing should inconvenience different overlays to the same extent if the allocation of rates is fair. In short, we focus on the parts of the substrate where multiple overlays are sharing bandwidth and calculate a satisfaction score for each overlay by judging how well it is treated by the allocation relative to the other overlays. Lower deviation among these scores indicates better fairness.

We adapt a few fairness definitions from the literature to the multi-overlay setting, and evaluate them with our metric. We demonstrate ways to treat overlays differently by tweaking certain parameters in fair allocation algorithms, and show that our fairness metric is able to capture unfair treatment with high sensitivity.

An examination of the effect of routing on the efficiency and fairness of bandwidth allocation

Fairness depends not only on the rate allocation, but also on the network topology and routing. We present a heuristic to fine-tune the routing in order to improve substrate bandwidth utilization and help more overlay flows receive rates closer to their demands. Our method aims to eliminate bottleneck situations if possible, or reduce the number of flows affected by bottlenecks. Evaluation shows that the heuristic improves rates, especially in non-homogeneous substrates. In addition, eliminating some bottleneck situations has a positive effect on the fairness of rate allocation, and

is ultimately a more reliable method to improve fairness than tweaking fair allocation algorithms.

An optimization solution to the problem of multi-layer overlay monitoring

We concentrate on the problem of measuring end-to-end delays for all overlay links in a given overlay network via active monitoring. The naive method of monitoring every overlay link directly burdens the network with redundant probing traffic, and this may cause serious problems if resources such as link bandwidth and CPU power are at a premium. Thus, we identify alternative solutions: monitoring all native links, monitoring a basis set [16] of overlay links, and our generalized approach of monitoring an optimal mix of native and overlay links (multi-layer monitoring). We formulate an integer linear program (ILP) to determine the optimal multi-layer monitoring mix, where the objective is to minimize the total cost of monitoring while providing sufficient monitoring information. This cost can be defined in different ways. Two simple definitions we consider are: the total number of probes, which represents the effort required to set up monitors, and the total amount of bandwidth consumed by probes, which represents the overhead experienced by the substrate for carrying monitoring traffic.

Evaluation with representative topologies suggests that the optimal mix almost always contains links from both the native and the overlay layer, justifying the multi-layer approach. We observe that if the substrate topology or the overlay placement causes high sharing between overlay links, the average monitoring cost per overlay link is reduced, but this trend does not continue forever. Similarly, as the fraction of substrate nodes that are selected as overlay nodes increases (i.e., as the overlay gets bigger on the substrate), the average cost per overlay link decreases. This is because the ILP is able to take advantage of the overlap between end-to-end paths to eliminate redundant measurements. A single measurement becomes more valuable

as it can be used in computing more end-to-end values in networks with high sharing between overlay links.

Inferring some end-to-end measurements from other native and overlay layer measurements may introduce an estimation error. We characterized these errors through experiments on PL-VINI, the VINI [10] prototype running on PlanetLab. After our analysis, we conclude that the multi-layer monitoring solution produces the lowest monitoring cost while introducing a small amount of error. The overlay basis set solution achieves slightly lower errors at the expense of higher cost, and the all-native solution minimizes the bandwidth consumed by probes, which is one of the possible cost definitions, but results in higher inference errors. Another observation is that in most cases, a large component of the total error is produced by a few inferred overlay link measurements. Once these links are identified, the multi-layer monitoring system can be instructed to always monitor them directly so that errors will be reduced significantly. Overall, the multi-layer monitoring approach is very flexible, and it produces minimal-cost solutions with low errors under different cost models and explicit constraints.

Design and analysis of techniques for mapping virtual networks to SDN substrates

Virtualization and SDN is a promising combination to facilitate innovation in networking. Each virtual network (VN) on an SDN substrate has its own controller which must communicate with all the nodes in the VN. This communication must not be hampered by long delays between the controller and virtual nodes, so it is important to incorporate delay-minimizing as an objective to VN assignment efforts in the SDN environment. We consider a stress-balancing and delay-minimizing as joint objectives. We modify definitions of node and link stress from previous work [96] to reflect the distinctions of virtualization in the SDN environment.

