DESIGN AND EVALUATION OF VIRTUAL NETWORK MIGRATION MECHANISMS ON SHARED SUBSTRATE

A Thesis
Presented to
The Academic Faculty

by

Sau Man Lo

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

School of Computer Science
Georgia Institute of Technology
December 2015

DESIGN AND EVALUATION OF VIRTUAL NETWORK MIGRATION MECHANISMS ON SHARED SUBSTRATE

To my maternal grandmother,

who shows me that women can do anything men can do,

my mom,

who has been a rock of stability throughout my life,

and

my dad,

whose loving spirit sustains me still

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my advisors Prof. Mostafa Ammar and Prof. Ellen Zegura for their guidance and support. They have always been patient with me. When I had thoughts about my work being fruitless or unimportant, they always encouraged me. They gave me the freedom to work on the topics I am interested in and taught me to grow into an independent researcher. I could not survive this PhD journey without them.

Besides my advisors, Prof. Ehab Al-Shaer and Prof. Marwan Fayed have been fantastic mentors. Ehab introduced me to network security. Through our collaboration, I learned a lot about not only security, but also how to look at my work in a broader perspective. Marwan gave lots of insightful suggestions for my work and writing. Through our discussions, he has been a useful sounding board. His passion working with students makes him a fun person to work with.

I am also very grateful to the rest of my committee members, Prof. Ling Liu and Prof. Jim Xu, for their constructive suggestions to my dissertation which widen my research from various perspectives. I also owe a great debt to Prof. Constantine Dovrolis, Prof. Nick Feamster, Dr. Russell Clark, and Prof. Jennifer Rexford for their discussions, encouragement, and suggestions on my research and career. I am also thankful to have Prof. Rocky Chang as my research advisor when I was an undergrad and MPhil student at The Hong Kong Polytechnic University. He introduced me to the world of networking research. All of their insights broaden my perspective on research. I learned a lot from them.

I am thankful to have Dr. Srini Seetharaman, Dr. Mehmet Demirci, and Fida

Gilani from UNC Charlotte to collaborate with. Srini and Mehmet laid a solid foundation on overlay network research for me to follow. It was great working with them. Fida's dedication to his work makes him a pleasure to work with. Alina Quereilhac from INRIA has been a great resource to me. Although I never meet her in person, she is still willing to put up with my continual questions about Vsys and PlanetLab. She always responses quickly, and I am thankful to her generosity.

The work in this dissertation would not be possible without PlanetLab and GENI. I would like to express my gratitude to the support team of PlanetLab and GENI Project Office including Dr. Andy Bavier, Dr. Vicraj Thomas, Sarah Edwards, Dr. Niky Riga, and the rest of the teams for their hard work on building and maintaining PlanetLab and GENI. Their dedication facilitates innovations and educations in computer networks.

I am fortunate to have a group of wonderful colleagues at Georgia Tech. In particular, Dr. Cong Shi, Dr. Ahmed Mansy, Dr. Yang Chen, Dr. Mukarram Bin-Tariq, Karim Habak, AliReza Khoshgoftar Monfared, Ahmed Saeed, Tarun Mangla, Yimeng Zhao, Dr. Hyojoon Kim, Dr. Srikanth Sundaresan, Dr. Robert Lychev, Dr. Shuang Hao, Dr. Sam Burnett, Dr. Valas Valancius, Dr. Murtaza Motiwala, Dr. Anirudh Ramachandran, Bilal Anwer, Yogesh Mundada, Maria Konte, Abhinav Narain, Sean Donovan, Swati Roy, Mi Seon Park, Sathya Gunasekaran, Dr. Walter de Donato, Dr. Ruomei Gao, Dr. Ravi Prasad, Dr. Amogh Dhamdhere, Dr. Partha Kanuparthy, Dr. Saeideh Bakhshi, Dr. Demetris Antoniades, Dr. Aemen Lodhi, Dr. Saamer Akhshabi, Ilias Foudalis, Danny Lee, Yizheng Chen, Dr. Mukil Kesavan, Dr. Chaitrali Amrutkar, Dr. Dave Lillethun, and Dr. Brian Railing for their friendship, support and advice.

A special thanks to my family and friends. My mother, Yuk-lin Lo, has always been my support throughout the journey of my life. My late father, Wah Lo, has been a constant source of love still. They encouraged me to pursue knowledge and wisdom as no one could possibly take these away from me. I am thankful to have them in my life. My grandmother has showed me that women can do anything men can do and has been an occasional source of financial support. I am thankful to my uncles and aunts for their endless support, as my winter break hosts, and occasional source of financial support. I am thankful to my mother- and father-in-law Jan and Bob for their love and support. My brothers Wai-keung and Wai-yin have always been the funny ones and supported me with lots of laughter. My best friend Alana Ho has been providing her support remotely through WhatsApp chats, phone calls, and care packages in the mail. I am grateful to have Grace Lee, Irene Lai, Jennifer Song, and Hyojung Jun as my best buddies in the US. I am also thankful to the cooking night and Hong Kong Student Association folks, Dr. Ken Chui, Dr. Jeff Wong, Wenny Liu, Dr. Adrian Lam, Dr, Samson Lai, Wes Wang, Dr. Ken Pradel for their yummy food on Saturday nights. Prof. Dana Randall and Grad Women@CC have been a great support for me. I am thankful for the congregation of St. Paul's Presbyterian Church for their prayers and support. Last but not least, I would like to express my appreciation to my beloved husband, Dr. Brian Gestner, who has been a great support and sometimes the last minute editor for my writing.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The Internet faces well-known challenges in realizing modifications to the core architecture. To help overcome these limitations, the use of network virtualization has been proposed. Network virtualization enables the deployment of novel network architectures and services on existing Internet infrastructure. Virtual networks run over physical networks and use Internet paths and protocols as essentially a link layer in the virtual network. Virtual networks can also share the resources in the physical substrate. Effective use of the underlying substrate network requires intelligent placement of virtual networks so that underlying resources do not incur over-subscription. Because virtual networks can come and go over time, and underlying networks can experience their own dynamic changes, virtual networks need to be migrated—remapped to the physical network during active operation—to maintain good performance. While virtual network placement, and to a lesser extent migration, has been studied in the past, little attention has been devoted to designing, deploying and evaluating migration mechanisms for virtual networks. In this dissertation, we design virtual network migration mechanisms for different substrate platforms and further design a system to mitigate the effects of virtual network migration. In particular this dissertation makes the following contributions:

- With the goal of minimizing the disruption during a virtual network migration, we design three algorithms for scheduling the sequence of virtual router moves that takes a virtual network from its original placement to its new placement.
- We design and implement a controller-based architecture for virtual network migration on PlanetLab. This work explores the challenges in implementing

virtual network migration on real infrastructure. Recommendations are given for infrastructure that support virtual network migration.

- We propose and implement a mechanism to mitigate performance degradation resulting from virtual network migration through transport and application layer collaboration. We utilize a centralized controller to notify the end-systems or the gateways about the time of the virtual network migration such that we prevent packet loss to the application traffic of the end-systems.

CHAPTER I

INTRODUCTION

The Internet has become a critical global infrastructure in the past twenty five years since its commercialization. The Internet has facilitated various innovative applications, such as electronic commerce, video streaming, video conferencing, and VoIP. There were more than 2.7 billion Internet users worldwide in 2013 [51] and the number of Internet users is increasing. As a result, there is a growing need for the Internet to evolve and support more users, protocols, and applications. However, it is difficult to change the architecture of the Internet because these changes require either upgraded hardware or software updates on existing routers and switches. It is also a challenge for Internet Service Providers (ISPs) and networking hardware vendors to agree on new architecture adoption and deployment. For example, Internet Protocol version 6 (IPv6) can address the insufficient number of IP addresses in the current Internet architecture. Despite this ability, the development and deployment of IPv6 has continued over a decade and only 4% of domain names on the Internet supported IPv6 as of September 2013. These difficulties and challenges hinder innovation in the Internet, resulting in the Internet ossification problem.

Network virtualization has been proposed to overcome the Internet ossification problem and enable innovations at the core of the Internet ([14, 18, 30, 55, 81]). Virtual networks (VNs) run on top of physical networks and use Internet paths and protocols as essentially a link layer in the VN. The structure of such networks is illustrated in Figure 1, where two VNs are sharing a common physical substrate network. On each substrate node, the figure indicates which virtual routers are instantiated; virtual links may traverse more than one physical substrate link and are denoted by dashed lines. This type of virtualization allows a substrate network provider to be decoupled from (and offer services to) VN providers. This form of virtualization is distinguished from overlay networks where end users establish a network by using physical paths between end systems [30]. We focus in this dissertation on the specific form of virtualization where virtual routers and switches are instantiated in physical router and switch hardware and where multiple virtual routers belonging to different virtual networks may share the same physical router [78].

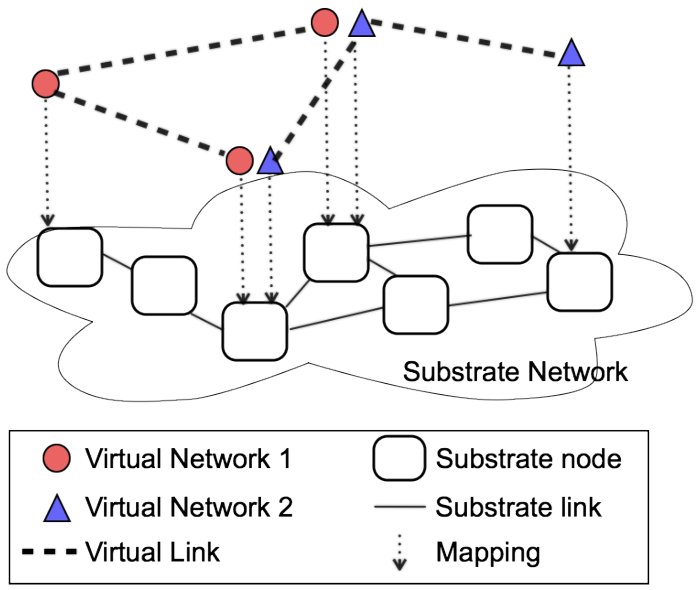


Figure 1: Two virtual networks map to a substrate network

VNs are attractive because they provide significant flexibility in operations. VN providers can easily map the VN requirements to physical network resources. Numerous studies have been conducted to investigate the effective use of the physical network by mapping VNs to physical substrate resources [91, 59, 90, 34]. Virtualization also allows flexibility in changing the mapping of a VN over time. Work in this area has explored reasons for network reconfiguration (or migration), such as changes in the traffic carried on the VN or on other VNs that share the same physical infrastructure [38]. Furthermore, the available substrate resources may change over time as VNs arrive or depart such that the VNs should be remapped [91, 59, 90, 39].

Additionally, VN migration can form the foundation of a "moving target defense" [12], where a VN evades detection or attack by changing its location in the physical network.

The focus of the work cited above is on questions of *policy* in VN migration. The VN migration policy determines the time to initiate the migration of an existing VN and the target new placement of the VN based on performance or security objectives. Our focus in this dissertation is on questions about the *mechanism* of actually moving the network.

A *migration mechanism* describes the deployment and execution of a particular VN migration strategy. More specifically, given the destination for a particular VN to be moved, the migration mechanism handles all the logistics of the movement including the exact sequence or schedule of steps for execution of the move. The migration mechanism is also responsible for interacting with the specific substrate virtualization technology to accomplish the move.

The process of moving virtual nodes and links can be a resource-demanding operation because it involves performing complex tasks [88]. VN migration is therefore a challenge because it must happen *live*, with a minimal interruption of the forwarding of network traffic; *efficiently*, by not consuming too many network resources; and *quickly*, by completing the move in a timely manner. The deployment of VN migration mechanisms also needs to be guided by desired performance objectives such as: 1) how to manage disruption to existing traffic on the VN, 2) what is the additional overhead incurred by the physical substrate while the migration is in progress and 3) how long the migration process lasts.

Another challenge for VN migration mechanism deployment is the consideration of the specific network virtualization technology. We can design VN migration mechanisms based on the virtualization technology of the switching, routing, and/or transport layers. To further deploy VNs across the Internet, we must also consider the effects of migration mechanisms to different layers including the disruption to the application layer. Furthermore, most of the substrate network do not provide features to support VN migration. These challenges cannot be neglected in developing VN migration mechanisms.

In the rest of this chapter, we first present a framework for VN assignment and migration, then we present the contributions of this dissertation.

1.1 Framework for VN Assignment and Migration

In this section, we present a framework for managing VNs and show the role of VN migration in this framework. Figure 2 shows the functional architecture of the VN management framework. When a VN provider receives a VN request, based on the required resources and topology requirements of the VN request, VN assignment policy of the VN provider, and the available substrate resources, the VN is assigned to a set of substrate resources. We called this assignment as the *VN placement*. Depending on the available resources of the VN provider, a VN can be assigned to resources in multiple substrate networks. After computing the VN placement, based on the virtualization technology of the substrate network(s) and the assigned VN placement, a VN orchestrator with the VN assignment mechanism deploys the VN on the substrate network(s).

Over time, the substrate resources can be fragmented or a VN's requirement changes, the VN requires a migration. Based on the migration or reconfiguration policy, with the given initial VN placement, and available substrate resources, the migration policy manager generates a final VN placement. The VN migration orchestrator with the corresponding migration mechanism takes that final placement, based on the virtualization technology of the substrate network, to orchestrate the VN migration.

In this dissertation, we focus on the mechanism for VN migration. We propose

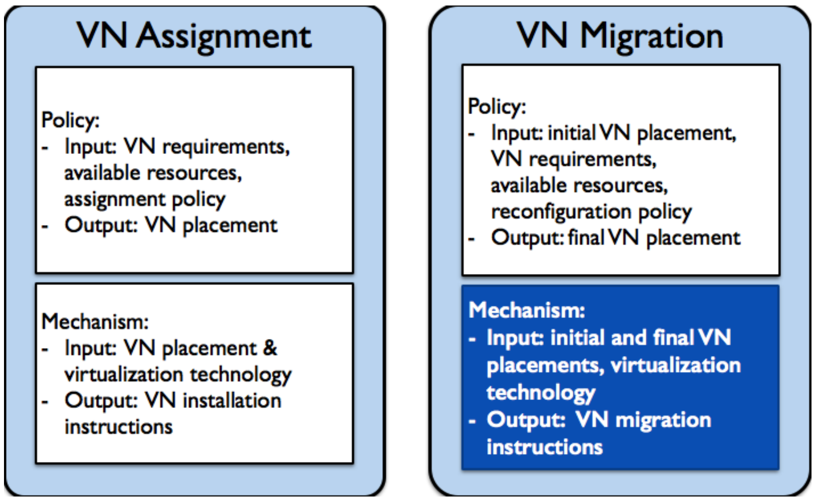

VN Assignment
Policy:
- Input: VN requirements, available resources, assignment policy
- Output: VN placement
Mechanism:
- Input: VN placement & virtualization technology
- Output: VN installation instructions
VN Migration
Policy:
- Input: initial VN placement, VN requirements, available resources, reconfiguration policy
- Output: final VN placement
Mechanism:
- Input: initial and final VN placements, virtualization technology
- Output: VN migration instructions

Figure 2: Functional architecture of VN management framework

algorithms for scheduling VN migration, design and implement a controller to facilitate migration, and propose a system architecture for mitigating the effects of VN migration. Figure 3 shows our VN migration mechanism framework. First, when the controller receives a VN migration request with the initial and final VN placement, based on the available network resources and the migration policy, a migration scheduler produces a schedule of the migration steps and gives the schedule to the VN migration orchestrator.

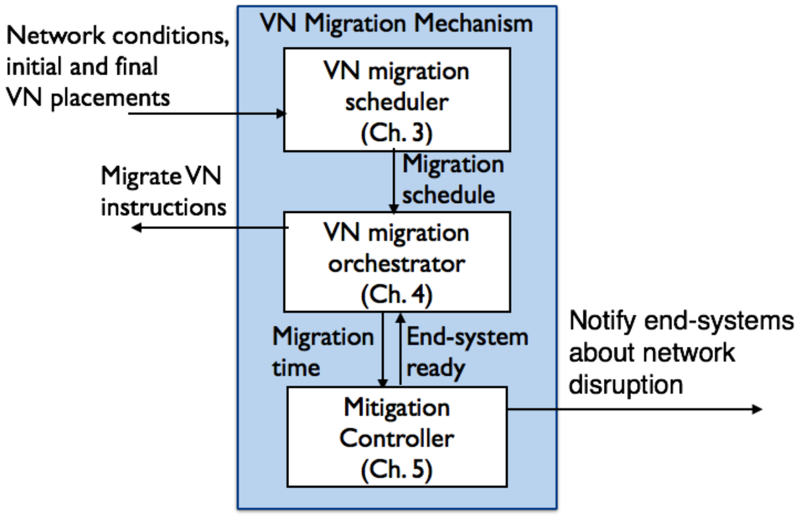

Network conditions, initial and final VN placements
VN Migration Mechanism
VN migration scheduler (Ch. 3)
Migration schedule
Migrate VN instructions
VN migration orchestrator (Ch. 4)
Migration time
End-system ready
Mitigation Controller (Ch. 5)
Notify end-systems about network disruption

Figure 3: VN migration mechanism framework

The VN migration orchestrator considers the schedule and further executes the migration based on the virtualization technologies that the substrate network is using. The orchestrator also provides feedback to the scheduler about the migration status and the actual new placement of the VN.

Before the migration orchestrator starts the migration, it also notifies the mitigation controller about the beginning and the end of the migration. The mitigation controller notifies the end-systems about the network disruption so that the end-systems can take actions to prevent packet loss. The VN migration orchestrator waits for the ready signal from the mitigation controller to start the VN migration.

1.2 Contributions

In this dissertation, we design and implement these three elements of VN migration mechanism. Figure 4 shows that our work in this dissertation spreads over different areas in a two dimensional space. Vertically we classify our work based on the migration control done on the layer 2, 3, 4, or above. Horizontally we classify our work among the ideal time control case under simulation, the emulation under perform environments, and the reality case in the real implementation on PlanetLab. We present a brief description of our work in this section.

1.2.1 Algorithms for VN Migration Scheduling

This work starts by leveraging prior work that considers the design of a *single virtual router* live migration mechanism [88]. The objective of this prior work is to perform virtual router migration such that the migration does not disrupt current data flows. This prior work operates on layer 2 and 3. We build on this single virtual router migration mechanism (outlined in more detail in the next section) and use this mechanism as a subroutine in the live migration of an *entire* VN on both the switching and routing layers. Specifically, given a new target placement within the substrate

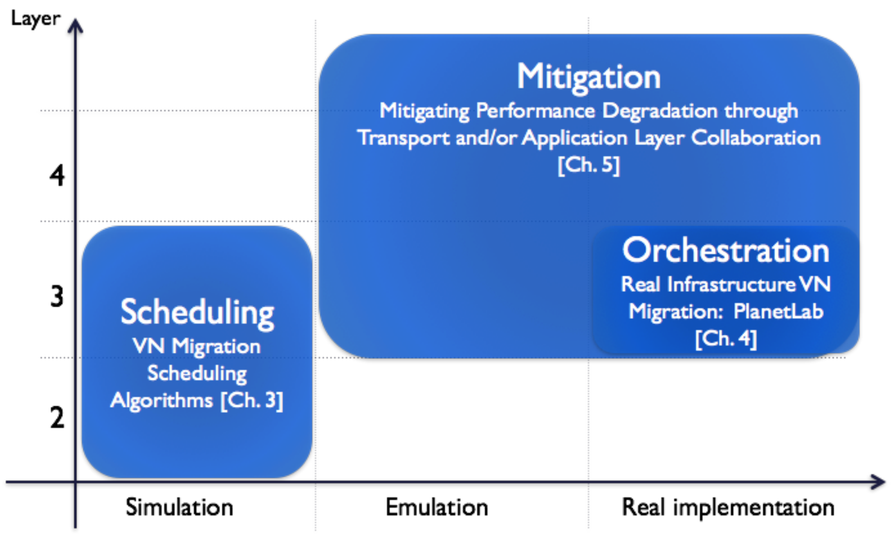

Layer
4
3
2
Mitigation
Mitigating Performance Degradation through Transport and/or Application Layer Collaboration [Ch. 5]
Orchestration
Real Infrastructure VN Migration: PlanetLab [Ch. 4]
Scheduling
VN Migration Scheduling Algorithms [Ch. 3]
Simulation
Emulation
Real implementation

Figure 4: Contributions of this dissertation

for a VN, we are interested in determining the best *schedule* of virtual router migrations that implement the desired VN move. We maintain the goal of minimizing the disruption to the current network traffic during the migration process and consider how to design a schedule that minimizes the overhead (cost) of network migration or the time it takes to complete the migration task. This work is the first we are aware of that considers the question of scheduling a live VN migration.

1.2.2 A Controller-based Architecture for VN Migration on PlanetLab

We study the VN migration problem on PlanetLab [70]. PlanetLab is a prominent virtualized infrastructure for deploying multiple VNs that share the substrate network resources distributed across hundreds of physical domains on the Internet. A user requests a slice on PlanetLab and selects a set of PlanetLab nodes to add to the slice. Once the user selects a set of nodes for the experiment, the user can connect these nodes together to form a VN within the slice. However, some of the nodes may fail or become unstable over time [67]. This can have a negative impact, for example, on experimental results, as well as the ability to experiment in the long-term. An effective virtual network migration system could mitigate negative effects on PlanetLab and similar platforms while facilitating application development.

A VN migration mechanism should minimize any disruption introduced to the VN. It is also desirable that the process of VN migration be automated and relatively fast. We propose a migration mechanism on the routing layer for VN migration on PlanetLab designed to meet these objectives. We design, implement, and evaluate with live experiments, a controller called *PL-VNM* that orchestrates the VN migration to new VN placements on PlanetLab. In principle, we can schedule the migration steps to minimize the disruption to the VN. In practice, our experiments have revealed that certain PlanetLab features make VN migration scheduling infeasible. This work leads to specific recommendations for PlanetLab that would enable VN migration scheduling, as well as recommend features for more general virtualization environments. We believe that the recommended features make long-term experiments and application deployments on shared virtualized infrastructures possible.

1.2.3 Techniques to Mitigate Performance Degradation through Transport and/or Application Layer Collaboration

VN migration leads to unexpected packet loss and bandwidth degradation for the application running on the VN. The VN migration mechanism from our PlanetLab implementation introduces short term packet loss. However, this short term packet loss can trigger a long recovery for TCP throughput. Furthermore, any kind of VN migration changes the quality of the VN because of the unknown performance of the newly assigned substrate resources. In this work, we study the effect of VN migration to the transport layer and further design mitigating solutions to prevent performance degradation to the applications running on the VNs. We propose a mitigation solution that utilizes a controller to signal the end systems and gateways about the time of the VN migration such that the end systems can pause the data flow before the VN migration to prevent packet loss and further the shrinking of the TCP congestion window. Our mitigation solutions are designed to be on the transport layer and/or application layer depending on the virtualization environment. We implement our mitigation solutions on GENI, which can be an emulation and a real experiment environment, and show that our solution effectively mitigate the effects of network disruptions. Additionally, the effects of VN migration are similar to the effects of route oscillation in the Internet to the transport layer. Therefore, this work also can be applied to mitigate performance degradation caused by both VN migration and route oscillation.

1.3 Dissertation Outline

The remainder of this dissertation is organized as follows. In chapter 2 we review the previous work in different areas related to the work in this thesis. Chapter 3 presents our work on scheduling VN migration. A case study on PlanetLab on VN migration is presented in chapter 4. In chapter 5 we present our work on mitigating the effects of VN migration with application collaboration. Finally, we conclude this dissertation and provide future work in Chapter 6.

CHAPTER II

RELATED WORK

In this chapter we review some of the prior work that address issues on network virtualization.

2.1 Background on Network Virtualization Framework

Network virtualization enables a diverse network architectures to share a single physical substrate network [85, 40]. Turner and Taylor have proposed a network virtualization framework that the physical infrastructure is managed by infrastructure providers while the virtual networks built on top of the physical infrastructure are managed by the virtual network providers [85]. Each virtual network provider can implement its own placement policy. These providers implement different virtualization techniques to support virtual networks. This framework has built a foundation on network virtualization and resource management on these networks.

In this section, we present the work in the area of VN embedding problems and reconfiguration policies. With these different embedding algorithms and reconfiguration policies, physical infrastructure providers and VN providers can make use of the available virtualization technologies to implement VNs.

2.1.1 VN Embedding and Placement Policy

One of the challenges in network virtualization is the VN embedding problem [41]. VNs are embedded in a substrate network to share the resources. The resources include the resources of the physical nodes and links. When a virtual network becomes the substrate network, i.e., nested virtualization, the resources of the virtual nodes and links are considered. The virtual nodes and links of a VN are assigned to some physical nodes and paths that composed of physical links, respectively. Resources can be allocated dynamically based on the available resources of the substrate network, the requirements of the VNs, and the goal of the placement policy. Numerous studies have been conducted to investigate the effective use of the physical network by mapping VNs to physical substrate resources [91, 59, 90, 34]. The policy can have different objectives such as QoS [84], economical profit [32], power consumption [26, 31, 36], security of the VNs [55]. The VN embedding problem is $\mathcal{NP}$-hard [13].

2.1.2 VN Reconfiguration Policy

Some of the VN embedding solutions suggest reconfiguration of the VNs or the mapping of the VNs to prevent resource fragmentation and increase the revenue of the infrastructure. In [28, 82], algorithms have been proposed for reconfiguring VNs by considering that the VNs evolve. Fajjari et al. has proposed an algorithm for VN reconfigurations [37] by migrating the virtual nodes that are connected to bottlenecked links. Through simulations, their reconfiguration scheme has reduced the rejection rate of the VN requests. Zhu and Ammar have proposed an algorithm to prioritize the VN reconfiguration and evaluated the algorithm through simulations [91]. Fan and Ammar have proposed heuristic methods for constructing reconfiguration policies to minimize the cost [38].

The problem of survivable VN embedding has been also studied [50]. These solutions included reserving more resources at the beginning of VN assignment or migrating virtual nodes or links. Crovella et al. and Guo et al. proposed a VN embedding policy that made used of allocating backup resources and rerouting traffic during link failures to improve the survivability of VNs [73, 46]. Yu et al. proposed to migrate the virtual nodes and links to prevent failure [89]. Both of these solutions have been proven by simulations to be effective to increase the survivability of VNs. The first type of solutions involved allocating resources that might be idle. The second type of solutions involved the overhead of migrating nodes or links. In this dissertation, we focus instead of the policy, on the mechanism of migrating VNs.

2.2 Virtualization Technology

2.2.1 System Virtualization

Virtual machine migration [33] has been proposed to facilitate live machine migration. Some techniques for virtual router migration are based on virtual machine migration technologies [88]. Virtual routers can be implemented on the virtual machines using commodity hardware [23, 35]. Mattos et al. have evaluated the performance of different virtualization tools including Xen [19], VMware [9], and OpenVZ [66] and concluded that OpenVZ, which is a container-based virtualization technology, introduces less overhead over CPU, disk, and memory usage, but Xen performs the best for implementing virtual routers based on scalability and fairness [61]. From this study, we can similarly understand the performance of PlanetLab because PlanetLab is using Linux server [8] and LXC [10] which are also container-based virtualization technology. With container-based virtualization, the overhead for network virtualization is minimized because the system calls in the virtual node share the operating system's normal system call interfaces without any emulation or another layer of virtual machine. As a result, it is suitable for implementing VNs on a shared substrate to minimize overhead. However, container-based virtualization systems cannot provide isolation as good as hypervisor based virtualization systems. Mattos et al. show that compare to Xen and VMware, OpenVZ performs the worst when the virtual router installed in the container is sending and receiving packets [61]. Xen performs the best because it only copies the packet to the memory once and then uses the memory location for handling the packet. In contrast, OpenVZ routes the packets between the host and the virtual container and the scheduling time of the host and the virtual containers also adds to the bad performance of receiving and sending packets.

