Towards More Manageable and Secure Enterprise and Data-Center Networks

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Cheng Jin

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Prof. Zhi-Li Zhang

March, 2018

Acknowledgements

Back to the year 2008, I started this journey into computer science. Along the way, many people offered me their advice, trust and help. I would not be here without them.

I learnt the most from my advisor, Zhi-Li. He devoted a lot (if not the most) of his time guiding me through my junior years—by teaching me how to think systematically and critically, how to approach and solve research problems, and how to articulate my work in the form of writing and presentation. He is always open and supportive for the projects I want to work on, but also never lets me slip away from the right path. The most beneficial "skill" I learnt from Zhi-Li is to keep learning. Zhi-Li himself loves learning (*e.g.*, reading mathematics books just for fun) and sharing with us what he knows about (various topics such as Geography, Humanities, History and Linguistics). At the same time, he would be very happy if we knock his door, "Hey Zhi-Li, you may want to check this out." I was impacted a lot by such passion for learning.

I had the privilege of working and interacting with many other great researchers and engineers. Abhinav Srivastava and Yu Jin were my first mentors, who showed me the good sides of working in research labs. I published my first paper with them. Cristian Lumezanu and Qiang Xu, the boldest mentors that I have ever worked with, had me twice as their intern. Cristian has been a wonderful mentor and a greater friend—always encouraging me to step out of my comfort zone and offering me both hands when I need one finger. Qiang is kind—he accepted an Ice Bucket Challenge from me with no hesitation. Mario Sanchez and Sujata Banerjee showed me how important research innovation is to industry. Yanping Li and Eric Sung along with the entire network platform team showed me that software engineering can be a lot of fun. Yanping, in particular, trained me to be a full-stack developer and patiently listened to me rambling on about software-defined networking.

For the last five and half years, I enjoyed working, learning, and having fun with the networking lab members, in particular: Yu Jin, Nan Jiang, Hesham Mekky, Eman Ramadan, Arvind Narayanan, Saurabh Verma, Pariya Babaie, Yang Zhang, Braulio Dumba, Golshan Golnari, Taihui Li, Feng Tian, Xinyue Hu and Zhenhua Li. I received a lot help from the university operators: Irene Jacobson, Samantha Thomas Grumdahl, and Scott Franzitta. I would also like to thank Abhishek Chandra, Andrew Odlyzko, Kangjie Lu, Tian He and George Karypis to serve as my dissertation/thesis proposal/WPE committee members and provide feedbacks on my research.

My research was supported by various sources of funds: NSF grants CNS-1117536, CNS-1411636, CNS-1618339, CNS-1617729, DTRA grant HDTRA1-14-1-0040, Doctoral Dissertation Fellowship and Quality Metrics Fellowship. I presented my work in conferences with the financial support from SOSR'17 student travel grant, ICNP'14 student travel grant, Council of Graduate Students Travel Grant Award, and our department conference travel award. I would like to thank these funds.

I was lucky that I met many great people during my internships and school life, and eventually became friends with them: Qingyun Liu, Xi He, Ning Ding, Chen Chen, Na Zhang, Yongjie Cai, Xing Xu, Lucy Ulanova, Rajarajan Sivaraj, Hyun-wook Baek, Markus Kusano, Sicong Zhang, Sirui Xu, Qian Du, Shan Yang, Liuyi Hu, Wenchen Wang, Jiade Li, An Wang, Xi Chen, Xiaoxiao Jiang, Sara Morsy, Ziqi Fan, Shuai Wang, Feng Liu, Sheng Chen, Shuai Li, Fenggang Wu, Shenye Hu, Lili Xing, and Christopher Ruth. Thank you for those good time, help and encouragement.

I would like to express my special gratitude to Mengyao Chen and Shang Zhang. For more than fifteen years, they have been there cheering me up when I am down and calming me down when I get carried way. They are the one whom I would not avoid disturbing (even annoying) when I want to. Ruonan Hao, Yanan Zhu and Yanning Shen told me I could do it even when I did not believe so. They laugh with me as well as laughing at me. Jiajun Wang helped me being a better me. I am lucky to have you all in this journey, and thank you for making it beautiful.

In the end (above all), my thanks go to my family. My cousin, Pingxi, transferred my worries and stress to courage and motivation. My parents teach me (chat with me) everyday the meaning of love and help me constantly grow as a better person. I dedicate this dissertation to them.

Dedication

To my parents.

Abstract

Past decades have seen ever more devices connected to the Internet and new networked services created. Demands for networks—whether campus or enterprise networks that support most of our daily work activities or data center networks that power today's cloud services such as web, social media, music or video streaming services—have seen rapid growth. *Managing* and *securing* these networks with growing size and complexity have become a daunting task, as today's networks are primarily "manually" managed by network operators. This task is further compounded by lack of effective tools for network configurations and monitoring systems to provide *visibility* as to what is going on inside a network. This thesis studies existing network management approaches and identifies their limitations. We develop new network management frameworks—in particular, leveraging emerging networking technologies—to assist network operators and users in better managing and securing networks. We specifically focus on three key management tasks: diagnosing security policy misconfigurations, enhancing routing flexibility, and gaining on-demand flow visibility for better network control.

First, we study security group (*i.e.*, the primary means for cloud customers to configure security policies to protect their virtual machine instances from attacks) configurations and usage by customers in a public cloud platform based on real-world datasets. Motivated by the results and insights obtained from this measurement study, we develop a cloud security group analysis system which helps cloud customers diagnose potential misconfigurations and provides suggestions to refine security group configurations.

Second, we propose a novel framework for incremental and graceful transition from legacy networks to Software-Defined Networking (SDN) networks in stages by gradually replacing legacy devices with SDN-enabled devices as needed and as budgets allow. Hence, network operators can gracefully experiment with SDN networks to gain experience and build confidence while eliminating or minimizing service disruption. More importantly, operators can enjoy the benefits as fully deployed SDN networks. We develop a novel unified network management controller that exerts SDN-like, fine-grained routing control over both SDN-enabled and legacy switches in hybrid networks.

Third, with the goal of obtaining on-demand visibility as to monitor "who is talking to whom", we propose clairvoyant networks to provide visibility for any network flow at any time with low cost. Clairvoyant networks are partially programmable—they require as few as one SDN switch—and rely on a specialized network controller that controls paths through both the SDN and legacy networks. Our proposed clairvoyant controller allows operators to define *what to see*, *where to see*, and *how to see*; then enables/disables the specified flows' visibility in a task scheduler, within milliseconds.

In summary, this thesis studies the management of enterprise and data center networks. Our developed systems are capable of: i) helping operators and users understand and diagnose security policy configurations; ii) providing unified routing control to enable incremental and graceful transition from legacy networks to SDN networks; and iii) gaining on-demand network visibility for better network control.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

With ever more devices connected to the Internet and new services created, demands for networks—whether campus or enterprise networks that support most of our daily work activities or data center networks that power today's cloud services such as web, email, social media, music or video streaming services—have seen rapid growth. It is reported that Google's current data center has more than 100 times the capacity of its first generation of data center [1]. Undoubtedly, network management complexity is also dramatically increasing [2]. According to a Avaya survey [3], 94% of European businesses are negatively affected by the complexities of their networks. Managing these networks typically needs a huge group of operators to perform daily management tasks such as registering new devices, configuring routing policies, setting up firewall rules, and maintaining efficient network utilization as well as reliable network availability. It is reported in a recent survey [4] that 69% of networking professionals rely on manual processes, and 97% of networking professionals experienced network outages as a direct result of human error.

With the goal of enhancing the network management in enterprises and data-centers, this thesis designs and develops new network management systems that enhance access control, routing, visibility, and controllability in enterprise and data center networks. The key challenges are the large number of hosts, switches, and applications in these networks and the need for dynamic policies, flexible routing paths, and real-time visibility. To address these challenges, we propose three key ideas: i) designing a configuration diagnosis system to help cloud tenants visualize and refine security policy settings; ii)

providing flexible and unified path control in enterprise networks by leveraging emerging Software-Defined Networking (SDN) paradigm through incremental and strategical deployment of programmable devices; iii) gaining on-demand network visibility for better network control.

1.1 Thesis Statement

The central thesis of this dissertation is as follows:

Today's network management, relying on extensive manual processes and low-level configurations, introduces high complexity and little manageability.

This thesis develops new tools and systems—in particular, leveraging emerging networking technologies—to assist network operators and users in better managing and securing networks. We specifically focus on three key management tasks: diagnosing security policy misconfigurations, enhancing routing flexibility, and gaining on-demand network visibility for better network control.

1.2 Outline and Contributions

This dissertation studies network management in security policy configuration, routing, and monitoring separately. The outline of this dissertation, along with the primary contributions of this dissertation are as follows:

Understanding Security Group Usage in a Public IaaS Cloud (Chapter 3). In this chapter, we investigate and understand how cloud tenants configure security groups and assist them in designing better security groups. We first conduct a measurement-oriented analysis of security group configuration and usage by tenants in a public IaaS cloud based on real-world datasets. The goal is to understand what are the usage patterns ("good" and "bad" practices) in how cloud tenants configure their security groups. Motivated by the results and insights obtained from this measurement study, we propose and develop a cloud security group analysis system called *Socrates*, which employs visual analytics to assist cloud tenants in understanding the static and dynamic access relations among VM instances. *Socrates* also helps diagnose potential misconfigurations and provides suggestions to refine security group configurations based

on observed traffic traversing tenants' VMs. By applying *Socrates* to all existing tenants hosted on the public IaaS cloud, Our results reveal that more than 80% tenants do not have security groups configured properly, which can lead to security vulnerabilities. To the best of our knowledge, our work is the first to analyze cloud security group usage based on real-world datasets, and to develop a system to help cloud tenants understand, diagnose and better refine their security group configurations.

Unified Fine-Grained Path Control in Legacy and SDN Hybrid Networks (Chapter 4). In this chapter, we argue that it is possible to achieve most of the benefits of a fully deployed SDN *at a fraction of the cost* by strategically replacing only few legacy switches with—or introducing a few—new SDN-enabled switches in a legacy network, thus creating a *hybrid* network. Hence, network operators can gracefully experiment with SDN networks to gain experience and build confidence while eliminating or minimizing service disruption. More importantly, operators can enjoy much of the benefits as fully deployed SDN networks. We design and build Magneto, a *unified* network controller that exerts SDN-like, fine-grained path control over both SDN-enabled and legacy switches in hybrid networks. Magneto i) introduces *magnet* MAC addresses and dynamically updates IP-to-magnet MAC mappings at hosts via gratuitous ARP messages for visibility and routing control; and ii) uses the ability of SDN switches to send "custom" packets into the data plane to manipulate legacy switches into updating forwarding entries with magnet MAC addresses for enhanced routing flexibility. Our evaluation on a lab testbed and through extensive simulations on large enterprise network topologies show that Magneto is able to achieve full control over routing when only 20% of network switches are programmable, with negligible computation and latency overhead.

Gaining Fine-Grained Network Visibility for On-Demand Monitoring and Better Policy Enforcement (Chapter 5). In this chapter, we are exploring beyond the unified fine-grained path control. Our goal is to obtain fine-grained network visibility as to monitor "who is talking to whom", "how much traffic is being sent to a destination, say Google". We propose *clairvoyant networks* to provide visibility for any flow at any time and with low cost. Clairvoyant networks are partially programmable—they require as few as one SDN switch—and rely on a specialized network controller that controls paths through both the SDN and legacy networks. The clairvoyant controller allows

operators to define *what to see*, *where to see*, and *how to see*; then enables/disables the specified flows' visibility in a task scheduler, within milliseconds. Our evaluation on a lab testbed and through extensive simulations on large enterprise network topologies show that, even with a single SDN-enabled switch, operators can make *any* flow visible for monitoring within milliseconds, albeit at 38% average increase in path length. With as many as 2% strategically chosen legacy switches replaced with SDN switches, clairvoyant networks achieve on-demand flow visibility with negligible overhead.

This thesis studies and designs management systems for enterprise and data center networks. Our proposed systems are capable of: i) helping operators and users understand and refine security policy configurations; ii) enhancing routing flexibility so as to increase network utilization and efficiency; and iii) gaining network visibility to for better policy control and fine-grained network monitoring.

The remainder of this dissertation introduces background and motivation (Chapter 2); presents the security group usage in a public IaaS cloud and our cloud security group analysis system (Chapter 3); presents our designed *unified* network controller that exerts SDN-like, fine-grained path control in hybrid SDN networks (Chapter 4); presents clairvoyant networks to provide visibility for any network flow at any time and with low cost (Chapter 5); discusses future directions and finally concludes (Chapter 6).

1.3 Bibliographic Notes

Part of the contents of Chapter 3 on studying security group usage and designing our cloud security group analysis system is from a conference paper, titled "*Understanding Security Group Usage in a Public IaaS Cloud*", which appeared in the Proceedings of the 35th IEEE Conference on Computer Communications (INFOCOM), San Francisco, CA, USA, April 10-14, 2016 [5]. Our developed *unified* network controller that exerts SDN-like, fine-grained path control in hybrid SDN networks is presented in a conference paper titled"*Magneto: Unified Fine-grained Path Control in Legacy and OpenFlow Hybrid Networks*", which appeared in the Proceedings of ACM Sigcomm Symposium on SDN Research (SOSR), Santa Clara, CA, USA, April 03 - 04, 2017 [6]. This constitutes Chapter 4. Part of Chapter 5 is from a paper titled "*Clairvoyant Networks*", which is currently under review in a conference in the networking area.

Chapter 2

Background and Motivation

Enterprise networks (*e.g.*, the networks in campuses and corporations) and data-center networks (*e.g.*, the network infrastructures hosting cloud services) play a critical role in modern society, since most users, devices and applications reside in these networks. With emerging techniques such as the Internet of Things, virtual and augmented reality, more devices are connected to these networks everyday. It is reported that the number of devices connected to the Internet will be three times as high as the global population in 2020 [7]. The global data center traffic will grow 3-fold from 2015 to 2020 [8], and the global enterprise networking market is expected to reach USD 64.63 billion by 2024 [9]. Judicious network management facilitates a healthy and sustainable network. Managing these networks to provide secure and reliable network services with high availability and performance is a central problem for computer networking research.

2.1 Today's Network Management

Network devices started from parcels of protocols. The control plane (*i.e.*, learning and building the routes in a network) and the data plane (*i.e.*, forwarding packets based on the decision made by the control plane) reside in a same network device, as shown in Figure 2.1. Managing a network generally works as: logging into the devices and running vendor-specific commands to set up configurations and tune protocol behaviors. Management tools are developed based on operators' experience and customized to specific cases—they are vendor-dependent, low-level, and inextensible.

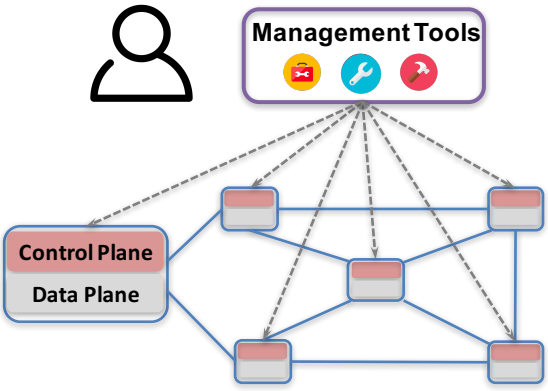


Figure 2.1: Today's network management.

2.1.1 Management Tasks

Network management in enterprises and data centers involves numerous tasks such as registering new devices (*e.g.*, servers, switches, and routers), setting up security policies, configuring routing policies, as well as obtaining network visibility to enable monitoring, measurement and trouble-shooting. This thesis focuses on security policy configuration, routing and monitoring.

Security policy configuration: Security policy rules are configured to restrict the traffic from/to certain source/destination hosts, in order to guarantee the network and system security. For example, in an enterprise network, traffic between unrelated teams and departments is isolated. In a multi-tenant data center, a tenant should not have access to other tenants' virtual machines (VMs) without permission granted. In addition, in each tenant, VMs should have different permissions to access resources based on their roles (*e.g.*, a public front-end web server shall not have open access to database servers). These security policies are typically fine-grained and involves low-level configurations.

Routing: A typical enterprise network is comprised of (legacy) Ethernet switches with VLAN capabilities. Standard layer 2 Ethernet switches perform two main functions: learning (the next-hop switch towards a destination MAC address) and forwarding (a packet according to learned information). To learn the next-hop switch for a packet, layer 2 switches broadcast the packet on all ports except the one on which the packet arrived. To prevent loops they restrict the underlying topology to a spanning tree by

turning off (*e.g.*, using the Spanning Tree Protocol (STP)) or aggregating (*e.g.*, using link aggregation) multiple links. In other words, ports associated with "off-tree" links are de-activated or blocked. The path of a packet is static and changes only if there are topology or configuration changes in the network. To increase path diversity, operators can slice the network into multiple VLANs, each with its own spanning tree and set of forwarding entries.

Monitoring: Operators need to monitor the network traffic for various purposes such as accounting, anomaly detection, troubleshooting, and traffic engineering. For example, operators in data centers may need to identify the large flows in the network to better configure their routing for traffic engineering (*i.e.*, a flow is a sequence of packets that share the same packet header properties such as source address/port, destination address/port, and/or protocol). In addition, having the visibility of network can help track network events and topological information.

2.1.2 Limitations of Existing Network Management

Coupled control plane and data plane: As shown in Figure 2.1, the control plane is coupled with the data plane. The control plane on each device exchanges information with other devices in the network, and then computes its routing/forwarding table. The data plane forwards packets based on the tables built by the control plane. Each device only has a partial (local) view of a network, so that it cannot make network-wide decisions and it is slow to recover from failures.

Vendor-specific and low-level configurations: Network devices are sold as monolithic boxes with the coupled control plane and data plane, and the configuration interface varies from vendor to vendor. No change on the control-plane or data-plane can be easily made since these boxes are closed and proprietary. In this case, network management eventually becomes configuring the control plane with the given vendor APIs. As a result, operators have to master low-level details to be able to tune protocol behaviors correctly.

Error-prone manual process: Manual configuration has been widely-used in network management and proved to be error-prone [4]. Dependency in different network elements and the increasing scale make it a Herculean task to manage a network without a good automated system. For example, just to bring a server online in a campus

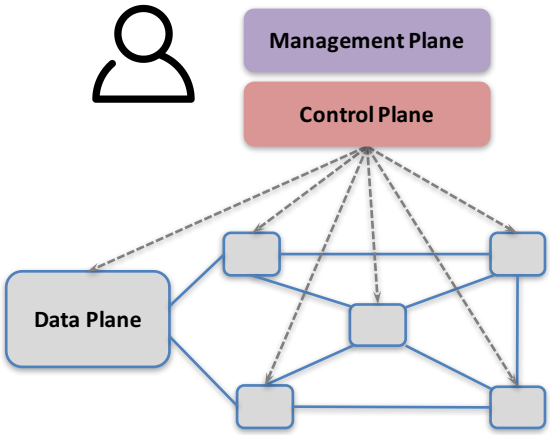


Figure 2.2: Network management with software-defined solutions.

network, operators need to add a new entry in the DHCP server, configure VLANs correctly, set up firewall rules, and make sure no blocking configuration exists in switches or routers.

2.2 Rethinking Network Management with Software-Defined Solutions

2.2.1 Software-Defined Networking

Software-Defined Networking (SDN) [10, 11, 12, 13, 14] decouples the control plane from the data plane, as shown in Figure 2.2. With a *(logically) centralized* control plane [15, 16] and a *programmatic match-action* data plane abstraction [17, 18, 19], SDN enables flexible, fine-grained network control and monitoring, and offers the potential to transform network management: from today's largely *manual* process to an *automated* process governed by (high-level) *network policies*. The control plane (*i.e.*, controller, a.k.a., network OS) decides the behaviors of data-plane switches by installing match-action rules using a standard protocol (*e.g.*, OpenFlow). The match determines which headers in the packet to match and their values, and the action(s) determines a sequence of actions to perform on the matched packets. For example, forwarding the packets destined to a server Alice to port 2 can be defined with "destination MAC address = Alice's MAC address" in the match field and "output to port 2" in the action field.

With the centralized control plane, network operators can easily access to the global view of a network in order to make good network-wide decisions. The interface between control plane and data plane is open and vendor-agnostic, so different controllers can be developed to serve diverse network set-ups. Management tasks can also be implemented as software applications running upon the control plane, so that automating management tasks with defining high-level intent is not mission impossible any more. SDN made a grand opening in providing software-defined solutions to network management. Taking this inspiration, we can explore how to manage network judiciously.

2.2.2 Security Policy Configuration in IaaS Clouds

Cloud computing enables ubiquitous access to a shared pool of computing, network, and storage resources. It provides users with convenient and on-demand capabilities to store, process, and retrieve data in data centers. In Infrastructure-as-a-Service (IaaS) cloud computing platform such as Amazon EC2 and OpenStack, cloud customers can even outsource the physical and virtual resources to develop their own applications. Nonetheless, security is one of the main concerns in the adoption of cloud computing. As an example, the data breach at Target resulted in the loss of personal and credit card information of up to 110 million individuals [20]. To this end, researchers have developed many security solutions to be offered as a cloud service. For instance, virtual machine introspection as a cloud service is offered to allow customers to develop their own tamper-resistant security tools without relying on cloud providers.

In IaaS cloud platforms, security group is the primary means for cloud tenants to configure security policies to protect their virtual machine (VM) instances against attacks. A security group is a (named) container for a set of security rules. It provides tenants the ability to specify the type and direction of traffic allowed by VM instances. Unlike the conventional network firewalls where rules are typically configured by experienced network administrators, security groups and their constituent security rules are specified by cloud tenants, some of whom may lack an adequate network management background to properly configure security groups. Unfortunately, vulnerabilities in one tenants VMs pose security threats not only to the tenant itself but also to the entire cloud platform. Ensuring that each cloud tenant properly specifies his/her security groups and the rules therein is therefore paramount to cloud platform providers.

In Chapter 3, we present a tool that helps operators and users understand and refine security group configurations.

2.2.3 Unified Fine-Grained Routing Control with Incremental SDN Deployment

Today's networks are maintained by "masters of complexity": network operators, who have accumulated tremendous experience, devote significant efforts to operate highly-available networks and troubleshoot complex problems. The reason behind is that legacy networks lack global visibility and proper abstraction which can enable centralized control. SDN provides a logically-centralized interface to control and interact with network devices. Operators perform network management tasks through software programs executed from a logically centralized controller. The flexible control and global visibility offered by SDN can reduce the cost of operating a network by half [21]. However, fully benefiting from SDN requires a considerable initial investment: network providers must upgrade or replace existing legacy switches with SDN-enabled switches (*e.g.*, whose forwarding behaviors are programmable remotely from a logically-centralized controller using a specialized protocol such as OpenFlow [17]).

Recent work, both in academia and industry, attempts to reduce the capital expenditure of SDN while maintaining most of its benefits, by upgrading only a few, strategically chosen legacy switches in a network. We refer to such networks as hybrid SDN networks. Although effective at controlling routing paths through SDN-enabled devices, the control points are also limited to the SDN-enabled devices. None of the previous work can dynamically affect the forwarding behaviors of the remaining legacy devices and, consequently, the paths through the legacy sub-network. To control those paths in the legacy sub-network, manual configurations or additional protocols need to be further applied. In Chapter 4, we present a system that enhances routing flexibility so as to increase network utilization and efficiency.

2.2.4 On-demand Network Visibility for Better Monitoring and Policy Enforcement

Real-time monitoring of *all* network flows is critical for preserving network health and detecting operational problems in enterprises. To make flows visible, network operators deploy monitoring tools (*e.g.*, NetFlow, SNMP [22, 23, 24]) pervasively throughout the network to cover flow paths or mirror packets to dedicated appliances. For example, to identify large flows, NetFlow-enabled switches sample packets and build flow-level packet counters. Monitoring tools must be strategically deployed across the data plane to enable network-wide visibility, and carefully tuned to avoid overloading the data plane [25].

Another approach is to jointly optimize routing and monitoring tasks such that flows traverse specific monitoring devices [26, 27, 28]. This requires a fully-programmable data plane (*e.g.*, SDN-enabled switches) which may not be readily available and is expensive to deploy. In Chapter 5, we present a system that gains on-demand network visibility by making a network partially programmable.

Chapter 3

Understanding Security Group Usage in a Public IaaS Cloud

3.1 Introduction

In Infrastructure-as-a-Service (IaaS) cloud platforms such as Amazon EC2 and Openstack [29, 30], *security group* is the primary means for cloud tenants to configure security policies to protect their virtual machine (VM) instances against attacks [31, 32]. Although similar to the conventional network firewalls in many ways, security groups have several distinct features that make their configuration somewhat more complex and trickier to use. Unlike firewalls where rules are typically configured by experienced network administrators, security groups and their constituent security rules must be specified by cloud tenants, some of whom may not be well-trained or lack an adequate network management background to properly configure security groups. Unfortunately, vulnerabilities in one tenant's VMs pose security threats not only to the tenant itself but also to the entire multi-tenant cloud platform. Ensuring that each cloud tenant properly specifies his/her security groups and the rules therein is therefore paramount to multi-tenant cloud platform providers.

