July 2020

Improving Computer Network Operations Through Automated Interpretation of State

Abhishek Dwaraki
Washington University in St. Louis

Abhishek Dwaraki

IMPROVING COMPUTER NETWORK OPERATIONS THROUGH AUTOMATED INTERPRETATION OF STATE

A Dissertation Presented

by

ABHISHEK DWARAKI

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2020

Electrical and Computer Engineering

IMPROVING COMPUTER NETWORK OPERATIONS THROUGH AUTOMATED INTERPRETATION OF STATE

A Dissertation Presented

by

ABHISHEK DWARAKI

DEDICATION

To my parents

We are at the very beginning of time for the human race. It is not unreasonable that we grapple with problems. But there are tens of thousands of years in the future. Our responsibility is to do what we can, learn what we can, improve the solutions, and pass them on.

Richard Feynman

ACKNOWLEDGMENTS

A lot of people have been instrumental in me being where I am today. Thanking all the people would demand a lengthy article by itself. But there are some that have had a significant and immeasurable bearing on my Ph.D, both in research direction and motivation. This is raising it to them.

I would like to express my special appreciation and thanks to Prof. Tilman Wolf, advisor, mentor, guide, career counselor extraordinaire. This space cannot do justice to my gratitude for your efforts in reading all my doctoral work, specially my dissertation. I have learned to be a better writer, thinker, and a methodologist from your guidance and mentoring. I would like to thank you for encouraging my research and more importantly, for allowing me to grow as a researcher. You knew when to push me and when to let me find my way.

Prof. Sitaraman, my sincere thanks to you for believing in my ideas. Your insights, guidance, thought-provoking research questions, while being grounded in practicality, have contributed invaluably in shaping my main project, and by extension, this dissertation.

I would like to thank Prof. Jim Kurose for agreeing to be on my committee and helping me strike one thing off my bucket list. To try and measure the value of your and Prof. Lixin Gao's inputs would be doing injustice. I am grateful for your insightful comments and encouragement, and for the hard questions that pushed me to widen my research from different perspectives.

I thank Dr. Sriram Natarajan for getting me to "drink the SDN Kool-Aid", as my advisor puts it. You have guided me through some of the hardest moments of my academic and professional life. My research directions would not have been possible without you.

Additionally, I would be remiss if I did not thank Dr. Puneet Sharma and Eric Crawley. Without the both of you, we would never have gone to the Innovation-Corps cohort, and this dissertation would have been but a figment of my imagination.

I can imagine my advisor at this point asking me why my acknowledgments section appears to be a “thank-you festival”. I have had the great privilege of working with some extremely intelligent and gifted individuals. Divyashri Bhat, Arman Pouraghily and Supreeth Shashtri, Drs. in their own right, have been influential in enabling me to think “out-of-the-box” and analytically challenging my work, constantly helping it improve. I would like to thank them for the endless stream of support and being my go-to people for brainstorming. I am glad your time here at UMass Amherst overlapped with mine and that we shared thoughts, ideas and workspaces. Your help and support has also helped shape this dissertation to what it is today. Priyanka Dattatri, I cannot state, or for that matter, even measure the importance of your friendship, presence, constant support and encouragement in pushing me towards my goals, academic and otherwise. You are a terrific role model and an inspiration to me.

Much of my research would be lacking but for my collaborators and co-authors, Drs. Xinming Chen and Richard Freedman. Xinming, you have a knack of tackling and solving hard problems quite elegantly. As for Rick, you somehow make the complex math behind machine learning not so complex after all.

Finally, I would like to thank my mother, who encouraged me to become a doctor (if not one who actually saves lives) and my father, who have always let me chart my own path.

ABSTRACT

IMPROVING COMPUTER NETWORK OPERATIONS THROUGH AUTOMATED INTERPRETATION OF STATE

MAY 2020

ABHISHEK DWARAKI

B.E., VISVESWARAYA TECHNOLOGICAL UNIVERSITY, KARNATAKA, INDIA

M.S., UNIVERSITY OF MASSACHUSETTS, AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Tilman Wolf

Networked systems today are hyper-scaled entities that provide core functionality for distributed services and applications spanning personal, business, and government use. It is critical to maintain correct operation of these networks to avoid adverse business outcomes. The advent of programmable networks has provided much needed fine-grained network control, enabling providers and operators alike to build some innovative networking architectures and solutions. At the same time, they have given rise to new challenges in network management. These architectures, coupled with a multitude of devices, protocols, virtual overlays on top of physical data-plane etc. make network management a highly challenging task. Existing network management methodologies have not evolved at the same pace as the technologies and architectures. Current network management practices do not provide adequate solutions for highly dynamic, programmable environments. We have a long way to go in developing management methodologies that can meaningfully contribute to networks becoming self-healing entities. The goal of my research is to contribute to the design and development of networks towards transforming them into self-healing entities.

Network management includes a multitude of tasks, not limited to diagnosis and troubleshooting, but also performance engineering and tuning, security analysis etc. This research explores novel methods of utilizing network state to enhance networking capabilities. It is constructed around hypotheses based on careful analysis of practical deficiencies in the field. I try to generate real-world impact with my research by tackling problems that are prevalent in deployed networks, and that bear practical relevance to the current state of networking. The overarching goal of this body of work is to examine various approaches that could help enhance network management paradigms, providing administrators with a better understanding of the underlying state of the network, thus leading to more informed decision-making. The research looks into two distinct areas of network management, troubleshooting and routing, presenting novel approaches to accomplishing certain goals in each of these areas, demonstrating that they can indeed enhance the network management experience.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

MACHINE LEARNING MEETS PROGRAMMABLE NETWORKING

The unprecedented pace of technological growth in recent times has resulted in strenuous demands being placed on computer networks. Traditional network architectures have evolved gradually over time to incorporate new technologies, but were never designed with these advances in mind. Subsequently, they are unable to support this pace and gamut of innovative ideas and methodologies. Enterprises and carriers are constantly on the lookout for cutting-edge technologies that can help them further their efforts at delivering new, innovative services to customers.

To this end, programmable networking is spearheading an architectural effort at making networks more flexible and dynamically adaptable to current requirements. These advances result in high levels of flexibility and network control, thus enabling enterprises to build highly scalable and dynamic networks that are extremely responsive and adaptive to changing business needs [34].

Perhaps the most significant influence contemporary technology may have on us has been its impact on the way we make decisions. Decision making, human, automated or otherwise, is exceedingly dependent on the amount of information available to base decisions on. In a realm where dynamism is the rule, rather than the norm, it is not just the quantity, but the quality of information that could potentially make the difference between a stable network and operational downtimes.

Human decision making works well with high-level activity and patterns. Manually mining and understanding low-level data affects efficiency adversely. On the contrary, evaluating large amounts of data and breaking them down in recognizable patterns is something that computers excel at. Correlating this low-level information to high-level activity, qualitatively enhancing the information available to a human operator could be very beneficial. Machine-learning algorithms, especially the subset of them dedicated to activity recognition form a significant core of this research effort. It is to this end of improving the efficiency of decision making in highly dynamic, flexible networking environments, that we attempt to meld the math behind machine-learning and the engineering innovation in programmatic networking.

1.1 A Brief History of Programmable Networking

Programmable networking is responsible for a paradigm shift in the way we architect and deploy networks. It continues to revolutionize network design and innovation. Before we delve into the exciting realm of network programmability and the efficient management of these dynamic entities, it would be really beneficial to understand how the technology came to be what it is today. Much of this background is from an excellent historical review of SDNs from [49].

Computer networks are complex entities that interconnect a diverse set of devices from switches and routers, to middleboxes such as firewalls, load balancers etc. The control software on these devices are vendor specific, that makes them proprietary and “closed”. There do exist some network management tools that allow administrators to manage their networks from a central point, but these are by far and large operationally vendor-protocol and interface dependent. Programmable networking is essentially ushering in change at this level, trying to establish a level-playing field, attempting to ease innovation and speed up the pace of deployment. Software-defined networks or SDNs in short, are fundamentally different in two ways: one, they separate the control plane (that handles routing) from the data plane (which is responsible for forwarding packets according to routing decisions) and two, a single, centralized control plane is responsible for managing *multiple* data planes. We use 'SDN' and 'programmable/programmatic network' interchangeably throughout.

This shift to programmable networks has been years in the making, with a lot of so called technology pushes and applications pulls (terms borrowed from [49]). Software-defined networking can trace its roots back to various projects. The Active Networks research from the mid-1990s, such as Smart Packets [123] and ANTS [143] projects coincided with the time when then Internet exploded from just email and file transfer to a whole slew of new applications. Much of this research explored approaches such as carrying network programming information in-band [143] and the possibility of having programmable routers and switches [19]. The 2000s saw projects exploring data and control plane separation such as the ForCES RFC [38] and the Ethane [25] project. The seminal OpenFlow [101] proposal that kickstarted much of the current SDN revolution and the Open Network Operating System (ONOS) project [16] have all contributed significantly to what software-defined networking is today.

There are also other network virtualization projects that support the concept of programmable networks. The GENI [17] initiative built a national experimental infrastructure for researchers to experiment with virtualized networks. Mininet [77] goes the other way by emulating a network, complete with switches, routers and hosts on a single machine. The Open vSwitch [112] project built a complete multi-layered software switch for multiple hypervisor platforms. FlowVisor [128] explored the concept of slicing the network elements to provide isolation. These are some of the notable ones. There are many other projects that have contributed greatly to the current push for open-source, programmable networks.

The main thrust of this body of research is to support computer networks in their evolutionary endeavor of being autonomous entities. We do this by investigating how to enhance the quality of decision making for network administrators by utilizing machine learning algorithms to continually monitor and understand the network. This research is an attempt

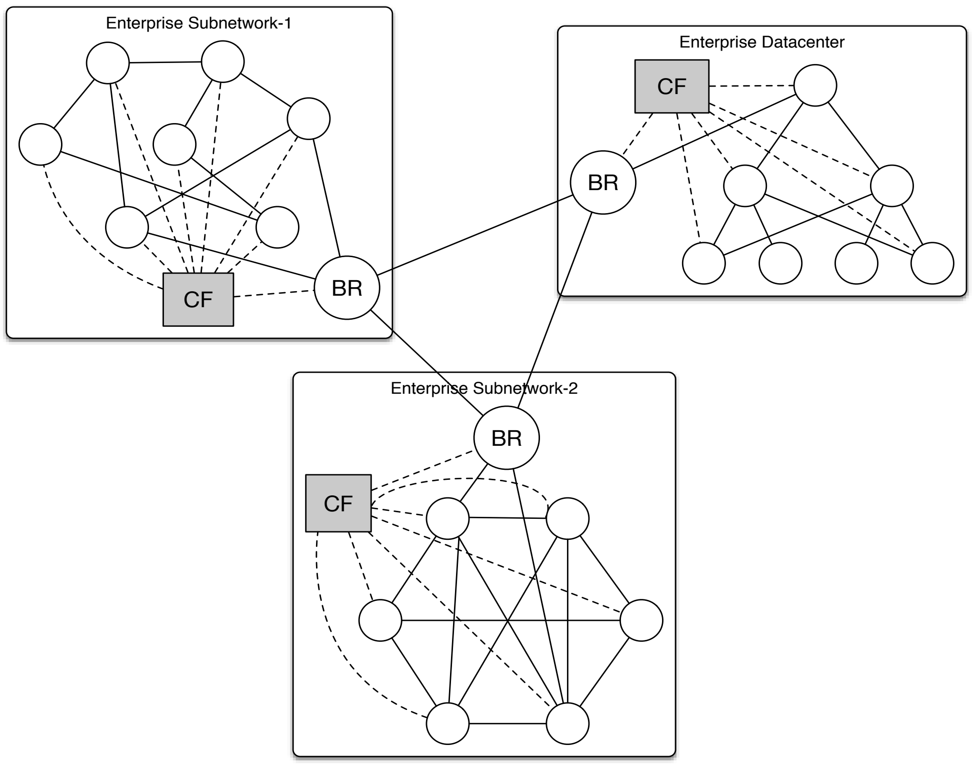


Figure 1.1: Example network architecture.

to integrate computer networks with machine learning and eventually contribute to this exciting new area of research in its own small way.

1.2 A Motivating Example

To understand the requirement for a robust management framework, let us consider Figure 1.1.

The figure depicts three sub-networks of a large enterprise network, analogous to two site offices and one data-center. “BR” nodes indicate border routers. Without the nodes marked “CF” and the dashed lines, this could very well represent a traditional enterprise network. Nodes, connectivity etc. are relatively stable and this allows for efficient management methodologies.

But if we do consider the “CF” nodes and the lines, they represent any programmable control framework on-site, such as Project Calico [24], Kubernetes [85] etc. that is responsible for managing the network (we are assuming a containerization scenario here). The same logic can be extended to include networks being managed by SDN controllers such as ONOS, OpenDayLight [16, 105] etc. VMs and containers in the data-center are scaled up or down based on demand. Enterprises now may not even require a dedicated network and could be run entirely in the cloud [66], and stability is somewhat an alien concept.

With legacy architectures, the inherent stability of the network allows for efficient management methodologies. The use of a virtualized environment changes the playing field. The environment in itself is indicative of an increase in dynamism (and a lack of stability thereof), but the type of virtualization actually dictates the extent of increase. These virtual networks play host to numerous services that are critical to an enterprise’s operation. VM-based environments (such as Amazon AWS, Google Cloud and Microsoft Azure, not to mention a plethora of other cloud environments) generate a lot of churn due to the movement of entities around the network. In such environments, the timescales are at least in the order of a few minutes (depending on the size of VMs). Containerization is the current favorite of Dev-Ops teams, and for good reason. These environments allow development, testing and deployment cycles at paces that were unheard of previously. Docker, LXCs etc based containerized environments move the timescales from minutes to seconds.

An administrator working with a highly virtualized environment has different needs and requirements. Even though the instability causes a lot of flux, what we can depend on in any network is that every device stores some state information for directing packets to the correct destination. It could come in various forms and formats, such as RiB and FiB entries, connection table information, ARP entries, interface information, logs etc. Any changes on the network *will* result in some change to this state information. Consider the case of a link failure and a backup link taking over in a SDN-controlled environment. This results in multiple changes of network state such as interface table, ARP table and RiBs changes. There could also be BGP updates triggered due to a new path becoming available. The controller is notified of every single one of these changes. What if this information was made available to an administrator in the event of a network fault? It could be potentially invaluable in identifying and remediating the issue even before it registers any significant user impact, not to mention the time it would save in root cause analysis.

1.3 Problem Statement and Approach

This dissertation focuses on exploring the possibilities of using network state to manage networks more efficiently by making better decisions. It focuses on two major aspects: troubleshooting, and routing network flows and as such comprises of two major parts.

We first investigate network routing algorithms for service-oriented networks, exploring the possibility of dynamically switching paths based on current network conditions. We then explore the potential for finding multiple paths through the network, based on multiple criteria, as opposed to single-criterion path-finding as is performed today.

As part of the second part of the research, we investigate the troubleshooting aspect utilizing network state intelligently. The research designs and develops a versioning system for network state, aimed at enhancing the troubleshooting experience for network administrators. It then looks into designing and building a robust, extensible network management framework for highly dynamic environments that interface with traditional architectures that operate at micro timescales. The goals of this framework are as follows:

1. Design a robust, pluggable network management framework that focuses on enhancing decision making, and
2. Design and demonstrate that natural language concepts can be used to understand and analyze network state, and
3. Design and implement models for the framework that are able to identify activities/events based on an event ontology. Relevant information is then extracted from

the identified events, thus contributing to building a knowledge-base of information that can then be exposed to tools built on top of it.

Activity recognition and topic modeling in particular has been explored extensively as detailed in Section 6.1. We look into the possibility of utilizing these approaches Supervised machine learning methods provide the benefit of identifying the most common activities that will be of importance, but they will also require large amounts of manual annotations to specify activities. Furthermore, the aforementioned activities of Section 1.2 are unique for each client and would be dismissed unless a sufficient number of examples were recorded during training. On the other hand, unsupervised methods will be able to develop their own clusters of network configurations for each presumed activity. The biggest challenge with such clusters is explaining them to system administrators who will have to interpret these activities perceived by the machine.

The approach we intend to take for the activity recognition algorithms is a balance between supervised and unsupervised machine-learning. We intend to facilitate this task by identifying potential activities that may occur throughout the network over time and presenting them to the system administrators. This should be able to assist them in making decisions about the network as well as what to do in the eventuality of any impending or predicted problems. We believe exploring various representations of the recorded information may identify regions of interest in the network that pertain to each recognized activity. Trivial representations will display the entire network with all its information. Flow-specific data and routing information can generate additional features that better describe network reachability and evolution for system administrators and typical users of network management software.

It is critical to design a system that is in symbiosis with methodologies that are bound to be used on the system [93]. As with any machine-learning related endeavor, data is of critical importance, that indirectly makes our data repository also a crucial component. We propose to identify a technology that suits our needs from a a networking perspective to serve as a data repository and adapt it to work for network state.

1.4 The Contributing Role of NSF Innovation-Corps

The pioneering entrepreneurship program from NSF, Innovation-Corps, has been instrumental in shaping much of the work being proposed here. This program was an interesting journey of learning what problems the industry is facing in the domain, and the impact research would have by addressing those issue. It began with the notion and assumption that versioning information for SDNs was a strong idea and worthwhile pursuing. The primary hypothesis fundamentally revolved around reducing the time required for root cause analysis for network administrators.

The initial set of customer interviews gave a clear indication of the industry's reception to programmable networking and SDNs specifically. While SDN technology is definitely on the upswing and, if solutions utilizing this are properly designed, it could save enterprises a lot of money. Unfortunately, a lack of mature solutions spanning a variety of issues is hindering adoption. One of SDN's greatest promises has been to provide network administrators and engineers with flexibility through programmatic control of their networks. To accomplish this, current solutions revolve around providing APIs and framework access to network engineers. However, these approaches do not consider an overburdened network engineer without the programming bandwidth to take advantage of flexibility through programmatic interfaces. This reason, in part, is one of our motivating factors to create a robust management framework that alleviates some of the burden being placed on network administrators.

A common refrain (observed across most of the I-Corps interviews) had to do with the tools available for network management. There are certain tools that perform some network management tasks very efficiently. But what network administrators are missing is the proverbial "The One Ring To Rule Them All." The feedback from interviews with network administrators, engineers and analysts focused on the need for a product that is able to collect, correlate and analyze information from these various monitoring tools, thus leading to meaningful inferences being built out of network state information.

Some of the value propositions gleaned from interviews that top potential customer wish-lists are:

- Network state and event information correlation is very important. It would be a definite value-add if the information can be collated and correlated from various sources over the network
- Providing insights as to ‘where’ and ‘what’ the problem is would be a big first step. The ‘why’ can gradually be investigated in later phases. For enterprises, answering the ‘where’ and ‘what’ immediately provides a toe-hold in the blame game, that being the ability to point a finger when things go wrong.
- Automation is the way forward. As networks evolve and become more complex, automating work-flows across the board takes center-stage.

The I-Corps interviews also helped identify a problem area. Pure SDN deployments are likely going to be rare. Even though SDNs claim savings on both capital and operational expenditures, it is definitely both impractical and cost-prohibitive to replace an entire enterprise network with new devices that support these architectures. A natural outcome of this constraint is the co-existence of programmable and traditional networks as hybrids. Based on these insights, we propose to translate this idea of utilizing state history and the power of machine learning to embed intelligence into the network itself. Consequently, the network can enable more efficient management because of the wealth of information it is providing the administrator to work with.

1.5 Dissertation Outline

The rest of this dissertation is organized as follows:

Chapter 2 discusses the advantages and downsides of programmable approaches to networking. We talk about the research space in general, and also touch upon the different facets of machine learning and how they apply to this problem.

Chapter 3 focuses on adaptively and dynamically finding routes through the network based on network conditions.

Chapter chapter placeholder here takes a slightly different approach, looking for the Pareto-optimal set of paths through the network based on multiple criteria, as opposed to the tried and tested single criterion, shortest path approach. This chapter discusses how this approach may be useful in the case of service-chain networks, as all the paths have to satisfy the service-chain ordering provided.

Chapter 5 presents our previous work on state versioning for programmable networks and how we envision it aiding management and troubleshooting. Most of this chapter deals with goal3 of Section 1.3.

Chapter 6 delves a little deeper into activity recognition and topic modeling. We also discuss how and how and why they may be applicable to our problem in general. Preliminary results of utilizing topic modeling on network state are presented here.

Chapter 8 is where all the pieces come together and form the final solution "An Intelligent, Pluggable Network Alerter Framework" as the final product of this dissertation.

CHAPTER 2

THE CHALLENGES IN MANAGING PROGRAMMABLE NETWORKS

While networking technology and architectures are evolving at a tremendous pace, the same cannot be said about network management. Although there have been significant improvements in the way networks are managed today, we need to fundamentally rethink management for environments that operate at micro timescales. It would be remiss to treat programmable networks similar to traditional architectures; it would also be a fallacy to conclude that legacy networks will be completely replaced by newer architectures. Since we are a potentially a long way from making networks autonomous entities that are capable of handling themselves, overcoming the challenge of successfully incorporating machine-learning methodologies into the management plane of the network may be an important step towards developing fully autonomous networks. But that may beg the question as to why the management/control plane?

With the demand for bandwidth always on the rise, it is imperative that the data plane deliver line rate speeds to whatever extent possible. In the case of programmable networks, there is already some latency introduced by control being moved to a software abstraction layer as opposed to custom designed ASICs. It would be in the best interests of network performance not to exacerbate this latency. But if we consider this holistically, the data plane does what the control plane instructs it to do, and the control plane itself is managed through the management plane. If there is one layer of abstraction that understands the network as a whole, it is the management plane. Especially in the case of programmable networks, where the control and management plane exist as a single entity, it logically follows that the best place to embed intelligence would be the management plane.

2.1 Mandatory Requirements

Most enterprise networks today are large-scale entities that have unique properties that distinguish them from other networks, which lead to many challenges are mandatory requirements. We tackle the most important of those requirements in this section. They are:

- *Scalability*: One of the major advantages that programmable networks bring to the table is their elasticity and ability to scale up or down on demand. An operational network may have tens of thousands of network nodes connected together (we postulate that the number of physical devices is brought down by virtualization, and we still will have millions of entities together in a connected network). Any management solution that we design should have the capability to handle networks of this magnitude.

- *Efficiency*: When we talk about this requirement, it is not only about performance efficiency, but also cost-efficient. In most cases, we want to design a solution that can be efficiently integrated into existing architectures without unduly affecting deployment costs. We also need to pay careful attention to our design since we do not want to introduce or exacerbate latency constraints.

- *Accuracy*: The level of accuracy and degree of confidence with which events are detected and analyzed strongly dictates how good, or conversely how terrible a system is performing. Since we are designing a system for networks that operate at very small time scales, we need to fine tune the performance on two fronts. It is not just enough to perform highly on the "true-positive or true-negative" front of the confusion matrix. It is also important to balance the "true-positive/negative" with the "false-positive" portion of the confusion matrix. A system is not going to prove very useful if it keeps popping up false-positive alerts at a high rate. It will only serve to detract the network administrator from focusing on actual true alerts.

- *Stability*: A good solution is expected to run for long durations accurately. We normally would have added a low human interference measure, but in our case here, we are designing the system to complement human actions, so human intervention is inevitable. On the contrary, we focus on incorporating these human decisions into the learning models so that they are eventually self-reliant.
- *Versatility*: The solution should be easily adaptable to different types of problems, which emphasizes our eventual goal of building a *pluggable, extensible* framework. Network administrators should easily be able to extend the system to recognize a diverse set of events.

2.2 Flexibility in Networks: A Double Edged Sword?

Programmable architectures are dynamic, manageable, cost-effective, and adaptable, making them ideal for the dynamic nature of today's applications. They have the capability to significantly lower capital and operational expenditure. For example, AT&T foresees a 40-50% reduction in operational expenditure when 75% of its network has been virtualized. A major portion of these cost savings will come from manual operations being moved to automation (which means network management is more automated). While SDNs can provide a such benefits to network providers and organizations, comfort with traditional architectures, habitual entrenchment with working solutions, and the lack of visibility resulting from the high network dynamics in programmable architectures impedes full-scale adoption.

The flexibility that allows network operators to control their networks are finer granularity and in a dynamic fashion is also responsible for the lack of visibility into network operations. This leads us to explore the next question about network visibility.

2.3 Visibility: How Much More is More?

There is a general consensus among the network administrator community about network visibility and it is about visibility always being inadequate. One of the most common responses to "how much visibility do you need into your network?" is "I will take everything you can give me and more". Administrators are forever on the lookout to increase visibility into network operations so that they can be operated at optimum efficiency. This makes the case stronger for a robust management framework that can deliver information in richer detail.

In contrast to fully automated solutions that are difficult to design for general case assessment, machine learning can be used to support the network administrator. Administrators will be able to understand causes of failures quicker and also receive indications of potential outages before they occur, in a manner similar to predictive analytics. When dealing with hyper-scale networks, the management problem increases multi-fold because the manual approach is very resource intensive. Thus, considering the complex nature of these networks and the rate at which they generate flux and churn, efficient management is heavily dependent on understanding what the network is doing. In order to develop a comprehensive understanding of the network, its activities and potential pitfalls, the "I will take everything you can give me and more" seems to hold good more than ever.

Thus, we qualify the requirement for a high-fidelity alerting and network management framework with enhanced intelligence capable of analyzing and predicting network events. The ability to correlate and interpret relevant information, in a near real-time fashion, will provide critical insights to network administrators and enable fine-grained control of the network. Our previous work [41] (dealt with in Chapter 5) describes a mechanism for versioning information in a network, but does not provide any solutions on how to reason about such information. We aim to address that gap, thus resulting in a more efficient and streamlined management of complex, programmable networks.

2.4 Statistical Machine Learning Methods

Statistical learning methods aim to extract useful information from data. These methods are useful for garnering insights in areas where expert knowledge is scarce, subjective or expensive to obtain. Statistical learning methods have already been used in various scenarios for traditional architectures [81, 130]. Some of the work by [122] still provides a strong basis for proactive network management techniques today. In this dissertation, we attempt to demonstrate that we can use newer statistical learning methods, albeit by rethinking and re-imagining solved network management problems as ones bounded by a different set of constraints and parameters.

Statistical methods are designed to extract knowledge from large amounts of data. This body of research proposes to version network state information (discussed in detail in Chapter 5). Network state information collected over an extended period of time will amount to large quantities of data. Also, this versioned information will prove useful in providing a historical perspective to the network and its temporal evolution. The inferences and correlations that statistical learning methods make on this entire dataset are bound to be more accurate and unbiased in comparison to rules inferred based on smaller sample spaces.

These reasons make statistical methods coupled with network state versioning a suitable fit for our management framework. It is also a secondary goal of this research to train models such that they are stable over time and across locations. The real advantages of this approach would come when models trained beforehand in testing and pre-production setups can be deployed in live production environments and demonstrate similarly high levels of recognition and predication accuracy with great degrees of confidence. This would result in a subsequent reduction in the time taken to move a network change into production.

In this work, we start with unsupervised learning methods to cluster data and learn what features and patterns that network is demonstrating. We then propose to gradually incorporate a network administrator's input to the model, thus transitioning to a middle ground known as semi-supervised learning.

CHAPTER 3

ADAPTIVE SERVICE-CHAIN ROUTING FOR VIRTUALIZED NETWORK FUNCTIONS

3.1 Introduction

Software-Defined Networking (SDN) is an emerging technology for controlling networks at a finer granularity than is possible in current networks. An SDN controller can set up the path of a flow through its network by configuring each switch along the way to match packets of this flow and forward them to a specific next hop [26, 95]. New flows can be set up dynamically when the first packet of a connection arrives at the edge of the SDN network (and the switch does not find a matching rule). The controller is then informed of the new packet and computes a suitable path for the packet. This allows for great flexibility in terms of network control and management.

An interesting aspect of SDN is the ability to not only consider forwarding of network traffic, but also its processing. Such "network functions" or "network services" [144] can implement header processing and payload processing functions, such as network address translation (NAT), firewalling, or virtual private network (VPN) termination. These functions, called virtual network functions (VNFs) can be implemented in software on conventional processing systems that are co-located with networking equipment. The sequence of functions that need to be set up for a specific flow is referred to as a "service chain."

In traditional IP networks, nodes that implement such functions need to be located in the path of traffic (e.g., a firewall at the ingress point to a network). However, in SDN, where the path of each flow can be determined independently, it is possible to virtualize the implementation of these network functions. Through network function virtualization

(NFV), the location of the processing node that implements the function can be anywhere within the SDN network or placed on a pre-computed path from source to destination.

This approach provides significant scalability benefits since multiple virtual nodes can implement processing-intense functions, such as VPN termination, for different flows independently in a distributed manner. However, multiple potential processing nodes pose interesting routing challenges. Specifically, these challenges are: (1) determining a flow path that traverses suitable processing nodes in the correct order to meet the requirements of a given service chain, and (2) considering network load and other dynamic characteristics when routing through existing VNFs.

We present an algorithm that can be used to compute paths efficiently, meeting all the requirements stated above. The primary idea is to transform the traditional routing problem in networks by merely modifying the structure of the network graph. Using a suitable metric that combines both link costs and processing costs, conventional shortest path algorithms can be used to solve this problem. To reflect dynamics, we exploit the ability of SDN controllers to collect run-time statistics from switches in the network to assess dynamic loads. We use models from queuing theory to translate this information into dynamic cost values for the placement problem.

Specifically, the contributions of this chapter are:

- Design of a Adaptive Service Routing (ASR) algorithm that uses a layered graph construction. Dijkstra's shortest path algorithm [37] is then used to determine the best path for a VNF chain.
- Use of cost functions that consider the dynamics of networks, such as load on links and processing nodes, to enable optimal dynamic path routing.
- Evaluation of the ASR algorithm to validate its correctness, both theoretically and on an SDN framework with ONOS [16], Mininet [77] and sFlow [125].

We believe that this work presents an important step toward realizing the full potential of network function virtualization in SDNs, especially in the context of dynamic adaptation under changing network loads.

3.2 Related Work

SDNs switch flows and flow aggregates rather than IP aggregates as in the current Internet [26,95]. Control mechanisms in SDN are centrally located in the SDN controller, rather than distributed across network nodes. The idea of service chains, as they are implemented in NFV today, have been alluded to in our prior work [42,144] and other approaches.

Path-finding in network graphs is an important problem and many efficient path-finding algorithms have been proposed. Our ideas for using a layered graph in Adaptive Service Routing are based on prior work in [33]. The work presented here expands on this by considering the dynamics of SDNs and how to implement such an approach in practice.

Placement of virtual network functions has been explored in [55]. Their approach considers more complex metrics, such as host resource consumption, but represent also a more complex solution. Similarly, multi-criteria path finding techniques consider multiple metrics, e.g., [30]. We, however, focus on a single metric, since this approach leads to efficient and effective solutions to VNF placement.

3.3 Order-Constrained Network Function Routing Problem

Network functions implement operations on network traffic, specifically on the packets that are transmitted over the network. The operations are dependent on the type of traffic. We assume that this distinction is made at the granularity of flows and flow aggregates, both of which can be represented by match-fields defining the values (or wild-cards) of different header fields. Within a network controlled by a single administrative entity, the problem is to determine a path for a flow that connects its end-points (either end-systems or ingress or egress points to the network) and traverses a set of nodes where network functions exist or can be instantiated.

3.3.1 Problem Statement

The network can be represented by a graph $G = (V, E)$ with n vertices, $V = \{v_1, v_2, \ldots v_n\}$ and m directed edges $E = \{(e_i, e_j), \ldots\}$ connecting these vertices. Each edge (e_i, e_j) has an associated cost $c(e_i, e_j)$. We define the set of t network functions that are implemented in this network as $F = \{f_1, f_2, \ldots, f_t\}$. Each vertex may implement some of these network functions, which are specified by the subset of F provided by function $f(v)$. A connection, $c = (v_s, v_t, (r_1, r_2, \ldots, r_k))$, is specified by the starting vertex, v_s, the terminating vertex, v_t, and the sequence of k requested network functions, $(r_1, r_2, \ldots, r_k)$.

The network function placement problem is to determine a mapping, $M = (a(\cdot), p(\cdot, \cdot))$, that consists of an assignment function, a, and a path function, p. The assignment function, $a(r_i)$, assigns each function, r_i, of a connection to certain vertices in the graph. The path function, $p(a(r_i), a(r_{i+1}))$, determines the sequence of vertices traversed between functions (and between the starting vertex and the first function and the last function and the terminating vertex). When we say assignment, this does not mean that we instantiate the VNF on a particular node. It means that we have prior knowledge of where the VNFs are located and we map them to their respective nodes.

A path $P = p_1 \ldots p_l$ computed by the path function is considered to be a valid, admissible path that services the virtualized function chain if there exist integers $i_1 \ldots i_l$ such that $1 \leq i_1 \leq \cdots \leq i_l \leq n$ and the function provided at p_{i_j}, $f(p_{i_j}) \in f(v)$ for $i \leq j \leq l$.

Transforming the graph by layering is to reduce the complexity of a problem that is combinatorial, thus allowing shortest path algorithms such as Dijkstra's algorithm to be run on the modified graph. Then, running a shortest path algorithm from the source vertex in the topmost layer to the destination vertex in the bottom layer reduces a possible exponential growth pattern down to one that is sub-quadratic in nature. The graph transformation does not aim to do anything more than achieve this complexity reduction. The other alternative would be to engineer a complex solution that solves the combinatorial problem as-is.

3.3.2 Constraints and Optimization Metrics

We restrict ourselves to a path-finding algorithm that uses a single criterion (multi-criteria path finding approaches are briefly discussed in Section 3.2). The constraints presented by the VNF routing problem are twofold: (1) find a path from source to destination with least cost, and (2) maintain the order of services requested for provisioning.

A critical aspect of the Adaptive Service Routing algorithm is that the metric captures cost for both communication and processing. There are many potential choices for such a cost (e.g., actual financial cost of use, delay, probability of meeting QoS requirements, etc.) and more complex multi-criteria cost functions could be used. For our work, we use *delay* as the single cost metric for both communication and processing. The use of delay yields practical solutions since it is an important consideration in many networks. Also, delay lends itself well to representing dynamic loads on network links and on VNF processing nodes.

3.4 Adaptive Service Routing

This section describes how the network graph can be transformed to help solve the complex service-chain routing problem using conventional methods.

3.4.1 Network Graph Transformation

We assume that $FC = f_1, f_2 \dots f_k$ denotes the network function chain that needs to be realized on the network topology, where $FC \subset F$. We transform the network graph into a "layered graph" by adding k layers to the graph (counting the existing graph as the base layer), where $k = count(FC)$ and each layer is an exact copy of the original graph. For every vertex v in the original graph, let v^i denote the corresponding node in the i^{th} layer ($i = 0 \dots k$). Every $(i-1, i)$ layer pair is connected vertically only by edges between nodes

v^{i-1} and v^i if that node provides the particular network function required by f_i. These vertical, directed edges are weighted by a cost $c(f, v)$ that is defined by the processing cost for utilizing network function f on node v.

Figure 3.1 illustrates this transformation. In this example, we consider a service chain with two functions, i.e., $F = f_1, f_2$. Function f_1 is available at node B, H, I and f_2 is available only at node F. Consequently, we construct three layers of the original graph. The first layer is the base layer of the network topology. The second layer is connected to the first layer only through nodes that provide processing for f_1, in this case, nodes B, H, I. Finally, the last layer is connected to the second layer through node node F that provides processing for f_2. These edges are weighted by their respective function processing costs.

3.4.2 Adaptive Service Routing Algorithm

It is always beneficial to reduce a problem to a certain state that can then be solved by an existing, optimized algorithm than formulating a completely new algorithm. In our approach described above, a conventional shortest path algorithm requires a slight modification to perform path-finding. The source node src is placed in the first layer of the layered graph and the destination node dst is placed in the $k + 1^{th}$ layer. Shortest-path routing is then conducted across all layers from src to dst.

