Towards Scalable Security State Management in The Cloud

by

Abdulhakim Sabur

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved August 2022 by the
Graduate Supervisory Committee:

Ming Zhao, Chair
Guoliang Xue
Hasan Davulcu
Yanchao Zhang

ARIZONA STATE UNIVERSITY

May 2023

ABSTRACT

Modern data center networks require efficient and scalable security analysis approaches that can analyze the relationship between the vulnerabilities. Utilizing the Attack Representation Methods (ARMs) and Attack Graphs (AGs) enables the security administrator to understand the cloud networks current security situation at the low-level. However, the AG approach suffers from scalability challenges. It relies on the connectivity between the services and the vulnerabilities associated with the services to allow the system administrator to realize its security state. In addition, the security policies created by the administrator can have conflicts among them, which is often detected in the data plane of the Software Defined Networking (SDN) system. Such conflicts can cause security breaches and increase the flow rules processing delay.

This dissertation addresses these challenges with novel solutions to tackle the scalability issue of Attack Graphs and detect security policy conflicts in the application plane before they are transmitted into the data plane for final installation. Specifically, it introduces a segmentation-based scalable security state (S3) framework for the cloud network. This framework utilizes the well-known divide-and-conquer approach to divide the large network region into smaller, manageable segments. It follows a well-known segmentation approach derived from the K-means clustering algorithm to partition the system into segments based on the similarity between the services. Furthermore, the dissertation presents unified intent rules that abstract the network administration from the underlying network controllers format. It develops a networking service solution to use a bounded formal model for network service compliance checking that significantly reduces the complexity of flow rule conflict checking at the data plane level. The solution can be expended from a single SDN domain to multiple SDN domains and hybrid networks by applying network service function chaining (SFC) for inter-domain policy management.

To My Mother, Samar, Who Sacrificed Her Life To Raise Us ......

To My Son, Hashem, Your Smile Will Always Keep Me Motivated ......

ACKNOWLEDGMENTS

To whoever is reading this part. You are the reason for me to write it. You kept me working, motivated, and reaching to where I am today. My words cannot express how grateful I am to you, my professors, my siblings, my parents, and above all, my better half.

Dr. Dijiang Huang, you have accepted me and included me among your SNAC family. You taught me how to become a good researcher, how to write papers, and how to conduct effective research. Thank you for meeting me anytime I wanted, thank you for keeping me motivated, and thank you for pushing me outside my comfort zone. You are the corner stone and the enabler for me to achieve my dream.

My committee members, Dr. Ming Zhao, who stepped-in in a critical time and guided me to complete my dissertation. Prof. Guolinag Xue, Prof. Hasan Davulcu, and Prof. Yanchao Zhang, thank you for all the support and the time commitment you have given me to successfully finish my dissertation

My mentor, best friend, brother, role model, and counselor, Mohammad. Thank you for being there for me whenever I needed you. Thank you for going through all of this experience before me, and telling me about your mistakes. You are my motivational engine and I could not have done this without your support, your guidance, and your honest advises to me. You held my hand since day one in school, and you helped me grow up a better man. I will never return all of your favors to me, but I promise you that I will be there whenever you need me, my dear brother.

My father, Mansour. Thank you dad for planning my future. Thank you for protecting me and my siblings from all kinds of risks and danger in the world. While you could not get your own Ph.D., this work would not have been possible without your guidance, your love, support, and positive discussions. You raised me to be a good man, an honest man, and that I seek pure to please Allah. I promise you I will be like that for the rest of my life.

My mother, Samar. You are my shine, my love, my joy. Thank you for your sacrifices in this life to raise me and my brothers. Thank you for making the best food in the world for us. Thank you for bringing us anything we wanted. This dissertation is dedicated for you mom, and I will always pray Allah to protect you and help me to return part of the favors you did for us.

Above all, my friend, partner, love and wife, Maryam, who stood by me every moment and gave me her endless and constant support. Thank you for being in my life, without your existence, I would not have been able to finish my degree on time, thank you for your understanding and patience while I worked long hours during nights. Thank you for waiting for me to come back home. Thank you for taking care of our son. Your efforts are countless and I will never forget what you do for our family. You are the one who brings peace and love to our home.

My sisters, Roah, Halah, and Zain. Your smiles makes me happy and motivated. Thank you for a great childhood, each one of you has a special place in my hurt. Your motivation to me enabled me to succeed.

My SNAC group friends, Ankur Chowdhary, Sowmya Myneni, Adel Alshamrani, Garima Agrawal, Yuli Deng, Krit Jha, and Neha Vadnere . Thank you for the great time and comapny while we were conducting an amazing research work and to help me succeed at ASU. I could not have made it without your help on different projects, long discussions, and papers collaboration.

Lastly, I gratefully acknowledge the financial support and scholarship I received from Taibah University through Saudi Arabian Cultural Mission (SACM).

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

The advancement of technology and reliant on automated systems in the day-to-day organization's operations poses a challenge on the security and privacy of these systems. There has been extensive research on the analysis and understanding of the security state of the different computing systems. Because of the huge advancement of cloud computing, many companies are migrating their IT infrastructure into the cloud. Furthermore, managing the cloud infrastructure involves managing advanced networking infrastructure as well.

With new technology comes new challenges. The security challenges are nowadays considered the top challenges and concerns for IT administrators because of the severity and how one attack can damage the entire network or cause data loss. This massive prices made the network administrators think twice about which services to put in the cloud, and it presented a problem for them to analyze the security state of those services. Managing the security situation requires analyzing the security vulnerabilities in the system and understanding how those vulnerabilities are interconnected. Research shows analyzing and predicting vulnerabilities at an early stage can save the organization from being exploited [4, 3].

System and Network vulnerabilities are defined as the weaknesses in the software, hardware, or network resources on the network that can be exploited by a persistent attacker who would use it to enter to or shut down a network. Another definition states that the vulnerability is the set of conditions, a weakness of or an absence of security procedure, or technical, physical or other controls that could be exploited by an adversary [123]. Vulnerabilities can exist also due to misconfigurations in policies and procedures, especially security policies that state the access rule and the security processes to handle the computing resources and data [85].

The sources of vulnerabilities vary due to the nature of the computing systems. The vulnerabilities can spread across any part of the system and thus we cannot identify a clear list of vulnerabilities sources. However, there are agencies that finds and classify vulnerabilities into their prospectus categories. These agencies include but not limited to: a) Computer Emergency Response Team (CERT) b) National Vulnerability database (NVD) and c) ExploitDB, a database of exploitable vulnerabilities and their proof of exploitation. These sources report vulnerabilities that are discovered due to poor design flaws, poor security management, incorrect implementation, exploited resources such as data leakage, etc.

Research has shown consistent increase in the number of vulnerabilities year-after-year [112] due to the increasing attention cybersecurity is getting from governments and private sectors. The huge number of vulnerabilities will posses a challenge on the network administrators on how they can prioritize the one vulnerability over the other. Furthermore, these rapid increase forces the administrator to ask how can they know the vulnerabilities that can create a dedicated attack path for an adversary to exploit the back-end system resources? This type of analysis is referred to as a low level-security analysis which requires the administrator to analyze the relationship between the vulnerabilities in order to prioritize security defense mechanism.

Security analysis has been an effective research area that attracted a lot of attention due to its importance as part of the defense mechanism for any system [29, 98, 155, 88, 95, 71, 154]. With growing networks, the need for automating the security defenses rise. Automation would allow the administrator to setup the high-level security and networking requirement such that the turn their attention into more important problem, or problems that are newly discovered such as zero-day attacks.

One of the analysis approaches that allows for such automation is the *Graphical security models* (GrSM) such as *Attack Graphs* (AGs) and *Attack Trees* (ATs). AGs are defined as a data structure, used to model all possible critical attack paths and vulnerabilities of a system, which an adversary can exploit in order to achieve his/her attacking goals. Researchers have developed methods and approaches to enhance the GrMS usability However, generating and analyzing AG in a security system requires a significant generation a AG usability and efficiency. However, due to the development of networks (e.g., static network to dynamic networks such as the cloud computing networks), and also the availability of AG generation tools like MulVal [115], led to new challenges in using the AG for security analysis. Specifically, the AG scalability problem when the AG is computed, the attack path searching when performing AG-based attack scenario analysis can be an NP-hard problem as noted by [49, 130], which depends on the density of a given AG. In a large network system, AGs are often incomprehensible to a user due to its complex interdependence among vulnerabilities. The identification of information regarding vulnerability dependencies becomes increasingly difficult as the number of services and vulnerabilities are increasing in the network system.

Another challenge the security administrator faces is that when the network IT infrastructure is migrated into the cloud, it requires them to specify different security and networking policies to ensure smooth operation of the networking services. Policies are defined as rules that correspond to the characteristics of the data plane flows, switches, or hosts. Because the cloud networks are often run on advanced networking solutions such as Software Defined Networking (SDN) [117, 5, 86, 30], researchers have focused on developing solutions for detecting the policy conflicts in SDN-based networks [34, 66, 124, 35]. Because the SDN networks allows for shared control domain by allowing the SDN controllers to of multiple domains to form a cluster, the shared control plane policies can create a conflict between each other and between the implemented policies in the data plane. Furthermore, unlike conventional networks, SDN allows diverse applications to establish flow rules on the data plane using application plane APIs. These flow rules and will not be checked by the network administrator, which makes them vulnerable to policy conflict violation. In addition, the security policy conflict can occur due to overlapping between the header space of the rules, redundant action, or when a rule is a subset of another rule.