In this chapter we first conduct a measurement-oriented analysis of security group configuration and usage by tenants in an IaaS cloud based on *real-world* datasets. Our goal of this measurement study is multi-fold: to understand what are the usage patterns ("good" and "bad" practices) in how cloud tenants configure their security groups, what

they attempt to achieve, what are the common issues and potential security vulnerabilities, and how to help cloud tenants refine their security group configurations to prevent these issues and vulnerabilities. As an example of "bad" practices and potential vulnerabilities revealed by our analysis of a multi-tenant IaaS cloud system security group dataset, we find that a number of tenants simply allow all traffic (`0.0.0.0/0`) from both the external Internet and within the cloud to access their VMs. In general many tenants inappropriately configure their security groups by using loose, and sometimes inconsistent, rules (see § 3.2 and §3.5 for more discussion on these and related points).

Motivated by the results and insights obtained from this measurement study, we propose and develop a cloud security group analysis tool called *Socrates*. Socrates takes the security group settings of each tenant, the VM mapping as well as the observed traffic flows (both *allowed* and *denied*) as inputs, and employs visual analytics to assist cloud tenants in understanding the static and dynamic *access relations* among VMs based on the security groups they have specified and the traffic observed. Furthermore, our tool also helps cloud tenants diagnose potential misconfigurations and provides suggestions to refine security group configurations based on real traffic traversing the tenant VMs. As a result, cloud tenants can view their security group configurations in a high-level, visualized manner, and revise their security group settings immediately after they realize some configurations do not meet their intent.

By applying *Socrates* to all existing tenants hosted on our IaaS cloud using the *week-long* datasets, we report some key results and lessons we have learned in §3.5. As alluded earlier, security groups are often set up by tenants who are "ordinary" application developers and may not be experts in network security. Hence we expect to see many configuration errors. Nonetheless we are surprised to find many configuration issues, some of which can lead to potential security vulnerabilities. For example, we find that more than 80% tenants configure security groups in a loose manner. In contrast, some tenants verbosely set rules leading to giant security groups with hundreds of rules. While many tenants create multiple security groups for their VMs, a large number of them do not have a clearly defined *structure* in mind when creating these security groups. *Socrates* also reveals many *redundant* or *inconsistent* rules in the security group configurations, likely the result of tenants' lack of knowledge about the intricacies of security groups (*e.g.*, rule ordering is immaterial) or mistakes in configuring rules. To

the best of our knowledge, this is the first work of analyzing cloud security groups. Our work sheds light on understanding the common usage for security groups and proposes a tool to better understand, diagnose and refine security group configurations.

3.2 Overview and Datasets

In this section, we first describe the basic concepts of IaaS cloud security groups and then the datasets used in our study.

IaaS Cloud: VMs and Security Groups. Creating a cloud application in an IaaS cloud starts with launching VM instances. One critical step in launching a VM is to configure security groups. A security group is a container for a set of security group rules. It provides tenants the ability to specify the type and direction of traffic allowed by VMs. Security groups are applied to individual VMs, whose private IP addresses are dynamically assigned only at the time they are launched – in other words, such private IP addresses are, in general, unbeknownst to the tenant at the time he/she specifies the security group rules. Unlike conventional firewall rules, the default action of security group rules is *deny*; thus, a tenant needs to explicitly specify what type of traffic (in terms of protocol and port) and from where (*e.g.*, in the form of a public or private IP address prefix) can access his/her instances. Furthermore, security groups can be "nested" in the sense that the security group rules in one security group, say, `SG-A`, can use the name of another security group (either belonging to the same tenant or another tenant), say, `SG-B` – in lieu of a (public or private) IP address prefix – to explicitly specify that the traffic from VMs in `SG-B` can access VMs in `SG-A` on ports permitted by the security group rules. Furthermore, the ordering of rules within a security group is immaterial; security group rules are not prioritized as in the case of firewall rules. Therefore, the most permissive rule gets applied if more than one rule is created for a specific port or IP range. Table 3.1 shows an example of a security group. Due to nested security group rules or IP ranges' coverage on VMs, there are *dependencies* among various security groups defined by one tenant (and sometimes among multiple tenants). Ideally, a tenant should create security groups based on the roles of VMs in a cloud service he/she develops.

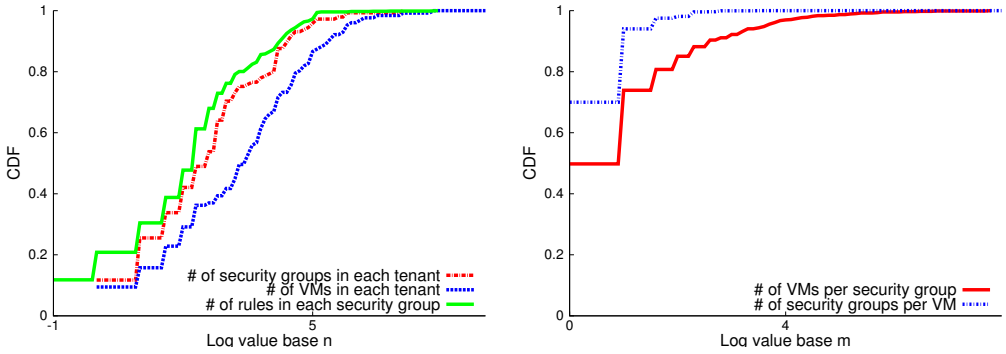
Before getting launched, each VM must be assigned with at least one security group.

Table 3.1: An example of security group with 3 rules.

Action	**Protocol**	**Port Range**	**IP Range**
ALLOW	TCP	80 – 5666	10.0.10.0/24
ALLOW	UDP	68 – 68	SG-A
ALLOW	ICMP	8,0	11.22.33.44/32

A *default* security group is defined for all tenants, which by default denies all ingress traffic and allows all egress traffic and the traffic among the VMs associated with the *default* security group. When a VM is launched, it is associated with the *default* security group if no security group is specified by the tenant. In addition, a tenant can define and customize new security groups. One VM can be associated with multiple security groups, and one security group can be assigned to a collect of VMs. Therefore, one tenant can have a set of security groups and VMs, and the mapping between them can be fairly complex. Finally, tenants can configure security groups by adding or deleting rules, but not modifying an existing rule (*A rule cannot be modified once it is created*). Changes are automatically applied to the running VMs associated with the security group.

Datasets. The datasets used in our study are collected from a single multi-tenant data center running the OpenStack cloud software. There are three types of datasets: the `secgroup` dataset, the `VM-layout` dataset and the `sFlow` dataset. The first type of dataset is called `secgroup` which contains security groups and the constituent rules defined by cloud tenants. It contains five main fields: tenant ID, security group name, protocol type (TCP, UDP, or ICMP), port range (or ICMP type and code), and the source (IP range in the CIDR notation or the name of a security group). A tenant ID allows us to match the tenant across multiple datasets. The second type of dataset is the `VM-layout` that stores information about running VMs in the cloud at any given time. The important fields are VM name, tenant ID, associated security group(s), public IP address (if assigned), and private IP address. Both the security group and VM layout datasets are collected from the cloud configuration database. The last type of dataset is `sFlow` that contains flow traces (both *allowed* and *denied* flows) collected at each switch by random sampling. It stores packet header information, including source and destination IP addresses, TCP/UDP port numbers, time, switch identifier,



(a) The number of security groups and VMs in each tenant, and the number of rules in each security group.

(b) The number of security groups associated with each VM and the number of VMs associated with each security group.

Figure 3.1: Basic statistics of security group usage by tenants.

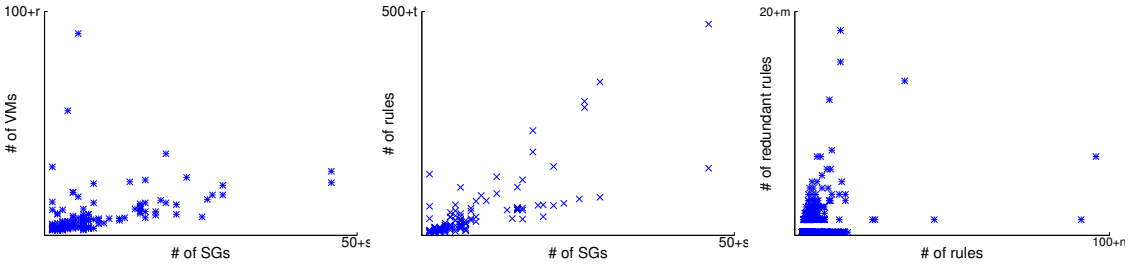
and source/destination switch ports associated with the packet.

3.3 Current Usage of Security Groups

As *security group* is still a relatively unknown concept to many IaaS cloud customers, we first conduct an extensive measurement-based analysis of security group configuration and usage by tenants in an IaaS cloud based on *real-world* datasets. In the following, we present some basic statistics and a few key results from this measurement-based analysis of the multi-tenant IaaS cloud security group, VM and flow datasets. The goal is to identify the common usage patterns in how cloud tenants generally configure their security groups. We also briefly point out a few "bad" practices in cloud tenant security group configurations, which we will expand on further in Section 3.5 in conjunction with the discussion of the results obtained from applying our *Socrates* tool.

3.3.1 Basic Statistics

Fig. 3.1a shows the number of security groups and the number of VMs in each tenant, as well as the number of rules that each security group has. The x-axis is the normalized value where n is a base value. As the results show, around 10% tenants have only one security group, and the remaining have at least two security groups. Most tenants have less than several dozen security groups, whereas not every security group plays a



(a) The relation between security groups and VMs.

(b) The relation between security groups and rules.

(c) The distribution of rules in every security group.

Figure 3.2: The relation between security groups and VMs, and between security groups and rules.

different role. The number of rules in security groups (log value) starts with -1 (it could be any negative value, and we use -1 for simplicity) at x-axis, because some tenants have empty security groups that do not have any rule. Apart from 15% no-rule security groups, most security groups have less than one hundred rules.

Given the tenants that have multiple security groups and multiple VMs, we are interested in the association between security groups and VMs (shown in Fig. 3.1b). Our results show 50% security groups are associated with only one VM. In the remaining half of security groups, most of them are associated with a few dozen VMs, and very few of them are associated with a very large number of VMs. 70% VMs are assigned with only one security group, and others are assigned with multiple security groups.

As depicted in Fig 3.2a, generally the more VMs a tenant has, the more security groups it tends to have, so the more sophisticated system the tenant is expected to build. However, we also notice that some tenants have a large number of VMs but only contain a few number of security groups. One reason is that these tenants have simple architectures but very high workload so that they need to launch a number of VMs to balance the workload. Another possible reason is that the tenants glue all rules in a few security groups instead of reasonably separating them into more security groups (discussed in Section 3.5).

3.3.2 Rules in Security Groups

To investigate how security groups are configured in tenants, we start from studying their rules. Each rule consists of port and IP range. Based on the IP range, a rule can be classified into three groups: only accepting the external traffic[1], only accepting the internal traffic, accepting both external and internal traffic (*e.g.*, 0.0.0.0/0). As a security group is a set of rules, we can further determine whether a security group is: accepting only the external traffic, accepting only the internal traffic, or both. In our `secgroup` dataset, we find that 42% rules allow external traffic (referred to as *external rules*) and they are distributed in 34% security groups. 39% rules allow internal traffic and they are distributed in 61% security groups (referred to as *internal rules*). 19% rules allow traffic from everywhere (0.0.0.0/0) and they are distributed in 50% security groups (also referred to as *external rules*). In addition, a rule can be very restrictive or very permissive by setting the decimal in CIDR notation. For example, decimal 32 is used to specify an individual IP address, and decimal 0 means cover all IP addresses. We find that 34% rules use decimal 32 (*e.g.*, a.b.c.d/32). Around 60% external rules use decimal 32 to set individual IP addresses, while most internal rules use IP blocks (*i.e.*, $0 <$ decimal < 32).

In terms of the port range used by each rule, our results show that the top five mostly-used TCP port ranges are 80, 443, 8080, 22, and 1-65535. We are surprised to see many rules use 1-65535 in port range, because simply allowing all ports is very risky. Moreover, ICMP rules' configurations are more biased, more than 90% ICMP rules are coarsely set to allow all types and all codes.

Furthermore, we also observe that 14% security groups distributed in 48% tenants contain *redundant* or *inconsistent* rules: for instance, two rules allow traffic on the same port (say, TCP 443) but from two different IP address prefixes, one a sub-prefix of the other. Such rules make little sense, as traffic will be allowed by the less restrictive rule. This appears to be a result of a tenant attempting to modify an existing rule by adding a new rule but forgetting to delete the old rule. Fig. 3.2c shows the number of rules and the number of redundant rules each security group has.

[1]We define *external IPs* as the addresses that do not belong to the IaaS cloud. In contrast, *internal IPs* are owned by the cloud. For simplicity, external traffic is referred to as the traffic between internal IPs and external IPs, and internal traffic denotes traffic between internal IPs.

3.3.3 Security Group Dependency

Based on the understanding of rule settings and the fact that a security group is actually a set of rules, now we study the security group usage at the tenant level. As a rule can be categorized into external rules and internal rules, a security group can also be categorized into *external* (only has external rules), *internal* (only has internal rules), and *mixed* (has both external and internal rules).

In our dataset, all tenants allow external traffic to some extent. 15% tenants consist of only external security groups. The security group rules for external traffic should be more carefully configured in order to protect the VMs from outside attacks. As most tenants have multiple security groups, we are interested in the relationship among the security groups in the same tenant. The relation can be depicted as a graph (discussed in details in Section 3.4), where each security group is a node and each directed edge indicates that the successor allows certain type of traffic from the predecessor. 70% security group graphs have bidirectional edges between each pair of security groups. Among them, around 40% share same port ranges on the same pair of bidirectional edges.

3.3.4 Bad Practice in Security Group Configurations

As part of the motivation for the *Socrates* tool, we provide some sample results from an initial analysis of the `secgroup` dataset (see Table 3.2). Our analysis shows that "good practice" (*i.e.*, use *nested* security groups to scope communications among VMs) is not widely adopted yet – only 5% tenants employ nested security group rules. It reveals a fact that many cloud tenants have not completely grasped the concept of security groups or the subtle intricacies involved, and as a result, often specify rules that are either semantically incorrect or too loose.

We find that 24% tenants open all ports on their VMs to accept traffic. Out of these tenants, 19% tenants allow traffic from 0.0.0.0/0, *i.e.*, accept traffic from anywhere on the Internet. This extremely-permissive setting exposes the tenants as victims of potential security attacks because it does not filter any traffic. When looking into the IP ranges specified in the rules, we find that some tenants do not even understand the CIDR notation. 13% tenants in our dataset have rules with `a.b.c.d/0` (where

a.b.c.d != 0.0.0.0) and 5% have rules with 0.0.0.0/x (where x!= 0), which is semantically incorrect. In addition, many tenants often use rules with 10.0.0.0/8 instead of *nested* security groups when their intention is to simply enable communications among VMs between certain security groups (see Section 3.5 for more detail).

Table 3.2: Initial analysis of `secgroup` dataset.

Usage	Tenants	Rules
Bad usage	24%	Open all ports (1–65535)
Bad usage	13%	Meaningless CIDR: a.b.c.d/0 (a.b.c.d != 0.0.0.0)
Bad usage	5%	Meaningless CIDR: 0.0.0.0/x (x!=0)
Good usage	5%	Use nested security groups

In some tenants' configurations, all of their security groups surprisingly open all ports for all VMs belonging to the tenants. This loose setting arouse our investigation in their flow usage. We find that their flows are much more restrictive (*i.e.*, only contacting some ports from a subset of VMs) compared to the configured rules. These observations motivated us to design and develop a tool which visualizes the security group setting, analyzes real flows against the security group rules, and generates diagnostic reports, which detailing problems with the security group rules. Section 3.4 explains the design of our tool *Socrates*.

3.4 *Socrates*: A Security Group Analysis Tool

In this section we provide an overview of *Socrates* – a cloud security group analysis tool that we have developed [2] – and briefly describe its key components. Part of the rationale for *Socrates* is our recognition that many IaaS cloud tenants are "ordinary" application developers who may not be very familiar with notion of security group and its intricacies, let alone being a network security expert. Ideally, when a tenant develops and deploys a service or application on an IaaS cloud platform, security groups should be created to reflect the roles of VMs and meet their security and management requirements. As we briefly discussed in Section 3.3 and further expanded on in Section 3.5,

[2]The name, *Socrates*, is derived as an anagram of the capitalized letters in *SECurity gROup AnalySis Tool*.

creating and configuring security groups can be quite a challenging task for many tenants. Unfortunately, vulnerabilities in one tenant's VMs pose security threats not only to other tenants but also to the entire multi-tenant cloud platform. Hence ensuring security for each tenant is crucial.

Socrates is designed to assist cloud tenants in understanding their security group settings and help them diagnose their configuration issues. *Socrates* takes the security group settings of each tenant, the VM mapping as well as the observed traffic flows (both *allowed* and *denied*) as inputs, and produces a visual representation of *security group/VM structure* as well as a *diagnosis and recommendation report* to help tenants diagnose and improve their security group configurations based on observed network traffic. *Socrates* consists of three key components: *visualizer*, *flow analyzer*, and *recommender*, see Fig. 3.3 for a schematic illustration.

Security Group/VM Structure Visualizer: It displays the dependencies of security groups and VMs through directed graph representations based on the (static) security group settings and the (dynamic) VMs to security group mappings. The dependency between security groups reveals the cloud service infrastructure design that a customer has envisioned. Hence, a directed graph (referred to as a *security group structure graph*) is generated to represent security groups of one tenant, where nodes stand for individual security groups and the edges encode dependencies between security groups. Each directed edge indicates the successor security group allows the traffic satisfying the specific port range and IP range from the predecessor security group (or external networks). From the graph, we further identify *tiers* to which security groups belong. A security group is defined as tier N if and only if it allows traffic from tier $N-1$ but not from any other lower tiers. For example, *tier 1* security groups contain at least one rule explicitly allowing external traffic. *Tier 2* security groups allow traffic from *tier 1* security groups but not from external networks. After building the security group structure graph, we next add the VM-level structure into the graph by mapping VMs to assigned security groups. VMs are displayed as rectangular nodes inside the corresponding security groups. In addition, we introduce edges between VMs within the same security group to indicate that traffic is allowed between a particular pair of VMs. On the other hand, the dependency between VMs across two security groups are already captured by edges between security groups. Fig. 3.4a depicts the security group/VM structure for a real

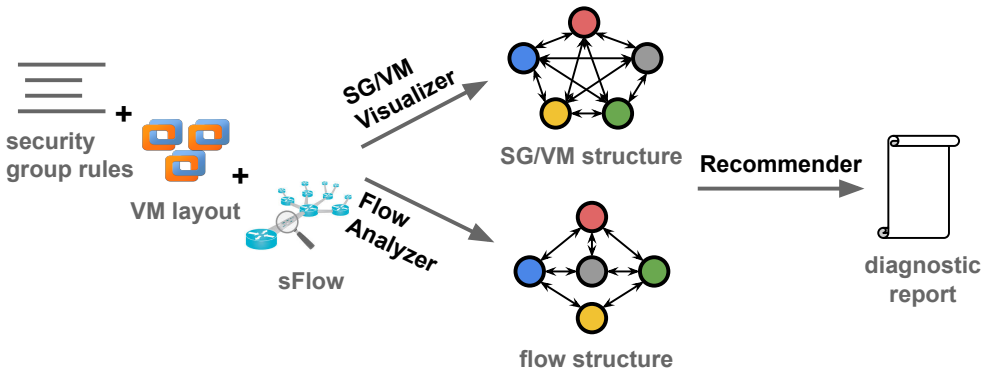
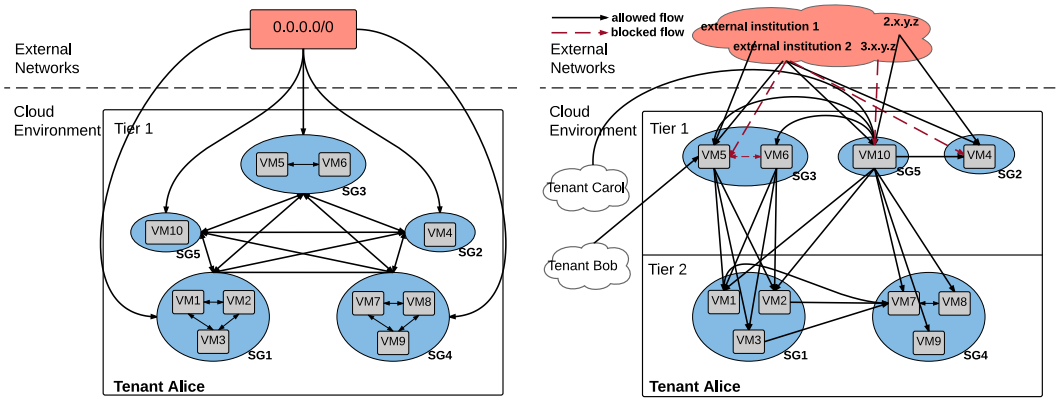


Figure 3.3: *Socrates* workflow.

tenant from our datasets, nicknamed "Alice", where all security groups belong to tier 1 since they all allow external traffic.

Flow Analyzer: It infers the cloud service infrastructure design by analyzing the traffic flows associated with the service, both *allowed* and *blocked*. A particular flow between a source VM and a destination VM is considered *allowed* or *blocked* based on whether it is allowed by rules in the destination VM security group or not. To build the flow structure, the analyzer marks flows as either *allowed* or *blocked* by checking each flow with the rules of all the associated security groups. With both allowed and blocked flows, we build the flow structure, a directed graph at the VM-level, based on flows' src_IPs, dst_IPs and dst_ports. The directed edges are labeled as "allow" or "block" to differentiate the flows are accepted by rules or not. This VM-level graph can also be easily converted to a security group level graph by aggregating the flows of VMs belonging to the same security group. An example of flow structure for tenant Alice is shown in Fig. 3.4b, where we see that the (dynamic) flow structure is more "sophisticated", *e.g.*, containing more "tiers", than the simple tier-1 structure depicted in Fig. 3.4a.

Recommender. It utilizes the information generated by the security group structure and flow structure in order to identify the differences between the rules created and the flows accepted or denied by customer VMs. It further alters customers about the mismatch as as well offers suggestions to modify security group by providing the analysis



(a) Security Group/VM structure of tenant Alice.

(b) Flow structure for tenant Alice.

Figure 3.4: Examples of SG/VM Structure and Flow Structure.

report[3]. If the security groups are defined too widely, we can recommend that tenants refine their security groups to restrict ports and IPs that do not appear in the flow structure. For example, given most security group and VM structures are complete graphs, the flow structure can show more sophisticated structures. It also analyzes the causes of blocked flows. In terms of the "block" edges, if the same kind (same src_IP, dst_IP and dst_port) of blocked flows keeps coming for a long time, *Socrates* raises alert to customers in case of potential misconfigurations or attempt of attacks.

3.5 Security Group Configuration Analysis and Diagnosis

To evaluate the efficacy of the proposed tool, we apply *Socrates* to examine and analyze the security group configuration issues of all tenants on our IaaS cloud, using *one-week* datasets of tenant security group settings, VM layouts and traffic flows. In the following, we will first provide a brief overview of the results we have obtained, highlighting a few configuration issues uncovered by *Socrates*. Then, we will discuss the *structural analysis* of security group configurations to illustrate how *Socrates* can help tenants visually

[3]We quantify mismatches using the Jaccard distances of corresponding IP ranges and port ranges within two structures. While the threshold on Jaccard distances can be set according to management needs, we choose a conservative value of 0.1 in our experiments. In other words, we only study most significant mismatches.

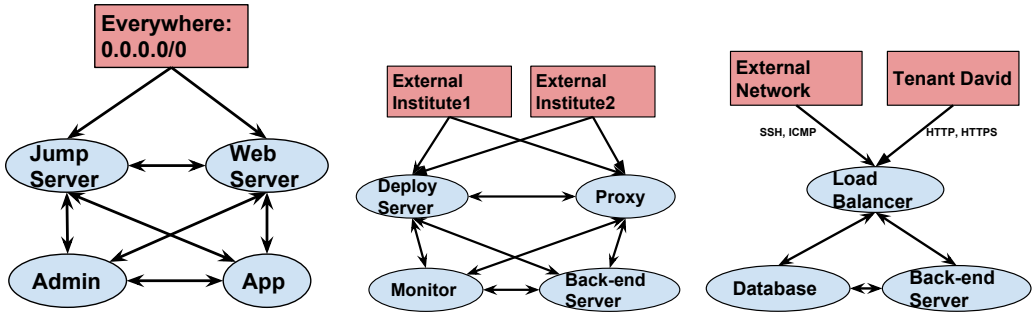
analyze their security group settings and track their changes over time. We will also present analysis and discussion of the uncovered configuration issues in the end.