In the example in Figure 3.1, the goal is to find the shortest path from node A to node J. After constructing the layered graph, we find a path from node A^0_{src} in the first layer to node J^2_{dst} in the last layer. The structure of the layered graph forces traversal of vertical edges in order to reach the destination in the last layer. (Since edges are directed, no loops across layers can occur.) Here, nodes B, H, I provide processing for f_1 and hence $(B^0, B^1), (H^0, H^1), (I^0, I^1)$ are the only edges that connected the first and second layers. The same argument applies to the second and third layers that are connected by (F^1, F^2).

The cross-layer path that Dijkstra's returns can be mapped to a path in the original graph by folding nodes with layer qualifiers (the superscripts) back into their base nodes. Here,

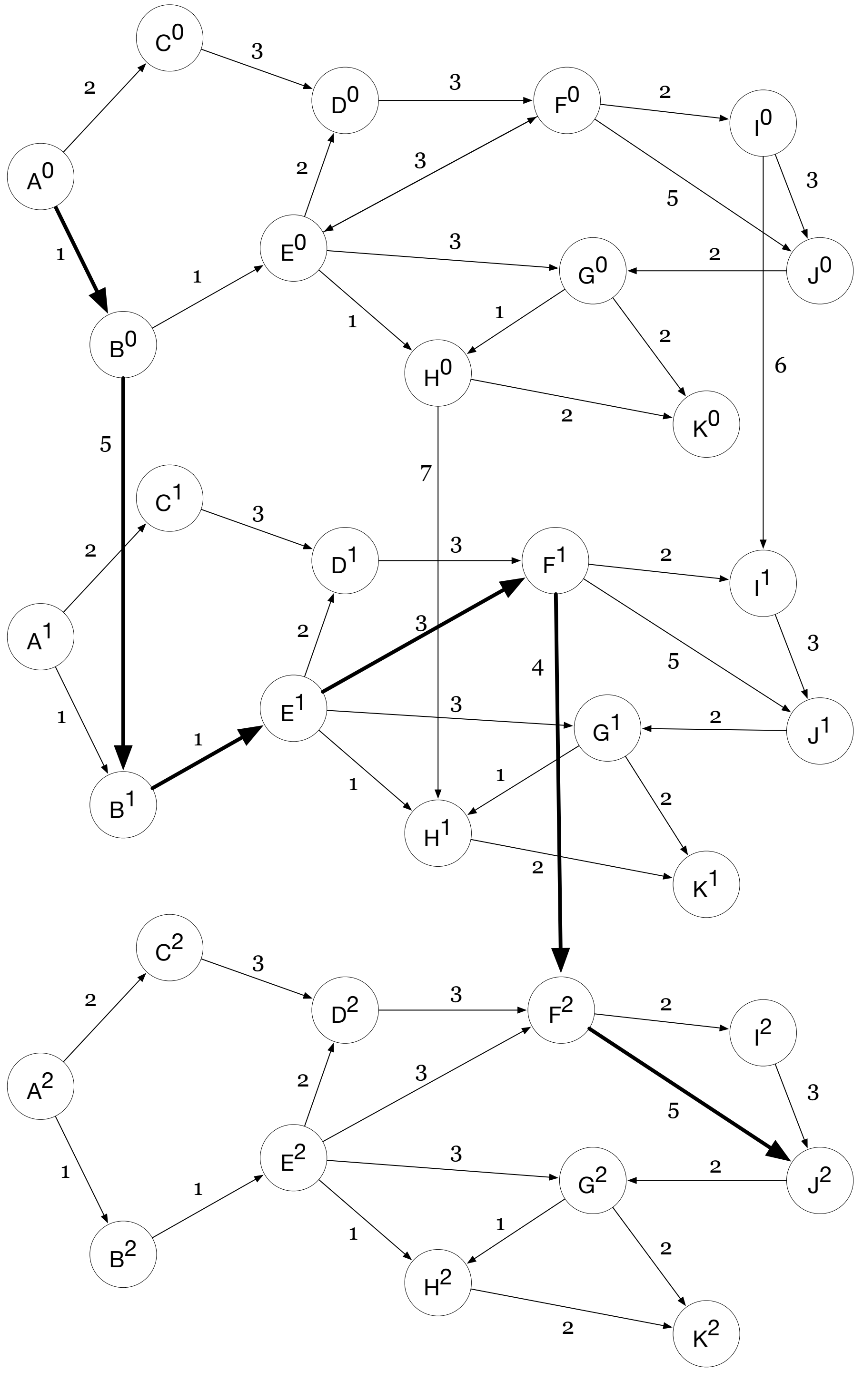


Figure 3.1: VNF processing represented on different layers.

Dijkstra's algorithm returns $A \to B \to B^1 \to E^1 \to F^1 \to F^2 \to J^2$, indicating that node B was picked for processing $f_1 \in FC$ and node F was used for processing $f_2 \in FC$. This path, nodes in lower layers overlaid on their originals, $A \rightarrow B \rightarrow E \rightarrow F \rightarrow J$, with the added processing costs at nodes B, F for f_1, f_2 respectively.

The proof of correctness is inherent in the methodology itself. The minimum cost path from the src node in the base layer to the dst node in the $k+1^{th}$ is an overlay of an actual path in the original, untransformed network graph. The costs of both these paths are the same, since corresponding edges in all the layers have the same edge weights, which implies that there cannot be another least cost path. Furthermore, if there were some other least cost path, then it would have to translate into the same path over all the layers as discussed previously, which is in clear contradiction of the least cost path obtained by Dijkstra's.

3.4.2.1 Communication Delay

The communication delay for traversing a single link consists of the following three components: (1) queuing delay d_{queue}, (2) transmission delay d_{tx}, and (3) propagation delay d_{prop}. The transmission delay is

$$d_{tx} = pktsize/bw,$$

where $pktsize$ is the size of the transmitted packet and bw is the link bandwidth. The propagation delay is

$$d_{prop} = l/c_{medium},$$

where l is the physical length of a link and c_{medium} is the propagation speed of signals in that medium.

The queuing delay at a node is dependent on the egress interface load, denoted by $load_{link}$. The load is the ratio of allocated bandwidth (from previous connections) to the total link bandwidth. Using a simple M/M/1 queuing model with an expected service time of d_{tx}, we get a queuing delay of

$$d_{queue} = load_{link}/(1 - load_{link}) * d_{tx}.$$

We use a superscript to denote the delay values for a specific node and edge. The total cost associated with an edge is thus $c(e_i, e_j) = d^{ei}_{queue} + d^{ei,ej}_{tx} + d^{ei,ej}_{tx}$.

3.4.2.2 Processing Delay

The nodal processing delay is time taken to process a packet on the network node. The advent of powerful network processors with their multi-processing capabilities etc. has managed to keep the processing delay under a manageable threshold on traditional network nodes, despite the increased time it takes for complex packet processing functions.

We can model processing delay based on Generalized Processor Sharing (GPS):

$$d_{proc} = (load_{processor}/load_{flow}) * t_{proc},$$

where $load_{processor}$ is the current load on the processor expressed in percent, $load_{flow}$ is an approximate fraction of the load the flow will contribute to the processor, and t_{proc} is the per-packet processing time (e.g., from experimentation or equations in [116]).

In practice, balancing load and avoiding allocations that utilize systems at their maximum capacity may be desirable.

$$d_{proc} = load_{proc}/(1 - load_{proc}) * t_{proc}.$$

An M/M/1 queuing model captures the nearly linear growth in delay for low loads and associated high costs near system capacity. We use this for our hypothesis. Other queuing models may be used as necessary.

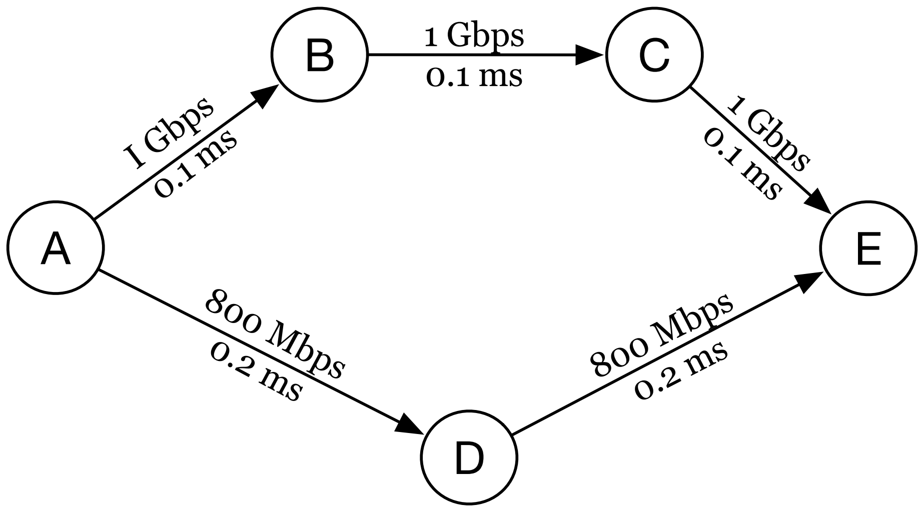


Figure 3.2: Network topology for example scenario.

3.5 Evaluation

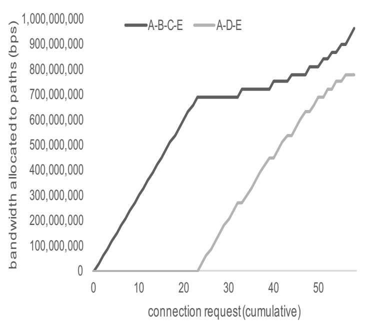
In our evaluation, we first present a simple scenario to show the validity of our approach. We then discuss our implementation on an emulated framework.

3.5.1 Theoretical Model

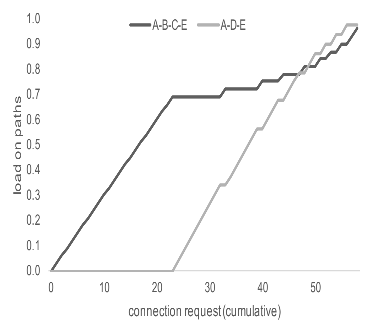
To demonstrate the effectiveness of our proposed approach, we show its operation on a small network topology. In this example, the various effects of resource load, delay modeling, and node and path choices are illustrated.

The network topology in Figure 3.2 shows the performance characteristics of the links and processing nodes. We assume that connections need to be routed from A to E and a VNF needs to be used (available on nodes B, C, and D).

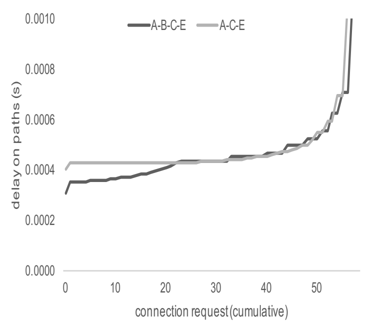
Figure 3.3 shows results of repeatedly mapping a 30 Mbps service request between nodes A and E onto the topology. The first 23 connection requests are assigned to the upper path ($A - B - C - E$) due to the lower latency on that path. Note that processing allocation is split between nodes B and C proportional to their capacity. As the load increases on the upper path, the lower path ($A - C - E$) is preferred and requests 24 through 56 are



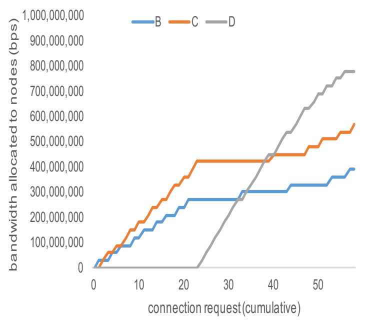
(a) Path allocation.



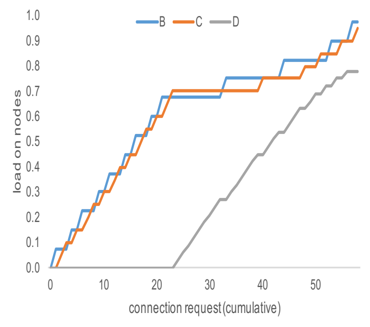
(b) Path load.



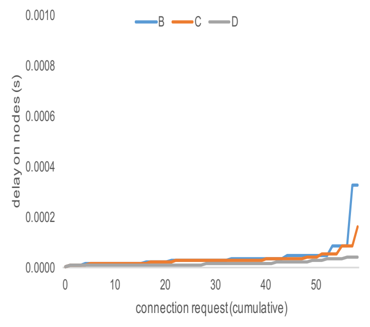
(c) Path delay.



(d) Node allocation.



(e) Node load.



(f) Node delay.

Figure 3.3: Performance characteristics of repeated service routing on example topology.

allocated to both paths (with the majority going to the lower path). As the network reaches its capacity, the final requests are allocated to the top path.

Figure 3.4 compares our algorithm with a number of other routing techniques. All approaches accommodate fewer connections (random: 43, round robin: 48, static B: 13, static C: 19, static D: 26) than our Adaptive Service Routing algorithm with 58. Also, our algorithm consistently achieves lower delays across all mappings (random and round robin achieve some lower delays in later mappings because earlier ones use resources on clearly suboptimal paths.)

The graph shows what the overall end-to-end delay is for the most recently allocated flow. Since the random allocation scheme chooses arbitrary paths, the delay may be short for one and long for another. With our adaptive approach, there is a small bump, because

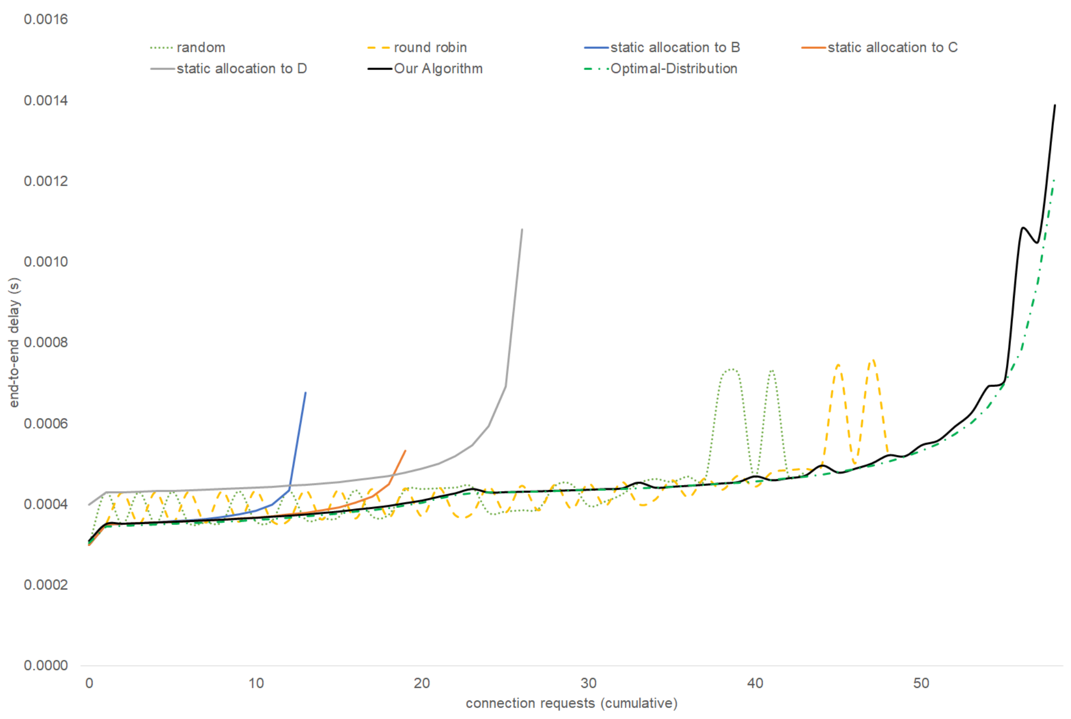


Figure 3.4: Comparison of VNF routing algorithms.

the algorithm looks at the least delay *before* allocating the flow. Once allocated, there may be some additional delay (due to capacity limits), which makes the curve fluctuate slightly.

If we were to consider optimal allocation of flows, on approach would be to distribute the load across the paths as evenly as permissible. This would mean splitting the flow up such that the utilization on each path is equal. We could consider this as a theoretical bound on how well flows can be allocated. This is illustrated by the “Optimal-Distribution” curve in 3.4. As we can see from the plots, our algorithm follows this bound consistently, thus demonstrating that we are allocating flows in a practically dictated optimal manner.

We also measure the processing time required by the algorithm to make path-finding decisions. The topologies chosen here are Waxman, Barabasi-Albert and GLP models of different sizes, generated using the BRITE topology generator [97]. Processing has been simulated for 3, 5 and 7 services in the VNF chain. Table 3.1 summarizes the results. We observe computational times in the order of half a minute for graph sizes of 10K nodes

Table 3.1: Adaptive Routing Path-Finding Times.

	Processing Time (s)		
Topologies	3 VNFs	5 VNFs	7 VNFs
Waxman-100	0.02	0.05	0.09
Waxman-1K	0.23	0.59	1.09
Waxman-10K	3.24	8.25	15.30
BA2-100	0.04	0.09	0.17
BA2-1K	0.46	1.16	2.12
BA2-10K	6.69	16.97	30.97
GLP-100	0.03	0.08	0.14
GLP-1K	0.41	1.02	1.85
GLP-10K	5.42	13.99	25.82

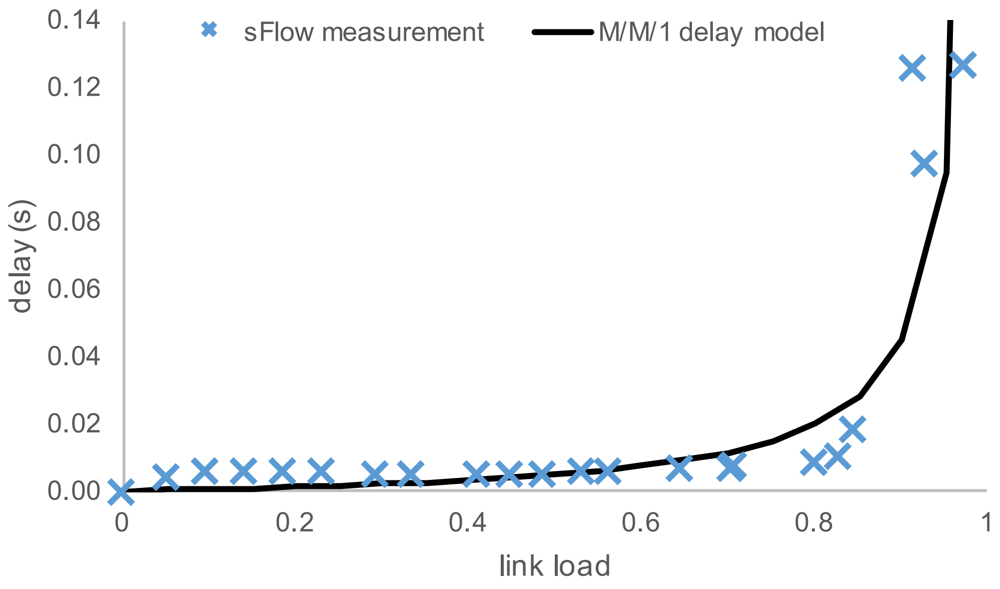


Figure 3.5: Delay model validation against sFlow data.

and 7 VNFs chained together. It would be beneficial to note that these are large sized graphs with multiple VNF site deployments. Most of the computational time is taken up by layering (which is currently non-parallelized). We believe they can be lowered further using graph optimization and parallelization techniques.

3.5.2 Prototype Implementation

We have implemented this service routing technique on a practical, emulated software-defined network running in Mininet [77] and controlled by ONOS [16]. Linux's TCLink library was used to define and constrain bandwidth and delay on the links. Due to Open vSwitch and Mininet limits on our hardware, emulated testing could be performed only for topologies with 60-70 switches and 20 hosts. Processing times for large topologies have been evaluated only for the algorithmic section as described in Section 3.5.1. We contend that the crux of the decision making is the graph transformation and the emulated testbed is just a method of instrumentation. We did observe that the practical scenarios do indeed follow our theoretical results for smaller topologies.

We use *sFlow-RT* and *host-sFlow*, instrumented flow sampling mechanisms of the sFlow standard [125], to obtain utilization of links and nodes. Figure 3.5 shows that dynamic network data matches our theoretical assumptions and the results from the small topology in Figure 3.3.

3.6 Summary and Conclusion

In this body of work, we presented an Adaptive Service Routing algorithm that routes traffic adaptively through various VNF nodes in an SDN based on instantaneous network latency. We use an example to show how this adaptive routing can help balance loads on VNF processing throughout the network. Due to the use of a single metric and translating the routing problem into a simple shortest path problem on a larger graph, our algorithm can prove to be useful in large-scale networks with multi-site VNF deployments.

CHAPTER 4

MULTI-CRITERIA ROUTING IN NETWORKS WITH PATH CHOICES

4.1 Introduction

When the concept of computer networking emerged, the basic intention was to enable resource sharing. Computer networks have evolved both in scale and complexity to the diverse, interconnected web of devices they are today. With data transmission costs being high previously, minimizing the path length was one of the main motivations behind path finding. There have been several seminal algorithms in the this area, with Djikstra and Bellman-Ford laying much of the foundation for the shortest path algorithms that are in use today.

Networks have grown significantly since then and the number of devices, protocols, constraints and guarantees handled by them have also increased proportionally. Single-objective shortest paths no longer fit the whole spectrum of services that exist in today's networks. Many applications require QoS delivery guarantees and have rigorous constraints on delay, cost, and many other parameters [28]. Application-centric networking, enabled by Software Defined Networks (SDNs), is an emerging force in current day networks. Decoupling the control plane from the data plane using SDN gives networks the required flexibility to gradually move from Infrastructure-As-A-Service (IaaS) models to applications as services. This gives applications the power to define their QoS requirements, security and access policies, deployment scenarios, etc. In such a scenario where each application's requirements and constraints are different, modeling these networks and applications on a single objective or constraining routing to a single, shortest path between source and destination would be a practical approach.

Routing a path for traffic to flow between communicating end-systems, is a core functionality of any computer network. Typically, routing is single criterion based, utilizing metrics such as path length, delay, or an artificially defined "weight." Popular routing protocols, such as OSPF [98], RIP [60] etc, use individual metrics and corresponding path-finding algorithms, such as Dijkstra's algorithm [37] and the Bellman-Ford algorithm [15], efficiently connect two networked nodes..

In networks where the user has the ability to choose paths [27, 145], the cost of a path and quality of a path need to be represented by independent metrics. Additionally, load-balancing and the use of backup paths for redundancy guarantees provided by protocols require multiple paths to be determined [73]. In these cases, criteria beyond the shortest path metric may be useful in determining which paths are functionally viable (e.g., available bandwidth, path reliability, etc.).

With a single metric, an optimal solution can be determined efficiently, meaning that single criteria shortest path algorithms have low complexity, such as Dijkstra's, which is linear in nature. In comparison, their multi-objective counterparts have been shown to NP-hard [59, 67] and tend to exponential in terms of complexity [136]. Subsequently, due to the complex nature of the problem, prior research in this area by Papadimitriou et al has shown that a $(1 - \epsilon)$ Pareto curve can be constructed that is an approximation of the superset of Pareto optimal solutions [137], in lieu of determining an optimal solution that may well be intractable. This means a set of paths can be found that represent the trade-offs among criteria. A key challenge here is the need to develop an efficient algorithm for determining suitable paths in a potentially large problem space (exponential to the number of nodes). Although multi-criteria path finding is NP-hard, it is possible to develop solutions for typical-sized networks that work well in practice.

Using a single, combined metric simplifies the path finding problem, simultaneously placing a fundamental limit on the ability to find solutions: a single optimization metric requires *a-priori weighing* of each metric [43]. Elaborating further, before the path finding algorithm is run, the relative "value" between different metrics needs to be set. The result of the search is then optimal (only) for this fixed weighing of metrics. In practice, however, there are situations where this weighing cannot be done a-priori. For example, a network may allow users to choose a specific path based on price and quality characteristics [27, 145]. In the marketplace of such a network, different paths need to be offered before knowing the users' sensitivity to price and quality (without any knowledge of weighting criteria). Similarly, in a Software-Defined Network (SDN) [106], an SDN controller may pre-compute paths requested by an SDN application without knowing the weights of metrics on the data plane. In such cases, the weighting can only be done *a-posteriori* and the multi-criteria optimal path problem needs to find the set of *all Pareto-optimal* paths. A path is Pareto-optimal if there is no other path that is better across all metrics. Since multiple metrics permit the existence of paths that are better than other paths in one or more metrics, but not all, there can be a large number of mutually Pareto-optimal paths. Based on the set of Pareto-optimal paths, one path can be chosen by an SDN application or by a network customer. In our work, we address this multi-criteria routing problem, which is important in practice, especially in environments where different metric weights are not known beforehand.

We present ParetoBFS, based on the well-known breadth-first search (BFS) algorithm. This algorithm uses Pareto constraints to prune the tree during traversal. Experiments show that ParetoBFS can find *all* Pareto-optimal paths in a network in a reasonable time since typical-sized networks do not exhibit the characteristics that cause the problem space to become intractable. The specific contributions of our work are:

- Design and develop the ParetoBFS algorithm that can find the entire set of Pareto-optimal paths in a network where the edges have arbitrary number of metrics, both sum, and bottleneck-type. Comparisons with two existing algorithms shows ParetoBFS is 10-100X faster and finds more paths.

- Demonstrate that the path stretch, defined as the ratio of a path in the Pareto-optimal set to the shortest path, is not extremely high, but in fact is only greater than the shortest path by a hop or two, at the most.

- Develop a sampling heuristic for ParetoBFS that reduces the number of elements in the set of Pareto-optimal solutions and thus decreases the complexity of the path finding process. We show that despite not yielding all optimal solutions, this heuristic still obtains solutions that are useful in practice.

- Illustrate the feasibility of this approach with results from simulation and implementation of ParetoBFS in an SDN-based prototype.

We believe that this work provides a practical foundation for systematically using multi-criteria routing in software-define networks to develop more effective network control applications in the future.

The remainder of the chapter is structured as follows. Section 4.2 describes background in the area. Section 4.3 formalizes the description of the multi-criteria path finding problem. Section 4.4 describes the ParetoBFS algorithm. Section 4.5 presents the complexity analysis and experimental results. Section 4.6 introduces several sampling heuristics for ParetoBFS. Section 4.8 compares ParetoBFS with related work. Finally, Section 4.9 describes an SDN-based prototype that uses of this algorithm. We summarize and conclude this body of work in Section 4.10.

4.2 Background

Multi-criteria path finding has been studied extensively in the operations research community. This problem arises in many practical applications, including route planning in traffic networks [14] and QoS routing and traffic engineering in communication networks [139]. Multi-constrained path optimization (MCPO) aims to find the optimal path with some constraints on one or more metrics given a directed graph with edges that have a set of metrics [29, 44, 76, 82, 137, 140, 147].

Previous work has addressed the multi-criteria optimal path problem in various contexts. for example, QoS routing. A central problem in QoS routing is to find feasible paths between a source and a destination that satisfy multiple constraints (e.g., bandwidth, delay). Then, the best path among the feasible paths is selected based on a given optimization metric (e.g., delay-constrained least-cost path routing). When there are multiple optimization metrics, most approaches rely on an combinatorial optimization function [82], which combines all metrics into a single metric (e.g., weighted sum). Without the constraints on the metrics, this problem then becomes the multi-criteria optimization (MCO) problem [43, 59, 82, 110]. Solutions to MCPO and MCO are usually similar in that they use a combinatorial function on the multiple metrics (a-priori) to find the optimal path.

Since the goal of ParetoBFS is to find all the Pareto-optimal paths, it is therefore a broader solution to address both MCPO and MCO problems since the resulting paths from previous approaches are usually a subset of the Pareto-optimal path set. These Pareto-optimal paths are important in many scenarios. For example, references [73] and [47] each describe a standalone routing service module that provides paths for other modules. Thus, the routing service module itself cannot make any choices for metric preferences. Also, in networks where paths are charged by their qualities, such as ChoiceNet [145], the cost and the quality of a path need to be represented by independent metrics. In these problems, there is no single objective function to select the best path, and it is impossible to give the paths an a priori ranking. Instead, the decision maker needs to see all the *Pareto-optimal* paths. Each Pareto-optimal path represents a trade-off between criteria, and may be equally important for the decision entity.

Section 4.8 compares the performance of ParetoBFS with some prior work in detail.

4.3 Problem Statement

Before describing the ParetoBFS algorithm in Section 4.4, we briefly introduce the network modelz and describe the formal definition of our path finding problem.

4.3.1 System Model

Suppose we define a graph as $G = (V, E)$, where $V = (V_1, V_2, ..., V_m)$ is a set of all nodes or vertices of the graph and $E = (E_1, E_2, ..., E_p)$ is a set of all edges of the graph. We model the network as a directed graph and n and m are the cardinalities of V and E, i.e., $n = |V|$, $m = |E|$, respectively. To make the problem general enough, we consider that G is a directed multi-graph (in practice, these multiple edges can correspond to different services that are offered on the same physical link, such as different QoS configurations).

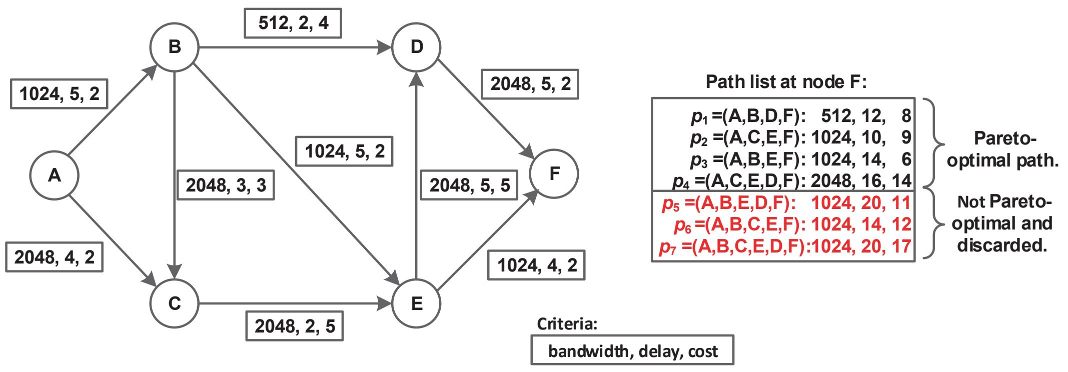
The following points are then true for the graph G:

1. $\{e_{u,v}|u, v \in V\} \in E$ is associated with an edge criteria vector $w(u, v) = (w_1, w_2, \ldots, w_k)$, where k is the number of criteria. This is edge characterization. Each w_i corresponds to one of the independent criteria used in routing, such as bandwidth, latency, packet pass rate, cost etc. The constraints could be any property that needs maximization or minimization.

2. A path P from a source v_1^p to a destination v_r^p is defined as a finite sequence of edges that connects a sequence of vertices $(v_1^p, v_2^p, ..., v_r^p)$, $v_{i(i \leq r)}^p \in V$.

3. A path P can be assigned a path criteria vector $w^p = \{w_1^p, w_2^p, ..., w_k^p\}$, where each w_i^p is a criterion for the network routing problem.

4. A path P_i is said to dominate a path P_j if and only if $(P_i(w_1^{P_i}) > P_j(w_1^{P_j})) \wedge (P_i(w_2^{P_i}) > P_j(w_2^{P_j})) \wedge \cdots \wedge (P_i(w_k^{P_i}) > P_j(w_k^{P_j})), \forall i, j \in \wp(P), \forall k \in w^p$. This means when calculating the path criteria vector, if a hop is added to the path, the optimality of the new path does not increase on any single criterion. This is the optimality condition.



A
B
C
D
E
F
1024, 5, 2
2048, 4, 2
512, 2, 4
2048, 3, 3
1024, 5, 2
2048, 2, 5
2048, 5, 5
2048, 5, 2
1024, 4, 2
Path list at node F:
p1 =(A,B,D,F): 512, 12, 8
p2 =(A,C,E,F): 1024, 10, 9
p3 =(A,B,E,F): 1024, 14, 6
p4 =(A,C,E,D,F): 2048, 16, 14
p5 =(A,B,E,D,F): 1024, 20, 11
p6 =(A,B,C,E,F): 1024, 14, 12
p7 =(A,B,C,E,D,F):1024, 20, 17
Pareto-optimal path.
Not Pareto-optimal and discarded.
Criteria:
bandwidth, delay, cost

Figure 4.1: Example of Pareto-optimal path computation from node A to F.

5. Criteria satisfying this property can usually be classified into two types: sum-type criteria (e.g., delay) where $w_i^p = \sum_{e_{u,v} \epsilon p} w_i(u,v)$; and bottleneck-type criteria (e.g., bandwidth) where $w_i^p = min(w_i(u,v))$. [1]

6. Each $P_i = (V_{i_1}, V_{i_2}, .., V_{i_r}) \in P$, $V_{i_{j \leq r}} \in V$ and no path P_i dominates any other path in P as per the optimality condition. This ensures that P is a set of only the candidate paths that are optimal in nature.

4.3.2 Pareto-Optimal Paths

Here, we further refine and formalize what Pareto-optimality means. For this, we first define a *dominant path* as follows. We use the notation $\geq$ to denote the left operand is more optimal than or equals to the right operand.

Definition 1. (Dominant path)*: A path* p *dominates path* q $\iff$

$$w_i^p \geq w_i^q, \forall i \in \{1, 2, ..., k\}.$$

[1]There also exist multiplicative criteria (e.g. link reliability, packet loss rate), but they can be transformed into additive criteria using a logarithmic function.

and the strict inequality holds at least once.

Then we can define a Pareto-optimal path-set as:

Definition 2. (Pareto-optimal path-set) *A path set P is called a Pareto-optimal set* $\iff$

$$p \textit{ does not dominate } q, \forall p, q \in P.$$

A path in a Pareto-optimal set is called a Pareto-optimal path.

Our goal is to find all the Pareto-optimal paths from a source node to a target node in a given graph G. For instance, if each edge $e \in E$ has three metrics: bandwidth (w_1), delay (w_2) and cost (w_3), then, in the set of the Pareto-optimal paths P, for $\forall p_i,\ p_j \in P, w_1^{p_i} > w_1^{p_j} \vee w_2^{p_i} < w_2^{p_i} \vee w_3^{p_i} < w_3^{p_j}$. This is different from the conventional multi-constrained optimal path problem [82], where a path optimization function f^p is used to combine all the metrics together and the optimal path is found by calculating the value of f^p on each path. As discussed above, the optimal path computed based on a single aggregated metric may not conform to the multiple constraints being considered.

Use-cases that impose a strict ordering on the service plan specifically require optimal paths to satisfy a certain set of services in a particular order. We further define a service plan as set of services $S = (S_1, S_2, .., S_k)$, where each $S_i \in \wp(S)$ is a service that may transform data or pass it through untransformed. All $S_i \in S$ are strictly ordered.We define a function of services, $f_S : V \mapsto \wp(S)$ which is to be used as described:

1. $\forall(a < b \leq k), S_a \in f_S(V_{i_c})$ and $S_b \in f_S(V_{i_d}) \Rightarrow i_c \leq i_d$. This represents the property of service ordering, that dictates that a service has to be performed before its successor can operate on data.

2. $\forall(0 < a \leq k), \exists\, b$ such that $S_a \in f_S(V_{i_b})$. This represents the condition of all services being satisfied.

Fig. 4.1 illustrates the approach that we will be taking here. The edges of the graph are labeled with their respective metrics comprising of bandwidth (w_1), delay (w_2) and cost (w_3).[2].