Security policy conflicts have severe consequences. They can create security vulnerabilities allowing attackers to exploit the system, or they can limit the efficiency and effectiveness in the system. Some of the issues that are heightened in an SDN-based network are issues caused by the flow rules chaining,, cross-layer policy conflicts, partial matches and by set-field actions as denoted by Pisharody *et al.* [124]. Recently, the SDN networks allowed the administrators to add policies using high-level intents. The application plane policies that are used to specify a certain action(s) for a target network are called to an intent[35, 117]. The intent help the administrator to use an abstracted language without knowing the details of the control and data planes to specify their requirements. Applications on the cloud can also use the intents using the APIs without the administrators interference. Once the intent is specified, it is translated into the prospective control domain's controller language. The controller then implement the received policy and install it on the data plane. Because of the multiple parties installing and modifying intents, the policies that are specified at the higher-level are often conflicted between each other and/or between existing policies. The network verification and testing domain often allows the administrator to check for bugs in the networks through error in the detailed configurations, which forces the control plane to reconverge to a new path.

The objective of this dissertation is to present a scalable security state analysis solution for the cloud networks. Specifically, we design and present a solution to overcome the AG scalability problem, which is the foundational layer for analyzing the low-level security state in the system. Furthermore, we present an intent-based security policy conflict checking and resolution that depends utilizing the bounded model checking [18] for formal verification of the high-level security policies. The dissertation is organized as follows:

- **Chapter 2: Background and Motivation.** In this chapter, we shed the light on the key technology used in this dissertation like Software Defined Networking (SDN), OpenFlow protocol as a foundational SDN protocol, Attack Graph (AG) scalability problem, and security policy conflict checking.

- **Chapter 3: S3: A DFW-based Scalable Security State Analysis Framework for Large-Scale Data Center Networks.** This chapter focuses on 1) explaining in detail the AG scalability problem. 2) present a distributed firewall (DFW) model to control the AG reachability, 3) present a segmentation model using the DFW to divide the large graph into smaller manageable graphs using the divide and conquer approach, 4) present an optimization approach that determines the best number of segments for the graph, and 5) through a series of experiments, we evaluate the proposed DFW-based AG segmentation approach and test the graph density, AG generation time, the SDN controller overhead, and measure the cycle detection time.

- **Chapter 4: Intent-Driven Security Policy Management for Software-Defined Systems.** In this chapter, we provide an overview of the *Intent-based Networking*. We provide a rational and abstraction approach for the network administrator such that they are do not have to adhere to every SDN controller's language to specify the high-level intents. Specifically, we 1) present INTPOL, an intent-based framework for translating security policy requirements into a unified format, 2) INTOPOL allows easy expression of complex scenarios like service function chaining (SFC) and hybrid network scenario, 3) we present a bounded-model checking (BMC) approach for the application plane policies to compose the translated intent's policies into control and data plane rules and checking for a possible conflict between those rules, 4) we demonstrate the benefits of INTPOL by showing the composition verification, and conflict resolution times are reduced when INTPOL is utilized for application plane policy checking. We used the *Stanford Topology* [81] dataset for the purpose of showing how the data plane conflict checking time is reduced using INTPOL as opposed to not using it.

- **Chapter 5: Conclusion and Future Work.** This chapter summarizes our work in this dissertation. We provide an overall description of the presented solutions summarize the results. The chapter concludes with a possible future work to extend the work in this dissertation.

Chapter 2

BACKGROUND AND MOTIVATION

2.1 Software Defined Networking

Software Defined Networking (SDN) is a networking paradigm that has emerged to enable high and scalable networking communication [13, 136, 23, 60, 163, 59]. The open networking foundation (ONF) defines SDN as "In the SDN architecture, the control and data planes are decoupled, network intelligence and state are logically centralized, and the underlying network infrastructure is abstracted from the applications. In SDN architecture, the system is composed of three layers: the control plane, the data plane, and the application plane. SDN decouples the control plane and the data plane. Using SDN-based networking, the administrator is able to implement a centralized control plane such that all of the forwarding rules and policies are implemented and specified in one entity. The data plane will have all the forwarding devices such as OpenFlow switches and routers. Because of the control plane and data plane separation, the data plane devices become a simple forwarding entities. The application plane will have several applications to enable high-level user requirements to be implemented at the data plane devices. Using APIs, there are two interfaces in the SDN architecture that enables the communication through the layers; The north bound interfaces enables the communication between the application plane and the control plane. The south bound interface enables the control plane to communicate with the data plane. Later in chapter 4, we expand the definition of high-level user's security and networking policy and we explain how these high-level security policies can have conflicts among them.

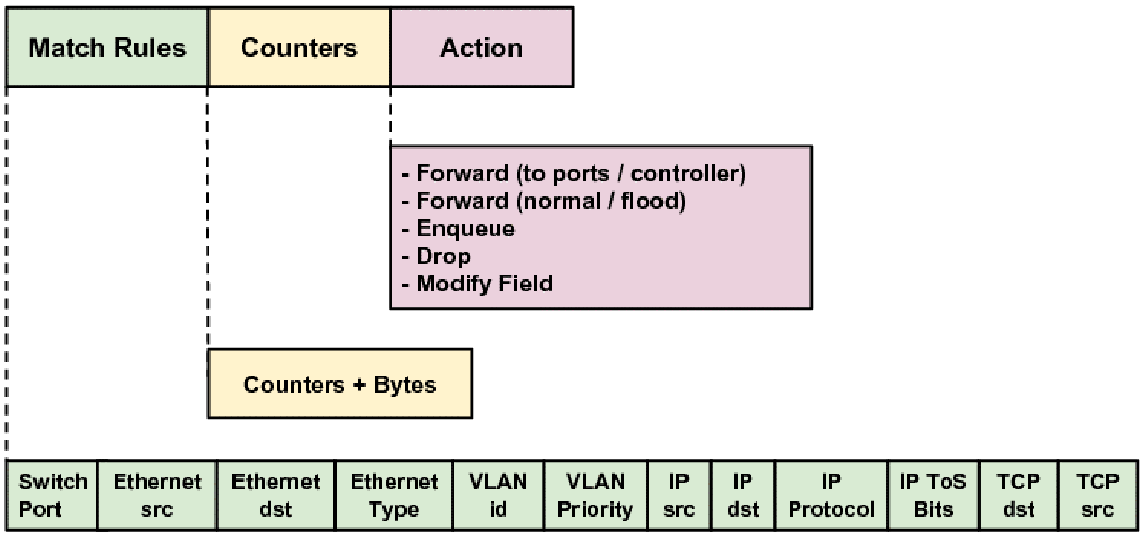


Figure 2.1: OpenFlow Protocol Header Fields.

2.1.1 OpenFlow

OpenFlow is an standardized protocol used to enable southbound communication between the SDN controller and forwarding elements (switches). OpenFlow properties can be summarized as follows:

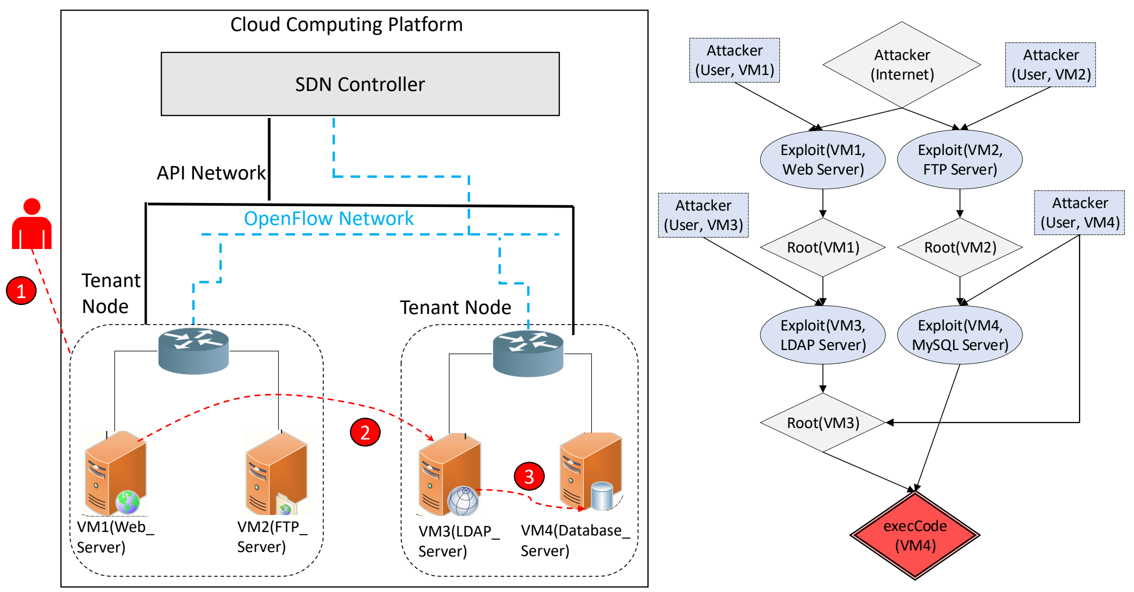
- OpenFlow is one of the first software-defined networking (SDN) standards and defined the communication protocol between SDN controllers and the forwarding plane of networking devices.