We develop two VN embedding techniques with different objectives: The goal of

the first technique is to balance the stress on the substrate while keeping all controller-to-node delays under a threshold. The goal of the second technique is to minimize controller-to-node delays while limiting the maximum stress. Evaluation suggests that it is possible to achieve good results even with delay and/or stress constraints. The primary objective can be chosen as low delays or balanced stress depending on network conditions. There are several decision parameters in our design, both in the definitions of node and link stress, and in the constraints that are placed on the techniques. We allow controller locations to be fixed or adjustable, and observe that the techniques are able to achieve considerably better results when they are allowed to select controller locations freely for a majority of the VNs.

7.2 Future Directions

7.2.1 Overlay Reconfiguration Policy Design

A multi-overlay system may experience faults and performance issues that prevent successful operation. It is important to avoid these issues when possible, and react to them effectively when avoidance proves insufficient. *Dynamic overlay reconfiguration* is a tool used to ensure that overlays are in a state facilitating smooth operation as well as fast detection and recovery from problems. Future work on this topic should investigate the problem of finding the optimal reconfiguration policy for a system of multiple overlay networks in the presence of limited resources and possible failures.

Researchers have proposed overlay reconfiguration policies that allow overlays to adapt to changing conditions. In the DaVinci paper [36], the authors propose a system where each substrate link periodically reallocates its bandwidth among the virtual links on it, and each virtual network maximizes its own performance objective, in order to achieve optimal sharing to maximize aggregate performance. The reconfiguration we consider involves not only reallocation of bandwidth but also rerouting and reassignment of overlay network nodes. In another work, Fan and Ammar [28]

propose an array of reconfiguration policies where the goal is to minimize the cost of data delivery. Another possible goal of overlay reconfiguration is to optimize the end-user experience by reacting to network faults and performance problems in a timely and inexpensive manner.

The biggest challenge in determining a reconfiguration policy is finding a balance between quick response to problems and low overhead on the network. Reconfiguration actions such as node and link movements consume CPU resources and network bandwidth, so an overly frequent or sensitive reconfiguration policy may cause intolerable overhead. On the other hand, infrequent reconfiguration may be insufficient in rectifying observed problems. The goal must be developing techniques to find optimal reconfiguration policies that define when reconfiguration will happen and what it will entail, depending on the relevant information obtained through monitoring of the system.

There are several aspects of an overlay reconfiguration:

- *Type*: How often will reconfiguration happen, will it be proactive or reactive?
- *Scope*: Where will reconfiguration happen, will it involve a subset of overlays or all of them?
- *Actions*: What will reconfiguration entail (reassignment of nodes, rerouting, reallocation of resources)?
- *Triggers*: If reconfiguration is reactive, what will trigger it?

The collection of answers to such questions forms a *reconfiguration policy*. The first decision to be made is about the *type* of reconfiguration. It is possible to utilize a strictly periodic or strictly reactive approach, but these approaches can also be combined into a hybrid policy where periodic reconfiguration is done as a preemptive step in order to reduce the probability of future problems, and reactive reconfiguration

is employed in case of a reconfiguration *trigger*. Triggers include such events as a node joining or leaving an overlay, an overlay joining or leaving the system, a faulty node or link, or performance issues such as high delay or loss rates observed by end users. Each trigger will lead to an *action*: resource reallocation, rerouting, node reassignment etc. It is necessary to prioritize these actions for each trigger to react in the most suitable way and to avoid superfluous actions. For instance, if a single faulty node can be avoided by rerouting paths away from it to alternate paths, this action must have precedence and a complete reassignment of nodes (i.e., moving the entire overlay) must be avoided if possible.

A possible approach is to sort reconfiguration actions by complexity and determine an order of precedence for each trigger event. Given a budget for reconfiguration over a time period, overlay performance can be optimized within this budget by choosing appropriate actions. It would be useful to employ periodic, reactive and hybrid approaches and compare these to each other.

There are at least two possible ways to evaluate the performance of a reconfiguration policy:

- Comparing end user experience by considering end-to-end metrics such as delay and loss rates over a period of time,
- Comparing a specially defined *overlay satisfaction* metric that depends on the amount of data an overlay is able to transfer.