There are seven paths $(p_1, p_2 ..., p_7)$ from source node A to destination node F. In that path-set, path $p_2 = (A, C, E, F)$ is strictly more optimal than path $p_5 = (A, B, E, D, F)$, since $w_2^{p_2} < w_2^{p_5}$ and $w_3^{p_2} < w_3^{p_5}$ while $w_1^{p^2} = w_1^{p^5}$. As a result, path p_5 is not a Pareto-optimal path and would be subsequently discarded during path pruning. Similarly, neither p_6, nor p_7 are Pareto-optimal paths because p_2 and p_3 are strictly more optimal than them. Finally, we obtain the Pareto-optimal path-set $p_{1\sim4}$. (In the ParetoBFS algorithm, we maintain a list on each node to record all the Pareto-optimal paths to this node and their corresponding parameters. Such a list is shown in black on node F in Fig. 4.1.)

4.4 ParetoBFS: Path-finding in a Multi-Criteria Environment

In this section, we first discuss a basic breadth-first search (BFS) based solution to the multi-criteria path-finding problem. Successive sections describe how we utilize pruning (enabled by branch and bound) to reduce the running time of the algorithm to a more practical, acceptable outcome.

4.4.1 A BFS-Based Brute Force Solution

One possible solution to the multi-criteria path finding problem is to enumerate all the possible paths, then extract the Pareto-optimal set from them. The potential problems associated with this approach are long runtimes for large graphs and post-processing that contributes to the complexity. Nevertheless, we explore that solution because the subsequent ParetoBFS algorithm is based off this approach, with some extended logic to support branch-and-bound.

[2]These are just the parameters in our discussion, but the constraints could be any quantifiable parameters.

Algorithm 1 Brute-force-based solution enumerating all simple paths, post-process to eliminate non-Pareto-optimal paths.

```
1: procedure BFS($G, source, target$)
2:     for all $v \in G(v)$ do
3:         $path\_set[v] \leftarrow \emptyset$
4:     end for
5:     $path\_queue.push([source])$
6:     while $path\_queue.length > 0$ do
7:         $path \leftarrow path\_queue.pop()$
8:         $s1 \leftarrow path.end()$
9:         for all $edge \in s1.out\_edges()$ do
10:            $s2 \leftarrow edge.dest\_node()$
11:            if $s2 \notin path$ then
12:                $new\_path \leftarrow path.append(edge)$
13:                $path\_set[s2] \leftarrow path\_set[s2] \cup \{new\_path\}$
14:                if $s2 \neq target$ then
15:                    $path\_queue.push(new\_path)$
16:                end if
17:            end if
18:        end for
19:    end while
20:    $pareto\_set \leftarrow \emptyset$
21:    for all $path \in path\_set[target]$ do
22:        $pareto\_set \leftarrow pareto\_add(pareto\_set, path)$
23:    end for
24:    return $pareto\_set$
25: end procedure
```

Algorithm 1 illustrates our variant of the popular breadth-first search technique that finds all the simple paths from the source node to a target node. Unlike the normal algorithm, it does not maintain "visited" tags on the nodes, because a node may be visited multiple times when the algorithm examines different paths. The algorithm starts with a source node and enqueues it into a path queue, i.e., $path\_queue$. The source node is then dequeued with all nodes reachable from the source's outgoing directed edges being added to the $path\_queue$ as new paths from the source node to some other node in the graph. Each time a node is dequeued from the $path\_queue$, it is stored in the path set corresponding to its last node. Meanwhile, the outgoing neighbors of the dequeued path's last node are added to its end to form new paths. These new paths are the added to the $path\_queue$.

Algorithm 2 Algorithm to validate a path's Pareto-optimal-ity for a Pareto-optimal set, including possible eviction of existing paths from the set.

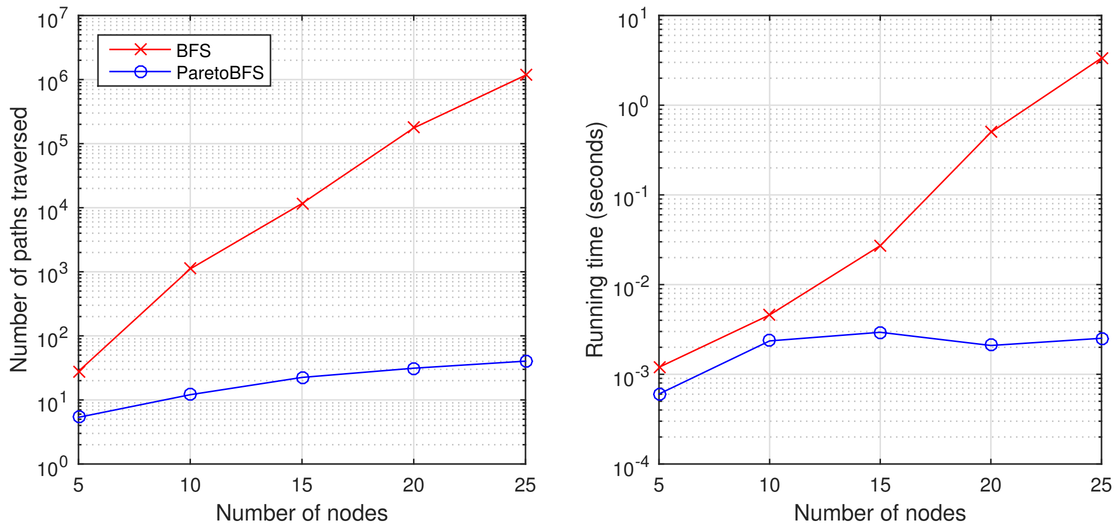
```
 1: procedure PARETO_ADD($pareto\_set, new\_path$)
 2:     $result\_set \leftarrow \varnothing$
 3:     for all $path \in pareto\_set$ do
 4:         if $path$ is strictly more optimal than $new\_path$ then
 5:             return $pareto\_set$
 6:         else if $new\_path$ is not strictly more optimal than $path$ then
 7:             $result\_set.append(path)$
 8:         end if
 9:     end for
10:     $result\_set.append(new\_path)$
11:     return $result\_set$
12: end procedure
```

To prevent loops, Line 11 checks whether the neighbor is already in the active path before appending it. This enqueueing and dequeueing is repeated until $path\_queue$ is empty, and the $path\_set$ contains all the simple paths [3] from source node to all other nodes. Selecting a Pareto-optimal set from it is straightforward, as depicted by the $pareto\_add$ function in Algorithm 2.

Algorithm 1 can then be easily extended to find the Pareto-optimal paths from one source node to all the other nodes, by replacing Line 13 with a $pareto\_add$ function, and removing Line 14 and Lines 20 - 23.

This algorithm clearly suffers from scalability issues. In a directed graph, the number of possible paths is usually exponential to the number of nodes. Moreover, for a multigraph with p potential parallel edges between each pair of nodes, the total number of paths increases with a factor of p^h, where h is the number of hops in a path. Fig. 4.2(a) shows the number of paths traversed by Algorithm 1. It is clear that the growth is exponential in nature; thus, enumerating all the possible paths is typically infeasible, in both a temporal and spatial sense.

[3] A simple path is a path without any cycles or loops.



(a) Traversed paths (b) Runtime

Figure 4.2: BFS Vs ParetoBFS on a BRITE generated topology: metrics-per-edge=2, parallel-edges=1 (averaged over 60 runs with different graphs and source-target pairs)

To make Algorithm 1's runtime practically acceptable, we need to constrain that problem space being explored by the algorithm as it progresses through the graph. We accomplish this by pruning paths (at the node currently under consideration) that are not Pareto-optimal, thus effectively constraining the impact those partial paths will have on the eventual solution space.

4.4.2 ParetoBFS– BFS with Pruning Support

As briefly discussed previously, we can constrain pathfinding by not considering a partial path any more if it is already strictly worse than other known paths up until that point. We refer to this interchangeably as *pruning* or *branch-and-bound*. Formally, during the search process, a path ending with node v_i can be pruned if one of the following conditions are satisfied:

1. The path is dominated by a path already in the Pareto-optimal path-set with destination node v_i, or

Algorithm 3 ParetoBFS

```
1: procedure ParetoBFS($G, source, target$)
2:     for all $v \in G(v)$ do
3:         $pareto\_set[v] \leftarrow \varnothing$
4:     end for
5:     $path\_queue.push([source])$
6:     while $path\_queue.length > 0$ do
7:         $p \leftarrow path\_queue.pop()$
8:         $s1 \leftarrow p.end()$
9:         if $p$ is PO for $pareto\_set[target]$ and $p \in pareto\_set[s1]$ then
10:            for all $edge \in s1.out\_edges()$ do        ▷ Check whether p satisfies PO conditions
11:                $s2 \leftarrow edge.dest\_node()$
12:                if $s2 \notin p$ then
13:                    $new\_p \leftarrow p.append(edge)$
14:                    if $new\_p$ is PO to $pareto\_set[target]$ and $pareto\_set[s2]$ then
15:                        $pareto\_add(pareto\_set[s2], new\_p)$    ▷ can $new\_p$ be added to the PO path set of $s2$?
16:                        if $s2 \neq target$ then
17:                            $path\_queue.push(new\_p)$
18:                        end if
19:                    end if
20:                end if
21:            end for
22:        else
23:            continue
24:        end if
25:    end while
26:    return $pareto\_set[target]$
27: end procedure
```

2. The path is dominated by a path in the Pareto-optimal path set with destination node $target$.

The modified algorithm with this branch-and-bound approach maintains the same theoretical worst-case time and space complexity. In practice, however, pruning dramatically reduces the path search space. Pruning does not affect the correctness of the final solution, since merely extending a path by adding hops cannot transform a sub-optimal path into an optimal one.

Modifying Algorithm 1 with the added logic for branch-and-bound, we arrive at the ParetoBFS algorithm as shown in Algorithm 3. 'PO' in the algorithm means Pareto-optimality. With this approach, instead of saving all the possible paths, a map, $pareto\_set$, is used to save the Pareto-optimal paths from the source node to each node. The notable differences from Algorithm 1 occur in Lines 9, 14 and 15. Lines 14 and 15 check the Pareto-optimality of the partial path with the node at that point in the traversal, before queueing the node, to eliminate any sub-optimal paths. There is another check incorporated after the dequeue step, in Line 9, since the Pareto-optimal sets may have changed during the time the node under consideration was sitting in the BFS queue.

Fig. 4.2(a) demonstrates that this dynamic bounding method can effectively reduce the number of traversed paths by several orders of magnitude. The detailed performance and complexity analysis is discussed in Section 4.5. Algorithm 3 can be extended to find the Pareto-optimal paths to all other nodes, by removing the condition checks involving the $target$ Pareto-optimal set in Lines 9, 14 and 16. It is only natural that the running time of the algorithm increases due of the more relaxed nature of the bounding in this case.

4.5 Evaluation and Complexity Analysis

In this section, we discuss the effectiveness of our ParetoBFS algorithmic approach in the context of network graphs to show that it can be practically applied to problems.

4.5.1 Methodology

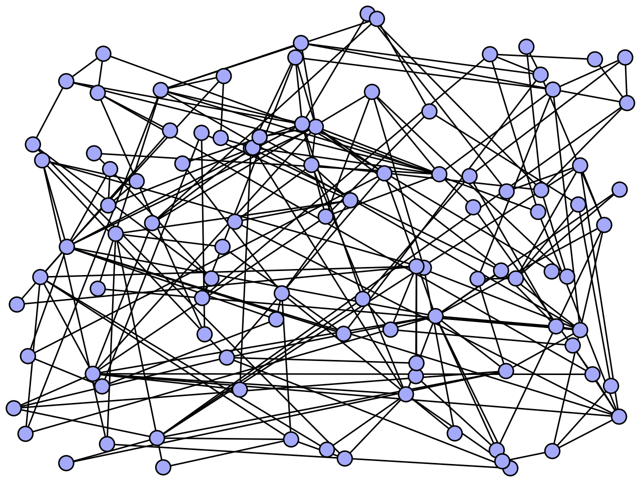
To test the performance of ParetoBFS, we use both synthetic and real-world topologies. Although ParetoBFS can apply to both intra-AS and inter-AS topologies, much of intelligent routing applications happens within private domains. Therefore, we focus our evaluation efforts on the intra-AS topologies. We use the BRITE topology generator [97] to generate router-level topologies. The sizes of these range from 100-10,000 nodes. BRITE provides three metrics for paths: length, bandwidth and latency. For tests that consider

Table 4.1: ParetoBFS runtime and average Pareto-optimal path-count comparison.

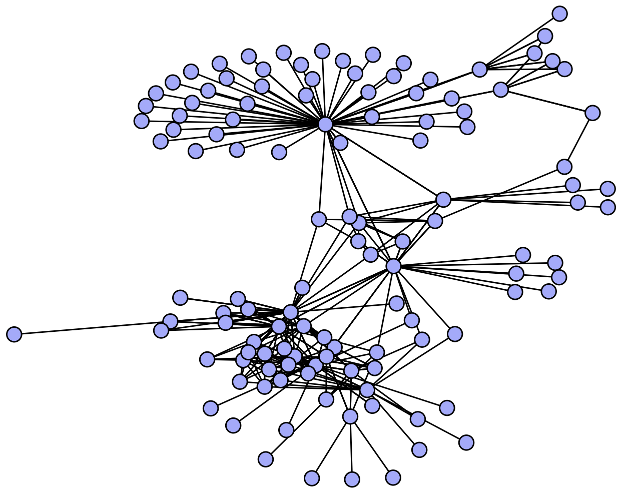
Node Placement	Bandwidth Distribution	Model							
		Waxman		BA		BA-2		GLP	
		time	paths	time	paths	time	paths	time	paths
Random	Constant	0.03	1.00	0.03	1.00	0.06	1.00	0.03	1.00
	Uniform	0.36	7.46	0.26	5.22	0.64	7.42	0.12	2.24
	Exponential	0.36	6.60	0.23	4.16	0.62	7.22	0.09	1.94
	HeavyTailed	0.42	6.64	0.28	4.84	0.68	7.40	0.12	2.36
Heavy Tailed	Constant	0.04	1.00	0.05	1.00	0.08	1.00	0.03	1.00
	Uniform	0.59	8.46	0.37	5.24	0.89	7.78	0.15	1.98
	Exponential	0.52	7.22	0.30	5.78	0.81	7.96	0.15	2.46
	HeavyTailed	0.48	7.62	0.25	4.68	0.84	7.76	0.13	2.70

more than three metrics, we choose random metrics as placeholders in addition to the ones mentioned. Since the algorithmic approach's complexity is determined by the number of metrics used, and not necessarily by the 'type' of metric in use, the random choices do not affect the outcome of the paths chosen in any way.

We focus our experiments on BRITE's graph generation models, node placement and bandwidth distribution parameters. BRITE provides four generation models: Waxman [141], BA [12], BA-2 [7] and GLP [23] (the GLP model being mainly for AS-level topologies). The node placement can be done in one of two ways: random and heavy-tailed. The bandwidth distribution has four options: constant, uniform, exponential and heavy-tailed. We test all the combinations and compare the running time and the Pareto-optimal path-count in Table 4.1. The experiments are averaged over 100 runs for 1,000 nodes, three metrics per edge and one parallel edges between nodes. .From the table, it is evident that many combinations of parameters do not influence the path-finding results in any significant way, except for the test with a constant bandwidth distribution. Therefore, we can arbitrarily pick these parameters. In the following experiments, the generation model used is the Waxman model, a commonly used intra-AS model. The node placement is random, and we utilize a uniform bandwidth distribution.



(a) 100-node synthetic topology.



(b) 100+ - node Rocketfuel topology, AS 4755 (121 nodes).

Figure 4.3: Example test topologies for ParetoBFS

To analyze the performance on real-world topologies, we use Rocketfuel [131], an ISP topology dataset. Each Rocketfuel data file represents a single-AS topology, ranging from a 100 nodes to 10,000 nodes. The data does not include any metrics, subsequently we randomly generate values for the metrics using a normal distribution.

Both the generated and the real-world topologies are unigraphs, i.e., topologies with only one edge between the same pair of nodes. Sometimes, we need more than one edge between two nodes. These parallel edges can be either physical links with a different set of metrics, or service offerings on the same link with different QoS limits. To account for this behavior, we can extend the unigraphs to multi-graphs, by duplicating each edge of the unigraph and assigning with its set of metrics.

We use Python to implement our algorithms because of its convenient graph libraries, and the ability to integrate it with the Pox SDN-controller [4]. Additionally, we utilize the Pypy [5] interpreter to run the experiments, to enable performance improvements, much closer to native code as compared to running it with a general interpreter. One notable exception is the convex sampling discussed in Section 4.6 for which we utilize CPython since the convex hull calculation use the *pyhull* package, which is not Pypy compatible.

The processor in use is an Intel Core2 Quad CPU Q9400 clocked at 2.66 GHz, running an Ubuntu 14.04 64-bit kernel (version 3.13.0-24) and pypy 2.6.0.

4.5.2 Complexity Analysis

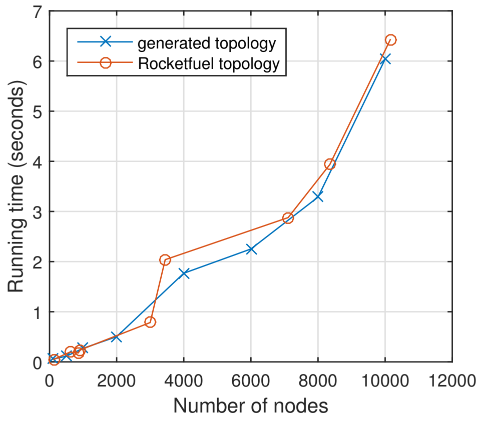
In this section, we provide a theoretical analysis and comparison of BFS and the ParetoBFS algorithms (i.e., Algorithms 1 and 3).

Let $G = (V, E)$ be the graph, where $V = (v_1, v_2, ..., v_n)$ is a set of all nodes, and $E = (e_1, e_2, ..., e_m)$ is a set of all edges in the graph. For the purposes of this analysis, we assume the source node is v_1 and the target node is v_n.

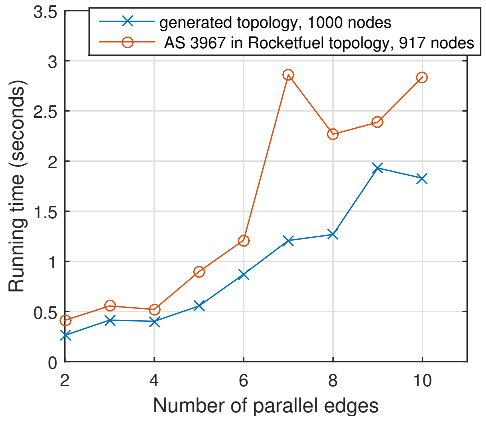
Just reiterating the different approaches, Algorithm 1 first finds all possible paths and then filters them to retain only the Pareto-optimal paths. On the other hand, Algorithm 3

[4] `http://www.noxrepo.org/pox/about-pox/`

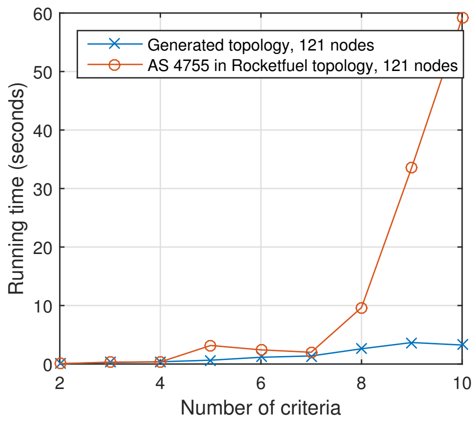
[5] A Python interpreter with JIT compiler. `http://pypy.org/`



(a) Runtime Vs Node Count (metrics-per-edge=2, parallel-edges=3).



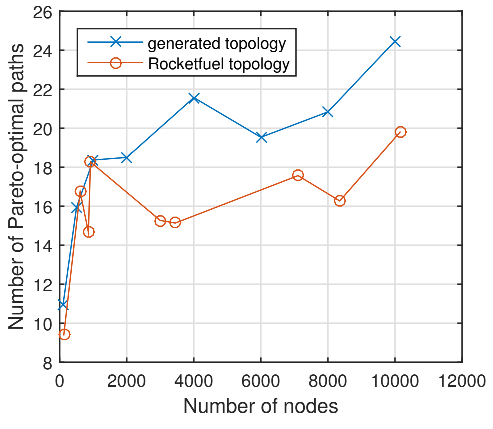
(b) Runtime Vs Parallel-Edge Count (metrics-per-edge = 2).



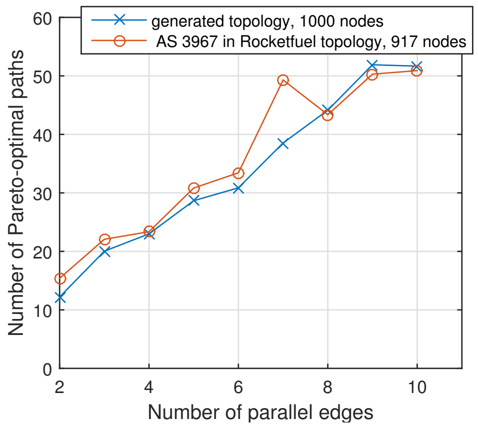
(c) Runtime Vs Metric Count (parallel-edges=3).

Figure 4.4: ParetoBFS runtime comparison with different parameters, averaged over 30 runs per data point.

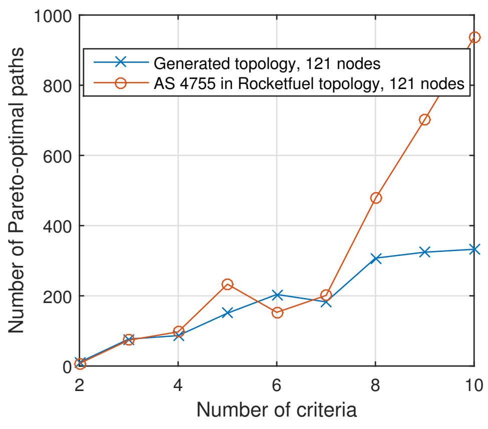
prunes the path-set to retain only Pareto-optimal paths as it traverses the graph. As discussed in Section 4.3, a sub-optimal path does not become optimal by just extending the path length. Therefore, all Pareto-optimal paths considered here are simple paths, without loops/cycles. In a directed graph, for a simple path, we can order the vertices such that edges only point forward. E.g., if node u is a descendent of node v, then node u comes after node v in the sorted list of nodes. In Algorithm 1, the times that each node v_i ($i = 1, 2, ..., n$) is visited is equal to the number of the paths from source v_1 to node v_i. Let v_2 be v_1's neigh-



(a) Path-count Vs Node Count (metrics-per-edge=2, parallel-edges=3).



(b) Path-count Vs Parallel-Edge Count (metrics-per-edge=2).



(c) Path-count Vs Metric Count (parallel-edges=3).

Figure 4.5: Pareto-optimal path-count comparison, averaged over 30 runs per data point.

bor. The number of paths from v_1 to v_2 is equal to the number of parallel edges between them. Furthermore, let v_3 be one of v_2's neighbours. The number of paths from v_1 to v_3 is the number of (direct) edges from v_1 to v_3, plus the paths that use v_2 as an intermediate vertex. Generalizing this pattern, if $e(i, j)$ be the number of directed edges between node v_i and v_j ($e(i, j) = 0$ if v_i and node v_j are not adjacent nodes), and $d(j)$ be the number of paths from v_1 to v_j, then we have:

$$d(j) = e(1,j) + \sum_{i=2}^{j} d(k)e(k,j)$$

For each node v_j, computing $d(j)$ takes time proportional to the in-degree of node v_j, and overall it will take $O(m)$ time. Therefore, Algorithm 1 visits each node $O(m)$ times, and thus, the total time to find all the possible paths in is also $O(nm)$.

To analyze the complexity of the Pareto-optimal path selection post-processing, let p be the path-count from source node v_1 to target node v_n. p could be 1 if there is only one simple path from node v_1 to node v_n, or p could also be $n!$ if the graph G is fully meshed. Algorithm 2 takes $O(1)$ computation times for each path in the input $pareto\_set$. The process of screening out the Pareto-optimal paths adds *one* Pareto-optimal path each time from the temporary $pareto\_set$, and the number of paths in $pareto\_set$ goes from 0 to $p-1$. The algorithm thus performs path computations or comparisons $O(1+2+\cdots+p) = O(p^2)$ times, which gives us a running time for Algorithm 1 of $O(nm+p^2)$.

In contrast to Algorithm 1, Algorithm 3 deletes the non-Pareto-optimal paths from source node v_1 to node v_j each time when it visits node v_j. Therefore, the number of paths saved in the $path\_queue$ will be less than what Algorithm 1 does. The number could be the same if all the paths are Pareto-optimal. Thus, in the worst case, Algorithm 3 also visits each node $O(m)$ times. We denote p^* as the Pareto-optimal paths between the source node v_1 and the target node v_n. The total running time for Algorithm 3 is $O(nmp^*)$.

The time complexity of Algorithm 1 is dominated by the number of paths p. In fact, with a typical network topology, p usually grows exponentially with the number of nodes n. Consider the graph in Fig. 4.1 as an example. If we have two parallel edges between each pair of connected node, then the number of paths from node A to F becomes $3 \times 2^3 + 3 \times 2^4 + 1 \times 2^5 = 102$, which is much larger than n ($n = 6$). Besides, the number of possible paths double when a new node is added to the graph. On the contrary, the time complexity of Algorithm 3 may *not* be dominated by the number of Pareto-optimal paths, if $p^* << p$. However, optimal path count p^* would grow rapidly when the number of metrics under consideration increases. In this case, the time complexity is dominated by p^*, and demonstrates an approximate exponential growth pattern with n. The experimental results discussed in the next section validate our analysis.

4.5.3 Experimental Results

Let us analyze the experimental results of the ParetoBFS algorithm. Fig. 4.2(b) compares the running time of the plain BFS and ParetoBFS approaches. It shows that the running time of plain BFS increases exponentially as the number of nodes increase. The complexity of ParetoBFS is sub-exponential, i.e., the runtime may grow faster than a polynomial solution but is still significantly smaller than an exponential one. This is in line with the worst-case exponential growth of ParetoBFS's runtime with a node-count increase, which happens when the number of Pareto-optimal paths makes up a majority of the paths between source and target nodes. In a realistic network topology, the Pareto-optimal paths are usually a small fraction of the total paths. Branching and bounding can keep the curve from becoming too steep since it keeps eliminating non-Pareto-optimal paths at each node, thus preventing sub-optimal partial paths from contributing to the runtime.

We focus on the ParetoBFS's running time. The memory consumption is proportional to the runtime, because it depends on the length of the $path\_queue$. Fig. 4.4 illsutrates how the average running time of ParetoBFS grows as the numbers of nodes, parallel edges and criteria increase. Fig. 4.4a shows that ParetoBFS can find all the Pareto-optimal paths on a 10,000-node topology in ≈ 30 seconds. The complexity with an increase in number of parallel edges between nodes, shown by Fig. 4.4b is similar to that illustrated by Fig. 4.4a. We believe this is reasonable because increasing either the number of parallel edges between nodes, or the number of nodes in the graph has similar effects on the graph traversal queue length, and pruning the paths also has similar effect. Subsequently, the complexities illustrated are similar in nature.

Fig. 4.4c, however, shows a steeper growth than the other results. For instance, if there are k metrics $w_1, w_2 ..., w_k$ on each edge (the value of w_k is generated randomly) for two neighboring nodes parallel edges connecting them, the probability that these two edges are Pareto-optimal is $1 - \frac{1}{2^{k-1}}$. As the criteria increase, the number of Pareto-optimal paths approaches the total path count between the edges. This is the worst case complexity for ParetoBFS where the runtime demonstrates exponential growth. The signifitcantly large number of metrics also lessens the effectiveness of path pruning, thus increasing the run-time complexity. In order to reduce the running time when the number of metrics is high, we propose several sampling methods in Section 4.6 to reduce the size of the Pareto-optimal path-set.

Fig. 4.5 illustrates how the number of Pareto-optimal paths, p^*, grows with the number of nodes, parallel edges and criteria, respectively. In Fig. 4.5a, the Rocketfuel-topology curve fluctuates, because each real topology has a unique interior structure which is not uniform like a synthetic topology. In Fig. 4.5b, it can be observed that the number of the Pareto-optimal paths increases linearly with the number of parallel edges with two criteria per edge. The curves validate the correctness of $O(nmp^*)$ runtime analysis for ParetoBFS. When the number of parallel edges double, p^* and m also double. If the curves of p^* in Fig. 4.5b can be considered linear, the curves in Fig. 4.4b are polynomial in nature. Fig. 4.5c shows how p^* varies with the number of criteria. It can be observed that p^* demonstrates a greater increase in Fig. 4.5c, than in Fig. 4.5a and Fig. 4.5b. As discussed in Section 4.5.2, a large number of criteria results in the number of Pareto-optimal paths approaching the total path-count.

4.6 Algorithm for reducing the Pareto-optimal solution space using sampling.

Since ParetoBFS finds all Pareto-optimal paths, the size of Pareto-optimal set may grow exponentially with an increase in the number of criteria under consideration. Even for a small 1,000-node network with just three criteria, there may be hundreds of Pareto-optimal paths between two nodes. Depending on how many Pareto-optimal paths are required, or

Algorithm 4 Sampling the Pareto-optimal set on path addition

```
1: procedure SAMPLING_ADD($pareto\_set, new\_path$)
2:     $result\_set = pareto\_add(pareto\_set, new\_path)$
3:     if $result\_set.length > threshold$ then
4:         $sampled\_set = sampling(result\_set)$
5:         return $sampled\_set$
6:     else
7:         return $result\_set$
8:     end if
9: end procedure
```

inordinately long run times, we introduce heuristic-based sampling of the Pareto-optimal pathset. Sampling the Pareto-optimal set can be useful in two ways: (1) sampling reduces path selection difficulty (by eliminating some Pareto-optimal paths), and (2) if sampling happens during the search, the number of traversed paths can be further reduced, thus allowing the algorithm to converge quicker towards a practically acceptable solution made with tradeoffs.

Algorithm 4 describes how to sample paths. Every time after a path is added to the Pareto-optimal set, the algorithm checks if the Pareto-optimal set is larger than a threshold $threshold$. If it is, a sampling method is used to reduce the Pareto-optimal set to l paths. It would be beneficial to note at this point that sampling can discard some useful paths too. It is also possible that the final result is not a subset of the original ParetoBFS result.

If $P = \{p_1, ..., p_m\}$ is the Pareto-optimal set found by ParetoBFS, and $Q = \{q_1, ..., q_n\}$ is the Pareto-optimal set found by ParetoBFS with sampling, we propose the following metrics to compare the effectiveness of reducing the solution space with sampling:

- Runtime Ratio (RT) is defined as the ratio of the running time to find Q versus P. This metric indicates how the sampling method affects the running time.
- Path Count Ratio (PC) is defined as the ratio of Q's size to P's size, that is, $PC = n/m$. This metric indicates how many Pareto-optimal paths can be found (or conversely lost) using sampling. It is in no way indicative of path optimality.

- Path Quality (PQ) is defined as the average k-dimensional Euclidean distance between P's and Q's criteria vector sets, $w^Q = \{w^{q_1}, ..., w^{q_n}\}$ and $w^P = \{w^{p_1}, ..., w^{p_m}\}$. Each w^{q_i} or w^{p_i} is a Pareto-optimal path's criteria vector. To calculate PQ, first normalize w^P and w^Q into w^P's range:

$$w_j^{p_i\prime} = \frac{w_j^{p_i} - \min(w_j^{p_1} \dots w_j^{p_m})}{\max(w_j^{p_1} \dots w_j^{p_m}) - \min(w_j^{p_1} \dots w_j^{p_m})}, {}_{j \in \{1 \dots k\}}^{i \in \{1 \dots m\}}$$

$$w_j^{q_i\prime} = \frac{w_j^{q_i} - \min(w_j^{p_1} \dots w_j^{p_m})}{\max(w_j^{p_1} \dots w_j^{p_m}) - \min(w_j^{p_1} \dots w_j^{p_m})}, {}_{j \in \{1 \dots k\}}^{i \in \{1 \dots n\}}$$

Then, for each $w^{q_i\prime}$, calculate the distance from its closest $w^{p_t\prime}$:

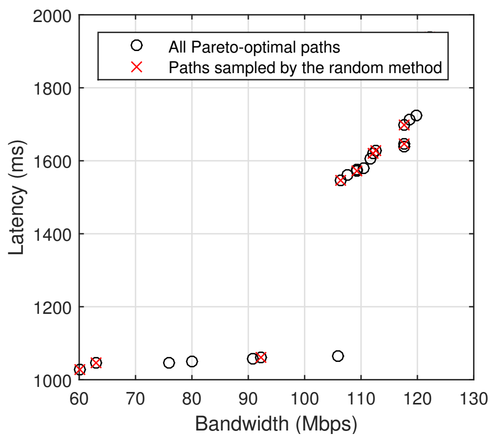
$$d^{q_i} = \min_{t \in \{1 \dots m\}} \sqrt{\sum_{j \in \{1 \dots k\}} (w_j^{q_i\prime} - w_j^{p_t\prime})^2}$$

Finally, PQ can be defined as: $PQ = \frac{1}{n} \sum_{i=1}^{n} (d^{q_i})$. It can be viewed as the average distance between w^P and w^Q, $PQ = 0$ means Q is a subset of P.

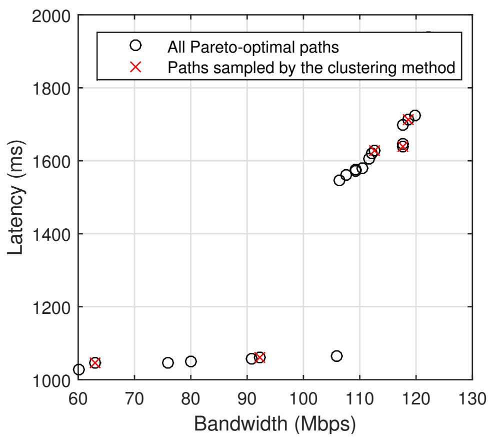
To achieve our goal of managing the runtime of ParetoBFS, the sampling method must be fast and possess the capability to process an arbitrary number of criteria. Also, the sampling methods should treat each criterion equally. In this section, we propose and investigate three sampling techniques: random, clustering, and convex sampling. The sampling methods are evaluated for two criteria, three parallel edges between nodes and a 10K node graph.

4.6.1 Random Sampling

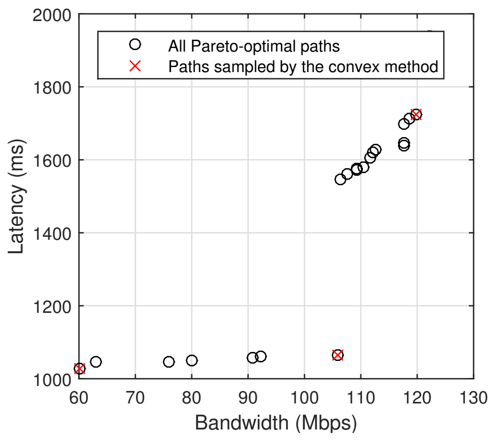
This method randomly samples l paths from the Pareto-optimal set. It is fast, but does not make use of any information from the data points. The result of a 2-criteria example is shown in Fig. 4.6a. Q mostly overlaps with P, which means that, after sampling, we can still find an approximate subset of the Pareto-optimal paths.



(a) Random sampling ($threshold$ = 10, l = 6).



(b) Clustering-based sampling ($threshold$ = 10, l = 6).



(c) Convex sampling ($threshold$ = 10).

Figure 4.6: Comparison of different sampling methods.