2.2.2 Existing Virtual Software Router and Network Migration Solutions

Quagga [72], XORP [48], and Click modular router [29] have been used as software routers. Quagga can be installed on virtual machines and act us a router. However, because Quagga was not designed for migration, to migrate a Quagga virtual router, we rely on the virtual machine migration and the performance of the migration may not be ideal. Martins et al. proposed ClickOS to allow programmability on the data plane [60]. ClickOS is a very light weight OS runs on top of Xen hypervisor and is based on Click modular router. Its migration time includes stop, copy, and restart the virtual machine is within seconds.

A number of approaches are proposed to separate the control and data plane for migration through virtualization. Agrawal et al. have proposed migrating a router functionality to another router in the network for maintenance purpose [11]. Wang et al. have designed a single virtual router live migration mechanism that does not disrupt current data flows using the migrating router [88]. That work assumes a router architecture with multiple virtual routers instantiated on a physical router. Three features of virtual router architecture make it "migratable": independence of the virtual router through OS virtualization support, separation of data and control planes, and the ability to dynamically bind a router's data plane to physical substrate interfaces. However, this work only has addressed the single node migration problem.

Keller et al. have proposed live migration for the entire network based on SDN [52, 53]. That work focuses on layer 2 migration. The traffic flows and the end hosts are migrated by cloning the states in the data planes of the switches to another set of switches and redirecting traffic to the migrated end hosts. Their later work utilized the cloning framework to prevent packet loss during network migration on layer 2 and provided correctness to the applications [44]. However, packet reordering was still observed in their work.

2.2.3 Software-Defined Networking (SDN)

SDN provides a separation on the data plane and control plane, from layer two to the application layer. The current Software-Defined Networking (SDN) deployment is instantiated by OpenFlow protocols between the controller and the data plane of OpenFlow-enabled switches [65]. Each such switch has a control interface connected to a controller, and a data forwarding plane. The controller installs flow entries of specific flows, which are forwarding rules for those flows, to the forwarding planes of switches on the network.

SDN controllers communicate with the OpenFlow-enabled switches to control the forwarding behaviors of the switches. The latency between the controllers and the switches becomes an issue to maintain the consistency of the data planes of all the switches in the network. Heller et al. study the problem of controller placement [49]. They find that one controller location is sufficient to support a network with the existing requirements on the network control reaction-time.

2.2.4 Network Virtualization Environments

Network virtualization environments enable the deployment of VNs. PlanetLab, VINI, and GENI are virtualization environments that are widely deployed on the Internet to facilitate experimentation.

PlanetLab [70] is a prominent virtualized infrastructure for deploying multiple VNs that share the substrate network resources distributed across hundreds of physical domains on the Internet. A user requests a slice on PlanetLab and selects a set of PlanetLab nodes to add to the slice. Once the user selects a set of nodes for the experiment, the user can connect these nodes together to form a VN within the slice.

VINI [21] was another environment similar to PlanetLab. It provides realistic and controlled environment for network experiments. It uses XORP for routing and Click modular router for packet forwarding and network address translation. VINI has been decommissioned in 2012.

There are two common methods to create and manage a VN within a slice on PlanetLab. The first implements VNs in user space. This requires the creation of a virtual router in each container using the Click router [2], and connecting the virtual routers together with UDP tunnels [20]. User space implementations increase the latency of forwarding packets due to the time copying packets between the kernel and user space and also the time waiting for Click to run on the CPU. With this method, both control and data planes are implemented on the user space without separation.

Another method to create a VN on PlanetLab is by setting up packet forwarding in the kernel space using the Vsys API [22]. With Vsys, a virtual router in a PlanetLab node can install routes to the forwarding table in the kernel space. When a PlanetLab node receives packets for one of the slices it hosts, it does not copy the packets from kernel space to user space before the redirection. Instead, the packets are forwarded directly according to the forwarding table of the node. Thus, the latency of packet forwarding within a node is considerably lower compare to using the user space routers [56].

Vsys currently is fully functional on the PlanetLab Europe nodes. It allows each slice to have access to certain super user `sudo` privileges, that include creating virtual interfaces and tunnels and forwarding table entries for that slice. Each user is assigned a private IP subnet for the slice. The forwarding table in the kernel is shared among other slices. Isolation is provided through Vsys such that actions performed on a PlanetLab node in a slice do not affect other slices sharing the same PlanetLab node. A user can only modify the forwarding table entries that belong to the assigned IP subnet. With this kernel space forwarding table sharing feature, a user space router can install routes to the forwarding table in kernel space directly.

GENI [42] is a newer environment for implementing and evaluating virtual network migration work. It has a similar slicing concept as Planetlab that each user has a slice which contains both computing and networking resources. It has been deploying the Software-Defined Networking (SDN) infrastructure on different campuses. OpenFlow-enabled switches [65], aggregate managers and FlowVisors [3] are implemented in the GENI infrastructure. The GENI clearing house acts as the main aggregate manager to allocation resources to users.

2.3 TCP and Routing Layer Events

VN migration does affect the performance of the transport layer. For example, TCP [71] adjusts its sending rate based on the measured path characteristics, such as path latency, available bandwidth, round trip time (RTT). When there is any route changes, the performance of TCP is highly affected.

Varadhan et al. have examined how route dynamics affect various versions of TCP [86]. A small topology change can trigger lots of packets to reorder. In [75], Ranadive and Medhi have measured the effects of route oscillations and link failure on TCP. Other than packet reordering, they have also observed false retransmission of the sent packet because of the duplicate acknowledgments (ACKs) from different paths. False retransmission also affects the goodput because after the retransmission, the size of the congestion window is reduced.

Another common phenomenon observed during route oscillation is TCP-asymmetry, which the data packets and the ACKs of a TCP flow traverse through paths with different latencies or throughputs. TCP-asymmetry highly affects TCP performance. Balakrishnan et al. have suggested a few ways to improve the condition including to decrease the rate of ACKs [17].

Other than packet reordering and TCP-asymmetry, packet loss and retransmission timeout occur when a link is unstable. Originally packet loss is a sign to TCP for congestion control. After a packet loss, that loss data segment is not retransmitted again until the retransmission timeout. If the sender has not received an acknowledgment from the receiver within the timeout time, the sender will retransmit the data segment again and restart the retransmission timer with a double of RTO [68]. If a link is interrupted for a few seconds, TCP may differ its back off and recovery time more than it's needed. Sarolahti et al. proposed a TCP algorithm to enhance the recovery time after TCP retransmission timeouts [77]. To further mitigate the effects of performance issues due to characteristics of a link on the path, performance enhancing proxies are proposed [25]. One of the performance enhancing proxy solutions was utilizing split TCP. Split TCP was proposed to mitigate the effects of frequently disrupted links [16].

Other than the proposed TCP algorithms and variants, to enhance the performance of TCP, Ghobadi et al. proposed a centralized controlled OpenTCP framework to allow the network operators tuning the TCP socket options of the machines in a data center network [43].

CHAPTER III

ALGORITHMS FOR SCHEDULING VN MIGRATION

In this chapter, we focus on the specific form of virtualization where virtual routers are instantiated in physical router hardware and where multiple virtual routers belonging to different virtual networks may share the same physical router. The structure of such networks is illustrated in Figure 5, where two virtual networks are sharing a common physical network. On each physical router, the figure indicates which virtual routers are instantiated; virtual links are denoted by dashed lines that may traverse more than one physical link. This type of virtualization allows a substrate network provider to be decoupled from (and offer services to) overlay network providers.[1]

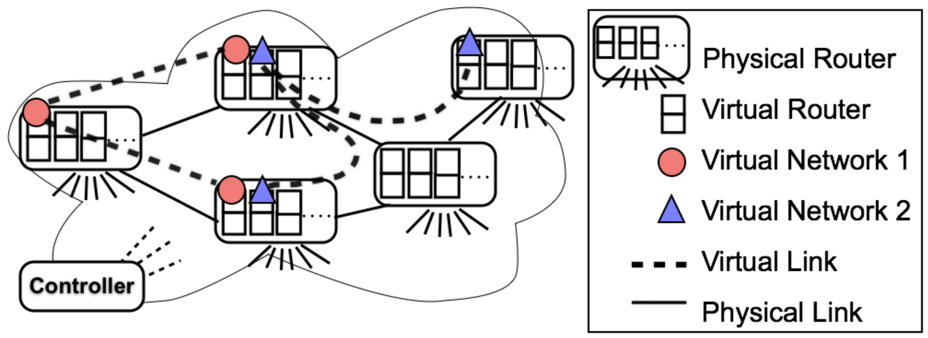


Figure 5: The system architecture for virtual network migration

A virtual network migration mechanism needs to take into account the specifics of the router virtualization technology. It also needs to be guided by desired performance objectives such as: 1) how to manage disruption to existing traffic on the virtual network, 2) what is the additional overhead incurred by the physical substrate while the migration is in progress and 3) how long the migration process lasts.

[1]This form of virtualization is distinguished from peer-to-peer virtual overlay networks, e.g. [30], where end users establish a network by using physical paths between end systems.

Our work starts by leveraging prior work that designs a *single virtual router* live migration mechanism [88]. That work aims to perform virtual router migration in a manner that does not disrupt current data flows using the migrating router. We build on this single virtual router migration mechanism (outlined in more detail in the next section) and use it as a subroutine in the live migration of an *entire* virtual network. Specifically, we are interested in determining, given a new target location within the substrate for a virtual network, the best *schedule* of virtual router migrations that implement the desired network move. We maintain the goal of not disrupting current network traffic during the migration process and consider how to design a schedule that minimizes the overhead (cost) of network migration or the time it takes to complete the migration task. Our work is the first we are aware of that considers the question of scheduling a live network migration.

We first give an overview of a single virtual router migration and virtual network migration process. Then we present our model in Section 3.2 that describes the cost and duration of a network migration sequence. In Section 3.3, we develop three algorithms for determining an optimal single-node-at-a-time sequence of moves to minimize migration overhead (cost) and schedules that allow for multiple node moves in parallel to minimize migration cost or time. We evaluate these algorithms in Section 3.4 with simulations of virtual networks on a physical network. We conclude our work in Section 3.5 with future work.

3.1 Background

In this section, we first review the process of migrating a single virtual router. Then we describe the process of migrating a virtual network and the system architecture that supports this functionality.

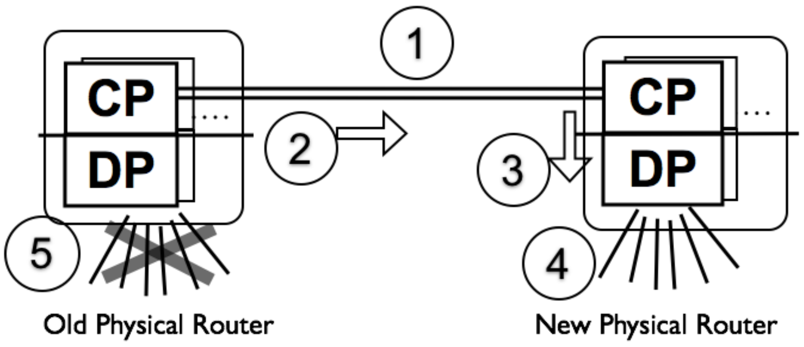


Figure 6: Virtual router migration steps 1-5. The old and new physical routers contain multiple virtual routers. (CP=Control Plane, DP=Data Plane)

3.1.1 Live Migration of a Single Virtual Router

Our work leverages the research in [88], which develops a mechanism for the live migration of a single router without disrupting current traffic traversing the virtual network. That work assumes a router architecture with multiple virtual routers instantiated on a physical router. Three features of virtual router architecture make it "migratable": independence of the virtual router through OS virtualization support, separation of data and control planes, and the ability to dynamically bind a router's data plane to physical substrate interfaces. With this router architecture, the work in [88] describes the following sequence of actions that are required to move a virtual router. The numbers in Figure 6 correspond to each step:

1. *Setup a tunnel* between the original and the final physical locations of the virtual router.

2. *Migrate control plane and copy memory*: Create an image of the virtual router's control plane and transfer it through the tunnel to the new physical router. At the same time, collect the routing update messages for this virtual router and forward them to the virtual router after the image transfer is completed such that they can be processed in the new location.

3. *Clone the data plane* based on the existing control plane on the new physical node and repopulate the FIB. After this is completed, both old and new data planes are running at the same time.

4. *Duplicate forwarding links between the virtual router and its neighbors*: Set up the outgoing virtual links from the new physical router to its neighbors. Once these are ready, set up original incoming virtual links to redirect traffic from the neighbors of the original virtual router to the virtual router in the new physical machine. The new traffic flows are assigned to the new forwarding links.

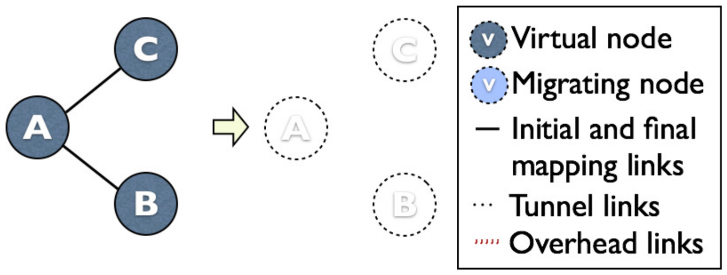
5. *Remove old forwarding links*: Once old traffic flows at the old forwarding links are complete, remove the data plane instance in the original physical location.

We use this single virtual router migration process as a subroutine to migrate each virtual router on a virtual network, as described next. In the rest of the paper, we use the term "virtual node" and "virtual router" interchangeably.

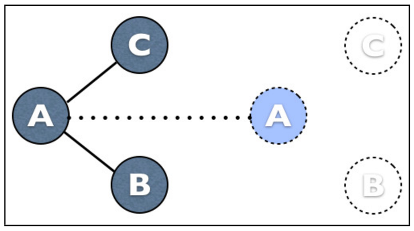
3.1.2 Network Migration Process

Figure 7 demonstrates a virtual network migration. The example network has three virtual nodes, A, B and C, which are moved one at a time, in three stages, across the substrate. Each stage has two parts. During the *preparation part*, the control plane moves to its new physical node and the data plane is cloned (Step 1-3 in Section 3.1.1). After the preparation part, during the *migration part*, the duplicate forwarding virtual links are created and eventually the old forwarding links are removed (Steps 4-5 in Section 3.1.1).

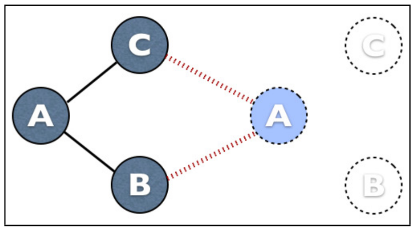
The migration sequence in the figure is (A, C, B). After A's control plane is migrated in the preparation part of Stage 0 in Figure 7(b), virtual links between the physical nodes of A and its neighbors, B and C are set up and are working in parallel with the original virtual links in the migration part in Figure 7(c). After the



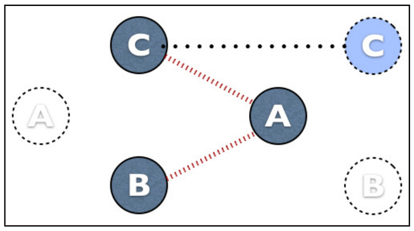
(a) A virtual network with node A, B and C on their initial mappings



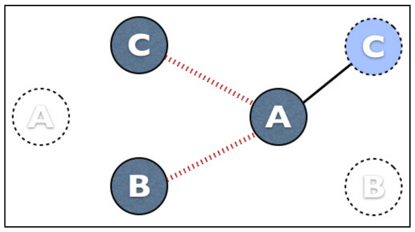
(b) Stage 0: Preparation part: copying the control plane of A



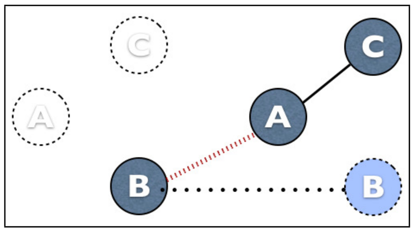
(c) Stage 0: Migration part: duplicating forwarding links between A and its neighbors



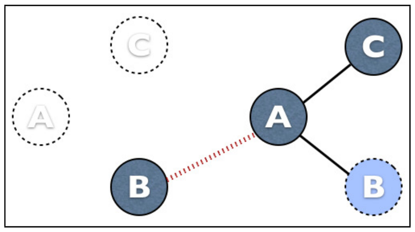
(d) Stage 1: Preparation part: copying the control plane of C



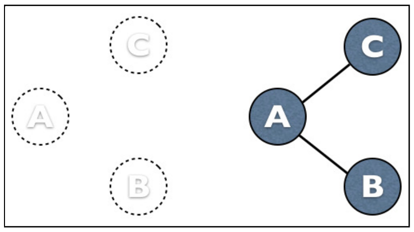
(e) Stage 1: Migration part: duplicating forwarding links between C and its neighbor



(f) Stage 2: Preparation part: copying the control plane of B



(g) Stage 2: Migration part: duplicating forwarding links between B and its neighbor



(h) Stage 2: Finished migration

Figure 7: Migrating a virtual network with virtual node A, B and C. (Physical network is not shown.) The migration sequence is A, C and then B.

migration of node A, the virtual links between the migrated node A and its neighbors are not finalized until all the neighbors have been migrated. We will return to this issue when we analyze the cost of a move, but for now we simply note that there are migration overhead links that must be maintained in support of a migration in progress. The length (physical network hop count) and the duration (time maintained) of any overhead link is influenced by the migration sequence. Our migration scheduling algorithms aim to reduce length and duration of overhead links.

3.1.3 System Architecture for Virtual Network Migration

To enable the network to migrate and determine the migration sequence, we adopt a simple system architecture. Figure 5 shows our system architecture which is similar to some existing overlay network architectures such as GENI [42]. In our architecture, the physical network has a controller to initiate and synchronize the virtual network migrations. The controller connects to all of the physical routers through in-band or out-of-band tunnels. When a request for a virtual network migration is initiated based on reconfiguration goals, the controller is given initial and final virtual network mappings. The controller decides the node migration sequence using algorithms running locally or remotely, and that are discussed in Section 3.3. The controller initiates the steps for each single node migration to accomplish the network migration.

3.2 Problem Statement

We now turn to a formal statement of the migration problem and a model for the cost and time associated with a given virtual node move sequence.

3.2.1 The Migration Problem

Let $P = (R, E)$ be the physical network where R is a set of physical routers and E is a set of physical links. Let $G = (V, L)$ be a virtual network where V is a set of virtual nodes and L is a set of virtual links.

Let $m : V \rightarrow R$ be a function that maps the set of the virtual nodes in G to a set of physical routers in P, i.e. $m(v) = r$ means a virtual node $v \in V$ is mapped to a physical router $r \in R$. A virtual link is mapped to a physical path (an ordered list of connected physical routers) where the endpoints of the virtual link are mapped to the first and last routers on the path and each pair of physical routers in the list are joined by a physical link.

We are given an initial mapping $m_0 : V \rightarrow R$ and a final target mapping $m_f : V \rightarrow R$. The goal of migration is to determine the most efficient schedule to move virtual nodes so as to accomplish the virtual network move from m_0 to m_f, where efficiency may be measured by overhead cost of migration, total migration time, or a combination of the two.

As explained earlier, network migration consists of a sequence of stages, where each stage migrates a subset of the virtual nodes. Let $S = (S_0, S_1, S_2, \ldots)$ be a sequence of node migrations that move the virtual network from m_0 to m_f. At stage i, the set of virtual nodes moved is $S_i \subseteq V$ such that $\forall i \neq j \; S_i \cap S_j = \emptyset$ and $\forall i \; S_i \neq \emptyset$ and $\bigcup_i S_i = V$.

There is an additional constraint on the sets S_i, namely that a given stage, we cannot move two nodes that are neighbors in the virtual network. This constraint arises because of the way that links are maintained in the single node move procedure. If two neighbors are moved at the same time, the virtual network will enter an inconsistent state. When we devise algorithms to move multiple nodes at a time, we are careful to satisfy the no-neighbors constraint.

We are interested in two important performance metrics: 1) the completion time of a virtual network migration, and 2) the cost of a network migration representing the bandwidth impact on the physical network of the migration process. We describe each of these formally in the next two subsections.

3.2.2 Network Migration Time

Recall that each stage consists of a preparation part and a migration part, as illustrated in Figure 8. The preparation part consists of setting up the tunnels, migrating the control planes and cloning the data planes by repopulating the FIB. The migration part consists of duplicating forwarding links, waiting for all old data flows to complete, and finally removing old forwarding links. After stage i, the network has fully completed all moves in sequence S up to and including S_i.

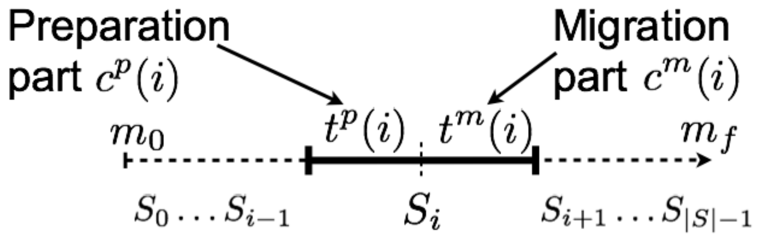


Figure 8: Illustration of the time of a virtual network migration

We denote the time taken in the *preparation part* and the *migration part* at stage i by $t^p(i)$ and $t^m(i)$, respectively. The total time for a particular migration sequence S is then

$$\sum_{i=0}^{|S|-1} (t^p(i) + t^m(i)). \tag{1}$$

The time spent in the preparation part $t^p(i)$ depends on the following characteristics of the migrating nodes in S_i: the available bandwidth of the tunnels, the size of the control plane images, and the time to repopulate each FIB. For simplicity of exposition, we assume for a given virtual network that the control plane image sizes are the same for all the nodes. When the sizes of the control planes are the same and the data plane technology of the physical routers are the same, the FIB repopulation times for all the nodes are also the same. An interesting effect takes place, however, when nodes migrate together at the same stage. As more nodes move together, there is a higher probability that the tunnels used to move the control planes may share a bottleneck physical link. Thus while it is generally advantageous to move multiple nodes at a time, too much interference in the control plane tunnels can reduce or eliminate the benefit.

We model the time in the preparation part $t^p(i)$ as a function of the nodes in the set S_i and the time to migrate a control plane and repopulate a FIB, $f(S_i, \alpha)$, where α is the time for migrating the control plane and populating the FIB of a single node. We explore different functions $f(S_i, \alpha)$ in Section 3.4.

The time in the migration part $t^m(i)$ mainly depends on the current traffic flows on the migrating nodes' old forwarding links. Once these flows end, the migrating nodes can completely switch to the new forwarding links. We assume that the time for these flows to end is constant over all stages and independent of the number of nodes moved at a stage.

Given the time computation, it should be clear that the primary method for reducing the migration time is to move as many nodes as possible together, thus reducing the number of stages, while taking care not to overload physical links that are needed to move the control planes.

3.2.3 Network Migration Cost

We model the impact of the virtual network migration on the physical network as a cost function that is divided into the cost during the preparation part ($c^p(i)$) and the cost during the migration part ($c^m(i)$) of each migration stage i. Our measure of cost is the number of physical links that carry migration-related traffic multiplied by the duration these links are used (stage).

In the preparation part, there is a bandwidth impact on the physical network resulting from the transfer of the control plane across the tunnel established from the original location to the final location. Regardless of the migration sequence, each control plane must be transferred, and the cost is simply the sum of all control plane sizes. Hence this cost component is a constant. The bandwidth impact can be different under different migration sequences when tunnels possibly share some physical links with different sets of overhead virtual links. We will address this issue in two of our proposed algorithms.

We are interested in the cost implications that derive from overhead virtual links. In both the preparation and the migration part, a set of virtual links are present only because of the migration process. These links are dependent on the migration sequence. To understand the impact of these overhead virtual links, we divide the nodes into three sets at stage i:

1. Nodes that will move in this stage (i.e., $S_i \in V$),
2. Nodes that have already moved in prior stages, denoted $A_i \in V$, and
3. Nodes that have not yet moved, denoted $Y_i \in V$.

where $S_i \cup A_i \cup Y_i = V$ and $S_i \cap A_i \cap Y_i = \emptyset$.

We classify overhead virtual links at stage i into three sets:

1. $L_1(i)$: Virtual links between the original mappings of the moving nodes (S_i) and final mappings of their already moved neighbor nodes (A_i), i.e. $l = (v, w)$ which the physical path is $(m_0(v), m_f(w))$ where $v \in S_i, w \in A_i$,
2. $L_2(i)$: Virtual links between the final mappings of the moving nodes (S_i) and the original mappings of their not moved neighbor nodes (Y_i), i.e. $l = (v, x)$ which the physical path is $(m_f(v), m_0(x))$ where $v \in S_i, x \in Y_i$, and
3. $L_3(i)$: Virtual links between the final mappings of the already moved nodes (A_i) and the original mappings of their not moved neighbor nodes (Y_i), i.e. $l = (w, x)$ which the physical path is $(m_f(w), m_0(x))$ where $w \in A_i, x \in Y_i$.

Figure 9 illustrates a simple three-node virtual network with nodes x, v, and w. In the stage illustrated, node v is moving, node w has already moved, and node

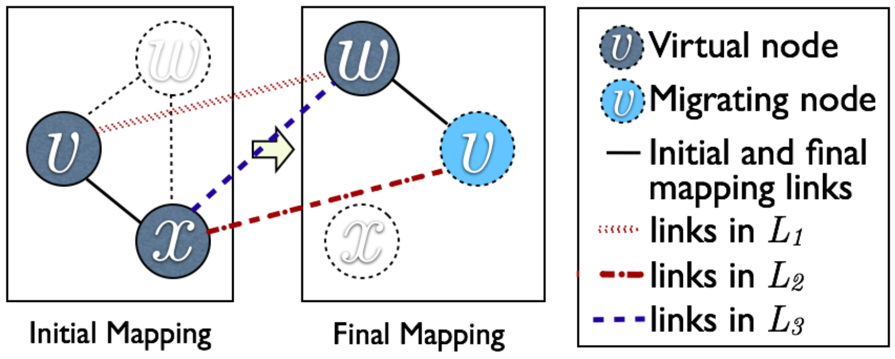


Figure 9: Illustration of types of links at stage i when v is migrating after w has moved at stage $i-1$.

x has yet to move. The virtual link (v, w) which is mapped to the physical path $(m_0(v), m_f(w))$ is of type L_1, the virtual link (x, v) which is mapped to the physical path $(m_f(v), m_0(x))$ is of type L_2, and the virtual link (w, x) which is mapped to the physical path $(m_f(w), m_0(x))$ is of type L_3.