- Benefits include its programmability, centralized intelligence, and how it abstracts network architecture We show a reference architecture of OpenFlow protocol in Figure 2.1 adopted from [44].

2.2 Motivation

To realize the effect of DFW on AG, we present a motivating example of lateral movement attack. In a large system such as a data-center network, traditional and centralized firewall system capabilities lay within the *north-south* flow [130]. The firewall acts as a central entity that determines the trust level associated with the traffic under considerations (allowed vs denied traffic). However, cyber attackers are gaining more power and knowledge. That said, once an attacker is inside the system, by compromising an internal user and using their credentials, they can laterally move inside the network and exploit key resources. Moreover, centralized firewalls do not protect networks from *multi-stage* attacks using *lateral movement* [32]. Usually, everyone on the internal networks is considered an allowed user, and the traditional firewall-based defense mechanisms do not rigorously inspect the traffic within the internal network as the firewall initially allows the access to the internal nodes. In a data-center environment, the amount of *east-west* traffic is about 76%, while it is 17% only for *north-south* traffic. Therefore, there is a high possibility of attacks involving lateral movement.

Consider the system shown in Figure 2.2 (a). The system is built in a general cloud computing platform framework. There are four end-hosts in the system, each of which runs a different service. In this example, the attacker is located on the internet, and the attack goal is to compromise the database server at node *VM4* that is running MySQL Server. Thus, the attacker is conducting a vulnerability exploitation attack to obtain privileged access and then exfiltrate data. In this threat model, the attacker will reach their goal by exploring multiple attack paths, as illustrated in Figure 2.2 (b). In Figure 2.2 (a), compromising the Web-server running on VM1 is the first step of the multi-stage attack. An attacker can use privileges gained from step (1)



(a) Cloud network with multi tenant. Lateral movement attack steps to compromise Database server are shown.

(b) Attack graph with attack goal VM4 (MySQL Server)

Figure 2.2: Representation Of Vulnerability Information and Corresponding Attack Graph in a Multi-tenant Data-center Network. The Absence of Distributed Firewall *(Dfw)* Allows the Attacker to Compromise the First Tenant Node and Laterally Move Towards the Second Tenant Node to Compromise and Achieve Their Attack Goal.

to advance in the system by exploring and exploiting the rest of the machines, as shown in steps (2) and (3), respectively. Traditional security architectures, such as centralized firewalls, will not detect nor prevent such lateral movement attacks since their purpose is to protect the network from attackers located outside the network. Also, the communication that allows the attacker to move laterally will look legitimate to the centralized firewall. However, suppose the system administrator adopts a DFW-based framework. In this case, it will be possible to analyze such exploits and create security policies using DFW rules to prevent attackers from achieving their attack goal (lateral movement).

Note that detection of lateral movement in a real attack scenario cannot be conducted using a vulnerability scanner such as Nessus alone. The identification of lateral movement can be done using the traffic received by the SDN controller, and the connectivity information between different tenants. The simulation of a lateral movement in the example Figure 2.2 (a) is based on the assumption that since VM1 has a Web Server, and the service present on VM1 is vulnerable, the goal of the attacker is to compromise key network services such as VM3 (LDAP), and VM4 (Database). It is likely that once attacker compromises VM1, he/she can scan for services present on local network, that are vulnerable, and compromise them (post-condition of the attack graph).

This attack model that involves lateral movement can be reinforced using network traffic analysis. The traffic information can be collected using an SDN controller. If there is a traffic request between VM1 and VM3, the SDN controller can log this information, and consider this as a possible case of lateral movement. We can additionally use an intrusion detection system (IDS) between different tenants to check traffic signature for the signs of lateral movement. This will increase the confidence in the possible attack path taken by an attacker with lateral movement as an intermediate step. The use of IDS agents if done in-line can increase the network latency, and it is a reactive solution, thus we can conduct cost-benefit analysis of selectively using IDS and traffic captured using an SDN controller to identify lateral movement.

The system administrator can configure the DFW to have granular rules that inspect the source of communication (attacker's L2 & L3 addresses) and inspect the user credentials for the required service access. The DFW can also prevent normal users who are not authorized to access a particular service in another tenant. For example, Palantir corporation explains that the most critical windows vulnerabilities is the SMB vulnerability. It allows the attackers to create backdoor and access to an administrative privileges which then can be used to remove certain restrictions on an internal network service. The use of *DFW rule* policy to block communication on the SMB ports is an effective solution that prevented such exploitation.

2.3 Attack Graph

The attack graph (AG) is a data structure developed to model the system's security state and identify the possible attack scenarios. The AG is a tuple consisting of states and transitions between the states. The first state is the initial state, which indicates the attacker's location. After that, the transition between states is conditioned upon exploiting the next node and connecting the current node and the target node. Hence, the AG has stages where the attacker transitions from one stage to another under exploiting the existing vulnerabilities. AG aims to model the vulnerable services and their relationship only because the idea is developed to understand the security situation in the system and not to emphasize normal or benign services.

There are two main types of AGs: 1) A directed graph that model network states as nodes, and exploits as edges that change the states of the network, depending on what exploits are performed [36]. 2) A *exploit dependency graph*, in which there are pre-conditions that need to be satisfied in order for an exploitation to be successful and result in post-conditions. Therefore, the movement of an adversary in the graph is based on the ability to exploit a pre-condition.

In this proposal, we use the *exploit dependency graph* [64], since it directly models dependencies between the vulnerabilities in a computer networked system. Also, all services for application-based on networked-based attacks are related in this graph and it shows what are the pre-requisites (pre-conditions) and post-conditions for those attacks. *Nodes* in such an AG are not the network states, but rather, they are *vulnerabilities*. AG can be formally defined as follows:

Definition 1 *(Attack Graph (AG)) An attack graph is represented as a graph $G = \{V, E\}$, where V is the set of nodes and E is the set of edges of the graph G, where*

1. *$V = N_C \cup N_D \cup N_R$, where N_C denotes the set of conjunctive or exploit nodes (pre-condition), N_D is a set of disjunctive nodes or result of an exploit (post-condition), and N_R is the set of a starting nodes of an attack graph, i.e. root nodes.*

2. *$E = E_{pre} \cup E_{post}$ are sets of directed edges, such that $e \in E_{pre} \subseteq N_D \times N_C$, i.e., N_C must be satisfied to obtain N_D. An edge $e \in E_{post} \subseteq N_C \times N_D$ means that condition N_C leads to the consequence N_D.*

MulVAL [115] is a well-known open-source tool to generate an attack graph. It uses datalog and logic programming as its modeling language [25]. The input to MulVAL is vulnerability information, which includes but is not limited to: Common Vulnerabilities and Exposures (CVE-ID) [109], affected applications or services, vulnerability consequences (whether the vulnerability results in data loss, remote exploitation, data integrity, etc.). Besides, network reachability information is required for each host, including IP address, vulnerable services or applications, and corresponding ports and protocols.

An example of *vulnerability* information, network *service* information, and *Host Access Control List* (HACL) represented in *datalog* format as described in MulVAL model [115] is shown as follows:

```
vulExists (ipaddr, cve-id, service)
networkServiceInfo(ipaddr, service, prot, port)
hacl(srcip, dstip, prot, port)
```

The AG uses HACL tuples to model network and firewall configurations. It uses a general rule to test and specify reachability information (i.e., any host can access any host using any port and protocol).

2.3.1 Attack Graph Scalability Challenge:

As can be seen in Figure 2.2 (b), a network consisting of 4 hosts resulted in a graph of 13 nodes. Large data-center networks have thousands of services, servers, and VMs. The expected AG size of such a system is enormous due to representing the network state using a conditional or combination of conditional and exploit representation of the security situation. This complicated representation leads to a massive number of nodes and edges in the AG. The attack graph generation is polynomial in terms of the number of hosts $O(N^6)$ [6], and $O(N^2)$-$O(N^3)$ [115]. Attack graph generation takes 2-3 minutes for current attack graph methods, even when the number of hosts is ~ 450 [80]. Naturally, this leads to the state space explosion problem in generation and analysis of attack graph as Yi *et al.* [153] discussed.

2.4 Security Policy Conflict Checking

The large size of the typical data center networks imposes a multi-domain nature for the network infrastructure. The configuration of this network requires setting-up a certain policies. These policies are divided into a security and networking policies. The way the administrator sets up the policies is by implying a high-level intent, where he/she can specify a generic requirement through the intent's syntax such as "host H1 in domain D1 can communicate with host H2 in Domain D2". The submitted intent will be then compiled and transmitted into the control plane of the prospective intent's hosts. Now each domain's control plane is responsible for generating the data plane flow rules by checking the received parameters from the intent. Next, the control plane needs to check for a possible conflict between the data plane's flow rules [34]. This process is considered expensive and inefficient in detecting the conflict between the policies, because it was performed after the SDN controller received intent and it started to compile it into a flow rules. We will show later in Chapter 4 that the number of SDN flow rules was much higher using this approach as opposed of detecting the conflict at the application plane.