4.6.2 Clustering-based Sampling

It is an intuitive idea to cluster Pareto-optimal points that are close to each other in a k-dimensional space, especially when redundant paths are not particularly beneficial. Here, we use Lloyd's clustering algorithm [86] to divide the points into l groups, and select the points closest to the center of each group. The time complexity of Lloyd's algorithm's is $O(nkli)$ (n being the number of points, k being the dimensions, l being the number of

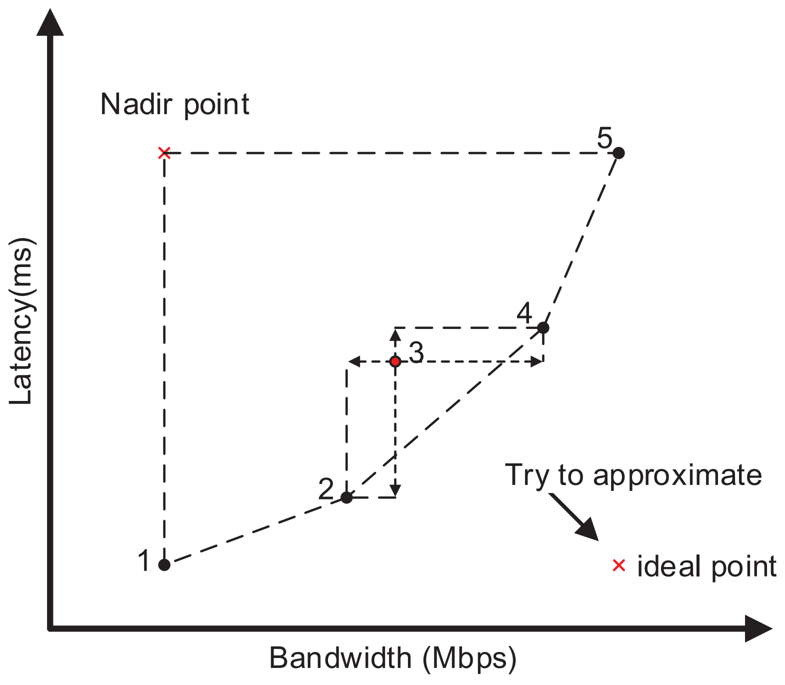


Figure 4.7: Convex sampling example.

groups, and i being the number of iterations). Fig. 4.6b illustrates one such result. The points are more dispersed than Fig. 4.6a, thus they are more representative.

4.6.3 Convex Sampling

The fundamental assumption of convex sampling is that the points on the convex hull are better than the ones inside. This can be illustrated by considering an example such as the one in Fig. 4.7. Points 1-5 are all Pareto-optimal points. Points 1, 2, 4, 5 and the *nadir point* (not a real data point) form the convex hull. Point-3 falls inside the hull. Compared to Point-2, Point-3 shows only a slight improvement in bandwidth, but sacrifices a lot in latency. A similar situation applies to Points 3 and 4. Therefore, Points 2 and 4 seems more preferable than Point 3. This method works better if the criteria are sum-type, since points on the convex hull are more likely to stay optimal when the path is extended.

Table 4.2: Effectiveness of sampling methods.

k	th	l	random			clustering			convex		
			RT	PC	PQ	RT	PC	PQ	RT	PC	PQ
2	10	5	1.175	0.850	0.141	1.632	0.869	0.004	1.058	0.828	0.001
3	20	10	0.530	0.461	0.022	1.405	0.455	0.026	0.431	0.546	0.007
4	100	10	0.473	0.384	0.030	1.087	0.413	0.032	0.393	0.502	0.030

We use the *qhull* library, which implements the Quickhull algorithm [13]. Its time complexity is $O(nlogv)$ in two and three dimensions, and $O(n\frac{v(\lfloor d/2 \rfloor-1)}{\lfloor d/2 \rfloor})$ for higher dimensions (n being the number of points, and v being the number of points on the convex hull). Fig. 4.6c demonstrates points inside the convex hull being ignored successfully. For higher order dimensionality (n = 4), the performance of *qhull* degrades rapidly and it may no longer help improve ParetoBFS's performance.

Analyzing the advantages of convex sampling, we see that it always reserves the points at the corners of the spectrum (e.g. Point 1 and 5 in Fig. 4.6c), that represent extreme values in one dimension. These are more important for making decisions being made about the highest value in a certain dimension. Also, calculating the convex hull does not require normalizing each dimension, thus improving the speed.

The downside to convex sampling is that it cannot control how many points are sampled. It is entirely possible that we are left with too few, or too many points. This again reintroduces the question of sampling quality and runtime increases.

4.6.4 Comparison of Sampling Techniques

We test the three sampling techniques on nine Rocketfuel topologies, whose sizes range from 121 nodes to 10,152 nodes, and average out the runtime RT, path count PC, and path quality PQ. The tabulated results are available in Table 4.2. The sampling threshold $threshold$ and sample size l also affect RT, PC and PQ. These parameters are chosen from trial runs, to obtain a good compromise between the runtime and result accuracy.

Clustering-based sampling does not reduce the runtime RT for k = 2 (k being the dimensionality), but does reduce the running time at higher dimensionalities. Random and convex sampling perform about the same in two dimensions. However, the computational time consumed for clustering, including data normalization, is prohibitive in some ways. Although the number of points and groups chosen are relatively small and fixed, the constant coefficient appears to dominate the runtime. We find the clustering-based approach to be slow enough to recommend it not be used.

Analyzing the path count PC, all the techniques find a similar amount of Pareto-optimal paths. Even for four-criteria problems, they still manage to find 40-50$\%$ of the Pareto-optimal paths. Convex sampling performs slightly better at higher dimensionalitites again. Looking into the quality of the results, defined by path quality PQ, convex sampling appears to produce the best results. The PQ produced by convex samplinng increases a greater rate than the other two, and this may be explained by the inability of convex sampling to control the sample size. Consequently, the accuracy of the result is not as tunable.

Summarizing the analysis, convex sampling appears to work the best, at least up to four dimensions. It is faster, finds more Pareto-optimal paths with higher path quality. It is clear that convex sampling can be used to benefit runtime reduction up to four dimensions. Higher order dimensionalities (which appear to be impractical, at least at this point, from the perspective of the problem at hand) are beyond the scope of this dissertation.

4.7 Extended Results

In this section, we discuss some extended results of the multi-criteria pathfinding problem, with respect to different graph types, sizes etc. We look at a new comparison metric called *path-stretch*, consider average path counts, average run-time comparisons for synthetic graphs and practical ISP topologies etc.

We begin by defining path-stretch as:

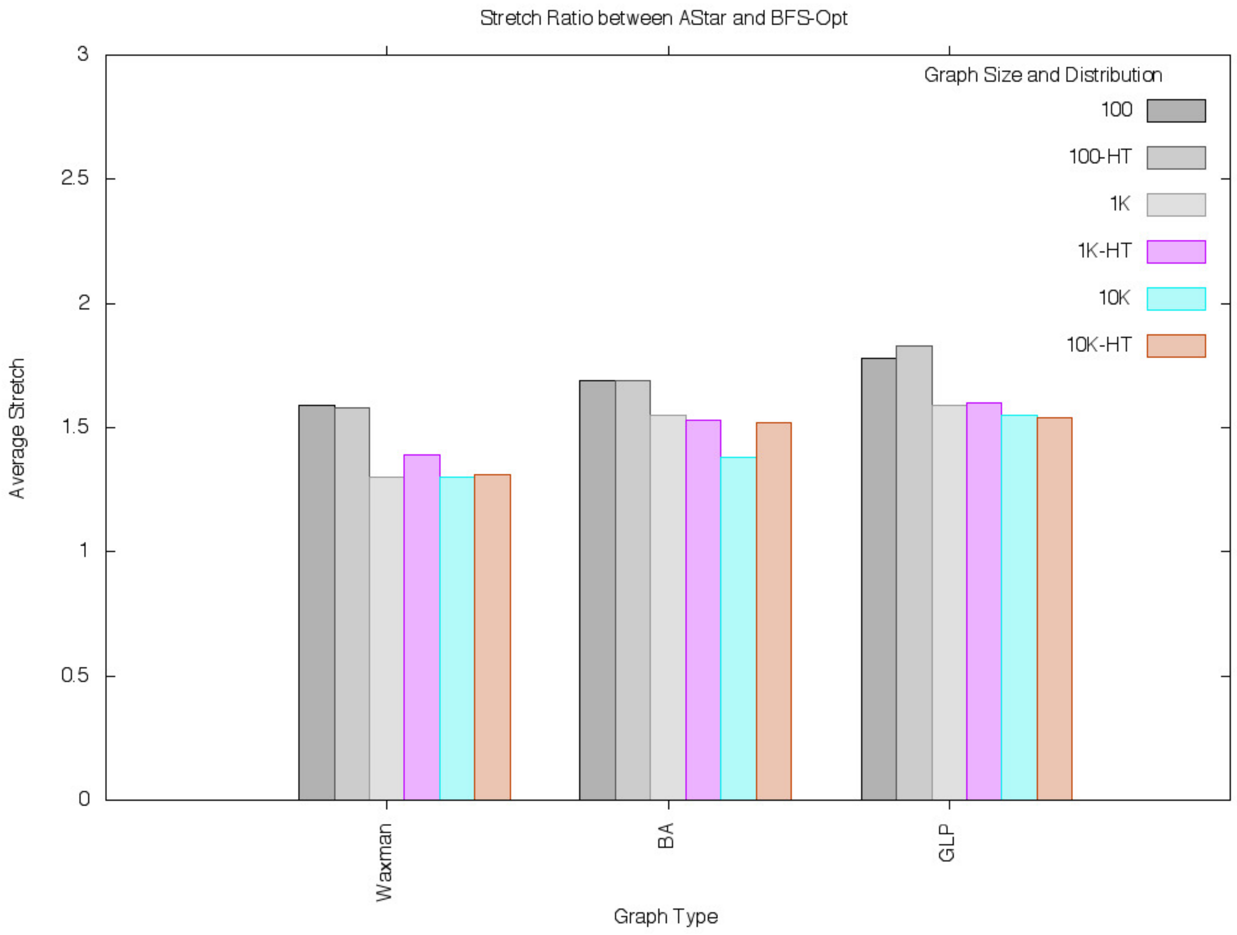


Figure 4.8: Average stretch for all graph models.

$$Stretch = \frac{Avg\ Optimal\ Path\ Hop\ Count}{Avg\ SP\ Hop\ Count}$$

where, SP Hop Count is the shortest path hop count.

Since the single criterion shortest path is the absolute optimal path from a source to a target, we would like to see how the Pareto-optimal paths compare to the shortest path in terms of path length. This metric is to obtain a sense how Pareto-optimality is affecting the path/hop length of network paths. It would not be very beneficial to have paths that are inordinately long.

Figure 4.8 plots the average stretch of resultant path-sets over various types of graph models tested. The average stretch acsross models is consistent at about 1.5, indicating that it is mostly independent of the graph topology. The tests performed with RocketFuel also resulted in an average stretch of 1.2. These results indicate a 50% and 20% increase in path length in the former and latter cases, which by all means is not inordinate or unacceptable, especially with the advantages multi-criteria routing brings to the table.

Next, we want to observe the path-count dependency on graph type and size. As earlier, we use the same BRITE models, Waxman, BA, and GLP with a normal and heavytailed node distribution policy. Table 4.3 tabulates the average paths found per graph model and size. The 'HT' indicates a heavy-tailed node distribution policy. The average number of paths found ranges between 4.5 and 9 in most cases, except for the one case on the Waxman-1K model. We attribute this anomaly to the random nature of the graph. These averages were measured across 100 iterations of pathfinding for random source-destination pairs in the graph.

Table 4.3: Average path count for all graph models and sizes.

Graph Size	Waxman	Waxman-HT	BA	BA-HT	GLP	GLP-HT
100 Nodes	5.68	5.53	5.27	5.37	5.71	5.75
1K Nodes	1.54	5.96	6.88	7.23	6.82	7.53
10K Nodes	9.1	4.5	6.8	8.6	7.0	9.0

To further investigate the runtime behavior of the algorithm (extending on the results from Fig. 4.4b and Fig. 4.4a), and understand its dependency on node distribution and edge connectivity, we looked at the runtime deviations for different graph models and generation strategies. We also used the Rocketfuel data to build a complete multigraph of interconnected POPs of different ISPs. Doing so gives us the chance to model this as a genuine multigraph based on real-world network topologies. Some POPs have up to 30 or 40 links between then, representing various ASes they are part of, or indicating the various ISPs/ASes that has a POP in that location.

To represent the edge constraint vectors, since Rocketfuel contains only latencies and no bandwidth information, we build the multigraph based on those latencies and use a uniform random distribution to assign bandwidths to the edges. For this experiment, we use cost statistics from [39] on a per Mbps basis for assign the cost metric. This gives us enough information to use on the edge constraint vectors.

The multigraph that is built with this information has around 530 nodes and 4000 edges connecting the nodes. We also generate two Barabasi-Albert (BA) graphs that are similar in size and scale to this topology, albeit that the synthetic graphs are simple graphs, but have a higher node connectivity than the other models we used. The generated graphs have 500 and 1000 nodes, and each node has a connectivity factor of 8 and 4 respectively, giving us 4000 edges (which is about the size of the Rocketfuel topology). Results from this experiment will enable us to ratify the runtime's dependence on either node count or edge connectivity.

The run time analysis in Table 4.4 shows that the runtime is mostly independent of the node distribution policy of the topology. It is clear from the table that deviations for both the random and heavy-tailed placements are comparable. The last row in the table is the deviation calculated from the data obtained for runtimes on the RocketFuel topology and the two equivalent size synthetic BA topologies. These results extend our understanding that the algorithm's runtime performance is edge-connectivity dependent as compared to node placement policies.

Table 4.5 is a comparison between the average run times and path counts for the RocketFuel topology and the two generated BA graphs discussed above. Although this is not an apples-to-apples comparison with respect to graph types (since RocketFuel is a multigraph), it again illustrates the fact that the algorithm is dependent on the edge connectivity factor, and more so in the case of RocketFuel, which has multiple edges between the same node pairs. This results in higher runtimes, but also finds more paths, since the edge density

Table 4.4: Deviation in runtimes for various graph types and sizes.

Graph Type	Deviation
100 Nodes	0.0135
100 Nodes HT	0.0081
1K Nodes	0.49
1K Nodes HT	0.307
10K Nodes	6.39
10K Nodes HT	6.41
500 Nodes HED	38.26

is much higher. These results also validate our previous findings discussed in Section 4.5, and bolstering the case for using sampling to reduce runtimes.

We also measured the run time for two larger graphs, both BA models. The first one was a 45K node graph that is roughly about the size of all the nodes and edges exhaustively that RocketFuel project discovered over the course of its experiments. The second one is just a regular graph with 100K nodes and 200K edges with a connectivity factor of 2. This was done primarily for graph scaling purposes. Apart from the proportional increase in run time for which optimization with sampling is always possible, the takeaway is that the average stretches on both graphs are 1.59 and 1.55 respectively, which is consistent with our earlier results for other graph models. This just exhibits a consistent performance across graph types and models.

Table 4.5: Comparison between Rocketfuel and a generated BA graph.

Graph Type	Avg. runtime (in secs)	Avg. path count
RocketFuel	85.78	50
BA-500-8	36.58	31
BA-1K-4	10.41	19
BA-45K	93.35	9.2
BA-100K	241.71	9.8

Finally, to demonstrate the applicability of this approach to other network graph models, we investigated the algorithm's performance with datacenter network (DCN) topologies. To simulate the algorithm's pathfinding in a DCN topology, we use Scafida [58], a scale-free DCN topology generation algorithm that is also based on a variation of the Barabasi-Albert model that we previously used for our non-random graph models. The topology generation and subsequent pathfinding work on the assumption that both switches and servers route packets, such as in the design of DCell [57] [58]. Since the ability to distinguish between switches and servers/leaves has to be built into the algorithm, this is something that is use case specific and can be easily addressed. Multiple topologies of different sizes (different number of leaves and switches) are generated and we test the algorithm's running time and pathfinding capabilities on these topologies.

Datacenter network topologies are normally well-ordered designs with well mapped end points. The topolgoies used are typically Fat-tree, Bcube or Dcell based. A simple glance at these topologies intuitively tells us that most of them are symmterically ordered. Many also follow a tree structure, which in our case, will still work, but the number of choices of paths are going to much lesser since trees generally do not contain loops.

Experiments were conducted for topologies containing 10K and 50K servers, with extra nodes generated to account for the number of networking devices. In our case here, for the 10K server network, we had 600 odd switches, subscription being at about 95%, although the 50K server network was oversubscribed with around 5K switches. For the current pathfinding implementation, we could not ignore the networking device count since the pathfinder doesn't have information built into it to differentiate between switches and servers (we assumed a working nature analogous to DCell). Table 4.6 contains the results of these experiments. They are definitely relatable to the BA-models that we used earlier, which is in accordance with the fact that Scafida uses an extended BA model to generate the DCN topology.

We have a slightly higher stretch in this case, but we attribute that to the nature of the topology and pathfinder's inability to avoid routing through servers, although the same reason contributes to higher average path counts too. This can be addressed in a version of the pathfinder modified to suit DCN topolgies.

Table 4.6: Results for a Scafida DCN topology.

Graph Type	Avg. runtime (in secs)	Avg. path count	Avg. stretch
Scafida-10K	9.8	19.7	1.68
Scafida-50K	15.7	121.51	1.91

With respect to SDN and datacenter networks, the algorithm has currently been tested only on Scafida, that is a proposed architecture for DCNs. It does not make a very good use case for DCNs built on tree topologies since the number of paths are bound by the redundancy provided in the network spine. The algorithm will need to take into account tree-like topologies to work for all DCN architecture. We believe that this algorithmic approach, albeit modified can definitely be used for DCN pathfinding.

4.8 Comparison with Related Work

As discussed in Section 4.2, prior research has extensively addressed the multi-criteria path finding problem. There are several survey papers and bibliographies [43, 54, 91, 138] that summarize more than forty papers about the multi-criteria shortest path problem. Unfortunately, most of the papers only deal with sum-type metrics, as it is a mandatory requirement for shortest path problems. Only two papers, Hansen [59] and Pelegrin et al. [110] consider one sum-type and one bottleneck-type metric. We have implemented Hansen's algorithm, and compare it with ParetoBFS in Fig. 4.9. The comparison experiements are performed on a Rocketfuel topology with one parallel edge between nodes and two metrics per edge.

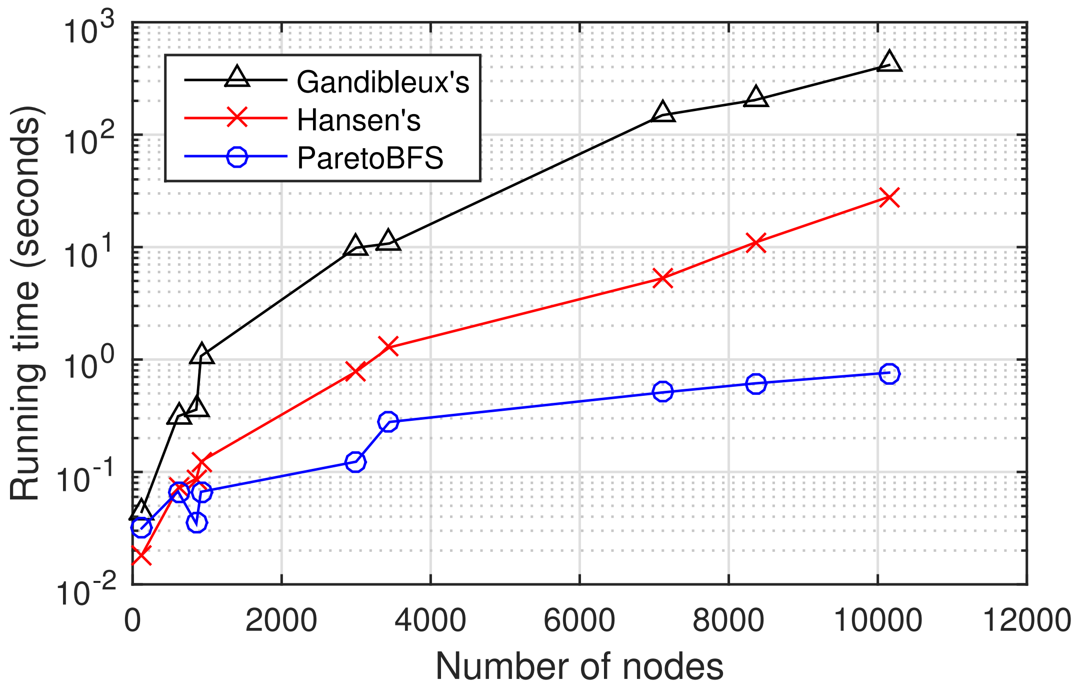


Figure 4.9: Runtime comparison of Hansen's algorithm and ParetoBFS.

In Hansen's algorithm [59], ten bi-criteria path problems are considered. The complexity of these problems is examined and a multi-label scheme is utilized. Since Hansen's algorithm finds the exact Pareto-optimal set, we only compare the runtimes here. ParetoBFS's runtime growth is slower than HAnsen's algorithm. Even for small topologies with a few hundred nodes, ParetoBFS is as fast as Hansen's approach. For large topologies, such as the one with 10,000 nodes, ParetoBFS is almost 40 times faster than Hansen's algorithm. The biggest advantage ParetoBFS holds in this regard over Hansen's algorithm is that the latter was designed for bi-criteria problems, whereas ParetoBFS can handle many more metrics per edge.

Other than the *exact methods* (i.e. to find all the Pareto-optimal paths) like ParetoBFS and Hansen's, many papers propose *approximation methods* to find a subset of Pareto-optimal paths in an efficient manner. These are known as *fully polynomial approximation*

Table 4.7: Comparison of Martins' Algorithm with ParetoBFS.

# of	k=2			k=3			k=4		
nodes	RT	PC	PQ	RT	PC	PQ	RT	PC	PQ
121	1.1	0.56	0.0000	1.3	0.41	0.0000	1.5	0.44	0.0000
609	23.7	0.42	0.0050	178.7	0.38	0.0018	121.2	0.38	0.0003
855	126.5	0.68	0.0000	233.4	0.61	0.0004	258.9	0.53	0.0007
917	34.4	0.41	0.0074	169.6	0.24	0.0008	279.1	0.37	0.0006

schemes (FPAS). All the FPAS' we investigated are only for sum-type metrics[6]. Here, we compare ParetoBFS with a popular FPAS, Martins' algorithm [90].

Martins' algorithm only approximates the Pareto-optimal set, which may differ from the exact Pareto-optimal set. So we compare the quality of results as described in Section 4.6. The result on 4 Rocketfuel topologies is shown in Table 4.7. Even for graphs with hundreds of nodes, Martins' is an order of magnitude (or more) slower than ParetoBFS. On larger Rocketfuel topologies, Martins' algorithm is not even feasible due to its inordinately high runtimes. Although Martins' algorithm finds a reasonable portion of the Pareto-optimal set (about 40-60%) and the quality of paths is very close to the exact Pareto-optimal set, it is prohibitively slow compared to ParetoBFS.

A full comparison of ParetoBFS with Hansen's and Martin's is shown in Table 4.8. p^* and p are Pareto-optimal paths= count and total possible path count between two nodes, respectively. From the comparison, we can see that ParetoBFS is superior than prior work in various aspects: it supports arbitrary number of sum-type and bottleneck-type metrics, and it finds the full Pareto-optimal set faster than other methods. Our experiments also show that it is even faster than certain FPAS in practice.

[6]Some research (e.g., [82]) suggests that bottleneck metrics can be converted to sum types by using reciprocals, i.e., define the optimal goal as: $f^p = \sum_e \frac{1}{bandwidth(e)}, e \in p$, where p is a path and e is an edge on p.

Table 4.8: Comparison of path-finding algorithms.

Type	number of criteria	number of Pareto-Optimal paths	Complexity
BFS variant	k	p^*	$O(mn + p^2)\ (p > p^*)$
ParetoBFS	k	p^*	$O(mnp^*)$
Hansen's [59]	2	p^*	$O(\frac{m^2n^2}{\epsilon}\log\frac{n^2}{\epsilon})\ (0 < \epsilon \leqslant 1)$
Martins' [90]	k (only sum-type metrics)	$\omega\ (\omega < p^*)$	$O(k^2mn\omega^2\log\omega)$

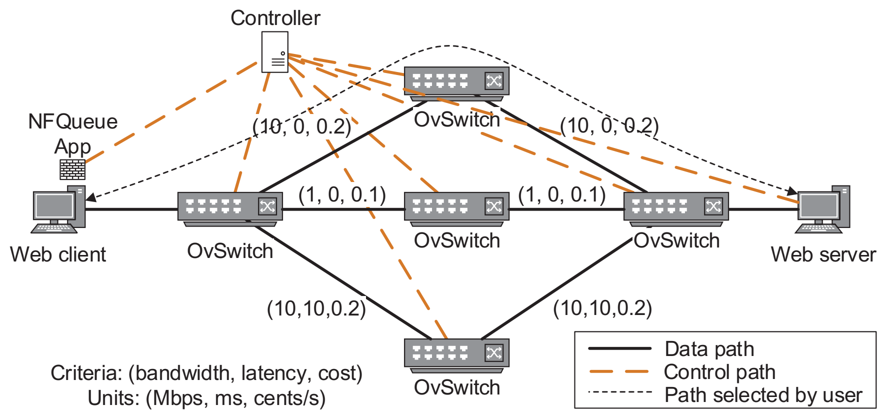


Figure 4.10: Test topology on GENI.

4.9 SDN-based Prototype

The emergence of SDN has provided an opportunity to implement multi-criteria path-finding as a foundational algorithm for service-based network architectures. The logical centralization of routing on the SDN controller and the ability to route each traffic flow independently makes the implementation and use of multi-criteria path-finding more practical than would be possible in a fully distributed network implementation. To test ParetoBFS and demonstrate its SDN use-case, we have built an SDN-prototye where the hosts can choose which Pareto-optimal link they want to use when initiating a connection.

The architecture of the network is illustrated in Fig. 4.10. Both switches and hosts connect to the controller, while the controller and switches use OpenFlow 1.0 to communicate with each other. Since the controller has a global view of the system, it possesses a topology graph of the network. There are three paths available between the client and server, two of them being Pareto-optimal. Any unspecified throughput is assumed to be 1 Gbps and unspecified latency to be 0 ms.

The hosts (including the client and server) connect to the controller with control paths to query available paths and submit their choices. We use Chen et al.'s protocol [31] on these control paths. Every time a host wants to establish a connection with other hosts, it sends a request for paths to the controller. The controller runs ParetoBFS on the topology and returns a Pareto-optimal path set to the host. The host then chooses a path based on its preference, and asks the controller to set up the path. The controller then installs the respective flows on each switch to enable packet transmission. To automate the interaction described above, an NFQueue[7] application is deployed on the host machine for intercepting the initial packet and communicating with the controller (until the path choice has been made and the flows installed).

The prototype was deployed on GENI [18], a test bed for new network architectures. All the nodes, including the hosts, switches and controllers are all GENI virtual machines that run Ubuntu 14.04. The switches are virtual entities running OpenvSwitch[8]. The SDN controller used is Pox[9]. The path metrics are hardcoded into the topology when it is built. The controller is pre-configured to know these metrics. This prototype enables the end hosts to choose their paths, but introduces additional overhead than standard SDN, due to the increased interaction between the hosts and controller. To quantify the additional overhead, we measure the average time it takes to perform each step of setting up a connection (see Table 4.9).

The results here assume the choice is made instantly. The path query (which encapsulates the ParetoBFS runtime) and provisioning time depend on the round-trip time between

[7]`http://www.netfilter.org/projects/libnetfilter_queue/`

[8]`http://openvswitch.org/`

[9]`http://www.noxrepo.org/pox/about-pox/`

Table 4.9: Breakdown of connection setup times.

Task	Time (ms)
path query	7.93
ParetoBFS running time (included above)	0.59
Provision the path	18.67
Total connection setup time	26.60

the controller, switches and hosts. Here, it is in the order of several milliseconds. Since the topology is small, the ParetoBFS runtime is negligible compared to other overheads. For larger topologies with several thousands of nodes, the ParetoBFS runtime may increase to the order of seconds.

Once the flow is set up, packets flow through the data plane without any added overhead. According to our measurement, the goodput of HTTP file downloads on 10-Mbps and 1-Mbps path reaches 1.19 MByte/s and 116 KByte/s, respectively, approximating the allocated capacities.

4.10 Summary

Our work addresses the problem of finding multiple Pareto-optimal paths in a network where multiple criteria are used for routing. Such information is necessary in networks where path choices need to be provided to consumers for a posteriori selection. We have described ParetoBFS, an algorithm to find all the Pareto-optimal paths in a network. The experiments show that the algorithm works well and can get a solution on a typical network in reasonable time. We have also proposed several sampling techniques to further reduce the running time when finding all the Pareto-optimal paths is not necessary or not feasible. We have presented results from an SDN prototype to show that ParetoBFS is practically useful. We believe that this work presents an important step toward enabling novel routing techniques in modern networks.

CHAPTER 5
THE CASE FOR VERSIONING NETWORK STATE

This chapter hypothesizes the need to version network state, not just as a temporally chronicled input for our activity recognition algorithms, but also as an important troubleshooting tool by itself.

5.1 Introduction

One of the most popular ways to implement programmable networking has been to adopt devices that support a standardized application programming interface (API), for example OpenFlow [96], for programming network behavior in the data plane through external software. This network programmability brings with it several challenges: migrating to new architectures, interoperating with legacy networks, coordinating control plane state in a centralized fashion, supporting different authors to the state, and restricting anomalous or malicious behaviors etc.

While the aspect of multiple co-existing authors editing flow state of a shared resource has been previously investigated [74, 99, 113, 128] from the perspective of conflict detection or resource isolation, the issue of *tracking* all changes to the flow state has not yet been examined sufficiently. In software development, this process of managing changes to shared data is referred to as *revision control*[1]. We see a need for similar revisioning and state change management rapidly manifesting in software-defined networks.

[1]The popular revision control software `git` provides author tracking, versioning and timestamping of changes, immutability of past alterations, automated conflict resolution, and annotations.

Traditionally, revision control in the networking world has been limited to managing CLI configurations of data plane devices. The state of flows and the control plane, however, is inaccessible. In the context of a programmable network, we have access to much more state than was previously possible. Based on existing software versioning techniques, we propose a system called State-Git that provides network state versioning in the SDN context. State-Git underlies all network state management that a node undertakes, and provides safety at the data plane and better programming discipline in the control plane.

Additionally, our framework can be easily extended for several other purposes: tracking control plane evolution (similar to past works on AS path evolution [36]); providing network level accountability (similar in motivation to past works on AIP [8] or packet provenance [115]); preventing flow space conflicts across different authors (without being as restrictive as the isolation efforts of FlowVisor [127] or overlap prevention of Frenetic [99]); and extracting network state in a form conducive for automated troubleshooting (beyond methods made possible by header space analysis [72] and NDB [100]).

5.2 Background

Programmable networks allow external network applications to directly access the state of the data plane through well-defined APIs. The data plane device type (virtual or physical), flow table types (e.g., VLAN table, MAC table) and size, and supported feature set (match fields and supported actions) vary depending on the deployment scenario (e.g., edge-gateways [103], wide-area networks [119]). For example, an SDN-based gateway solution is required to process multiple Ethertypes (MPLS, VLAN, IPv4), different network protocol traffic (BGP, BFD, ARP, VRRP, etc.), and support tenant-specific requirements (QoS, service-level agreements). Typically, the packet processing functionality is achieved via a match-action pairs that span a multi-table pipeline. As the flow table evolves, through route updates, gratuitous ARP replies or new tenants addition, multiple tables and their related attributes (e.g., meters attached to each flows) may be updated or modified by dif-

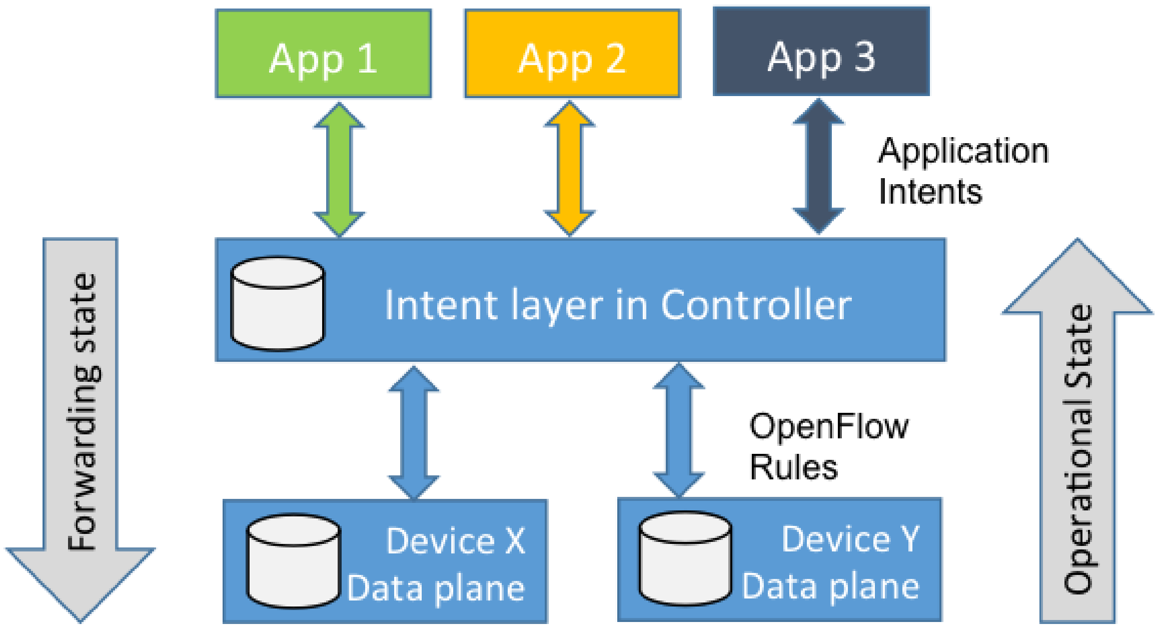


Figure 5.1: Programmability model in SDNs.

ferent applications/authors. All these characteristics have implications on the network state management.

A typical model of SDN solution is shown in Figure 5.1. One or more applications, either directly through the API or through an abstracted intent layer, edit the configuration state in the data plane, e.g., the flow table. Correspondingly, the data plane exports to the external applications certain operational state, e.g., statistics, relevant for the programming logic. We adopt this layered, decoupled model as our SDN programming model.

Such decoupling of the control and data planes in the context of SDN can be challenging when it comes to managing network state. Consequently, we only focus on the flow rule configuration state. Table 5.1 correlates each of SDN's programmability characteristics to a certain state management requirement. The safety and mutability requirements have been least investigated in the past, and we adopt it as the main focus here. In the subsequent sections, we describe our state management system and illustrate its use through examples.

Table 5.1: State management requirements.

Characteristics	State Requirements
Decoupled network control	Consistency
Dynamism in applications	Mutability
Co-existing applications/authors	Safety
Network troubleshooting	Provenance

5.3 STATE-GIT

In this section, we discuss our proposal for state management to improve network programmability.

5.3.1 State Management Abstraction

The existence of multiple authors attempting to operate on a shared resource(network state) dictates the following requirements for concurrent access and accountability:

- *Author or application tracking*: The system should possess the ability to exactly identify and report the author that was responsible for the change of network state.
- *Versioning and change tracking*: Transformation and evolution of network state can be tracked, providing valuable information that can be used for planning and provenance reasons.
- *State safety*: Preventing the alteration of network state that has been committed by a different author brings in safety that ensures deterministic system behavior. It is, however, important to allow the same author to alter actions for previously established match rule.
- *Conflict resolution and rebasing*: The mutability requirement in statement management brings with it a need to resolve conflicts or overlaps in flow space to a feasible extent. Rebasing is the process of merging changes to an extent allowable by a prespecified policy, based on the level of conflict.

- *Annotations*: The applications or authors can also include comments or other meta-data to be associated with the change.

In the software development world, the above features are commonly available in a revision control software, such as `git` [56]. By making an analogy between source code and network state, we envision that adopting a layer of revision control for managing network state becomes vital to the programmability of the network.