The links in $L_1(i)$ and $L_3(i)$ are present throughout both the preparation and migration parts of stage i. The links in $L_2(i)$, on the other hand, are present only during the migration part of stage i. Indeed they are created to maintain connectivity when the nodes in S_i are moving. We use $p(l)$ to denote the bandwidth impact of virtual link l. Later we simplify by assuming $p(l)$ is the number of physical links in path l.

The cost of the virtual network in the preparation part of stage i is the multiplication of the time taken in the preparation part and the sum of physical links in path l that are of type L_1 and L_3 as expressed by

$$c^p(i) = t^p(i) \sum_{l \in (L_1(i) \cup L_3(i))} p(l). \tag{2}$$

The cost of the virtual network in the migration part of stage i is the multiplication of the time taken in the migration part and the sum of physical links in path l that are of type L_1, L_2, and L_3 as expressed by

$$c^m(i) = t^m(i) \sum_{l \in (L_1(i) \cup L_2(i) \cup L_3(i))} p(l). \tag{3}$$

The total cost for a particular migration sequence S is the sum of the cost in all preparation and migration stages:

$$\sum_{i=0}^{|S|-1} c^p(i) + c^m(i). \tag{4}$$

3.3 Algorithms for Virtual Network Migration

We present three algorithms for migrating virtual networks. These algorithms can be divided into two types: moving one node at a time and moving multiple nodes at a time. We design the algorithms with two goals. One of the goals is to minimize the total migration time of the migration, which is described in Equation 1. The other goal is to minimize the total cost, which is described in Equation 4.

In the move-one-node-at-a-time scheme, a virtual node can start moving to its final placement only when the previous virtual node has finished its move. We find a move sequence $S^* = (S_0, S_1, \ldots)$ where $\forall i\ |S_i| = 1$.

In the move-multiple-nodes-at-a-time scheme, multiple nodes can be migrated at the same stage. However, we are constrained not to move nodes that are neighbors in the virtual network at the same stage.

In this section, we first introduce the algorithm Local Minimum Cost First (LMCF) which produces a move-one-node-at-a-time schedule to minimize the migration cost. Then we present two move-multiple-nodes-at-a-time algorithms: Maximal Independent Set-Size Sequence (MIS-SS) and Maximal Independent Set-Local Minimum Cost First (MIS-LMCF) which minimize the time and the cost of the migration. Note that we do not have an algorithm in the category of move-one-node-at-a-time with the goal of minimizing the time of migration because the number of stages required for any one-node-at-a-time scheme is the same.

3.3.1 Local Minimum Cost First (LMCF)

The goal of Local Minimum Cost First (LMCF) algorithm is to minimize the migration cost under the move-one-node-at-a-time scheme. LMCF is described in Algorithm 1. First we compute the initial migration cost of the migration part $c^m(0)$ of stage 0 for each virtual node and choose the cheapest one to migrate first. Then from the rest of the unmoved virtual nodes, we recompute the migration cost c^m for each of them based on the current link mapping at that stage. Again, we choose the one that gives the cheapest cost and migrate it. We repeat until the network is completely migrated. If there are nodes that give the same cost, we pick the node with the lowest cost of the preparation part c^p. If both of their migration and preparation parts' costs are the same, we break the tie by randomly picking one of the cheapest nodes to be moved next.

Algorithm 1 Local Minimum Cost First

```
1: function LMCF($G = (V, P)$, $P = (R, E)$, $m_0$, $m_f$)
2:     set sequence $S = [\ ]$                          ▷ Result sequence
3:     for $i = 0 \to |V| - 1$ do
4:         $j = 0$
5:         for $v \in V - (S_0 \ldots S_{i-1})$ do
6:             $S_i = v$
7:             $cost[v] = c^m(i)$             ▷ Find the cost for $v$ to be migrated at stage $i$
8:             if $j = 0$ then
9:                 $minv = v$
10:                $mincost = cost[v]$
11:            else if $cost[v] < mincost$ then
12:                $minv = v$
13:                $mincost = cost[v]$
14:            end if
15:            $j = j + 1$
16:        end for
17:        $S_i = minv$                 ▷ Put the minimum cost node to move at stage $i$.
18:    end for
19:    return $S$
20: end function
```

3.3.2 Maximal Independent Set-Size Sequence (MIS-SS)

The goal of MIS-SS is to minimize the migration time. In MIS-SS, we simplify the time as the number of stages involved in a virtual network migration. We consider minimizing the number of migration stages by migrating multiple nodes at a time. We first determine the nodes which can be migrated at the same stage. We call the set of nodes to be migrated at a stage as *migration set*. Because of the no-neighbor constraint, the nodes to be migrated at the same stage cannot be neighbor among each other. The set of these non-neighbor nodes is called *independent set*. We find the maximum sets among these independent sets such that each set is not a subset of another set. These sets are called *maximal independent sets* (MISs). We use an algorithm from [4] to generate MISs. This MIS algorithm generates a maximal independent set with inputs of a graph and a node in the graph. We use this algorithm and input each node to the algorithm to generate the maximal independent sets of the graph.

With the migration sets produced by the MIS algorithm, we design an algorithm to determine the migration sequence of these sets to minimize the time. MISs are not *maximum* independent sets. MISs can have intersections. Because of their intersections, when we migrate a MIS, some of the nodes in other MISs may be migrated as well. Thus, the remaining sets shrink. The number of stages involved in a virtual network migration can be highly reduced from LMCF, to the number of MISs, then to the number of migration sets in a set sequence which eliminates more nodes at the earlier stages.

In MIS-SS (Algorithm 2), first we use the MIS algorithm (FindMIS(G=(V,P))) to determine migration sets of a virtual network. Then, MIS-SS determines the migration sequence by picking the largest MIS to be migrated first. At each stage, MIS-SS picks the largest remaining set to be migrated. After each stage, MIS-SS removes the nodes which have been migrated from the remaining sets. The remaining sets shrink when more nodes are migrated. If the sets are highly intersected, the time for migration can be reduced significantly.

Algorithm 2 MIS-Size Sequence (MIS-SS)

```
 1: function MIS-SS($G = (V, P)$, $P = (R, E)$, $m_0$, $m_f$)
 2:     $MSets[\,] = FindMIS(G = (V, P))$    ▷ Call the MIS algorithm to generate all maximal
    independent sets (MISs)
 3:     $S_0 = MaxSize(MSets[\,])$    ▷ First migrate the largest MIS
 4:     for $i = 1 \rightarrow |MSets[\,]| - 1$ do
 5:         Remove $\forall v \in S_0 \ldots S_{i-1}$ from all $MSets[\,]$    ▷ Remove the migrated nodes from $MSets[]$
 6:         if $MaxSize(MSets[\,]) = \emptyset$ then
 7:             break
 8:         end if
 9:         $S_i = MaxSize(MSets[])$    ▷ Migrate the set with the most number of nodes left. If there
    are equal number of nodes in more than one sets, pick one set randomly.
10:     end for
11:     return $S$
12: end function
```

3.3.3 Maximal Independent Set-Local Minimum Cost First (MIS-LMCF)

In MIS-LMCF, we attempt to minimize a combination of cost and time. When we shorten the time of the migration, we can minimize the time of the overhead links staying at the migrating state and further minimize the cost of the migration. In Algorithm 3, we move multiple nodes at a time to minimize the migration time. We also consider the migration cost of each migration set to schedule the migration sequence.

We start with the sets of nodes $MSet[\,]$ from the MIS algorithm. We use a modified version of LMCF to determine the set migration sequence. We migrate the set with the lowest cost first. After each stage, we remove the migrated nodes from the remaining sets and recompute the migration cost for each remaining set. Again, we choose the set with the lowest cost to be moved next.

3.4 Evaluation

We evaluate the migration algorithms with simulations. We are interested in a set of basic questions about algorithm performance relative to optimal, to worst case, and to one another. Specifically, (Q1) how important is it to have an intelligent migration algorithm? (Q2) how close do the proposed algorithms come to optimal? (Q3) how

Algorithm 3 MIS-Local Minimum Cost First (MIS-LMCF)

```
1: function MIS-LMCF($G = (V, P)$, $P = (R, E)$, $m_0$, $m_f$)
2:     $MSets[] = FindMIS(G = (V, P))$    ▷ Call the MIS algorithm to generate all MISs
3:     for $i = 0 \to |MSets[\,]| - 1$ do
4:         $j = 0$
5:         Remove $\forall v \in S_0 \ldots S_{i-1}$ from all $MSets[\,]$  ▷ Remove the migrated nodes from $MSets[]$
6:         if $MaxSize(MSets[\,]) = \emptyset$ then
7:             break
8:         end if
9:         for $set \in MSets[\,] - \{S_0, \ldots S_{i-1}\}$ do
10:            $S_i = set$
11:            $cost[set] = c^m(i)$ when $S_i = set$  ▷ Find the cost for the set of nodes to be migrated at stage $i$
12:            if $j = 0$ then
13:                $minset = set$
14:                $mincost = cost[set]$
15:            else if $cost[set] < mincost$ then
16:                $minset = set$
17:                $mincost = cost[set]$
18:            end if
19:        end for
20:        $S_i = minset$    ▷ Return the set of nodes with minimum cost to move at stage $i$
21:    end for
22:    return $S$
23: end function
```

do the algorithms compare to one another? We next turn to two characteristics of the virtual and physical networks that play a role in algorithm performance. (Q4) What is the influence on cost of migration distance, i.e., how far apart in the physical network are the original and final virtual network (VN) locations? (Q5) What is the influence of VN node degree on the time and cost of migration? Based on the results, we make some preliminary recommendations about the operation of networks that support migration.

3.4.1 Simulation Setup

We evaluate the algorithms with 50 different 10-node VNs and a 100-node physical network. The 10-node VNs are randomly generated by the Python NetworkX package [5] with average node degree distributions from 2.4-5.4 and diameter from 2-7. The 100-node physical network is generated by GT-ITM [45] with the transit-stub model with 3.7 average node degree and diameter of 11.

In our simulations, each VN is mapped to different initial and final mappings on the physical network. These mappings are randomly chosen with the constraint that there are no physical nodes in common in the initial and final mappings. We simulate the migration sequences produced by the algorithms. We also exhaustively try every one-node-at-a-time move sequence. We calculate the cost and time for each migration sequence generated by the exhaustive search and the three algorithms. We use the control plane transfer time from [88] for the preparation part ($t^p = f(S_i, \alpha)$). We assume that each of the virtual routers on the 10-node VNs has less than 1k routes. The memory copy time is around 1 second with 10k routes. The FIB repopulating time of the software data plane is 2.1 seconds for 1k routes. Thus, $\alpha = 1 + 2.1 = 3.1$ *seconds*. We experiment with two choices for the function $f(S_i, \alpha)$. In the first case, we assume we will migrate one control plane at a time, hence $f_1(S_i, \alpha) = \alpha|S_i|$. In the second, we assume that we can migrate all control planes in parallel with no bandwidth interference, hence $f_2(S_i, \alpha) = \alpha$.

For the time in the migration part, we model the longest finishing time of any current flow traversing the migrating router(s). This is not easy to estimate, since in general it depends on the uses of the VN, the capacities of the physical and virtual links, and the degree of sharing among flows and VNs. Further, long running flows might lead to policy decisions to avoid migrating them altogether or to disrupt these flows so that migration can be accomplished more quickly. In the absence of specific information, and consistent with our desire to use values derived from [88], we use the longest control plane transfer time as our longest flow finishing time. Figure 8 in [88] shows a time of about 3 seconds to transfer a 124 MB control plane. We choose to use $t^m = 3.5$ *seconds*, providing a bit of cushion above the 3 second measurement. In future work, we will examine models for flow finishing time and adaptations to the migration process with long flows.

3.4.2 Baseline Results

We first consider basic algorithm performance (Q1). Table 1 shows the optimal and worst case costs of all possible one-node-at-a-time migration sequences for VN 1-5, as well as the ratio of worst cost to optimal cost. It is clear that the ratio of worst cost to optimal cost can be substantial, exceeding 2.5 for all five networks and as high as 3.37 for VN 1, which has lowest average node degree and highest diameter. We conclude from this experiment that there is value to investigating migration algorithms that can achieve cost that drives towards optimal.

Table 1: The costs of the best cases, the worst cases and LMCF of VN 1, 2, 3, 4 and 5 (move one-node-at-a-time sequences) with $t^p = f_1(S_i, \alpha = 3.1)$ sec and $t^m(i) = 3.5$ sec

VN	Optimal S^*	Worst	Ratio	LMCF
1	890.42	3000.64	3.37	890.42
2	1116.06	3263.55	2.92	1179.68
3	1413.76	3585.74	2.53	1635.49
4	1739.87	4864.23	2.80	1897.57
5	2751.34	6984.34	2.54	2985.42

We compare the results from our greedy LMCF algorithm with the best cost. The rightmost column of Table 1 shows that LMCF finds migration sequences with cost close to the optimal solution on all five VNs. Without an exhaustive search on all $10! \approx 3.6$M combinations of migration sequences, LMCF can give a migration sequence with cost between 1 and 1.16 times the optimal migration sequence.

We next turn to the performance of the two algorithms that can move multiple nodes at a time. Recall that MIS-SS attempts to minimize time by moving as many nodes at a time in each stage and hence reduce the total number of stages. MIS-LMCF attempts to minimize a combination of cost and time by moving large sets but choosing which set to move based on cost considerations. Both algorithms rely on the MIS algorithm to generate candidate migration sets. It is computationally prohibitive to exhaustively try all possible multiple-node-at-a-time sequences, hence

Table 2: Results of MIS-SS, MIS-LMCF and LMCF with five of the 10-node VNs with $t^p = f_1(S_i, \alpha = 3.1) = |S_i|\alpha$ (the best costs are underlined)

	MIS-SS		MIS-LMCF		LMCF	
VN	Cost	Time	Cost	Time	Cost	Time
1	1087.42	44.88	1093.18	44.88	<u>890.42</u>	65.88
2	1415.38	41.38	<u>1162.36</u>	48.38	1179.68	65.88
3	1714.10	44.88	<u>1402.41</u>	51.88	1635.49	65.88
4	1603.54	44.88	<u>1482.29</u>	44.88	1897.57	65.88
5	<u>1710.82</u>	44.88	3335.73	44.88	2985.42	65.88

Table 3: Results of MIS-SS, MIS-LMCF and LMCF with five of the 10-node VNs with $t^p = f_2(S_i, \alpha = 3.1) = \alpha$ (the best costs are underlined)

VN	MIS-SS Cost	Time	MIS-LMCF Cost	Time
1	797.15	29.44	<u>657.78</u>	29.44
2	881.15	25.94	<u>872.09</u>	32.94
3	<u>1102.67</u>	32.53	1105.96	39.53
4	1090.94	32.53	<u>1056.14</u>	32.53
5	1710.82	26.35	<u>1501.46</u>	26.35

we focus our attention on the comparison of these algorithms to one another and then to the LMCF algorithm. Table 2 shows the results assuming one control plane is migrated at a time, while Table 3 shows the results assuming all control planes in a stage can migrate together without interference.

The goal of the MIS-SS algorithm is to minimize the migration time. It successfully achieves the lowest migration time of any of the three algorithms on each VN, and it improves on the LMCF algorithm by about 1.5 times when the control planes move one at a time and by more than 2 times when the control planes move in parallel. If migration time is the key consideration, the MIS-SS algorithm provides a good solution.

The goal for MIS-LMCF is also to reduce the number of stages but to do so taking cost into consideration. The MIS-LMCF algorithm successfully achieves relatively low migration time as compared to the LMCF algorithm, and frequently matches the time performance of MIS-SS. Turning to cost, the MIS-LMCF algorithm achieves the lowest cost more often than MIS-SS or LMCF, but it does not always have lowest cost. For example, in Table 2, for VN 1, the lowest cost is achieved by the LMCF algorithm rather than either of the MIS algorithms, while for VN 5 the lowest cost is achieved by MIS-SS. If migration cost is the key consideration, it is important to understand whether control planes can move in parallel with limited interference. If they can, one of the MIS-based algorithms will improve cost. If they cannot, the MIS algorithms generally offer only modest improvement over LMCF, though there is a notable exception for VN 5 where LMCF cost is twice the cost of the MIS-based algorithms.

3.4.3 Effect of Migration Distance

We now dig deeper into the effect of virtual and physical network characteristics on the performance of the algorithms. In particular, we examine the effect of migration distance and then the effect of VN node degree (Q4). Migration distance is determined by the initial and final mappings of a VN to the physical network. We use VN 2 and simulate 90 migrations with 10 different mappings on the 100-node network. For each pair of mappings m_0 and m_f, we calculate the average migration distance by averaging the physical path lengths of the virtual node locations between $m_0(v)$ and $m_f(v)$. For our VN locations and physical network characteristics, we see average migration distances in the range of roughly 3.5 to 6.5 physical node hops. Recall that the physical network has diameter 11, so these distances are consistent with that constraint on the maximum distance a node can move.

Our model for time is based on the time to move one control plane, the interference between control plane moves, and the time to complete the longest outstanding flow. None of these depend on migration distance in our current model, though a refinement of the model might take into consideration that longer distance moves may take more time and that distance might influence how much interference exists between control plane moves.

We expect cost to be influenced by migration distance, since overhead links must traverse the physical network between original node locations to final node locations. In Figure 10, we show the costs of the migration sequences from LMCF, MIS-SS and MIS-LMCF as a function of the average migration distance. The LMCF plot contains 90 points, one for each initial-final pair, while the MIS-SS and MIS-LMCF plots contain two sets of 90 points, one for each model of control plane interference ($f_1(S_i, \alpha) = \alpha|S_i|$ and $f_2(S_i, \alpha) = \alpha$). In all three cases, the cost increases with average migration distance, as expected. The MIS-SS and MIS-LMCF algorithms tend to achieve comparable cost that improves noticeably over LMCF when the control planes can move in parallel. The impact of higher average migration distance is somewhat modest—the highest migration distance settings have cost approximately 1.4x the lowest migration distances. The cost advantage of MIS-LMCF is more pronounced when control plane moves can interfere than when they cannot, and this persists across migration distances.

3.4.4 Effect of Node Degree

Finally we turn to Q5, the question of the impact of VN node degree on the time and cost performance of migration. In our initial experiments, we observed variability in cost across the five VNs, with cost growing as VN node degree increased. Higher node degree means more neighbors, with two cost effects. First, more dense networks have more smaller size MISs, hence more stages and more time is required. Second, more dense networks require more overhead links to connect nodes that have moved to neighbors that have not. Both effects will increase cost. In time, we saw very little variability as average node degree increased.

To explore more deeply, we use all 50 different 10-node VNs. We migrate each from and to the same initial and final mappings so as to eliminate the effect of migration distance. We compare the time for the two MIS-based algorithms and the cost for all three algorithms. Figure 11 shows the migration time of MIS-SS and MIS-LMCF. With increasing average node degree, fewer nodes can be grouped into the same migration set. Thus, the time for migration increases with the average node degree, albeit slowly. Overall, MIS-SS achieves its goal to minimize the migration time for networks with different average node degrees. Because the time is dominated by the migration sets, both algorithms give similar migration times, but the time grows more slowly when the control planes interfere than when they do not.

Figure 12 shows the costs of sequences from LMCF, MIS-SS, and MIS-LMCF. As expected, an increase in average node degree has a substantial effect on the cost. The LMCF algorithm is most significantly affected by the increase in average node degree, since there are more overhead links and they must remain in place until the affected nodes have been moved. The MIS-based algorithms also see increases in cost with average node degree, with slightly slower cost growth when control planes do not interfere. In the limit, when the VN is a full mesh, the MIS-based algorithms must move single nodes at a time and hence cannot do any better than LMCF.

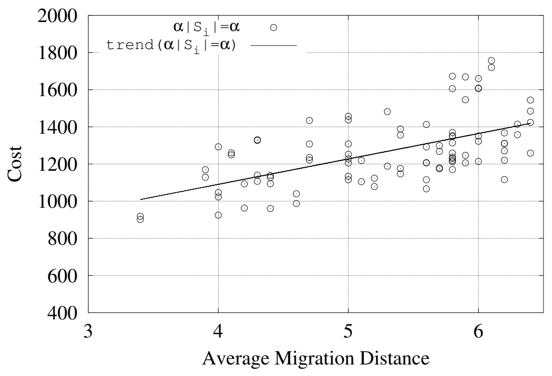
3.5 Summary

From our simulations, all algorithms perform reasonably well. They all give migration sequences with reasonable costs and time. We also observe that no one algorithm consistently exhibits the lowest cost across all the average distance and node degree cases considered in our experiments.

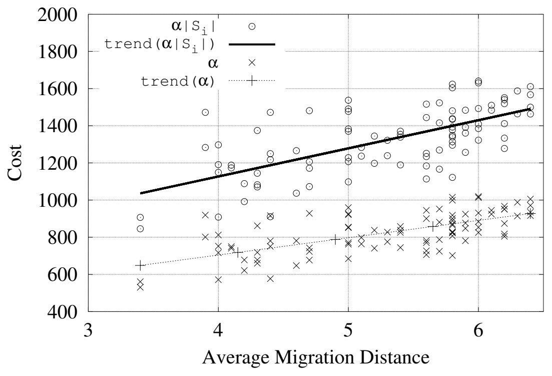
We believe our work makes the application of re-mapping policies in virtual networks possible, especially in architectures where virtual networks require reconfiguration. Our work considers the problem of the mechanism to migrate an entire virtual network from its initial mapping to its final mapping. We design scheduling algorithms for virtual network migrations that minimize the disruption to the current data traffic and the overhead traffic in the migration process. We propose three algorithms, LMCF, MIS-SS, and MIS-LMCF to discover the virtual router migration sequences for migrating an entire virtual network with lower overhead (cost) and shorter time. The LMCF algorithm produces a move-one-node-at-a-time schedule that exhibits cost close to the optimal solution. We then consider move-multiple-nodes-at-a-time schedules to minimize the time and the cost of migration. Our simulations of the MIS algorithms show that move-multiple-nodes-at-a-time schedules effectively minimize the time and cost.

The interactions among multiple virtual network migrations have not been studied in this paper. In future work, the scheduling problem of migrating multiple virtual networks that share the same substrate becomes important. The scheduling problem of multiple virtual network migrations involves the dynamic of migrating multiple virtual routers belonging to different virtual networks. When multiple virtual networks are migrating at the same time, some of the physical links and physical routers are fully occupied and may exceed the capacities of the links and physical routers. Thus, scheduling multiple virtual network migrations is more complicated than a single virtual network migration.

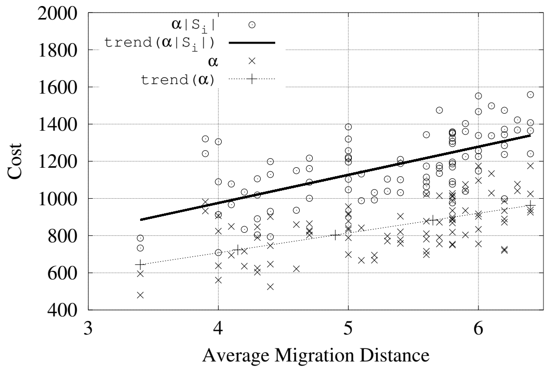
Our work further raises the questions of how we design the applications and system architectures for virtual networks. A possible application of virtual network migration is defending against attack traffic on virtual networks. Future work on extending our virtual network migration scheduling to other technologies such as architecture with FlowVisor [3] and OpenFlow enabled switch networks [65] will be very useful as well.



(a) LMCF

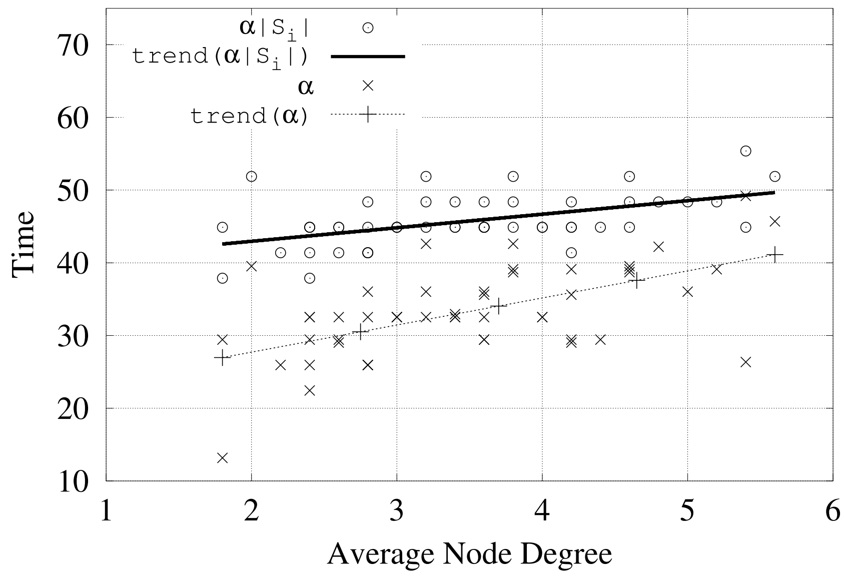


(b) MIS-SS

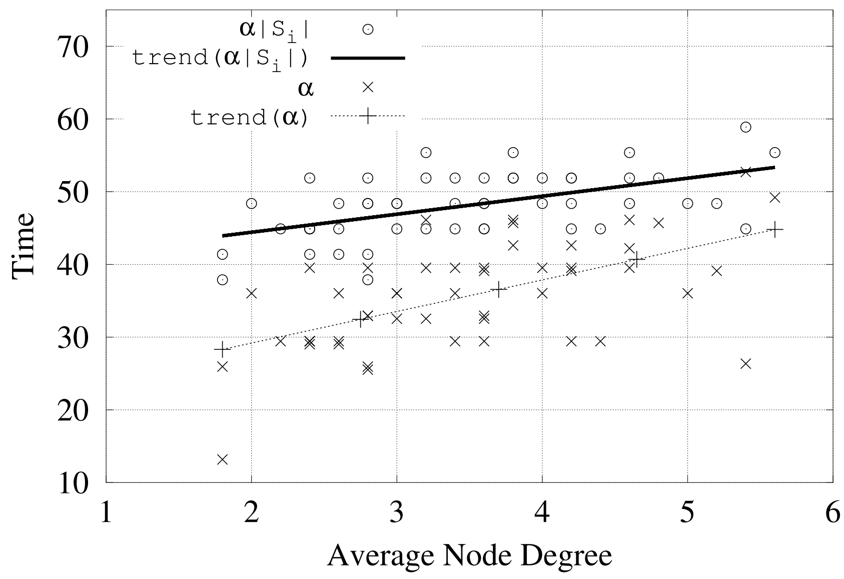


(c) MIS-LMCF

Figure 10: Cost (in the multiplcation of time and the physical path lengths of the links) for migrating VN 2 among 10 different initial and final mappings (90 migrations) versus migration distance (in average physical path length between the initial and final mappings of the VN) with algorithm LMCF, MIS-SS and MIS-LMCF

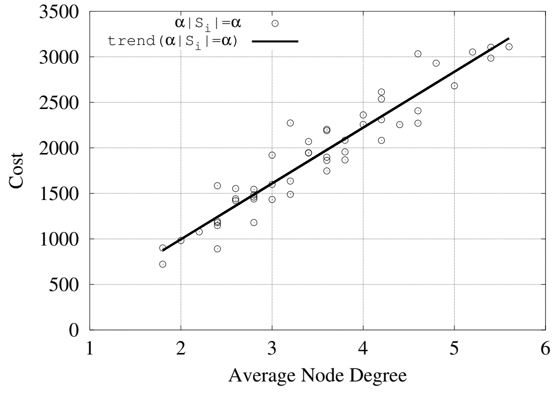


(a) MIS-SS

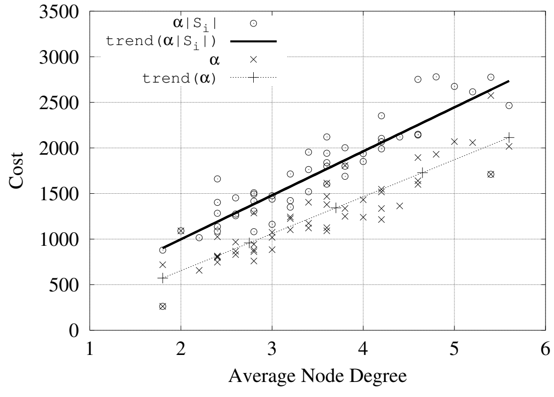


(b) MIS-LMCF

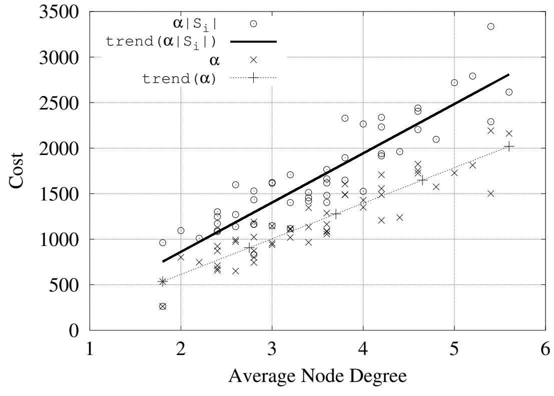
Figure 11: Time of migrating 50 different 10-node VNs on the same initial and final mappings with algorithm MIS-SS and MIS-LMCF



(a) LMCF



(b) MIS-SS



(c) MIS-LMCF

Figure 12: Cost of migrating 50 different 10-node VNs on the same initial and final mappings with algorithm LMCF, MIS-SS and MIS-LMCF

CHAPTER IV

A CONTROLLER-BASED ARCHITECTURE FOR VIRTUAL NETWORK MIGRATION ON PLANETLAB

PlanetLab is a prominent virtualized infrastructure for deploying multiple VNs that share the substrate network resources distributed across hundreds of physical domains on the Internet. A user requests a slice on PlanetLab and selects a set of PlanetLab nodes to add to the slice. Once the user selects a set of nodes for the experiment, the user can connect these nodes together to form a VN within the slice. However, some of the nodes may fail or become unstable over time [67]. This can have a negative impact, for example, on experimental results, as well as the ability to experiment in the long-term. An effective virtual network migration system could mitigate negative effects on PlanetLab and similar platforms while facilitating application development.