Why Security Policy Conflict Checking is Important?

Misconfiguration of Virtual Network Functions (VNFs) such as intrusion prevention system and firewall can lead to security violations and business outage for an organization. In July 2019, *Capital One* bank, one of the largest banks in the United States suffered from a data breach. The breach caused leakage of over 100 Million customer's private data such as credit card numbers, Social Security Numbers *SSN*, and other information [102]. An attacker was able to access this critical information by exploiting a security vulnerability that existed in the company's cloud server, which was hosted by *Amazon Web Services (AWS)*. A misconfigured firewall application on AWS allowed the attacker to access the user's data without authentication. Although the AWS instance was itself secure, the bank's misconfigured security policy led to the data breach. An effective and efficient policy checking module is required to identify and address issues associated with misconfigured and conflicting security policies. The VNFs perform operations over the traffic received such as forwarding, packet header modification, and routing. The domain of operations provided by different VNFs shows dependence upon each other. For instance, a Firewall can also provide Network Address Translation (NAT) feature.

Chapter 3

A DISTRIBUTED FIREWALL APPROACH FOR LARGE SCALE DATA CENTER NETWORK SECURITY ANALYSIS

3.1 Abstract

Cloud-based systems and services are seeing exponential growth in the last few years. Many companies and digital services are actively migrating their storage and computational needs to the cloud. With such an expansion of virtual services, security threats are also significantly increasing. Utilizing the Attack Representation Methods (ARMs) and Attack Graph (AG) enables the security administrator to understand the cloud network's current security situation. However, the AG suffers from scalability challenges. It relies on the connectivity between the services and the vulnerabilities associated with the services to allow the system administrator to realize its security state. This approach caused the AG to be vast and challenging to generate and analyze. To address the scalability challenges, this chapter introduces a segmentation-based scalable security state (S3) framework for the network. The framework utilizes the well-known *divide-and-conquer* approach to divide the large network region into smaller, manageable segments. We follow a well-known segmentation approach derived from the *K-means* clustering algorithm to partition the system into segments based on the similarity between the services. A distributed firewall (DFW) separates the segments to ensure the attacker cannot move laterally and compromise them. Our evaluation shows that the separation of segments not only preserves the original reachability and connectivity but also enhances the scalability of the AG. The presented framework (a) provides a scalable attack graph

generation algorithm by reducing attack graph generation time and density, which in turn reduces the complexity of security analysis on an extensive cloud network, (b) ensures a loop-free attack graph through the utilization of cycle detection and removal algorithm, and (c) presents an approach to provide the optimal number of segments based on the cost of implementing the segmentation using the distributed firewall rules.

3.2 Introduction

Understanding any computerized system's security situation requires a detailed analysis of the existing vulnerabilities and potential threats. According to an article published in Forbes [50], by the year 2025, 80% of enterprise IT infrastructure will be migrated into the cloud. The same article points out that the most significant concern for IT administrators is the security and privacy of cloud data. The amount of IT companies using Amazon Web Services (*AWS)* have been steadily increasing. An average data center in *AWS* consists of around 50,000 to 80,000 servers [76]. These huge servers, virtual machines, and virtual services require a significant effort to develop and maintain robust security analysis methods. Applying appropriate security solutions require the administrator to know the current security situations, the vulnerabilities in the running services, and how those vulnerabilities can create dependencies to allow an adversary to exploit the system. Specifically, administrators better understand the security situation by identifying dependencies between security vulnerabilities. Moreover, the administrator should know which attack path is the most critical and should be fixed before the others.

One of the well-known approaches to evaluating the network's current security situation is the attack graph (AG). AG allows the system administrator to analyze the connectivity relationship between network hosts, dependencies between the services, and the services' vulnerabilities. AG provides a framework to formally model the network attacks, as Husak *et al.* [69] discussed. The network administrator can utilize AG to analyze the possible attack paths for an attacker and the corresponding attack goals. The attack graph nodes, which represent conditions necessary for exploiting a vulnerability, also known as *pre-conditions*. The privileges obtained by an attacker from the pre-conditions are described in the form of attack graph nodes, known as *post-conditions*. In effect, the administrator can utilize the information from AG to make an informed decision for securing a network and selecting an appropriate countermeasure [36].

There are various types of AGs. According to *Hong et al.* [64], AG categories can be identified as: (1) Exploit Dependency Graph [111], (2) Topological Vulnerability Analysis [73], (3) Logical Attack Graphs [114], (4) Bayesian Attack Graph [96], (5) Multiple Pre-requisites Attack Graphs [71], (6) Conservative Attack Graph [162], (7) Compromise Attack Graph [103], (8) Hierarchical Attack Graph [150], (9) Countermeasure Attack Graph [11], and (10) Attack Scenario Graph [2]. Each of these works present a different modeling technique for AG generation and security analysis. The focus of this dissertation is the utilization of an exploit-dependency graph [111] for security analysis. In this research work, we analyze the relationship between vulnerabilities across the services (pre-conditions), and consequence of exploiting vulnerabilities (post-conditions). An exploit dependency graph is more fitting to our threat model, hence the attack graph formalism in our research work is based on exploit-dependency graph. We aim to provide an approach to generate a scalable graph.

Although the AG is used for security state analysis, its scalability issue concerns many researchers [130, 70, 155]. An early approach by Amman *et al.* [6] achieved AG generation with the scalability of the order $O(N^6)$. MulVAL [115], a well-known attack graph tool, reduces the AG generation and analysis complexity from $O(N^6)$ to $O(N^2) - O(N^3)$, where N is the number of network hosts. However, these AG complexities are not practical, and the utilization of AG based on these solutions will make the final graph output extremely huge and difficult to analyze by the system administrator. Furthermore, AG is used for graphical security analysis and used as a modeling tool for many security analysis applications. For example, game theory-based solutions utilize the AG to analyze the possible implementations of defense systems when the attacker is changing their behavior [133].

This chapter presents a distributed firewall (DFW) based micro-segmentation [26, 67] approach that provides a *scalable* AG solution for security state monitoring and analysis. By utilizing Software Defined Networking *(SDN)* as the base networking architecture for the system, we create a cloud-based system to model a network's security state. Our segmentation algorithm partitions the cloud network into smaller regions using the well-known *divide-and-conquer* approach [43]. Our framework constructs a sub-attack graph (sub-AG) for each segment after getting all the vulnerabilities and connectivity information. The exploit-dependencies between sub-AGs can also lead to cycles across sub-AGs, as Homer *et al.* [62] illustrated. The resulting sub-AGs are analyzed for cycle presence at both the intra-graph and the inter-graph levels to address the cyclic dependencies between network services' vulnerabilities. Lastly, the sub-AGs are merged according to the *DFW rules* that may allow connectivity between segments.

Our approach utilizes a segmentation heuristic, which is equivalent to clustering, to enhance the AG generation process. The segmentation divides the services among segments based on the similarity measure and study the cost of implementing the segmentation algorithm taking into account the number of *DFW rules.* This approach will help us identify the optimal number of segments and reduce the segmentation cost. We further address two important problems (i) the impact of loops in the attack graph generation - we provide a depth-first search-based solution to detect and remove loops (Section 3.7.6). (ii) the effects of frequent updates in the network topology. We use empirical evaluation to show that the attack graph generation algorithm converges quickly, even in the case of a dynamic network (new services added frequently) (Section 3.7.3).

3.2.1 Chapter Contributions

The contributions of this chapter are summarized as follows:

- We present our framework, namely *S3 (Scalable Security State).* S3 utilizes a new segmentation algorithm that is derived from the well-known K-means clustering algorithm. S3 provides segmentation based on the similarity between the services and considers the cost of separating the services using *distributed firewall* (DFW) rules. We measure the cost of the segmentation using the Bayesian Information Criterion (BIC) [120] that adds a penalty for using the DFW rules to separate the generated segments. We utilize the BIC to get the optimum number of segments that minimize the cost of segmentation and DFW rules.

- Based on the complexity analysis of the proposed algorithm, S3 scales well in comparison to prior works addressing the problem of attack graph generation [115, 65, 107]. Our segmentation algorithm achieves a complexity $O((\frac{N}{K})^2)$, where N is the total number of vulnerable services and K is the number of established segments. The proposed algorithm in this work shows how the system is segmented by distributing the vulnerable services into multiple segments to achieve a scalable state for security analysis and evaluations. The essential technique used in our work is divide-and-conquer, where we distribute the AG generation into multiple sub-graphs. We combine all the small graphs to get the final output. Moreover, our empirical evaluation shows a significant reduction in constructing AG for many services compared to existing research works.

- To ensure that the attack graph is loop-free, we utilize an algorithm for cycle detection and removal - Algorithm 5, that traverses the graph and checks for the cases where an attacker can use an existing post-condition in one segment and exploit a node in one of the previous segments in a monotonic attack path (pre-condition), which in effect leads to cycles. Our algorithm resolves all such cyclic dependencies in linear time, with the number of services present across all segments, as shown by the experimental results in Section 3.7.6.