5.3.2 State-Git Architecture

We architect the abstraction described above in a manner similar to that used by `git` by adopting an "authoritative" State-Git server that keeps up-to-date state that has been committed and programmed onto the flow tables. In reality, this server and the authoritative copy of the state it protects can be placed in two locations:

- Co-located on the data plane devices: each data plane device can host a separate State-Git server and give it direct access to the flow table. With this approach, the state and its management are distributed. Each controller and its applications will act as State-Git clients that conduct transactions with each of these servers. If an operator were to make flow state edits on the device, they will use a client to achieve that.
- Located external to the device: the State-Git server is hosted external to the device, in a centralized location. All readers and writers to the flow state will use a State-Git client to access it. The flow state in the server is always kept in-sync with the flow table in the data plane devices.

To avoid the overhead on switch CPU and to keep state management simpler to reconstruct on a network-wide scale, we prefer the latter approach of the State-Git repository being external to the devices in a centralized server[2]. The implementation of this server becomes even simpler if one were to host it at a controller instance. For the remainder of our work here , we assume State-Git server being co-located architecturally with a controller instance.

In this approach, all readers (often switch) and writers (often controller), similar to `git`'s distributed approach, have their own local object database and staging indexes. Our architecture requires the writers to use a *flowmod monitor* module (running in the background) to monitor and analyze flow-modification messages received by the SDN/OpenFlow agent, as well as to version network state information that passes through it. The changes are then pushed to the State-Git server. The pushed changes are serialized to disk regularly. Figure 5.2 provides an architectural overview of this process. When a change is made and committed to the set of past changes, we create a new "snapshot", which is a copy of the flow state that is versioned and uniquely identified.

When the system initializes, the server sets up the revision control framework by creating an empty master branch as part of the State-Git server. Any data plane device that connects to the controller triggers the fork of a branch from the master's initial state. This new branch is then tagged with the datapath-id associated with the corresponding device. The branching behavior is illustrated in Figure 5.3. The device then clones this branch locally. The staging index and repository local to the device are used to store flow modifications occuring as a result of programming changes effected directly on it. The local changesets are, then, pushed to the State-Git server on the controller to keep the network state in sync. It is foreseeable that there can be policies to regulate how frequently state changes are pushed upstreeam. An instance of the *flowmod monitor* running on the switch handles these operations.

[2]The centralized server can be sufficiently replicated and prevented from being a weak link in the SDN programming.

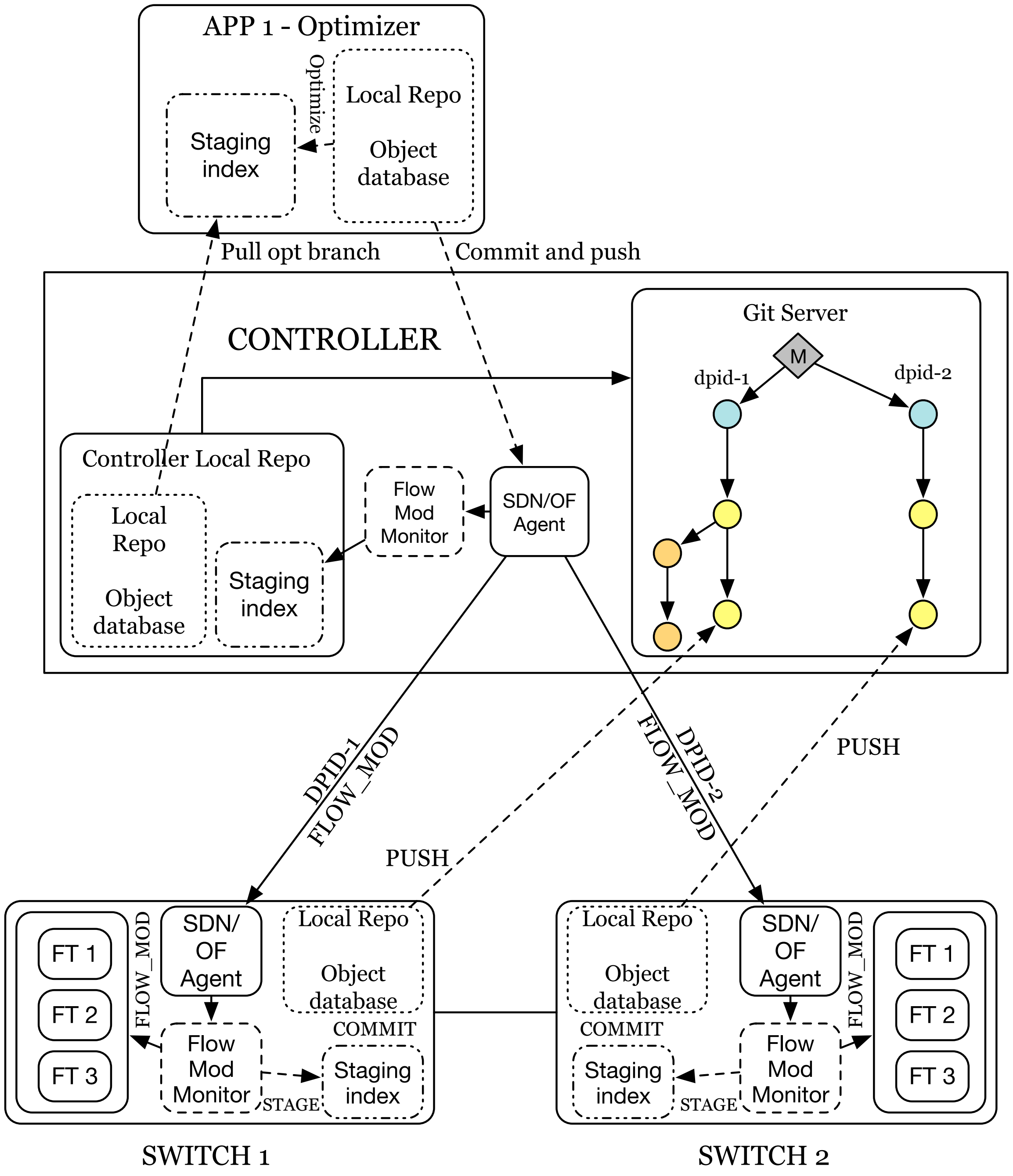


Figure 5.2: State-Git Architecture.

The flow information and metadata that is important to track changes across versions is represented as a facile data table in Table 5.2.

Revision control can be performed at two levels of granularity: ‘per-flow’ and ‘flow-table’. Based on the application and operational environment, it is understandable that an

Table 5.2: Configuration state model.

Type	Config data	Example
Timestamp	{time}	1425884076.342
Author	{app_uuid}	...1681e6b88ec1
Version	{change_uuid}	...a79d110c98fb
Table	{table_id}	5
Flow rule	{priority, match, action}	priority: 32768, match: tp_dst=22, action: drop
Timeout	{idle, hard}	idle: 10 hard: 0
Metadata	{OF config input}	vxlan_remote_ip= 1.1.1.5

user might pick a different granularity to perform state management at. The controller can choose to make available a snapshot at either granularity across its other instances, or the other instances can choose to directly "pull" from the State-Git repository. This ensures that all the instances are consistent with the underlying network state.

Although we (and the Table 5.2) primarily deal with managing flow state, there are other network states worth preserving, especially the operational state pertaining to liveness of switch resources. For instance, there are OpenFlow flow configurations, such as failover group actions, that specify actions conditional on the liveness of ports (logical and physical). During troubleshooting or provenance inspection, such actions may be incorrectly interpreted if the operational state of the ports is not correlated with the configuration state. To address this need, we add an additional agent for tracking this operational state and archiving it to a centralized store alongside the State-Git repository.

5.3.3 Conflict Resolution

During flow state programming, it is common to experience conflict with the flow space of an existing rule. In the past, SDN implementations based on OpenFlow relied on external mechanisms or implementations, such as FortNOX [113], FlowVisor [128] or that used in [102], to perform conflict resolution or provide flow space isolation. Recently, the Open-

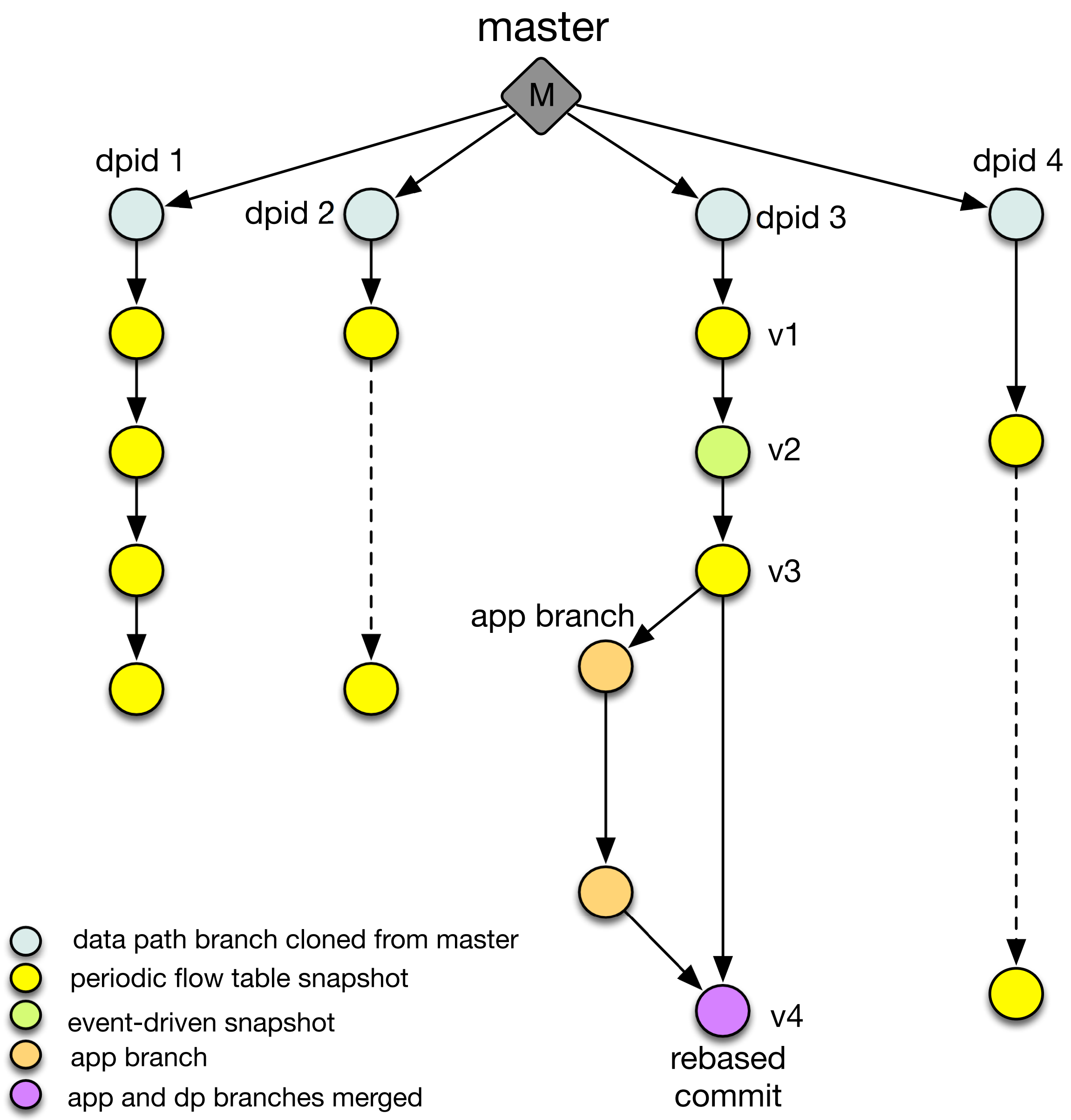


Figure 5.3: State-Git Branching Model.

Flow specification [107] included an option to mark a flag called `OFPFF_CHECK_OVERLAP` on a `FLOW_MOD` operation to force the data plane to verify if the current `FLOW_MOD` overlaps with an existing rule on the match set; in the event of an overlap, the switch must refuse the addition and respond with an OpenFlow error message. This overlap check, however, is too restrictive and insufficient when frequent flow state programming from multiple authors (applications) becomes common.

Revision control and rebasing (locally replaying and merging changes), as used by `git`, can be an important tool for programmers. In the State-Git architecture, we include the logic proposed in Algorithm 5 to attempt local conflict resolution to a feasible extent. The algorithm takes as input some merging policies, to specify what the action should be when State-Git detects a conflict and is unable to automatically resolve. Our approach relies on building a set of flow-spaces, overlapping and non-overlapping, for the new flow modification and the existing rule that it potentially overlaps with. Non-overlapping spaces are accepted right away. The overlapping space is checked against the current rule a second time to determine the set relation between the two and the requisite action to be performed.

The approach will need further extension to make the rebasing author-aware, i.e., only allowing merges based on the author privileges or based on whether changes are overwriting the same author's past commits. A `git`-like approach offers user authentication and privilege tracking to specify what source code can be modified by which user. We reserve all author-aware rebasing for future work.

5.4 State Versioning Use-Cases

To illustrate the value of State-Git, we discuss the following scenarios where versioning the flow table can be useful for network operations and explain the practical value of our abstraction:

- Understanding the evolution of network state maintained in the state table.
- Tracing and identifying security issues or misconfigurations in the network state.
- Adopting state extraction for network troubleshooting.

5.4.1 Tracking Flow Table Evolution

As discussed above, flow tables evolve over time through additions, deletions, and modifications. Current switch implementations and controller applications have limited capabilities to track these changes. Applications are required to periodically query the network

Algorithm 5 Conflict Resolution and Rebasing.

```
1: procedure $resolve\_rebase(dpid, flow\_mods)$
2:     for all $flow\_mod \in flow\_mods$ do
3:         $S \leftarrow$ current snapshot of flow space
4:         compare $flow\_mod$ to $S$
5:         if $disjoint(flow\_mod,$S$)$ then
6:             accept flow_mod
7:         else
8:             compute overlap(flow_mod, S)
9:             $X \leftarrow$ non-overlapping space
10:            $Y \leftarrow$ overlapping space
11:            $Z \subset S, match(Z) = match(Y)$
12:            accept $X$
13:            if $(Y, Z)$ is exact_overlap then
14:                $action(Y) \leftarrow action(Z)$
15:            else if $Y \supset Z$ then
16:                apply superset merge policy
17:            else
18:                apply subset merge policy
19:            end if
20:        end if
21:        commit flow_mod
22:    end for
23: end procedure
```

state maintained in the switch and parse that information to comprehend the updates made to each flow. This approach is complex and might require state management in application logic. State-Git can provide basic snapshotting and inspection features to efficiently track the evolution of the flow table state. Here we discuss some simple versioning features to highlights its value:

- *gitflow commit*: The *commit* feature takes a snapshot of the changes made to the flow table from the previous committed state. When the controller programs flow modification messages (e.g, either a single or bundle of flows), State-Git considers this as a new commit. When flows are programmed in the flow table, the committed changes are merged to the master copy and recorded as a version of the network state. A commit highlights valuable information on changes made to the flow table and eases an application or administrator to track the evolution of the flow table.

- *gitflow diff*: With *diff*, GitFlow can compare and highlight the updates between commits or between the current master state and any previously snapshotted version. This helps in comprehending the modifications made to individual flows, specific tables and the updated actions.
- *gitflow tag*: Tagging is a critical component in State-Git. Flow modifications (commits) can be annotated with tags to carry metadata information about the updates. For example, a security update from a firewall application can add annotations specific to the application and provide descriptions about the updates.
- *gitflow grep*: *grep* searches for the specified input pattern (e.g., all entries matching on port 80, or tagged commits) either within a single commit or across commits.

5.4.2 Investigating Security Issues

Multiple interfaces (sets of controllers, side-channel access via SSH logins, malicious applications with flow-level access) with write access to the flow table can introduce security issues in the data plane. A security violation or malicious update of any manner can impact the packet forwarding functionality, thereby bringing down the network. For example, network operators can login to the switch and introduce simple flow updates that can undermine the network policies maintained by the applications. Similarly, multiple controllers in EQUAL roles can introduce conflicting flow behavior that can violate the security policies programmed in the switch. Existing security enforcement kernels are built as an intermediate layer between the controller and the switches to detect such violations, however, they lack the feature to track the local changes made in the data plane (e.g., via SSH access). We exhibit some State-Git features that can complement such security offerings:

- *gitflow blame*: The *blame* feature provides information about the changes and the entity/user that modified the interested flow entries. Additional annotations such as timestamps, impacted tenant information, updates (revisions) to individual flows are provided to detail the interested changes.
- *gitflow bisect*: *bisect* identifies the changes that introduced a bug or violation between two versions of the flow table. In addition, the options provided with *bisect* can help operators backtrack the life of a packet in the data path (e.g., set of tables which processed the packet in the multi-table pipeline).
- *gitflow reset*: While debugging a flow related security issue, an application (or operator) can make use of *reset* feature to revert the current flow table to a specific secure state. For example, a compromised controller entity can introduce a flow update to re-direct data traffic to the master controller (via send to controller action); if the operator identifies such a flow update as a security violation, the reset command can help reset the flow table state to the last known working state and discard all changes made to flow table since the previous committed version.

5.4.3 Revision Control in Troubleshooting

Past work on network troubleshooting [61, 72, 100, 146] highlight how flow space can prove handy to identify issues. State-Git allows extracting different snapshots of the flow space, thereby empowering troubleshooting. This section attempts to address some important questions posed by Heller et al. [61] in their analysis for troubleshooting SDNs and how State-Git can assist in these scenarios.

1. **How can we integrate program semantics into network troubleshooting tools?** We discusses network equivalents to *gdb, valgrind, gprof*, but interestingly, does not talk about revision management. Network specific versions of these tools help identify errant network states, but revision control can ensure that these situations do

not occur again. Troubleshooting information can be stored as version metadata that can assist in avoiding faulty network configurations from occurring again.

2. **How can we integrate troubleshooting information into network control programs?** The presence of multiple versions of network state actually help troubleshooting tools revert to a working version in the case of mis-configurations, failures or such. The use-cases presented in this section reinforce this point.

3. **What abstractions are useful for troubleshooting?** Exposing and abstracting the network state is in itself an interesting proposition. Previously, the absence of accountability was a major driver in not exposing network state to applications running on the controller. With revision control, we have an extra layer of backup to make sure the exposed network state is not misused.

As networks grow to be more autonomous and self-healing in nature, automated troubleshooting tools become a focal point of operations. Flow-level revision control can potentially aid these tools exercise a more intricate level of inspection than just investigate changes to the flow tables.

These illustrative use-cases represent the necessity for versioning across different disciplines such as security and troubleshooting. We believe that State-Git makes a strong case towards filling an existing gap in network state management for software-defined networks.

5.5 Summary

Based on the premise that SDN can also benefit from source code development and troubleshooting tools in the software world, we proposed State-Git as an important abstraction for flow state management. We show several examples where adopting revision control in the network programming workflow allows us to achieve higher level of safety, provenance, ease of programmability, and support for multiple applications.

While the use cases discussed above are mostly investigating single switch or single table scenarios, we envision and expect our revision control system to provide substantial support in network-wide and multi-table state management. Our revision control framework can be used to correlate state across multiple switches to identify related disruptive changes, or across multiple flow tables to identify the exact sequence of rules matched for a packet.

As part of our future work, we will implement State-Git in a realistic SDN deployment setting. The usefulness of its features will be evaluated against the presented use cases and overhead implications investigated.

State-Git, as an abstraction, is easily extensible and allows for adding intelligence on how the local copy of the state is mutated. Our framework can be reused for other purposes, including batch transactions, state rollbacks, and clustering consistency. We can also extend State-Git to address network state optimizations, such as conserving flow table space. As the flow table state evolves over time, it is critical to maintain the optimal state in the forwarding agent as well as handle the limitation of flow table size (i.e., TCAM limitation). Efficient aggregation schemes address this problem by reducing the number of flow entries, but still maintaining the same forwarding behavior. State-Git can facilitate such optimizations with efficient resolution features.

CHAPTER 6

ACTIVITY AND BEHAVIOR RECOGNITION FOR NETWORKS

In this chapter, we discuss a sub-area of artificial intelligence, more precisely of machine learning, called activity recognition. We then hypothesize how activity recognition and topic modeling can be applied to analyzing network state information.

6.1 The How and Why of Activity Recognition

Activity recognition is a challenge in artificial intelligence research that addresses identifying high-level descriptions of sequences of events represented using low-level information. Approaches for analyzing the low-level information often rely on machine learning algorithms. The main focus of such activity recognition research is to determine optimal data formats so that the learned model yields the best classification results. Supervised learning approaches convert the low-level data into a list of features that can distinguish between each sequence of information and use a high-level label to note the respective activity. Then the supervised learning algorithm of choice uses this data to find a function that identifies the best label for a given list of features. This however requires manual labeling of all the sequences by an expert; a process that can be tiresome and/or expensive.

An alternative that avoids the need for labeling the dataset is unsupervised learning. The algorithms are now tasked with finding clusters of sequences or sequence elements whose features appear to be similar by some optimization criteria. Due to the absence of labels, the learned high-level descriptions are simply generic labels that may be annotated as before at a smaller scale. However, the number of clusters to find is unknown and usually provided as a parameter for the optimization criteria.

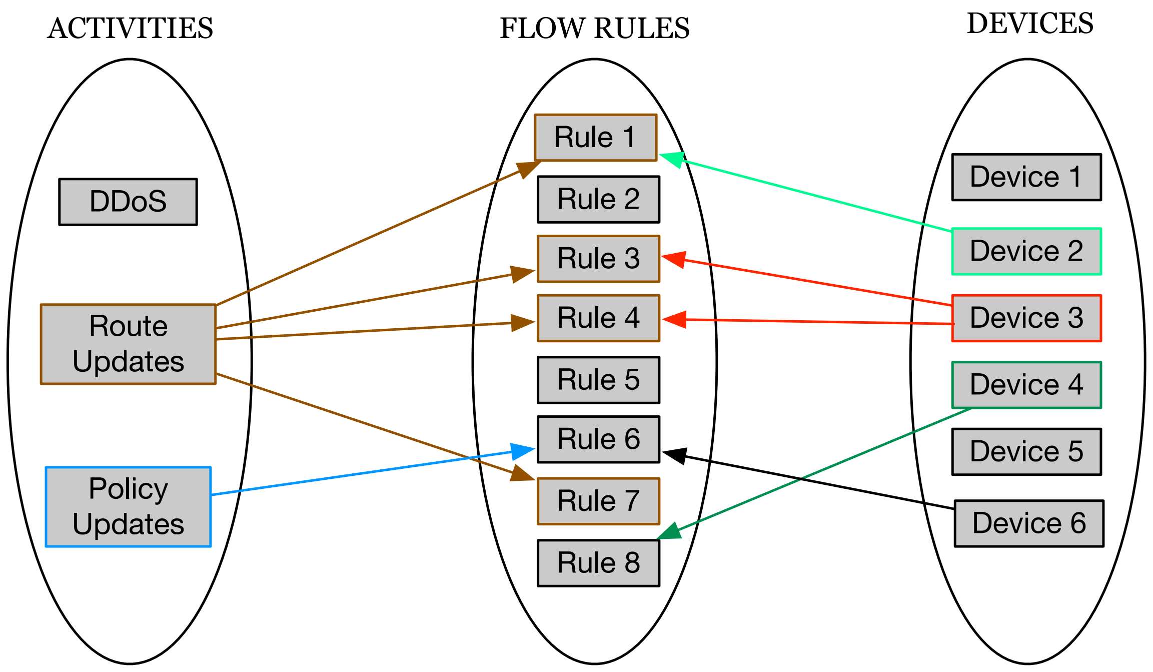


Figure 6.1: Relationship of high-level activity and low-level network state with data-plane devices.

We use the work by Freedman et al. [50] [51] as our basis for activity recognition. Even though the work pertains to developing quantitative representations of RGB-Depth sensor inputs that are visually interpretable for humans, we believe that the underlying principle can be adapted for network activity recognition. Being able to correlate this low-level information to high-level network activity that a human operator can comprehend easily is the focus of our research.

6.1.1 Background

It seems difficult to justify identifying a seemingly arbitrary number of activities without any discerning label. Consequently, Huynh et al. [64] provided strong empirical evidence for the success of activity recognition using the well-known Latent Dirichlet Allocation (LDA) topic model [21], which was initially designed for identifying clusters of semantically similar words throughout collections of text documents. Their verification method presented a time-line of the observed user during a sensor data's collection duration and overlaid it with the clusters assigned to the regions of collected data — each distinguishable event in the time-line had its own cluster assignment. However, many unsupervised datasets for activity recognition lack such a time-line or labeling method and simply enter the data's feature list into the algorithm, try multiple numbers of clusters, and then take the one with the best performance likelihood. This is an advantage of supervised learning methods, but the rigidity of predefined activities means that novel inputs cannot be recognized properly due to the specificity of labels.

Freedman et al. [50] [51] focus on building a variant of the aforementioned LDA that accounts for additional information such as temporal consistency and environmental information that can disambiguate similar activities. This variation of the LDA suits our network use-case quite well, since the requirement here is to detect network activity in a temporally consistent manner. Figure 6.2 illustrates the generic LDA from the work. Our research involves tailoring this LDA for a topic model that is network activity specific. The black nodes represent the vanilla LDA module, blue nodes represent a hierarchical hidden Markov model module for temporal consistency, and the red nodes represent parallel LDA modules for objects in the environment with which the observed subject interacts.

Automated interpretations of unsupervised learning-derived models for such applications have not been widely studied, and Kim et al.'s [75] Bayesian Case Model is one of the few other known works in this area. It was not designed specifically for activity recognition, and thus assumes that a single input in the cluster can represent it entirely. Because it can still be difficult to identify which features are most important for each cluster's definition, Freedman and Zilberstein [52] introduce a higher-level list of features that can describe the findings more qualitatively, and the lists can be combined to autonomously generate a qualitative list of features that also describe the more commonly shared features of the cluster. They instead find an "average description" over all the inputs in the cluster with a potential for disjunctions of descriptions to account for the variability of features that may take place in an activity.

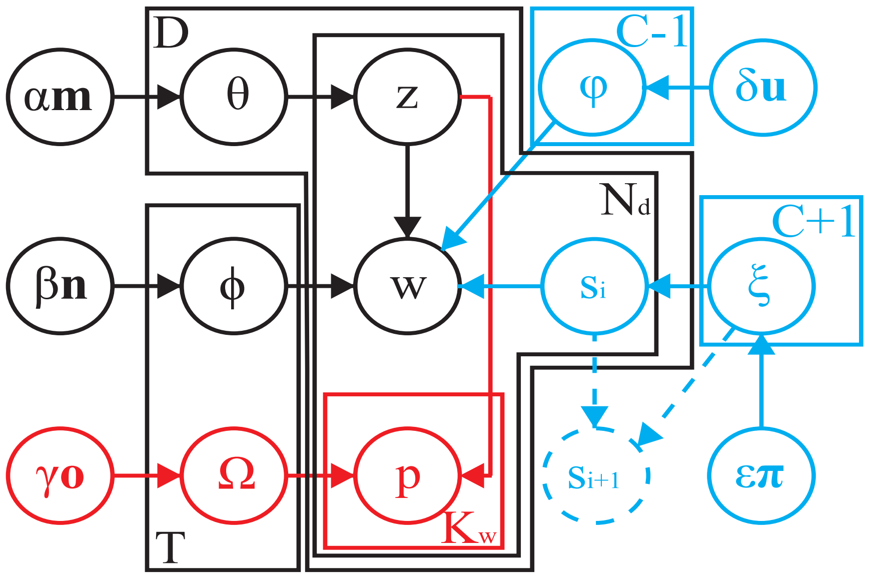


Figure 6.2: Graphical model representation of Freedman et al.'s activity recognition method.

6.1.2 An Illustration of Activity Recognition for Network Management

We specifically investigate the feasibility of adapting the LDA discussed in Section 6.1.1 to a topic model for network-based activities.

A single epoch's snap-shot of the network yields a low-level representation, and a sequence of updates will produce an evolution of a certain network entity, flow, connection or device. We believe that this may be attributed to activities performed in/on the network itself. Possible activities/events to expect over an enterprise network include node, topology and path changes, traffic patterns, malicious attacks, route fluctuations, service disruptions etc. Other interesting avenues to explore would be to create communication "signatures" for each pair (or set in the case of broadcast/multicast) of communicating hosts to be incorporated into data models etc. Events could vary depending on the network's specifications and purpose.

Section 1.2 illustrates the motivation for this work with an example. Supervised machine learning methods will provide the benefit of identifying the most common activities that will be of importance, but they will also require large amounts of manual annotations to specify activities. Furthermore, the aforementioned issues that are unique for each client would be dismissed unless a sufficient number of examples were recorded during training. On the other hand, unsupervised methods will be able to develop their own clusters of network configurations for each presumed activity. The biggest challenge with such clusters is explaining them to system administrators who will have to interpret these activities perceived by the machine. Looking at a collection of network state changes is not a trivial means of diagnosis; otherwise, a manual perusal of the data would suffice. We intend to facilitate this task by identifying potential activities that may be occurring throughout the network over time and presenting them to the system administrators. This should be able to assist them in making decisions about the network as well as what to do in the eventuality of any impending or predicted problems.

We believe exploring various representations of the recorded information may identify regions of interest in the network that pertain to each recognized activity. Trivial representations will display the entire network with all its information. Flow-specific data and routing information can generate additional features that better describe network reachability and evolution for system administrators and typical users of network management software.

6.2 Topic Modeling Applied to Networking

When it comes to networking and management, formulating the problem in a specific, formal manner is hard. When we are looking to identify activities and events on a network, being specific would mean listing out the hundreds of thousands of events that occur on diverse networks. What complicates matters further is the subjectivity in event identification and the understanding attached to that subjective evaluation. One network administrator would could consider a certain behavior normal for their network, while it could be diametrically opposite for another administrator managing a different network.

But what is topic modeling and how it is useful? Topic models are a class of algorithms that "magically" discern what topics different documents talk about. It is a fairly versatile area that can be applicable to a variety of applications, of which we believe, networking is one. The algorithms parse the documents and cluster the document vocabulary into different topics (clusters of words that may occur together with some probability). Ultimately, it is an attempt to infuse vocabulary with semantic meaning. Topic modeling is a simple and powerful tool to run on a corpus of documents to make some semantic sense. The subsequent problem that arises is that none of the topics have any labels associated with them. The models require us to interpret them meaningfully, and that, can be done using supervised classifiers as explained a little later in this section.

The authors in [65] lucidly formulate the challenges of their domain in the quote below. Even though their work pertains to anomaly detection for video surveillance, the similarities of the problem at hand are striking and the applicability of this statement to networking is quite compelling.

> *Behavior analysis is an important area in intelligent video surveillance, where abnormal behaviour detection is a difficult problem. One of the challenges in this field is informality of the problem formulation. Due to the broad scope of applications and desired objectives there is no unique way, in which normal or abnormal behaviour can be described. In general, the objective is to detect unusual events and inform in due course a human operator about them. This paper considers a probabilistic framework for anomaly detection, where less probable events are labelled as abnormal.*

Subsequently, the reasoning presented above drives us to design this alerter framework as a probabilistic one, rather than a deterministic one. Section 6.1 briefly talks about the differences between supervised and unsupervised methods of learning. Abnormal behavior detection can be re-formulated as a classification problem (which is supervised), but only as a single-class problem, since it is labor intensive and difficult to collect and tag all kinds of abnormal behavior. Therefore, one-class classifiers are quite applicable to these cases, for example, a one-class Support Vector Machine (SVM) [32]etc. When we talk about analyzing temporal correlations of how the network has behaved at certain instances and identifying patterns, we use topic modeling (which is unsupervised in nature) to break down the dataset into smaller clusters that a supervised classifier can then run through quickly.

6.3 Behavior and Anomaly Detection

In the initial phase of this work, we use commonly available topic models on network state data to show that topic modeling can break down the problem in clusters that are quicker to mine for patterns. More specifically, we use the MAchine Learning for LanguagE Toolkit (MALLET) to perform the analysis [92]. The dataset construction is detailed below.

6.3.1 Dataset Characterization

For the purposes of this illustration, we use BGP message dump datasets from the RIPE Routing Information Service (RIS) database. The dataset is prepared as follows:

- The RIPE-RIS BGP collectors dump BGP message at 5-minute intervals every day. We use five of the collectors, RRC01, RRC04-07.
- We form the dataset over the period of one month, from September 8th, 2017 to October 8th 2017. We pick five 5-minute intervals every day, from every collector.
- We use the BGPStream [108] site to look for potential BGP hijacks that have occurred in the one month time-frame. Subsequently, we collect the message dumps for the time-intervals identified with potential prefix hijacks and tag the files also as potential hijacks. For this illustration, we do not consider partial prefix hijacks, only complete ones.
- Each 5-minute interval or hijacked interval becomes a document for the topic model.

Table 6.1: Sample BGP Message.

BGP Field	BGP Field Value
TIME	09/07/17 00:00:04
TYPE	BGP4MP/MESSAGE/Update
FROM	195.66.224.159 AS8966
TO	195.66.225.241 AS12654
ORIGIN	IGP
ASPATH	8966 17557 9260 132165
NEXTHOP	195.66.224.159
COMMUNITY	8966:53 8966:8888 17557:65111
ANNOUNCE	111.88.223.0/24 111.88.222.0/24 111.88.221.0/24 111.88.220.0/24 111.88.219.0/24 111.88.218.0/24

6.3.2 Analytical Methodology

The BGP MRT/RIB dumps are parsed with RIPE's *libbgpdump* to convert the zipped dumps into actual messages. An example of a parsed BGP message is illustrated in Table 6.1.

There is a singular advantage to using topic modeling. Since the algorithms are designed to work well with documents and vocabularies of unique 'words', it is up to the domain to define what a 'word' is and what comprises a 'document'. For example, the authors in [65] use topic modeling for video analysis, where cell information constitutes a 'visual word' and video segments of certain lengths form 'documents'. This is advantageous in our case because we can develop models for different events by defining different types of 'network-words'.

If we take the network connection table as the source of information, then considering a (*src_ip*, *dst_ip*) pair as a single 'word' will allow us to cluster connection information using topic modeling. This way we can understand node communication patterns, discern any irregularities etc.

In this BGP message example, since we want to identify prefix hijacks, the pieces of information that we require are the prefix's originating ASN, whether the ASN is announcing or withdrawing the prefix and the prefix itself. Using the contents of the message in Table 6.1, we see the originating ASN is **132165** (from the last entry in the ASPATH) and there are quite a few prefixes being announced. This information allows us to form a number of unique words in the format *ASN-ANNOUNCE/WITHDRAW-PREFIX*, such as "132165-ANNOUNCE-111.88.222.0/24", "132165-ANNOUNCE-111.88.221.0/24" and so on. This allows the topic modeling algorithms to cluster around (ASN, prefix) announcements and withdrawals.

Our dataset for a month, has close to 920 documents, totaling more than 20 million entries, hijacked and non-hijacked. Depending on the level of accuracy and the nature of data, running classifiers will take time to train and test. For example, running a K-Nearest Neighbors classifier over 20 million individual entries takes over an hour of training time and over two hours of testing time, but is very accurate. Introducing a semantic layer of abstraction between the data and classification algorithms. Introducing a semantic layer of abstraction between the data and classification algorithms breaks the dataset down into a fraction of its size.

The goal of this evaluation is two-fold:

1. Primarily demonstrate that topic modeling can be adapted to work with network state. If we can achieve a high degree of accuracy by using generic activity recognition algorithms, it lays down a precedence for using these methods with any kind of network data, and

2. Reduces the computational and spatial requirements for analyzing large amounts of network data in the future, while still maintaining a high level of accuracy.

Table 6.2: TP/FP ratios for different classifiers.

Classifier Type	**Topic Count**					
	5 Topics	6 Topics	7 Topics	8 Topics	9 Topics	10 Topics
Gaussian Bayes (Direct)	0.0/0.0					
D-Tree (Direct)	1.0/1.0					
Nearest Centroid (Direct)	0.76/0.74					
Gaussian Bayes	0.72/0.24		0.96/0.34	0.84/0.21	0.16/0.00	0.56/0.05
Decision Tree	0.12/0.02	0.44/0.10	0.04/0.00	0.40/0.05	0.88/0.28	0.88/0.2
Nearest Centroid	0.84/0.29	0.88/0.28	0.92/0.22	0.84/0.22	0.92/0.33	0.96/0.35

6.3.3 Modeling and Classification Results

In this section, we summarily illustrate the comparison between running the dataset described in Section 6.3.1 directly using a classifier and after introducing an topic modeling layer between them.