The choice will determine the objective of performance optimization. In summary, potential contributions from this work can be listed as follows:

- Formulating definitive objectives for overlay reconfiguration,
- Classifying reconfiguration triggers and corresponding actions,

- Developing a comprehensive reconfiguration policy that matches actions to objectives and selects a suitable schedule.

7.2.2 Dynamic VN Remapping in SDNs

VN migration [47,58], based on live virtual router migration [89], has been proposed as a tool to make VNs more adaptable, reduce operating costs and improve security [41]. A systematic approach to determine when and where to move VNs in lacking in the literature. We touched upon the decision of when to move in the previous section using the concept of triggers. Once VN migration is triggered, the decision of where to move must be made carefully. The destination must be one that eliminates the conditions that led to triggering the migration, but at the same time, it should not cause a cascade of migrations, and it should be chosen with regard to the cost of the migration. Studying this problem in the SDN realm is a promising future work direction.

7.2.3 Multi-layer Monitoring in SDNs

SDN provides new opportunities to improve network monitoring and management [49]. Applying the multi-layer monitoring approach to SDN is an interesting new dimension. Our optimization formulation would still apply to the SDN case after specifying a reasonable cost model for SDN monitoring. Mininet offers a good evaluation environment where one could experiment with a wide variety of custom topologies to better understand which topologies provide the best opportunities in terms of cost-saving for multi-layer monitoring.

A useful extension to this work would be to incorporate *dynamic cost modeling*. Originally, our multi-layer monitoring scheme works with a predetermined cost structure. For instance, the cost of monitoring an overlay link directly can be a fixed value, *cost*. Alternatively, it could be *cost* plus the number of substrate links in the overlay link times a coefficient, to account for the bandwidth overhead on substrate

links. Once a cost structure is determined, the optimization works to minimize the total cost of monitoring the overlay. In dynamic cost modeling, cost definitions can be modified in response to changing network conditions. For example, in the initial cost structure, monitoring an overlay link will have a fixed cost. But suppose available bandwidth diminishes after some time, then the cost definition can be changed to include a component for the number of links in an overlay link, in order to punish probes that consume bandwidth on more substrate links. After this change, the optimization will be executed again to determine the updated monitoring solution. In short, dynamic cost modeling would give multi-layer monitoring the ability to be responsive and produce optimal solutions throughout the lifetime of the overlay.

7.2.4 Diagnosing Soft Faults in Overlays

Our work on overlay assignment for diagnosability assumes that the goal of diagnosis is to detect and localize persistent faults clearly observable by different end users. A useful extension would be to consider soft degradations and performance problems. Such problems are not necessarily observable by different users in a consistent manner because each user might have different expectations from the network in terms of delay and bandwidth requirements or the levels of tolerable loss rate. Hence, a refined diagnosis method for dealing with such faults, as well as an analysis of how the existence of such faults affect the usefulness of our overlay assignment technique and potential changes to the assignment approach are worth studying.

REFERENCES

[1] Adams, A., Bu, T., Caceres, R., Duffield, N., Friedman, T., Horowitz, J., Presti, F. L., Moon, S. B., Paxson, V., and Towsley, D., "The use of end-to-end multicast measurements for characterizing internal network behavior," *IEEE Communications Magazine*, vol. 38, no. 5, pp. 152–159, 2000.

[2] Agrawal, S., Naidu, K., and Rastogi, R., "Diagnosing link-level anomalies using passive probes," in *INFOCOM*, 2007.

[3] Ahuja, S. S., Ramasubramanian, S., and Krunz, M., "Srlg failure localization in optical networks," *IEEE/ACM Trans. Netw.*, vol. 19, pp. 989–999, Aug. 2011.

[4] Ahuja, S. S., Ramasubramanian, S., and Krunz, M. M., "Single-link failure detection in all-optical networks using monitoring cycles and paths," *IEEE/ACM Trans. Netw.*, vol. 17, pp. 1080–1093, Aug. 2009.

[5] "Akamai Technologies, Inc.." `http://www.akamai.com`.

[6] Andersen, D., Balakrishnan, H., Kaashoek, M. F., and Morris, R., "Resilient Overlay Networks," in *Proceedings of 18th ACM SOSP*, October 2001.