To implement a virtual network on PlanetLab, each virtual server acts as a virtual router of a virtual network. One way to set up a virtual network on a PlanetLab slice is to connect these virtual servers with point-to-point tunnels. Vsys [22] on PlanetLab allows users to have access to some of the `sudo` superuser privileges which include creating virtual interfaces and tunnels, and modifying forwarding tables for the virtual servers. Isolation is provided such that these actions performed on one virtual server do not affect other virtual servers on the same PlanetLab node.

The migration on a PlanetLab slice is different from the migration with virtualized separated routers described in Chapter 3. We do not actually need to move the virtual server content within the substrate from one PlanetLab node to the other. Furthermore, we do not have access to the layer 2 traffic redirection on PlanetLab. Instead, we clone the *routing states* of the virtual routers. The routing states are the entries for the data traffic in the forwarding table with respect to the virtual node. Because each virtual node has different virtual interfaces, the forwarding table entries cannot be directly copied to the new node. Thus, we clone the routing states of the migrating nodes and translate the entries with respect to the new virtual node.

Compared with the network migration with virtualized separated routers, packet loss is observed during the traffic redirection. Because the forwarding and redirecting of traffic are done on the routing layer, a single best route is always selected from the forwarding table. The old and the new virtual nodes are not available at the same time during the migration. If some of the packets are buffered on the old virtual node, these packets can be lost during the traffic redirection. As a result, packet loss is one of the metrics to take into consideration for formulating the migration scheduling problem.

We propose a migration mechanism for VN migration on PlanetLab designed to meet these objectives. We design, implement, and evaluate with live experiments, a controller called *PL-VNM* that orchestrates the VN migration to new VN placements on PlanetLab. In principle, we can schedule the migration steps to minimize the disruption to the VN. In practice, our experiments have revealed that certain PlanetLab features make VN migration scheduling infeasible. This work leads to specific recommendations for PlanetLab that would enable VN migration scheduling, as well as recommend features for more general virtualization environments. We believe that the recommended features make long-term experiments and application deployments on shared virtualized infrastructures possible.

In this chapter, we first overview the PlanetLab architecture and how VNs are implemented on PlanetLab. Then we present the architecture of our proposed migration controller PL-VMN in Section 4.2. In Section 4.3, we discuss the challenges of VN migration scheduling to minimize the disruption to the VN in the ideal case. We further experiment with gateway scheduling in PL-VNM as the realistic case in

Section 4.4. We then give recommendations in Section 4.5 for enabling VN migration scheduling on both PlanetLab environment and generic virtualization environment. We summarize this work in Section 4.6.

4.1 Virtual Networks on PlanetLab

In this section we briefly describe two well known methods for implementing and managing VN resources in PlanetLab. We begin first by discussing the ways in which VNs may be allocated the resources that are granted by PlanetLab.

4.1.1 VN-to-PlanetLab Resource Allocation

PlanetLab uses a *slice* abstraction to gather a collection of resources across different PlanetLab components. Slices are requested by a user. Once allocated, the user can add nodes to the slice. All nodes are shared by different slices. For each node that is added to the slice, a virtual container is created. This container holds the node within the slice, and isolates the node from other slices. This virtual container can be accessed by the user with an appropriate SSH key.

Figure 13 shows two types of virtual containers. Circles, triangles, and diamonds represent the virtual containers that are allocated to slices 1, 2, and 3, respectively. The shapes colored white are not actively participating in any routing or forwarding. We call them the non-active virtual containers. The colored shapes represent the virtual containers that are allocated to the slices and have virtual routers running on them. We call them the active containers.

Consider a slice that is allocated entirely to a single VN. In this allocation the slice and the VN are identical, and include all resources, active and non-active containers. In this context the migration of a VN in PlanetLab means migrating either the slice itself, or all the active and non-active virtual containers from nodes in the current slice to nodes in another slice. Either type of migration requires changes to the infrastructure on PlanetLab and the way it assigns resources to different slices. This capability is not currently available, nor is it possible, in the PlanetLab environment.

Alternatively, it is possible to partition resources within a single slice. This feature can be used to build multiple VNs *within* a single slice. The VN is first mapped to a subset of virtual containers. These virtual containers are active and hold the VN's virtual routers. Virtual routers are connected virtual links using point-to-point tunnels. In Figure 13, a VN of slice 1 is placed on the three active virtual containers (colored circles) and the virtual links between them. The three non-active virtual containers of slice 1 (white circles) are not part of the VN. The substrate resources available for a VN placement and migration are confined to the resources of a slice. For example, migration of the VN in slice 1 can be done by migrating the active containers to the three non-active containers within slice 1.

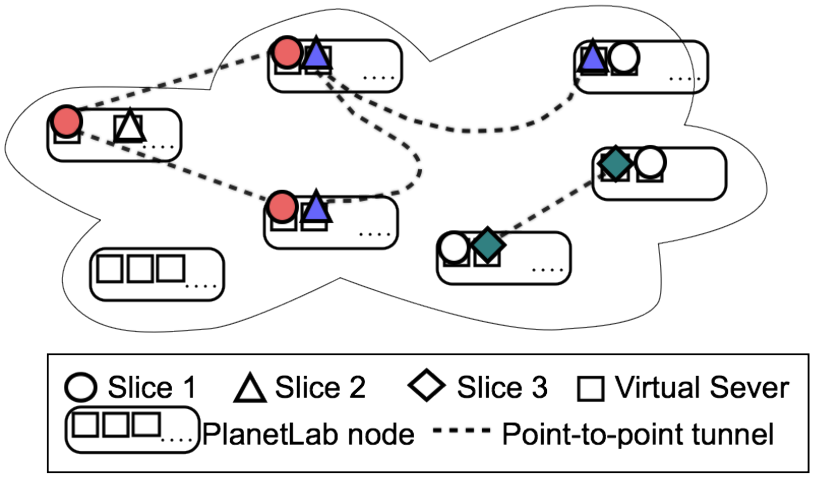


Figure 13: An example assignment of VN nodes within PlanetLab

4.1.2 Implementation and Management of VNs on PlanetLab

There are two common methods to create and manage a VN within a slice on PlanetLab. The first implements VNs in user space. This requires the creation of a virtual router in each container using the Click router [2], and connecting the virtual routers together with UDP tunnels [20]. User space implementations increase the latency of forwarding packets due to the time copying packets between the kernel and user space and also the time waiting for Click to run on the CPU.

In our design and evaluations we have selected to implement VNs in kernel space by setting up packet forwarding in kernel space using the Vsys API [22]. With Vsys, a virtual router in a PlanetLab node can install routes to the forwarding table in the kernel space. When a PlanetLab node receives packets for one of the slices it hosts, it does not copy the packets from kernel space to user space before the redirection. Instead, the packets are forwarded directly according to the forwarding table of the node. Thus, the latency of packet forwarding within a node is considerably lower compare to using the user space routers [56].

Vsys currently is fully functional on the PlanetLab Europe nodes. It allows each slice to have access to certain super user `sudo` privileges, that include creating virtual interfaces and tunnels and forwarding table entries for that slice. Each user is assigned a private IP subnet for the slice. The forwarding table in the kernel is shared among other slices. Isolation is provided through Vsys such that actions performed on a PlanetLab node in a slice do not affect other slices sharing the same PlanetLab node. A user can only modify the forwarding table entries that belong to the assigned IP subnet. With this kernel space forwarding table sharing feature, a user space router can install routes to the forwarding table in kernel space directly.

PlanetLab uses ingress filtering to prevent packets from forged IP source addresses. The packets are filtered according to the strict mode of unicast Reverse Path Forwarding (uRPF) [15]. The packets are still forwarded directly in kernel space according to the forwarding table entries. However, packets are filtered based on their source IP addresses. When a PlanetLab node receives a packet from one of its virtual interfaces, the node checks for the best route of the packet's source IP address in the forwarding table. If the virtual interface of that best route does not match with the receiving virtual interface, i.e., the arriving virtual interface is not used to forward packets to that source IP address, the packet will be dropped. If it passes the check, the node looks for the best route of the destination IP address and forwards the packet accordingly. Note that only the best route is considered. Other existing routes in the forwarding table for the same IP address or subnet with lower priority are ignored. This feature prevents asymmetric paths on the VN between two nodes and has negative implications on VN migration, as shown in Section 4.3.

To setup VNs, we use a Python Vsys API package provided by NEPI [62]. According to the VN topology, we connect the PlanetLab nodes with point-to-point tunnels through the Vsys API. We assign private IP addresses within the assigned subnet of our slice to the virtual interfaces for those tunnels. Then we install the pre-computed routing table entries to the forwarding tables of the PlanetLab nodes through the Vsys API. In our current implementation, we assume that we are using static routing. In the future implementation, we can install a routing daemon to interact with the forwarding table in the kernel space through the Vsys API for updating forwarding table entries. [1]

4.2 Virtual Network Migration on PlanetLab

We now describe the proposed virtualization architecture that supports VN migration in PlanetLab, the migration process and how to evaluate it.

[1]Prior work investigated the interaction between dynamic routing in the native and overlay layers [79]. With dynamic routing and a high node degree VN topology, the migration process can be improved through rerouting. The VN migration process is similar except for handling the HELLO messages among the virtual routers. For example, in OSPF, if the time of the migration is longer than the HELLO message timer, the effects of the migration will be recognized as a link failure or a network instability event. Currently the recommendation for the time between sending HELLO messages is 10 seconds which is longer than the time for migration, which takes less than 2 seconds. So, the migration cannot be noticeable by the virtual routers. In case it is noticeable, it will be recognized as a link failure and the traffic will be rerouted. As a result, we believe our work still applies to the virtual networks with dynamic routing.

4.2.1 Migration Architecture and Process Overview

At the center of our VN migration architecture we have designed and implemented our own migration controller, PL-VNM. Gateways are used in addition to redirect traffic between virtual networks. An example instantiation of our architecture is shown Figure 14. Shown in the upper-right, PL-VNM orchestrates the VN setup and migration remotely. It connects to all the virtual nodes in the VN and the gateways to set up the VN and the virtual links between the nodes.

We focus on connections between VN virtual routers and gateways hosted by PlanetLab nodes in the same slice. While it is possible to connect end hosts that are not PlanetLab nodes through UDP tunnels, in our experiments, we choose to maintain all the virtual nodes including the end hosts on PlanetLab. Figure 14 shows the initial and final placements of the VN and they are mapped to different sets of PlanetLab nodes. Only the VN is migrated to its final placement in our experiment. The end hosts and gateways maintain their placements.

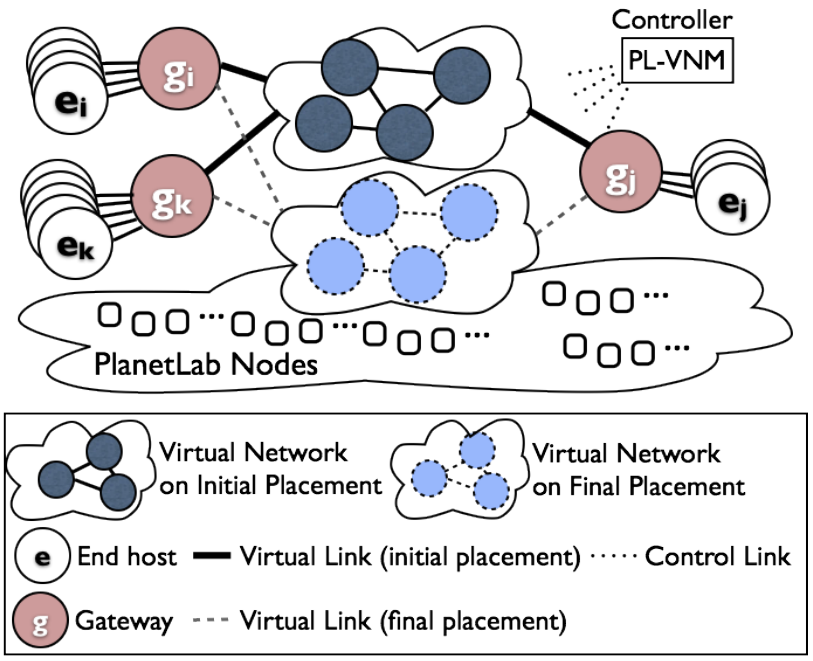

Controller
PL-VNM
g_i
e_i
g_k
e_k
g_j
e_j
PlanetLab Nodes
Virtual Network on Initial Placement
Virtual Network on Final Placement
e End host
Virtual Link (initial placement)
Control Link
g Gateway
Virtual Link (final placement)

Figure 14: An example VN migration setup in our PlanetLab experiment

In our VN migrations, we focus on network elements between gateways. Since migration is done in the IP layer, one of the roles of the gateways is to hide the IP route changes from the end host applications. If end hosts were directly connected to the VN, then when the VN migrates to its final placement, the interfaces of the end hosts would also have to adopt new IP addresses. Such changes of IP addresses adversely affect the connectivity of the applications.

Besides hiding IP address changes from end hosts, the gateways also perform traffic redirection during VN migration. The changes of routes in the forwarding tables of the gateways determine which placement of the VN the packets should be forwarded to. For example, in Figure 14, the forwarding table of gateway g_i starts with the routes to destination e_j and e_k with the next hop to the initial placement of the VN. During the VN migration, these routes are changed to reflect the next hop as the final placement of the VN.

The process of VN migration proceeds as follows:

1. Launch new virtual routers on the final placement nodes.

2. Clone the routing states from the active to the final virtual routers.

3. Redirect traffic at the gateways to the final VN placement.

During the migration, PL-VNM installs the required packages at the final placement nodes. Then PL-VNM clones the routing states [2] of the virtual routers on the VN to the virtual routers at the final placements. After cloning, PL-VNM issues traffic redirection commands to all the gateways g_i, g_j, and g_k to change the forwarding table entries with next hop to the final placement of the VN nodes.

[2]The *routing states* of the PlanetLab virtual routers are defined as the entries in the forwarding tables of the virtual routers on PlanetLab. We clone the routing states instead of copying them to the final placement nodes directly because the virtual interfaces and IP addresses of these virtual interfaces are assigned differently on each virtual node.

4.2.2 Migration Evaluation Metrics and Migration Scheduling

We evaluate migration performance by measuring packet loss and the time to complete the migration. In our architecture packet loss can occur at the gateways when the old route to the VN's initial placement is replaced by a new route to the final placement. Because the forwarding and redirecting of traffic are done at the routing layer, the packets are only forwarded to the best route from the forwarding tables of the gateways. Furthermore, because of the aforementioned uRPF check, packets that are buffered on the initial placement of the VN can be dropped or lost during the traffic redirection.

In prior work [57, 54], it has been established that migration scheduling, i.e., the sequence and timings of gateway redirection requests can have a significant effect on migration performance. Also, there is often a tradeoff between migration duration and VN disruption as measured by packet loss. For example, in our system if all gateways are asked to redirect their traffic at the same time, this will cause the migration to complete quickly. This, however, is likely to cause the most packet loss to applications.

Implementing a migration schedule, however, requires the network control (to deploy the required traffic redirection) at small time granularity. As it turns out this is very challenging in the PlanetLab environment. In our work we explore two possible control strategies and evaluate them experimentally as is described in Section 4.4.

4.2.3 The PL-VNM Controller

Our controller, PL-VNM, performs both VN installation and migration. During a VN installation or migration, PL-VNM connects to all the virtual routers and gateways through SSH connections to initiate and synchronize the VN installation or migration procedures. We implement PL-VNM with the Python Vsys API package. The package has to be installed on all nodes in the slice. The user also has to request to add the Vsys tags to the slice.

Figure 15 shows its detailed architecture. PL-VNM stores the VN topologies, substrate network topology, and the mapping of the VNs with SQLite database [7]. The *VN orchestrator* installs the VNs according to the mappings with the Python Vsys API to the PlanetLab nodes. The VN orchestrator installs the virtual links between the virtual routers and installs routes to the forwarding tables of the virtual nodes through the API.

When a request for VN migration is initiated with initial and final VN placements, PL-VNM schedules and automates the migration. The *migration routing state translator* identifies the routing states that are affected at the gateways and translates these routes according to the final placement of the VN. With the routing state changes from the migration routing state translator, the *migration orchestrator* first computes a *migration schedule*, and orchestrates the migration according to the schedule. The migration orchestrator installs new translated routes to the VN's final placement PlanetLab nodes. After that, according to the migration schedule, it sends redirection commands to the gateways to redirect traffic to the VN's final placement. These commands change the routes in the forwarding tables of the gateways. Once that is finished, the routing state updates are sent back to PL-VNM with the final VN mapping.

In our prototype, we have two choices of implementation for controlling traffic redirection at the gateways. The first choice is to execute the commands according to the schedule produced by the algorithm at PL-VNM through SSH sessions to the gateways. With this technique, called *remote scheduling*, there is a lag between the time the control command is issued by PL-VNM and the execution time of the command at the gateway. This lag can be considerable and unpredictable and makes it difficult, if not impossible, to control the actual migration timing.

By contrast we schedule at the gateway nodes with `at` job scheduling utility [1].

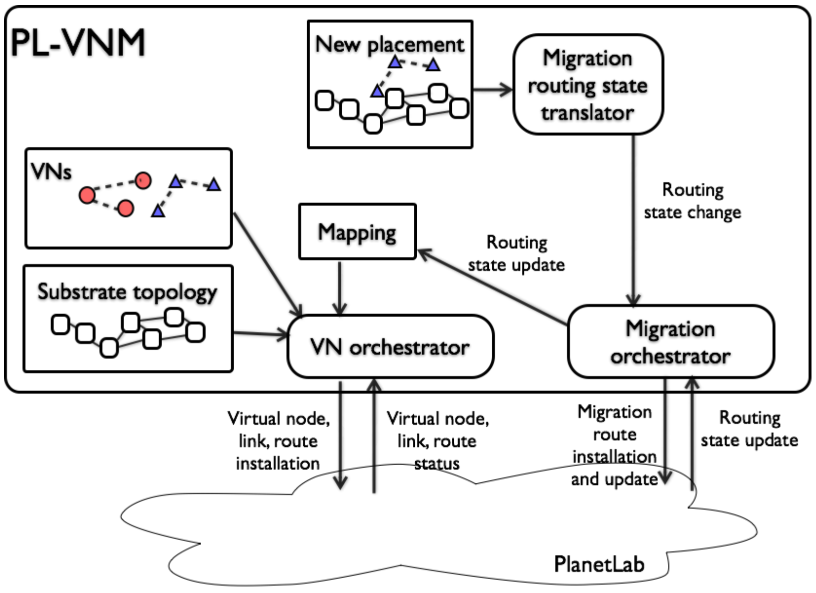


Figure 15: The detailed architecture of PL-VNM

With this technique, called *gateway scheduling*, the gateways should be synchronized through NTP [63]. PL-VNM computes the migration schedule and uses `at` to schedule the commands at different gateways. In this case the latency between PL-VNM and the gateways is not an issue. However, OS process timing resolution and NTP accuracy can, again, make it hard to control migration timing. We explore the relative performance of these two approaches in our experiments in Section 4.4.

4.2.4 Our Process for VN Migration in PlanetLab

The migration process is described by the steps below, and accompanied by matching depictions of each step in Figure 16:

1. Add the PlanetLab nodes for the new node placements to the slice such that the virtual containers for the slice are created on those nodes. This step can be done through the PlanetLab API or the PlanetLab website.

2. Set up the new VN on the final placements of the virtual nodes:

   (a) Set up virtual interfaces on each new virtual router according to the configuration of the old virtual routers.

   (b) Connect the virtual routers at their final placements according to the VN topology with point-to-point tunnels through the Vsys API.

3. Clone the routing states of the old virtual routers to the new virtual routers of the VN. The routing states cannot be directly copied from the old virtual router because the virtual interfaces of these states are different.

4. Establish point-to-point overlay links between the gateways to the VN on its final placement.

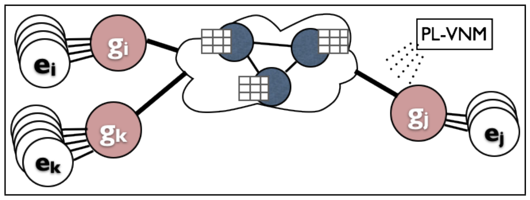
5. PL-VNM issues commands to the gateways according to the schedule to redirect traffic to the new VN. These redirection commands change the forwarding table entries on the gateways between the VN and the end hosts. This step requires synchronization among all the gateways.

6. Disconnect the old VN from the gateways and remove the routing states on the old virtual routers.

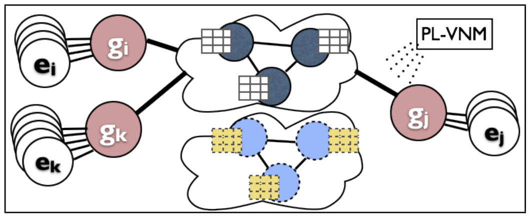
Step 1-4 and 6 can be done in the background. Step 5 requires accurate redirection timing in order to implement a given migration schedule.

4.3 VN Migration Scheduling Challenges in PlanetLab

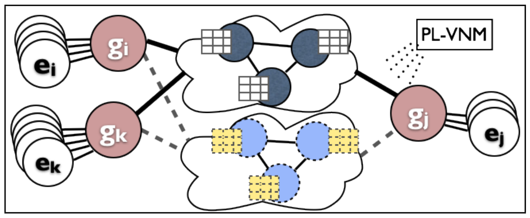
We illustrate PlanetLab VN migration scheduling challenges here through a 2-node VN example as shown in Figure 17. We consider $f_{1,2}$ a traffic flow from end host e_1 to e_2. $f_{1,2}$ has a virtual path of $(e_1, g_1, A, B, g_2, e_2)$. Nodes A and B move from an initial placement to a final placement through redirection of traffic at both g_1 and g_2. PL-VNM issues traffic redirection commands to g_1 and g_2. We assume these redirection commands are executed (*not* issued by PL-VNM) at times t_1 and t_2, respectively.



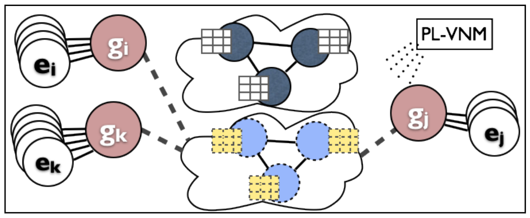
(a) A VN with three nodes on their initial mappings serving three sets of end hosts.



(b) Step 1-3: Set up the new VN on the final placements of the virtual nodes. Clone the virtual nodes' routing states to their final placements.



(c) Step 4: Establish point-to-point overlay links between the gateways g_i, g_j and g_k to the VN on its final placement.



(d) Step 5-6: At the gateways, redirect traffic between the end hosts by switching the next hops of the forwarding table entries from the VN's initial to the final placement. Disconnect the gateways from the initial placement.

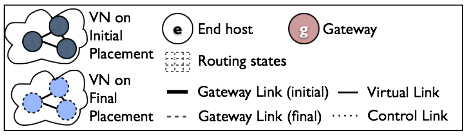


Figure 16: Migrating a VN with three virtual nodes serving three sets of end hosts e_i, e_j and e_k.

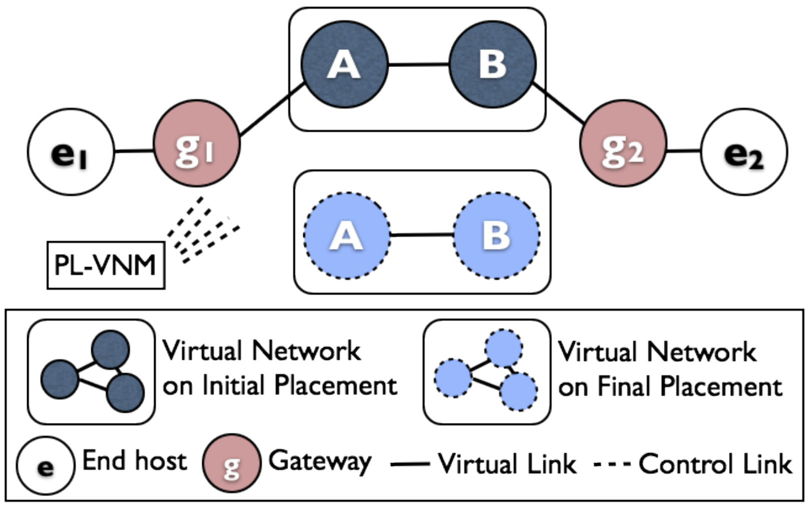

A
B
e1
g1
g2
e2
PL-VNM
A
B
Virtual Network on Initial Placement
Virtual Network on Final Placement
e End host
g Gateway
Virtual Link
Control Link

Figure 17: The topology of the 2-node VN on PlanetLab

We define l_i and l_f as the one-way path latencies of the virtual paths of the flow $f_{1,2}$ from e_1 to e_2 at the initial placement and final placement, respectively. Assuming that the capacity of the virtual paths (g_1, A, B, g_2) of the initial and final placements are the same, we consider three cases:

1. When $l_i = l_f$: After g_1 switches from the initial mapping path to final mapping path at t_1, the last packet traveled on the initial mapping path takes l_i to arrive to g_2. At the same time, the next packet (after the redirection) takes the final mapping path and arrives to g_2 at $t_1 + l_i$. Since $l_i = l_f$, g_2 should switch to the final path at time $t_1 + l_i$ to prevent packet loss.