We use three different classifiers to test for accuracy and performance. The Gaussian-based Naive-Bayes classifier is just used as a baseline to understand classification behavior. The other two classifiers are a Decision-Tree classifier and a Rocchio classifier. The latter operates on the basis of finding the centroid nearest to a data-point.

Every run of the topic model is different in how it probabilistically determines what the word-topic allocation is. Subsequently, each topic model execution, for n topics, is run 10 iterations to ensure we have a consistent topic distribution reading. We then pick the results of a single iteration at random to analyze with a classifier. To avoid over-fitting or under-fitting the data, each run of the classifier is validated with a $k - fold$ cross-validation technique. We use $k = 10$ for our runs.

For the topic-modeling runs, the training sample contains 205 non-hijack cases and 25 hijack cases (based off a 80-20 split from the original dataset for training and testing). The ratios are represented with respect to these numbers. The results are tabulated above in Table 6.2.

The numbers in the table do not tell us much with respect to direct classification. It is as much as we expected with the decision-tree classifier scoring a 100% accuracy on all counts, hijack and non-hijack. What is interesting though, is that the same classifier demonstrates an extremely poor level of performance when the topic model layer is introduced in between. The nature of data being generated as an output by the topic model compels us to look for alternatives to the D-Tree classifier. Since the number of data points are not too high, we opt for a bounding classifier such as the Nearest Centroid (NC) or Rocchio classifier.

Using the NC classifier on the topic model output is interesting in a way that it is able to maintain a high degree of accuracy for a majority of the time, but only with the hijacked classes. It is a bit over-sensitive in its approach and hence the false positive count is incomparable to what the D-Tree obtains. We believe this phenomenon is due to the absence of domain information in the topic model. It is quite possible to bring the false positive count down with custom-designed topic models for network state.

Figure 6.3 plots the false positive count against the topic count for 10 iterations of every run of the topic model. It is clear from the graph that one of the ways to reduce the false positive count is empirically determine the best number of topics to group the network vocabulary into. But this is going to reduce the count only to a certain extent. To make it as efficient as possible, the activity recognition methods need to have domain information encoded in them. Building such a model is one of the primary objectives of this framework.

6.4 Summary and Conclusion

In this section, we explored the possibility of utilizing topic modeling to identify anomalies in network state information. Based on the results, it appears that the topic modeling or bag of words based approach is not extremely suited for this task specifically. Conse-

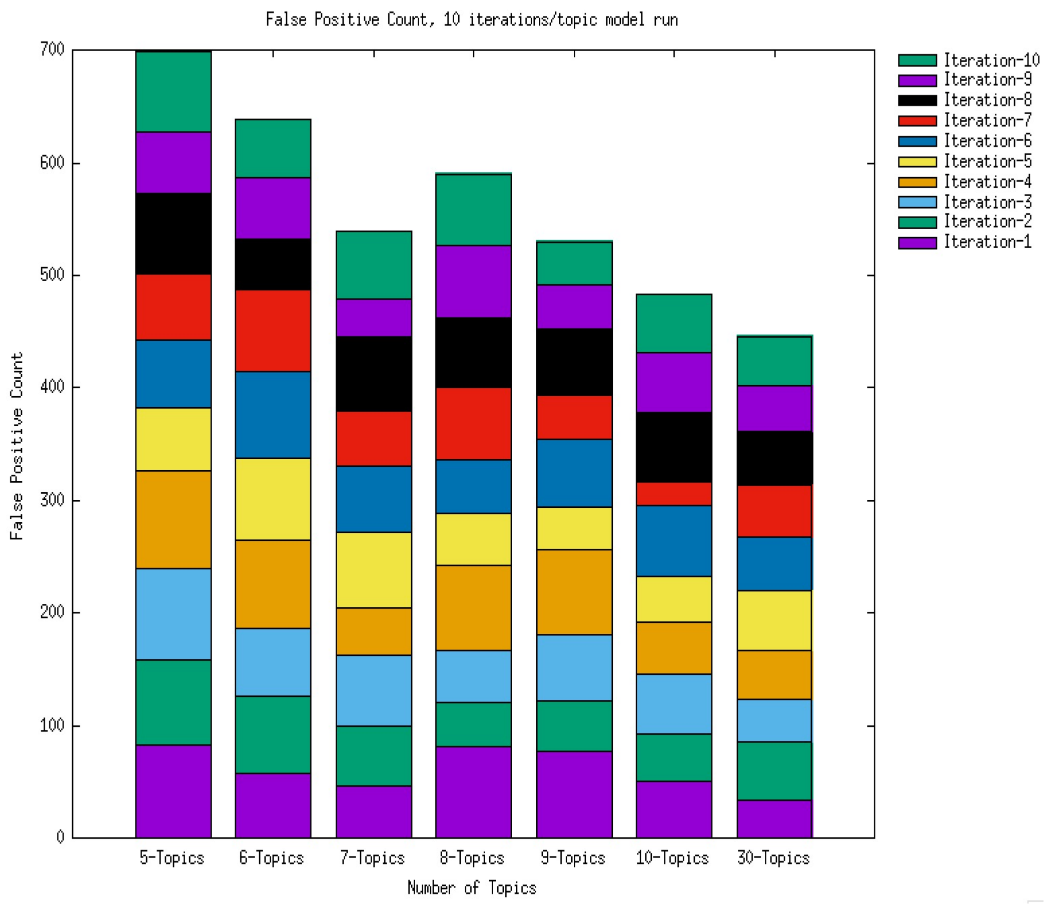


Figure 6.3: False positive count for topic modeling.

quently, we require other reliable methodologies for network state analysis. In the next set of chapters, we explore the potential of natural language understanding and concepts derived from that area to try and make sense of network state information.

CHAPTER 7

USING NATURAL LANGUAGE CONCEPTS TO ANALYZE NETWORK STATE

7.1 Introduction

Networked systems today operate at an unprecedented scale. Data communication networks provide core functionality for today's distributed services and applications that span personal, business, and government use. Various developments in technology, networking or otherwise, continue to put a massive strain on these networked devices in our daily lives. Virtualization is re-defining hyper-scale on-demand computing by pushing the boundaries of how much can be achieved on a single hardware device. Containerization is revolutionizing operational time-scales in DevOps and traditional design-develop-test cycles. The Internet of Things promises to connect every device to the Internet. Each of these technologies place demands of varying nature on communication networks, such as real-time responsiveness, rapidly changing traffic workloads or security concerns [48].

The advent of programmable networks that provide fine-grained control of network traffic have given rise to new challenges in network management. It is now critical to maintain correct operation of these networks to avoid adverse business outcomes. Unfortunately, existing network management methodologies have not evolved at the same pace as these networking technologies and architectures. Subsequently, current network management practices do not provide adequate solutions for highly dynamic, programmable environments. This is summed up perfectly by:

"The widespread integration of these applications into our daily lives raises the bar for network management, as users elevate their expectations for real-time interaction, high availability, resilience to attack, ubiquitous access, and scale." [48]

Our research attempts to explores a key question in network management – can the power of machine learning be efficiently harnessed such that network administrators can make better decisions in highly dynamic environments? Specifically, we propose an approach to network management that is conceptually based on statistical Natural Language Processing (NLP).

Today, data is available in abundance. Such data includes log files, routing tables, BGP tables, IP tables etc. With machine learning making great advances, we believe this data can be put to good use. Computational linguistics tries to characterize and explain the expressions appearing in the language, taking into account semantics and context [89]. What if network management could be improved with the help of a framework developed specifically to understand network vocabulary, terminology, semantics and context? Such an approach that can be trained to analyze the semantics and context behind network paradigms, etc. We believe that network management can definitely benefit from a Network Processing Language framework.

The contributions are as follows:

1. We state the case for developing a network processing language,
2. We demonstrate how collocations can be used to process network management data, and
3. We evaluate a BGP dataset with these approaches and show that it can be used to successfully detect route prefix and sub-prefix hijacks.

The primary motive here is to present the need for a framework that can contextually and semantically construct reasoning when provided with network state and information. We utilize BGP routing data only to illustrate the feasibility of using natural language constructs as a source of inspiration to analyze network data.

Section 7.3 discusses the need to start embedding intelligence in the network itself, with the aim of improving network management. Subsequent sections 7.3.1 and 7.3.2 deal with the potential advantages of developing a network processing framework and constructs from statistical NLP. Section 7.4 characterizes the dataset and tools used. An illustration of detecting route prefix hijacks on BGP data is discussed in Section 7.5 and preliminary results are presented in Section 7.6. Section 7.7 summarizes our contributions.

7.2 Related Work

Traditionally, machine learning has been used in some form or the other to aid network analysis. Pytheas [71] explores a data-driven method to optimize quality of experience (QoE) using machine-learning techniques. Li et al. provide a novel machine learning based approach for efficient traffic classification [81]. A really old approach by Sasisekharan et al. [122] demonstrates that such machine-aided and data driven approaches are not new to networking or network management. They have been around for a long period of time, but now are at the forefront of research and development due to their capabilities and applicabilities to problems being seen in present-day networks, especially the newer architectures.

A holistic view of network management will tell us many problems that programmable architectures face are not even new. They are newer manifestations of solved problems from legacy networks that are re-appearing due to the scale and levels of dynamism present today.

Some of the conceptual ideas for our research are tenets of linguistic science, which are explored and explained in lucid detail by Manning and Schutze in their seminal book “Foundations of Statistical Natural Language Processing” [89]. This research borrows just some fundamental concepts to demonstrate that these can be generalized and extended to networking too. We do this by applying these concepts on public BGP routing data from the RIPE-RIS database.

The Border Gateway Protocol is a research world unto itself, to the extent that the subject matter is voluminous. Much of our understanding of ASes, their relationships, BGP anomalies and misconfiguration has been drawn from [11, 20, 53, 88, 114, 149]. Machine learning has previously been used to analyze and understand BGP data. Li et al. [78] use machine learning to extract features from RouteViews [4] data and present an Internet Routing Forensic Framework to reason about BGP anomalies. [6] talks about a 2-stage machine-learning model for BGP anomaly detection and [87] approach the same problem by considering prefix visibility.

A comprehensive summarization of a wide-range of BGP anomaly detection techniques is presented in [5]. We use ARTEMIS [124] as our primary source of information for state-of-the-art and to draw parallels with our methods of detecting BGP route anomalies.

7.3 Augmenting Network Management with Intelligence

Over the years, advances in network management have lagged behind those in protocols, application, architectures and designs. But as we keep falling behind the curve, it is only going to get harder to manage the extremely dynamic networking devices and architectures in use today. For network management to improve, the network needs to change [48]. It is essential to infuse the network with a certain degree of intelligence that will allow it to assist us in making decisions. We do not envision this approach in the context of entirely replacing the human operator, but bolstering the quality of information being provided, thus leading to enhanced decision making capabilities.

7.3.1 The Need for Network Language Processing

The advantages of network programmability are well documented [16, 101, 126] with programmatic networks being extensively researched in both the industry and academia [105]. Google's B4 WAN [68] is a prime example of a wide-area programmable deployment. The presence of various open source programmable frameworks such as OpenContrail [10], OpenStack, OP-NFV etc. bear evidence of industrial interest in the area.

In recent times, there has been a surge of interest in developing languages for networks. Most of the languages have been dictated by domain and application requirements rather than general purpose network programmability. One of the more notables ones advocating for platform and protocol independence on the data-plane has been P4 [22]. There have been other ones such as NetKAT that are based on mathematical foundations and equational theory [9], rather than ad-hoc programmability, Frenetic [117] that facilitates building SDN applications in a facile manner. But most of these languages focus on packet manipulation, data-plane operations, telemetry and application programming. Their focal point has been to abstract the network away from the data-plane, thus raising the level of abstraction away from hardware, and making the network easier to program.

If network programmability can be applied to almost all the aspects and component layers of the architecture, then why not for network management? It is our belief that developing a network processing language framework for managing these new, hyper-scale architectures merits thorough investigation.

7.3.2 Statistical Sequence-Based Analysis

Drawing parallels from [89], we can define two basic questions, which if answered, should be able to broadly capture the essence of a language. (i) what are the kinds of things networking devices are "saying"?, and (ii) what do these things tell us about the "world" (in our case, the network)?

There is a clear distinction here with the scope of these two questions. The first question deals primarily with structural aspects of the language. The latter addresses semantics and a contextual understanding of the information, relative to the greater whole. In a networking context, we know what devices are "saying". This is network state, both hard and soft-state [70]. At this point, the state is constitutionally just raw information, without structure, semantics or context. A network language framework will allow this data to have a "linguistic" structure. A structure, such as NetKAT's [9] will help extract knowledge from it. This helps put information in context to the "network world", making it useful and providing us with better insights into network operation, thus improving network management.

Here, we elaborate on a few concepts from NLP that we use in our research. A sequence is a collection of entities (here, words) that occur one after the other. In our BGP example, it might be the *peer-address* and *peer-asn* occurring one after the other. They just appear together, and possess a pseudo-semantic meaning, only telling us where the BGP message is coming from. Alternatively, a “collocation” is by definition a pair or group of words that are habitually juxtaposed. In linguistic parlance, this means “the whole is perceived to have an existence beyond the sum of the parts” [89]. For example, the words in the ASPATH are semantically related to each other, and offer us more information together as a whole. Even if the ASPATH is broken down into individual bigrams, each bigram still retains a semantic significance, because it tells us what edges are more popular than the others (based on pure statistical inference). We utilize collocations in BGP dataset analysis to identify prefix announcements/withdrawals and pathlets that make up the ASPath.

We look for collocations and sequences in the resulting text in both cases of analyzing a corpus of BGP message or a live stream. The subtle difference between a collocation or a sequence comes from the input. A collocation finder works extremely well on large corpora, which would be the offline approach. The parser, while looking to infuse semantic meaning, evaluates collocations not just when they occur, but their occurrences in the corpora as a whole. But then, a live BGP stream does not afford us the luxury of a huge corpora. Consequently, we look for sequences in each BGP message and then condition the sequence statistically, building conditional probability distributions around the sequences.

7.4 Tools and Datasets

We build our dataset using publicly available control-plane monitoring resources, such as RIPE’s Routing Information System (RIS) [118]. RIS is connected to 21 route collectors

Table 7.1: Sample BGP Message.

BGP Field	BGP Field Value
TIME	09/07/17 00:00:04
TYPE	BGP4MP/MESSAGE/Update
FROM	195.66.224.159 AS8966
TO	195.66.225.241 AS12654
ORIGIN	IGP
ASPATH	8966 17557 9260 132165
NEXTHOP	195.66.224.159
ANNOUNCE	111.88.223.0/24 111.88.222.0/24

in geographically diverse locations, peering with ≈ 300 ASes. The route collectors provide both RiB dumps and updates collected from monitors.

We also use CAIDA's BGPStream [109] framework. This enables live streaming of BGP messages from the RouteViews [4] project, CAIDA's own OpenBMP and RIPE's RIS collectors. Some RIS collectors live-stream updates and information in available on other collectors typically within minutes. OpenBMP also supports low-latency streaming for applications that do monitoring at shorter time-scales.

Table 7.1 illustrates a BGP update message in human-readable format. The corpus that we use to train the offline model uses a slightly different, one-entry per line format that is lighter to process and quicker to generate.

7.4.1 Offline Modeling

The dataset from the RIPE/RIS database for the offline modeling is prepared as follows:

- We use five of the RIS collectors, RRC01, RRC04-07. The collectors dump BGP message at five-minute intervals. Consider this as one time epoch.
- We form the dataset over a period of one month, from September 8th, 2017 to October 8th 2017. We pick five epochs every day, from each collector.

- We use the BGPStream [108] site to look for potential BGP hijacks that have occurred in the one-month time-frame. Subsequently, we collect the message dumps for time-epochs identified with potential prefix hijacks. This forms our test dataset that we run against our trained model.

7.4.2 Modeling Based on Live BGP Streams

For the online model, we pick a random time from April 8th 2018 and go back one year. We then consider a 30-minute window starting from this random timestamp and collect announcements from a RIS collector picked at random from RRC01, RRC04, RRC05, RRC06 and RRC07. This process is repeated 360 times, and we end up with 10800 hours worth of BGP updates. The model is trained as the updates are live-streamed using PyBGPStream, a Python bindings package to BGPStream [109].

7.5 Detection Methodology

In this section, we elaborate on how to detect Type-0, Type-1 and sub-prefix hijacks. Borrowing from [124], the percentage of invisible, higher-order hijacking events tend to pollute smaller sections of the Internet. Subsequently, here we deal with only Type-0, Type-1 and sub-prefix hijacking of both types.

7.5.1 Overview

Section 7.3.2 elaborates on the constructs used for our analysis. Here, we illustrate how to actually detect hijacks based on a model that is combination of an offline and online approach. We use collocations, or pairs/triplets/quadruplets of words that occur more frequently than expected judged on the frequency of their individual words. Collocations are a whole beyond the sum of the parts. Finding collocations primarily requires the frequencies of words and their occurrence in the context of other words. They, more often than not, need to be filtered to get rid of juxtapositions that do not make sense, thus only retaining useful content terms. We then score each sequence (we use sequence here to establish the notion of a collocation) of words with an association measure, to determine the relative likelihood of each sequence being a collocation. Our approach utilizes a combination because of some limiting factors of streaming updates. When we stream information, each message is independent of the other and bigrams end up being simple sequences of words. As a result, the semantic understanding provided by their appearance in a corpus is lost. To correct for this phenomenon, we first train the model on a corpus of BGP updates downloaded from the RIPE-RIS [118] database, then we further train the model on live streams to keep it up-to-date.

Collocations, frequencies and their probabilities work well if given a sizeable corpora to work with. With BGP data, this is not a concern since there is an inordinately large amount of data available to work with. Also, any analysis of a natural language involves pre-processing to prepare the corpora. In the case of NLP, it would be normalizing the text, converting everything to lower-case, removing punctuation and stop-words etc. Our pre-processing involves normalizing fields found in the messages etc, tokenization etc. It is the network equivalent of preparing a natural language corpus for analysis.

7.5.2 Detecting Type-0/1 Hijacks

Origin-AS or Type-0 hijacks are by far the simplest ones to observe. These are route hijacks in which an AS announces its ownership of a prefix, that is neither its own, nor does it have authorization to originate an announcement. Suppose *AS1 - 192.168.0.0/23* is a valid mapping, if another AS, say *AS2* suddenly advertises *192.168.0.0/23*, that constitutes a Type-0 hijack.

Understanding the data to be analyzed is critical. The key to detecting a Type-0 route hijack is the prefix and its originating AS, that is the last entry of the ASPATH. Thus, we look for collocations of size 2, or bigrams, where either the left value is a prefix accompanied by a string consisting of only numerals, or vice versa. Other bigrams can be discarded. In BGP MRTs, the prefix appears first, followed by the ASPATH. If the order were to be swapped, it would be trivial to just find bigrams over these two entities, which would definitely result in one of the bigrams being an advertisement. Since our data is the positioned in the opposite order, we can do one of two things: (i) swap them around and find bigrams, or (ii) adjust the window size for finding collocations.

The latter approach uses a look-ahead parameter to form bigrams. Since the prefix is followed by the ASPATH, and self-prepending is prevalent, we process the ASPATHs to remove AS-prepending and adjust the window size accordingly to find bigrams. The frequencies of occurrence of the bigram itself, in conjunction with the frequency of each component of the bigram occurring with the other, dictate the level of correlation between the components. This information, analyzed over a sizeable corpus, will most definitely establish a certain confidence level for prefix advertisements that normally occur together.

It would be fair to question the notions of a bigram, as it appears to be a tuple of two components, but subtleties lie in the interpretation. Tuples are just two words put together with no real relationships, but an n-gram (and thus a bigram) establishes a semantic relationship between the two components themselves put in perspective of the entire corpus. Detecting Type-1 route hijacks follows a similar line of reasoning, except that we look for collocations of order 3, rather than 2. Again, in this case, we end up with substantially more collocations and these have to be filtered. For example, our *test corpus* of a 140-odd documents results in $\approx$ 22 million collocations. When we filter these to take into account only announcements, the number drops to $\approx$ 3 million.

7.5.3 Sub-Prefix Hijacks and Potential Misconfiguration

One of the concerns of using NLP approaches for BGP anomaly detection is its ability to detect sub-prefix hijacks. In this section, we illustrate that it is definitely possible to identify sub-prefixes using language processing techniques.

To identify sub-prefixes with natural language methods, we have to consider them as strings of characters and not IP addresses. The relationship between a sub-prefix and

Table 7.2: RiB Prefix Corpus Sizes.

Collector	RiB Prefix Count
RRC00	771388
RRC01	771133
RRC04	750499
RRC05	737207
RRC06	756660

its super-prefix (or at many pairs for that matter) is not very different when they are treated as strings and not IP addresses. For example, *64.106.0.0/17* is the super-prefix of *64.106.64.0/18*. The difference between both strings is a single character transformation. It is possible to transmute one string into the other by either deleting one character and adding the other or replacing the character in question. This concept of string similarity is termed as word-distance and is defined by the number of operations it takes to transform a string into another. Possible operations are additions, deletions, replacements and transpositions. There are various fuzzy string matching techniques that use different measures of distance. One of the well-known ones is the "Levenshtein distance" that considers only additions, deletions and substitutions.

We utilize this metric to list similar prefix strings extracted from an exhaustive list of prefixes observed during the training phase. This "prefix-corpus" can also be built using the CAIDA AS-relationship dataset [1] listing all advertised prefixes. It is then trivial to eliminate the candidates from this list that do not share a network space overlap with the sub-prefix. One might be led to question the advantage of this approach, because trie-based subnet lookups are faster and it is just a reverse-trie traversal to obtain all the super-nets for a certain prefix. To explain this, we look to Mahajan et al's study of BGP misconfiguration [88]. The study shows that a lot of short-lived routes are indeed misconfiguration. In this eventuality, it might not exactly be a sub-prefix, but one that appears to be lexically similar. A list of such strings helps us narrow down any similar looking prefixes (if there are no super-prefixes present). We can subsequently estimate the probability of this prefix belonging to any of the ASes that own prefixes on the candidate list. This is an advantage that trie-based approaches cannot provide.

7.5.4 A Note on Type-N, N≥2 Hijacks

We have not dealt with Type-N hijacks as part of this work, mainly due to the difficulty in finding a sufficient number of Type-N hijacks to test the model with (in a simulated scenario). Since our primary focus is to make a case for developing a network processing language framework and BGP data analysis only serves as the means to that end, we reserve this for future investigation. That being said, it is not very difficult to cast the Type-N hijack problem as a collocation finding problem.

One approach to accomplish this would be to vary the window size for collocation finding until a best-fit is found for accuracy. [2] provides a good starting point to experiment with this approach. The window size can be set to accommodate the average ASPATH length on the Internet and varied around it to observe accuracy variations.

7.6 Evaluation Methods and Experimental Results

We now analyze the results of our experiments with respect to the model's accuracy in predicting outcomes. Doing this allows us to evaluate whether the model was able to successfully utilize the semantics of collocation to arrive at the right result. To evaluate the collocation-based classification, receiver operating characteristic (ROC) curves are plotted and the area under the ROCs for the Type-0 and Type-1 hijacks calculated. The ROC curves give us a good estimate of how the classifier is working.

The sub-prefix matching evaluation is carried on the dataset constructed in Section 7.4.1. A prefix corpus is also constructed from the dataset, that consists of all the unique prefixes contained in it. First, we pick 100 prefixes at random from the prefix corpus. A subnet is generated for each prefix (provided that the subnet itself is not present in the corpus). It is then matched against all prefixes in the corpus to find strings that are most similar to the prefix, using the Levenshtein distance metric. If the super-prefix picked earlier is part of the candidate list, it is counted as a successful sub-prefix identification. The hijack analysis then proceeds as described already for regular Type-0 and Type-1 hijacks with evaluation being performed for the sub-prefix:ASN mapping.

The test dataset contains ≈ 2.9 million genuine prefix announcements and 45000 route hijack announcements.

These tests are run for RiB prefixes from five collectors, all of them being approximately the same size (to keep the evaluation consistent). The precision of sub-prefix matching and the execution time are compared to ensure that this is not a bottleneck. Table 7.2 shows the number of unique prefixes present in the respective collector's dataset.

7.6.1 ASPath Change Detection

We use one experiment from Blazakis et al. [20] to demonstrate that we can successfully utilize the concept of collocations to extract useful information from network data. The authors in [20] describe an ASPATH edit distance metric (AED), based on the concept of Levenshtein distance to track changes to the ASPATH. Using a Levenshtein distance-based metric normally means making code changes to account for the ASPATH being treated as a special case of a string. The same results can be achieved with what we term the bigram edit distance, or BED. A bigram edit distance metric can accomplish the same without the need for any external code changes or modifications to the Levenshtein distance algorithm.

In one of the experiments, the authors try to characterize how the AED behaves on a global scale. For this example, they consider BGP routing data from the week starting December 21, 2004 through December 28, 2004. The authors reason that a BGP route-leak during this period should show up definitely on the AED characterization. Figures 7.1 and 7.2 illustrate the cumulative distribution of the BED over the specified time period. The route leak incident is clearly reflected in the BED trace for December 24, 2004, very similar to the results that Blazakis et al. demonstrate in their research.

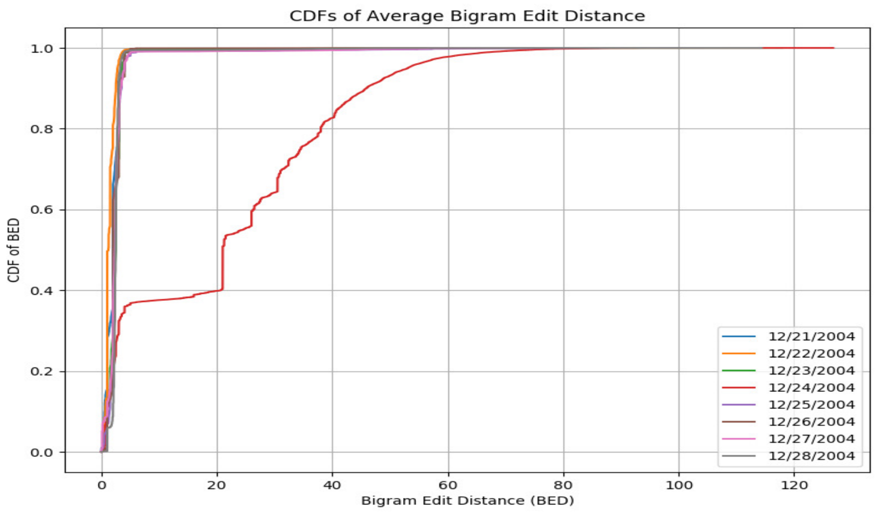


Figure 7.1: CDF of Bigram Edit Distance.

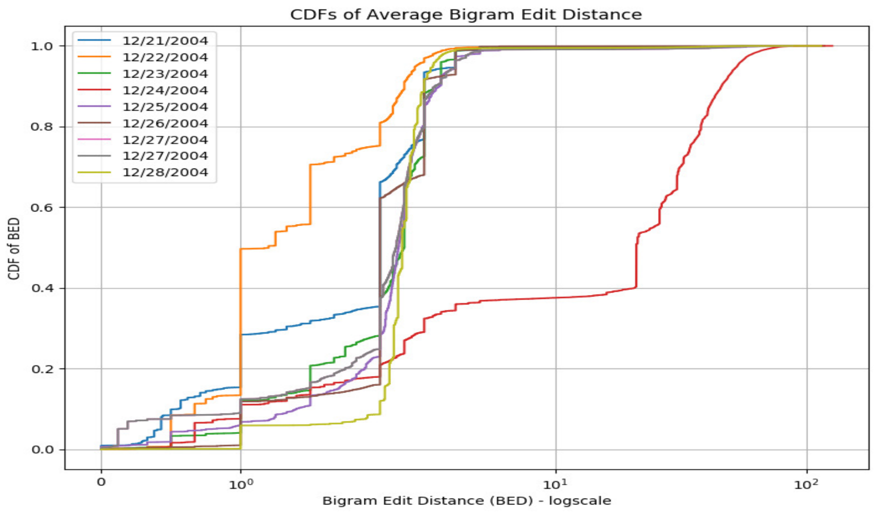


Figure 7.2: CDF of Bigram Edit Distance (logscale).

7.6.2 Hijack Detection Results

Figures 7.3 and 7.4 are the ROC curves for the Type-0 and Type-1 route hijacks respectively. Since we do not know what is a good threshold to use for classification, we make use of the ROC curve to plot the specificity against sensitivity. Since the diagonal of an

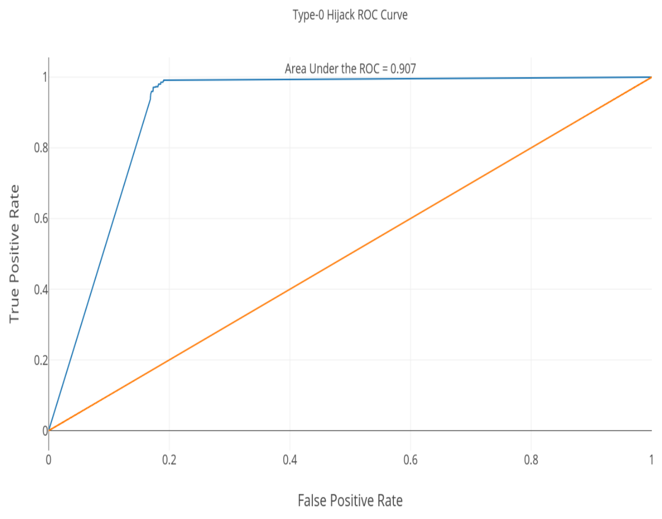


Figure 7.3: Type-0 ROC Curve.

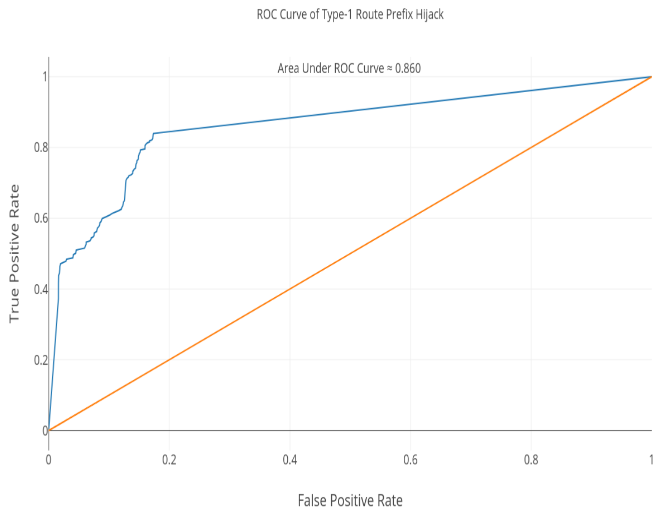


Figure 7.4: Type-1 ROC Curve.

ROC curve describes a uniformly random classifier such as a coin flip, classifiers should minimally perform better than this. The extent to which they score higher than this dictates how much better they are performing, since the area under the ROC, or AUROC is greater. In both our cases, the true positive counts are high and false positive counts are minimal, which is why we see the curve being pulled towards the ideal (0, 1) corner of the graph.

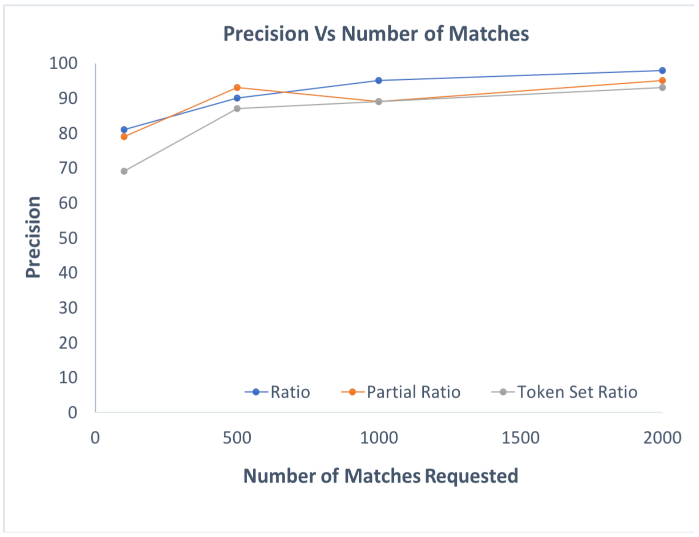


Figure 7.5: Sub-Prefix Matching Precision.

The other avenue where the ROC helps us is to choose a good threshold value which minimizes the false positive rate and maximizes the true positives. In the case of Type-0 hijacks, a good threshold appears to be 0.98, which is very close to 1. For the Type-1 ROC curve, it is around 0.85. The false positive rate is ≈ 0.18 in both cases. We believe that using a maximum likelihood estimate (MLE, that is based on raw frequencies of occurrence), coupled with the huge difference in classes themselves (2.9 million to 45K), results in some lesser seen announcements to be classified as hijacks. A few ways to fix this would be to use a probability estimate other than MLE, such a Kneser-Ney distribution, that allocates certain margins for unknowns. Increasing the number of collectors that the model is getting information from is also a potential solution, followed by developing a weighted method to score the likelihood of appearance.

7.6.3 Sub-prefix Matching Results

Sub-prefix matching was performed as explained in the experimental methodology. Figures 7.5 and 7.6 plot performance of the matching algorithms on a single collector's RiB prefix corpus. We evaluate three different metrics, ratio, partial ratio and token set ratio. Ratio works purely on the basis of Levenshtein distance-based matching. Partial

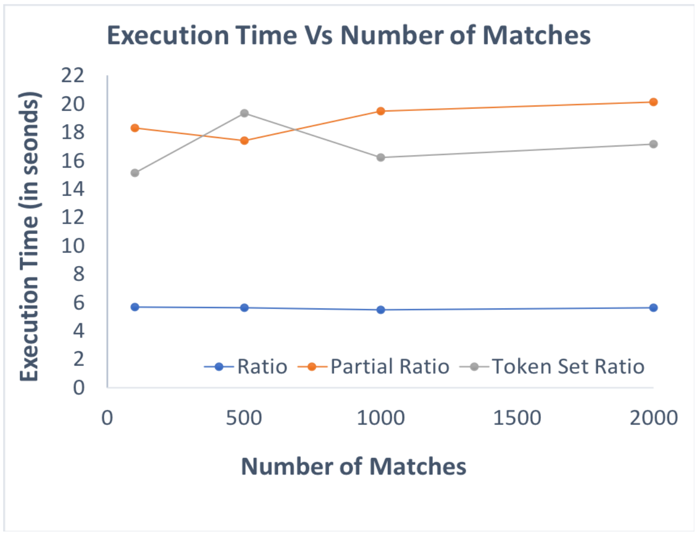


Figure 7.6: Sub-Prefix Matching Runtime.

ratio evaluates potential matches based on best sub-string matches and token set ratio approach tokenizes the strings and compares their intersection with the remainder. It is clear from these two figures that pure Levenshtein distance-based matching is extremely fast and maintains a consistent level of performance, invariant with an increase in the number of requested matches. It takes about 5 seconds to return anywhere from 100-2000 matches in the candidate list. The Levenshtein distance-based metric also plateaus at 1000 requested matches and the trade-off may not be worth the extra computational effort. A 98% accuracy on prefix-matching appears to be a acceptable solution.

Since we pick the Levenshtein distance based metric for our string matching, we analyze its performance across RiBs from different RIPE-RIS collectors. Figures 7.7 and 7.8 illustrate its performance. Again, it is evident that the performance is fast and consistent across a variety of scenarios, at the same time maintaining a high degree of accuracy.