3.5.1 A Brief Overview of Results Obtained via *Socrates*

As alluded earlier, in contrast to firewall rules which tend to be configured by professional network operators, security groups are often set up by tenants who are "ordinary" application developers who may not be an expert in network security. Hence we expect to see many configuration errors. Nonetheless we are surprised to find that around 50% tenants have at least one security group without any rule configured. A few of them even have VMs assigned to these empty security groups. As revealed by the flow analysis, many tenants configure rules *loosely*, for example, using rules with sources such as 0.0.0.0/0 or 10.0.0.0/8, without regards to the actual application requirements. Other tenants configure rules *verbosely*, *e.g.*, by creating one rule per VM (*i.e.*, using a /32 IP address as the source), which leads to a giant security group with many rules. While many tenants create multiple security groups for their VMs, a large number of them do not seem to have a clearly defined *structure* in mind when creating these security groups (see Section 3.5.2). Very few leverage (*nested*) security group names as an effective way to permit only traffic between VMs of specific security groups and restrict traffic from other VMs not belonging to these security groups; instead they often resort to either using overly permissive rules with 10.0.0.0/8 or 10.0.0.0/24 or creating one rule per VM address as stated earlier. *Socrates* also reveals many *redundant* or *inconsistent* rules in the security group configurations, likely the result of tenants' lack of knowledge about the intricacies of security groups (*e.g.*, rule ordering is immaterial) or mistakes in configuring rules.

3.5.2 Structural Analysis of Security Group Configurations

Socrates takes the security group settings of each tenant, the VM mapping as well as the observed traffic flows as inputs, and employs visual analytics to assist cloud tenants in understanding the static and dynamic access relations among VMs based on the security groups they have specified and the traffic observed. In this section we report some key results we have obtained by applying *Socrates* to all tenants' security group



(a) Public Customer Facing Tenant. (b) Private Enterprise Application Tenant. (c) Back-end Service Support Tenant.

Figure 3.5: Three categories of tenant structures.

settings using the one-week datasets.

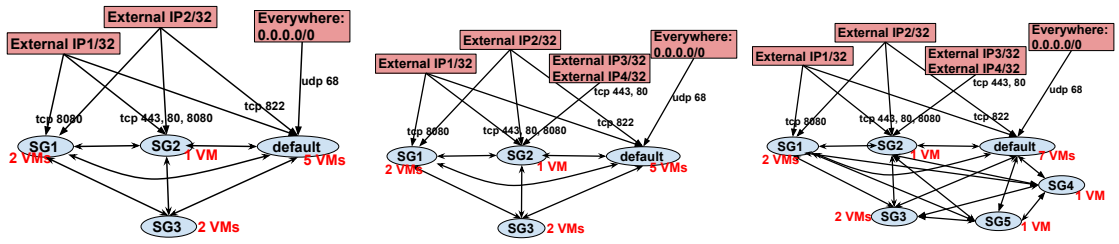
The goal of *structural analysis* of security group configurations is to help tenants visualize and understand the relations among various security groups they have configured, whether they reflect the roles and application requirements of the VMs associated with these security groups, and how the observed traffic (both *allowed* and *blocked*) traffic match what the security group rules are intended to accomplish. We find that although a majority of tenants have more than one security group configured, many do not appear to have a clearly defined structure in mind. We observe that 51% tenants tend to have a single-tier, whereas the remaining have two tiers. No tenant has more than two tiers, despite some of them have configured a large number of rules that apply to a large number of VMs.

Fig. 3.5 depicts three *representative* examples of *two-tiered* security group structures generated by *Socrates*, which we classify them as: (i) *public customer facing web service*, (ii) *private enterprise application*, and (iii) *back-end service support*. The tenants in category (i) use the IaaS cloud platform to deploy a public web service serving customers from everywhere (0.0.0.0/0), while the tenants in category (ii) may have likely migrated a private enterprise application to the IaaS cloud platform and thus restrict it to a specific set of IP address ranges belonging to the private enterprises. The tenants in category (iii) on the other hand leverage the the IaaS cloud platform for back-end service (*e.g.*, databases) support for another service (or tenant). In this case, we often see that traffic from another tenant (often in category (i)) is allowed. Judging based on the

names of the tenants involved, the two tenants likely belong to the same owner. In category (iii), although some traffic from one or two external networks are allowed, they are primarily for the management purpose (SSH or ping from the external networks). The remaining rules are all restricted to internal VMs, and the commonly used ports are for web proxy services, databases services, synchronization services, and monitoring services. For tenants with two tiers, 61% are public customer facing, 32% tenants are private enterprise application, and 7% tenant are back-end service support.

The (static) structure of the security group settings is also reflected by the *dynamic structure* in the observed traffic flows through the flow analysis. We find that VMs associated with the tier-1 security groups often function as web servers/web proxies, load balancers, or jump servers. VMs associated with many tier 2 security groups are running database services, certain application services or monitoring services. In particular, we notice that VMs associated with the "monitoring" security groups only send traffic to other VMs, but hardly allow traffic from other VMs.

Potential Vulnerabilities. As stated earlier, we find that many tenants have a single-tier structure. Further analysis reveals that for a majority of tenants (70%), their security groups form a *full mesh*, *i.e.*, any pair of security groups are allowed to communicate with each other. Based on our observation, the existence of many full meshes is caused by tenants extensively using 10.0.0.0/8 and 10.x.x.x/24 to grant access to their VMs. In particular, we find that 16% tenants use 10.0.0.0/8, 23% of tenants use 10.x.x.x/24, and 44% tenants use 10.x.x.x/y where $8 < y < 24$. On the other hand, based on the analysis of observed traffic flows of these tenants, these rules are meant to apply to VMs belonging to the tenants' own security groups. These overly permissive rules imply that any other VMs in the cloud platform (even those not belonging to the tenants) are allowed to access these VMs, thereby creating potential security vulnerabilities. As a tenant may not know the private IP address range dynamically assigned to its VMs, many resort to simply use 10.x.x.x/8 or 10.x.x.x/24 to cover its VMs, as opposed to use the names of its security groups directly. A particularly concerning problem with these tenants with such a "full-mesh" single-tier structure is that as some of the VMs are associated with security groups which are "public customer facing", *i.e.*, allowing external traffic to access them. As a result, one compromised customer-facing VM can lead to other VMs (even though they are not assigned any public IP address, thus



(a) (b) (c)

Figure 3.6: Snapshots of an actively-developing tenant Eric. The number of VMs is normalized.

not directly addressable from the outside world) being potentially compromised. By analyzing both the *static* security group settings and *dynamic* VM layouts as well as the observed traffic flows, *Socrates* is capable of alerting tenants about such potential security vulnerabilities and suggest alternative security group structures based on the common traffic patterns observed among VMs.

3.5.3 Tracking Configuration Changes

By applying *Socrates* to the security group settings, VM layouts and flow datasets over one week, we also track how tenants modify the security group rules to experiment with and refine their settings to meet application needs, or adapt to changing application requirements. By observing what flows are allowed and what are blocked, and how they vary over a period of one week, we can also get a sense of what are "normal" traffic activities, but what may be "anomalous" traffic activities.

In our datasets, 14% of the tenants made security groups configuration changes in the one week period. Some tenants made many changes, such as adding new security groups, deleting existing security groups. Other tenants made slight modifications to existing security groups by either adding new rules (*e.g.*, open more ports or allow more IPs) or deleting existing rules. In addition, some new VMs were launched with newly-added security groups, while some existing VMs were terminated with removing existing security groups. We observe that among the tenants which generate most traffic (top 11% tenants), their security group configurations hardly change at all over the one week period, although the numbers of VMs launched and the amount of flows may vary over

time. This observation indicates that the services operated by these top tenants are well-developed and running in a stable mode. In contrast, we find that a few tenants with quite less traffic frequently changed their security group configurations and VM association over the one week period, suggesting that they were still developing their services and were experimenting with the security group settings.

Fig. 3.6 provides an example where a tenant Eric modifies its security groups in the one week period. Initially (see Fig. 3.6a), the tenant has four security groups and five VMs. The number beside each security group indicates the number of VMs associated with it. Note here all VMs are also associated with the *default* security group. Except SG3, the other security groups allow external traffic, so that they are in Tier 1. After half a day (Fig. 3.6b), additional rules are added to SG2 to allow HTTP and HTTPS traffic from more external IPs. By analyzing the observed flows of this tenant, we see traffic from these newly-allowed external IP addresses in the same hour as the rules were added. Several days later (Fig. 3.6c), two new security groups, SG4 and SG5, were added, with rules allowing traffic from other security groups. Similar to SG3, these two new security groups function as back-end application services, but with different ports open. Two new VMs were launched, one associated with SG4 and one with SG5. The flow datasets reveal that indeed there is traffic between the two VMs.

This example helps illustrate that when a tenant modifies its security group settings, its intention is often to permit or restrict certain traffic. Therefore, the dynamic structure in the observed flows should also change accordingly. However, we have also observed that the dynamic flow structures change before the security group configuration is modified. While flow structures change may be due to, *e.g.*, attacks, when such changes persist over time, they can be an indication of changing application requirements or a change in the nature of services. For example, if the same type of flows continuously get blocked for a long time, this may be due to a "misconfiguration" (a previously too restrictive rule may need to be relaxed). In this case, our tool will raise a red flag to notify the tenant.

Potential Vulnerabilities. As tenants add new rules or modify their existing security group settings over time to meet changing application or service requirements, many forget to delete their old rules. These lead to *redundant* or *inconsistent* rules in the security group configurations, say, with multiple rules apply to the same or overlapping

or a subset of IP address blocks which permit traffic on a different set of TCP/UDP ports. Some of these configuration issues may be due to tenants' lack of knowledge in security group configurations: they may not realize that once a rule is set, it cannot be modified/updated; creating a new rule, say, applies to the same IP prefix block but with a new port range, does not invalidate the previously configured rule – old rules must be explicitly deleted when they are no longer needed. Some tenants may simply forget to delete old rules when creating new rules or forget about the existence of these old rules. Given that the ordering of rules in a security group does not matter, such mistakes can potentially create security holes, especially when a new rule is put in place to limit certain unwanted traffic that an old rule previously allows. *Socrates* is able to explicitly flag such redundant or inconsistent rules and alert the tenants about such configuration issues which potentially create security vulnerabilities.

3.5.4 Loose, Verbose, and Inconsistent Configurations

As mentioned earlier, it is surprising that most tenants (more than 80%) set security groups in a loose manner. Tenants are suggested to restrict IP ranges to credible IP blocks by using proper CIDR notation or security group names. In addition, tenants are encouraged to use nested security groups to specify IP ranges. This feature enables allowing traffic from all VMs associated with the nested security group without using individual IPs or IP ranges. If there is any VM newly-launched or stopped, the tenant does not need to modify the rules. Based on our observation, the flow structure often time reveals a subset of the access relationship than the security group structure generated by security group settings. It also tends to reveal more about the tier structure. One of the key reasons is that tenants extensively set security groups loosely, such as 10.0.0.0/8 and 10.x.x.x/24. Hence, the corresponding security group settings can be refined to be more restrictive based on the flow structure. In addition to setting rules loosely, some tenants also set security groups loosely. Specifically, instead of setting security groups distinctly to present their roles, the tenants simply replicate security groups over and over again. In this case, these security groups have exactly the same rules but different security group names. However, by looking into their flow structures, we clearly see each of these security group's real intentions and functions are entirely different. Hence, we suggest the tenant should refine security groups to reflect their

distinct roles.

In contrast to setting security groups loosely, a few tenants in our cloud set their security groups in an extremely verbose manner. Especially some tenants only have one giant security group with hundreds of rules. We observe that it is because the rules are set by using individual IPs of VMs. If there is any VM launched or stopped, the same type of rules need to be added or deleted.

Redundant or *inconsistent* rules are the multiple rules which apply to the same or a subset of IP address blocks/ports which permit traffic on a different set of ports/IP address blocks, one a subset of the other. Such rules make little sense, as traffic will be allowed by the most permissive rule. Among the tenants which have redundant rules, 30% tenants have more permissive rules followed by more restrictive rules, 40% tenants have more restrictive rules followed by more permissive rules, and 30% tenants have both cases. With the analysis of `sFlow` dataset, in terms of the tenants which have more permissive rules coming first, 83% tenants have most flows allowed by the former permissive rule but cannot be allowed by the latter restrictive one. 17% tenants have most flows allowed by the former permissive rule and could also be allowed by the restrictive rule. In terms of the tenants which have more restrictive rules coming first, we find that 75% of them have only a few flows allowed by the former restrictive rule and most flows accepted by the later permissive rule, which indicates the customer intends to create a more permissive rule to replace the restrictive one, but unfortunately forgets to delete the restrictive rule. 25% tenants have most flows allowed by the former restrictive rule while only a few allowed by the latter permissive rule.

3.6 Summary

The contributions of this chapter are summarized below: i) Using the real-world datasets from a multi-tenant IaaS cloud, we have conducted a first measurement-based analysis of security group configuration and usage. Through this measurement-based analysis, we have studied the common usage patterns in how cloud tenants generally configure their security groups. We revealed some issues and potential vulnerabilities in cloud tenant security group configurations. ii) Motivated by the results and insights obtained from this measurement study, we then proposed and developed a security group analysis

tool called *Socrates*. *Socrates* enables tenants visualize and hence to understand the static and dynamic access relations among VMs. *Socrates* also helps diagnose potential misconfigurations and provides suggestions to refine security group configurations based on real traffic traversing tenants VMs. iii) We have applied *Socrates* on all tenants hosted on the IaaS cloud and demonstrate its effectiveness in helping cloud tenants analyze, visual, diagnose and refine their security group settings. To the best of our knowledge, we believe that our work is the first to analyze cloud security group usage based on real-world datasets, and to develop a tool to help cloud tenants to understand, diagnose and better refine their security group configurations. Our work sheds light on the common usage ("good" and "bad" practices) of cloud security groups and on how to design better and more secure cloud systems and services.

Chapter 4

Unified Fine-Grained Path Control in Legacy and OpenFlow Hybrid Networks

4.1 Introduction

With a *(logically) centralized* control plane [15, 16] and a *programmatic match-action* data plane abstraction [17, 18, 19], software-defined networking (SDN) [10, 11, 12] enables flexible, fine-grained network control and monitoring, and offers the potential to transform network management: from today's largely *manual* process to an *automated* process governed by (high-level) *network policies*. Studies show that SDN can reduce the cost of operating a network by half [21]. Thanks to these benefits, earliest adoption of SDN occurs in data centers, where size renders manual network management difficult. SDN has also been applied to wide-area networks (WANs), *e.g.*, those connecting multiple data centers [33, 34, 35], to more effectively manage expensive bandwidth of WANs and the edge networks of data centers that interconnected with multiple other Autonomous Systems (ASes) [36, 37]. Internet Service Providers (ISPs) or carrier networks have also started considering the adoption of SDN [38].

However, the majority of networks on the Internet are *enterprise* networks, where

deployment of SDN faces major challenges. Unlike data center networks with well-structured topologies, enterprise networks often evolve in a not well-planned, "organic" fashion as the need for network connectivity and bandwidth grows. As a result, enterprise network topologies can be arbitrary—often with many quasi-tree like structures as access networks and a "semi-mesh" campus core network connecting those access networks. Further, most enterprise networks [39, 40, 41] comprise layer 2 (L2) Ethernet switches supporting VLANs and use layer 3 (L3) IP routers as gateways to route between VLANs or for external Internet connectivity.

Converting enterprise networks to SDN is difficult. First, budget constraints make it cost-prohibitive [42] to perform a "wholesale" upgrade from a Ethernet-based "legacy" network to a programmable[1] SDN network. In addition, enterprises often run *mission-critical* applications that rely on existing legacy hardware devices and/or software components. Recent work has proposed partial SDN deployments where only a fraction of the switches are upgraded to SDN [43, 44, 45, 46, 42]. Operators control the SDN-enabled devices but cannot affect the paths through the legacy network. Much of the routing must be coarsely engineered using VLANs or tunnels [44, 46] or left to the latitude of L2 protocols such as Spanning Tree Protocol (STP) or ECMP. This limits network control as some policies cannot be installed. Most network operators of enterprise networks have little or no experience in managing and operating new SDN networks. They need to gradually gain experience and build confidence in running SDN networks.

In this chapter, we present a novel framework for *incremental and graceful transition* of legacy networks comprised primarily of L2 Ethernet switches to SDN-capable networks. Rather than performing an expensive and disruptive *wholesale upgrade* or converting parts of the network into *"SDN islands"*, we argue and advocate that it is not only possible but in fact *advantageous* to migrate a network of legacy switches to a *hybrid* network of *mixed* legacy switches and SDN-capable switches while at the same time reaping as much benefit as a *fully deployed* SDN network. The key idea behind our proposed framework, which we call Magneto, is that by replacing one or a few *strategically placed* L2 legacy switches with SDN-capable switches, or by adding SDN

[1]We interchangeably use the terms *programmable*, *OpenFlow(-enabled)*, or *SDN(-capable)* to refer to devices whose forwarding tables can be configured remotely from a centralized controller.

switches, we can influence the forwarding behavior of legacy switches and end hosts (*i.e.*, "magnetize" them). This allows us to *gain visibility and exert control* over legacy devices *without the need to make any modifications to existing legacy hardware devices* or *software components* (*e.g.*, configuring VLANs or virtualization).

Magneto employs two key mechanisms to exert SDN-like control over legacy L2 switches: *telekinesis* where we leverage OpenFlow switches to inject *seed* packets to manipulate legacy switches' forwarding tables; and *magnet addresses* where we use gratuitous ARP messages to populate the ARP tables at end hosts with "fictitious" or "illusory" MAC addresses for the purpose of gaining network visibility and controlling routing and forwarding behaviors of end hosts and legacy switches. We describe the baseline *telekinesis* mechanism *without the use of magnet MAC addresses* in Section 4.3. This is the path control mechanism used in our prior work [47] for hybrid networks. This baseline mechanism injects seed packets with the *native* MAC address of a destination host of the path to install. This mechanism suffers from two shortcomings: i) it can only exert limited, coarse-grained (*i.e.*, per-destination) path control and ii) the path installed may be unstable. In Section 4.4, we introduce magnet addresses and outline how they can be used to exert *fine-grained* (*i.e.*, per source-destination pair) path control in hybrid networks and formulate the (path) *controllability* condition. We present the detailed Magneto fine-grained path control components in Section 4.5.

We evaluate Magneto using simulations on larger enterprise network topology and on a real-world testbed (Section 5.7). We demonstrate that Magneto is capable of enforcing complex policies in hybrid networks, *e.g.*, routing along multiple disjoint paths to the same destination for congestion control or load balancing [48, 49, 50]. Magneto can install diverse paths with little control and data plane overhead, and exert full control over routing even when only 20% of the switches are SDN-capable.

In a nutshell, Magneto provides a *unified* network controller to exert SDN-like control over both programmable and legacy switches in hybrid networks. It enables network operators to transition legacy networks to SDN networks *in stages* by gradually replacing more and more legacy switches with SDN-capable switches *as needed and as budgets allow*. Further, it allows network operators to gracefully experiment with SDN networks to gain experience and build confidence while eliminating or minimizing service disruption. Our work demonstrates that it is possible to enjoy much of the benefits of

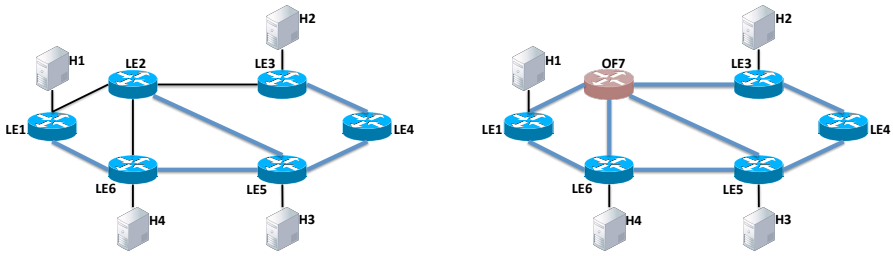


Figure 4.1: Path diversity in legacy (left) and hybrid (right) networks: In legacy networks, the spanning tree created by STP (solid blue lines) constrains the end-to-end paths. In hybrid networks, all links that are part of the spanning tree or adjacent to an OpenFlow switch can be used.

a wholly deployed SDN network but *at a fraction of the cost* by strategically replacing only a few (e.g., 20%) legacy switches with SDN-capable switches.

4.2 Background and motivation

We discuss previous work on partial SDN deployment and identify their benefits and shortcomings. We then introduce our solution for unified network management for hybrid legacy and OpenFlow networks.

4.2.1 Hybrid Networks

There are several approaches to transition a legacy network to an SDN-capable network [45, 46, 51, 52, 43, 42, 53]. First, vendors can install additional software modules on legacy switches to make them programmable. ClosedFlow [51] configures legacy switch features to mimic and support the OpenFlow API and make the switch appear OpenFlow-enabled to an SDN controller. This approach however requires modification and installation of additional software modules to process and support OpenFlow APIs; the solution is vendor-specific and highly depends on the features supported by the legacy switches.

Another approach is through *access edge* control via virtualization. For example, VMWare's NSX [44] forgoes physical programmable switches altogether and implements SDN at the edge of the network as part of hypervisors. This approach requires upgrading and installing new networking software on all end devices in a network. This can be a

challenging task in most enterprise networks, and may not be feasible in some enterprise networks where many devices are BYOD (bring your own device).

Third, operators can replace all legacy switches in a subnet with SDN switches to create *SDN islands* [54, 43, 45]. The SDN and legacy zones are independent and managed separately. The benefits of SDN are limited solely to SDN islands and cannot be extended to legacy networks. In addition, network operators must run multiple control & management planes, one for legacy networks, and one for each SDN island. This can add additional burden on network operators and further complicate their management tasks.

First proposed by Levin *et al.* [46], a fourth approach is to simply replace a few (strategically placed) legacy switches with, or introduce a few, new SDN switches *in piecemeal* fashion. We refer to such a network of mixed legacy and SDN-capable switches as a *hybrid* network. Hybrid networks offer the potential to benefit from the flexibility and visibility offered by SDN without the considerable initial investment of fully transitioning to SDN. By replacing legacy switches with SDN-capable switches (*e.g.*, OpenFlow switches), we add control entry points into the network to implement more complex policies and exploit path diversity in the underlying physical network topology by going beyond the default spanning tree used by legacy switches.

Consider the example topology on the left in Figure 4.1. The paths between every pair of hosts in the legacy network are constrained by the L2 spanning tree constructed by STP. This can create congestion on the spanning tree links, while the other links are not utilized. If we upgrade switch $LE2$ to an OpenFlow switch to create a hybrid network (Figure 4.1(right)), we expose alternate paths through the OpenFlow switch. This allows us to install more diverse policies (*e.g.*, balance traffic across multiple links to eliminate congestion). Further, the addition of OpenFlow switches provides fine-grained flow-level visibility (*e.g.*, between two hosts).

Most existing approaches for managing hybrid networks incur significant management complexity, as they control legacy and SDN switches via different mechanisms. For example, Panopticon [46] resorts to VLANs (whereas NVP [44] employs tunnels) to set up paths through the legacy network, requiring additional (manual) configurations. Further, they do not provide sufficient agility (as VLANs cannot be reconfigured rapidly [45]) nor diversity (as tunnels cannot select the underlying physical path). In

summary, while offering the potentials for increased flexibility and visibility at reduced cost, hybrid networks still face complex management issues. Ideally, we would like an unified framework to control both legacy and SDN switches that offers *flexible forwarding control* with *simple network management* and at *low operating cost*.

4.2.2 Our Solution

We propose Magneto, a network controller framework to *incrementally* and *gracefully* transition a legacy network to an *SDN-capable* network by *strategically* placing – or replacing a few legacy switches with – OpenFlow switches. We use SDN-capable switches to influence and exert control on the forwarding behavior of legacy switches and end hosts and to obtain similar network *visibility* and routing *control* as in a fully deployed SDN. This is achieved via two mechanisms: *telekinesis* where we leverage OpenFlow switches to inject *seed* packets to effect changes in legacy switches' forwarding tables; and *magnet addresses* where we employ gratuitous ARP messages to populate the ARP cache tables at end hosts with "fictitious" or "illusory" MAC addresses for the purpose of gaining network visibility and controlling forwarding behaviors of end hosts and legacy switches.

Magneto unifies hybrid network management using a single OpenFlow-based network controller. Unlike previous approaches, Magneto does not need switch-vendor support or additional modules on legacy switches. Although it does not obviate the use of VLANs or tunnels, Magneto provides path control and flexibility without the overhead of configuring VLANs or setting up tunnels.

Conceptually similar to Fibbing [55], Magneto indirectly affects network routing by injecting fake and harmless information into the network. However, due to the self-learning switch algorithm, STP and VLANs used by L2 switches, they pose unique and different challenges from L3 IP distributed routing, and therefore call for different mechanisms. Magneto operates at the data link layer by affecting the forwarding behavior of legacy L2 switches. In contrast, Fibbing [55] aims at introducing a centralized control over distributed L3 IP routing by injecting carefully crafted "fake" routing messages via OSPF. Fibbing's goal is to enhance the flexibility, diversity and reliability of L3 routing, *not to transition legacy enterprise networks to SDN-capable networks*, as enterprise networks comprise primarily legacy switches.