2. When $l_f > l_i$: Similar to the previous case except that the packets on the final mapping path take a longer time to arrive to g_2. As a result, the time for g_2 to switch to the final path should fulfill $t_1 + l_i < t_2 < t_1 + l_f$ or $l_i < t_2 - t_1 < l_f$.

3. For the case of $l_i > l_f$, packet loss is unavoidable. Once g_1 has issued the redirection command at t_1, the packets after t_1 switch to the final mapping path with latency $l_f < l_i$. The first packet on the final path reaches g_2 earlier than the last packet on the initial path. No matter when g_2 switches the path,

packet loss would be observed. To minimize packet loss, g_2 should switch the path between t_1 and $t_1 + l_i$, i.e., $t_1 \leq t_2 \leq t_1 + l_i$.

Note that the above analysis is for one direction of traffic flow. Avoiding packet loss in the opposite direction of traffic (from e_1 to e_2 in the above case) will most likely require different timing considerations for the gateway redirection. Because PlanetLab requires symmetric paths, it is not possible to manage the two flow directions independently.

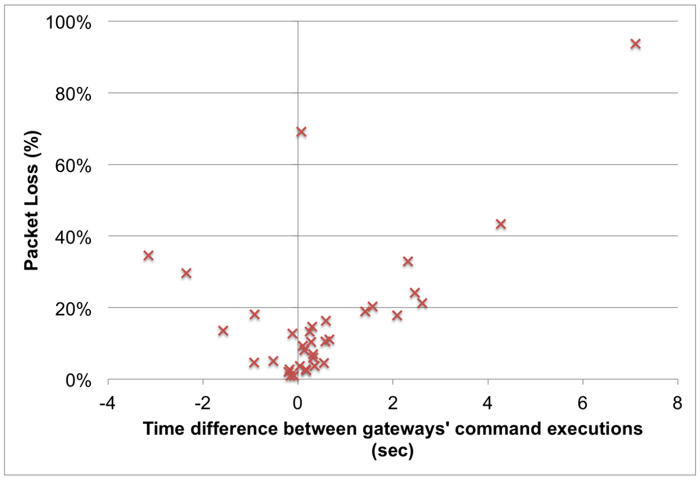


Figure 18: Percentage of packet loss vs. redirection command execute time at g_2 minus redirection command execute time at g_1, i.e., $(t_2 - t_1)$

.

4.3.1 Experimental Results

We now show results from an experiment on the topology in Figure 17. We migrate the VN from its initial placement to its final placement 35 times. We use iperf to send a UDP flow with 1.5 Mbps rate from e_1 to e_2 for 10 seconds. PL-VNM issues the redirection commands to the two gateways at the same time. Under ideal conditions this will cause the simultaneous redirection at both gateways. However, as noted earlier, various factors can cause the actual execution times to be quite variable. In fact, we observe values for $t_2 - t_1$ ranging from -7.1 to +3.2 seconds. This confirms our earlier assessment of the difficulty of controlling migration timing in PlanetLab.

Figure 18 shows a scatter plot of the packet loss versus the time between the redirection commands executed on each gateway (i.e., $t_2 - t_1$). Each point on the figure represents one measurement result. The path latencies in the initial and final placements are 240 ms and 160 ms, respectively. We measure the percentage of packet loss within 10 seconds of the traffic redirection. This falls under case 3 in the discussion above. Thus we expect packet loss to be minimized when $t_2 - t_1$ is less than 240 ms. Indeed we do observe packet loss at a minimum (with some experimental variation, at and around this interval). The minimum time difference of $t_2 - t_1 = 89$ ms gives a minimum packet loss of 0.009%. However, there are also other cases with roughly a time difference of 75 ms with 79% packet loss.

4.4 Gateway Scheduling in PL-VNM

In this section, we evaluate whether gateway scheduling (where redirection tasks are scheduled in advance at the gateways themselves) can provide better timing control for VN migration in PlanetLab. We do this using our PL-VNM prototype to control a 3-node VN migration experiment (Figure 19). The VN has three virtual routers that are connected to three end hosts through three gateways. In our experiment, we migrate the VN between its initial and final placements back and forth 110 times. We also use iperf to measure the packet loss for 60 seconds between all pairs of end hosts, i.e., total of 6 UDP flows with 1.5Mbps rate among 3 end hosts.

4.4.1 Remote Scheduling Baseline

We first run baseline experiments using remote scheduling. During the migration, PL-VNM first issues commands to clone the VN on the VN's final placements. Then PL-VNM issues redirection commands to the gateways for switching the flow to the final VN placement at the same time through remote scheduling (SSH). We record the time of command execution at the gateways and measure the packet loss for the flows.

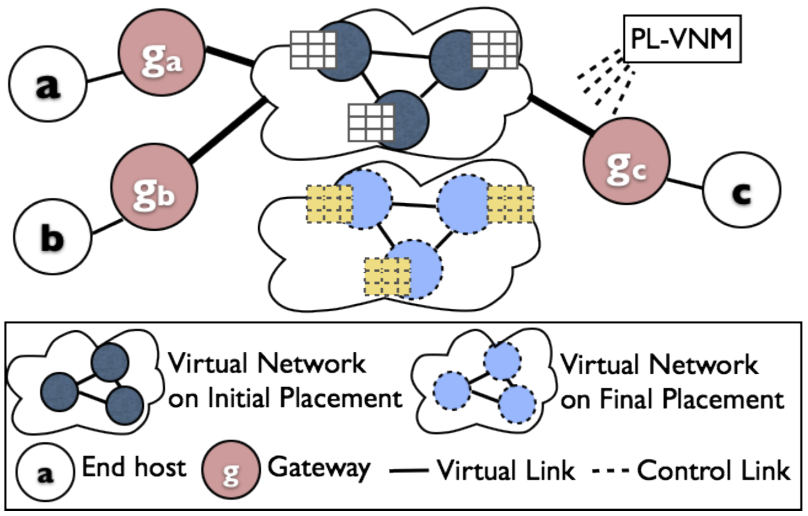

a
ga
b
gb
PL-VNM
gc
c
Virtual Network on Initial Placement
Virtual Network on Final Placement
a End host
g Gateway
Virtual Link
Control Link

Figure 19: The topology of the 3-node VN in our PlanetLab experiment

Figure 20 shows the result of this experiments with remote scheduling. The percentage of packet loss on the y-axis is based on the total measurement time of 60 seconds with iperf. The x-axis shows the time difference between the traffic redirection command issued times to the gateways for the corresponding flows. Even though the redirection commands are issued at the same time, we find the time differences ranging from -7.8 to +13.1 seconds and again the time differences illustrate the serious timing problems of the remote scheduling approach. Flows from node b are not shown in the figure because we notice that flows from node b to any other nodes experience high packet loss. The loss rate on these two flows are too high even there is no migration.

We observe that when there is almost zero time difference between the redirection command execution time at the two related gateways, the packet loss is close to zero. This is because the difference in latencies between the initial and final placement of the paths is also small (tens of milliseconds). We still observe cases that time difference is close to zero while 10% packet loss is observed. This effect can be caused by a background event happening in the substrate network.

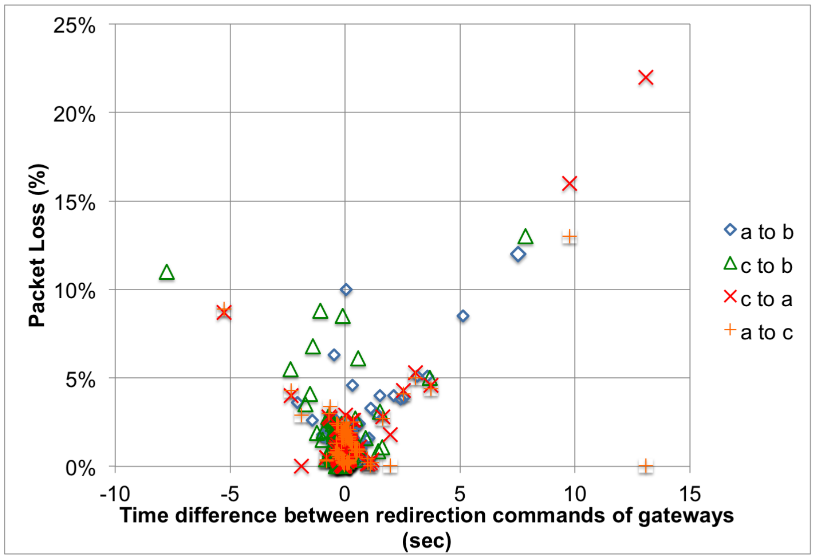


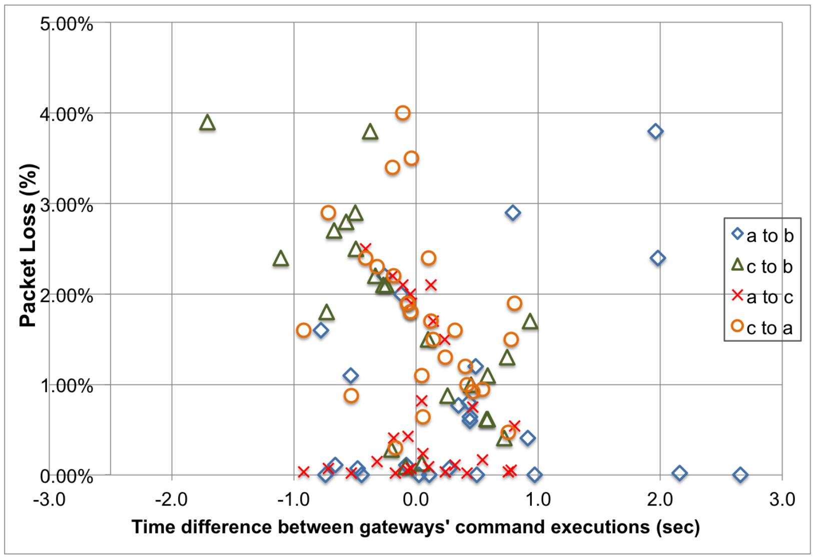
Figure 20: Packet loss vs time difference between the execution time of the related gateways' redirection commands using PL-VNM remote scheduling on the 3-node VN with only 4 flows

4.4.2 VN Migration Scheduling through Gateway Scheduling

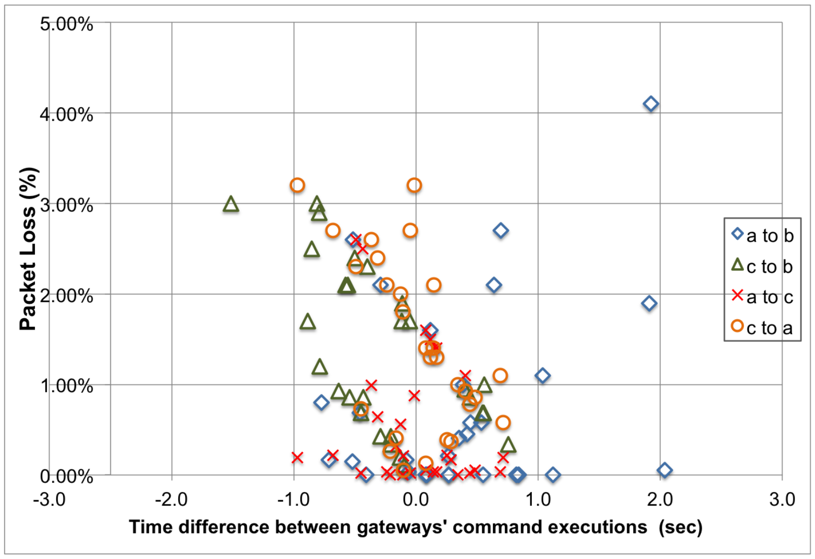
We now consider migration control via gateway scheduling. We run a set of experiments on the same topology as before. However, we schedule the traffic redirection commands ahead at the gateways with `at` utility at the same time. All the PlanetLab nodes are synchronized with NTP so that the error of the time is limited to 100 ms.

Figure 21(a) shows the results of the experiments that migrating the VN from its initial to its final placement. Figure 21(b) shows the results of the experiments that migrating the VN from its final to its initial placement. First, the range of time difference is within -1.7 to +2.7 seconds given that the commands on different gateways are scheduled to be executed at the same time. The command execution timings behave much better than in the remote scheduling case. This also causes packet loss to be much less with this control paradigm.

The time variation is caused by the accuracy of NTP synchronization and the load on gateway CPUs. The resources including CPU time and memory of the gateways



(a) Switching from the initial to the final placement



(b) Switching from the final back to the initial placement

Figure 21: Packet loss vs time difference between the execution time of the related gateways' redirection commands using gateway scheduling on the 3-node VN with only 4 flows

are shared among different PlanetLab slices. If the gateways are executing some other jobs for other slices, the traffic redirection commands will be prioritized to be executed after the other jobs. Thus, the execution time will be delayed.

Even though timing control is significantly better with gateway scheduling, we still believe it is not adequate to provide the fine-grained control required to minimize loss caused by VN migration.

4.5 Recommendations for Platforms to Support VN Migration

When we started this work, our goal was to complete our prototype with a migration scheduling heuristic and to demonstrate how this heuristic can be used to minimize disruption in VN migration on PlanetLab. It is relatively straightforward to take the analysis of the scenarios in Section 4.4 and produce a scheduling heuristic that under ideal conditions will provide the least packet loss when migrating a VN. We show a sketch of such a heuristic in the Appendix. However, on shared infrastructure, the difficulty in controlling redirection timing due to network latency and resource availability makes this an uninteresting exercise.

Instead we conclude by giving some recommendations about substrate features that are required to enable VN migration. Some of these are feasible to deploy on PlanetLab while others point to design elements in a future infrastructure such as GENI [42].

4.5.1 Providing Improved Timing Control in Substrate Nodes

The performance and time to execute the migration command is constrained by available synchronization and scheduling mechanisms. We recommend the following features at the substrate network to provide improved timing control. These features also apply to a general substrate infrastructure that supports VN migration.

4.5.1.1 Reduce clock granularity to facilitate synchronization

VN migration scheduling as described in Section 4.4 depends on the path latencies between end hosts. The path latencies typically encountered are in the few hundred milliseconds range. By contrast, scheduling with the `at` utility is limited to a granularity of seconds. Our evaluations suggest that a job scheduler with a granularity of 1 ms at the gateways is appropriate. Quartz, for example, is a commercial-grade java-based job scheduler that shows job scheduling with a granularity of 1 ms is feasible [6].

4.5.1.2 Enable prioritization of migration commands at gateways

We observe that the time to complete migration commands is characterized by delayed starts in addition to slow execution. Time averages measured over 110 executions are summarized in Table 4. Lag time is the time between the scheduled start and actual start times, and averages at least 0.5 seconds at each of the gateways. Furthermore, we see in Table 4 that execution time is highly dependent on load, with values ranging from a few milliseconds at gateway g_c, to a many hundred milliseconds at remaining gateways. Prioritization of migration commands would reduce these timing effects within a slice. The ideal is for prioritization commands to be respected at the substrate layer, in addition.

Table 4: The average lag time between command issue time and the time the gateways start executing the commands and the average redirection command execution time at the gateways

	g_a	g_b	g_c
Average lag time (sec)	0.536	0.869	0.564
Average execution time (sec)	0.813	0.702	0.066

4.5.2 An Asymmetric Routing Policy to Reduce Packet Loss

Currently, PlanetLab supports only symmetric routing, a policy that is enforced by way of the strict mode in uRPF [15]. This policy insists that changes in forwarding behavior must be synchronized across gateways with matching segments along forward and reverse paths, as described in Section 4.1.2. At any time, if the gateways are not synchronized with symmetric paths, i.e. one gateway is using the initial placement to forward packets while the other one is using the final placement, the traffic flowed through these gateways will be dropped. During migration this causes long contiguous steams of lost packets.

To prevent packet loss during the traffic redirections we recommend support at the gateways for feasible mode (see [15]) in uRPF in addition to strict mode. In feasible mode, asymmetric routing is allowed. As long as the forwarding tables maintain simultaneous routes to the initial and final placements, packets will pass the checks and will be forwarded.

4.6 Summary

We believe that our work makes long-term experiments and application deployments on shared virtualized infrastructures possible. We propose a VN migration mechanism for PlanetLab. We design and implement a controller called PL-VNM to orchestrate the VN migration. We evaluate our work with experiments. Although we only perform small scale experiments, the challenges also apply to public networks that support virtualization through the routing layer. Ideally, a migration schedule would minimize the disruption to the VN. However, certain PlanetLab features make migration scheduling infeasible. We further recommend features for general virtualization environments to enable VN migration scheduling.

We note an alternative approach may exist by way of collaboration between the VN migration controller and the end systems. This may be particularly advantageous in cases where it is possible to recover gracefully from the packet loss that occurs within the short disruption period that follows a redirection. Applications that deliver data with an on-off behavior are appropriate targets, with dynamic adaptive streaming over HTTP (DASH) as just one example [80]. In this context a VN controller could trigger migrations between the transmission of video segments. Additionally, the controller could notify the end systems of impending migrations and the need to buffer packets locally until the migration is complete. We leave this investigation for future work.

CHAPTER V

TECHNIQUES TO MITIGATE PERFORMANCE DEGRADATION THROUGH TRANSPORT AND/OR APPLICATION LAYER COLLABORATION

Planned device deployment and maintenance, virtual network migration, middleboxe migration, and route updates and convergence introduce disruptions to the application traffic. For example, our previous work in Chapter 4 shows that it can take up to 10 seconds for a virtual network to finish its migration [58]. The convergence time of RIP can take over 30 seconds [24]. Sun et al. show that BGP takes over 70 seconds for route convergence during a new route propagation [83]. These disruptions affect the performance of the applications. The network bandwidth at the time of convergence is highly affected [69].

VN migration leads to unexpected packet loss and bandwidth degradation for the application running on the VN. We rely on the transport layer to mitigate the effects from these disruptions. TCP can mitigate some of the effects of the path quality changes and even some path changes. It recovers packet loss through retransmission. However, TCP's exponential backoff introduces a longer period for recovery and prevents a connection to recover on time after a network event. TCP's congestion avoidance leads to additional bandwidth reduction and additional latency. The VN migration mechanism from our PlanetLab implementation introduces short term packet loss [58] during VN migration. Figure 22 shows a TCP trace captured during a VN migration. Because of packet loss, the sender retransmits the packet. However, the first retransmitted packet is lost and leads to the long recovery for TCP throughput. A new protocol, QUIC was developed to address this issue [76]. However, if we do not have access to the kernel of the virtual machine, we do not have the privilege to make changes to the TCP stack.

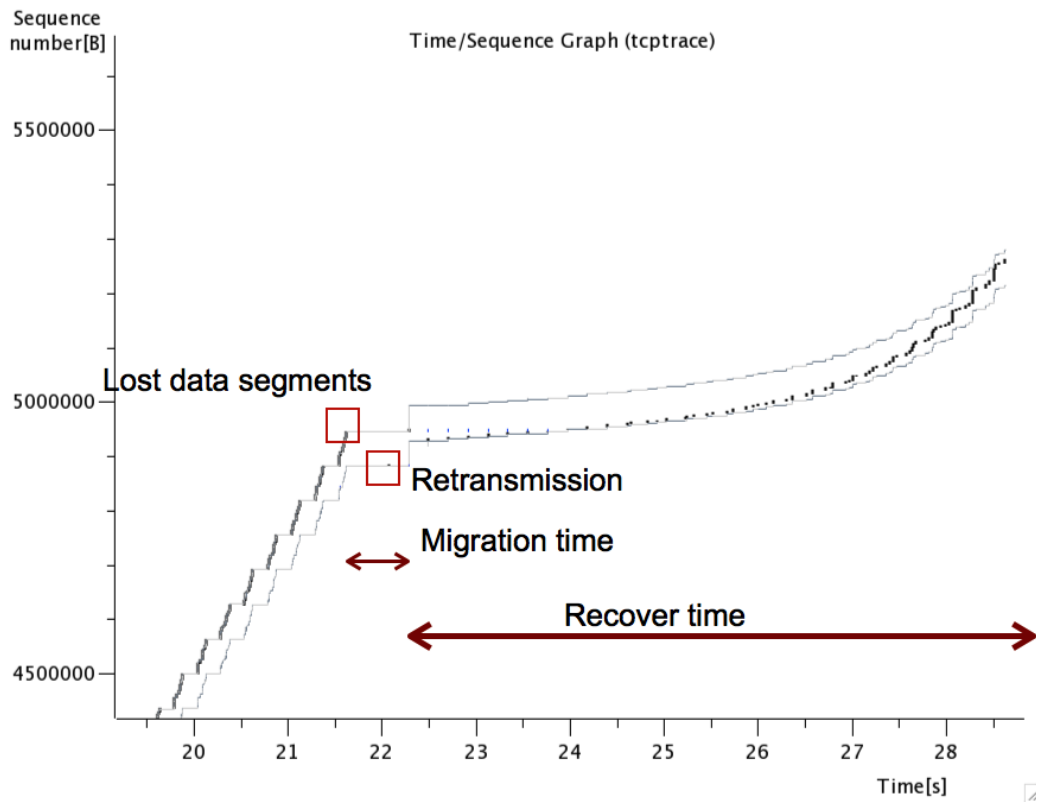


Figure 22: TCP trace captured at the sender during VN migration

In this chapter, we consider the problem of mitigating the effects of scheduled network disruptions, specifically, the effects to the transport and application layers. TCP uses packet loss as an indicator of a congested path. However, TCP's timeout mechanism as well as slow start defers the recovery process and postpones the process for the session to fully recover from the disruption. When there is a network disruption, the packet loss can be significant during the network disruption period. After the network disruption is over, the recovery at the application can take much longer because of TCP's slow start. Although adjusting the TCP socket options or using different TCP variants may solve the problem [43], an application running in a virtual machine or on an application layer usually does not have the privilege to access the kernel to make changes to the TCP socket options. Thus, our challenge is to prevent packet loss and TCP retransmission timeout from the application layer to mitigate the effects of network disruptions to the application traffic.

To mitigate the effects of any scheduled network disruption, we propose a mitigation system with two solutions to assist network system adaptation to network changes on application layer. Our two proposed solutions include a controller to signal the end-systems or gateways about the network disruptions. In our first solution, the controller sends commands to the decision agents at the end-systems about the network disruptions. The agents at the end-systems notify the applications to pause and resume after the disruptions, implemented via a user-level library and API between applications and the agents.

Our second solution does not require any changes on the end-systems. This solution is similar to split TCP [27] except that our controller notifies the gateways about the time of the network disruptions. The end systems are connected to the network through gateways. The gateways can pause, buffer, and resume the application traffic when the gateways receive signals from the controller. Our proposed architecture also allows an further expansion on forwarding the buffered application traffic temporally to middleboxes and cloud storage. The challenges of designing this type of system include accommodating scalability, managing the timing of the control messages, buffer estimation, and network condition estimation. We implement these two solutions on GENI [42] and we show that our solutions mitigate the effects of scheduled network disruptions effectively.

The rest of this chapter is arranged as the following. In Section 5.1 we describe the system architecture of our designs. We present our experiment setup in Section 5.2. Then in Section 5.3 we show the results of our experiments. We further discuss the results of this work in Section 5.4 and summarize this work in Section 5.5.

5.1 System Architecture

Our network architecture is shown in Figure 23. The *end-systems*, including end-hosts and servers, are connected to a *virtual network* (VN) through *gateways*. The end-systems use the VN to transfer application traffic. An end-system connects to a gateway to connect to the VN and other end-systems. The gateways simply forward the application traffic between the VN and the end-systems. The VN can be an overlay network or a network sharing physical network resources with other VNs [78].

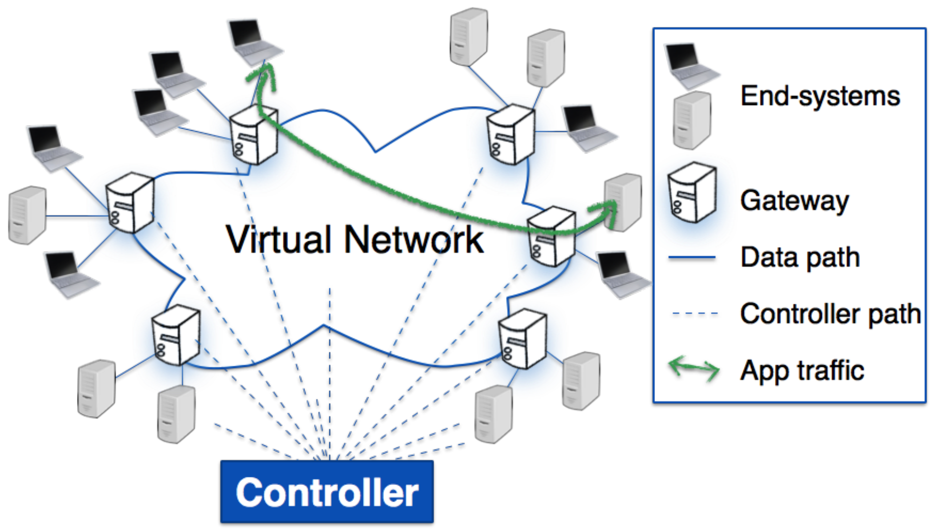


Figure 23: Virtual network with mitigation system

When we perform any maintenance on the VN, for example, migrating the VN, the maintenance introduces disruption to the application traffic. These disruptions introduce packet loss and packet reordering to the application traffic. To mitigate the effects of these planned disruptions, we propose a mitigation system with two solutions to prevent packet loss such that we can lessen the effects of these disruptions.

Our mitigation system consists of two major components: 1) A centralized controller and 2) decision agents.

Centralized Controller The centralized controller consists of two main parts: 1) instructs network events and changes configuration to the VN, 2) notifies the decision agents at the end-systems or gateways about when a network disruption starts and stops. It notifies the decision agents of the end-systems or the gateways that are affected by the disruption to take action to prevent packet loss to application traffic. We assume that the controller has all the information about the expected and actual duration of the network disruption. The controller also keeps track of the end-systems that subscribe to the mitigation service. It has the information of the end-systems that subscribe to the service and only notifies the involved end-systems.

Decision Agent Depending on the system design, the decision agents are installed at the end-systems or gateways. They are responsible for pausing the applications in the end-systems or buffering packets at the gateways to reduce packet loss. The decision agent provides some intelligence to judge whether it should pause an application based on the information from the controller. For example, if the disruption is not expected to be long enough to affect an application, the decision agent can decide not to pause that application.