The rest of the chapter is organized as follows; we discuss existing literature on attack graph generation and scalability in Section 3.3. The system architecture, the Attack Graph formalism and scalability challenges, are discussed in Section 3.4. Our main segmentation approach, algorithm description, optimal number of segments calculations, and cycle detection algorithm are elaborated in Sections 3.5 & 3.6, respectively. Section 3.7 provides a detailed performance evaluation of the S3 framework in terms of graph density reduction, generation time, and cycle detection time. Finally, we summarize some related topics that we could not address in this research work and conclude the chapter in Section 3.8.

3.3 Related Work

3.3.1 The Scalability of Attack Graphs

The Generation of scalable attack graphs has been a popular area of research. Amman *et. al.* [6] presents a scalable solution compared to prior research works

Table 3.1: A Summary of Related Work of AG Generation Approach and Complexity. N is the Number of AG Nodes, E is the Number of AG Edges, V is the Number of Vulnerabilities, X is the Number of Graph Branch Points, and K is the Total Number of Segments.

Research	**AG Approach**	**Complexity**
Amman *et al.* [6]	AG Monotinicity	$O(N^6)$
Ou *et al.* [114]	Logic-based AG engine with vulnerability correlation	$O(N^3)$
Wang *et al.* [148]	Probabilistic AG vulnerability correlation	Not Reported
Hong *et al.* [63]	Hierarchical AG Structure	
Kaynar and Sivrikaya [80]	Distributed AG generation using shared memory	$O(N * E)$
Mjihil *et al.* [107]	AG Decompositon	$O(\|V\| + \|E\|$
Cao *et al.* [22]	Parallel AG computation	Not Reported
Chen *et al.* [29]	Process mining to find AG branches	$O(E(N + X))$
This work	DFW-based AG generation	$O((\frac{N}{K})^2)$

[137] by assuming that attack progression in a network will be monotonic. This assumption allowed them to achieve scalability of $O(N^6)$ [64]. To mitigate the state space explosion problem, most of the existing solutions try to reduce the dependencies among vulnerabilities by using logical representation [115]. Hong *et. al.* [63] applied a hierarchical strategy to reduce the computing and analysis complexity of constructing and using AGs by grouping and dividing the connectivity of the system into hierarchical architecture. The performance time is, however, ~ 50 seconds for 50 services. Our framework generates a scalable AG of a similar scale in *2.2* seconds.

Ou *et. al.*[115] introduced a logic-based network security analyzer called MulVAL to model the interaction of software bugs with system and network configurations. It correlates system vulnerabilities to predict the possible threats and attacks in the network. Wang *et al.* [148] proposes attack-graph-based on probabilistic metrics utilized to calculate attack resistance based on probabilistic measurement obtained by combining CVSS scores. They consider a fixed probability for measuring vulnerabilities in the network where each exploits e and condition c associate with two probabilities $p(e)$ and $p(c)$ for the individual score and $P(e)$ and $P(c)$ to calculate the cumulative score. These scores can indicate possible exploit e being executed. However, this work's limitation is that the probability calculation assumes the probabilities along multiple paths leading to a node are independent, which is not valid for attack graphs.

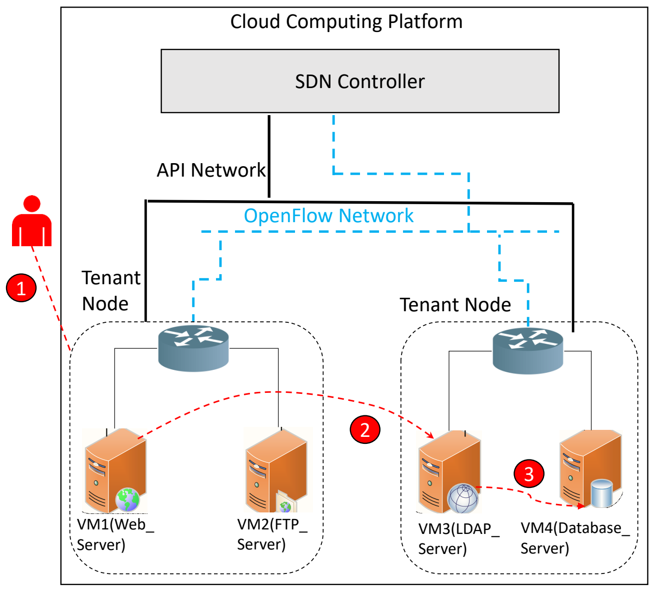
Kaynar and Sivrikaya [80] proposed a framework for distributed AG generation that utilizes a shared memory approach. However, the graph generation time is of the order *2-3 minutes* for *450 hosts*, which cannot be used for real-time security analysis. Cao *et al.* [22] proposes an approach to compute AG in parallel. Nevertheless, the division is based on each host's privileges and based on the network topology, so their approach has a different level of division and merging, which introduces greater complexity. In contrast, we utilize connectivity information and *DFW rules* to split the graph in our approach. The experimental analysis in this work shows that the required generation time for $\sim$ 500 hosts is $\sim$ *20* seconds, while in our work, it takes only $\sim$ *6.5* seconds.

Mjihil *et al.* [107] uses a parallel graph decomposition approach. The evaluation in this research work tests the effects of the number of vulnerabilities on the AG generation time, which is not reliable since it does not explain how the number of vulnerabilities is related to each service in the system, as we do in this chapter. The research work tested a maximum of 50 vulnerabilities in which they obtained an AG in $\sim$ 10 seconds, while in our work, we obtain an AG of a similar number of vulnerabilities in ~ 2 seconds.

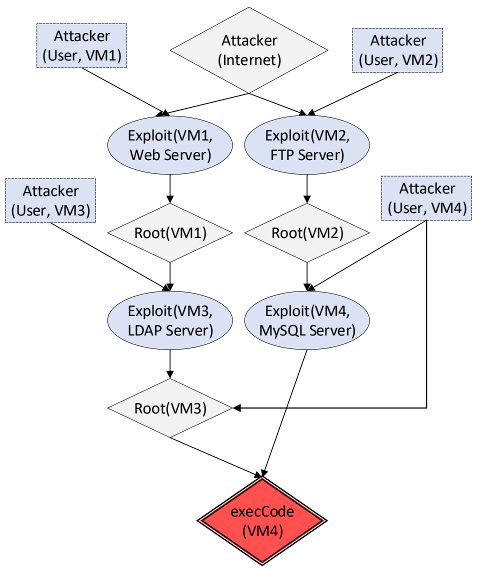
Hong et al. in [64] provides a comprehensive study to identify the usefulness of Graphical Security Models (GrSMs) in the context of modern networked systems that are typically very large and dynamic. This study is conducted based on *efficiency, application of metrics, and availability of tools.* These three significant points focus on the scalability of GrSMs, distinguishing which types of security metrics can be used, and how the user may access the GrSM in the form of tools. The authors summarize the complexity of multiple GrSMs, which are generally categorized into tree-structured, graph-structured, or hybrid-structured. This study considers different metrics to evaluate AG's types: (Dependent, Scalable, Heuristic, and Exponential). A summary of attack graph based works and thier complexity has been provided in Table 3.1.

3.3.2 Attack Graph Segmentation:

The graph segmentation solutions are classified into approximate and exact algorithms. The approximate algorithms are based on heuristic algorithms like genetic [51], ant colony [118], spectral clustering [27, 47, 134], K-L algorithms [82], and linear programming [90]. The exact algorithms can get an exact segmentation results of the graph such as branch-and-cut [21]. The problem with these graph segmentation approaches is that the time complexity increases significantly as the number of nodes and edges increases [151]. Some researchers proposed a similar approach towards the segmentation approach presented in this chapter. Chen *et al.* [29] uses a process mining algorithm to find branches of the AG. This algorithm aims to search the entire AG to find branch points where the subgraphs are split. Hence, their approach first generates the complete AG and then they establish the segments. The algorithm complexity is $O(E(N + X))$, where N is the number of vertices in the attack graph,



(a) Cloud network with multi tenant. Lateral movement attack steps to compromise Database server are shown.



(b) Attack graph with attack goal VM4 (MySQL Server)

Figure 3.1: Representation Of Vulnerability Information and Corresponding Attack Graph in a Multi-tenant Data-center Network. The Absence of Distributed Firewall *(Dfw)* Allows the Attacker to Compromise the First Tenant Node and Laterally Move Towards the Second Tenant Node to Compromise and Achieve Their Attack Goal.

X represents the number of branch points and N the number of edges. Our approach does not require generation of the entire AG. Rather, we apply the segmentation algorithm on the extracted system information and we construct the sub-AG based on that.

3.4 System Model and Architecture

In this section, we provide a motivating example of the benefit of using DFW to reduce the AG scalability problem. We then describe the system architecture of *S3*. Moreover, we provide a model for the Attack Graph and DFW, respectively.

We showed in Chapter 2 a motivational scenario (provided again in Figure 3.1

for ease of readability) about how an attacker can move laterally in the system, and how the AG can help us model this movement. We will utilize the same example to explain the system model in this chapter. The Figure 3.1 (a) shows how the lack of *DFW rules* allowed the attacker to transition between the two cloud tenant networks. Figure 3.1 (b) shows the corresponding AG for that threat model.