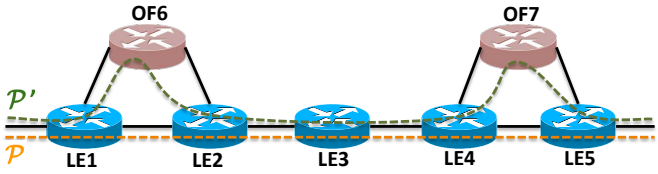
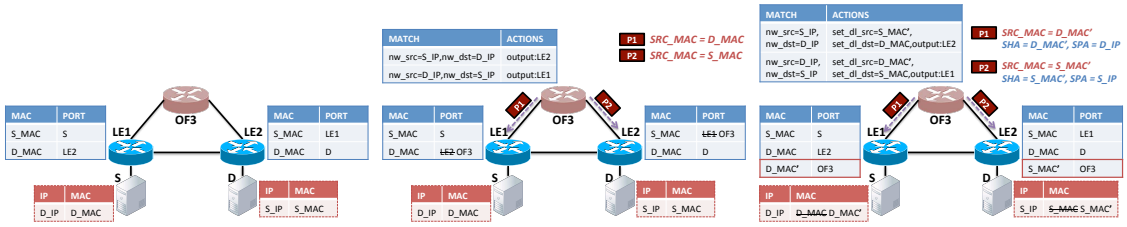


Figure 4.2: Example of path update: $\mathcal{P}$ is the current path, $\mathcal{P}'$ is the new path; $LE1$, $LE2$, $LE3$, $LE4$, $LE5$ are legacy switches, $OF6$ and $OF7$ are OpenFlow switches; $(LE1, OF6, LE2)$ and $(LE4, OF7, LE5)$ are the subpaths that need to be updated.



(a) (b) (c)

Figure 4.3: Path update between two hosts, S and D, in a hybrid network consisting of two legacy switches ($LE1$ and $LE2$) and one OpenFlow switch ($OF3$). Switch forwarding tables are in blue, host ARP caches are in red. (**(a): original network state**) Traffic between S and D flows through path $(LE1, LE2)$; (**(b): basic path update**) $OF3$ injects seed packets to $LE1$ and $LE2$, triggering updates in their forwarding tables and thereby changing the path between S and D to $(LE1, OF3, LE2)$; (**(c): enhanced path update**) $OF3$ injects seed packets with magnet MACs to both legacy switches and end hosts changing the path to $(LE1, OF3, LE2)$.

4.3 Baseline Telekinesis Mechanism

By replacing a few strategically placed legacy switches with SDN-capable switches, we are able to, not only directly control the SDN switches, but also configure and influence the forwarding behavior of legacy switches. This allows us to enhance routing flexibility and increase network utilization through path diversity. We start by describing the baseline *telekinesis* mechanism to control paths through legacy devices, introduced in our prior work [47]. We then discuss the shortcomings of this baseline (coarse-grained) path control mechanism. In Section 4.4 and 4.5 we describe an enhanced design for fine-grained path control which circumvents these shortcomings.

4.3.1 Basic Idea and Key Mechanisms

Assumptions. In a hybrid network with both *legacy* switches and *programmable* switches such as OpenFlow switches, we can only control the programmable switches via a central SDN controller, but we cannot directly update the legacy switch forwarding entries. We assume that each legacy switch runs MAC learning and that the legacy network is configured, either manually or automatically, to avoid forwarding loops (*e.g.*, with STP). We call the collection of legacy links that results after this configuration the *network underlay*. The underlay is always a tree or a collection of trees. End hosts maintain ARP tables to map MAC addresses to IP addresses.

Goal. Given a path (*i.e.*, a sequence of switches) $\mathcal{P}$ between two hosts A and B in a hybrid network and a candidate new path $\mathcal{P}'$, reconfigure the network so that all traffic between A and B traverses $\mathcal{P}'$ [2] or decide that the new path is infeasible. This may require updating all switches along the new path. In Figure 4.2, $(LE1, LE2, LE3, LE4, LE5)$ is the old path $\mathcal{P}$ and $(LE1, OF6, LE2, LE3, LE4, OF7, LE5)$ is the new path $\mathcal{P}'$.

Seed Packets. The key idea behind *telekinesis* is to use OpenFlow switches to send special ("custom-made") packets–referred to as *seed* packets–to the legacy switches on the new path. This relies on the ability of an SDN controller to send PacketOut control messages to OpenFlow switches and instruct them to send custom-made packets into the network. The seed packets take advantage of MAC learning to *manipulate* legacy switches into updating a single forwarding entry in their routing tables.

Under the baseline telekinesis mechanism, seed packets must satisfy two requirements. First, their source MAC address must be the same as the destination MAC of the path we want to install in the legacy switch. This ensures that only the forwarding entry corresponding to this MAC address is updated. Second, they must arrive at a legacy switch on a link that is part of the path we want to install. This ensures that the affected entry is correctly updated with the next-hop information. For example, if we want to modify the action of a forwarding entry for MAC m from *"send to port p1"* to *"send to port p2"*, we create a packet whose source address is m and make sure it

[2] Throughout the chapter, we refer to this process as installing, configuring, enforcing, or updating $\mathcal{P}'$.

arrives at the switch on port $p2$. The MAC learning algorithm sees the packet arriving on $p2$ and assumes its source address m is reachable on $p2$, therefore updating the corresponding forwarding entry.

Path Update. Updating the path $\mathcal{P}$ between two hosts to a new path $\mathcal{P}'$ requires updating all switches on $\mathcal{P}'$ if the two paths are disjoint. When old and new paths overlap, we need to update only the switches where the paths diverge. We define an *update subpath* as the sequence of adjacent switches that must be updated during a path change. For example, in Figure 4.2, we must update OpenFlow switches $OF6$ and $OF7$ and legacy switches $LE1$, $LE2$, $LE4$, and $LE5$. Legacy switch $LE3$ remains unchanged. The update subpaths are $(LE1, OF6, LE2)$ and $(LE4, OF7, LE5)$.

The above example illustrates that by simply replacing one or a few legacy switches with OpenFlow switches, we can in fact gain more by leveraging these programmable switches to effect changes in legacy switches via *telekinesis*. However, there are limits as to what path *telekinesis* may control. This is because the seed packets that *telekinesis* uses to remotely manipulate a legacy switch's forwarding table must arrive at the switch on a link that is part of the path telekinesis wants to install. This leads to the following *control* condition of the *baseline* telekinesis mechanism.

(Control condition of baseline telekinesis) *A path is feasible if (a) every link on it is part of the L2 underlay or adjacent to an OpenFlow switch, and (b) every update subpath contains at least one OpenFlow switch.*

The first part of the condition ensures that a seed packet reaches the right interface on a legacy switch so it can trigger a forwarding entry update. The second part of the condition ensures that there is at least one OpenFlow switch to send a seed packet to every legacy switch on the update subpath. We will show in Section 4.4 how these conditions can be further relaxed via Magneto's enhanced *fine-grained* path control mechanisms.

4.3.2 Shortcomings of Baseline Telekinesis

This baseline telekinesis mechanism suffers from two shortcomings: i) it can only exert limited, *coarser-grained* (i.e., per-destination) path control and ii) the path installed may be *unstable*. We discuss them in more details below.

Data Rate (Mbps)	Update Success
0.1	94%
1	80%
10	59%
100	0%

Table 4.1: Successful path updates using the basic *telekinesis* mechanism, when we vary the data plane rate. A path is successfully updated if it becomes stable in less than five seconds from the time when we send the first seed packet.

Coarse-grained paths. Legacy network L2 routing is destination-based: a destination MAC is associated with a single interface (and implicitly, path) on each switch. Legacy network operators create path diversity at increased management cost using VLANs or ECMP. OpenFlow networks can install more fine-grained paths as they can match traffic based on both source and destination MACs. Our basic scheme inherits the limitations of legacy networks: the update of a path triggers updates on all paths to the same destination. In the example on the right of Figure 4.1, both $H1$ and $H4$ send traffic to $H3$. The legacy switch $LE6$ will forward all the packets destined to $H3$ towards $OF7$, including the packets from $H4$ to $H3$, if we change the path between $H1$ and $H3$ to $(LE1, LE6, OF7, LE5)$.

Unstable paths. MAC learning reacts to all incoming packets, regardless of whether they are seed packets or not. A forwarding entry for a MAC address m may change every time the switch relays a packet from m. This can make even the simplest path update unstable. To better understand this limitation, we consider a common scenario that can lead to unstable paths: traffic between two hosts flows in both directions, such as when the hosts use TCP to communicate. Consider the example in Figure 4.2. If the update from $\mathcal{P}$ to $\mathcal{P}'$ on the direct path is not fast enough, packets on the reverse path (which is still $\mathcal{P}$) can invalidate the forwarding entry updates and revert them to the original states corresponding to $\mathcal{P}$. A simple solution to make paths stable when reverse traffic is present is to continually inject seed packets until forwarding entries reach a stable state. The frequency of seed packets depends on the rate of data packets. As long as seed packets arrive faster than data packets, they can override any change made by reverse path packets and the original direct path will eventually be updated.

We evaluate this scheme in a small real-world testbed, shown in Figure 4.3. We

set up a simple mesh topology with one OpenFlow switch and two legacy switches. Each legacy switch is connected to a server. Initially, the default path between servers traverses only the legacy switches. We continually send TCP traffic between the servers. At the same time, we update the path to traverse the OpenFlow switch as well. An update is successful if the path becomes stable in less than five seconds from the first seed packet. We compute the percentage of successful updates as we vary the data rate over one hundred runs. Table 4.1 shows the results. The basic update mechanism success rate decreases as the data plane rate increases and falls to 0 for rates of at least 100 Mbps. In summary, flooding legacy switches with seed packets does not guarantee a successful path update. In addition, it may generate significant network overhead. In the next section we present an enhanced path control mechanism that installs stable paths with almost zero network overhead.

4.4 Magnet MAC Addresses and Fine-Grained Path Control

We now enhance the baseline telekinesis by integrating it with magnet addresses to achieve *fine-grained* (*i.e.*, per source-destination pair) path control. In the following we first introduce magnet (MAC) addresses and briefly discuss how they can be used to gain visibility and enforce access control for IP-based applications and services in hybrid networks. We then outline the key ideas behind Magneto's fine-grained path control. The detailed path control processes is described in Section 4.5.

4.4.1 Magnet MAC Addresses & Visibility

Magneto introduces the key notion of *magnet (MAC) addresses* to influence and manipulate both end hosts forwarding behaviors as well as those of legacy switches. A *magnet address* is a fictitious MAC address that does not correspond to any real host on the network, but is created by our Magneto controller for the purpose of gaining network visibility and controlling routing & forwarding behaviors of end hosts and legacy switches. These magnet addresses are the main reason we name our framework Magneto: similar to the magnetism in physics, by manipulating the magnet addresses, we can dynamically attract end hosts and legacy switches to route and forward packets

towards OpenFlow switches (*paramagnetism*), as well as "repulse" routing away from OpenFlow switches (*diamagnetism*). We "magnetize" a hybrid network by controlling the (magnet) MAC address mappings at end hosts via unicast gratuitous ARP messages generated by the Magneto controller (via OpenFlow switches).

To gain visibility and enforce access control (for unicast IP-based applications), we can pre-populate hosts ARP cache via gratuitous ARP to eliminate the broadcast ARP query process. For some "assets" servers that we want to monitor and control the access all the time, we can pre-populate the IP-MAC address mappings in all hosts on the same L2 LAN segment with the "assets" servers' magnet addresses. Since the ARP packet size is small (though it may vary but is typically less than 80 Bytes), the overhead of doing this pre-population is negligible. Further, the controller can adjust the mappings dynamically via new gratuitous ARP messages to alter forwarding paths of host.

4.4.2 Telekinesis with Magnet Addresses

We now present the Magneto's fine-grained path control mechanism which seamlessly integrate telekinesis with magnet addresses to achieve fine-grained path control.

When sending seed packets, we set the source address as a magnet MAC address associated with the path destination, rather than the real (native) MAC address of the destination host. The seed packet triggers the installation of a forwarding entry for the magnet MAC address. We also require that the seed packets are ARP packets and can reach the source host of the path. Thus, the source learns to associate the destination with its new magnet MAC address. Magneto enhances routing by enabling multiple paths between source-destination pairs, which enables re-routing a portion of the traffic on a congested path to a new path instead of the default spanning tree path. In the baseline mechanism, if one source changes its path to a destination, it will affect the paths from all other sources too. Magneto uses different magnet MAC addresses for other source hosts to update legacy switches, hence packets destined to the same destination from different sources can now traverse different paths. The last OpenFlow switch on the path rewrites the magnet MAC address to the native MAC address based on the destination IP address.

Figure 4.3 illustrates the enhanced path control at the granularity of per-source-destination pair. To install a new path between $(LE1, OF3, LE2)$ between S and D,

we generate a new magnet MAC address D_MAC' associated with D and send a seed (*unicast*) ARP packet from $OF3$ to S with the new magnet MAC as the source MAC address in the Ethernet packet. The sender hardware address (SHA) field of the ARP message is also set to D's magnet address, *i.e.*, $SHA = D\_MAC'$. This packet triggers the addition of a new forwarding entry at switch $LE1$ and the update of the ARP table on S to add one entry for D's magnet MAC address and corresponding incoming port. The forwarding table of switch $LE2$ is updated in a similar manner.

By integrating telekinesis with magnet MAC addresses, we are able to exert fine-grained (per source-destination pair) path control, thereby significantly increasing path diversity that can be exploited for routing and traffic engineering. As a destination host can be associated with multiple magnet MAC addresses (for different source hosts), we can install multiple paths to the same destination host. Compared to the baseline telekinesis mechanism, this leads to the following *relaxed* path control conditions:

(Control condition of telekinesis with magnet MAC addresses) *A path is feasible if (a) every link on it is part of the L2 underlay or adjacent to an OpenFlow switch, and (b) the network contains at least one OpenFlow switch.*

The use of magnet MAC addresses also isolates the old path (*e.g.*, the default spanning tree path) and the new path between two hosts. This eliminates the unstable path problem associated with the baseline telekinesis mechanism. We note that packets traversing along the reverse direction of an old path (*e.g.*, the default spanning tree path $(LE1, LE2)$ in the bottom example in Figure 4.3) cannot rewrite the forwarding entries for a new path in the legacy switches, since these packets must contain either the native MAC address or a different magnet MAC address. In a sense, magnet MAC addresses achieve a form of network versioning, similar in spirit to the consistent network update mechanisms for SDNs proposed in [56, 57]. As the native MAC addresses of hosts can always be learned by broadcasting on the default spanning tree, if we want to revert a new "off-spanning-tree" path back to the default spanning tree path, Magneto can generate a seed packet with the native MAC address in gratuitous ARP message (while the magnet MAC address is used as the source MAC address in Ethernet packet header) and send it via an OpenFlow switch on the off-spanning-tree path. Using the bottom example in Figure 4.3, to revert the path back from $(LE1, OF3, LE2)$ (the off-spanning-tree path) to the default spanning tree path $(LE1, LE2)$, Magneto crafts a seed packet

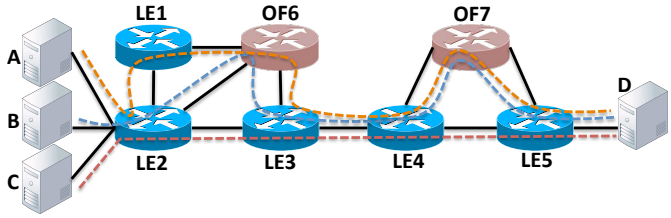
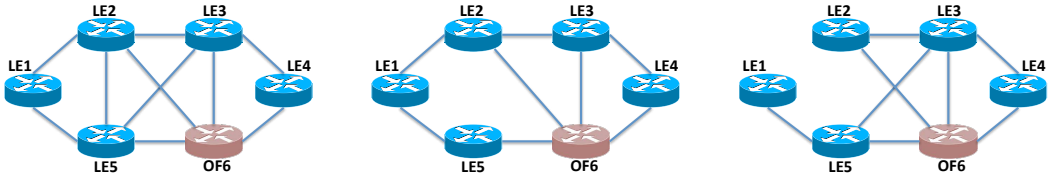


Figure 4.4: Three source hosts A, B, and C send traffic to the same destination host D via different paths.



(a) (b) (c)

Figure 4.5: The network topology and underlay affect the diversity of paths enabled by Magneto. Given a topology with five legacy switches and one OpenFlow switch (a), the performance of Magneto varies across two possible sets of usable links (b,c) (spanning tree links plus OpenFlow-adjacent links).

and sends it towards S with $SRC\_MAC = D\_MAC'$ and $SHA = D\_MAC$ (the similar process is applied for D).

4.5 Magneto Path Control Components

In this section we describe the detailed fine-grained path control components employed by Magneto: *path verification*, *path update*, and *magnet routing*. Given a network configuration (*i.e.*, forwarding tables on all switches and the network underlay) and a new path $\mathcal{P}'$ to install between two hosts attached to the network, Magneto first checks whether the path is feasible. It then installs the path by sending seed packets with magnet MACs to every legacy switch on the path. To route each packet to the destination along the new path, Magneto must rewrite packet headers and eventually replace the magnet MACs with the real MACs.

4.5.1 Path Verification and Path Update

Given a path $\mathcal{P}'$ and the current network configuration, path verification determines whether $\mathcal{P}'$ is feasible in the network. For each link in the new path that is not present in the old path, Magneto verifies whether it is part of the L2 spanning tree or adjacent to an OpenFlow switch. This ensures that seed packets can install the path. To maintain an updated view of the spanning tree, Magneto periodically queries port information from each legacy switch. In addition, Magneto checks that at least one switch on the new path is OpenFlow-enabled, unless the new path is only in the L2 spanning tree. This ensures that we can send seed packets.

To install a new path, Magneto generates seed packets and sends them to both legacy switches and hosts. We describe both actions next.

Generating seed packets. The role of seed packets is to trigger updates to legacy switch forwarding tables and host ARP caches. Each seed packet is an ARP packet whose source MAC address in the Ethernet header is a magnet MAC address associated with the destination of the path. In addition, we set the ARP header to map the magnet MAC to the destination's real IP address.

How do we generate magnet MAC addresses? The simplest way is to generate one magnet MAC address for each path through the network. However, this would create a large number of magnet MAC addresses and may inflate unnecessarily the size of switch forwarding tables. We observe that all feasible paths are constructed from the same set of usable links (*i.e.*, links that are part of the underlay or adjacent to OpenFlow switches). Further, adjacent legacy switches are controlled by the same seed packet.

We define a *magnet subpath* as a sequence of adjacent legacy switches on the path to install. A magnet subpath is part of the L2 network underlay and lies between two OpenFlow switches or between a host and an OpenFlow switch. All legacy switches in the same magnet subpath can be updated by the same seed packet from the same OpenFlow switch. Magnet subpaths are different from update subpaths, defined in Section 4.3 as sequences of adjacent switches, not necessarily legacy, that must be updated when installing a new path.

We generate one magnet MAC for each unique magnet subpath. We associate the

first 42 bits of the address with the OpenFlow switch used to update the magnet subpath (*e.g.*, we hash the OpenFlow switch DPID) and the last six bits with the interface[3] of the same switch used to send the seed packet that updates the path. This assignment ensures that *the maximum number of magnet MACs is at most the sum of the number of interfaces across all OpenFlow switches in the network.* In our experiments, we generated at most 5,000 different magnet MACs in a network with 100 OpenFlow switches.

Consider the example in Figure 4.4. The paths between A and D and between B and D have a common magnet subpath $(LE3, LE4)$. Magneto generates one, rather than two, magnet MAC address for this subpath. The OpenFlow switch $OF7$ sends a seed packet with the magnet MAC to both switches on the subpath.

Sending seed packets. To support forwarding entry updates on legacy switches, we introduce a new primitive, called LegacyFlowMod. We use LegacyFlowMod to generate seed packets and send them to the switches we want to update. LegacyFlowMod relies on OpenFlow's PacketOut functionality, which allows us to use any OpenFlow switch we control to send a packet on the data plane. Given a path to update, LegacyFlowMod calls PacketOut for every legacy switch to update. We must be careful to call PacketOut with respect to an OpenFlow switch that can reach the intended legacy switch using a link that is on the new path we want to enforce.

Each seed packet must reach all legacy switches in the magnet subpath that precedes the OpenFlow switch sending the packet. In addition, the seed packet sent by the first OpenFlow switch on the path must reach the source host, to update its ARP table. In Figure 4.4, if C wants to reach D through the same path as B's, Magneto uses $OF6$ to send a seed packet to C to updates its ARP cache with the same magnet MAC address that B uses to reach D. In contrast, if A or B wants to use the default path in the spanning tree, Magneto uses $OF6$ to send a seed packet to A or B to update its ARP cache with the real MAC address of D.

[3]We assume at most 48 interfaces on a switch; for more interfaces, we can change the bit distribution between the OpenFlow ID and the interface ID.

4.5.2 Magnet Routing

Associating magnet MACs with subpaths rather than paths helps reduce the size of forwarding tables. However, because each magnet subpath of a path is installed using different magnet MACs, OpenFlow switches between subpaths must rewrite packet headers.

Given a path to be updated, the source hosts sends packets towards the magnet MAC associated to the first magnet subpath on the path (assuming a seed packet already updated the source's ARP cache). Legacy switches simply forward packets to the next hop according to their forwarding tables. We insert rules in the OpenFlow switches that rewrite each packet's source and destination MAC fields according to the next magnet subpath along the path to be installed. The final OpenFlow switch rewrites the destination MAC field with the destination's real MAC address, as the last magnet subpath does not have its own magnet MAC.

In the example in Figure 4.4, to set up the both the direct and reverse paths between B and D, $OF6$ crafts a seed packet with source MAC address as OF6:2, source hardware address as OF6:2, source protocol address as D's IP, and sends it to B through $LE2$. Also, $OF6$ crafts another seed packet with source MAC address as OF6:3, source hardware address as OF6:3, source protocol address as B's IP, and send it to D through $LE3$. Similarly, $OF7$ crafts one seed packet with magnet MAC OF7:1 to B and another seed packet with magnet MAC OF7:2 to D respectively. A packet sent from B to D starts with source MAC address as B's real MAC address and destination MAC address as OF6:2. When it reaches $OF6$, $OF6$ rewrites its source MAC address to be OF6:3 and destination MAC address to be OF7:1. Later, $OF7$ rewrites the packet header again, whose source MAC address to be OF7:2 and destination MAC address to be D's real MAC address.

4.5.3 Interoperability, Reversibility & Incremental Deployment

We discuss various aspects of deploying Magneto in a real-world enterprise network environment.

Interaction with STP in Magneto does not require additional configuration on legacy switches. Magneto adds a rule in every OpenFlow switch to forward BPDU

messages to the controller, so it can passively listen to all BPDU messages and not forward them further. This behavior guarantees any interface adjacent to an OpenFlow link is not blocked, while a loop-free underlay is still formed among legacy switches.

BUM traffic represents L2 broadcast, unknown unicast, and multicast traffic. The usage of magnet MAC addresses allows Magneto to coexist with broadcast/multicast traffic assuming that such traffic cannot update the hosts' ARP tables such as broadcast ARP messages (which are under the control of Magneto). Unknown unicast traffic (*e.g.*, used by non-IP services) with real destination MAC addresses can reach destinations through the default spanning tree path. On the other hand, unknown unicast traffic with magnet destination MAC addresses will be routed through OpenFlow switch(es) and their Ethernet packet headers will be rewritten.

Inter-VLAN Routing and L3 Routers are used in enterprise networks to isolate traffic and restrict broadcast domains. Magneto works with existing L3 routing by either: (1) utilizing OpenFlow switches on the path between source-destination pairs to rewrite VLAN tags, and therefore it can reduce the traffic latency and the load on the L3 router, or (2) it breaks the path into segments with one segment for each broadcast domain if the policy requires that the traffic go through the L3 router. Then, each segment can be assigned different magnet MAC addresses. Finally, Magneto enables diverse L2 paths, which can be combined with Fibbing [58, 55] (which enables L3 diverse paths) to provide opportunities for joint L2/L3 routing optimization and traffic engineering.

Path diversity depends on the network underlay (*i.e.*, spanning tree), and the location of the OpenFlow switch. Consider the topology in Figure 4.5(a) where the OpenFlow switch is adjacent to four legacy switches. The controllable links change based on the network underlay. For instance, Figure 4.5(b) shows an examples of all controllable links (both spanning tree and OpenFlow links) when the spanning tree is rooted at $LE1$. Figure 4.5(c) shows another examples when the spanning tree is rooted at $LE4$ with less controllable paths. Consequently, the placement of OpenFlow switches during incremental deployment and configuring the STP is critical in enabling many paths in the network that can be controlled by Magneto, which we discuss later in Section 4.6.1.

Site	Source	# Switches	Max/Avg/Min Degree
Large	[41]	1577	65 / 2.15 / 1
Emulated	this work	415	17 / 5.94 / 1
Small	[60]	16	15 / 4.5 / 3

Table 4.2: We evaluate Magneto on three diverse network topologies, two of them from large campus networks and one randomly generated. Figure 4.6 shows the node degree distribution of each topology.