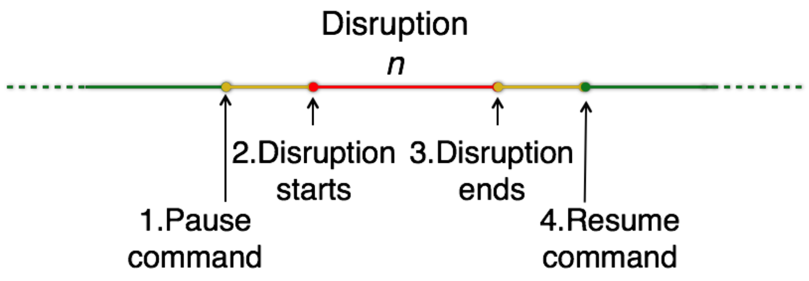


Figure 24: Mitigation system time line

5.1.1 Timeline of a Scheduled Network Event with the Mitigation System

Figure 24 shows the timeline of the migration system. The time line of mitigating a scheduled network event is as the following:

Step 1 The controller sends a pause command to the decision agents at the end-systems or the gateways.

- With different design solutions, the decision agents pause or buffer the application traffic if the applications have signed up for the mitigation service.
- The controller can include the expected duration of the disruption in the pause command. The decision agents can use this information to compute the timeout of this disruption period if the resume command is lost later in the forth step.

Step 2-3 The controller executes the scheduled network disruption commands such as VN migration, planned network maintenance, and virtual router migration.

Step 4 After the planned network disruption ends, the controller sends a resume command to the decision agents.

- The decision agents resume the application traffic once the resume command is received.
- If the resume command is lost, the decision agent can wait until after the timeout period and resume the application traffic. The decision agent can determined the length of the timeout period with the information provided by the controller. For example, if the controller has provided the expected duration of the disruption in the pause command in step 1, the timeout period can be in a multiple of the expected disruption duration.

We propose two solutions in this mitigation system using these two components to mitigate the effects of these planned network disruptions.

5.1.2 Solution 1: End-system Application Aware

Figure 25 shows the architecture of the end-system application aware design. Each end-system operates some applications which require the support from the decision agent. These applications register with the decision agent. Between the end-systems and the controller, the controller paths forward the control messages to the decision agents.

When the controller sends a pause signal to the decision agent, the decision agent sends a signal to the registered applications to pause these applications for sending any more packets to the end-system's socket. After a network disruption, the controller sends the decision agent a resume command so that the applications can resume sending packets to the socket. In this design, the applications have to be modified to accept signals from the decision agent.

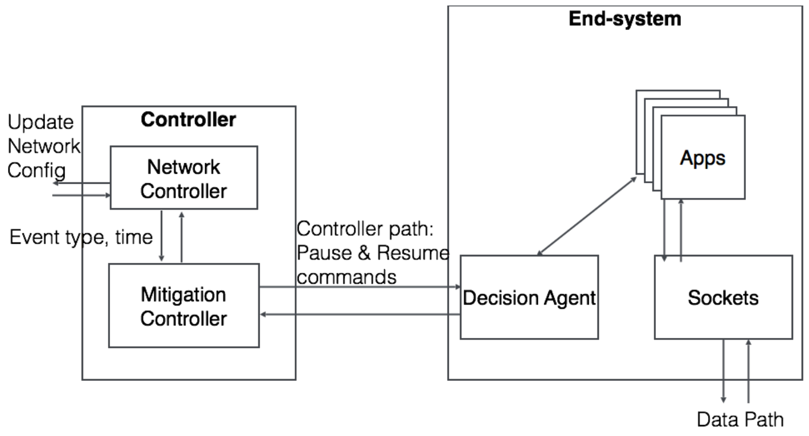


Figure 25: System architecture of end-system application aware design

5.1.3 Solution 2: Gateway Pausing and Buffering

In this design, we utilize the properties of split TCP [16, 27]. The traffic flow between an end-system sender and receiver is split to three TCP sessions: 1) the end-system sender to its gateway, 2) the gateway of the sender to the gateway of the receiver through the VN, and 3) the gateway of the receiver to the end-system receiver (Figure 26). This solution is similar to the work in performance enhancement proxies [25] and M-TCP [27] except the controller actively notifying the gateways about the disruption.

We can use DNS to redirect the end-system's requests to its closest gateway. This technique is similar to how current CDNs utilize DNS to redirect users to the edge servers [64]. In our design, a gateway, on behalf of the end-system sender, sets up a TCP session to the end-system receiver's gateway. The second gateway sets up the third TCP session with the end-system receiver. We call the entire session between a sender and a receiver through the two gateways and the VN as *an application session*.

To enable mitigation, the decision agents are operated by the gateways which connect the end-systems to the VN. Figure 27 illustrates the system architecture of this design. When the decision agent receives a pause command from the controller, the gateway can buffer the packets or forward the packets to the cloud or middleboxes for storage. The gateways can also send packets with TCP ECN to notify the end-systems to change its TCP behavior to prevent the socket to continue sending packets to the network. However, in this work, the gateways do not directly modify the TCP sockets of the end-systems.

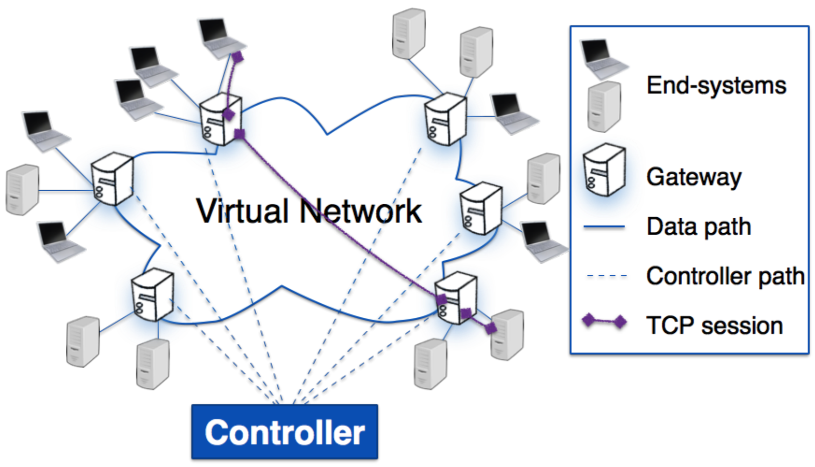


Figure 26: Network architecture of gateway pausing and buffering design

In this design, the challenges come down to the control of the buffer at the gateway, especially the buffer draining rate and the buffer size. After a network event, the controller sends a resume command to the gateways to resume the application traffic. The gateway has to drain the buffer fast enough and to resume forwarding the new packets from the sender. If the gateway's buffer draining rate is the same as the sender's sending rate, the gateway's buffer would never be able to drain. However,

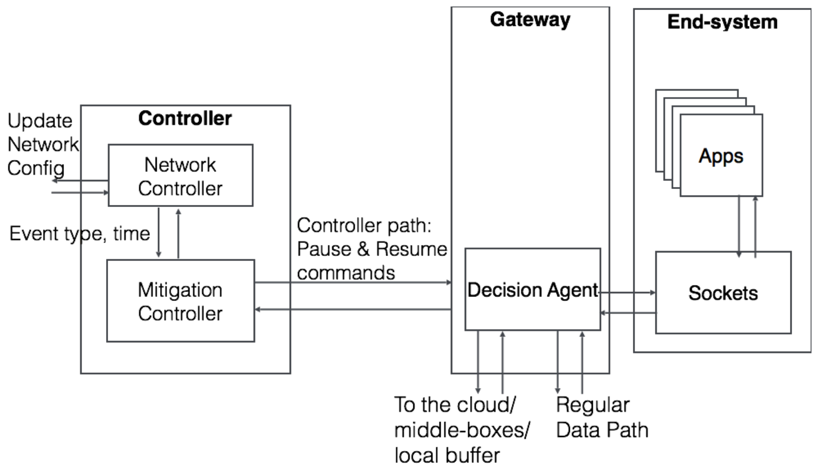


Figure 27: System architecture of gateway pausing and buffering

this buffer draining rate is limited by the VN's available bandwidth. It becomes one of the settings for the gateways that we have to intelligently adjust.

The gateway's buffer size is another setting that requires attention. When the gateway has unlimited buffer, it can continue receiving data from the sender no matter whether the gateway has been paused or not. However, if the gateway has limited buffers, the buffer size should be adjusted based on the characteristics of the VN, the applications, and the disruption duration. We will present our findings about buffer size in the following sections.

5.2 Experiment Setup

We implement and evaluate the end-system and gateway buffering designs in the GENI testbed in this work. Figure 28 shows the experiment topology. We prototype the designs with two clients, a sender (A) and receiver (B). Between these two end-system nodes, we have two gateways, G1 and G2 which are connected through a node that simulates a VN. We use iptables and tc on the VN node to change the link condition between the two gateways. Our controller controls both the network disruption at the VN and the pause/resume notification to the client application for our measurement purpose.

We implement and test our solutions on GENI [42]. With GENI, each node on our topology is a virtual machine. GENI uses OpenVZ and Xen as the virtual machines. By default, CUBIC TCP [47] is installed on all the virtual machines. Even though it is possible to change the TCP socket options from the guest OS, we do not have the permissions to do so. As a result, we implement our two designs solely in the application layer.

The controller paths between the controller and the elements in the network use TCP. It prevents packet loss during the experiment and provides a reliable acknowledgment after the end hosts or gateways receive the commands from the controller.

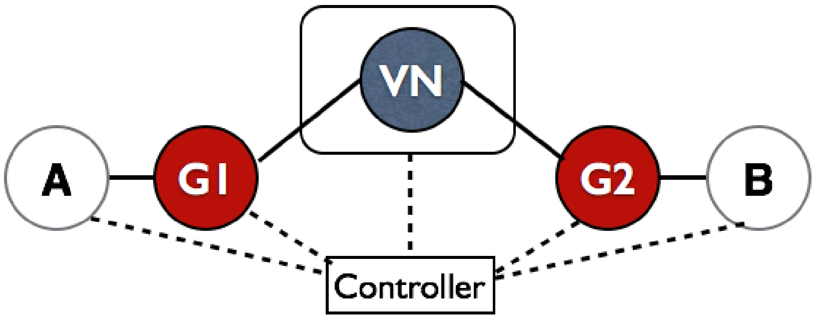


Figure 28: Experiment topology

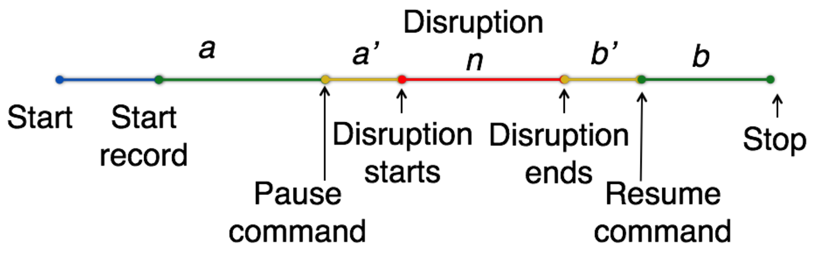


Figure 29: Experiment timeline

In our experiment, we use the following process (Figure 29):

1. The controller sends a start signal to the participating nodes which include the sender, receiver, and the gateways.

2. After the start period s, the controller sends a "record signal" to the hosts

and gateways to start recording all the events and number of packets sent and received by the hosts.

3. After the record period starts for a period of a, the controller sends a pause command to the participating hosts.

4. After the pause command is sent for period a', the controller sends a disruption signal to the VN to add the iptable rule for dropping packets.

5. After the network disruption period n, the controller sends a command to the VN to remove the iptable rule for dropping packets.

6. After a period of b', the controller sends a resume command to the participating hosts. The hosts can resume their normal operations.

7. After a recovery period b, the controller sends a stop command to all the hosts and gateways. The end hosts and gateways record all the results from the period that starts the recording to the stop.

In our experiments, we keep the period from starting the recording to stopping the experiment, $a + a' + n + b' + b$, to $30 + a' + b'$ seconds. Originally we want to keep the entire time to 30 seconds. However, a' and b' are smaller than 1 second. With the small adjustment of a' and b', we find it difficult to adjust the other variables to match the total 30 second period. So, we keep the period $a + n + b = 30$ seconds and keep a' and b' constant. We adjust the length of the disruption duration n from 1 to 10 seconds in our experiment. Because we want to keep the entire measurement period to 30 seconds, we vary a and b with each disruption duration setting accordingly. For example, when the disruption duration is set to 1 second, we set $a = 14$ seconds and $b = 15$ seconds such that $a + n + b = 30$ seconds.

In our experiments, we design our own application to transfer data from the sender to the receiver. We control the sending rate of this application. The application also pauses and resumes the sending if it receives a command from the controller. With our own application, we get all the information from the application such as the number of application data packets have been sent and received.

End-system app-aware design Only the applications at the end-systems respond to the commands from the controller. After the application receives a pause command, it stops sending packets to the socket. The gateways simply forward the packets without participating in the mitigation system.

Gateway pausing and buffering When the gateways receive a pause command from the controller, the gateways stop forwarding and start buffering the packets from the end-systems. This buffering is implemented at the application layer. Once the gateways receive the resume signal from the controller, the gateways start draining the buffer and continue forwarding packets from the end-systems. In our implementation, each application session has its own buffer at each gateway.

We design the application at the sender to directly connect to the buffering pausing application at the gateway. Between the two gateways, we implement a forwarding, pausing and buffering application. The gateway application forwards the packets received from the sender. When the controller sends a pause command to this gateway application, the gateway stops forwarding packets and buffers the packets received from the sender at its local buffer. The gateway application has three settings to 1) pause the application traffic without buffering the packets, 2) pause and buffer the application traffic with limited buffer size,or 3) pause and buffer the traffic with unlimited buffer size.

5.3 Experiment Results

In this section, we first present the experiment results without any mitigation mechanism. We show that in most of the cases, mitigation is necessary to prevent severe disruption to the application. Then we present the results of the two mitigation solutions with the adjustments of different parameters. We also show the performance of the two solutions under different conditions.

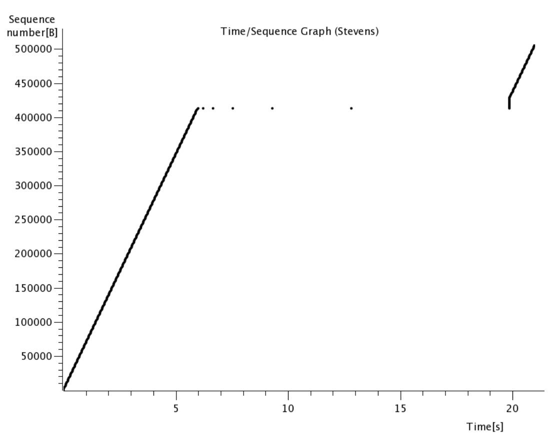
5.3.1 TCP Behavior Interferes with the Recovery of the Connection

We first study the effects of disruption to a TCP connection without any mitigation. We study whether mitigation is required for all cases or just some cases.

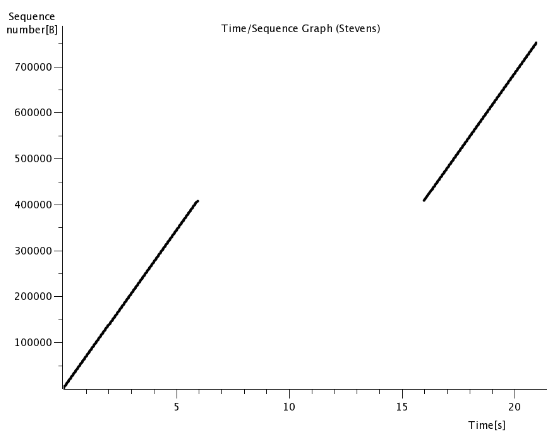
5.3.1.1 Is it necessary to mitigate the effects of network disruptions?

Figure 30 shows the TCP traces of disruption of $n = 10$ seconds without and with the end-system solution implemented. After the disruption starts, without any mitigation, the application at host A continues sending packets to host B. During the network disruption period, the packets would be lost. This packet loss triggers retransmission timeout and the exponential backoff mechanism of TCP. After the network connection for host A and B resumes, because of the exponential backoff, the connection takes 4 more seconds to start sending the first packet after the disruption time. In contrast, with mitigation, because the application stops sending packets to the socket after it receives the pause command from the controller, it prevents packet loss and the exponential backoff of TCP. The connection resumes smoothly after the 10 seconds of network disruption. From the y-axis, we see the sequence number of the TCP segments. Note that the scale of the y-axes in these two figures are different. Figure 30(b) shows at the end of the trace, the packet's sequence number is over 750,000 which is higher than in Figure 30(a), which is at around 500,000. That means when there is a disruption, the sender can deliver more packets with mitigation than without any mitigation.

The performance gap can be explained by looking at the last packet delivered before and the first packet delivered after the disruption period. After a packet loss during the disruption, that loss data segment is not retransmitted again until the



(a) Without mitigation

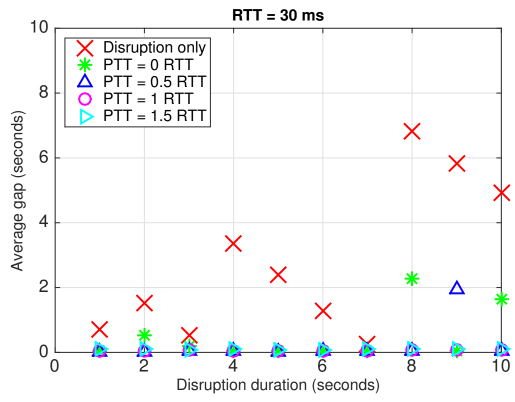


(b) With mitigation

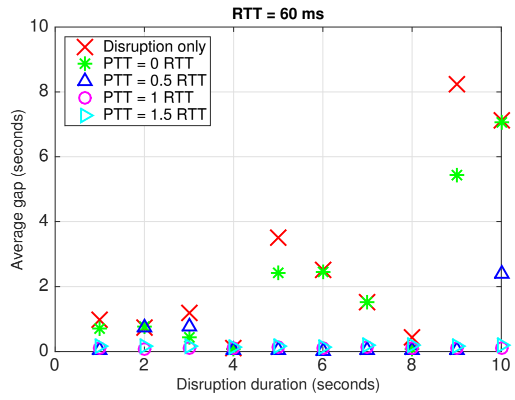
Figure 30: TCP trace of 10 seconds of disruption without the end-system (a) and with the end-system solution (b)

retransmission timer expires. If the sender has not received an acknowledgment from the receiver within the timeout time, the sender will retransmit the data segment again and restart the retransmission timer with a double of the RTO [68]. If the disruption duration ends at the time when the retransmission timer just started again, the data segment will not be retransmitted until the timer expires. As a result, the gap can be much larger than the disruption duration. With different data path RTT, for each disruption duration (1-10 seconds), Figure 31 shows the average measured gap minus the disruption duration n. The x-axes are showing the disruption time n from 0 to 10 seconds. Without any mitigation, with the 'disruption only' cases, the measured gap minus the disruption duration can be as much as 10 seconds. For example, in Figure 31(a), when the disruption duration is 4 seconds, on average, the sender waits for over 3 seconds after the connection resumed to retransmit the loss data segment. This time is larger in the $RTT = 90\ ms$ case. In Figure 31(c), when the disruption time is 10 seconds, the sender waits in average almost 10 seconds after the connection resumed to retransmit the data segment.

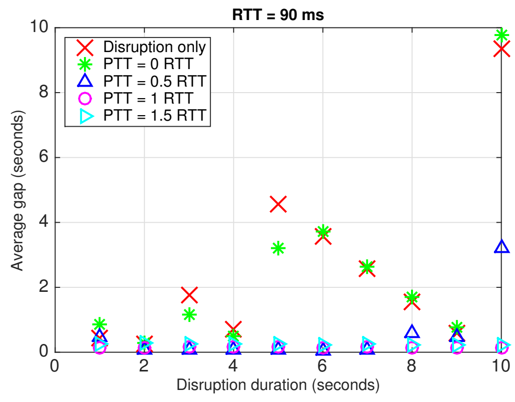
With mitigation, we expect the connection to resume immediately such that there



(a) RTT=30ms



(b) RTT=60ms



(c) RTT=90ms

Figure 31: Retransmission delay: Average measured gap between the last packet delivered before the pause and the first packet delivered after the pause minus the actual disruption duration for each disruption duration case (without any mitigation, with mitigation and different PTT).

would not be any gap between the time resume command issued and the first packet sent after the resume. However, we observe that when there is not enough time, a', waited between the end of the network disruption and the controller sending the resume command, there is packet loss. We further study the amount of time a' in Section 5.3.2.1 in detail.

5.3.1.2 When do we need mitigation?

We try to estimate the RTO of each node on the network and to determine whether we need mitigation in the network all the time. We find that under some cases of data path latency and some of the disruption duration, the gap between the last packet delivered before and the first packet delivered after the disruption duration is close to zero. For example, in Figure 31(a), the case of disruption duration of 7 seconds shows that without any mitigation, the gap between the resume time and the first packet sent after the resume time is less than 1 second, i.e., the connection takes less than 1 second to resume. This gap indicates that the sender's retransmission timeout (RTO) is triggered right after the connection resumes such that the sender can retransmit the data segment immediately. If we know the estimated RTT and the disruption duration already, we may be able to predict the RTO and the sequence of doubling the RTO. If the predicted RTO and doubling of RTO fall right after the end of the disruption period, the sender can retransmit the packet right after without waiting for the next RTO to be over and we do not need any mitigation at this perfect timing. However, there are only two to three cases in the 10 seconds disruption duration that we do not need mitigation to prevent a large gap. Also, it is still challenging to estimate the RTO of each node on the network. As a result, it is much easier to have the mitigation mechanism built in to the network system for all cases.

5.3.2 Performance of the Mitigation Solutions

After we run the experiment without any mitigation, we realize that it is very important to prevent a packet loss during the disruption period such that we can prevent TCP's exponential backoff. Then we implement the two mitigation solutions, end-system application-aware and gateway buffering designs. We first study the performance of the controller with the two mitigation solutions. Then we study how the disruption time, multiple flows, high sending rate, Poisson process packet arrival rates, and gateway buffer size affect the performance of the two mitigation solutions.

5.3.2.1 Delay between the controller and gateways and end-systems

From our experiment, we find that the time delay between the controller sending out the pause command and the VN disruption command highly affects the performance of the mitigation solutions. After the controller issues a pause command to the end-systems or gateways, it also issues the disruption command to the VN to start the disruption. We call the time between these two commands, which is a' in Figure 29, as the *Path Takedown Time* (PTT). We adjust the PTT at the controller so that the pause command to the end hosts or the gateways will be sent and arrive before the network disruption starts. If we set the PTT at the controller to zero, given that we assume that the path latencies of the controller paths (path between the controller and any nodes on the network) are the same, the commands will arrive at the VN and the end-systems or gateways at the same time. So, the VN will start the disruption and the end-systems or gateways will start the pausing at the same time. In our experiment, in Figure 31(a) we find that if the PTT is zero and the RTT of the data path is 30 ms, for most of the disruption duration, the average gap is still very small. However, if the PTT is zero and the RTT is 60 or 90 ms (Figure 31(b) and Figure 31(c)), we still experience packet loss and the exponential backoff. From this observation, we realize that the relative values of RTT of the data path and the

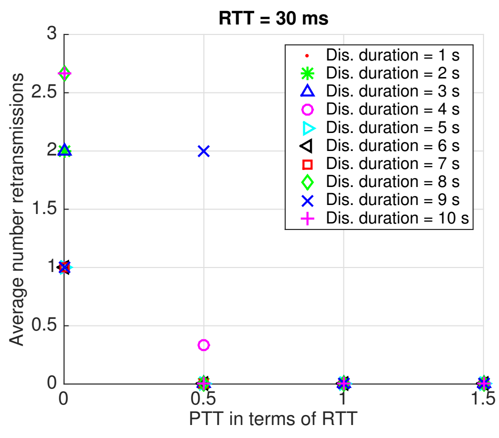
PTT can influence the performance.

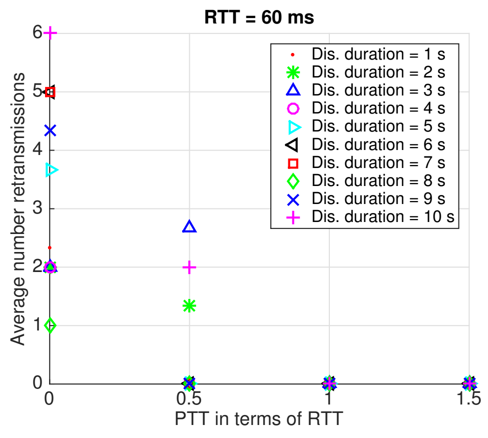
We find that when $PTT < RTT$ of the data path, the time between the end-systems or gateways receiving the pause command from the controller and the time the VN receiving the disruption command is smaller than the data path's RTT. Some of the data segments and TCP acknowledgment packets can be still in flight. At that time, even though the application or the buffering application of the gateways stop pushing data to the TCP sockets, if a TCP acknowledgment arrives at the sender right before the disruption starts, the TCP socket can still send out the data segments that are buffered in the socket buffer. Because the VN has started the disruption period, the data segment will be lost, potentially trigger retransmission timeouts.

We adjust the PTT to different values in terms of the RTT such that the VN will receive the starting disruption command after the end-systems or gateways have stopped sending packets. The packets in flight have been received by the end-systems or gateways. Figure 31 shows various PTT in terms of the RTT. For example, in Figure 31(a), PTT is set to 0, 0.5, 1, and 1.5 RTT, which are equivalent to 0 ms, 0.5×30 ms, 1×30 ms, and 1.5×30 ms, respectively. For the cases of $PTT = 0$ and 0.5 RTT, for some of the disruption durations, we can still observe larger than one seconds of gaps between the resume command sent and the first resume packet sent. Figure 32 shows the number of retransmissions with our end-system mitigation solution with different RTTs and different disruption durations. If we adjust the PTT to at least 1 RTT, the average number of retransmissions is zero in all the disruption duration cases. For the rest of the experiments, we set $PTT = 1\ RTT$ to prevent unnecessary packet loss and retransmission.

5.3.2.2 The relationship between the disruption duration and mitigation solutions performance

We next perform experiments with disruption duration from 1 to 10 seconds. Figure 33 shows the average number of received packets of different mitigation solutions with


(a) RTT=30ms


(b) RTT=60ms

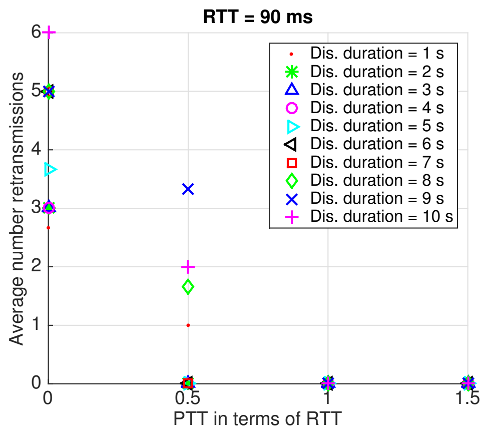

(c) RTT=90ms

Figure 32: Average number of retransmissions versus the PTT in terms of the data path RTT with distruption duration of 0 to 10 seconds. PTT is set to 0, 0.5, 1 and, 1.5 times of RTT.