3.4.1 System Architecture

We consider the cloud infrastructure shown in Figure 3.2 as the architecture for the S3 framework, where the networking infrastructure is based on *Software Defined Networking* (SDN) solution. SDN is an advanced technology aiming to enhance the current networking protocols by separating the control plane from the data plane. The choice of using SDN benefits the network administrator by facilitating the programming and unifying the communication with the data plane devices. If we do not use SDN, we will need to design a controller software that can be integrated with multiple vendor devices such as CISCO and JUNIPER. Furthermore, the SDN controller can be integrated in a hybrid-network scenario as highlighted by [5]. The SDN can be used to fetch the global network topology of SDN-based devices and traditional networking devices like BGP router. The *Application Plane* comprises a *vulnerability scanner* (Nessus) that collects vulnerability information from each network host. On the other hand, the vulnerability scanner interacts with the individual hosts at the *data plane* using *API network*.

The *Security Policy Database* (SPD), as shown in Figure 3.2, creates security policies to define what security rules should be applied when implementing the DFW rules. These policies dictate how segments are separated. The traffic between segments is regulated using security policies defined by SPD. The SPD interacts with the SDN controller using *northbound REST APIs* to update *security policy* information.

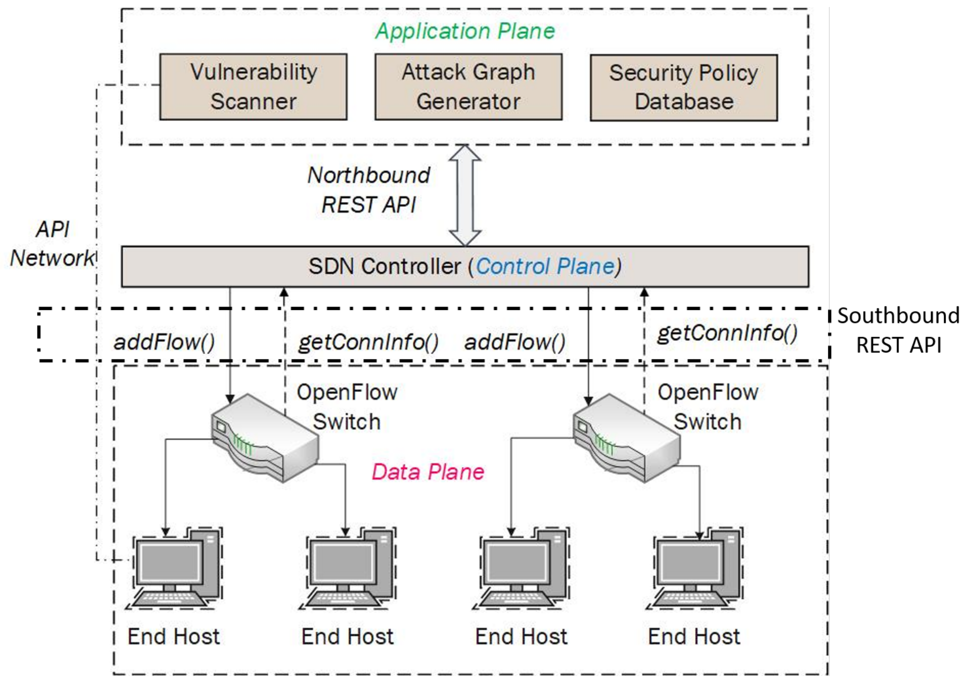
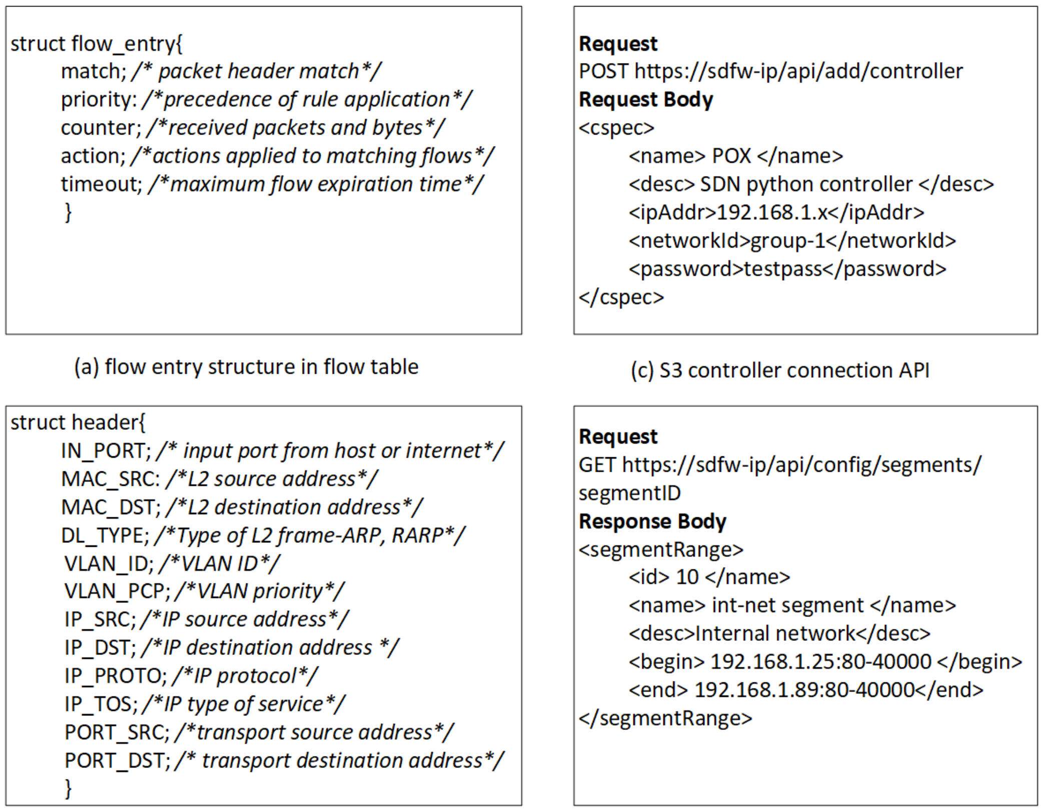


Figure 3.2: S3 System Architecture and Operating Layers.

The *Attack Graph Generator* module is a wrapper program we developed for the generation of sub-attack graphs *sub-AGs* at the level of each segment. This module interacts with a vulnerability scanner and SPD to create segments using an algorithm (proposed later), where we generate a sub-AG for each segment, and finally, utilize a *merge* algorithm to merge sub-AGs into fully connected AG.

S3 framework utilizes OpenFlow *southbound APIs* to provide flexible and programmable segmentation architecture. OpenFlow protocol is also used to communicate with the switches at the *data Plane* level. As shown in the Figure 3.2, the SDN controller (control plane) collects connection information from software switches situated at the *data plane* using *getConnInfo()* API. The security policies are implemented on each switch using *addFlow()* API as shown in the communication channel between controller and switch - Figure 3.2. Each *OpenFlow* switch has *flow tables* used to store incoming/outgoing flow rules based on the packet header match. The



```
struct flow_entry{
        match; /* packet header match*/
        priority: /*precedence of rule application*/
        counter; /*received packets and bytes*/
        action; /*actions applied to matching flows*/
        timeout; /*maximum flow expiration time*/
        }
```

(a) flow entry structure in flow table

```
Request
POST https://sdfw-ip/api/add/controller
Request Body
<cspec>
        <name> POX </name>
        <desc> SDN python controller </desc>
        <ipAddr>192.168.1.x</ipAddr>
        <networkId>group-1</networkId>
        <password>testpass</password>
</cspec>
```

(c) S3 controller connection API

```
struct header{
        IN_PORT; /* input port from host or internet*/
        MAC_SRC: /*L2 source address*/
        MAC_DST; /*L2 destination address*/
        DL_TYPE; /*Type of L2 frame-ARP, RARP*/
        VLAN_ID; /*VLAN ID*/
        VLAN_PCP; /*VLAN priority*/
        IP_SRC; /*IP source address*/
        IP_DST; /*IP destination address */
        IP_PROTO; /*IP protocol*/
        IP_TOS; /*IP type of service*/
        PORT_SRC; /*transport source address*/
        PORT_DST; /* transport destination address*/
        }
```

(b) Header structure in flow table

```
Request
GET https://sdfw-ip/api/config/segments/
segmentID
Response Body
<segmentRange>
        <id> 10 </name>
        <name> int-net segment </name>
        <desc>Internal network</desc>
        <begin> 192.168.1.25:80-40000 </begin>
        <end> 192.168.1.89:80-40000</end>
</segmentRange>
```

(d) S3 segment query API

Figure 3.3: S3 Data Structures Utilzed by a Control Plane Software (a), (B) and Application Plane Rest Api Used by Network Admin (c), (d).

rules are stored in the *Ternary Content Addressable Memory* (TCAM) format.

The Flow Table consists of other necessary fields besides *match* and *action* fields. Each flow entry has *priority*, *counter*, and timeout fields, as shown in the Figure 3.3 (a). The *header* structure - Figure 3.3 (b) of each flow is used for matching traffic against incoming traffic. The *REST API* at the application plane helps in the management of the distributed control plane. For instance, if a new controller needs to be added to the control plane, the *POST API* - Figure 3.3 (c) is used by the controller in order to announce an intent to *join* the network. The *application plane* checks the vulnerability, network topology, and reachability information periodically to update the network segments. Information about each segment can be obtained using *GET API* as shown in Figure 3.3 (d). The network segment generated by *S3* framework in this case *int-net segment*, with segment ID *10* consists of services present on ports *80-40000* on all machines in the range from *192.168.1.25* to *192.168.1.89*.