[7] Andersen, D. G., "Theoretical approaches to node assignment," *Computer Science Department*, p. 86, 2002.

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

[9] Batsakis, A., Malik, T., and Terzis, A., "Practical passive lossy link inference," in *Passive and Active Measurement Workshop (PAM)*, 2005.

[10] Bavier, A., Feamster, N., Huang, M., Peterson, L., and Rexford, J., "In vini veritas: realistic and controlled network experimentation," in *in Proc. ACM SIGCOMM*, pp. 3–14, 2006.

[11] Bertsekas, D. and Gallager, R., *Data Networks*. Prentice-Hall, 1987.

[12] "BitTorrent." `http://bramcohen.com/BitTorrent`.

[13] Byers, J. W., Considine, J., Mitzenmacher, M., and Rost, S., "Informed content delivery across adaptive overlay networks," in *In Proceedings of ACM SIGCOMM*, pp. 47–60, 2002.

[14] Chawathe, Y., "Scattercast: an adaptable broadcast distribution framework," *Multimedia Systems*, pp. 104–118, July 2003.

[15] Chen, Y., Bindel, D., Song, H., and Katz, R., "An Algebraic Approach to Practical and Scalable Overlay Network Monitoring," in *Proceedings of ACM SIGCOMM*, 2004.

[16] Chen, Y., Bindel, D., Song, H. H., and Katz, R. H., "Algebra-based scalable overlay network monitoring: algorithms, evaluation, and applications," *Networking, IEEE/ACM Transactions on*, vol. 15, no. 5, pp. 1084–1097, 2007.

[17] Chowdhury, N. and Boutaba, R., "A survey of network virtualization," *Computer Networks*, vol. 54, no. 5, pp. 862–876, 2010.

[18] Chowdhury, N. M. K., Rahman, M. R., and Boutaba, R., "Virtual network embedding with coordinated node and link mapping," in *INFOCOM 2009, IEEE*, pp. 783–791, IEEE, 2009.

[19] Chu, Y., Rao, S., and Zhang, H., "A Case for End System Multicast," in *Proceedings of ACM SIGMETRICS*, June 1999.

[20] Chun, B.-G., Fonseca, R., Stoica, I., and Kubiatowicz, J., "Characterizing selfishly constructed overlay routing networks," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 2, pp. 1329–1339, IEEE, 2004.

[21] Clark, D., Lehr, B., Bauer, S., Faratin, P., Sami, R., and Wroclawski, J., "Overlay Networks and the Future of the Internet," *COMMUNICATIONS & STRATEGIES*, no. 63, 2006.

[22] Cooper, B. F., "Trading off resources between overlapping overlays," in *Middleware '06: Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware*, (New York, NY, USA), pp. 101–120, Springer-Verlag New York, Inc., 2006.

[23] "CPR: Campus Wide Network Performance Monitoring and Recovery." `http://www.rnoc.gatech.edu/cpr`.

[24] Drutskoy, D., Keller, E., and Rexford, J., "Scalable network virtualization in software-defined networks," *Internet Computing, IEEE*, vol. 17, no. 2, pp. 20–27, 2013.

[25] Duan, Z., Zhang, Z., and Hou, Y., "Service overlay networks: Slas, qos, and bandwidth provisioning," *IEEE/ACM Trans. Netw.*, vol. 11, pp. 870–883, 2003.

[26] Duffield, N., "Network tomography of binary network performance characteristics," *IEEE Transactions of Information Theory*, vol. 52, pp. 5373–5388, 2006.

[27] Eriksson, H., "Mbone: the multicast backbone," *Commun. ACM*, vol. 37, pp. 54–60, Aug. 1994.

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

[29] Feamster, N., Gao, L., and Rexford, J., "How to lease the internet in your spare time," *SIGCOMM Comput. Commun. Rev.*, vol. 37, pp. 61–64, Jan. 2007.

[30] Feldmann, A., "Internet clean-slate design: what and why?," *SIGCOMM Comput. Commun. Rev.*, vol. 37, pp. 59–64, July 2007.

[31] Fischer, A., Botero, J., Beck, M., De Meer, H., and Hesselbach, X., "Virtual network embedding: A survey," *Communications Surveys Tutorials, IEEE*, vol. PP, no. 99, pp. 1–19, 2013.