7.7 Conclusions

Here, we made the case for a network processing language framework. It is our belief, based on prior research, ongoing work in the area, and current problems being faced in

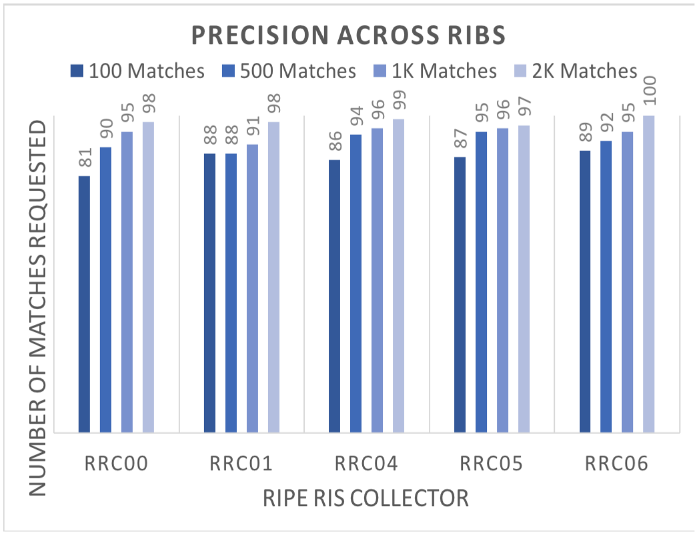


Figure 7.7: Fuzzy Matching Precision.

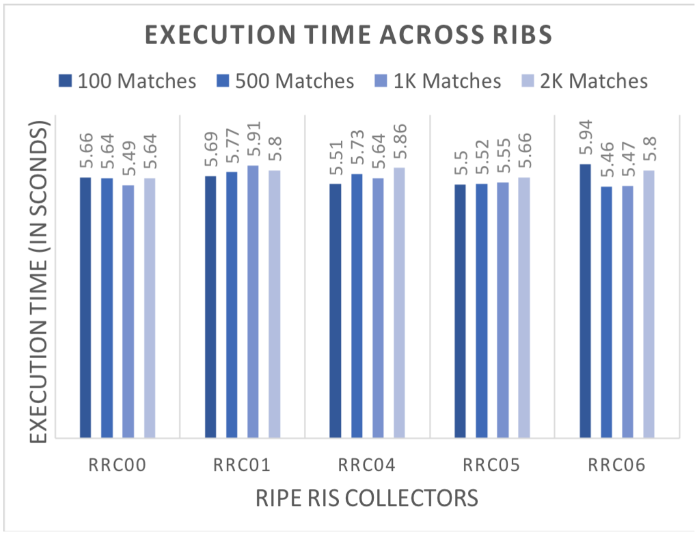


Figure 7.8: Fuzzy Matching Performance.

network management, that network management needs a novel approach of analyzing and correlating huge amounts of information. We hypothesized that network state inherently possesses semantics and structure, similar to a natural language that can be harnessed and utilized to make decisions. This hypothesis was then tested using publicly available BGP routing information to identify route prefix hijacks by analyzing the semantics of their occurrences.

Based on our results, we can confirm that our approach utilized network state semantics to *accurately* classify and predict route hijacks. These results lay a good foundation for further investigative research into developing a network processing language framework that can aid and improve network management techniques.

CHAPTER 8

TOWARDS AUTOMATED INTERPRETATION OF NETWORK STATE

So far, in Chapters 6 and 7, we discussed the methodologies and approaches that are central to the design of our network alerter framework. In this chapter, we will deal with the design, development and integration of these methodologies into components that can ultimately be integrated into a framework that can identify events and extract information from them.

8.1 System Architecture

The architecture of the framework is illustrated in Figure 8.3. The components can be delineated into three different categories on a temporal scale, from left to right, as indicated.

Various information and data sources are monitored for network state, such as routing information bases, connection tables, log files etc. When any change of network state is detected, in any of the sources, a variant of our model (proposed in Chapter 5) will version that change into an appropriate state repository. Activity recognition models that are developed specifically to extract the most important information from network state then process the state repository at pre-defined intervals of time, building models that are state-information specific. This phase of the process put together constitutes "event identification". Every event on the network will have a unique signature based on the different pieces of information that it depends on. For example, just looking at a $(src_ip, dst_ip, src_port, dst_port, protocol)$ tuple tells us this is one node connection to another node. We can build an initial set of events based on these activity patterns.

The second phase involves assessment of the event. This phase determines “what” the event is triggering and “how” is it going to affect the system. This in itself is a two-step process. When the changes from data sources run through their respective models, they are assessed against an established set of patterns in the network. The models will tag their output with these potential flags. The output from the models is then sent into the rule repository. This is a store for well-known rules and actions that have been programmed and also have been learned from the actions of network administrators. Depending on the rule specifications, and the degree of confidence the system has about the rule, either corrective action is taken, or an alert is sent upward to the network administrator.

8.2 Introduction and Related Work

As modern computer and communication systems scale in size to meet emerging consumer demands, they continue to push the boundaries of technology. With the increase in complexity, problems associated with these systems also rise, brought about by bugs and unexpected operations. Consequently, legacy methods for system management and troubleshooting become increasingly cumbersome and involve a substantial amount of manual work in debugging the problem.

Machine learning (ML) offers potential solutions to a wide variety of these problems. With complex systems and hyper-scale networks, data is abundant and developing new methods that incorporate machine learning appears to be the logical path to take. Since a large amount of the available data is in the form of textual log data, Natural Language Processing (NLP) and Information Extraction (IE) techniques can be utilized to understand the contents of this data.

The authors of DeepLog [40] demonstrate this research direction by performing anomaly detection on log files using a deep learning-based method. We propose an approach that is similar to DeepLog, but with a different methodology and goals. Our work focuses on building a robust event identification framework that can automatically extract events, their extents, and relevant attributes. This information can then be utilized in different ways, with anomaly detection being one of them.

Computer networking has recently moved significantly towards programmatic networking. Programmability provides administrators with much needed flexibility to manage their networks, but comes at a cost of operational and management inefficiency. As these networks dynamically scale, a small problem can potentially have magnified effects. Conversely, tracking down this small problem in a large deployment can be difficult and time-consuming. Combining these challenges with human factors can result in high troubleshooting turnaround times and significant management workloads.

We posit that an event in a log file is ultimately constructed by a human programmer to convey some information about system state. Therefore, the event log follows the constructs of natural language, albeit semi-structured in nature. Inspired by traditional event extraction schema, such as Automatic Content Extraction (ACE) [83] and Entity Relation Events (ERE) [84], we create an event ontology, aiming to be as comprehensive as possible. We then use a rule-based event extraction framework based on this ontology to define the salient event types that should be extracted from the logs.

In IE, event extraction is the task of identifying and classifying event trigger words and their arguments in text. A prodigious amount of work has been done on event extraction for unstructured data [62, 63, 69, 79, 80, 104]. Many of these efforts rely upon manually created schema or ontologies [83, 84] to define the events of interest. However, existing ontologies are domain-specific and unable to generalize to a network management setting. Our ontology is designed for network management and captures the underlying semantics of network events that can be applied across networks for analytics.

Approaches to applying NLP techniques to network logs typically are difficult to transfer to other applications and rely upon either the discovery of the log statements in the source code [40] or clusters of potentially unstable word embeddings [142]. In contrast, our approach has the potential to be used for different applications without a change in the underlying framework as it does not need a priori information about the log file structure.

The remainder of the chapter describes the event identification framework in Section 8.3, use cases in Section 8.4, the methodology in Section 8.5, and an evaluation in Section 8.6.

8.3 Event Identification Framework

An event is an occurrence of something specific that happens involving certain actors. The event invariably results in a change of state. Since these occurrences and changes of state are considered to be important, log files record them assiduously to help with troubleshooting and root cause analysis in case of an untoward occurrence. In this section, we describe the event identification framework and ontology.

8.3.1 Event Ontology

There are two important pieces of information in a log message that relate to event identification: the event *extent* or event *scope* and the event *trigger*. The event extent is a sentence that contains a taggable event and the trigger is the word or words that most clearly explain the reason for the event's occurrence. In addition to the extent and the trigger, we need to identify the entities involved in each event. These entities constitute the event participants. Many events contain infrequent occurrences of certain entities, that are part of the event, but not primary participants. These entities constitute event attributes and act to further qualify the event. Attributes and participants are collectively called event arguments.

To perform this identification as accurately as possible, we need a well-defined event ontology that describes the different classes of events and their sub-types and arguments. We define a potential event identification ontology as depicted in Figure 8.1. This ontology

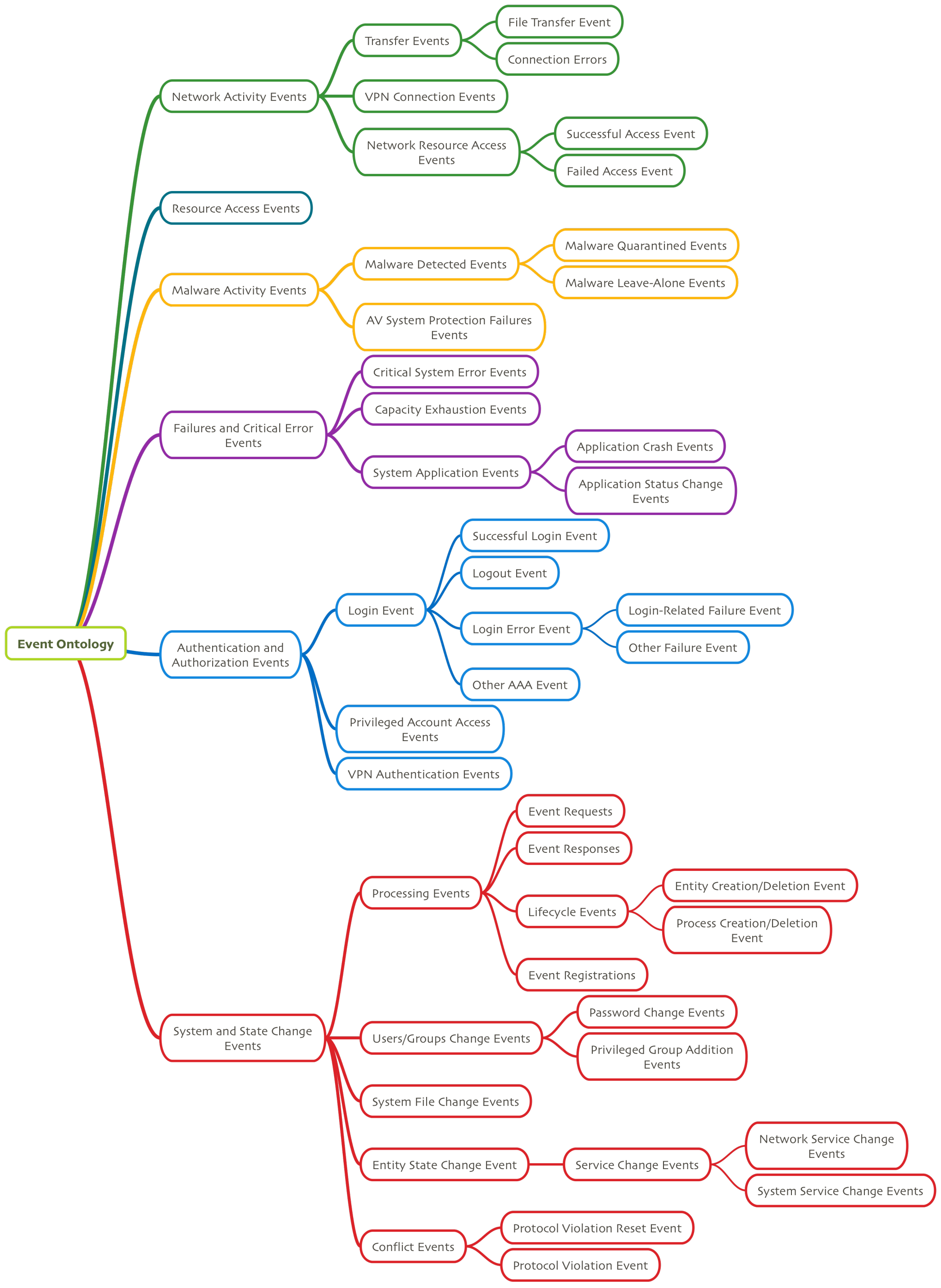

Event Ontology
Network Activity Events
Transfer Events
File Transfer Event
Connection Errors
VPN Connection Events
Network Resource Access Events
Successful Access Event
Failed Access Event
Resource Access Events
Malware Activity Events
Malware Detected Events
Malware Quarantined Events
Malware Leave-Alone Events
AV System Protection Failures Events
Failures and Critical Error Events
Critical System Error Events
Capacity Exhaustion Events
System Application Events
Application Crash Events
Application Status Change Events
Authentication and Authorization Events
Login Event
Successful Login Event
Logout Event
Login Error Event
Login-Related Failure Event
Other Failure Event
Other AAA Event
Privileged Account Access Events
VPN Authentication Events
System and State Change Events
Processing Events
Event Requests
Event Responses
Lifecycle Events
Entity Creation/Deletion Event
Process Creation/Deletion Event
Event Registrations
Users/Groups Change Events
Password Change Events
Privileged Group Addition Events
System File Change Events
Entity State Change Event
Service Change Events
Network Service Change Events
System Service Change Events
Conflict Events
Protocol Violation Reset Event
Protocol Violation Event

Figure 8.1: Ontology for Event Identification.

is a combination of classifications from sources such as [121], the AIX Audit Events guide, and various other manuals.

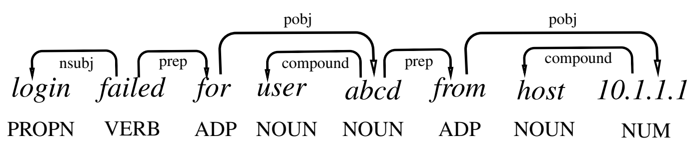


Figure 8.2: Log Dependency Parse Tree Visualization.

8.3.2 Event Identification

Identifying events can be accomplished in one of two ways: selecting event triggers and arguments from the log message, and then classifying them as an event type in the ontology, or two, by building event identification into the NLP pipeline. We illustrate how the latter can be accomplished using sentence similarity models and text categorization models, both built using convolutional neural networks (CNNs).

Event logs have well-defined scopes and thus event extents are simple to identify. Triggers can be determined by performing part-of-speech tagging on the entry with dependency parsing [94]. This is illustrated in Figure 8.2. This process helps to identify event triggers as described in section 8.3.3.2.

The syntactic relations in the dependency parse form a tree and every word has exactly one head. We can process the arcs in the tree by iterating over the words in the sentence. Arcs of interest can then be processed to tag event arguments with their appropriate entity types, which the schema dictates. In Figure 8.2, we may start from the root "*failed*" and crawl prepositional (`prep`) or primary object (`pobj`) edges, taking nodes labeled as nouns or numbers as argument candidates. We then assign an entity type from our ontology to each candidate argument.

We finally match each candidate argument to its correct argument role. A canonical representation of the argument roles in our example would be "*Login failed for user* `AGENT-ARG` *from host* `DEVICE-ARG`". Of the various entity types available, this `LOGIN RELATED FAILURE EVENT` from our ontology could contain entity types USER and

Table 8.1: Sample event schema.

Event Arg	Entity Type	Entry Match
`AGENT-ARG`	USER	Login failed for user **abcd** from host 10.1.1.1
`DEVICE-ARG`	HOSTNAME IPADDRESS ACCOUNT	Login failed for user abcd from host **10.1.1.1**

DEVICE. The schema would then map "*abcd*" to `AGENT-ARG` and "*10.1.1.1*" to `DEVICE-ARG`, as illustrated in Table 8.1.

For the purposes of this discussion, we perform event identification using sentence similarity and text categorization techniques. The text categorization models are trained and evaluated using Prodigy [45] and are subsequently used in the sPacy [46] NLP pipeline for extended analysis. We develop the sentence similarity models using Keras running on a Tensorflow backend.

8.3.3 Automatic Content Extraction

Automatic content extraction (ACE) [83], is a set of guidelines laid out by the Linguistic Data Consortium, for annotating and identifying entities, relations, and events. These guidelines facilitate accurate and precise event extraction. This section explains how the ACE schema relates to our problem and what parts are adapted to develop our framework.

8.3.3.1 Event Extent

The semi-structured nature of log files is advantageous: every event being a self-contained message in itself almost always makes identifying the event extent facile, and the event boundaries are distinct. The crux of the problem lies in identifying event triggers accurately and mapping its corresponding event arguments.

8.3.3.2 Event Triggers

An event trigger is the word (or words) that most clearly explain the scope of the event. In many cases, the main verb of the log will sufficiently describe the event. Consider the two following examples: "*Login failed for user ABC from host xyz*" and "*Protocol test:proc is violated at location for 604 times, started at 2017-10-04 20:20:38 EDT*".

In both cases, the main verbs, namely "*failed*" and "*violated*" adequately describe what the event is about. If we want to be more specific, "*login failed*" can be used the event trigger to further set it apart from successful logins, for instance. As part of our identification framework, we list potential trigger identification semantics below:

- Verb: The main verb suffices in most cases.
- Verb + X + Adjective: In slightly more complex cases, event triggers are identified by a combination of words and occur in attempts to adequately describe certain consequences of an event.
- Multiple Verbs: These occur, but are rare, due to semantics not being a priority in event description.

8.3.3.3 Event Properties

Event types and sub-types are not the only constituents of an event description. There are many properties that lend meaning to an event. The *polarity*, *tense*, *genericity* and *coreference* are some examples of such properties. A discussion about these fall beyond the scope of this chapter (and this dissertation, since that is a subject domain by itself) since much of the work here is focused on event identification.

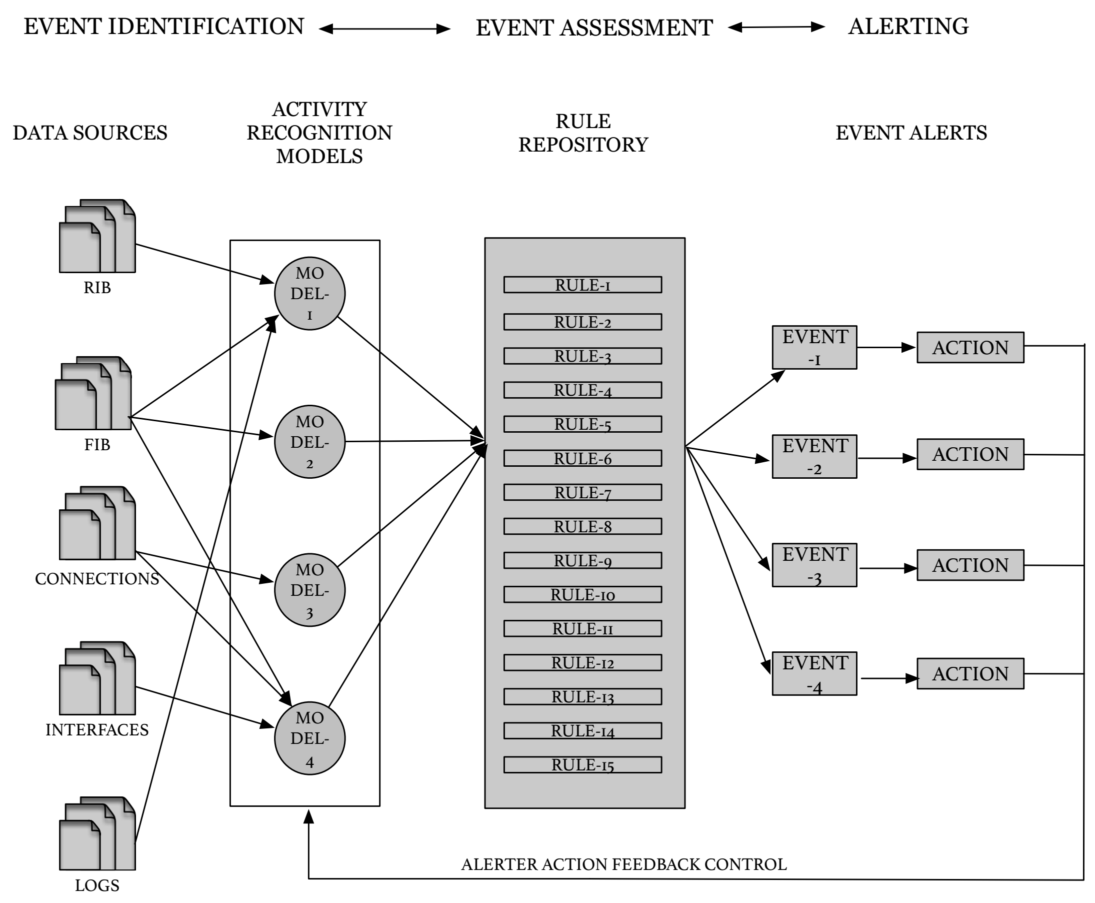


Figure 8.3: Event Identification Framework.

8.4 Use Case Scenarios

8.4.1 Anomaly Detection

The reference architecture of our intelligent alerting system is illustrated in Figure 8.3. The components can be delineated into three categories on a temporal scale of occurrence, from left to right, as indicated.

Event identification models can be built based on the information stored in the various log sources depicted in Figure 8.3. Event identification algorithms may then process the incoming logs in a streaming fashion against the models, generating fundamental event identifiers. For example, one algorithm could process the interface logs for changes. This phase of the process constitutes "event identification". Every event on the network will have a unique signature based on the different pieces of information that it depends on.

Identifying these components as event arguments makes it easier to correlate information across the board.

The second phase involves assessing the event. This phase determines “what” the event is triggering and “how” is it going to affect the system. When the changes from data sources are run through their respective models, they are assessed against an established set of patterns. If there are any deviations noticed, or anomalies detected, the models will tag their output with these potential flags. These output from the models are then sent into the rule repository, with the repository being a data-store for well-known rules and actions. Many common rules and actions can be programmed a priori into the repository. The data-store can also contain rules and actions learned from network administrators.

Based on the rule specifications, the degree of confidence the model has about its output matching a certain rule, and the availability/confidence in the appropriate actions to be taken, the system decides whether to employ a proactive, corrective action or send an alert to the administrator.

8.4.2 Extractive Summarization

A good illustration of an NLP application would be that of text summarization. Summarization is the process of automatically generating a summary of the text being processed. Consider a scenario in which a network administrator has identified the occurrence of a fault and wants to perform root cause analysis. The administrator then manually sifts through the logs for that time period, looking for clues that might throw some light upon the network fault. Our framework can potentially extract all the log entries for each device for the specified time period, and create an extractive summarization of the key events occurring on a per-device basis. These summaries can then be compared by the administrator to gain baseline knowledge of what is happening across the network. Now, this certainly is not sufficient to analyze the problem entirely, but we argue that it gives the administrator a good handle on the situation of what is involved and reduces manual effort.

8.4.3 Knowledge Graphs and QA Systems

This use case is valuable in aiding administrators in troubleshooting their systems. A knowledge graph (KG) is a graph form that captures entities and events (nodes) as well as their attributes and corresponding relationships (edges). KGs are used in many different domains and for a variety of purposes, ranging from improving searches [129] to fighting human trafficking [135]. One particularly important usage of KGs is in Question Answering (QA), where automated systems are able to use this powerful and structured knowledge to answer human-created natural language questions [148, 150]. Since a KG can represent complex semantic relationships between entities and events on the network, administrators have the ability to execute complex natural language queries, such as "*What users failed to login on 10.1.1.1*", that may reduce the dependency on filter-based reporting. Predictive analytics over the KG can be performed, similar to how patterns in heterogeneous graphs can be used for link prediction [133, 134]. This helps understand network entity and/or event interactions as well as learn patterns that correspond to important higher-level events, such as man-in-the-middle attacks.

8.5 Methodology

We limit the scope of this research to the task of identifying events successfully. The successive sections deal with accomplishing this task.

8.5.1 Using CNN-based Text Classification

The event identification problem can be considered a special case of text classification. The eventual goal of event identification is to attach a specific label to a set of words and entities, thus signifying that it denotes the occurrence of a particular event. Tagging events is not very different from attaching labels to sentences, where the label immediately relates the text to a topic. We refer to event identification from a systems management perspective of attaching a label from the ontology to a state change. It does not refer to event identification in NLP, that deals with identifying real-world events occurring in natural language sequences.

CNNs are used typically for sentence classification, that is grouping a sentence into a set of pre-determined categories by considering n-grams, or a sequence of words in a window. Using this, events are assigned to categories, such as the ones defined in our event ontology in Figure 8.1.

To understand the process and how a CNN works, consider a sequence of words $w_{1:n} = w_1, \ldots, w_n$, where each word is associated with an embedding vector of dimension d. A 1-dimensional convolution of width-k is the act of sliding a window of size k over the sentence (typically viewed as n-grams). The same convolution filter or kernel, a dot-product between the concatenation of embedding vectors in the window and a weight vector u, is applied to each window in the sequence. A non-linear activation function g is finally applied to the entire convolution.

If we look at a window of words $w_i, \ldots, w_{i+k}$ the concatenated vector of the ith window is then: $x_i = [w_i, w_{i+1}, \ldots, w_{i+k}] \in R^{k \times d}$. The convolution filter is applied to each window, resulting in scalar values r_i, each for the ith window: $r_i = g(x_i \cdot u) \in R$. In practice, more filters are applied, $u_1, \ldots, u_l$, which are then represented as a vector multiplied by a matrix U, with a bias b: $r_i = g(x_i \cdot U + b)$, where $r_i \in R^l, x_i \in R^{k \times d}, U \in R^{k \cdot d \times l}, b \in R^l$.

The vectors resulting from different convolution windows are subsequently combined into a single l-dimensional vector using a pooling operation. The maximum or average value observed in resulting convolution vectors are considered. Ideally, this vector should capture the most relevant features of the sentence. This vector is then fed forward into the network, most probably to a fully connected layer to perform predictions [35].

As part of developing a proof-of-idea that text categorization approaches can be applied to do event identification (which can also be considered a form of categorization), we annotated, trained and evaluated models developed using Prodigy [45]. We do not go into detail about them because they are part of the annotation tool and thus are out of scope. Section 8.6 contains more details.

8.5.2 Using Sentence Similarity Metrics for Event Identification

Sentence similarity metrics and measurement approaches are used extensively to accomplish various NLP tasks, such as identifying duplicate questions on online forums, sentiment analysis from user reviews on various online sites, etc. The reader might wonder how or why identifying two sentences that are semantically and contextually similar is applicable to event identification, especially from a management perspective. How would the system be able to identify events that are similar? Foremost, building a comprehensive and representative dataset of all events across platforms is infeasible. For example, authentication failure logs invariably convey the same information but vary in format and semantics across systems. However, teaching the model the context and semantics of event types is definitely possible and also transferable to new event formats.

Here, we aim to illustrate that sentence similarity can be used to recognize events that are semantically and contextually relatable to each other. By using sentence distance metrics and feature engineering a supervised classification to establish a baseline; and by building a deep learning model, it can be demonstrated that certain properties in event logs can be utilized to attain a high level of precision and recall, especially in very unbalanced datasets.

Our methodology is conceptually derived from the one succinctly explained in [3], albeit adapted for event logs. This is an attempt to mimic real-world scenarios where the event types we are interested in tend to be outnumbered by other dissimilar types. There are a few properties of event logs that we can leverage to our advantage. First, they are largely repetitive in nature, and second, they exhibit low variance in content. These properties mean that extracting semantically useful patterns are much easier when compared to the same operations on unstructured text that occurs in natural languages.

Sentence similarity metrics are organized into feature sets and then divided into three fundamental groups. The first set consists of basic sentence features such as word and character counts, similar word counts, etc. The second set contains more advanced fuzzy features such as token set ratios, etc., and the final set comprises of sentence distance metrics such as cosine distance, Manhattan and Jaccard indices, word-mover distance etc. We also combine these feature sets to generate more coverage and allow for clustering.

To demonstrate that deep learning can utilize the properties of event logs described above and do much better than feature-engineered supervised classifiers, we build a model as depicted in Figure 8.4. The model is an integration of both LSTM and CNN representations of the log files that go through a series of dense and batch normalization layers. We experiment with different layer combinations and depth, maintaining the basic merged-model structure. We use GloVe (Global Vectors) embeddings [111], a popular technique for generating word representations, to initialize the LSTM and CNN representations. Specifically, we use pre-trained 300-dimension vectors. For the convolution layers, we use 64 filters, with filter length of 5. We also use a dropout rate of 0.2 and train the model over 200 epochs. We evaluate their performances in the next section.

8.6 Evaluation

The dataset is built from event logs gathered from four different sources, namely network routers, access points, switches and OpenSSH logs. It contains around five million log entries. The dataset is divided into a 70-30% train-test split. There are about 12-13% of logs that are similar (authentication failure, login-related failure, etc.). This percentage is maintained in the train-test split using stratification. The models are then trained and evaluated on the 70% and we then test the model on the 30% that has been held back. The supervised classifier is constructed using Python and scikit-learn and the deep-learning model is built using Keras on top of Tensorflow.

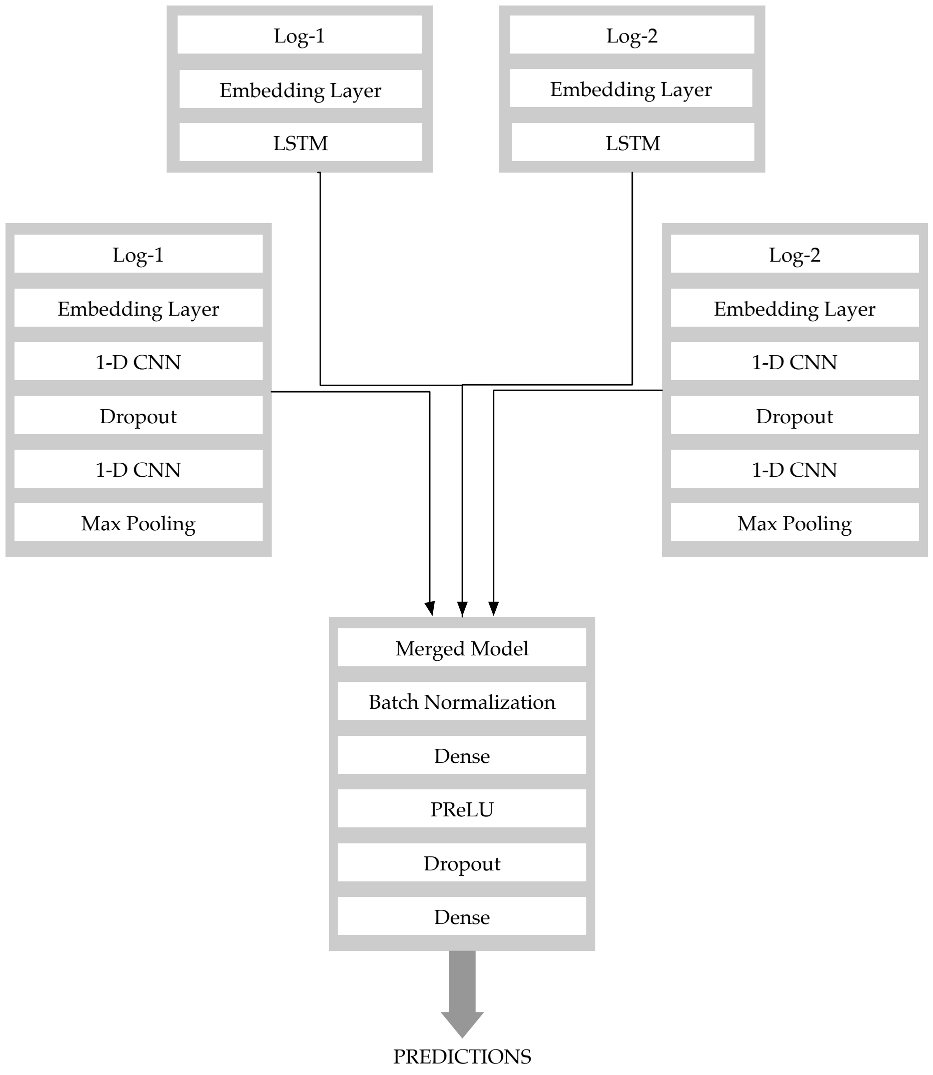
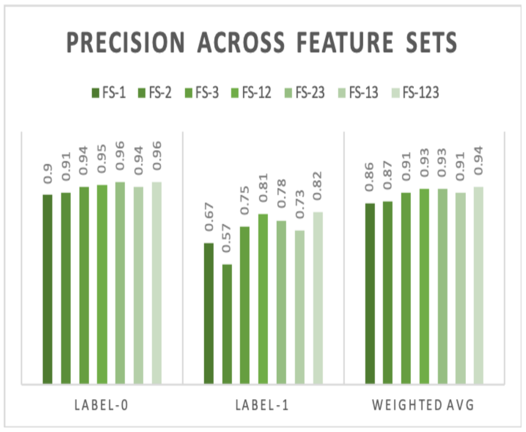
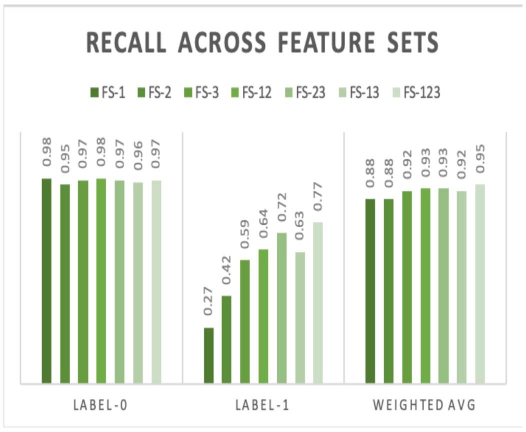


Figure 8.4: Sentence Similarity Model.

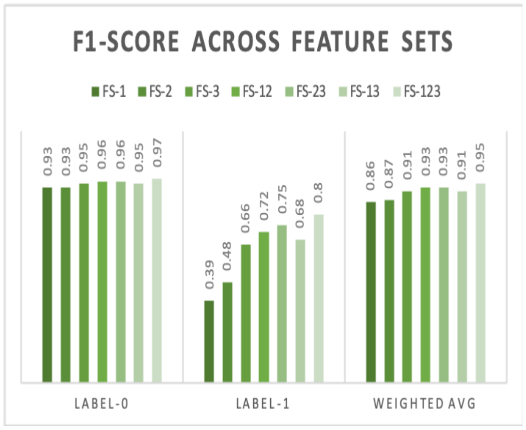
Prodigy [45] is a training and annotation tool that operates with the model in the loop and its models are CNN-based extensions of the work described in [132]. As features and categories are annotated, the model is continually updated in the background. We use this method to illustrate that a CNN-based text categorization approach can be used to identify events. We annotated around 10K logs with various authentication related labels, sourced from the same logs are described in the dataset creation section above. Because the model was in the loop, we used only 50% of the examples for training and the remaining for evaluation. This is again tested on an unbalanced dataset (which is not the same as the one



(a) Precision.



(b) Recall.



(c) F1-Score.

Figure 8.5: Performance Baseline for Supervised Classifier.

used for the next section). Due to scarcity of space, we just list the numbers inline here. The model attained 98% precision, recall, and F1-scores.

Figures8.5 and 8.6 illustrate our comparison of a baseline supervised classifier and the deep learning-based models. The 0 and 1 on the axes labels indicate whether logs were similar (1) or not (0). We consider the preicion, recall, F1-score and weighted average metrics to compare the performance. We use precision-recall features due to the properties of the dataset, as demonstrated in [120]. The behavior of the supervised classifier is in accordance with what we expect when the event logs are dissimilar because they form the majority portion in the dataset. Advanced fuzzy feature alone are not enough to gener-

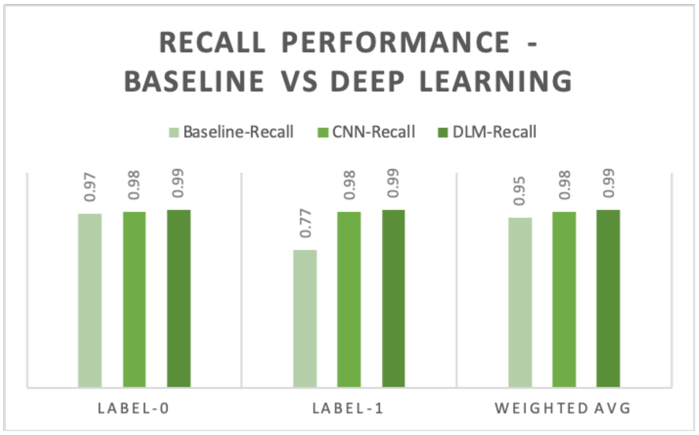


Figure 8.6: Recall Comparison Across Models.

ate enough clustering for confident predictions. It averages 48% when identifying similar logs, which is not The performance improves drastically when multiple feature sets are combined, climbing to 80%. This is an indication that extracting NLP-specific features are definitely beneficial.