3.4.2 DFW System Model

Software-Defined Networking (SDN) [86] simplifies network management by decoupling the control plane and data plane. The SDN controller can dynamically configure multiple physical or virtual network switches. A network should be split into secure zones for comprehensive security coverage, each zone with its own security requirements. These requirements should identify which traffic can access a resource by identifying the access control list (*ACL*) policy.

Micro-segmentation is an approach that utilizes SDN capabilities to create security zones in the system. This approach is primarily utilized in a cloud-based architecture to isolate critical services and strict access to those services from unauthorized users. This can be achieved by deploying Firewall rules at the granularity of the service level. By using a *distributed firewall (DFW)* and micro-segmentation approach, we can establish segments in the network and assign services to them. The services in the threat model shown in Figure 3.1 is divided into two segments. The reachability from one segment to the other is controlled by a *DFW rule* to limit the attacker's movement in the system and apply security policy at the individual service level. This granular level of controlling reachability between vulnerable services turned out to effectively reduce the AG generation time, which we will demonstrate in section 3.7. From Figure 3.4, using *DFW* will allow the system administrator to control the reachability access from *Web Server (VM1)* and *FTP Server (VM2)* in the first segment. Likewise, in segment 2, the DFW rule will limit the access from *LDAP*

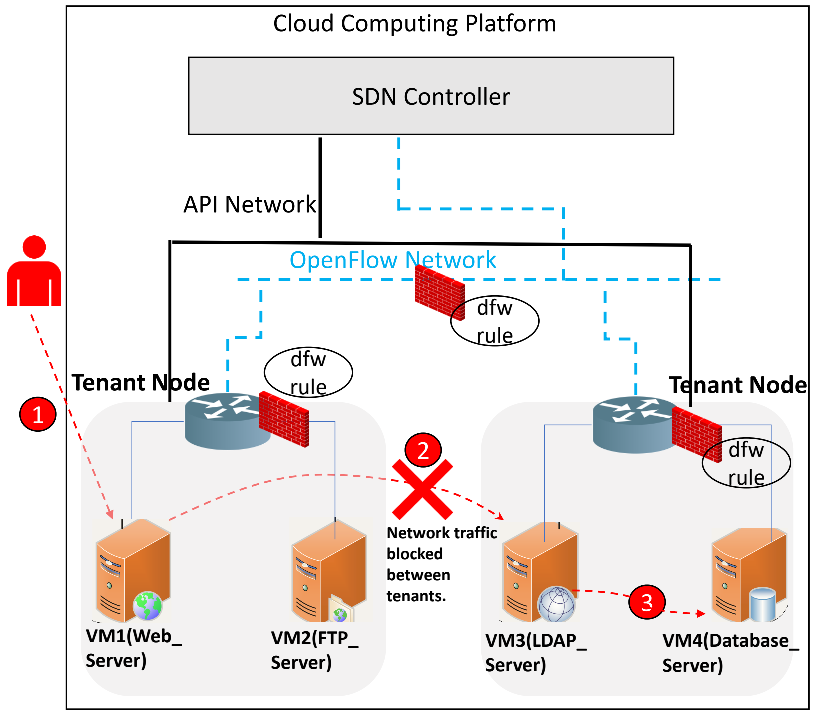


Figure 3.4: Cloud Network with Multi-tenant after Deploying *DFW Rules*. Network Traffic from Tenant Node 1 to Tenant Node 2 Is Blocked Due to Enforcing DFW Rules. The Attacker Is Now Confined to One Machine, and They Are Unable to Move Laterally in the System.

Server (VM3) to *SQL Server (VM4)* that will eventually protect the database server from the data exfiltration attack. The attacker is now unable to transition from segment 1 to segment 2, as can be seen in Figure 3.4.

Table 3.2 shows the services in Figure 3.4, as well as the vulnerability on the services and the attack path to compromise the service. If DFW is not deployed, the attacker will access any service from the internet and exploit its vulnerability, as can be seen from Figure 3.1. Figure 3.1 (b) is a simple example to illustrate how the AG looks like and how multi-step attacks can be initiated. For large systems such as data-center networks, the AG will be huge and will have a significant amount of

Table 3.2: Example of Network Topology Vulnerabilities and Connectivity Information.

Segment	VM	Service	Vulnerability	Attack Path
Segment 1	VM1	WebServer	Cross-Site Scripting	Internet- VM1 port 80
	VM2	FTPServer	Remote Code Execution	VM1 - VM2 port 25
Segment 2	VM3	LDAP Server	Local Buffer Overflow	via Segment 1
	VM4	SQL Server	SQL Injection	VM3 - VM4 port 3306

nodes and edges.

3.5 S3 Framework Segmentation Approach

In the previous section, we discussed the AG scalability issues in the absence of the segmentation approach. In this section, we provide our approach on how to achieve an efficient segmentation. This problem can be looked at as a clustering problem, where the goal is to develop the best clustering approach and get the best K clusters that correspond to optimal segmentation. In this proposal, we consider 'segments' instead of 'clusters.' There are two main requirements for our proposed segmentation process; segment compactness and segment separation. The compactness means we examine the similarity between the vulnerable services and put similar ones together in one segment. The separation means we aim to make the segments separated by *DFW rules* without increasing them. The growth in the number of *DFW rules* will have a high cost of managing the rules and will add high overhead to the administrator to ensure conflict-free rules. However, where do these DFW rules come from? After assigning the services to the segments, the benefits will still connect to services in other segments. This connectivity is now being controlled by a *DFW rule* to limit the attacker's reachability to new segments. The definition of vulnerability dependency is given by definition 2. We show Table 3.3, which has all the used symbols throughout the chapter to enhance the readability.

Definition 2 *Vulnerability Dependency is defined by the connectivity relationship between two vulnerable services. Suppose there are two vulnerable services, vs_i & vs_j, in the system. Suppose the attacker wants to exploit service vs_j. In that case, they must first exploit the vulnerable service vs_i, where vs_i & vs_j are connected via communication link C. There is no security rule to block this communication, then we say vs_i & vs_j have vulnerability dependency, and exploiting vs_i is a pre-condition to exploit vs_j.*

Consider there are N vulnerable services $VS = \{vs_1, ..., vs_N\}$. There are a total of K established segments, i.e. $P = \{p_1, ..., p_K\}$, such that the services are assigned to the segments as: $\{vs_1, .., vs_i\} \in p_1$, $\{vs_{i+1}, ..., vs_j\} \in p_2$, $\{vs_{j+1}, ..., vs_N\} \in p_K$. We define the following segmentation properties that are used as a criteria and constrains for the segmentation process.

Table 3.3: Chapter 3 Table of Notations

Symbol	Definition
C	Communication link between services
CAG	Composite attack graph
d	Number of service features
DFW	Distributed firewall
E	Edges in CAG
i	Number of iterations
K	Total number of segments
μ_{p_i}	Centroid of segment p_i
N	Nodes in CAG
p_i	A particular segment i, i $\in$ K
R	The set of DFW rule
r	Distributed firewall rule
s	A network service
s_{com}	Segment compactness
s_d	Segment separation
v	A vulnerability
VS	All the vulnerable services
vs	A vulnerable service

1. **Segment Compactness:** The compactness is defined as the relationship between entities, or points in the euclidean space being closed (having all limit points) and bounded (having all its points lie within the same amount of distance from each other). The segment compactness in this proposal is defined by

the similarity between the services in the segment, such that the distance from one service to the center of the segment is minimum. To measure the similarity, it is essential to note that the network services are considered categorical data, i.e., the services are categorized into groups like the web-server group, file exchange group, etc. The problem of measuring the similarity between categorical groups has been studied for a long time, according to [61]. In this research, we rely on the *Spearman's Correlation* similarity function to calculate the similarity between the network services. Spearman's Correlation is well known for measuring non-linear data similarities by creating a mapping function between the data and ranking the data into prospected categories. The administrator can define the categories. However, we have provided a broad set of categories covering the most popular types of services in networks. Our approach converts each network service into a vector representation, which is the pre-processing step before calculating Spearman's similarity. The vectors act as the ranking of every service to match it with one of the pre-defined categories. We consider every network service belonging to one or more of the following categories:

- Web-based category such as HTTP/HTTPS.
- File exchange category such as FTP.
- Remote access category such as ssh, telnet.
- Storage category such as SQL.
- Timing category such as NTP.
- User management category such as LDAP/AD.
- Mail server category such as IMAP/POP3.
- Other service groups.

The service is ranked from *1-8* (number of available categories) were *1* indicates the highest-ranking and *8* is the minor ranking.

To map service to one of these categories, we should take several factors such as the service's properties and the network environment. If we only consider the service's properties, we will not have a complete understanding of how this service is utilizing the network. To overcome this problem, we examine the network service's traffic to check the type of data this service provides and exchange with other services and remote hosts. For example, a *web-server* is generally used to host web applications, but it can also be used for file transfer. Suppose an *http server* is used to store files primarily and host web applications. In that case, this *HTTP server* will be ranked under the storage category first and then the web categories. Table 3.4 shows an example of ranking the services among the categories.