[32] "Freenet." `http://freenetproject.org`.

[33] "GENI: Global Environment for Network Innovations." `http://www.geni.net`.

[34] "GNU Linear Programming Kit (GLPK)." `http://www.gnu.org/software/glpk`.

[35] Han, J., Watson, D., and Jahanian, F., "Topology aware overlay networks," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 4, pp. 2554–2565, IEEE, 2005.

[36] He, J., Zhang-shen, R., Li, Y., yen Lee, C., Rexford, J., and Chiang, M., "Davinci: Dynamically adaptive virtual networks for a customized internet," in *in Proc. CoNEXT*, 2008.

[37] Heller, B., Sherwood, R., and McKeown, N., "The controller placement problem," in *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, (New York, NY, USA), pp. 7–12, ACM, 2012.

[38] Ho, T., Leong, B., Chang, Y., Wen, Y., and Koetter, R., "Network monitoring in multicast networks using network coding," in *International Symposium on Information Theory (ISIT)*, 2005.

[39] "Internet2." `http://www.internet2.edu`.

[40] "Iperf - TCP and UDP bandwidth performance measurement tool." `http://iperf.sourceforge.net/`.

[41] Jafarian, J. H., Al-Shaer, E., and Duan, Q., "Openflow random host mutation: Transparent moving target defense using software defined networking," in *Proceedings of the first workshop on Hot topics in software defined networks*, pp. 127–132, ACM, 2012.

[42] Jain, R., *The Art of Computer Systems Performance Analysis*. John Wiley and Sons Inc., 1991.

[43] Jannotti, J., Gifford, D., Johnson, K., Kaashoek, F., and O'Toole, J., "Overcast: Reliable Multicasting with an Overlay Network," in *4th USENIX OSDI Symposium*, October 2000.

[44] Kamel, M., Scoglio, C., and Easton, T., "Optimal topology design for overlay networks," in *NETWORKING 2007. Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet*, pp. 714–725, Springer, 2007.

[45] Karbhari, P., Ammar, M., and Zegura, E., "Optimizing End-to-End Throughput for Data Transfers on an Overlay-TCP Path," in *Proceedings of Networking 2005*, May 2005.

[46] Karbhari, P., Zegura, E., and Ammar, M., "Multipoint-to-point session fairness in the internet," in *Proceedings of IEEE INFOCOM*, 2003.

[47] Keller, E., Ghorbani, S., Caesar, M., and Rexford, J., "Live migration of an entire network (and its hosts)," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, HotNets-XI, (New York, NY, USA), pp. 109–114, ACM, 2012.

[48] Keromytis, A., Misra, V., and Rubenstein, D., "SOS: Secure Overlay Services," in *Proceedings of ACM SIGCOMM*, 2002.

[49] Kim, H. and Feamster, N., "Improving network management with software defined networking," *Communications Magazine, IEEE*, vol. 51, no. 2, pp. 114–119, 2013.

[50] Kleinberg, J., Tardos, E., and Rabani, Y., "Fairness in routing and load balancing," in *FOCS '99: Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, (Washington, DC, USA), p. 568, IEEE Computer Society, 1999.

[51] Knight, S., Nguyen, H., Falkner, N., Bowden, R., and Roughan, M., "The internet topology zoo," *Selected Areas in Communications, IEEE Journal on*, vol. 29, pp. 1765 –1775, October 2011.

[52] Krishnamurthy, B., Wills, C., and Zhang, Y., "On the use and performance of content distribution networks," in *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pp. 169–182, ACM, 2001.

[53] Lantz, B., Heller, B., and McKeown, N., "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, (New York, NY, USA), pp. 19:1–19:6, ACM, 2010.

[54] Li, Z. and Mohapatra, P., "The impact of topology on overlay routing service," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 1, IEEE, 2004.

[55] Li, Z. and Mohapatra, P., "Qron: Qos-aware routing in overlay networks," *IEEE Journal on Selected Areas in Communications*, vol. 22, pp. 29–40, 2004.

[56] Li, Z. and Mohapatra, P., "On investigating overlay service topologies," *Computer Networks*, vol. 51, no. 1, pp. 54–68, 2007.