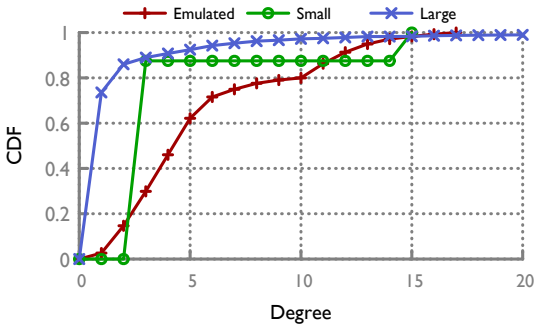
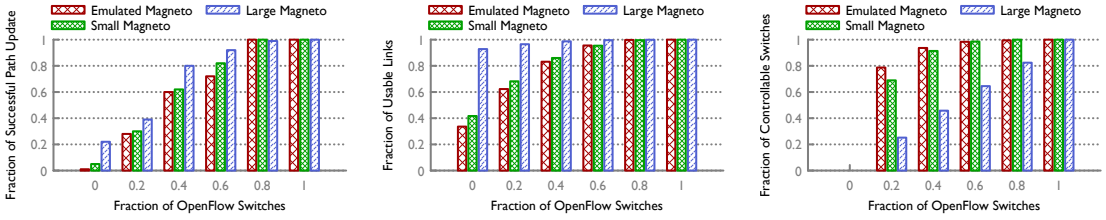


Figure 4.6: Switch degree distribution for the three evaluated network topologies.

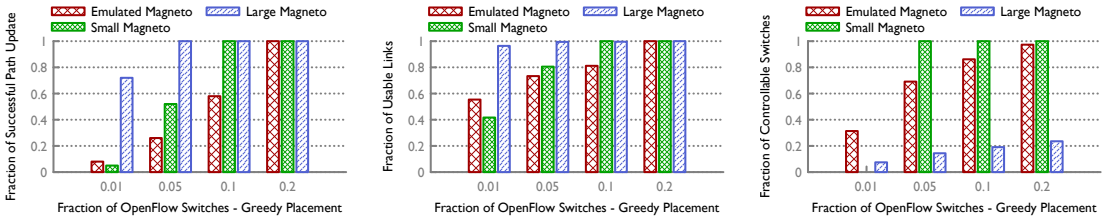
Network failures may affect the functionality of Magneto. Magneto detects data plane failures by monitoring the TCN and root bridge ID fields in STP BPDU messages (for legacy links) or port status messages (for OpenFlow links). Once it identifies a failure, Magneto excludes the failed link from the known topology and recomputes and updates the flow paths affected by the failure. Because the STP failure recovery may change the original spanning tree containing the failed link, the newly updated paths may become unusable once the STP recovery finishes. To avoid frequent path recomputations, Magneto has the option to exclude the entire spanning tree containing the failed link, rather than the link itself, from the known topology before recomputing the affected paths.

If control links fail, the network data plane is still functional, although Magneto may not be able to update paths. However, existing magnet MACs eventually expire and the network reverts to a standard L2 network. We are currently exploring how to make Magneto robust to control network failures [59, 16].



(a) (b) (c)

Figure 4.7: Magneto enables control over a hybrid network with a few OpenFlow switches. We show the path update success in (a), fraction of usable links in (b), and fraction of controllable switches in (c) achieved by Magneto as we upgrade more and more legacy switches to SDN. We assume which switch is updated is a random decision.



(a) (b) (c)

Figure 4.8: When we upgrade the high degree switches first, Magneto achieves control at a fraction of the cost incurred when the upgrade strategy is greedy. Only 20% of OpenFlow switches achieve full routing flexibility.

4.6 Evaluation

We evaluate Magneto from three perspectives. First we show that Magneto provides high path diversity in various hybrid network topologies, with various OpenFlow placement strategies, even when the number of OpenFlow switches is low. Second, we demonstrate that path updates are fast and introduce negligible delay to the data traffic. Finally, we show that the network overhead introduced by Magneto is negligible.

We run Magneto both in simulation and on a small hybrid lab testbed. Our simulations use three topologies: two real-world and one synthetic, randomly generated. Table 5.1 describes the topologies and Figure 4.6 shows the degree distribution of switches in each topology. The "Large" topology represents a large-scale campus network [41] while the "Small" topology is the backbone network of a large campus [60]. To generate

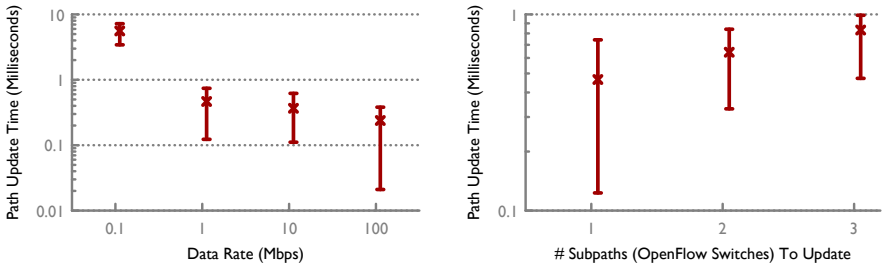


Figure 4.9: Control delay (the time to install a path) of Magneto remains low as we vary the data rate (left) and the number of update subpaths (right) on the path to install.

the "Emulated" topology, we randomly choose the number of switches (between 400 and 600) and the number of links, ensuring the topology is connected. In our experiments, we vary the number and placement strategy of OpenFlow switches in each of these topologies, thus simulating various SDN transition scenarios.

4.6.1 Path Control

The main goal of Magneto is to provide control over the network without the cost of making the network fully programmable and at low management cost. We ask how effective Magneto is in installing paths across various hybrid network topologies. We run Magneto on each of the three topologies described in Table 5.1, and the degree distribution of switches is shown in Figure 4.6. For each run, we randomly select two hosts and compute the five paths with fewest hops between them. We select at random among them a new path to be installed. This makes the simulation realistic since we always install good paths.

The number and location of OpenFlow switches play a key role in the performance of Magneto. We vary the percentage of switches that are OpenFlow and place them in the network using two strategies: random, where random switches are upgraded to OpenFlow, and greedy, where switches are upgraded in decreasing order of their degree.

Random OpenFlow switch placement. We upgrade random legacy switches to OpenFlow switches. We vary the percentage of OpenFlow switches and compute the fraction of successful path updates. Figure 4.7a shows averages over 100 runs. A spanning tree is built as the network underlay when there is no OpenFlow switch

introduced (*i.e.*, the fraction of OpenFlow switches is 0). As expected, as we increase the number of OpenFlow switches the more paths we can install. This is because it is more likely that the feasibility condition in Section 4.4 is satisfied: links on the paths to install are more likely to be adjacent to an OpenFlow switch. Our results show that with as much as 40% of all switches transitioned to OpenFlow, we can install any path with a probability of 0.6. Recall that these paths are among the best five between the pair of end hosts. Other hybrid network controllers, such as Panopticon [46] may achieve a higher success rate but at the cost of increased management complexity due to the need to configure VLANs.

The results above are based on several realistic running scenarios and do not capture the number of total paths we can install. To understand this, we compute the number of links that Magneto can control. A link we cannot control cannot be part of a new path. These are the links that are adjacent to an OpenFlow switch or on the network underlay. Figure 4.7b shows that with less than half of OpenFlow coverage, at least 80% of the links are usable.

Finally, we define the controllable switches as the switches whose forwarding behaviors can be manipulated by Magneto. These are the OpenFlow switches and the legacy switches whose forwarding tables we can modify. Our results in Figure 4.7c show that even when only 20% of the switches are OpenFlow-enabled, Magneto can control as many as 75% total switches. The plots show a discrepancy among the different metrics used to evaluate the "Large" topology. While the path update success and fraction of usable links are high, the fraction of controllable switches is much lower than for the other topologies. This is because there are many switches (more than 70%) with degree 1 in the "Large" topology, as shown in Figure 4.6. These switches provide usable links as part of the spanning tree but are not connected to OpenFlow switches therefore not controllable.

Greedy OpenFlow switch placement. Strategic OpenFlow placement can improve the degree of control offered by Magneto. We propose to upgrade the most influential switches first. We rank the importance of switches according to their degree: the more adjacent links a switch has, the more important it is. Figure 4.8 shows the fractions of successful path updates, usable links, and controllable switches as we vary the percentage of OpenFlow switches. Greedy OpenFlow placement provides a significant

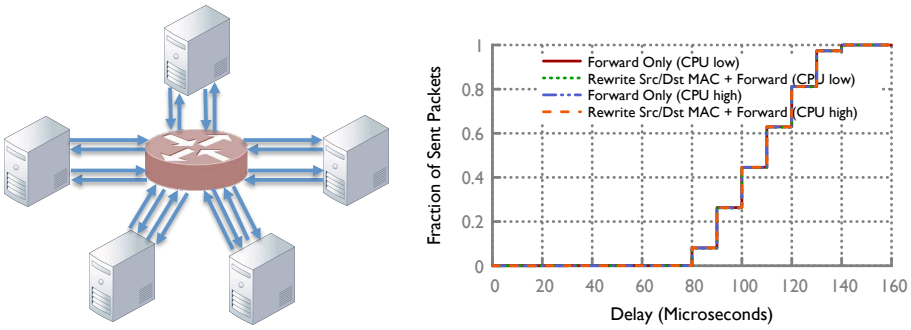


Figure 4.10: Packet header rewriting by OpenFlow switches does not affect the data plane delay. We use one OpenFlow switch and five servers, with each server sending 2 Gbps through the switch and back to itself (left); path installation introduces negligible delay even at high switch CPU loads (right).

boost in efficiency: we can install any path successfully when only 20% of the switches are programmable. Because the most connected switches are OpenFlow-enabled, we do not need to control many legacy switches. As Figure 4.8(c), controlling few legacy switches (less than 10%) is sufficient.

4.6.2 Control Delay

The *control delay* is the time it takes to install a new stable path, *i.e.*, the time between when the controller sends the first seed packet and when the first data packet traverses the new path without the path reverting to the original.

We perform experiments on a real-world testbed in our lab. The testbed consists of eight Dell servers, five Cisco Catalyst legacy switches [61], and two iwNetworks OpenFlow switches [62]. First, we consider a single update subpath and repeatedly vary the data rate on the path to update. Figure 5.9(a) shows that the control delay remains low when we increase the data rate. That the control delay decreases as we increase the data rate is an artifact of our measurement: when the data rate is low, the time between two consecutive packets is higher therefore our measurement error is higher.

Next, we set the data rate at 1 Mbps and increase the number of subpaths that need to be updated. For this, we place one OpenFlow switch on every subpath. Recall that we need to generate and propagate a different magnet MAC for each update subpath.

Figure 5.9(b) shows the results. The control delay is not significantly affected by the number of subpaths, as generating and propagating magnet MACs are independent operations and can be parallelized.

4.6.3 Overhead

We quantify the overhead introduced when running Magneto from two perspectives: impact on applications and impact on the network.

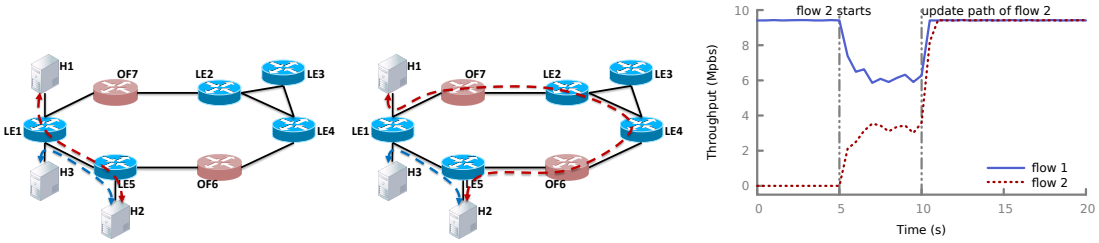
Data delay. The data delay is the additional delay introduced in the application traffic due to packet transformations along the path performed by OpenFlow switches, *i.e.*, rewriting MAC addresses. Recall that, because Magneto uses magnet MACs, OpenFlow switches must rewrite the source/destination MAC address of every packet traversing a newly installed path.

To measure the data delay, we connect five servers to an iwNetworks OpenFlow switch as shown in Figure 4.10 (left). Each server has four 1 Gbps Ethernet interfaces, and we use two interfaces as senders and the other two as receivers. Each server generates 2 Gbps traffic traversing the OpenFlow switch, together all servers generate traffic at 10Gbps (or 15 million packets per sec). Each server sends traffic that returns back to itself. To measure accurate one-way delay, we use PF_RING [63]. We modified the *pfsend* and *pfcount* codes to timestamp every packet before it is sent out and compute its one-way delay when it is received.

Figure 4.10 (right) shows the delay incurred when rewriting the Ethernet header of each packet and when simply forwarding the packet both under low and high (99%) CPU load. Rewriting packet headers introduces negligible data plane delay even at high CPU load. This matches the findings of an earlier work on application-aware data processing in SDN [64].

CPU and memory overhead. Injecting seed packets from OpenFlow switches could increase the CPU and memory overhead on both legacy switches and OpenFlow switches. We measure the CPU utilization and memory usage on our Cisco legacy switches and iwNetworks OpenFlow switches, when Magneto controller injects control packets with magnet MAC addresses.

Our results in Table 5.4 prove that Magneto introduces very little CPU and memory



(a) *flow 1* and *flow 2* use the same path and compete for its bandwidth.

(b) Magneto changes *flow 2* to take another route to alleviate congestion.

(c) Flow throughput over time.

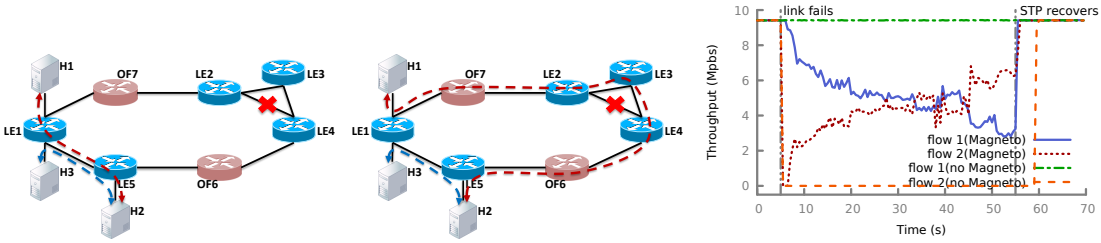
Figure 4.11: Magneto alleviates congestion by reconfiguring flows traversing both legacy and OpenFlow switches. *flow 1* and *flow 2* start on the same path and compete for its bandwidth. As soon as Magneto updates the path of *flow 2*, both flows can use all available bandwidth.

Number of magnet MACs	**CPU (iwNetworks)**	**CPU (Cisco)**	**Mem (iwNetworks)**	**Mem (Cisco)**
1,000	4.80%	1.75%	16 KB	8 KB
5,000	6.09%	2.46%	55 KB	35 KB
10,000	7.36%	2.89%	146 KB	78 KB

Table 4.3: CPU and memory load introduced by Magneto on OpenFlow and legacy switches when the number of magnet MACs varies.

overhead on both legacy and OpenFlow switches. *Address Learning* in Cisco switches *ofswd* and *ofprotocol* in OpenFlow switches are the main processes affected by the sending of seed packets. Even with a large number of magnet MAC addresses (10,000), the total memory overhead increase was only 78 KB on Cisco switch and 146 KB for iwNetworks switch, a small fraction of the total memory available. We collected the CPU utilization on the switches every minute immediately after we started injecting seed packets. The utilization was systematically low, at most 7.36% for iwNetworks switch and 2.89% for Cisco switch.

Control traffic. Magneto introduces little control traffic into the network. In the worst case, the number of seed packets needed to update a path must be twice the number of subpaths. Because forwarding entries in legacy switches expire, Magneto must repeatedly re-inject the same seed packet. Given a standard timeout of five minutes, the additional network overhead is still negligible.



(a) Magneto updates *flow 2* to take the route used before to restore end-to-end connectivity.

(b) Magneto changes the route of *flow 2* after STP recovers.

(c) Flow throughput over time.

Figure 4.12: In face of the link failure on $(LE2, LE4)$, Magneto switches *flow 2* to the original path $(LE1, LE5)$ to rapidly restore connectivity instead of waiting for STP to recover. After STP converges, Magneto updates the path of *flow 2* again to achieve maximum throughput.

4.7 Case Study: Better Routing and Failure Recovery with Magneto

We show how Magneto improves network performance by exploiting routing diversity and reacting quickly to network failures. We deploy a hybrid testbed consisting of three servers, five Cisco switches and two iwNetworks OpenFlow switches (Figure 4.11a). STP runs on the Cisco switches.

Flexible routing. To underline Magneto's ability to find alternate paths quickly, we start two flows, from $H3$ to $H2$ (*flow 1*) and from $H1$ to $H2$ (*flow 2*). Both flows share the link $(LE1, LE5)$ initially, whose capacity we artificially set to 10Mbps. *Flow 1* starts five seconds before *flow 2* (Figure 4.11a). As soon as *flow 2* starts, it will compete with *flow 1* for the entire capacity on the default path. Neither of the flows can benefit from the entire capacity. After 10 seconds, we use Magneto to update the default path of *flow 2* to $(LE1, OF7, LE2, LE4, OF6, LE5)$. As soon as the update finishes, both flows can run at full rate as they do not compete with each other. Figure 4.11c shows the rate of each flow during the experiment.

Quick failure recovery. We now demonstrate how Magneto can recover from network failures. Consider the end of the previous experiment where *flow 1* and *flow 2* take non-overlapping paths to their destination. After five seconds, the link $(LE2, LE4)$

fails. As STP forms a loop-free underlay among connected legacy switches, no alternative path is available for *flow 2* until STP recovery finishes. On the other hand, Magneto can adapt immediately by detecting the propagated STP BPDU frames and re-routing *flow 2* on its original path $(LE1, LE5)$. Although *flow 2* competes once again with *flow 1* for the capacity on $(LE1, LE5)$, the end-to-end connectivity is restored. Magneto detects when STP finishes recovery by sending probes between $OF6$ and $OF7$ every second. Once the probe is received on the other end, Magneto knows STP finishes recovery and redirects *flow 2* to the path $(LE1, OF7, LE2, LE3, LE4, OF6, LE5)$. Relying solely on on STP to recover from the failure disconnects *flow 2* during the STP recovery process, whereas with Magneto, end-to-end connectivity is preserved.

4.8 Summary

We present Magneto, a network controller that enables unified, fine-grained routing control in hybrid networks. Magneto uses OpenFlow's ability to send custom-made packets into the data plane to manipulate legacy switches into updating forwarding entries for specific MAC addresses. Via magnet addresses, Magneto gains visibility to the network and allows access control for IP-based applications and services in a hybrid network. Our evaluation on a lab testbed and simulations on large enterprise network topologies show that Magneto is able to achieve full control over routing when only 20% of network switches are programmable and with negligible computation and latency overhead. Magneto also poses a number of new research questions such as the strategic placement and number of SDN switches as well as magnet addresses needed to exert SDN-like control over legacy networks and to what extent such control can be exercised.

Chapter 5

Gaining Fine-Grained Network Visibility for On-Demand Monitoring and Better Policy Enforcement

5.1 Introduction

Real-time monitoring of network flows is critical to preserve enterprise network health and detect problems, such as abnormal bandwidth usage [65, 66], inflated paths [67], QoS violations [68] or security threats [69]. To identify and quickly react to such issues, operators require network-wide visibility, *i.e.*, the ability to monitor *any* flow at *any* time.

Traditionally, to achieve network-wide visibility, operators follow the *routing-then-monitoring* approach: deploy monitoring tools on the data plane, such that they cover all flows' paths. Indeed, most switches and routers today support NetFlow or similar monitoring protocols [24, 23]; intrusion detection systems are inserted at network ingress points to inspect all external flows [70]. Such on-path monitoring requires strategic, if not exhaustive, deployment and fine tuning to avoid overloading the data plane [25]. Offloading the monitoring tasks to specialized off-path appliances by mirroring packets,

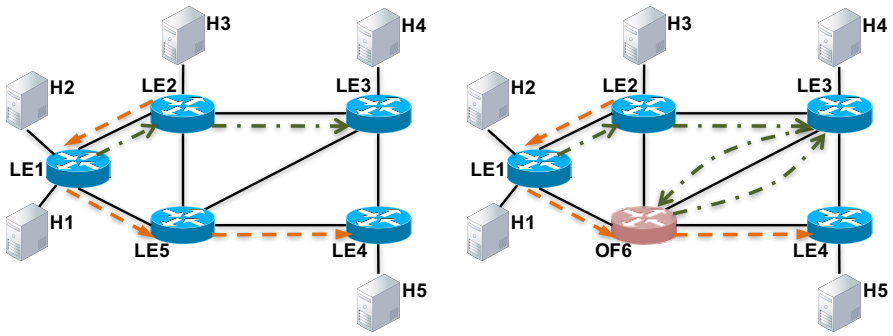


Figure 5.1: Flow visibility in legacy (left) and hybrid (right) networks. Legacy switches are shown in blue, and OpenFlow switches are shown in red. In this example, the network policy is updated from an old one (*i.e.*, $H1 \rightarrow H4$, $H2 \rightarrow H4\&H5$, $H3 \rightarrow H5$) to a new one (*i.e.*, $H1 \rightarrow H4$, $H2 \rightarrow H4$, $H3 \rightarrow H4\&H5$). The green arrow indicates the path to reach $H4$ and the orange arrow indicated the path to reach $H5$. In order to verify this network policy update, operators need to deploy monitoring software (*e.g.*, sFlow) on $LE3$ and $LE4$ in legacy networks. In hybrid networks, all the flows can be visible on $OF6$.

e.g., using SPAN [71] or TAP [72], may relieve the load on the data plane, but requires careful coordination to avoid oversubscribing the mirroring ports or paths and may not be amenable to real-time analysis.

With the goal of making monitoring more flexible and efficient, several efforts promote a combined *routing-and-monitoring* approach to network visibility: deploy monitoring tools at select locations in the data plane and set up flow paths to traverse these locations. This approach is enabled by software-defined networking (SDN), which allows operators to program the data plane remotely. SDN removes the rigidity of traditional monitoring and allows the flexibility to install forwarding entries that meet both monitoring and routing goals [73, 74]. In addition, SDN-enabled switches provide yet another monitoring device, by supporting counting [75] and inspecting [76] packets, or through custom monitoring scripts [65]. Unfortunately, SDN-based monitoring requires a significant upfront investment in deploying or upgrading to SDN-enabled switches [42]. Most enterprises are reluctant to invest in SDN without a clear understanding of its benefits and disadvantages.

We propose *clairvoyant networks* to enable both low-cost and flexible network-wide flow monitoring. Clairvoyant networks are *partially programmable* networks that offer

full control over all paths. Any enterprise network can become clairvoyant by adding at least one SDN-enabled[1] switch and a specialized network controller. In this way, one can reap the benefits of SDN-based monitoring at a fraction of the deployment cost.

Clairvoyant networks offer SDN-based visibility: they may modify the paths of flows to redirect them through SDN switches and expose them to SDN-based monitoring techniques [65, 77, 78, 79]. To do this, the Magneto controller incorporates two mechanisms, telekinesis and magnet MACs, introduced in a previous paper [6], that can update the forwarding tables of legacy switches from an SDN switch.

Modifying the path of a flow to make it visible is an intrusive policy, which may not be acceptable for some enterprises, either due to privacy or performance concerns. To make the case that clairvoyant networks can provide significant advantages to SDN-based monitoring, we perform a measurement study on their benefits and costs. We first study the degree of visibility that clairvoyant networks offer (Section 5.3). Using real-world and synthetic topologies, we show that even a single OpenFlow switch enables monitoring of any flow with various possible paths to choose from.

We study the performance cost of enabling network-wide visibility (Section 5.4), by answering the question of how much the performance of the flows and the network suffers in exchange for visibility. The cost of enabling network-wide visibility is high when few OpenFlow switches are deployed (paths may be as much as twice longer than default) but decreases as we add more OpenFlow switches. Thus, clairvoyant networks give operators a trade-off between the upfront cost to enable SDN-based monitoring and the performance penalty incurred by enabling such monitoring.

We target clairvoyant networks for SDN-based monitoring carried through SDN switches. However, it is possible to redirect some flows through legacy monitoring devices, such as NetFlow-enabled switches [24] or deep packet inspection appliances, as long as the monitoring device lies on the path between the source or destination of the flow and an SDN switch. This helps control the trade-off between upfront investment and network overhead even further. When operators prefer to use traditional monitoring devices, only up to 10% of all legacy devices need to support monitoring to cover all flows, as opposed to all in current practices.

[1]We interchangeably use the terms SDN(-enabled), OpenFlow(-enabled), or programmable to refer to devices whose forwarding tables can be configured remotely from a centralized controller.

In the second part of this chapter, inspired by our measurement results, we present a basic design for clairvoyant networks. We show how to integrate the existing mechanisms of telekinesis and magnet MACs with the visibility tasks to design the Magneto controller (Section 5.5). With a goal to inform network architects and operators on the trade-offs of adopting a clairvoyant network, we identify specific key performance and cost indicators. We then provide a customized design, including a balanced SDN deployment strategy and a flow scheduling mechanism, that reduces both the upfront deployment cost and the flow and network overhead to offer a practical solution for deploying multiple visibility tasks at the same time (Section 5.6).