From the numbers, it is quite clear that the deep-learning model is especially efficient at identifying similar events, even though they are scarce in the dataset, much like the CNN-based text categorization model. The deep learning model outperforms the baseline classifier comfortably, being able to extract more features through iterated convolutions than the feature engineered methods tried above. The high numbers can be attributed to a few reasons: repetitiveness, low variance and the effectiveness of CNNs in extracting language-specific patterns.

8.7 Conclusion

Based on the administration and troubleshooting requirements for contemporary systems, we proposed a robust event identification framework based on natural language processing concepts. We illustrated how log files and their associated entries are extensions of natural language, albeit semi-structured and can be processed accordingly. We then presented various scenarios where the event identification network can potentially be utilized to help administrators utilize their time more efficiently. From the results, it is clear that NLP-based approaches can help in processing large volumes of event logs, helping to identify and extract relevant information. These results lay the foundation for further research, such as zero-shot transfer learning, that can demonstrate the effectiveness of deep-learning approaches applied to systems/network management.

CHAPTER 9
SUMMARY

We started by hypothesizing that networks are becoming more dynamic in nature and we need novel methods and approaches to enhance network management. Subsequently, we presented four different methods to enhance the network management experience.

- *Adaptive Service-Chain Routing for NFVs*: We explored how a problem in the context of a dynamic, programmable networking environment that could be intractable was solvable through a network graph transformation, compared to designing complex algorithmic approaches. We demonstrated the feasibility of such a solution by prototyping it on an emulated network with DockerNet, ONOS and sFlow.
- *Multi-Criteria Routing in Networks with Path Choices*: Here, we presented ParetoBFS, a variant of a breadth-first search that uses branch-and-bound techniques to find all the Pareto-optimal paths while effectively limiting the potentially very large search space. We explored several sampling techniques to further improve search performance while limiting the degradation in quality of the results to only a marginal amount. Our simulation results show that existing multi-criteria combinatorial optimization approaches can only search a small fraction of all the Pareto-optimal paths while ParetoBFS can obtain the whole path set in shorter time. We also present results from an implementation of ParetoBFS on a software-defined network prototype.
- *Flow Revision Management for Software-Defined Networks*: We presented a design for a versioning system for software-defined networks. Our work addresses the problem of revision control for flow state management in SDN-enabled networks, so that

the underlying data plane might be able to provide better state protection, provenance, ease of programmability, and support for multiple applications. Inspired by the revision control tools in the software development world, we propose an abstraction and a system called GitFlow, which provides flow state versioning in the SDN context.

- *Automated Event Identification and Information Extraction from System Logs Using NLP*: With the increased pervasiveness of machine learning, this technology is capable of changing the way newer network management systems are built. We propose an event identification and management framework based on natural language processing concepts. Our system processes events in a log file as a natural language sequence and builds models of the extracted events to be used in various online or post-processing scenarios. We demonstrate how text categorization and sentence similarity concepts can be used to automatically identify events in logs. We also illustrate the advantages of our event extraction framework in different use cases and how our system helps to make network troubleshooting and management more efficient.

These approaches demonstrate how novel methods can be utilized to adapt to a rapidly evolving networking landscape, albeit one where network management approaches have failed to keep up. It is the author's sincere hope that these exploratory approaches contribute to expanding the body of knowledge in this area in a meaningful manner, however small it may be.

BIBLIOGRAPHY

[1] The CAIDA as-relationship dataset. `http://data.caida.org/datasets/as-relationships/`.

[2] The CAIDA aspath analysis. `https://labs.ripe.net/Members/mirjam/update-on-as-path-lengths-over-time`.

[3] Abhishek Thakur. *Is That a Duplicate Quora Question?*, 2019. `https://www.linkedin.com/pulse/duplicate-quora-question-abhishek-thakur/`.

[4] Advanced Network Technology Center, University of Oregon. *Route Views Project Page*, 2003. `http://www.routeviews.org/`.

[5] Al-Musawi, Bahaa, Branch, Philip, and Armitage, Grenville. BGP Anomaly Detection Techniques: A Survey. *IEEE Communications Surveys and Tutorials 19*, 1 (2017), 377–396.

[6] Al-Rousan, N. M., and Trajković, L. Machine learning models for classification of bgp anomalies. In *2012 IEEE 13th International Conference on High Performance Switching and Routing* (June 2012), pp. 103–108.

[7] Albert, Réka, and Barabási, Albert-László. Topology of Evolving Networks: Local Events and Universality. *Phys. Rev. Lett. 85* (Dec 2000), 5234–5237.

[8] Andersen, David G., Balakrishnan, Hari, Feamster, Nick, Koponen, Teemu, Moon, Daekyeong, and Shenker, Scott. Accountable internet protocol (AIP). In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication* (Seattle, WA, USA, 2008), SIGCOMM '08, ACM, pp. 339–350.

[9] Anderson, C J, Foster, N, Guha, A, SIGPLAN, JB Jeannin ACM, and 2014. NetKAT: Semantic foundations for networks. *dl.acm.org*.

[10] Ankur, Singla, and Bruno, Rijsman. Opencontrail project. Tech. rep., Juniper Networks, November 2004. `http://opencontrail.org/`.

[11] Ballani, Hitesh, Francis, Paul, and Zhang, Xinyang. A study of prefix hijacking and interception in the internet. *ACM SIGCOMM Computer Communication Review 37*, 4 (2007), 265.

[12] Barabasi, Albert-Laszlo, and Albert, Reka. Emergence of Scaling in Random Networks. *Science 286*, 5439 (1999), 509–512.

[13] Barber, C. Bradford, Dobkin, David P., and Huhdanpaa, Hannu. The Quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software 22*, 4 (1996), 469–483.

[14] Barrett, Chris, Bisset, Keith, Holzer, Martin, Konjevod, Goran, Marathe, Madhav, and Wagner, Dorothea. Engineering Label-Constrained Shortest-Path Algorithms. In *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management* (2008), AAIM '08, pp. 27–37.

[15] Bellman, Richard. On a routing problem. *Quarterly of Applied Mathematics 16*, 1 (Jan. 1958), 87–90.

[16] Berde, Pankaj, Gerola, Matteo, Hart, Jonathan, Higuchi, Yuta, Kobayashi, Masayoshi, Koide, Toshio, Lantz, Bob, O'Connor, Brian, Radoslavov, Pavlin, Snow, William, and Parulkar, Guru M. ONOS: towards an open, distributed SDN OS. *HotSDN* (2014), 1–6.

[17] Berman, Mark, Chase, Jeffrey S., Landweber, Lawrence, Nakao, Akihiro, Ott, Max, Raychaudhuri, Dipankar, Ricci, Robert, and Seskar, Ivan. Geni: A federated testbed for innovative network experiments. *Computer Networks 61*, 0 (2014), 5 – 23. Special issue on Future Internet Testbeds - Part I.

[18] Berman, Mark, Chase, Jeffrey S, Landweber, Lawrence, Nakao, Akihiro, Ott, Max, Raychaudhuri, Dipankar, Ricci, Robert, and Seskar, Ivan. GENI: A federated testbed for innovative network experiments. *Computer Networks* (2014).

[19] "Bhattacharjee, Samrat, Calvert, Kenneth L., and Zegura, Ellen W.". *High Performance Networking VII: IFIP TC6 Seventh International Conference on High Performance Networks (HPN '97), 28th April – 2nd May 1997, White Plains, New York, USA*. Springer US, Boston, MA, 1997, ch. An Architecture for Active Networking, pp. 265–279.

[20] Blazakis, D, Karir, M, and Baras, J S. Analyzing BGP ASPATH Behavior in the Internet. In *IEEE Global Internet 2006* (2006), University of Maryland, College Park and Merit Network Inc.

[21] Blei, David M, Ng, Andrew Y, and Jordan, Michael I. Latent Dirichlet Allocation. *Journal of Machine Learning Research* (2003).

[22] Bosshart, Pat, Daly, Dan, Gibb, Glen, Izzard, Martin, McKeown, Nick, Rexford, Jennifer, Schlesinger, Cole, Talayco, Dan, Vahdat, Amin, Varghese, George, and Walker, David. P4 - programming protocol-independent packet processors. *Computer Communication Review* (2014).

[23] Bu, T., and Towsley, D. On Distinguishing between Internet Power Law Topology Generators. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE* (2002), vol. 2, pp. 638–647 vol.2.

[24] Calico, Project. *Secure networking for the cloud native era.* Tigera Inc, 2016. `https://www.projectcalico.org`.

[25] Casado, Martín, Freedman, Michael J, Pettit, Justin, Luo, Jianying, McKeown, Nick, and Shenker, Scott. Ethane: taking control of the enterprise. In *SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications* (Aug. 2007), Stanford University, ACM.

[26] Casado, Martin, Freedman, Michael J., Pettit, Justin, Luo, Jianying, McKeown, Nick, and Shenker, Scott. Ethane: taking control of the enterprise. In *SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications* (Kyoto, Japan, Aug. 2007), pp. 1–12.

[27] Castro, Ignacio, Panda, Aurojit, Raghavan, Barath, Shenker, Scott, and Gorinsky, Sergey. Route Bazaar: Automatic Interdomain Contract Negotiation. In *15th Workshop on Hot Topics in Operating Systems (HotOS XV)* (Kartause Ittingen, Switzerland, May 2015), USENIX Association.

[28] Chen, Shigang, and Nahrstedt, K. On finding multi-constrained paths. In *Communications, 1998. ICC 98. Conference Record. 1998 IEEE International Conference on* (1998), IEEE, pp. 874–879.

[29] Chen, Shigang, and Nahrstedt, K. On Finding Multi-constrained Paths. In *Communications, 1998. ICC 98. Conference Record. 1998 IEEE International Conference on* (Jun 1998), vol. 2, pp. 874–879 vol.2.

[30] Chen, Xinming, Cai, Hao, and Wolf, Tilman. Multi-criteria routing in networks with path choices. In *Proc. of 23rd IEEE International Conference on Network Protocols (ICNP)* (San Francisco, CA, Nov. 2015).

[31] Chen, Xinming, Wolf, Tilman, Griffioen, Jim, Ascigil, Onur, Dutta, Rudra, Rouskas, George, Bhat, Shireesh, Baldin, Ilya, and Calvert, Ken. Design of a Protocol to Enable Economic Transactions for Network Services. In *Proceedings of 2015 IEEE International Conference on Communications (ICC)* (2015).

[32] Cheng, Kai-Wen, Chen, Yie-Tarng, and Fang, Wen-Hsien. Abnormal crowd behavior detection and localization using maximum sub-sequence search. *ARTEMIS@ACM Multimedia* (2013).

[33] Choi, Sumi Y., Turner, Jonathan S., and Wolf, Tilman. Configuring sessions in programmable networks. *Computer Networks 41*, 2 (Feb. 2003), 269–284.

[34] Committee, ONF Market Education. Software-defined networking: The new norm for networks. Tech. rep., Open Networking Foundation (ONF), 2012.

[35] David Batista. *Sentence Classification Using CNNs*, 2019. `http://www.davidsbatista.net/blog/2018/03/31/SentenceClassificationConvNets/`.

[36] Dhamdhere, Amogh, and Dovrolis, Constantine. Twelve years in the evolution of the internet ecosystem. *IEEE/ACM Trans. Netw. 19*, 5 (Oct. 2011), 1420–1433.

[37] Dijkstra, Edsger W. A note on two problems in connexion with graphs. *Numerische Mathematik 1* (Dec. 1959), 269–271.

[38] Doria, A, Salim, J H, Haas, R, Khosravi, H, and Wang, W. Forwarding and control element separation (ForCES) protocol specification. *Internet Requests for ...* (2010).

[39] DrPeering International. *Internet Transit Pricing*. `http://drpeering.net/white-papers/Internet-Transit-Pricing-Historical-And-Projected.php`.

[40] Du, Min, Li, Feifei, Zheng, Guineng, and Srikumar, Vivek. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. In *Proc. of the 2017 ACM SIGSAC Conference on Computer and Communications Security - CCS '17* (New York, New York, USA, 2017), ACM Press, pp. 1285–1298.

[41] Dwaraki, A, Seetharaman, S, and Natarajan, S. GitFlow: flow revision management for software-defined networks. In *Proc. of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research (SOSR)* (New York, New York, USA, 2015), ACM Press, pp. 1–6.

[42] Dwaraki, Abhishek, and Wolf, Tilman. Service instantiation in an Internet with choices. In *Proc. of the 22nd IEEE International Conference on Computer Communications and Networks (ICCCN)* (Nassau, Bahamas, Aug. 2013).

[43] Ehrgott, Matthias, and Gandibleux, Xavier. A Survey and Annotated Bibliography of Multiobjective Combinatorial Optimization. *OR-Spektrum 22*, 4 (2000), 425–460.

[44] Ergun, Funda, Sinha, Rakesh, and Zhang, Lisa. An Improved FPTAS for Restricted Shortest Path. *Inf. Process. Lett. 83*, 5 (Sept. 2002), 287–291.

[45] Explosion AI. *Prodigy: Radically efficient machine teaching.*, 2019. `https://www.prodi.gy`.

[46] Explosion AI. *sPacy: Industrial Strength Natural Language Processing in Python*, 2019. `https://www.spacy.io`.

[47] Farrel, Adrian, Vasseur, Jean-Philippe, and Ash, Jerry. A Path Computation Element (PCE)-Based Architecture. RFC 4655, Network Working Group, Aug. 2006.

[48] Feamster, Nick, and Rexford, Jennifer. Why (and How) Networks Should Run Themselves. *arXiv.org* (Oct. 2017).

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

[50] Freedman, Richard G., Jung, Hee-Tae, and Zilberstein, Shlomo. Plan and activity recognition from a topic modeling perspective. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling* (Portsmouth, New Hampshire, USA, 2014), pp. 360–364.

[51] Freedman, Richard G., Jung, Hee-Tae, and Zilberstein, Shlomo. Temporal and object relations in unsupervised plan and activity recognition. In *Proceedings of AAAI 2015 Fall Symposium on AI for Human-Robot Interaction* (Arlington, Virginia, USA, 2015), pp. 51–59.

[52] Freedman, Richard G., and Zilberstein, Shlomo. Using metadata to automate interpretations of unsupervised learning-derived clusters. In *First Workshop on Human is More Than a Labeler (BeyondLabeler) at IJCAI 2016* (New York, New York, USA, 2016), pp. 1–7.

[53] Gao, Lixin. On inferring autonomous system relationships in the Internet. *IEEE/ACM Transactions on Networking 9*, 6 (2001), 733–745.

[54] Garroppo, Rosario G., Giordano, Stefano, and Tavanti, Luca. A Survey on Multi-constrained Optimal Path Computation: Exact and Approximate Algorithms. *Comput. Netw. 54*, 17 (Dec. 2010), 3081–3107.

[55] Ghaznavi, Milad, Khan, Aimal, Shahriar, Nashid, Alsubhi, Khalid, Ahmed, Reaz, and Boutaba, Raouf. Elastic virtual network function placement. In *Proc. of IEEE International Conference on Cloud Networking (CloudNet)* (Niagara Falls, Canada, Oct. 2015), pp. 255–260.

[56] Git-A Distributed Version Control System. `http://git-scm.com`.

[57] Guo, Chuanxiong, Wu, Haitao, Tan, Kun, Shi, Lei, Zhang, Yongguang, and Lu, Songwu. Dcell: A Scalable and Fault-tolerant Network Structure for Data Centers. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication* (New York, NY, USA, 2008), SIGCOMM '08, ACM, pp. 75–86.

[58] Gyarmati, László, and Trinh, Tuan Anh. Scafida: a scale-free network inspired data center architecture. *SIGCOMM Computer Communication Review 40*, 5 (Oct. 2010), 4–12.

[59] Hansen, Pierre. Bicriterion Path Problems. In *Multiple Criteria Decision Making Theory and Application*, Günter Fandel and Tomas Gal, Eds., vol. 177 of *Lecture Notes in Economics and Mathematical Systems*. Springer Berlin Heidelberg, 1980, pp. 109–127.

[60] Hedrick, C. Routing information protocol. RFC 1058, Network Working Group, June 1988.

[61] Heller, Brandon, Scott, Colin, McKeown, Nick, Shenker, Scott, Wundsam, Andreas, Zeng, Hongyi, Whitlock, Sam, Jeyakumar, Vimalkumar, Handigol, Nikhil, McCauley, James, Zarifis, Kyriakos, and Kazemian, Peyman. Leveraging SDN layering to systematically troubleshoot networks. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking* (Hong Kong, China, 2013), HotSDN '13, ACM, pp. 37–42.

[62] Hong, Yu, Zhang, Jianfeng, Ma, Bin, Yao, Jianmin, Zhou, Guodong, and Zhu, Qiaoming. Using cross-entity inference to improve event extraction. In *Proc. of ACL* (2011), pp. 1127–1136.

[63] Huang, Lifu, Cassidy, Taylor, Feng, Xiaocheng, Ji, Heng, Voss, Clare R, Han, Jiawei, and Sil, Avirup. Liberal event extraction and event schema induction. In *Proc. of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (2016), vol. 1, pp. 258–268.

[64] Huynh, Tâm, Fritz, Mario, and Schiele, Bernt. Discovery of activity patterns using topic models. In *Proceedings of the 10th international conference on Ubiquitous computing - UbiComp '08* (New York, New York, USA, 2008), ACM Press, p. 10.

[65] Isupova, Olga, Kuzin, Danil, and Mihaylova, Lyudmila. Learning Methods for Dynamic Topic Modeling in Automated Behavior Analysis. *IEEE Transactions on Neural Networks and Learning Systems PP*, 99 (2017), 1–14.

[66] Izrailevsky, Yury, Vlaovic, Stevan, and Meshenberg, Ruslan. *Completing the Netflix Cloud Migration*. Netflix Inc, 2016. `https://media.netflix.com/en/company-blog/completing-the-netflix-cloud-migration`.

[67] Jaffe, Jeffrey M. Algorithms for Finding Paths with Multiple Constraints. *Networks 14*, 1 (1984), 95–116.

[68] Jain, Sushant, Kumar, Alok, Mandal, Subhasree, Ong, Joon, Poutievski, Leon, Singh, Arjun, Venkata, Subbaiah, Wanderer, Jim, Zhou, Junlan, Zhu, Min, Zolla, Jon, Hölzle, Urs, Stuart, Stephen, and Vahdat, Amin. B4: experience with a globally-deployed software defined wan. In *SIGCOMM '13: Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM* (New York, New York, USA, Aug. 2013), ACM Request Permissions, p. 3.

[69] Ji, Heng, and Grishman, Ralph. Refining event extraction through cross-document inference. *Proc. of ACL-08: HLT* (2008), 254–262.

[70] Ji, Ping, Ge, Zihui, Kurose, Jim, and Towsley, Don. A comparison of hard-state and soft-state signaling protocols. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications* (Karlsruhe, Germany, Aug. 2003), pp. 251–262.

[71] Jiang, J, Sun, S, Sekar, V, and Zhang, H. Pytheas: Enabling Data-Driven Quality of Experience Optimization Using Group-Based Exploration-Exploitation. *NSDI* (2017).

[72] Kazemian, Peyman, Varghese, George, and McKeown, Nick. Header Space Analysis: Static Checking for Networks. *NSDI* (2012), 113–126.

[73] Khetrapal, Gautam, and Sharma, Saurabh Kumar. Demystifying Routing Services in Software-Defined Networking. Tech. rep., Aricent Inc., 2013.

[74] Khurshid, Ahmed, Zou, Xuan, Zhou, Wenxuan, Caesar, Matthew, and Godfrey, Philip Brighten. VeriFlow: Verifying Network-Wide Invariants in Real Time. *NSDI* (2013), 15–27.

[75] Kim, Been, Rudin, Cynthia, and Shah, Julie A. The Bayesian Case Model - A Generative Approach for Case-Based Reasoning and Prototype Classification. *NIPS* (2014).

[76] Korkmaz, Turgay., and Krunz, Marwan. Multi-constrained Optimal Path Selection. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE* (2001), vol. 2, pp. 834–843 vol.2.

[77] Lantz, Bob, Heller, Brandon, and McKeown, Nick. A network in a laptop: rapid prototyping for software-defined networks. *HotNets* (2010), 19–6.

[78] Li, Jun, Dou, Dejing, Wu, Zhen, Kim, Shiwoong, and Agarwal, Vikash. An internet routing forensics framework for discovering rules of abnormal bgp events. *SIGCOMM Comput. Commun. Rev. 35*, 5 (Oct. 2005), 55–66.

[79] Li, Qi, Ji, Heng, and Huang, Liang. Joint event extraction via structured prediction with global features. In *Proc. of ACL* (2013), pp. 73–82.

[80] Li, Qi, Ji, Heng, Yu, HONG, and Li, Sujian. Constructing information networks using one single model. In *Proc. of EMNLP* (2014), pp. 1846–1851.

[81] Li, Wei, and W Moore, Andrew. A Machine Learning Approach for Efficient Traffic Classification. *MASCOTS* (2007).

[82] Li, Zhenjiang, and Garcia-Luna-Aceves, J. J. A Distributed Approach for Multi-constrained Path Selection and Routing Optimization. In *Proceedings of the 3rd International Conference on Quality of Service in Heterogeneous Wired/Wireless Networks* (New York, NY, USA, 2006), QShine '06, ACM.

[83] Linguistic Data Consortium. *ACE (Automatic Content Extraction) English Annotation Guidelines for Events*, 2008. `https://www.ldc.upenn.edu/collaborations/past-projects/ace`.

[84] Linguistic Data Consortium. *DEFT ERE Annotation Guidelines: Events v1.1*, 2013.

[85] The Linux Foundation. *Production Grade Container Orchestration*, 2016. `https://kubernetes.io`.

[86] Lloyd, Stuart P. Least Squares Quantization in PCM. *Information Theory, IEEE Transactions on 28*, 2 (Mar 1982), 129–137.

[87] Lutu, Andra, Bagnulo, Marcelo, Pelsser, Cristel, and Maennel, Olaf. Understanding the reachability of ipv6 limited visibility prefixes. In *Passive and Active Measurement* (Cham, 2014), Michalis Faloutsos and Aleksandar Kuzmanovic, Eds., Springer International Publishing, pp. 163–172.

[88] Mahajan, Ratul, Wetherall, David, and Anderson, Tom. Understanding BGP misconfiguration. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications - SIGCOMM '02* (New York, New York, USA, 2002), ACM Press, p. 3.

[89] Manning, Christopher D, and Schütze, Hinrich. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[90] Martins, Ernesto Queirós Vieira. On a Multicriteria Shortest Path Problem. *European Journal of Operational Research 16*, 2 (1984), 236 – 245.

[91] Martins, Ernesto Queirós Vieira. *Bibliography of papers on Multiobjective Optimal Path Problems*. `http://www.mat.uc.pt/~eqvm/cientificos/biblio/mo.ps.Z`, 1996.

[92] McCallum, Andrew Kachites. Mallet: A machine learning for language toolkit. http://mallet.cs.umass.edu, 2002.

[93] McConaghy, Trent. *Blockchains for Artificial Intelligence*. BigChain DB, 2016. `https://blog.bigchaindb.com/blockchains-for-artificial-intelligence-ec63b0284984`.

[94] McDonald, Ryan, Nivre, Joakim, Quirmbach-Brundage, Yvonne, Goldberg, Yoav, Das, Dipanjan, Ganchev, Kuzman, Hall, Keith, Petrov, Slav, Zhang, Hao, Täckström, Oscar, et al. Universal dependency annotation for multilingual parsing. In *Proc. of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)* (2013), vol. 2, pp. 92–97.

[95] McKeown, Nick, Anderson, Tom, Balakrishnan, Hari, Parulkar, Guru, Peterson, Larry, Rexford, Jennifer, Shenker, Scott, and Turner, Jonathan. OpenFlow: enabling innovation in campus networks. *SIGCOMM Computer Communication Review 38*, 2 (Apr. 2008), 69–74.

[96] McKeown, Nick, Anderson, Tom, Balakrishnan, Hari, Parulkar, Guru, Peterson, Larry, Rexford, Jennifer, Shenker, Scott, and Turner, Jonathan. OpenFlow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev. 38*, 2 (Mar. 2008), 69–74.

[97] Medina, A, Lakhina, A, Matta, I, Byers, J Modeling Analysis, of Computer, Simulation, and on, Telecommunication Systems 2001 Proceedings Ninth International Symposium. BRITE: an approach to universal topology generation. In *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2001. Proceedings. Ninth International Symposium on* (2001), IEEE Comput. Soc, pp. 346–353.

[98] Moy, John. OSPF version 2. RFC 2328, Network Working Group, Apr. 1998.

[99] N. Foster, et al. Frenetic: A network programming language. In *Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming* (2011).

[100] N. Handigol, et al. Where is the debugger for my software-defined network? In *Proceedings of the first HotSDN workshop* (New York, New York, USA, Aug. 2012), pp. 55–60.

[101] N. McKeown, et al. OpenFlow: enabling innovation in campus networks. *SIGCOMM Computer Communication Review 38*, 2 (Mar. 2008), 69.

[102] Natarajan, S, Huang, Xin, and Wolf, T. Efficient conflict detection in flow-based virtualized networks. In *Computing, Networking and Communications (ICNC), 2012 International Conference on* (2012), IEEE, pp. 690–696.

[103] Natarajan, Sriram, Ramaiah, Anantha, and Mathen, Mayan. A Software defined Cloud-Gateway automation system using OpenFlow. *CLOUDNET* (2013), 219–226.

[104] Nguyen, Thien Huu, Cho, Kyunghyun, and Grishman, Ralph. Joint event extraction via recurrent neural networks. In *Proc. of NAACL-HLT* (2016), pp. 300–309.

[105] Nunes, Bruno Astuto A, Mendonca, Marc, Nguyen, Xuan Nam, Obraczka, Katia, and Turletti, Thierry. A Survey of Software-Defined Networking - Past, Present, and Future of Programmable Networks. *IEEE Communications Surveys and Tutorials* (2014).

[106] Open Networking Foundation. *Software-defined Networking: The New Norm for Networks*, 2012.

[107] Openflow specification. `https://www.opennetworking.org/sdn-resources/technical-library`.

[108] Orsini, C., King, A., and Dainotti, A. BGPStream: a software framework for live and historical BGP data analysis. Tech. rep., Center for Applied Internet Data Analysis (CAIDA), Oct 2015.

[109] Orsini, Chiara, King, Alistair, Giordano, Danilo, Giotsas, Vasileios, and Dainotti, Alberto. Bgpstream: A software framework for live and historical bgp data analysis. In *Proceedings of the 2016 Internet Measurement Conference* (New York, NY, USA, 2016), IMC '16, ACM, pp. 429–444.

[110] Pelegrin, Blas, and Fernández, Pascual. On the sum-max bicriterion path problem. *Computers & Operations Research 25*, 12 (1998), 1043 – 1054.

[111] Pennington, Jeffrey, Socher, Richard, and Manning, Christopher D. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)* (2014), pp. 1532–1543.

[112] Pfaff, B, Pettit, J, Koponen, T, Jackson, E, and Zhou, A. The design and implementation of open vswitch. … *USENIX Symposium on …* (2015).

[113] Porras, P, Shin, S, Yegneswaran, V, and Fong, M. A security enforcement kernel for OpenFlow networks. In *Proceedings of the first HotSDN Workshop* (2012).

[114] Qiu, Jian, Gao, Lixin, Ranjan, Supranamaya, and Nucci, Antonio. Detecting bogus BGP route information: Going beyond prefix hijacking. In *2007 Third International Conference on Security and Privacy in Communications Networks and the Workshops - SecureComm 2007* (2007), IEEE, pp. 381–390.

[115] Ramachandran, Anirudh, Bhandankar, Kaushik, Tariq, Mukarram Bin, and Feamster, Nick. Packets with provenance. In *Proceedings of the ACM SIGCOMM Poster* (2008).

[116] Ramaswamy, Ramaswamy, Weng, Ning, and Wolf, Tilman. Characterizing network processing delay. In *Proc. of IEEE Global Communications Conference (GLOBECOM)* (Dallas, TX, Nov. 2004), pp. 1629–1634.

[117] Reich, J, Monsanto, C, Foster, N, the, J Rexford Proceedings of, and 2013. Composing software defined networks. *Citeseer*.

[118] RIPE RIR. *RIPE Network Coordination Center*, 2018. `https://www.ripe.net`.

[119] S. Jain, et al. B4: Experience with a globally-deployed software defined wan. *SIGCOMM Comput. Commun. Rev. 43*, 4 (Aug. 2013), 3–14.

[120] Saito, Takaya, and Rehmsmeier, Marc. The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets. *PLOS ONE 10*, 3 (Mar. 2015), e0118432.

[121] SANS Technology Institute, Security Laboratory. *The 6 Categories of Critical Log Information*, 2018. `https://www.sans.edu/cyber-research/security-laboratory/article/6toplogs`.

[122] Sasisekharan, Raguram, Seshadri, V, and Weiss, Sholom M. Proactive Network Maintenance Using Machine Learning. *KDD Workshop* (1994).

[123] Schwartz, Beverly, Jackson, Alden W, Strayer, W Timothy, Zhou, Wenyi, Rockwell, R Dennis, and Partridge, Craig. Smart packets: applying active networks to network management. *ACM Transactions on Computer Systems (TOCS) 18*, 1 (Feb. 2000).

[124] Sermpezis, Pavlos, Kotronis, Vasileios, Gigis, Petros, Dimitropoulos, Xenofontas, Cicalese, Danilo, King, Alistair, and Dainotti, Alberto. ARTEMIS: Neutralizing BGP Hijacking within a Minute. *arXiv.org* (Jan. 2018).

[125] sFlow Standard v5. `http://sflow.org/sflow_version_5.txt`.

[126] Shalimov, A, Zuikov, D, Zimarina, D, and Pashkov, V. Advanced study of SDN/OpenFlow controllers. In *Proceedings of the 9th ...* (2013).

[127] Sherwood, R. Can the production network be the testbed. In *USENIX OSDI* (2010).

[128] Sherwood, R, Gibb, G, and Yap, K K. Flowvisor: A network virtualization layer. *Open Networking Foundation* (2009).

[129] Singhal, Amit. Introducing the knowledge graph: things, not strings, 2012. `https://googleblog.blogspot.com/2012/05/introducing-knowledge-graph-things-not.html`.

[130] Sommer, Robin, and Paxson, Vern. Outside the Closed World - On Using Machine Learning for Network Intrusion Detection. *IEEE Symposium on Security and Privacy* (2010).

[131] Spring, N, Mahajan, R, Wetherall, D, and Anderson, T. Measuring ISP topologies with Rocketfuel. *Networking, IEEE/ACM Transactions on 12*, 1 (2004), 2–16.

[132] Strubell, Emma, Verga, Patrick, Belanger, David, and McCallum, Andrew. Fast and Accurate Entity Recognition with Iterated Dilated Convolutions. *arXiv.org* (Feb. 2017).

[133] Sun, Yizhou, Barber, Rick, Gupta, Manish, Aggarwal, Charu C, and Han, Jiawei. Co-author relationship prediction in heterogeneous bibliographic networks. In *Proc. of ASONAM-ICASNAM2011* (2011).

[134] Sun, Yizhou, Han, Jiawei, Yan, Xifeng, Yu, Philip S, and Wu, Tianyi. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proc. of the VLDB Endowment* (2011).

[135] Szekely, Pedro, Knoblock, Craig A, Slepicka, Jason, Philpot, Andrew, Singh, Amandeep, Yin, Chengye, Kapoor, Dipsy, Natarajan, Prem, Marcu, Daniel, Knight, Kevin, et al. Building and using a knowledge graph to combat human trafficking. In *International Semantic Web Conference* (2015), Springer, pp. 205–221.

[136] Tarapata, Zbigniew. Selected Multicriteria Shortest Path Problems: An Analysis of Complexity, Models and Adaptation of Standard Algorithms. *International Journal of Applied Mathematics and Computer Science 17*, 2 (June 2007).

[137] Tsaggouris, George, and Zaroliagis, Christos. Multiobjective Optimization: Improved FPTAS for Shortest Paths and Non-linear Objectives with Applications. In *Algorithms and Computation*, Tetsuo Asano, Ed., vol. 4288 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2006, pp. 389–398.

[138] Ulungu, E. L., and Teghem, J. Multi-objective Combinatorial Optimization Problems: A Survey. *Journal of Multi-Criteria Decision Analysis 3*, 2 (1994), 83–104.

[139] Wang, Z., and Crowcroft, J. Quality-of-service routing for supporting multimedia applications. *Selected Areas in Communications, IEEE Journal on 14*, 7 (Sep 1996), 1228–1234.

[140] Warburton, A. Approximation of Pareto Optima in Multiple-objective, Shortest-path Problems. *Oper. Res. 35*, 1 (Feb. 1987), 70–79.

[141] Waxman, B.M. Routing of multipoint connections. *Selected Areas in Communications, IEEE Journal on 6*, 9 (Dec 1988), 1617–1622.

[142] Wendlandt, L., Kummerfeld, J., and Mihalcea, R. Factors influencing the surprising instability of word embeddings. *NAACL-HLT* (2018).

[143] Wetherall, D J, and Guttag, J V. ANTS: A toolkit for building and dynamically deploying network protocols. *Open Architectures and …* (1998).

[144] Wolf, Tilman. Service-centric end-to-end abstractions in next-generation networks. In *Proc. of Fifteenth IEEE International Conference on Computer Communications and Networks (ICCCN)* (Arlington, VA, Oct. 2006), pp. 79–86.

[145] Wolf, Tilman, Griffioen, James, Calvert, Kenneth L., Dutta, Rudra, Rouskas, George N., Baldin, Ilya, and Nagurney, Anna. ChoiceNet: Toward an Economy Plane for the Internet. *SIGCOMM Comput. Commun. Rev. 44*, 3 (July 2014), 58–65.

[146] Wundsam, Andreas, Levin, Dan, Seetharaman, Srini, and Feldmann, Anja. OFRewind: enabling record and replay troubleshooting for networks. In *Proceedings of the 2011 USENIX ATC* (June 2011).

[147] Xue, Guoliang, Zhang, Weiyi, Tang, Jian, and Thulasiraman, K. Polynomial Time Approximation Algorithms for Multi-Constrained QoS Routing. *Networking, IEEE/ACM Transactions on 16*, 3 (June 2008), 656–669.

[148] Zhang, Yuyu, Dai, Hanjun, Kozareva, Zornitsa, Smola, Alexander J, and Song, Le. Variational reasoning for question answering with knowledge graph.

[149] Zhao, Xiaoliang, Pei, Dan, Wang, Lan, Massey, Dan, Mankin, Allison, Wu, S Felix, and Zhang, Lixia. An analysis of BGP multiple origin AS (MOAS) conflicts. In *Proceedings of the First ACM SIGCOMM Workshop on Internet Measurement - IMW '01* (New York, New York, USA, 2001), ACM Press, p. 31.

[150] Zheng, Weiguo, Yu, Jeffrey Xu, Zou, Lei, and Cheng, Hong. Question answering over knowledge graphs: question understanding via template decomposition. *Proc. of the VLDB Endowment 11*, 11 (2018), 1373–1386.