[57] Liu, Y., Zhang, H., Gong, W., and Towsley, D., "On the Interaction Between Overlay Routing and Traffic Engineering," in *Proceedings of IEEE INFOCOM*, 2005.

[58] Lo, S., Ammar, M., and Zegura, E., "Design and analysis of schedules for virtual network migration," in *Proceedings of IFIP Networking*, 2013.

[59] Lu, J. and Turner, J., "Efficient Mapping of Virtual Network onto a Shared Substrate," tech. rep., Washington University in St. Louis, June 2006.

[60] Lua, E. K., Crowcroft, J., Pias, M., Sharma, R., and Lim, S., "A survey and comparison of peer-to-peer overlay network schemes," *IEEE Communications Surveys and Tutorials*, vol. 7, no. 2, pp. 72–93, 2005.

[61] Madhyastha, H. V., Isdal, T., Piatek, M., Dixon, C., Anderson, T., Krishnamurthy, A., and Venkataramani, A., "iplane: An information plane for distributed services," in *OSDI*, 2006.

[62] Mazumdar, R., Mason, L., , and Douligeris, C., "Fairness in network optimal flow control: Optimality of product form," *IEEE Transactions on Communication*, vol. 39, pp. 775–782, 1991.

[63] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J., "Openflow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 69–74, Mar. 2008.

[64] Mussman, H., "Geni: an introduction." `http://groups.geni.net/geni/attachment/wiki/GeniSysOvrvw/GENI-AnIntroduction-28Feb2012.pdf`, February 2012.

[65] Nakao, A., Peterson, L., and Bavier, A., "A routing underlay for overlay networks," in *Proceedings of ACM SIGCOMM*, August 2003.

[66] "NOX." `http://www.noxrepo.org/`.

[67] Padmanabhan, V. N. and Qiu, L., "Network tomography using passive end-to-end measurements," in *DIMACS Workshop on Internet and WWW Measurement, Mapping and Modeling*, Citeseer, 2002.

[68] Peterson, L., Culler, D., Anderson, T., and Roscoe, T., "A Blueprint for Introducing Disruptive Technology into the Internet," in *Proceedings of ACM HotNets-I*, October 2002.

[69] "Planetlab." `http://www.planet-lab.org`.

[70] "Rocketfuel: An ISP Topology Mapping Engine." `http://www.cs.washington.edu/research/networking/rocketfuel/`.

[71] Savage, S., Anderson, T., Aggarwal, A., Becker, D., Cardwell, N., Collins, A., Hoffman, E., Snell, J., Vahdat, A., Voelker, G., and Zahorjan, J., "Detour: Informed internet routing and transport," *IEEE Micro*, vol. 19, pp. 50–59, Jan. 1999.

[72] Seetharaman, S. and Ammar, M., "Overlay-friendly Native Network: A Contradiction in Terms?," in *Proceedings of ACM HotNets-IV*, November 2005.

[73] Seetharaman, S. and Ammar, M., "On the Interaction between Dynamic Routing in the Overlay and Native Layers," in *Proceedings of IEEE INFOCOM*, April 2006.

[74] Seetharaman, S., Hilt, V., Hofmann, M., and Ammar, M., "Resolving cross-layer conflict between overlay routing and traffic engineering," *IEEE/ACM Trans. Netw.*, vol. 17, pp. 1964–1977, December 2009.

[75] Shafer, G., *A Mathematical Theory of Evidence*. Princeton University Press, 1976.

[76] Sherwood, R., Gibb, G., Yap, K.-K., Appenzeller, G., Casado, M., McKeown, N., and Parulkar, G., "Can the production network be the testbed?," in *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, (Berkeley, CA, USA), pp. 1–6, USENIX Association, 2010.

[77] Shi, E., Stoica, I., Andersen, D. G., and Perrig, A., "Overdose: A generic ddos protection service using an overlay network," *Computer Science Department*, p. 76, 2006.

[78] Simpson, W., "IP in IP tunneling." RFC 1853, October 1995.

[79] Stavrou, A., Cook, D. L., Morein, W. G., Keromytis, A. D., Misra, V., and Rubenstein, D., "Websos: an overlay-based system for protecting web servers from denial of service attacks," *Computer Networks*, vol. 48, no. 5, pp. 781–807, 2005.