Clairvoyant networks provide a low-cost flexible monitoring substrate for enterprises where changing the path of flows is an acceptable policy. They can open up new directions in flow monitoring by allowing *hybrid monitoring applications* that take advantage of the monitoring capabilities of both SDN and legacy devices to build accurate, flexible, and efficient monitoring.

5.2 Clairvoyant networks

We discuss related research on SDN-based monitoring and introduce the concept of clairvoyant networks which provide low-cost, flexible, network-wide monitoring to operators.

5.2.1 SDN-based monitoring

A network flow is *visible* when its path traverses a monitoring device, such as an NetFlow-enabled switch, a polling-enabled SDN switch, or any dedicated monitoring or packet capture appliance. Network-wide visibility of all flows is important for many network management applications such as traffic engineering, access control, anomaly detection, or heavy hitter detection [80, 81, 82, 83, 84, 85, 86].

Traditional flow monitoring achieves visibility by defining static monitoring tasks that require switch support [24, 23] or dedicated monitoring appliances [72, 87]. For example, to identify large flows, NetFlow-enabled switches sample packets and build flow-level packet counters. Monitoring tools must be strategically deployed across the data plane to enable network-wide visibility, and carefully tuned to avoid overloading

the data plane [25].

SDN disrupts traditional monitoring practices by providing better control and visibility over the network. First, SDN allows operators to remotely update switch forwarding entries on demand, enabling more flexible and dynamic monitoring tasks [65, 88, 89, 90]. Second, SDN-enabled switches double as monitoring devices. They support flow-based counters to monitor utilization [91, 92, 77, 93] or help inspect traffic to detect unauthorized access [73, 74] or security threats [76].

An important impediment to SDN-based monitoring has been the significant upfront investment cost it requires. Upgrading the network to SDN is prohibitive for most enterprises as it requires replacing most, if not all, legacy switches with SDN-enabled switches [42]. Recent work proposes hybrid SDN and legacy (or partially programmable) networks to lower the deployment cost of SDN while providing most of its benefits. However, with hybrid networks, operators have visibility only over the flows that traverse the SDN switches and cannot monitor the traffic in the legacy part [43, 45, 46, 42].

5.2.2 Use Cases

Dynamic flow monitoring enables fine-grained on-demand network visibility. It is desirable to have programmable network visibility, if we can program *what to see, where to see, and how to see*. Such on-demand visibility provides flexible monitoring capabilities for enterprise networks, given that not all flows need to be monitored all the time with the same priority. For example, monitoring flows to critical servers is typically prioritized over monitoring the traffic generated from a student's laptop.

Network policy verification. Network policies change overtime. When new network policies get deployed, operators need to verify they are correctly functioning by seeing (no) traffic from the affected source hosts (or, to the destination hosts). Dynamic flow monitoring makes it possible for operators to verify the updated network policy has been enforced successfully at any time anywhere. If any misconfiguration found during the verification, operators can fix the policies based on the flow record and re-check until the new policies are correctly deployed. One example is shown in Figure 5.1, where the updated policy aims to block the traffic from H2 to H5, and allow the traffic from H3 to H4. Once operators start to deploy the new network policy, they steer the traffic destined to H4 to OF6 and further check whether being able to see traffic from

H3 to H4. If yes, the new policy for H3 is successfully updated. Similarly, operators query statistics from OF6 to see whether traffic exists from H2 to H5. Under correct configuration, no packet shall be seen from H2 to H5.

Flow performance monitoring. Knowing performance of flows is basis of some routine network management tasks, such as traffic engineering and troubleshooting. Dynamic flow monitoring makes it possible for operators to select which flows to monitor in real-time. Following the same example in Figure 5.1, after the new policy mentioned above was deployed, operators noticed that the link (LE2, LE3) got congested and complaint from H1 and H2's users for slow network. To help further diagnose, operators want to know fine-grained performance information (*e.g.*, throughput) for each flow to H4, *i.e.*, (H1, H4), (H2, H4), and (H3, H4). By instructing their traffic to go through OF6, operators can easily see which each flow's performance and alleviate the congestion by either rate limiting or rerouting certain flows.

5.2.3 Proposed idea

In line with previous research [91, 92, 77, 93, 76], we consider a flow to be visible[2] when it traverses an SDN switch. We propose to make *all* flows visible in a hybrid network by redirecting them (temporarily) through an SDN switch. In this way, operators could apply existing SDN-based monitoring mechanisms to monitor all flows, including those whose default path does not traverse an SDN switch. When monitoring is finished, the flows would be reverted to their original path. This would dramatically decrease the cost of deploying and using SDN-based monitoring, as a wholesale [42] upgrade to SDN is not necessary to enable network-wide visibility.

Towards this goal, we introduce *clairvoyant networks*: partially programmable networks that offer operators the ability to monitor any flow any time. Any enterprise network can become clairvoyant by deploying at least one SDN-enabled switch and a specialized controller, which we call the Magneto controller. Clairvoyant networks are made possible by previous work [6] on using SDN switches to control routing through legacy devices. As we describe in detail in Section 5.5, we can change the path of any

[2]Throughout the chapter, a flow is "visible" when it traverses an SDN switch and "invisible" otherwise. In Section 5.3, we discuss how to make a flow visible to legacy monitoring devices rather than SDN switches.

flow traversing the legacy network using simple OpenFlow-based mechanisms.

Clairvoyant networks raise several questions about the feasibility and cost of flow monitoring by changing the path of flows. First, how many flows can we make visible by updating their paths compared to a simple hybrid networks? While clairvoyant networks focus on SDN-based monitoring (*i.e.*, a flow is visible when it traverses an SDN switch), is it possible to redirect flows through traditional monitoring devices (*e.g.*, NetFlow-enabled switches). Finally, what are the cost and performance trade-offs involved in changing the path of a flow to make it visible? We explore these questions through data-driven simulations and real-world deployments in Sections 5.3 and 5.4, then present a basic design for clairvoyant networks in Section 5.5.

5.3 Flow visibility

Do clairvoyant networks make more flows visible than simple hybrid SDN networks that have no ability to update legacy paths? In this section, we investigate the extent to which clairvoyant networks provide visibility both through SDN switches and using legacy monitoring devices.

We evaluate the feasibility of clairvoyant networks by investigating three questions:

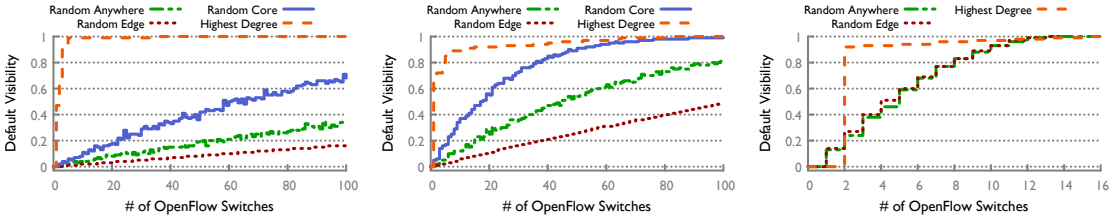
1. what is the degree of visibility that we can introduce compared to a regular network? Recall that a flow is visible if it traverses a monitoring device.

2. what is the performance penalty necessary to make flows visible? Making a flow visible requires changing its path to traverse a monitoring device.

3. what are the side-effects on the network or other flows?

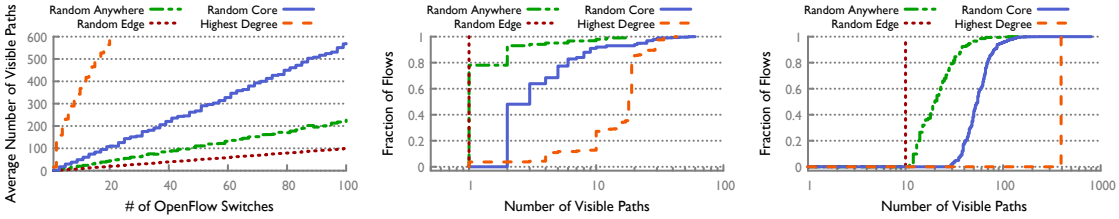
5.3.1 Methodology

Network topologies. We evaluate the feasibility of clairvoyant networks on three network topologies, described in Table 5.1. The "Large" and "Small" are the real topologies of a large-scale campus network [41] and of a smaller campus backbone network [60]. We generate the "Medium" topology to model a medium-size enterprise network. In doing so, we try to preserve the features observed in the real "Large" topology: more edge



(a) "Large" topology (b) "Medium" topology (c) "Small" topology

Figure 5.2: Default visibility, as we vary the number and placement of OpenFlow switches.



(a) (b) (c)

Figure 5.3: (a) The average number of possible visible paths for flows whose default paths are not naturally visible, for the "Large" topology; we cut the line for highest-degree at 20 OpenFlow switches, when the default visibility becomes 1. The distribution for the number of visible paths for each flow, when we use (b) one OpenFlow switch, or (c) ten OpenFlow switches.

switches than core switches, and multiple components connected through high-degree core switches.

Deployment. We consider four placement strategies for SDN-enabled switches: random anywhere, random edge, random core, and highest-degree. Random strategies select a legacy switch at random and replace it with an OpenFlow switch. Random anywhere and random core provide base cases for comparison, while random edge is intended to model a scenario where operators deploy software switches on edge hypervisors or servers. The highest-degree strategy replaces legacy switches in decreasing order of their degree and reflects a best case scenario where the most influential switches are upgraded first.

Network flows. We consider all flows that could be installed in the network, *i.e.*, between all pairs of edge switches. We do not take into account the popularity of a pair

Name	Source	# Switches/Edge/Core	Max/Avg/Min Degree
Large	[41]	1577 / 1160 / 417	65 / 2.15 / 1
Medium	this work	493 / 355 / 138	19 / 3.11 / 1
Small	[60]	16 / 14 / 2	15 / 4.5 / 3

Table 5.1: We use two real-world ("Large" and "Small") and one synthetic ("Medium") network topologies to demonstrate the feasibility of clairvoyant networks.

of switches (*e.g.*, some edge switches connect to more hosts) because it does not affect the visibility of a flow. We assume a flow is between two different IP addresses, without taking into account port numbers, to match the granularity provided by the path update mechanism [6]. Unless otherwise noted, every experiment provides aggregated values over 100 runs, resetting the switch placement after each run.

Visibility. We define the *visibility* of a network as the probability that a random flow in the network is visible, *i.e.*, traversing a monitoring device. The visibility of a network takes values between 0 and 1. All flows in a network with visibility 1 can be monitored. For example, a network where all switches and routers support NetFlow or where all switches are SDN-enabled has visibility 1. We further classify visibility according to the type of device that provides it. *Natural visibility* (or simply visibility) represents the visibility achieved from monitoring flows at SDN-enabled switches, while *supervisibility* characterizes a network where flows are monitored at legacy monitoring devices such as NetFlow-enabled routers or IDSes. We measure both the natural and supervisibility that a clairvoyant network provides while varying both the number of OpenFlow switches and their placement strategy.

5.3.2 Natural visibility

Natural visibility describes the ability of a clairvoyant network to make any flow visible by routing it through an SDN-enabled switch. As the controller can set up any path through an OpenFlow switch, *the natural visibility of any clairvoyant network is 1.*

However, part of the natural visibility may not even require setup from the controller: if the flow's default path traverses an OpenFlow switch, then it is not necessary to use the Magneto controller to make it visible. To understand the benefit that clairvoyant networks provide, we must evaluate how much of their natural visibility is achieved using the Magneto controller. For this, we compute the *default visibility*: the probability that

any flow is visible initially on its default path.

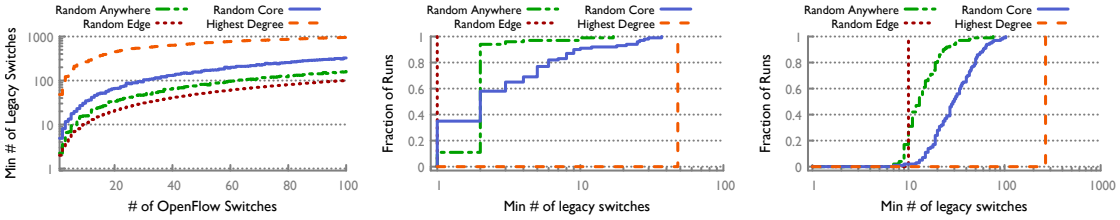
Figure 5.2 and Table 5.3 show the default visibility of each network, as we vary the number and placement of OpenFlow switches. When replacing more switches, more flows are likely to be visible initially, without having their paths updated. The highest-degree placement performs best. This is because high-degree nodes partition the network in many separate connected components. Most flows are likely to be between components and therefore must traverse a high-degree node. This result implies that when upgrading the top highest degree legacy switches to SDN, most flows are visible by default. However, upgrading the high degree switches is also costlier as they would need to support more flows and higher throughput.

Although the ability to set up a flow's path through an OpenFlow switch is important, the number of possible paths for a flow is equally critical. Path diversity offers operators more flexibility in reaching both monitoring and routing goals in path setup. Figure 5.3a shows the average number of visible paths that exist for flows whose default paths are not naturally visible, *i.e.*, do not traverse an SDN-enabled switch, in the "Large" topology (Table 5.3 shows results for all topologies). Replacing the high-degree switches first increases path diversity and enables more flexible monitoring. Figures 5.3b and 5.3c zoom in and show the distribution of the number of visible paths for each flow when we have one and ten OpenFlow switches. Path diversity is significant, regardless of the switch placement strategy. We assess the performance of these paths in Section 5.4.

Summary: Clairvoyant networks offer full visibility and provide ample path diversity to set up flow paths. Operators should consider upgrading the high degree switches to SDN to gain more default visibility.

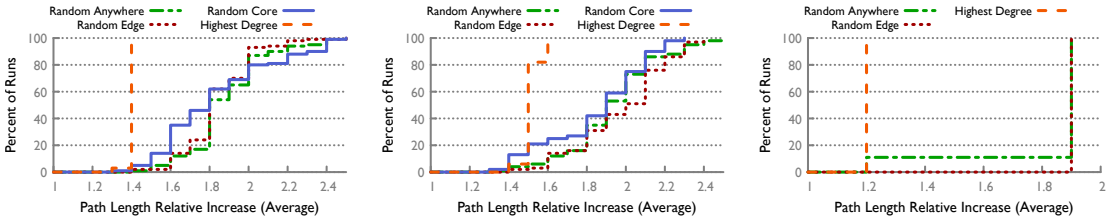
5.3.3 Supervisibility

When upgrading to a clairvoyant network, only a few legacy switches may be replaced with OpenFlow switches. Although OpenFlow switches provide monitoring capabilities [65, 76], being able to use traditional monitoring devices, such as NetFlow-enabled legacy switches or intrusion detection systems, may alleviate some of the monitoring load on OpenFlow switches. While all flows can be set up through a specific OpenFlow switch, not all flows can be set up through a particular legacy device. In fact, flow paths can be set up through a legacy device only if the device is on a path between an



(a) (b) (c)

Figure 5.4: (a) The minimum number of legacy monitoring devices needed to achieve full supervisibility (*i.e.*, all flows traverse at least one legacy monitoring device) for the "Large" network. The distribution of the minimum number of legacy devices to achieve full supervisibility for when we use (b) one OpenFlow switch, or (c) ten OpenFlow switches.



(a) "Large" topology (b) "Medium" topology (c) "Small" topology

Figure 5.5: The average flow stretch increase for the top five shortest visible paths when we have one OpenFlow switch.

OpenFlow switch and the source or destination of a flow. The supervisibility reflects the ability of a clairvoyant network to set up paths through legacy devices.

We compute the minimum number of legacy monitoring switches necessary to achieve network-wide supervisibility, *i.e.*, any flow's path would traverse at least one of these legacy switches. Figure 5.4a presents the results for the "Large" topology. Interestingly, the highest-degree strategy performs poorly compared to the other strategies: more monitoring-enabled legacy switches are needed to cover all flows and achieve a supervisibility of 1. This is because there are more paths through high degree switches and we need more legacy monitoring devices to cover all of them.

Figures 5.4b and 5.4c offer a closer look at achieving full supervisibility with one and ten OpenFlow switches. One interesting finding is that when we place OpenFlow switches at edge, the minimum number of legacy switches needed to cover all flows

is lowest and the same as the number of OpenFlow switches. The reason is any SDN switch can redirect all the flows to go through itself and then through one of its adjacent legacy switches. Of course, placing few switches at the edge may increase the path length unnecessarily.

To maximize the number of visible flows it sees, a monitoring-enabled legacy switch should be located as close to an OpenFlow switch as possible. We confirm that all legacy switches in the experiments from Figures 5.4b and 5.4c are indeed adjacent to OpenFlow switches. This observation also defines an upper bound on how many monitoring-enabled legacy switches we need to cover all flows: the total number of active interfaces on all OpenFlow switches.

Summary: Clairvoyant networks offer full supervisibility through few monitoring-enabled legacy devices, bounded only by the number of active interfaces on all SDN switches. Unlike for natural visibility, the high-degree placement performs poorly. Operators should consider the other strategies to gain supervisibility with few legacy monitoring devices.

5.4 The cost of visibility

Setting up flow paths through monitoring devices may introduce performance penalties to flows and overhead in the network. While monitoring applications may have their own overhead, here we focus on several key cost indicators related the effect of updating the path of a flow and whose value depends little, or not at all, on how flows are monitored.

5.4.1 Overhead on flows

How does visibility affect the performance of a flow? We consider only natural visibility. As we saw in the previous section, supervisibility is closely tied to natural visibility and flow paths are likely to be similar. We evaluate two flow performance metrics. The *flow stretch* represents the relative increase of the number of hops in the new flow path compared to the default path. It reflects the penalty in end-to-end latency that a flow would pay for becoming visible. The *flow stress* captures the maximum number of other distinct flows with which a flow shares any link. Flow stress models the change in throughput that a flow may see when it becomes visible, and

		Random anywhere			Random edge			Random core		
	# OF switches	1	5	20	1	5	20	1	5	20
	Default visibility	0.0	0.02	0.08	0.0	0.01	0.03	0.01	0.06	0.18
	Possible paths	1.4	10.9	44.4	1.0	5.0	20.0	2.9	29.6	110.4
	Supervisibility	2.24	9.47	33.77	1.0	5.0	20.0	4.86	17.72	66.33
L	**Flow stretch**	1.9	1.4	1.3	1.8	1.5	1.4	1.7	1.4	1.2
	Flow stress increase	21.2	7.6	3.7	21.3	7.5	3.2	20.9	7.7	3.2
	Network stress increase	4.4	2.3	1.5	4.4	2.1	1.5	4.3	2.2	1.3
	# OF switches	1	5	20	1	5	20	1	5	20
	Default visibility	0.01	0.06	0.25	0.01	0.03	0.11	0.05	0.17	0.55
	Possible paths	3.2	16.3	60.6	1.0	5.0	20.0	8.2	42.6	173.2
	Supervisibility	2.54	8.18	29.37	2.0	6.0	21.0	3.72	13.62	56.19
M	**Flow stretch**	1.9	1.4	1.2	2.0	1.5	1.3	1.7	1.3	1.1
	Flow stress increase	10.1	3.6	1.6	10.5	4.0	1.6	9.3	2.7	1.4
	Network stress increase	3.6	1.8	1.2	3.7	1.9	1.1	3.3	1.5	1.0
	# OF switches	1	5	10	1	5	10	1	-	-
	Default visibility	0.13	0.59	0.93	0.14	0.6	0.93	0.0	-	-
	Possible paths	6.7	31.8	41.7	3.3	15.9	30.5	34.1	-	-
	Supervisibility	1.89	5.9	11.2	2.0	6.0	11.0	1.0	-	-
S	**Flow stretch**	1.5	1.1	1.0	1.9	1.3	1.3	1.2	-	-
	Flow stress increase	5.8	1.5	2.2	6.7	2.0	1.5	0.9	-	-
	Network stress increase	3.2	1.0	1.0	3.5	1.4	1.2	0.5	-	-

Table 5.2: Results for visibility and cost metrics for the three topologies. We show the default visibility, the average number of visible paths for an invisible flow, the minimum number of monitoring-enabled legacy switches to achieve full supervisibility, the average flow stretch, and the relative increase in flow and network stress between a flow's default and visible paths. For highest-degree strategy, we only present results when we have one OpenFlow switch, since the default visibility increases significantly with a few more OpenFlow switches (*i.e.*, higher than 0.85 with five OpenFlow switches). In the small topology, both the core switches are the highest-degree switches, so their results are the same.

captures the ability of clairvoyant networks to offer monitoring paths that are lightly loaded.

We compute the average flow stretch of the top five shortest visible paths for each flow for all runs. Figure 5.5 shows the detailed results for when we have a single OpenFlow switch; Table 5.3 shows statistics for more switches. As expected, placing OpenFlow switches at the edge has the largest performance penalty, since a visible path may need to stretch to the other side of the network. The results show that with only 2% of switches upgraded to OpenFlow, the average visible path is only 1.3 times greater than the default path. This means that, even given the choice between several paths, a

		High-degree	Every-edge
L	**# OF switches**	1	25
	Default visibility	0.48	0
	Possible paths	16.6	2320+
	Supervisibility	48.0	25
	Flow stretch	1.4	1.2
	Flow stress increase	7.7	1.0
	Network stress increase	2.1	1.0
M	**# OF switches**	1	8
	Default visibility	0.71	0
	Possible paths	19.7	710+
	Supervisibility	6.0	8
	Flow stretch	1.5	1.2
	Flow stress increase	2.7	1.0
	Network stress increase	1.1	1.0
S	**# OF switches**	1	1
	Default visibility	0.0	0
	Possible paths	34.1	28+
	Supervisibility	1.0	1
	Flow stretch	1.2	1.5
	Flow stress increase	0.9	1.1
	Network stress increase	0.5	1.0

Table 5.3: Results for visibility and cost metrics for the three topologies. We show the default visibility, the average number of visible paths for an invisible flow, the minimum number of monitoring-enabled legacy switches to achieve full supervisibility, the average flow stretch, and the relative increase in flow and network stress between a flow's default and visible paths. For highest-degree strategy, we only present results when we have one OpenFlow switch, since the default visibility increases significantly with a few more OpenFlow switches (*i.e.*, higher than 0.85 with five OpenFlow switches). In the small topology, both the core switches are the highest-degree switches, so their results are the same.

monitoring application would still likely select a fairly short visible path for a flow that is not visible by default.

Table 5.3 presents the average relative flow stress increase when making a flow visible. As expected, as we add more OpenFlow switches, and thus enable more paths, the flow stress change is smaller. With only 20 OpenFlow switches, a a flow is likely to increase its stress as much as 3.7 times in exchange for being monitored.

Summary: Making flows visible has little effect on the number of hops they traverse. However, it has a significant effect on flow stress as it forces multiple flows through few monitoring-enabled switches. Increasing the number of such switches helps spread the

load more evenly.

5.4.2 Overhead on the network

Making flows visible requires changing their paths which in turn may pose an additional burden on some network links and switches.

How does visibility affect the network links? We define the *network stress* as the maximum number of flows that traverse any link in the network. Table 5.3 shows the relative increase in network stress across various placement strategies. High-degree strategies do not add much to the network stress when making flows visible, while the other strategies require more OpenFlow switches to keep the network stress low.

How does visibility affect the network switches? The OpenFlow switches may see an increased overhead in clairvoyant networks, when compared to simple SDN networks, as they are queried more frequently by the controller or mirror traffic for further analysis. We consider three metrics for the cost imposed on switches in clairvoyant networks—memory usage, CPU utilization, and number of forwarding entries—and study each metric as we increase the number of flows made visible.

First, we measure the CPU utilization and memory usage on an iwNetworks OpenFlow switch in two scenarios: when the Magneto controller polls the flow statistics every second and when the switch mirrors traffic (*e.g.*, to the Magneto controller or a dedicate server). Previous research [94] shows that the performance of OpenFlow switches decreases as the controller polls for statistics. Mirroring packets to the controller, on the other hand, packs the captured packets as the payload of PacketIn messages [17], which is done by the switch's CPU. Though it is also possible to send packets to the controller as the same to send packets to any destination—output to a specific port by

Number of flows	**CPU (Query)**	**CPU (Mirror)**	**Mem (Query)**	**Mem (Mirror)**
1	0.05 %	2.26 %	0.22 KB	0.33 KB
10	0.15 %	2.37 %	0.22 KB	0.51 KB
100	1.05 %	5.31 %	0.22 KB	2.12 KB

Table 5.4: CPU and memory load increase on an OpenFlow switch when the number of flows varies, under two scenarios: when the controller polls the switch for statistics every second and when the switch mirrors packets to the controller.