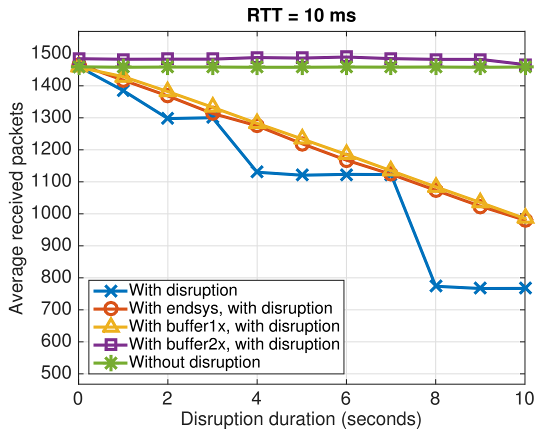
RTT of 10 ms, 30 ms, and 100 ms. The x-axes show the disruption duration of the VN from 0 to 10 seconds. The y-axes show the average received packets by the receiver of three experiments for each disruption duration.

First, in Figure 33, the case 'No pause with buffer 1x, with disruption' shows that if the controller does not pause the gateway, even we have a buffer installed at the gateway with the same draining speed as the sender, the number of received packets follows the gap latency in Figure 31. The gateway still buffers the packets that are not delivered. However, because of packet loss, the average number of packets delivered still follows the same behavior as the case when there is no mitigation. Thus, for the gateway buffering solution, we still need the controller to pause the gateways to prevent any packet loss and retransmission timeout.

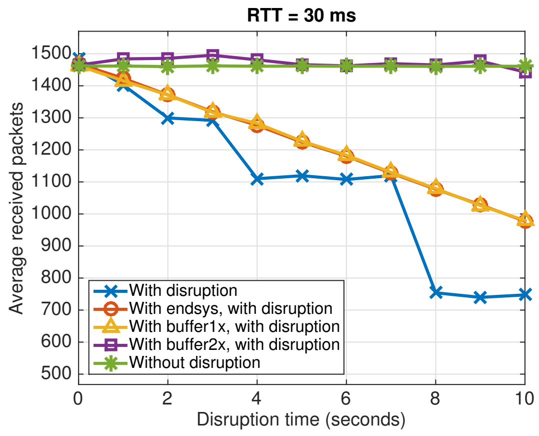
Then we look at how the two mitigation solutions perform. In Figure 33, when the gateway buffering solution is set to have the same draining rate (1x) as the sending rate of the sender, it performs the same as the end-system aware solution. Both solutions behave similarly as UDP. The number of received packets from the application level decreases as the disruption duration increases linearly. When the disruption duration is 10 seconds, which is one third of the experiment time, the average number of received packets drops from around 1500 to 1000. That means during network disruptions, both mitigation solutions effectively prevent packet loss such that the sender can resume sending immediately after receiving the resume command from the controller.

5.3.2.3 Does Split TCP design in gateway buffering enhance performance?

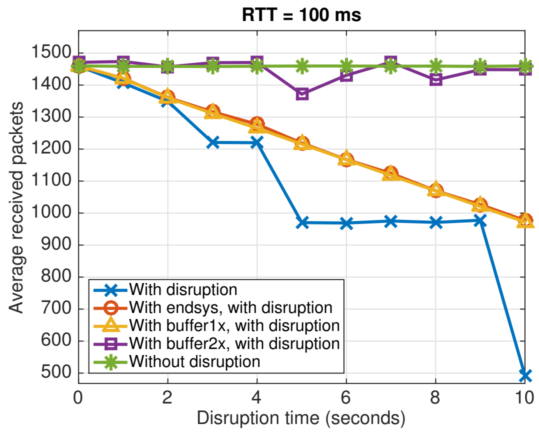
We further perform an experiment of the gateway buffering solution with 2 times and 10 times of buffer draining rate. In our experiment, the buffer at G1 has a 2 times draining rate of the sender sending rate. We do not limit the G2's buffer draining rate. Because the buffer can drain much faster and the split TCP sessions have shorter path latency compare to the end-system solution, the sender can send more packets with the TCP session between the sender and G1. Figure 33 shows that gateway buffering with 2 times buffer draining rate (with buffer2x, with disruption) does the same or better in the $RTT = 10$ ms cases than when there is no disruption at all. This observation also has been mentioned in [25].



(a) RTT=10ms



(b) RTT=30ms



(c) RTT=100ms

Figure 33: Average number of received data packets versus disruption durations with end-system and gateway buffering solutions with 1 and 2 times buffer draining rates.

5.3.2.4 *How does the buffer draining rate at the gateways affect the performance of the gateway buffering solution?*

We observe that it is difficult to drain the buffer at the gateway after a pause if the gateway is draining the buffer at the same rate as the sender. So, we adjust the buffer draining rate of the gateway to 2 times and 10 times of the sender's sending rate.

We compare the gateway buffering solution with different buffer draining rates. Figure 33 shows the average number of packets delivered with buffer draining rates of 1, 2, and 10 times of the sender's sending rate. We first compare the results with the end-system solution. In the previous section, we have mentioned when the draining rate of the gateway buffering solution is the same as the sender's sending rate, the result is similar to the end-system aware solution Now we increase the buffer draining rate of the gateway. Because the gateway can drain the buffer at a higher rate, if the time between the resume and the end of each experiment is long enough, the buffered packets can be drained from the buffer and the new packets from the sender after the resume time can be forwarded within that time. In our experiments, the disruption duration n is always less than or equal to the time between the resume and end of the experiment (b), i.e., $n \leq b$. Within b seconds the gateway has to drain the packets buffered in n seconds and forward the new packets from the sender. That means within b seconds, the gateways has to deliver $b+n$ seconds of packets. If the gateway draining rate is double of the sender's sending rate and $n \leq b$ always, the buffer will be drained eventually within this time and the sender does not notice the disruption at all.

5.3.2.5 *How does the buffer size at the gateways affect the performance of the gateway buffering solution?*

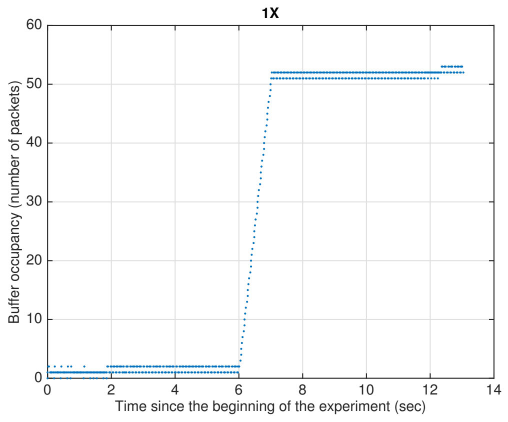
Next we turn to the questions of buffer size and buffer occupancy. In our experiment, we first assume we have unlimited buffer to store the packets during the network disruption time. Figure 34 shows the buffer occupancy over the time of the case with network disruption duration of 1 second. When the gateway drains the buffer with the same rate as the sender, the buffer can never be able to drain the entire buffer. It introduces a one second delay to all the rest of the flow. In Figure 34(b) and Figure 34(c), the buffer draining rates of the gateway are two times and ten times faster than the sender's sending rate, respectively. As a result, the buffer can be drained in a short time and the connection recovers easily.

We also observe that if the disruption time is longer, it requires a larger buffer to store all the packets sent in the disruption period. However, a larger buffer also takes a longer time and a higher draining rate to drain the buffer completely. Thus, if the VN can handle a higher gateway buffer draining rate and the disruption duration is within the time for the buffer to drain all the buffered packets, it is reasonable to buffer all the packets of the disruption duration at the gateways.

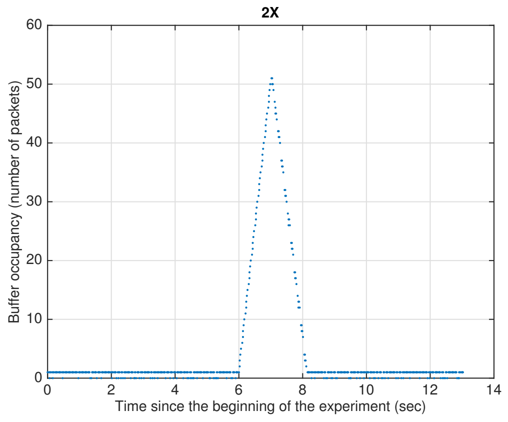
We then limit the buffer size of the gateway to 10 packets. Figure 35 shows the TCP traces of a disruption duration of 10 seconds when the buffer is limited to 10 application packets. When the buffer is full, the gateway stops reading packets from its TCP socket. The gateway's TCP socket sends 'TCP window size zero' back to the sender. The sender keeps sending TCP keep-alive with exponential backoff retransmission timer until the gateway has enough space in the application buffer and starts reading from its TCP socket. At this time, the gateway's TCP socket sends a TCP window update to the sender such that the sender can send data again. This way, the sender does not wait for the retransmission timeout to send the packets after a resume. As a result, it prevents the effects of TCP's exponential backoff. With limited buffer size, we can still mitigate the network disruption effectively.

5.3.2.6 How does the gateway buffering solution perform under multiple sessions?

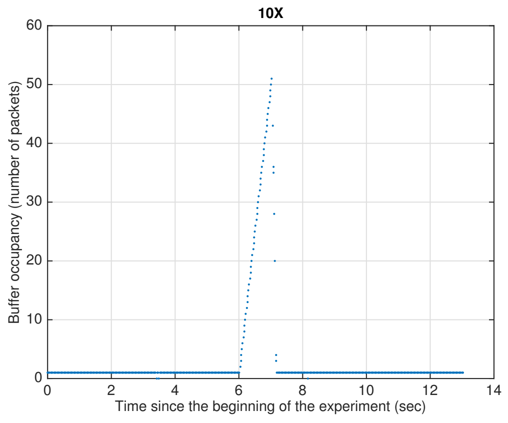
We study the effects of multiple flows to the gateway buffering solution. We expand our experiment topology to four end hosts. Figure 36 shows our new topology. We



(a) Same Rate

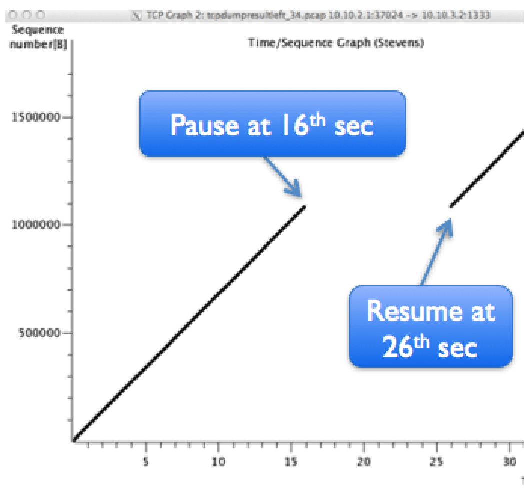


(b) 2x

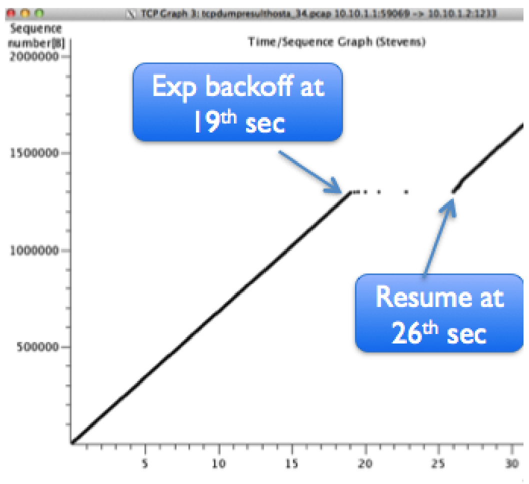


(c) 10x

Figure 34: Buffering occupancy at the first gateway when the disruption duration is 1 second with the gateway buffering solution



(a) Captured at gateway



(b) Captured at sender

Figure 35: TCP traces of disruption duration of 10 seconds with the gateway buffering solution under limited buffer size of 10 application packets. X-axes are showing the time of the experiment and y-axes are showing the sequence number.

add one more end host on each side of the VN. All four end hosts send and receive data. We have separate application sessions for each pair and each direction of data flows between A1 and B1 and between A2 and B2. In this topology, we limit the bandwidth of the links between the end-systems and the gateways to 10 Mbps and the links between the gateways and the VN to 100 Mbps. We keep our sending rate at the end hosts to 560 kbps, which is equivalent to sending 1400 bytes every 20 ms.

Figure 37(a) shows the average received packets for all four flows compare between cases when there is disruption and no disruption. The flows are running simultaneously in each experiment. All the flows behave similarly.

Then we compare the multiple session result to a single session result. Figure 37(b) shows the average received packets for the session sending data from A1 to B1. We also plot the result of the single flow from A1 to B1 when there is no other sessions with other end-systems running in the background. They both behave similarly. Thus, with multiple sessions, the solution still behave the same so far.

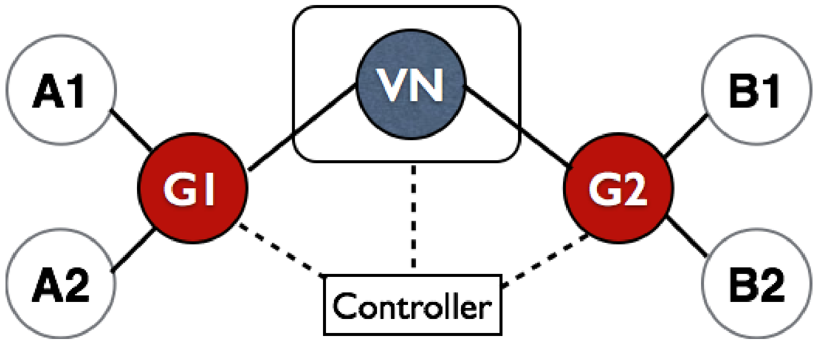


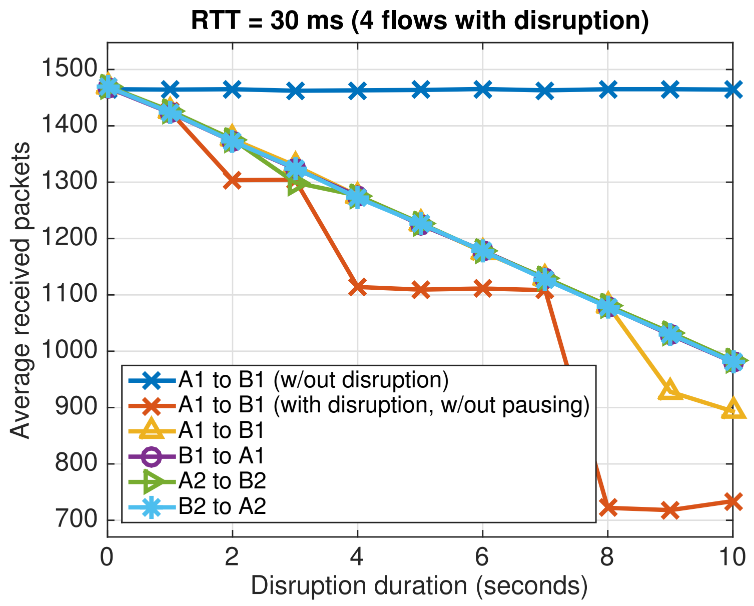
Figure 36: Experiment topology with 4 end hosts and 4 application sessions

5.3.2.7 What is the effect of high sending rate?

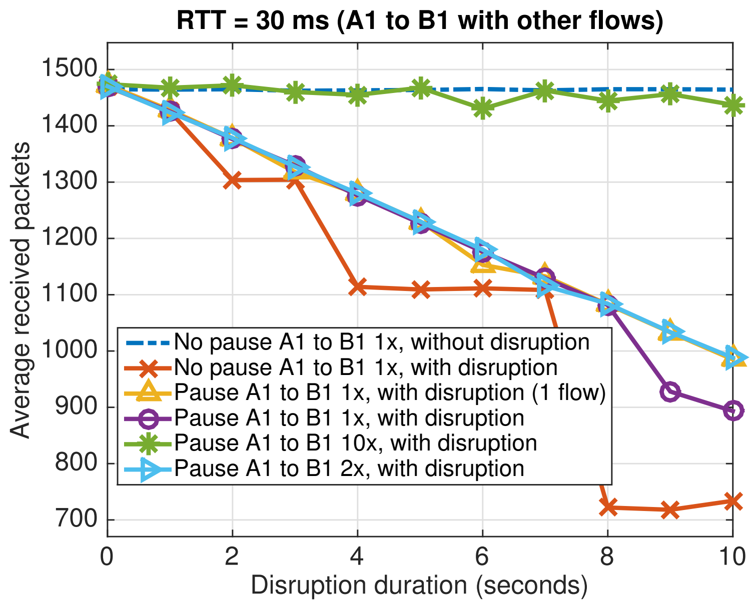
We increase the sending rate of the sender from 560 kbps to 11.2 Mbps. It is over the bandwidth limit of the links between the end hosts and the gateways, which are 10 Mbps. Notice that only the link between the gateways is shared among the four application sessions. Figure 38 shows the average number of received packets of session B1 to A1 with the sending rate of 11.2 Mbps. In this figure, the other three flows are also present together during the experiment. When there is zero disruption, the average number of received packets is roughly 18,000. When the disruption duration is 10 second, which is one third of the entire measurement period of 30 seconds, the average number of received packets drops to below 13,000. This drop is roughly one third of 18,000. It does not show any packet loss during the disruption duration. Then we compare it with the rate of 560 kbps. Roughly there are 18 times more packets are received by A1 than the 560 kbps sending rate. We conclude that with high sending rate, the gateway buffering solution still performs reasonably well.

5.3.2.8 What is the effect of the packet arrival rate following Poisson process?

After testing the two solutions with constant bitrate (CBR), we study the effect of Poisson process to the gateway buffering solution. We generate packets with an inter-arrival time of an exponential distribution. We use the same rate as the rate of 560 kbps and 11.2 Mbps. Figure 39 shows the average number of received packets



(a) Average number of received packets with 4 application sessions running simultaneously with 1x buffer draining rate at G1



(b) Average number of received packets of session A1 to B1 when the other three sessions running simultaneously and compare the average received packets with different buffer draining rates and the result of a single session of A1 to B1

Figure 37: 4 applications sessions running simultaneously with gateway buffering

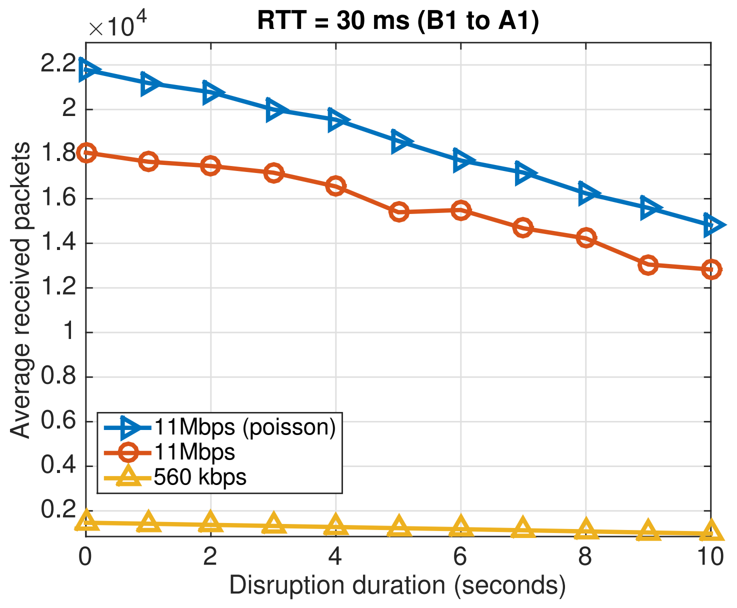


Figure 38: Average number of received packets of session B1 to A1 with sending rate of 11.2 Mbps and 560 kbps with gateway buffering.

of session B1 to A1 when the sending rate at B1 is exponentially distributed. The case of no disruption varies between 1400 and 1500. The results of the exponentially distributed sending rate and the CBR case are very similar. It is because the gateways perform the buffering with the same draining rate as the sender rate. The gateways do not drain the buffers with exponentially distributed rate. As a result, the CBR case and the Poisson process case both perform similarly.

5.4 Discussion

After implementing our mitigation system with two solutions, we further compare these two solutions and provide insights to implement these two solutions.

5.4.1 Scalability

We first look at the scalability issue of both solutions. For the end-system solution, because the controller sends commands to every end hosts, the number of commands to be sent is equal to the number of end hosts. In contrast, for the gateway buffering solution, only the gateways are notified by the controller to pause and buffer traffic.

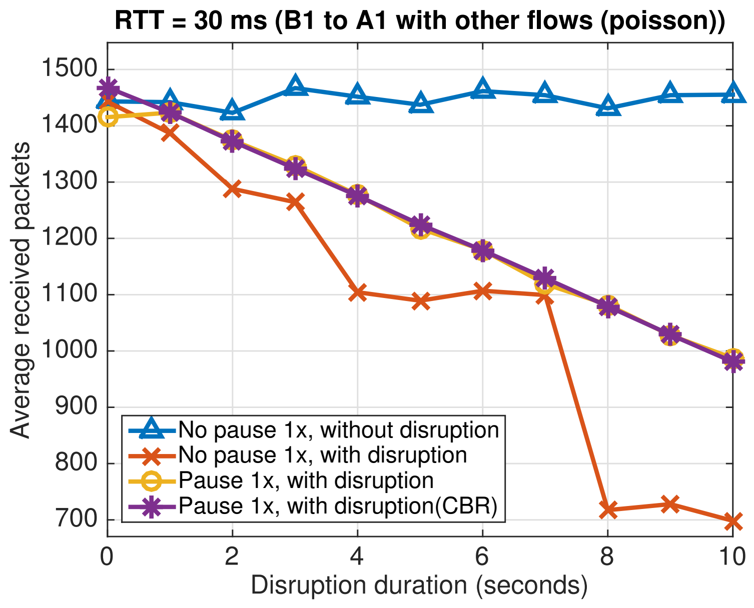


Figure 39: Average number of received packets of session B1 to A1. Comparison of the sender using exponentially distributed inter-arrival times and constant bit rate (CBR) with gateway buffering.

Because with gateway buffering, the number of nodes to be handled by a single controller is bounded by the total number of gateways, gateway buffering would scale better than the end-system solution. Based on the number of commands to be sent and the nodes to be reached by the controller, gateway buffering is preferred.

In terms of the number sessions to be maintained, with increasing number of end-systems, gateway buffering requires a lot more resources to maintain the application sessions and buffers for these sessions at the gateways. If network disruptions are not prevalent in the network, these gateways can be virtualized and set up dynamically in the VN. We further discuss the implementation of gateways in the following section.

Both solutions require changes either at the core or at the end-systems. With the end-system solution, there is no infrastructure change to the network. Only the end host applications have to support the features on the application layer. With the gateway buffering solution, we need to implement pausing and buffering at the gateways and redirection application traffic to the gateways. It is a trade-off between changes at the core and the end-systems.

5.4.2 Implementation Issues

Controller path latency The path latency between the controller and the gateways and end hosts affects the command arrival time. If the pause commands do not arrive the gateways or end hosts before the disruption command is issued, the gateways or end hosts cannot stop sending data packets before the network disruption happened. Even though the gateways or end hosts receives the pause command and stops sending packets before the disruption happens, the packets in flights can be lost. Thus, the controller should be able to estimate the over all RTT of the longest data end-to-end path and the latency between the controller and the gateways or end hosts to compute the PTT. In that case, both mitigation solutions require the estimation of the best PTT to prevent packet loss.

Implementation of gateways With gateway buffering, we have to implement gateways in the network for buffering packets. We can install these gateways permanently or virtually. The permanent gateways can be installed with the network proxies currently in the network.

The virtualized gateways can be set up and installed dynamically with some virtualized middleboxes. The VN can redirect traffic to these virtualized middleboxes to prepare the disruption. With these dynamic virtualized gateways, the application traffic can be redirected to avoid any disruption in the network. We leave the implementation of virtualized gateways and redirect traffic on VN to future work.

Buffer control with gateway buffering With gateway buffering, there are three main issues related to the buffer control. These three issues are gateway application buffer size, and the draining rate of the first and second buffers.

In our experiment, we have studied both with limited and unlimited application layer buffer sizes. They both can mitigate the disruption effectively. The only difference is whether we want to utilize the benefit of split TCP to let the sender send more packets. When we limit the buffer size, we indirectly limit the TCP socket to stop receiving packets from the sender. Thus, we can pause the sender from sending packets to the gateways and prevent the exponential backoff after the session is resumed. With unlimited buffer size, we use a lot more space at the gateways. However, we can let the senders send more packets during the disruption period. If the draining rate of the gateways is comparatively higher than the sender's sending rate and the VN has a much higher bandwidth than the end hosts' links, we can possibly increase the amount of data to be delivered through the VN.

If we limit the buffer size but do want to buffer most of the data during most of the disruption, the gateway buffer should be large enough to store the data for the data from all the sessions going through that gateway at a disruption duration. The buffer size can be calculated based on the disruption duration, the sender sending rate and the buffer draining rate.

Other than the buffer size, gateway buffer draining rate is another setting required some adjustment. For each application session, two gateways are involved. One of them is closer to the sender. And the other one is closer to the receiver. For the first gateway, after it receives a resume command from the controller, it has to drain the buffer and also forward the packets from the sender at the same time. If the buffer is draining at the same rate as the sender, the buffer is never going to be drained.

If the VN has a higher bandwidth than the links between the end-systems and the gateways, the first gateway should be able to drain the buffer with a higher rate than the sender's sending rate. The draining rate also depends on the number of application sessions the gateway is handling. We estimate the bandwidth of the VN and use that as the overall draining rate of the first gateway. The overall draining rate is the sum of the rates of all the application sessions the gateway handled. For each application session, the corresponding buffer should drain at the same rate as the other application session sharing the same gateway.

If the second gateway forwards the packets from the first gateway with the same rate as the first gateway to drain the buffer, the connection can be resumed to normal in a short period of time. However, if the links between the gateway and the receivers have a much lower bandwidth than the VN, the second gateway cannot forward the packets faster than the first gateway. As a result, the second gateway becomes the bottleneck of the application session. We should adjust these rates according to the application sending rate and the available bandwidth of the VN, and the links between the end hosts and the gateways. The problem with estimating these rates is that, if the VN is migrating or under maintenance, the available bandwidth for the VN will vary over time. We can only estimate the available bandwidth of the links between the gateways and the end hosts and use that as the second gateway draining rate.

Notification to users In the gateway buffering solution, the users do not know there is a disruption because the gateways buffer the packets. If the application requires an immediate feedback such as biding or stock exchange applications, we should have notify the users about this possible latency or design the application with another layer of protocol.

5.5 Summary

We present a system with end-systems and gateways for mitigating the effects of VN migration and other planned network disruptions. We propose two solutions for this mitigation system. In the end-system application aware solution, the applications at the end-systems are directly controlled by a controller to pause and resume their traffic. In the gateway pausing and buffering solution, the applications are not involved in the mitigation system. Instead, the gateways handle all the pausing and resume of the traffic for the end-systems. We have implemented these two solutions on GENI and shown that these two solutions can effectively mitigate the effects of planned network disruptions.