[80] Stoica, I., Morris, R., Liben-nowell, D., Karger, D. R., Kaashoek, M. F., Dabek, F., and Balakrishnan, H., "Chord: a scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM Transactions on Networking*, vol. 11, pp. 17–32, 2003.

[81] Subramanian, L., Stoica, I., Balakrishnan, H., and Katz, R., "OverQoS: offering Internet QoS using overlays," in *Proceedings of ACM SIGCOMM*, August 2003.

[82] Tang, Y. and Al-Shaer, E., "Sharing end-user negative symptoms for improving overlay network dependability," in *IEEE IFIP International Conference on Dependable Systems and Networks (DSN)*, June 2009.

[83] Tang, Y., Al-Shaer, E., and Boutaba, R., "Efficient fault diagnosis using incremental alarm correlation and active investigation for internet and overlay networks," *Network and Service Management, IEEE Transactions on*, vol. 5, no. 1, pp. 36–49, 2008.

[84] Tapolcai, J., Wu, B., and Ho, P. H., "On Monitoring and Failure Localization in Mesh All-Optical Networks," in *Proceedings of IEEE INFOCOM*, April 2009.

[85] Touch, J., "Dynamic Internet Overlay Deployment and Management Using the X-Bone," *Computer Networks*, pp. 117–135, July 2001.

[86] Turner, J. and Taylor, D., "Diversifying the Internet," in *Proceedings of IEEE GLOBECOM*, 2005.

[87] Venkatasubramanian, V., Rengaswamy, R., and Kavuri, S., "A review of process fault detection and diagnosis. part i: Quantitative model-based methods," *Computers and chemical engineering*, vol. 27, no. 3, pp. 293–311, 2003.

[88] Vieira, S. L. and Liebeherr, J., "Topology design for service overlay networks with bandwidth guarantees," in *Quality of Service, 2004. IWQOS 2004. Twelfth IEEE International Workshop on*, pp. 211–220, IEEE, 2004.

[89] Wang, Y., Keller, E., Biskeborn, B., van der Merwe, J., and Rexford, J., "Virtual routers on the move: live router migration as a network-management primitive," in *ACM SIGCOMM Computer Communication Review*, vol. 38, pp. 231–242, ACM, 2008.

[90] Wu, B., Yeung, K. L., and Ho, P.-H., "Monitoring Cycle Design for Fast Link Failure Localization in All-Optical Networks," *Journal of Lightwave Technology*, vol. 27, pp. 1392–1401, May 2009.

[91] Wu, B., Ho, P.-H., and Yeung, K. L., "Monitoring trail: on fast link failure localization in all-optical wdm mesh networks," *Lightwave Technology, Journal of*, vol. 27, no. 18, pp. 4175–4185, 2009.

[92] Yu, M., Yi, Y., Rexford, J., and Chiang, M., "Rethinking virtual network embedding: substrate support for path splitting and migration," *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 17–29, March 2008.

[93] Zhao, B., Duan, Y., Huang, L., Joseph, A., and Kubiatowicz, J., "Brocade: Landmark routing on overlay networks," in *1st International Workshop on Peer-to-Peer Systems (IPTPS)*, March 2002.

[94] Zhou, L. and Sen, A., "Topology design of service overlay network with a generalized cost model," in *Global Telecommunications Conference, 2007. GLOBECOM'07. IEEE*, pp. 75–80, IEEE, 2007.

[95] Zhu, Y., Dovrolis, C., and Ammar, M., "Dynamic Overlay Routing Based on Available Bandwidth Estimation: A Simulation Study," *Computer Networks Journal (Elsevier)*, vol. 50, pp. 742–762, April 2006.

[96] Zhu, Y. and Ammar, M., "Algorithms for Assigning Substrate Network Resources to Virtual Network Components," in *Proceedings of IEEE INFOCOM*, 2006.

[97] Zhu, Y. and Ammar, M., "Overlay network assignment in PlanetLab with NetFinder," Tech. Rep. GT-CSS-06-11, College of Computing, Georgia Institute of Technology, August 2006.