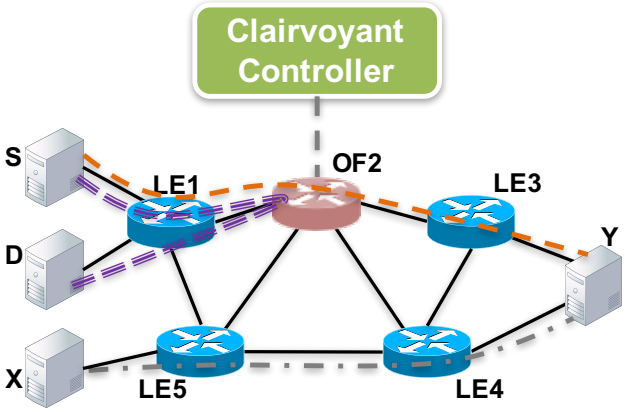


Figure 5.6: Clairvoyant networks require as few as one SDN-enabled switch. The Magneto controller can make the flow (S, D) visible to switch $OF2$ by setting up the path $S - LE1 - OF2 - LE1 - D$ and the flow (S, Y) visible to $OF2$ by installing the path $S - LE1 - OF2 - LE3 - Y$. (X, Y) is an invisible flow.

the switch hardware, we do not study this approach in this work since it does not involve the switch's CPU. Table 5.4 shows the results as we increase the number of concurrent flows. Clairvoyant networks add little overhead to the SDN switches even with many flows being monitored at the same time.

The number of forwarding entries required by making flows visible may impact the performance of switches. For legacy switches, a single additional entry is sufficient to forward the monitored flows to an OpenFlow switch. The number of forwarding rules in one SDN switch is bounded by the number of simultaneous flows this switch handles, since OpenFlow switch needs to rewrite source and destination MAC addresses for every monitored flow. How to further compress the forwarding rules in SDN switches is out of scope of this work [95, 96, 97].

5.5 Design

In this section, we present a basic design for clairvoyant networks. As mentioned earlier, any enterprise network can become clairvoyant by deploying at least one SDN-enabled switch and a specialized controller—which we call the clairvoyant controller.

The controller consists of two layers: path update and visibility enabler. It receives

visibility tasks from operators specifying what flows to monitor and, if necessary, updates the paths of the flows to make them visible. For this, it implements Magneto, a mechanism, first introduced in a previous paper [6] and summarized below, that can change the path of any flow, even when the flow does not traverse an SDN switch. The visibility enabling layer reads and schedules enable visibility tasks. How to monitor a flow, *i.e.*, polling specific counters, sampling packets, checking header field values is a separate process, at the latitude of the operator, and outside the design of the clairvoyant controller.

5.5.1 Changing paths

Central to clairvoyant networks is Magneto, a framework to change the path of any network flow in a hybrid SDN network, described in detail in a a previous paper [6]. Magneto can use one or a few strategically placed SDN-enabled switches to influence the forwarding behavior of legacy switches and end hosts. This allows us to gain visibility over any network flow without the need of making any modifications to existing legacy hardware devices or software components. As shown in Figure 5.6, Magneto can make the flow (S, D) visible to the SDN-enabled switch $OF2$. We summarize the design and properties of Magneto below.

Two key mechanisms enable Magneto to exert SDN-like control over legacy switches: *telekinesis* and *magnet addresses*. With telekinesis, OpenFlow switches send special *seed* packets to the legacy switches on the new path to be installed. This relies on the ability of an SDN controller to send PacketOut control messages to OpenFlow switches and instruct them to send custom-made packets into the network. The seed packets take advantage of MAC learning to *manipulate* legacy switches into updating a single forwarding entry in their routing tables.

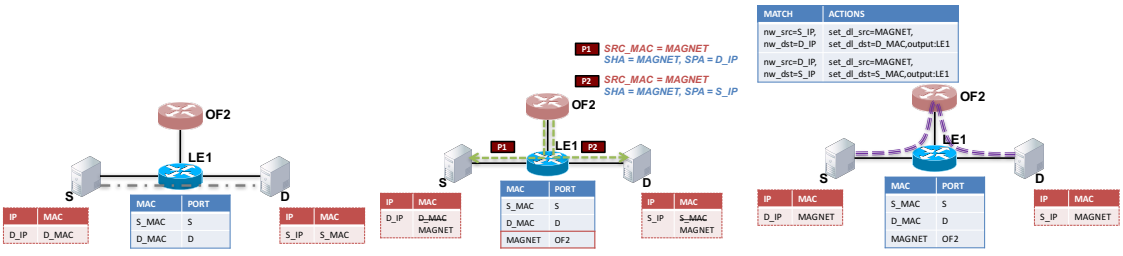
Magneto routes using fictitious MAC addresses (called *magnet* MAC addresses) associated with end hosts. Magnet MAC addresses are fictitious MAC addresses that do not correspond to any real host on the network, but are created by Magneto for the purpose of gaining network visibility and controlling routing & forwarding behaviors of end hosts and legacy switches. We "magnetize" a hybrid network by controlling the (magnet) MAC address mappings at end hosts via unicast gratuitous ARP messages generated by Magneto (via OpenFlow switches). When sending seed packets, we set the

source MAC address as a magnet MAC address associated with the path destination, rather than the real (native) MAC address of the destination host. The seed packet triggers the installation of a forwarding entry for the magnet MAC address. We also require that the seed packets are ARP packets and can reach the source host of the path. Thus, the source learns to associate the destination with its new magnet MAC address. Magneto uses different magnet MAC addresses to set up different paths for delivering traffic from other source hosts to the same destination. The last OpenFlow switch on each path rewrites the magnet MAC address to the native MAC address based on the destination IP address.

Magneto can set up a path through both SDN-enabled switches and strategically placed traditional monitoring devices. Figure 5.7 demonstrates how Magneto can set up a path through an SDN switch. To make the flow between S and D visible to the SDN-enabled switch $OF2$ (*i.e.*, the purple dashed line in Figure 5.6), we update its path from the top figure to the bottom figure. To install this new path, Magneto generates a new magnet MAC address *MAGNET*. It then crafts a seed packet with source MAC address as *MAGNET* and destination MAC address as S's MAC address (in the Ethernet header), source hardware address as *MAGNET* and source protocol address as D's IP address (in the ARP header). Magneto uses PacketOut to send this seed packet from $OF2$ to S. This packet triggers the addition of a new forwarding entry in $LE1$ for the *MAGNET* MAC address with corresponding incoming port and the update of the ARP table on S. Another seed packet with *MAGNET* MAC address and S's IP address is sent from $OF2$ to D, and the ARP table on D is updated in a similar manner.

5.5.2 Enabling visibility

To make flows visible, we provide a simple language for network operators to create *visibility tasks* for the Magneto controller. With a visibility task, the operator simply sets up a flow to be monitored at a specific location in the network. A visibility task consists of an action, a monitoring target (what flow(s) to monitor), a monitoring location (at what device to monitor the flows), and, optionally, a monitoring mirror (where to mirror the monitored flows). The action specifies whether the controller should add or delete the task. The monitoring target is a tuple of (source IP, destination IP) and



(a) (b) (c)

Figure 5.7: Path update between two hosts, S and D, in a hybrid network shown in Figure 5.6. Switch forwarding tables are in blue, host ARP caches are in red. (**(a): original network state**) Traffic between S and D flows through path in gray dotted line; (**(b): path update**) $OF2$ injects seed packets with magnet MACs to the legacy switch; (**(c): updated network state**) end hosts change the path to $(LE1, OF2, LE1)$.

represents the source and destination of the flow to be monitored. The monitoring location represents the SDN switch where the flow will be monitored. If operators do not have a preference for the location, the field can be empty or null. In this case, the task is assigned to the switch closest to the flow source or destination (if the source is a wildcard). For example, in Figure 5.6, the visibility task "(S, D) $OF2$" indicates that traffic between S and D will be monitored at $OF2$, "$(*, Y)$ NULL" indicates that traffic to Y can be monitored anywhere. Optionally, the operator can specify a monitoring mirror to have the monitoring switch mirror the flow to another device.

The Magneto controller takes visibility tasks as input and translate them into seed packets with magnet MACs that, in turn, generate forwarding rules that change the path of the flows. Given a visibility task, the controller generates a set of parameters about the flow and its monitoring location and generates magnet MAC addresses. Setting up a path using the magnet MACs follows the description in Section 5.5.1. Disabling a visibility task is similar and it requires the controller to send seed packets that revert the path of the flow back to default. In Section 5.6, we describe a more complex task scheduling mechanism, inspired by experimental results, and designed to reduce the cost of achieving visibility for multiple flows at the same time.

We illustrate these operations using Figure 5.6. When a network operator inputs "add (S, D) $OF2$", the Magneto controller generates a magnet MAC address for S to reach D, another magnet MAC address for D to reach S. These two magnet MAC

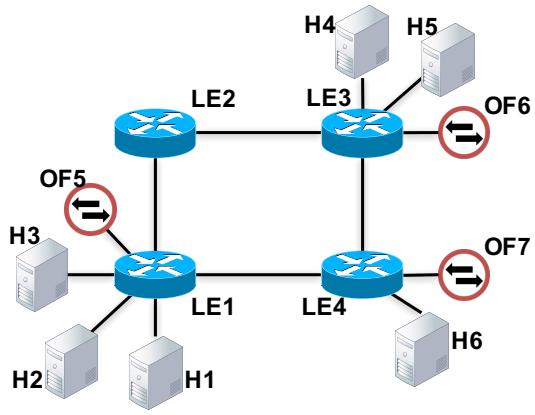


Figure 5.8: In a clairvoyant network, we can place SDN-enabled switches in every-edge—connecting each edge legacy switch to one SDN switch. The SDN-enabled switch can be either a hardware switch or a software switch running on a server. In this example, SDN-enabled switches are depicted in red and legacy switches are in blue. $LE1$, $LE3$, and $LE4$ are edge legacy switches, since they connect to end hosts. $H1$, $H2$, $H3$, $H4$ represent source hosts, and $H5$, $H6$ represent destination hosts. Every source host is sending traffic to every destination host.

addresses can be the same one if $OF2$ uses the same link to deliver the traffic to S and D (as shown in Figure 5.7). Together with S and D's IP addresses and real MAC addresses, these parameters are used by Magneto to set up the new path between S and D traversing $OF2$ as mentioned in Section 5.5.1. Later when the operator inputs "del (S, D) $OF2$" to delete this visibility task, the Magneto controller retrieves the related information and injects seed packets to revert the path to default.

5.6 Case study: edge visibility

In Section 5.5 we presented a general design for clairvoyant networks that can be used by operators as a basic building block towards deployment. As observed from previous analysis in Table 5.3, the target flow's performance can be affected due to the change of its path. Such change may even affect other flows' performance since those flows can compete for available bandwidth if their paths share some links. *Can we enable a flow's visibility with negligible performance degradation on itself as well as zero-touch effect on other non-target flows?* Here, we consider a specific deployment scenario and associated

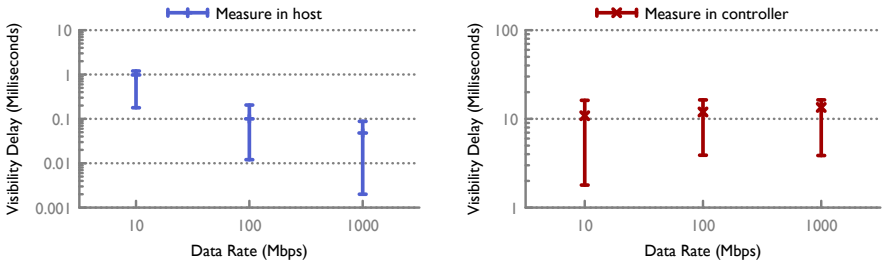


Figure 5.9: Visibility delay (the time to make a flow visible) of the Magneto controller remains low as we vary the data rate. We measure the visibility delay from both the host side (left) and the controller side (right).

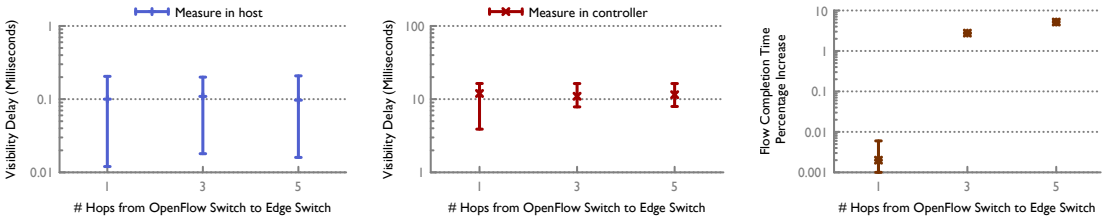


Figure 5.10: Visibility delay (the time to make a flow visible) of the Magneto controller remains low as we increase the distance between the OpenFlow switch and the edge legacy switch. We measure the visibility delay from both the host side (left) and the controller side (middle). Worst-case flow completion time has negligible increase (right).

design decisions that enable us to reduce the cost of achieving visibility for flows.

SDN switch deployment. To reduce the path stretch of monitored flows, we propose to introduce a few SDN switches (hardware or software [98]) to connect to all edge legacy switches (*i.e.*, all legacy switches that connect to end hosts) such that each edge switch connects to at least one SDN switch. In this way, changing the path of any flow adds at most two hops (from the edge legacy switch next to the source or destination to the connected SDN switch and back). Figure 5.8 shows an example with three edge legacy switches (*i.e.*, $LE1$, $LE3$, and $LE4$) connected to SDN switches. To make the flow between $H1$ and $H5$ visible, the controller redirects it through $OF5$ or $OF6$.

By pushing visibility to the edge of the network, we guarantee that any flow has negligible performance degradation when made visible. In addition, as the only new link in the flow's path is that from its source or destination legacy switch to the adjacent SDN

switch, the impact of changing the path on the other network flows is zero. However, when multiple flows are monitored by the same SDN switch, they may compete for the bandwidth of the link between the SDN switch and its adjacent legacy switch. We discuss how to alleviate this problem later in this section.

The last column in Table 5.3 shows the cost of this deployment strategy. With 48-port hardware OpenFlow switches and each port connected to one edge legacy switch, we need only 2% more OpenFlow switches to cover every edge switch. As expected, the average flow stretch and stress are smaller than other deployment strategies with the same number of SDN switches. Flow paths can extend on the average 1.5 times when made visible, while the competition for the same monitoring device is slightly higher than on the default path.

Visibility scheduling. When multiple flows are made visible through the same SDN switch, they will compete for the capacity of the link(s) connecting the SDN switch to its adjacent legacy switches. We propose a time-based scheduling in which one or more flows become visible in separate time slots such that the throughput of all flows in the same slot is lower than the capacity of the shared link.

First, the clairvoyant controller measures the throughput of each competing flow in a round-robin manner: it makes each flow visible for a small period of time (*e.g.*, 1s) and polls the counters associated with flow at the end of the visibility period. In Section 5.7, we show that making a flow visible and reverting it back to its original path is fast and consumes negligible resources.

Second, the controller combines all visibility tasks with the same monitoring locations in such a way that the sum of the throughputs of all flows from the same group of tasks does not exceed the capacity of the shared network link. We use a greedy heuristic to assign groups of tasks to each monitoring link at each monitoring interval. The visibility tasks in each group are enabled for each interval then disabled then enabled again until a task is deleted.

We illustrate the visibility scheduling using Figure 5.8, where each link has speed of 1 Gbps. An operator inputs two visibility tasks "add $(H1, H5)$ $OF5$" (say, $flow1$) and "add $(H2, H6)$ $OF5$" (say, $flow2$). The Magneto controller first enables the visibility for $flow1$ for one second to measure its throughput (say, 500 Mbps) and disables $flow1$'s visibility (*i.e.*, reverts its path back to the default). Then the controller enables $flow2$'s

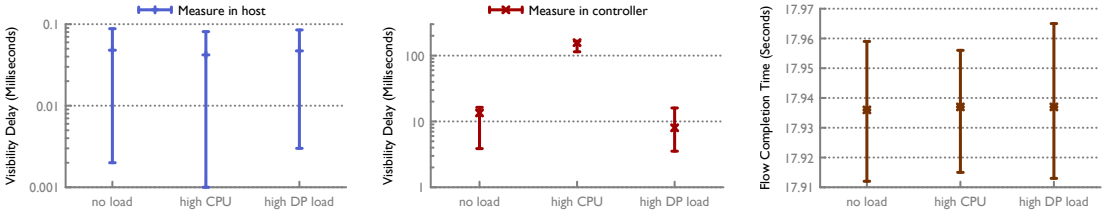


Figure 5.11: Visibility delay (the time to install a path) of the Magneto controller remains low as we introduce high load on the OpenFlow switch's control plane (*i.e.*, saturate CPU usage to be 99%) or data plane (*i.e.*, generate 10 Gbps additional traffic to go through the OpenFlow switch). We measure the visibility delay from both the host side (left) and the controller side (middle). Worst-case flow completion time has negligible increase (right) compared to when there is no additional load.

visibility for another second to measure its throughput (say, 200 Mbps) and disable its visibility. Since the total throughput of $flow1$ and $flow2$ is lower than the link capacity of $(LE1, OF5)$, they can be combined in the same time slot to be monitored.

5.7 Evaluation

In this section, we first show the Magneto controller can enable a flow's visibility very fast while introducing negligible performance degradation. Second, we demonstrate the Magneto controller is scalable—can handle tens of thousands of simultaneous visibility tasks on one OpenFlow switch.

We perform the following experiments on a real-world testbed in our lab. The testbed consists of six Dell servers, five Cisco Catalyst legacy switches [61], and two iwNetworks OpenFlow switches [62]. Each experiment is conducted for 100 times, unless otherwise noted.

5.7.1 Visibility delay

We define the visibility delay as the time it takes to make a flow visible, *i.e.* to update its path to traverse an SDN switch. We can measure the visibility delay from the controller or from one of the endpoints of the flow. The controller visibility delay represents the time between when the controller sends the first seed packet and when it receives the first mirrored packet. The endpoint visibility delay is the time between when the host

receives the first seed packet and when it sends the first data packet on the new path.

To measure the visibility delay, we connect two servers and one SDN switch to different ports of a Cisco legacy switch. We start a flow between the two servers and vary its data rate. Initially, the flow traverses only the legacy switch, but detours through the SDN switch once we submit a visibility task for it.

Figure 5.9 (left) shows the visibility delay measured from the end host. It remains low when we increase the data rate. That the visibility delay decreases as we increase the data rate is an artifact of our measurement: when the data rate is low, the time between two consecutive packets is higher therefore our measurement error is higher. Figure 5.9 (right) shows the visibility delay measured from our Magneto controller. It is higher than the delay measured from the end host, because it contains (1) the round-trip time from the controller to the OpenFlow switch where the seed packet is injected, and (2) the round-trip time from the OpenFlow switch to the host. The first round-trip time is dominant due to the overhead involved in forwarding a data packet on the control channel.

Next, we set the data rate at 100 Mbps and increase the number of hops between the OpenFlow switch and the edge legacy switch. The results in Figure 5.10 show that the visibility delay is not significantly affected by increasing the distance to the monitoring SDN switch. Figure 5.10 (right) shows the percentage increase of the flow completion time compared to the case when the flow is forwarded on the default path. We send 2,000 MB flows on the default path, one-hop hairpin path, three-hop hairpin path, and five-hop hairpin path. The result proves that the Magneto controller can provide visibility of a flow with negligible impact on completion time.

We can keep a flow visible for as little as 0.1ms—the minimum amount of time we achieved between sending two consecutive seed packets. However, ARP implementations on end hosts often have protection against ARP trashing, which limits the time between consecutive updates to the same ARP entry to one second. As a result, in practice, the smallest amount of time to maintain a flow's visibility is one second. Even with such a small visibility window, repeatedly enabling and disabling the visibility of a flow does not reduce its completion time. We observed only a 0.38% increase for a 10 GB flow when we enable and disable its visibility every second for 89s.

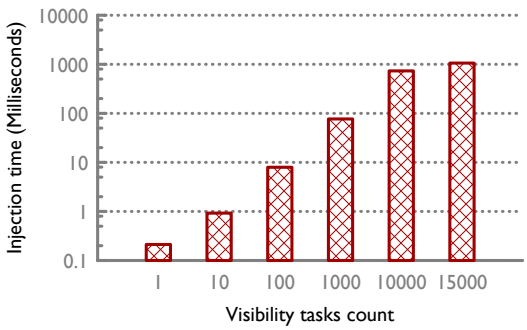


Figure 5.12: The Magneto controller can create/update/delete 15,000 individual visibility tasks on one OpenFlow switch in one second.

5.7.2 Scalability

Switch load. We evaluate clairvoyant networks when the OpenFlow switches are heavily-loaded using the same setup as in Section 5.7.1. To increase the load on the control plane, we saturate the CPU by adding dummy flows and querying flow statistics. To saturate the data plane, we introduce background traffic.

Figure 5.11 shows that when we saturate the control plane on the OpenFlow switch to reach 99% CPU usage, the visibility time measured on the controller increases by about 100 milliseconds. Yet the visibility time measured from the host is not affected. In terms of high data plane (DP) load, we introduce 10 Gbps more background traffic to the OpenFlow switch and observe that the visibility time from both the controller side or the host side is not affected. The flow completion time changes are negligible among the cases where there is no additional load, high CPU load, and high data plane load.

Many visibility tasks. How does the clairvoyant controller perform when operators submit many simultaneous visibility tasks? We vary the number of visibility tasks and measure the time it takes the controller to enable them. A visibility task triggers two seed packets, one to the source host(s) and the other to the destination host(s). There are no flows running for this experiments; we measure the time for the Magneto controller to inject seed packets and insert the forwarding rules. Figure 5.12 shows that the Magneto controller is capable of serving 15,000 individual visibility tasks on one OpenFlow switch in one second.

Next, we want to understand what happens when each visibility task updates an existing flow. For this experiment, we generate 100 flows and submit a visibility task for each of them. We are unable to generate more than 100 flows due to the limited number of servers in our testbed. Results in Table 5.4 show that making 100 flows visible at the same time increases CPU usage by 5.31% and memory usage by 2.12 KB.

5.8 Discussion

Deployment of clairvoyant networks in any enterprise is straightforward. Operators need to add at least one OpenFlow switch and the clairvoyant controller. To enable monitoring, one could proactively set up routes among all hosts through monitoring devices (for network-wide monitoring) or set up paths when flows start (for selective on-demand monitoring).

Who can use clairvoyant networks? Primarily enterprises that require fine-grained monitoring of their applications while accepting a little performance degradation. As they may increase the application latency by rerouting flows through monitoring devices, clairvoyant networks are not suited for enterprises that run latency-sensitive applications. For such specialized networks, hardware-based solutions installed on the data plane provide a better benefit/cost trade-off for flow monitoring [67].

Programmable monitoring platforms offer customizable and dynamic monitoring by relying on the visibility and control provided by SDN [65, 77]. Clairvoyant networks open new directions for programmable monitoring by allowing flexible monitoring tasks that capture and analyze data from both OpenFlow and legacy devices.

Interoperability. Clairvoyant networks work with STP, VLAN, BUM traffic and ARP poisoning mitigation technique, as described in detail in previous work. Network failures may pose a challenge as failed links trigger STP recomputation which in turn may lead to some paths becoming unusable. Failures in control plane may pause the Magneto controller to serve visibility tasks but the data plane is still functioning as usual.

5.9 Summary

We introduced *clairvoyant networks*, hybrid SDN networks that offer full control over all paths. Clairvoyant networks provide a low-cost medium for SDN-based monitoring by providing mechanisms to update the path of specific flows to make them traverse SDN switches and thus expose them to SDN-based monitoring techniques.

We studied the feasibility of clairvoyant networks using real-world and emulated network topologies and showed that, even with a single SDN-enabled switch, operators can make *any* flow visible for monitoring by an SDN-enabled switch, albeit by increasing the average path length by 38%. When clairvoyant networks contain more SDN-enabled switches (as little as 2% of all switches), their performance improves: most flows can also be monitored on the legacy data plane with little impact on network performance.

We also provided a basic design for clairvoyant networks by integrating an existing mechanism for updating path with a novel approach to specify and compile visibility tasks. Inspired by the feasibility study, we proposed a specific deployment scenario for clairvoyant networks. By connecting all edge legacy switches to at least one OpenFlow switch and implementing flow scheduling in the clairvoyant controller, we are able to significantly reduce the cost of making a flow visible.

Our current work focuses on building a programmable monitoring platform using clairvoyant networks. We are developing path selection and load balancing algorithms to improve the performance and reduce the cost of visible paths. We are also exploring hybrid monitoring applications that use both SDN and legacy monitoring devices to offer more efficient and accurate flow monitoring.

Chapter 6

Conclusion and Discussion

In this chapter, we summarize our contributions in Section 6.1, discuss open issues and future directions in Section 6.2, and conclude in Section 6.3.

6.1 Summary of Contributions

Our main contributions in this dissertation are as follows:

Our research in [99, 5] first conducts a measurement-oriented analysis of security group configuration and usage by customers in a public cloud platform based on real-world datasets. The goal is to understand what are the usage patterns ("good" and "bad" practices) in how cloud customers configure their security groups. Motivated by the results and insights obtained from this measurement study, we develop a cloud security group analysis system which employs visual analytics to assist cloud customers in understanding the static and dynamic access relations among VM instances. Furthermore, our system helps cloud customers diagnose potential misconfigurations and provides suggestions to refine security group configurations. By applying the proposed system to all existing customers hosted on the public cloud, more than 80% customers are identified to have improperly configured security groups. Hence, the novel analysis and diagnose system helps prevent cloud applications from potential security vulnerabilities and enhance cloud platform security.