We find that as an application running in a virtual machine, although the application cannot change the TCP socket options in the host OS, the application can indirectly affect the socket. For example, when an application stops receiving packets from the TCP socket buffer, the TCP socket cannot receive any packets if the buffer is full. By doing that, the TCP socket notifies the sender with TCP window size zero. This notification automatically stops the sender from continuing to send data. In the future, the gateway buffer size and draining rate should be formulated based on the network condition before and after the network disruption. A proper method to set up the gateways using virtual middleboxes is another addition to this work.

CHAPTER VI

CONCLUDING REMARKS

VN migration provides significant flexibility in the deployment of virtual networks. However, we lack the mechanism to migrate VNs on the routing and switching layers. In this dissertation, we present challenges and solutions to migrate virtual networks. We propose algorithms to schedule VN migration, implement a VN migration method on a shared substrate, PlanetLab, and further propose an architecture for mitigating the effects of VN migration and other planned network disruptions. We first summarize the contributions of this dissertation. Then, we present the lessons we learned in this dissertation and future directions for VN migration.

6.1 Summary of Contributions

6.1.1 Algorithms for Scheduling VN Migration

We leverage prior work that considers the design of a *single virtual router* live migration mechanism [88] and propose algorithms to schedule VN migrations. The objective of this prior work is to perform virtual router migration such that the migration does not disrupt current data flows. We build on this single virtual router migration mechanism. We use this mechanism as a subroutine in the live migration of an *entire* VN on both the switching and routing layers to schedule a VN migration. Specifically, given a new VN placement within the same substrate network, we propose three algorithms to determine the best *schedule* of virtual router migrations that implement the desired VN move. We maintain the goal of not disrupting current network traffic during the migration process and consider the design of a schedule that minimizes the overhead of network migration or minimizes the time it takes to complete the migration task. This work is the first we are aware of that considers the question of scheduling a live VN migration for VNs.

6.1.2 A Controller-based Architecture for Virtual Network Migration on PlanetLab

We study the VN migration problem on PlanetLab [70], propose a VN migration framework based on PlanetLab, and implement a controller for VN migration. An effective VN migration system could mitigate negative effects on PlanetLab and similar platforms while facilitating application development. A VN migration mechanism should minimize any disruption introduced to the VN. It is also desirable that the process of VN migration be automated and relatively fast. We propose a migration mechanism on layer 3 for VN migration on PlanetLab designed to meet these objectives. We design, implement, and evaluate with live experiments, a controller called *PL-VNM* that orchestrates the VN migration to new VN placements on PlanetLab. This work leads to specific recommendations for PlanetLab that would enable VN migration scheduling, as well as recommend features for more general virtualization environments. We believe that the recommended features make long-term experiments and application deployments on shared virtualized infrastructures possible.

6.1.3 Techniques to Mitigate Performance Degradation through Transport and/or Application Layer Collaboration

VN migration leads to unexpected packet loss and bandwidth degradation for the application running on the VN. The VN migration mechanism from our PlanetLab implementation introduces short term packet loss. However, this short term packet loss can trigger a long recovery for TCP throughput. We study the effects of VN migration on the application layer and further design mitigating solutions to prevent performance degradation to the applications running on the VNs. We propose two mitigation solutions with a centralized controller to signal the end hosts or gateways about the VN migration. These end hosts pause the data flow without shrinking the TCP congestion window. The gateways buffer the data from the end hosts to prevent packet loss and retransmission timeout. These solutions effectively minimize the performance degradation to the applications. Additionally, because the effects of VN migration are similar to the effects of planned network events such as router migration and equipment upgrades to the transport layer, this work can be applied to mitigate performance degradation caused by both VN migration and planned network events.

6.2 Lessons Learned

In this dissertation, we experiment with multiple shared network experiment platforms, PlanetLab and GENI. We provide insights on designing virtual network environments and in general, designing network systems.

During VN migrations, schedule each step with timing uncertainty in mind. Because we operate in a virtual machine, we cannot control how the jobs are scheduled. The latency of the scheduling a job in the hypervisor can highly affect the performance of the VN migration. For example, when we schedule our VN migration on PlanetLab, we observe there can be a huge delay between the command received time and the command execution time. This delay affects the performance of VN migration. The same effect is observed when the mitigation controller sends commands to different elements in the network. The executions of these commands can be delayed because of the scheduling of the commands in the system hypervisor with a lower priority or the network latency. With this possible delay in mind, we should have synchronization or checkpoints in our solution.

Without access to the kernel, designing systems and running experiments with system virtualization can be challenging. We cannot verify what exactly happens in the kernel because we do not have access to the kernel environment of the machines that we are using on both PlanetLab and GENI. We run our experiments and measure the performance in the guest OS. When the machine we chose is overloaded, it is challenging to verify our experiment results without enabling the kernel debugging mode. The results can be unpredictable because of various factors such as our implementation, the virtualization itself, or the current network performance. Sometimes the resource utilization of the physical nodes is unknown. Without any access to the kernels of the machines we use, we do not know the cause of the unpredictable results.

How well a VN performs depends on the amount of root privilege one can obtain. When we start migrating VNs on PlanetLab, we use the Vsys API to clone the routing table in the kernel space. However, we find that PlanetLab is using uRPF. If the VN has the privilege to disable uRPF, we can avoid packet loss during VN migration. Similarly, to mitigate the effects of VN migration, if we have the privilege to change the TCP socket options, we can avoid the TCP exponential backoff from the socket level instead of the application layer. The performance would be improved.

Limited root privileges should be made available. Although in the previous lesson we should have granted as much privilege as possible to enable a better VN performance, it does not mean we should have the privilege to change everything in that machine. The amount of privilege should only be limited to the resources that are granted to the VN. For example, in PlanetLab, with the Vsys API, we can only change the routes of the prefix we own in the routing table, not the routes belong to other experiments. In GENI, we should be able to change the TCP socket options of only the virtual machines in the VN.

A phenomenon observed: Network researchers design future systems with current use cases When the Internet was first commercialized, it was not designed based on the applications we currently use widely such as video streaming, live video conferencing. We observe that we are often asked to *design future systems with current use cases* in the process of our work. In our work, we attempt to migrate virtual networks on PlanetLab and GENI. We find that these platforms were designed for performing experiments with existing tools or existing routing protocols and technologies, not for network migration. Migrating a single node on these platforms without information about the substrate nodes becomes a challenge in itself. We lack some privileges on these systems to facilitate migration. Furthermore, these platforms are intended for short term experiment usage rather than long term usage. Once an experiment is over, we should release the resources. It is not intended for us to migrate our network to another set of nodes and continue our experiment for a long period of time. Although we are restricted by the limitations of these platforms, we still believe these platforms are well-suited for virtualization experiments.

Collaboration with the people who design or maintain the platforms is the key to succeed. Documentation for testbed systems such as PlanetLab and GENI is sparse and hard to follow even when it exists. We therefore often had to rely on direct interaction with researchers who work on these platforms to learn about the feature of these systems. For example, we used Vsys to instruct the PlanetLab nodes to execute sudo commands such as adding new routes or adding new virtual interfaces to the node. At the time of the experiment, the Vsys API was not widely available for download. It also required special settings on the PlanetLab nodes to work with Vsys. Each PlanetLab node might have a slightly different setting that did not work with Vsys. Throughout the process of our implementation, we have contacted researchers in Europe who have been working extensively on Vsys to work with PlanetLab European nodes and researchers in Princeton who maintain the central PlanetLab nodes. Without them, as a user of PlanetLab, when our code did not work, it was difficult to tell whether it was a problem with our code or the PlanetLab node. They often worked with us to debug the issue. Without their collaboration, the problem would be much harder to solve. This lesson is also true for any large network systems that are deployed widely on the Internet. Different parties contribute to the equipment and parts of the network. The deployment of a large system only works if all parties collaborate.

6.3 Future Directions

Centralized control versus distributed architecture for VN management There has been an argument whether centralized control or distributed routing should be the ultimate network control architecture. Centralized control gives flexibility and direct control to the network while distributed systems provides robustness. Vissicchio et al. proposed an architecture that combines the benefits of the flexibility of a centralized controller and the robustness of distributed routing [87]. However, for managing VNs, we do not know whether a controller architecture or a distributed architecture can provide better support. This design choice question should be studied.

Controller Placement Our work highly relies on the timeliness of controller commands reaching the elements in the VN. The network latency between the controller and network elements should be predictable or minimized. The controller placement problem is also a problem with SDNs [49]. In our work, we face similar issue as SDNs because we rely on a controller to handle VN migration. The controller should be placed close to the network elements. The communication between the controller and the network elements should be secured and guaranteed that the communication would still be available after the VN migration. One controller is not scalable for a large VN or large substrate network. Thus, multiple controllers would be the possible solution for this problem.

Security of the Controller The controller easily becomes the target of attacks because it becomes the single point of failure in VN migration. The challenge is to secure the controller, the controller paths, and the correctness of the messages that the controller sent out during and after the VN migration. The controllers in our work have been using SSH to communicate with the rest of the network. However, SSH is not as light weight as a dedicated protocol for this type of communications. Furthermore, the controller path could be lost after the VN migration. To ensure the security and availability of the communication between the controllers and the network elements, we should design a controller path setup mechanism based on VN migration.

Virtual middleboxes as gateways, and their setup and placement In the last chapter, we propose to use gateways to buffer packets in order to mitigate the effects of VN migration. To set up these gateways, we can have them permanently or virtually installed. To permanently install them, we can install them with the proxies in the network. To virtually install them, we can set up middleboxes with virtual machines and redirect traffic to these middleboxes. If we set them up using virtual machines, we also have the possibility of migrating them [74]. The challenge is to decide the location of these middleboxes on the network and how to set them up correctly so that the traffic would go through them before and after VN migration.

Quality of experience with VN migration We migrate VNs because of various migration policy. If the migration policy is to increase the quality of experience (QoE) of the user, certain metrics should be used to quantify QoE. Currently from the network, VN, and application layers, we do not have a set of metrics to justify the QoE. Furthermore, during a VN migration, the network should still provide the same QoE to the users. Thus, a QoE model for VN migration is very useful.

System architecture to support VN migration Currently our VN architecture is set up as an overlay network or by connecting virtual machines installed with router daemons. The virtualization facilities we use assume that the virtual network would not be migrated. We can migrate the router daemons or the control plane of each virtual router but we cannot migrate the mapping of the virtual and physical interfaces and traffic easily without the collaboration of the substrate layer. If the substrate nodes open some APIs for migration, such as prioritize the migration commands before other commands, these features can be added to the next generation system architecture to support VN migration.

REFERENCES

[1] "at - execute commands at a later time." http://pubs.opengroup.org/onlinepubs/9699919799/utilities/at.html. URL retrieved November 2013.

[2] "The click modular router project." http://www.read.cs.ucla.edu/click/.

[3] "FlowVisor." http://www.openflowswitch.org/wk/index.php/FlowVisor.

[4] "Maximal independent set algorithm implementation." https://networkx.lanl.gov/trac/ticket/282. URL retrieved Apr 2012.

[5] "Networkx: A python package." http://networkx.lanl.gov/. URL retrieved Apr 2012.

[6] "Quartz: an open source job scheduling service." http://http://quartz-scheduler.org/. URL retrieved November 2013.

[7] "SQLite." http://www.sqlite.org/. URL retrieved November 2013.

[8] "Linux-vserver." http://linux-vserver.org/, 2003.

[9] "VMware: Virtualization, Virtual Machine and Virtual Server Consolidation." http://www.vmware.com, 2009.

[10] ""lxc—linux containers"." https://linuxcontainers.org/, 2013.

[11] Agrawal, M., Bailey, S. R., Greenberg, A., Pastor, J., Sebos, P., Seshan, S., Merwe, K. V. D., and Yates, J., "Routerfarm: Towards a dynamic, manageable network edge," in *In Proceedings of ACM SIGCOMM Workshop on Internet Network Management (INM*, 2006.

[12] Al-Shaer, E., "Toward network configuration randomization for moving target defense," in *Moving Target Defense*, vol. 54 of *Advances in Information Security*, pp. 153–159, Springer New York, 2011.

[13] Andersen, D. G., "Theoretical approaches to node assignment," 2002.

[14] Andersen, D. G., Balakrishnan, H., Kaashoek, M. F., and Morris, R., "Resilient Overlay Networks," in *Proc. 18th ACM SOSP*, (Canada), pp. 131–145, Oct. 2001.

[15] Baker, F. and Savola, P., *Ingress Filtering for Multihomed Networks*. Internet Engineering Task Force, 2004. RFC 3704.

[16] Bakre, A. and Badrinath, B. R., "I-TCP: Indirect TCP for Mobile Hosts," in *Proc. 15th Intl. Conf on Distributed Computing Systems*, (Vancouver, BC, Canada), May 1995.

[17] Balakrishnan, H., Padmanabhan, V. N., and Katz, R., "The Effects of Asymmetry on TCP Performance," in *Proc. ACM MOBICOM*, (Budapest, Hungary), Sept. 1997.

[18] Banerjee, S., Bhattacharjee, B., and Kommareddy, C., "Scalable application layer multicast," in *Proc. ACM SIGCOMM*, (Pittsburgh, PA), 2002.

[19] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A., "Xen and the Art of Virtualization," in *Proc. 19th ACM Symposium on Operating Systems Principles (SOSP)*, (Lake George, NY), Oct. 2003.

[20] Bavier, A., Huang, M., and Peterson, L., "An overlay data plane for planetlab," in *Telecommunications, 2005. advanced industrial conference on telecommunications/service assurance with partial and intermittent resources conference/e-learning on telecommunications workshop. aict/sapir/elete 2005. proceedings*, pp. 8–14, 2005.

[21] Bavier, A., Feamster, N., Huang, M., Peterson, L., and Rexford, J., "In VINI Veritas: Realistic and Controlled Network Experimentation," in *Proc. ACM SIGCOMM*, (Pisa, Italy), Aug. 2006.

[22] Bhatia, S., Di Stasi, G., Haddow, T., Bavier, A., Muir, S., and Peterson, L., "Vsys: A programmable sudo," in *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'11, (Berkeley, CA, USA), pp. 22–22, USENIX Association, 2011.

[23] Bhatia, S., Motiwala, M., Muhlbauer, W., Valancius, V., Bavior, A., Feamster, N., Peterson, L., and Rexford, J., "Hosting Virtual Networks on Commodity Hardware," Tech. Rep. GT-CS-07-10, Georgia Institute of Technology, Atlanta, GA, Oct. 2007.

[24] Bohdanowicz, F., Jakobs, M., and Steigner, C., "Statistical convergence analysis of routing algorithms," in *Networks (ICN), 2010 Ninth International Conference on*, pp. 365–370, April 2010.

[25] Border, J., Kojo, M., Griner, J., Montenegro, G., and Shelby, Z., *RFC 3135: Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations.* Internet Engineering Task Force, June 2001.

[26] Botero, J., Hesselbach, X., Duelli, M., Schlosser, D., Fischer, A., and De Meer, H., "Energy efficient virtual network embedding," *Communications Letters, IEEE*, vol. 16, pp. 756–759, May 2012.

[27] Brown, K. and Singh, S., "M-TCP: TCP for Mobile Cellular Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 27, pp. 19–43, Oct. 1997.

[28] Cai, Z., Liu, F., Xiao, N., Liu, Q., and Wang, Z., "Virtual network embedding for evolving networks," in *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pp. 1–5, Dec 2010.

[29] Calarco, G., "High performance click router." `https://amsterdam.lcs.mit.edu/pipermail/click/2004-November/003364.html`, Nov. 2004. Message to the click-users mailing list.

[30] Castro, M., Druschel, P., Ganesh, A., Rowstron, A., and Wallach, D., "Secure routing for structured peer-to-peer overlay networks," in *Proc. 5th USENIX OSDI*, (Boston, MA), Dec. 2002.

[31] Chang, R.-S. and Wu, C.-M., "Green virtual networks for cloud computing," in *Communications and Networking in China (CHINACOM), 2010 5th International ICST Conference on*, pp. 1–7, Aug 2010.

[32] Chowdhury, M., Rahman, M., and Boutaba, R., "Vineyard: Virtual network embedding algorithms with coordinated node and link mapping," *Networking, IEEE/ACM Transactions on*, vol. 20, pp. 206–219, Feb 2012.

[33] Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I., and Warfield, A., "Live Migration of Virtual Machines," in *Proc. 2nd USENIX NSDI*, (Boston, MA), May 2005.

[34] Demirci, M. and Ammar, M., "Fair allocation of substrate resources among multiple overlay networks," in *Proc. of MASCOTS*, vol. 0, (Los Alamitos, CA, USA), pp. 121–130, IEEE Computer Society, 2010.

[35] Egi, N., Greenhalgh, A., Handley, M., Hoerdt, M., Huici, F., and Mathy, L., "Towards high performance virtual routers on commodity hardware," in *Proceedings of the 2008 ACM CoNEXT Conference*, CoNEXT '08, (New York, NY, USA), pp. 20:1–20:12, ACM, 2008.

[36] Eramo, V., Miucci, E., Cianfrani, A., and Listanti, M., "Evaluation of power saving in an mpls/ip network hosting a virtual router layer of a single service provider," in *Proc. ICTC 2013*, (Jeju Island, South Korea), Oct. 2013.

[37] Fajjari, I., Aitsaadi, N., Pujolle, G., and Zimmermann, H., "Vnr algorithm: A greedy approach for virtual networks reconfigurations," in *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, pp. 1–6, 2011.

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

[39] Farooq Butt, N., Chowdhury, M., and Boutaba, R., "Topology-awareness and reoptimization mechanism for virtual network embedding," in *Proc. IFIP*, NETWORKING'10, (Berlin, Heidelberg), pp. 27–39, Springer-Verlag, 2010.

[40] Feamster, N., Gao, L., and Rexford, J., "How to lease the Internet in your spare time," *ACM Computer Communications Review*, vol. 37, no. 1, pp. 61–64, 2007.

[41] Fischer, A., Botero, J., Till Beck, M., de Meer, H., and Hesselbach, X., "Virtual network embedding: A survey," *Communications Surveys Tutorials, IEEE*, vol. 15, pp. 1888–1906, Fourth 2013.

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

[43] Ghobadi, M., Yeganeh, S. H., and Ganjali, Y., "Rethinking end-to-end congestion control in software-defined networks," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, HotNets-XI, (New York, NY, USA), pp. 61–66, ACM, 2012.

[44] Ghorbani, S., Schlesinger, C., Monaco, M., Keller, E., Caesar, M., Rexford, J., and Walker, D., "Transparent, live migration of a software-defined network," in *Proceedings of the ACM Symposium on Cloud Computing*, SOCC '14, (New York, NY, USA), pp. 3:1–3:14, ACM, 2014.

[45] "GT-ITM." `http://www.cc.gatech.edu/projects/gtitm/`.

[46] Guo, T., Wang, N., Moessner, K., and Tafazolli, R., "Shared backup network provision for virtual network embedding," in *Communications (ICC), 2011 IEEE International Conference on*, pp. 1–5, June 2011.

[47] Ha, S., Rhee, I., and Xu, L., "Cubic: A new tcp-friendly high-speed tcp variant," *SIGOPS Oper. Syst. Rev.*, vol. 42, pp. 64–74, July 2008.

[48] Handley, M., Hudson, O., and Kohler, E., "XORP: An open platform for network research," in *Proc. 1st ACM Workshop on Hot Topics in Networks (Hotnets-I)*, (Princeton, NJ), Oct. 2002.

[49] 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.

[50] Herker, S., Khan, A., and An, X., "Survey on survivable virtual network embedding problem and solutions," in *ICNS 2013, The Ninth International Conference on Networking and Services*, (Lisbon, Portugal), pp. 99–104, 2013.

[51] (ITU), I. T. U., "Key ICT indicators for developed and developing countries and the world (totals and penetration rates)." http://www.itu.int/en/ITU-D/Statistics, 2013.

[52] Keller, E., Arora, D., Botero, D. P., and Rexford, J., "Live migration of an entire network (and its hosts)," *Princeton University Computer Science Technical Report*, vol. TR-926-12, June 2012.

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

[54] 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.

[55] Keromytis, A. D., Misra, V., and Rubenstein, D., "SOS: Secure overlay services," in *Proc. ACM SIGCOMM*, (Pittsburgh, PA), pp. 61–72, 2002.

[56] Liao, Y., Yin, D., and Gao, L., "Pdp: parallelizing data plane in virtual network substrate," in *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, pp. 9–18, ACM, 2009.

[57] Lo, S., Ammar, M., and Zegura, E., "Design and Analysis of Schedules for Virtual Network Migration," in *Proceedings of the 11th International IFIP TC 6 Conference on Networking*, IFIP Networking 2013, 2013.

[58] Lo, S., Ammar, M., Zegura, E., and Fayed, M., "Virtual Network Migration on Real Infrastructure: A PlanetLab Case Study," in *Proceedings of the 12th International IFIP TC 6 Conference on Networking*, IFIP Networking 2014, 2014.

[59] Lu, J. and Turner, J., "Efficient mapping of virtual networks onto a shared substrate," *Technical Report WUCSE2006*, vol. 35, no. 2006-35, pp. 1–11, 2006.

[60] Martins, J., Ahmed, M., Raiciu, C., Olteanu, V., Honda, M., Bifulco, R., and Huici, F., "Clickos and the art of network function virtualization," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, (Seattle, WA), pp. 459–473, USENIX Association, Apr. 2014.

[61] Mattos, D. M. F., Ferraz, L. H. G., Costa, L. H. M. K., and Duarte, O. C. M. B., "Evaluating virtual router performance for a pluralist future internet," in *Proceedings of the 3rd International Conference on Information and Communication Systems*, ICICS '12, (New York, NY, USA), pp. 4:1–4:7, ACM, 2012.

[62] "NEPI: Network Experiment Programming Interface." `http://nepi.inria.fr/`, 2010.

[63] "NTP: The Network Time Protocol." `http://www.ntp.org`.

[64] Nygren, E., Sitaraman, R. K., and Sun, J., "The akamai network: A platform for high-performance internet applications," *SIGOPS Oper. Syst. Rev.*, vol. 44, pp. 2–19, Aug. 2010.

[65] "OpenFlow Switch Consortium." http://www.openflowswitch.org/.

[66] "OpenVZ: Server Virtualization Open Source Project." http://www.openvz.org.

[67] Oppenheimer, D., Chun, B., Patterson, D., Snoeren, A. C., and Vahdat, A., "Service placement in a shared wide-area platform," in *Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference*, ATEC '06, (Berkeley, CA, USA), pp. 26–26, USENIX Association, 2006.

[68] Paxson, V., Allman, M., Chu, J., and Sargent, M., *"Computing TCP's Retransmission Timer"*. Internet Engineering Task Force, June 2011. RFC 6298.

[69] Pei, D., Wang, L., Massey, D., Wu, S., and Zhang, L., "A study of packet delivery performance during routing convergence," in *Dependable Systems and Networks, 2003. Proceedings. 2003 International Conference on*, pp. 183–192, June 2003.

[70] "PlanetLab." http://www.planet-lab.org/, 2010.

[71] Postel, J. B., *Transmission Control Protocol*. Internet Engineering Task Force, Sept. 1981. RFC 793.

[72] "Quagga software routing suite." http://www.quagga.net/.

[73] Rahman, M., Aib, I., and Boutaba, R., "Survivable virtual network embedding," in *NETWORKING 2010*, vol. 6091 of *Lecture Notes in Computer Science*, pp. 40–52, Springer Berlin Heidelberg, 2010.

[74] Rajagopalan, S., Williams, D., Jamjoom, H., and Warfield, A., "Split/merge: System support for elastic execution in virtual middleboxes," in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, nsdi'13, (Berkeley, CA, USA), pp. 227–240, USENIX Association, 2013.

[75] Ranadive, U. and Medhi, D., "Some observations on the effect of route fluctuation and network link failure on tcp," in *Computer Communications and Networks, 2001. Proceedings. Tenth International Conference on*, pp. 460–467, 2001.

[76] Roskind, J., "QUIC: Design Document and Specification Rationale," 2012. Available from https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/edit.

[77] Sarolahti, P., Kojo, M., and Raatikainen, K., "F-RTO: An Enhanced Recovery Algorithm for TCP Retransmission Timeouts," *SIGCOMM Comput. Commun. Rev.*, vol. 33, pp. 51–63, Apr. 2003.

[78] Schaffrath, G., Werle, C., Papadimitriou, P., Feldmann, A., Bless, R., Greenhalgh, A., Wundsam, A., Kind, M., Maennel, O., and Mathy, L., "Network virtualization architecture: Proposal and initial prototype," in *Proceedings of the 1st ACM Workshop on Virtualized Infrastructure Systems and Architectures*, VISA '09, (New York, NY, USA), pp. 63–72, ACM, 2009.

[79] Seetharaman, S. and Ammar, M., "On the interaction between dynamic routing in native and overlay layers," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pp. 1–12, April 2006.

[80] Stockhammer, T., "Dynamic adaptive streaming over http–: standards and design principles," in *Proceedings of the second annual ACM conference on Multimedia systems*, pp. 133–144, ACM, 2011.

[81] Subramanian, L., Stoica, I., Balakrishnan, H., and Katz, R. H., "Overqos: offering internet qos using overlays," *SIGCOMM Comput. Commun. Rev.*, vol. 33, pp. 11–16, January 2003.

[82] Sun, G., Yu, H., Anand, V., and Li, L., "A cost efficient framework and algorithm for embedding dynamic virtual network requests," *Future Gener. Comput. Syst.*, vol. 29, pp. 1265–1277, July 2013.

[83] Sun, W., Mao, Z., and Shin, K., "Differentiated bgp update processing for improved routing convergence," in *Network Protocols, 2006. ICNP '06. Proceedings of the 2006 14th IEEE International Conference on*, pp. 280–289, Nov 2006.

[84] Trinh, T., Esaki, H., and Aswakul, C., "Quality of service using careful overbooking for optimal virtual network resource allocation," in *Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 2011 8th International Conference on*, pp. 296–299, May 2011.

[85] Turner, J. and Taylor, D., "Diversifying the internet," in *In Proceedings of IEEE GLOBECOM*, pp. 755–760, 2005.

[86] Varadhan, K., Estrin, D., and Floyd, S., "Impact of Network Dynamics on End-to-End Protocols: Case Studies in TCP and Reliable Multicast," in *Technical Report USC-CS-TR 98-672, University of Southern California*, 1998.

[87] Vissicchio, S., Tilmans, O., Vanbever, L., and Rexford, J., "Central control over distributed routing," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, (New York, NY, USA), pp. 43–56, ACM, 2015.

[88] 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 *Proc. ACM SIGCOMM*, (Seattle, WA), 2008.

[89] Yu, H., Anand, V., Qiao, C., and Di, H., "Migration based protection for virtual infrastructure survivability for link failure," in *Optical Fiber Communication Conference and Exposition (OFC/NFOEC), 2011 and the National Fiber Optic Engineers Conference*, pp. 1–3, March 2011.

[90] Yu, M., Yi, Y., Rexford, J., and Chiang, M., "Rethinking virtual network embedding: Substrate support for path splitting and migration," *ACM SIGCOMM Computer Communications Review*, Apr. 2008.

[91] Zhu, Y. and Ammar, M., "Algorithms for Assigning Substrate Network Resources to Virtual Network Components," in *Proc. IEEE INFOCOM*, (Barcelona, Spain), Mar. 2006.