Second, we propose a novel framework [100, 47, 6] for incremental and graceful transition of legacy networks, which enables operators to transition legacy networks to SDN

networks in stages by gradually replacing legacy devices with SDN-enabled devices as needed and as budgets allow. Hence, network operators can gracefully experiment with SDN networks to gain experience and build confidence while eliminating or minimizing service disruption. More importantly, operators can enjoy the benefits as fully deployed SDN networks. we design and build a novel unified network management controller that exerts SDN-like, fine-grained routing control over both SDN-enabled and legacy switches in hybrid networks. Our system can install diverse paths with little control overhead, and exert full control over routing even when only 20% of the switches are SDN-enabled. Our work successfully demonstrates that it is possible to enjoy the benefits of a wholly deployed SDN network but at a fraction of the cost by strategically replacing only a few legacy switches with SDN-enabled switches.

Third, with the goal of obtaining fine-grained network visibility as to monitor "who is talking to whom","how much traffic is being sent to a destination, say Google", we propose *clairvoyant networks* [101] to provide visibility for any network flow at any time and with low cost. Clairvoyant networks are partially programmable—they require as few as one SDN switch—and rely on a specialized network controller that controls paths through both the SDN and legacy networks. The clairvoyant controller allows operators to define *what to see*, *where to see*, and *how to see*; then enables/disables the specified flows' visibility in a task scheduler, within milliseconds. Our evaluation on a lab testbed and through extensive simulations on large enterprise network topologies show that, even with a single SDN-enabled switch, operators can make *any* flow visible for monitoring within milliseconds, albeit at 38% average increase in path length. With as many as 2% strategically chosen legacy switches replaced with SDN switches, clairvoyant networks achieve on-demand flow visibility with negligible overhead.

6.2 Open Issues and Future Directions

Network management has always been worthwhile endeavor, and operators used to drive networks with "manual transmission". Driven by the rising attention to network availability, performance, security, resilience and scalability, network management calls for the upgrade to "auto transmission" or even "self-driving networks". The works presented in this thesis focused on building systems to make networks more secure and

manageable, and raised the following open questions and directions.

6.2.1 System Integration and Deployment

We proposed Socrates, a security group configuration diagnosis system, based on security group configurations from servers run by an IaaS cloud. Security groups are currently implemented on the servers that host the associated VMs. The main limitation of such implementation is: the decision of allowing/denying traffic happens in the end—destination hosts, which occupies additional network bandwidth to route those traffic to the destination hosts. With the adoption of SDN, one future direction is to enforce security groups in SDN switches, as close as possible to the source hosts.

Although Magneto focuses primarily on reaping benefits of SDN in a hybrid L2 network, one open question is how it integrates with other network components and services in real deployment. Generally, enterprise networks consist of L2 switches, L3 routers, middleboxes (*e.g.*, firewalls, NATs), DHCP and DNS servers. Link-state routing protocols (*e.g.*, OSPF and IS-IS) are widely used in legacy L3 networks. Unfortunately, these protocols are also relatively inflexible, since they direct all traffic over shortest paths. Integrating Magneto and Fibbing [55] will provide opportunities to joint L3/L2 routing optimization, VLAN management and traffic engineering.

In Clairvoyant networks, we introduced a software solution to enable network visibility on-demand and proposed to place those visibility enablers in the edge. Though we focused on introducing a new software solution to enable dynamic network visibility, one future direction is how to integrate our Clairvoyant framework with legacy monitoring solutions such as NetFlow and sFlow in order to make use of different monitoring techniques to maximize monitoring coverage and benefits.

6.2.2 Automating Network Management

With the rapidly increasing scale, production networks need automated management systems. Direct human interaction with network devices should be reduced as much as possible for two main reasons: *efficiency*—manual configurations are much slower than automated processes, and *correctness*—manual configurations are more error-prone than a program that can handle different cases.

Network researchers have made great efforts on the control plane and data plane, but much less study has been done on the management plane. One inevitable future direction for network management is: *how to automate the network management process that consists of design, operation, monitoring, and troubleshooting?*

6.2.3 Building Self-Running Networks

Beyond automating network management, a more ambitious future direction is to build self-running networks. A northbound API is provided to network operators to initially declare network designs (*e.g.*, device connections, subnet arrangement) and high-level policies (*e.g.*, SLAs, ACLs). Taking the input, the network management system automatically configures network devices, enforces network policies, and monitors network states and performance.

The runtime of a self-running network should be automatically *learning and adapting*. It translates the pre-defined high-level policies into specific control and monitoring tasks, and deploys these tasks correctly and efficiently. Using data analytic techniques, a self-running network learns about network states and performance. It then feeds the learned information into the control operations. As a result, network control benefits from being integrated with network monitoring and measurements, and adapts its control decision to achieve better network and application performance.

6.3 Concluding Remarks

In summary, this thesis studies the management of enterprise and data center networks towards better manageability and security. We proposed systems that are capable of: i) helping operators and users understand and refine security policy configurations; ii) enhancing routing flexibility to increase network utilization and efficiency; and iii) enabling on-demand network visibility for better network control.

References

[1] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, et al. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. *ACM SIGCOMM Computer Communication Review*, 45(4):183–197, 2015.

[2] Theophilus Benson, Aditya Akella, and David A Maltz. Unraveling the complexity of network management. In *NSDI*, pages 335–348, 2009.

[3] Network Downtime and Complexity Results in Job and Revenue Loss plus Missed Business Opportunities. `http://www.avaya.com/en/about-avaya/newsroom/news-releases/2015/pr-040215/`.

[4] Veriflow Launches Disruptive Platform; Survey Reveals Complexity, Change and Human Factors Cause Network Outages and Vulnerabilities. `http://www.veriflow.net/veriflow-launches-disruptive-platform-survey/`.

[5] ©2016 IEEE. Reprinted, with permission from, Cheng Jin, Abhinav Srivastava, and Zhi-Li Zhang. Understanding security group usage in a public iaas cloud. In *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*, pages 1–9. IEEE, 2016.

[6] Cheng Jin, Cristian Lumezanu, Qiang Xu, Hesham Mekky, Zhi-Li Zhang, and Guofei Jiang. Magneto: Unified fine-grained path control in legacy and openflow hybrid networks. In *Proceedings of the Symposium on SDN Research*, pages 75–87. ACM, 2017, DOI: http://dx.doi.org/10.1145/3050220.3050229.

[7] White paper: Cisco VNI Forecast and Methodology, 2015-2020. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html.

[8] Cisco Global Cloud Index: Forecast and Methodology, 20152020. http://www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.pdf.

[9] Enterprise Networking Market Analysis By Equipment (Ethernet Switch, Enterprise Routers, WLAN, Network Security) And Segment Forecasts To 2024. http://www.grandviewresearch.com/industry-analysis/enterprise-networking-market.

[10] Nick McKeown. How sdn will shape networking. Open Networking Summit 2011, 2011. Available from https://www.youtube.com/watch?v=c9-K5O_qYgA.

[11] Scott Shenker. The future of networking, and the past of protocols. Open Networking Summit 2011, 2011. Available from https://www.youtube.com/watch?v=YHeyuD89n1Y.

[12] SDN Architecture Overview. v1.0, 2013.

[13] Nick Feamster, Jennifer Rexford, and Ellen Zegura. The road to sdn: an intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review*, 44(2):87–98, 2014.

[14] Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2015.

[15] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. Nox: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, 2008.

[16] Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki

Hama, et al. Onix: A distributed control platform for large-scale production networks. In *OSDI*, volume 10, pages 1–6, 2010.

[17] Openflow switch specification, 1.5.1. `https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf`.

[18] Pat Bosshart, Dan Daly, Martin Izzard, Nick McKeown, Jennifer Rexford, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communications Review*, July 2014.

[19] László Molnár, Gergely Pongrácz, Gábor Enyedi, Zoltán Lajos Kis, Levente Csikor, Ferenc Juhász, Attila Kőrösi, and Gábor Rétvári. Dataplane specialization for high-performance openflow software switching. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*, pages 539–552. ACM, 2016.

[20] 9 Worst Cloud Security Threats. `http://www.informationweek.com/cloud/infrastructure-as-a-service/9-worst-cloud-security-threats/d/d-id/1114085`.

[21] Software Defined Networks Study. `http://www.currentanalysis.com/news/2014/pr-SDN-NFV-Deployment.asp`.

[22] The SNMP Protocol. `http://www.snmp.com/protocol/`.

[23] sFlow. `http://sflow.org/`.

[24] Introduction to Cisco IOS NetFlow - A Technical Overview. `http://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html`.

[25] C. Estan, K. Keys, D. Moore, and G. Varghese. Building a Better NetFlow. In *ACM Sigcomm*, 2004.

[26] Jianning Mai, Chen-Nee Chuah, Ashwin Sridharan, Tao Ye, and Hui Zang. Is sampled data sufficient for anomaly detection? In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 165–176. ACM, 2006.

[27] Nick Duffield, Carsten Lund, and Mikkel Thorup. Estimating flow distributions from sampled flow statistics. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 325–336. ACM, 2003.

[28] Cristian Estan, Ken Keys, David Moore, and George Varghese. Building a better netflow. In *ACM SIGCOMM Computer Communication Review*, volume 34, pages 245–256. ACM, 2004.

[29] Amazon EC2. *http://aws.amazon.com/ec2/*.

[30] OpenStack. *http://www.openstack.org/*.

[31] AWS EC2 security groups. *http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-network-security.html*.

[32] OpenStack security groups. *http://docs.openstack.org/network-admin/admin/content/securitygroups.html*.

[33] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. B4: Experience with a globally-deployed software defined wan. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 3–14. ACM, 2013.

[34] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. Achieving high utilization with software-driven wan. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 15–26. ACM, 2013.

[35] Building Express Backbone: Facebooks new long-haul network. `https://code.facebook.com/posts/1782709872057497/building-express-backbone-facebook-s-new-long-haul-network`.

[36] Brandon Schlinker, Hyojeong Kim, Timothy Cui, Ethan Katz-Bassett, Harsha V Madhyastha, Italo Cunha, James Quinn, Saif Hasan, Petr Lapukhov, and Hongyi Zeng. Engineering egress with edge fabric: Steering oceans of content to the

world. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 418–431. ACM, 2017.

[37] Kok-Kiong Yap, Murtaza Motiwala, Jeremy Rahe, Steve Padgett, Matthew Holliman, Gary Baldus, Marcus Hines, Taeeun Kim, Ashok Narayanan, Ankur Jain, et al. Taking the edge off with espresso: Scale, reliability and programmability for global internet peering. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 432–445. ACM, 2017.

[38] AT&T Vision Alignment Challenge Technology Survey - AT&T Domain 2.0 Vision White Paper. Online Document, Available: . `https://www.att.com/Common/about_us/pdf/AT&T%20Domain%202.0%20Vision%20White%20Paper.pdf`.

[39] Cisco. Campus Network for High Availability Design Guide. `http://bit.ly/1ffWkzT`.

[40] Juniper. Juniper Campus Networks Reference Architecture, 2010. `http://juni.pr/1rR0vaZ`.

[41] Yu-Wei Eric Sung, Sanjay G Rao, Geoffrey G Xie, and David A Maltz. Towards systematic design of enterprise networks. In *Proceedings of the 2008 ACM CoNEXT Conference*, page 22. ACM, 2008.

[42] David Ke Hong, Yadi Ma, Sujata Banerjee, and Z. Morely Mao. Incremental deployment of SDN in hybrid enterprise and ISP networks. In *SOSR*, 2016.

[43] New Generation Network testbed. `http://www.jgn.nict.go.jp/jgn2plus_archive/english/index.html`.

[44] Teemu Koponen and et al. Network virtualization in multi-tenant datacenters. In *USENIX NSDI*, 2014.

[45] Hui Lu, Nipun Arora, Hui Zhang, Cristian Lumezanu, Junghwan Rhee, and Guofei Jiang. HybNET: Network Manager for a Hybrid Network Infrastructure. In *Middleware*, 2013.

[46] Dan Levin, Marco Canini, Stefan Schmid, Fabian Schaffert, and Anja Feldmann. Panopticon: Reaping the Benefits of Incremental SDN Deployment in Enterprise Networks. In *USENIX Annual Technical Conference*, 2014.

[47] Cheng Jin, Cristian Lumezanu, Qiang Xu, Zhi-Li Zhang, and Guofei Jiang. Telekinesis: controlling legacy switch routing with openflow in hybrid networks. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, page 20. ACM, 2015.

[48] Richard Wang, Dana Butnariu, and Jennifer Rexford. OpenFlow-based Server Load Balancing Gone Wild. In *Hot-ICE*, 2011.

[49] Naga Katta, Mukesh Hira, Changhoon Kim, Anirudh Sivaraman, and Jennifer Rexford. Hula: Scalable load balancing using programmable data planes. In *Proceedings of the Symposium on SDN Research*, page 10. ACM, 2016.

[50] Naga Katta, Mukesh Hira, Aditi Ghag, Changhoon Kim, Isaac Keslassy, and Jennifer Rexford. Clove: How i learned to stop worrying about the core and love the edge. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pages 155–161. ACM, 2016.

[51] Ryan Hand and Eric Keller. Closedflow: Openflow-like control over proprietary devices. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 7–12. ACM, 2014.

[52] Michael Markovitch and Stefan Schmid. Shear: A highly available and flexible network architecture. In *ICNP*, 2015.

[53] Tim Nelson, Andrew D Ferguson, Da Yu, Rodrigo Fonseca, and Shriram Krishnamurthi. Exodus: Toward automatic migration of enterprise network configurations to sdns. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, page 13. ACM, 2015.

[54] Global ONOS and SDN-IP deployment. `http://onosproject.org/wp-content/uploads/2015/06/PoC_global-deploy.pdf`.

[55] Stefano Vissicchio, Olivier Tilmans, Laurent Vanbever, and Jennifer Rexford. Central control over distributed routing. In *ACM SIGCOMM*, 2015.

[56] Naga Praveen Katta, Jennifer Rexford, and David Walker. Incremental consistent updates. In *HotSDN*, 2013.

[57] Mark Reitblatt, Nate Foster, Jennifer Rexford, Cole Schelsinger, and David Walker. Abstractions for network update. In *ACM Sigcomm*, 2012.

[58] Stefano Vissicchio, Laurent Vanbever, and Jennifer Rexford. Sweet little lies: Fake topologies for flexible routing. In *ACM HotNets*, 2014.

[59] ONOS. `http://onosproject.org/`.

[60] Hongyi Zeng, Peyman Kazemian, George Varghese, and Nick McKeown. Automatic test packet generation. In *Proceedings of the 2012 ACM CoNEXT Conference*, pages 241–252. ACM, 2012.

[61] Cisco switches. `http://www.cisco.com/c/en/us/products/switches/index.html`.

[62] iwNetworks switches. `http://www.iwnetworks.com/main/products`.

[63] PF_RING. `http://www.ntop.org/products/packet-capture/pf_ring/`.

[64] Hesham Mekky, Fang Hao, Sarit Mukherjee, Zhi-Li Zhang, and TV Lakshman. Application-aware data plane processing in sdn. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 13–18. ACM, 2014.

[65] Minlan Yu, Lavanya Jose, and Rui Miao. Software defined traffic measurement with opensketch. In *NSDI*, volume 13, pages 29–42, 2013.

[66] Anukool Lakhina, Mark Crovella, and Christophe Diot. Diagnosing network-wide traffic anomalies. In *ACM SIGCOMM Computer Communication Review*, volume 34, pages 219–230. ACM, 2004.

[67] Myungjin Lee, Nick Duffield, and Ramana Rao Kompella. Not All Microseconds are Equal: Fine-Grained Per-Flow Measurements with Reference Latency Interpolation. In *ACM Sigcomm*, 2010.

[68] Ahsan Habib, Sonia Fahmy, and Bharat Bhargava. Monitoring and controlling qos network domains. *International Journal of Network Management*, 15(1):11–29, 2005.

[69] Seungwon Shin and Guofei Gu. Cloudwatcher: Network security monitoring using openflow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?). In *Network Protocols (ICNP), 2012 20th IEEE International Conference on*, pages 1–6. IEEE, 2012.

[70] Vern Paxson. Bro: A System for Detecting Network Intruders in Real-Time. In *Usenix Security*, 1998.

[71] Configuring span and rspan. `http://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst2960/software/release/12-2_55_se/configuration/guide/scg_2960/swspan.html`.

[72] Understanding network taps. `https://www.gigamon.com/sites/default/files/resources/whitepaper/wp-understanding-network-taps-the-first-step-to-visibility-3164.pdf`.

[73] Martin Casado, Michael J Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. Ethane: Taking control of the enterprise. In *ACM SIGCOMM Computer Communication Review*, volume 37, pages 1–12. ACM, 2007.

[74] Seungwon Shin, Phillip Porras, Vinod Yegneswaran, Martin Fong, Guofei Gu, and Mabry Martin. Fresco: Modular composable security services for software-defined networks. In *Network and Distributed Security Symposium*, 2013.

[75] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: enabling innovation in campus networks. *ACM Sigcomm CCR*, 38:69–74, 2008.

[76] Chiang Liu, Arun Raghuramu, Chen-Nee Chuah, and Balachander Krishnamurthy. Piggybacking network functions on sdn reactive routing: A feasibility study. In *SOSR*, 2017.

[77] Curtis Yu, Cristian Lumezanu, Vishal Singh, Yueping Zhang, Geoff Jiang, and Harsha V. Madhyastha. Monitoring network utilization with zero measurement cost. In *PAM*, 2013.

[78] Curtis Yu, Cristian Lumezanu, Abhishek Sharma, Qiang Xu, Guofei Jiang, and Harsha V Madhyastha. Software-defined latency monitoring in data center networks. In *International Conference on Passive and Active Network Measurement*, pages 360–372. Springer, 2015.

[79] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*, pages 101–114. ACM, 2016.

[80] Srinivas Narayana, Jennifer Rexford, and David Walker. Compiling path queries in software-defined networks. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 181–186. ACM, 2014.

[81] Omid Alipourfard, Masoud Moshref, and Minlan Yu. Re-evaluating measurement algorithms in software. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, page 20. ACM, 2015.

[82] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. Scream: Sketch resource allocation for software-defined measurement. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, page 14. ACM, 2015.

[83] Xuemei Liu, Meral Shirazipour, Minlan Yu, and Ying Zhang. Mozart: Temporal coordination of measurement. In *Proceedings of the Symposium on SDN Research*, page 13. ACM, 2016.

[84] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. Flowradar: a better netflow for data centers. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 311–324. USENIX Association, 2016.

[85] Vibhaalakshmi Sivaraman, Srinivas Narayana, Ori Rottenstreich, S Muthukrishnan, and Jennifer Rexford. Heavy-hitter detection entirely in the data plane. In *Proceedings of the Symposium on SDN Research*, pages 164–176. ACM, 2017.

[86] Qun Huang, Xin Jin, Patrick PC Lee, Runhui Li, Lu Tang, Yi-Chao Chen, and Gong Zhang. Sketchvisor: Robust network measurement for software packet processing. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 113–126. ACM, 2017.

[87] Martin Roesch. Snort - Lightweight Intrusion Detection for Networks. In *LISA*, 1999.

[88] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. Dream: dynamic resource allocation for software-defined measurement. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 419–430. ACM, 2014.

[89] Changhoon Kim, Anirudh Sivaraman, Naga Katta, Antonin Bas, Advait Dixit, and Wobker Lawrence J. In-band network telemetry via programmable dataplanes. In *SOSR Demos*, 2015.

[90] Haoxian Chen, Nate Foster, Jake Silverman, Michael Whittaker, Brandon Zhang, and Rene Zhang. Felix: Implementing traffic measurement on end hosts using program analysis. In *Proceedings of the Symposium on SDN Research*, page 14. ACM, 2016.

[91] Amin Tootoonchian, Monia Ghobadi, and Yashar Ganjali. OpenTM: Traffic Matrix Estimator for OpenFlow Networks. In *PAM*, 2010.

[92] Lavanya Jose, Minlan Yu, and Jennifer Rexford. Online measurement of large traffic aggregates on commodity switches. In *USENIX Hot-ICE*, 2011.

[93] Ye Yu, Chen Qian, and Xin Li. Distributed and collaborative traffic monitoring in software defined networks. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 85–90. ACM, 2014.

[94] Andrew R Curtis, Jeffrey C Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. Devoflow: Scaling flow management for high-performance networks. *ACM SIGCOMM Computer Communication Review*, 41(4):254–265, 2011.

[95] Curtis Yu, Cristian Lumezanu, Harsha V Madhyastha, and Guofei Jiang. Characterizing rule compression mechanisms in software-defined networks. In *International Conference on Passive and Active Network Measurement*, pages 302–315. Springer, 2016.

[96] Masoud Moshref, Minlan Yu, Abhishek B Sharma, and Ramesh Govindan. Scalable rule management for data centers. In *NSDI*, volume 13, pages 157–170, 2013.

[97] Naga Katta, Omid Alipourfard, Jennifer Rexford, and David Walker. Cacheflow: Dependency-aware rule-caching for software-defined networks. In *Proceedings of the Symposium on SDN Research*, page 6. ACM, 2016.

[98] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, Keith Amidon, and Martin Casado. The Design and Implementation of Open vSwitch. In *Proc. of NSDI*, 2015.

[99] Cheng Jin, Abhinav Srivastava, Yu Jin, and Zhi-Li Zhang. Secgras: Security group analysis as a cloud service. In *Network Protocols (ICNP), 2014 IEEE 22nd International Conference on*, pages 215–220. IEEE, 2014.

[100] Cheng Jin, Cristian Lumezanu, Qiang Xu, Nipun Arora, Abhishek Sharma, Zhi-Li Zhang, Guofei Jiang, and Zhuotao Liu. Underlay Computation for Network Virtualization in Hybrid SDN Networks. `https://sites.google.com/site/2014socc/home/posters`.

[101] Cheng Jin, Cristian Lumezanu, Zhi-Li Zhang, and Haifeng Chen. Clairvoyant networks. In *Under Review*.

Appendix A

Publications

In addition to this dissertation, the presented results are also documented in the following published papers.

A.1 Publications by Date

- **Cheng Jin**, Cristian Lumezanu, Zhi-Li Zhang, and Haifeng Chen, "Clairvoyant Networks," Under Review
- Hyun-wook Baek, **Cheng Jin**, Guofei Jiang, Cristian Lumezanu, Jacobus van der Merwe, Ning Xia, and Qiang Xu, "Towards Traffic Usage Accountability via Coarse-grained Measurements in Multi-tenant Data Centers," The ACM Symposium on Cloud Computing (SoCC'17)
- **Cheng Jin**, Cristian Lumezanu, Qiang Xu, Hesham Mekky, Zhi-Li Zhang, and Guofei Jiang, "Magneto: Unified Fine-grained Path Control in Legacy and OpenFlow Hybrid Networks," The ACM Sigcomm Symposium on SDN Research (SOSR'17), DOI: http://dx.doi.org/10.1145/3050220.3050229
- **Cheng Jin**, Cristian Lumezanu, Qiang Xu, Hesham Mekky, Zhi-Li Zhang, and Guofei Jiang, "Exerting Fine-Grained Path Control over Legacy Switches in Hybrid Networks," University of Minnesota, Technical Report. UMN CS TR 16-035 2016

- ©2016 IEEE. Reprinted, with permission, from **Cheng Jin**, Abhinav Srivastava, and Zhi-Li Zhang, "Understanding Security Group Usage in a Public IaaS Cloud," The IEEE International Conference on Computer Communications (INFOCOM'16)

- **Cheng Jin**, Cristian Lumezanu, Qiang Xu, Zhi-Li Zhang, and Guofei Jiang, "Telekinesis: Controlling Legacy Switch Routing with OpenFlow in Hybrid Networks," The ACM Sigcomm Symposium on SDN Research (SOSR'15)

- **Cheng Jin**, Cristian Lumezanu, Qiang Xu, Nipun Arora, Abhishek Sharma, Zhi-Li Zhang, Guofei Jiang, and Zhuotao Liu, "Poster: Underlay Computation for Network Virtualization in Hybrid SDN Networks," The ACM Symposium on Cloud Computing 2014 (SoCC'14)

- **Cheng Jin**, Abhinav Srivastava, Yu Jin, and Zhi-Li Zhang, "Secgras: Security Group Analysis As a Cloud Service," The 22nd IEEE International Conference on Network Protocols (ICNP'14)

- Zhenhua Li, **Cheng Jin**, Tianyin Xu, Christo Wilson, Yao Liu, Linsong Cheng, Yunhao Liu, Yafei Dai, and Zhi-Li Zhang, "Towards Network-level Efficiency for Cloud Storage Services," The Internet Measurement Conference (IMC'14)

- Hesham Mekky, **Cheng Jin**, and Zhi-Li Zhang, "VIRO-GENI: SDN-based Approach for a Non-IP Protocol in GENI", The GENI Research and Educational Experiment Workshop (GREE'14)

- Zhenhua Li, Christo Wilson, Zhefu Jiang, Yao Liu, Ben Y. Zhao, **Cheng Jin**, Zhi-Li Zhang, and Yafei Dai, "Efficient Batched Synchronization in Dropbox-like Cloud Storage Services," The ACM/IFIP/USENIX Middleware 2013

- Eman Ramadan, Hesham Mekky, **Cheng Jin**, Braulio Dumba, and Zhi-Li Zhang, "Provably Resilient Network Fabric with Bounded Latency Requirements," Under Submission