Table 3.4: Ranking Network Services into Different Categories.

Service	**Web**	**File Transfer**	**Storage**	**Timing**	**User Manage-ment**	**Mail**	**Remote Access**	**Other**	**Port**
HTTP	1	2	3	8	8	7	6	8	80
SQL	3	2	1	0	0	0	0	0	118
FTP	4	1	2	8	8	6	3	7	21

The *HTTP* service, for example, have the following ranking: $[1, 2, 3, 8, 8, 7, 6, 8]$ which indicates the service is ranked in the *Web servers* group firstly, and in the file transfer group secondly, and so on. This ranking is essential for Spearman's correlation coefficient to compute the similarity score between the services. The Spearman's Correlation is defined by the following equations, where x & y are the data input after converting them into a ranking format, $\bar{x}$ & $\bar{y}$ are the mean of x & y respectively, and n is the total number of features (categories) in the data:

$$SCORR(x, y) = \frac{\sum_{i=1}^{n}(x_i^r - \bar{x})(y_i^r - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{3.1}$$

Where:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \quad \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i \tag{3.2}$$

The Spearman's score is a correlation coefficient. That means if the value of the *SCORR* is 1, the services have a 'positive' relationship, or they are positively correlated. On the contrary, if the *SCORR* is -1, then the services have a 'negative' relationship, or they are negatively correlated. A value of zero means the services do not correlate with them. The positive correlations mean the two services are " moving" in the same direction in the euclidean space. The two services are providing the same type of traffic, according to the categorization they fall under. The negative correlation means that the two services are moving in the opposite direction in the euclidean space. The two services are providing different types of network traffic according to the category they fall under. Since this similarity measure is a correlation function and depends on the variance between the data, equation 3.1 will not yield a number if the variance between the data is zero (the service is equally ranked among all categories). This will mean the service does not have a variance and cannot be measured and compared with its mean or another service (or segment centroid, as we will see later).

2. **Segment Separation:** The segment separation is defined by the set of *DFW rules* ($r_{i,j} \in R$) from p_i to p_j, where $i, j \in K$. $|r_{i,\ j \ldots K}|$ is then the total number of firewall rules specified from segment p_i to all other segments $p_j \ldots p_K$. Note that

in our definition $r_{i,j} \neq r_{j,i}$ and they are directional firewall rules. For example, the firewall rule that will block the communication from the VM1 in Figure 3.4 and prevent the attacker from going to VM3, is a directional firewall rule and we will have to add another firewall rule to block the communication from VM3, to VM1. In our proposed framework, the segment separation between p_i and p_j is defined by $s_{d_{(i,j)}}$:

$$s_{d_{(i,j)}} = |r_{i,j} \cup r_{j,i}|. \tag{3.3}$$

Then, the overall segment separation s_d for all segments is defined as:

$$s_d = \sum_{i \neq j}^{R} s_{d_{(i,j)}} \tag{3.4}$$

Assuming we only have 2 *DFW rules* between segment 1 & segment 2 in Figure 3.4, then the value of $s_{d_{(p_1,p_2)}} = 2$ and the total s_d value $= 2$.

3.5.1 Segmentation Procedure

Category Extraction:

The network traffic category extraction problem has been studied extensively before. Our final goal of this step is to rank a particular network service into the prospective services' categories. For this purpose, we follow famous works in network traffic classifications presented in [152, 157]. The details of these algorithms were skipped due to space constraints. Once the categories are obtained, they are transferred to the segment initialization module for the next processing step. Note that if the number of services in the system is not large, the administrator may perform this step manually as they know the exact usage of the service.

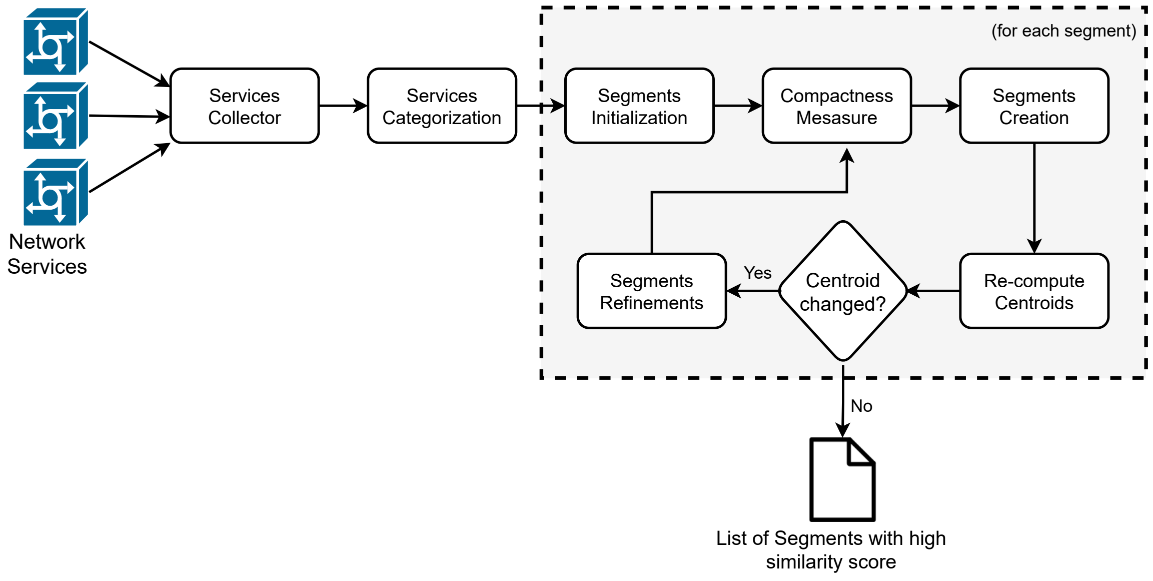


Figure 3.5: The Process of Establishing and Choosing the Best Number of K Segments Based on the Compactness Requirements.

Segments' Establishment:

To establish the segments, we follow the famous K-Means clustering algorithm approach [48]. The goal of the process is to partition the data points into K segments such that all points (services) in a given segment are close to the segment's centroid (i.e., they belong to the same segment). The algorithm keeps track of the segments' centroids and continues the rest of the iterations as follows. Our overall segmentation process is shown in Figure 3.5. After collecting the services and extracting their categories, the next step is to initialize the segments by setting the centroids, usually randomly. We do not consider completely random centroids; instead, we consider the centroids that represent each category. For example, if we pick three initial centroids, then we may choose the centroids that correspond to the commonly used services that exist in our system. For example, if the system has many *Web,FTP, and storage* services, then the corresponding ranking of the centroids should be [1.2.3.4.5.6.7.8],

$[3,1,2,4,5,6,7,8]$, and $[3,2,1,4,5,6,7,8]$, respectively. The choice of picking a centroid depends on the type of services in the given system environment, which the administrator determines.

Segments Compactness Measurement:

After establishing the segments, we measure the compactness between the centroids and the services based on Spearman's similarity measure. Find the closest centroid with the highest similarity score for each service and associate the service with this centroid. Each segment has a centroid represented by μ_{p_i}. Equation 3.1 will calculate the correlation coefficient. To get the distance between the centroids and the services, the following equation is utilized:

$$d(vs, \mu_{p_i}) = \frac{1 - SCORR(vs, \mu_{p_i})}{2} \tag{3.5}$$

The services will be assigned to the centroid with the minimum distance (minimum distance means the highest similarity). If the services are similar to the centroid, i.e., the similarity score is closer to 1, then the distance will be closer to 0. Likewise, if the similarity score is closer to -1, i.e., the services are not similar, the distance will be closer to 1. Hence, the segmentation goal is expressed using the following formula, which creates segments with services identical to each other:

Step 1: Iterate until convergence:

$$vs_i = arg\ min_{p_i} \{d(vs_i, \mu_{p_i})\} \quad \forall\ vs_i \in VS\ , p_i \in K \tag{3.6}$$

Segments Refinements:

After we establish the segments, we need to make sure that the centroids are representative of each segment, i.e., each service in the segment belongs to the correct segment. We complete this and re-estimating the centroids by taking the center of

Algorithm 1 Segments' Establishment

```
1: In: VS, $\mu_k$
2: out: Established segments with services
3: procedure SEGMENT ESTABLISHMENT
4:     Choose initial number of Segments K
5:     for $vs_j \in VS$, $\mu_i \in K$ do
6:         Compute similarity between $vs_j$ & $\mu_{i...K}$
7:         Assign $vs_j$ to nearest $\mu$
8:         Re-compute centroids $\mu_{i...K}$
9:         if Centroids $\mu_{1...K}$ do not change: then
10:            break
```

the mass points (services) associated with it. For every segment, we calculate the adjusted centroid by taking the average distance between the services. Suppose the total number of services in the segment p_i is ns_{p_i}, the new centroid becomes:

Step 2: Set each segment's centroid to the mean of all assigned services:

$$\forall p_i \in K, \quad \mu_{p_i} = \frac{1}{ns_{p_i}} \sum_{vs \in p_i} vs \tag{3.7}$$

These steps are repeated until the data converge and all the similar services are assigned to the same segment. The final output is the list of segments that contain services with a high similarity between them.

The above algorithm spans the vulnerable services *VS* and tries to group them according to the similarity measure. Assume we have a total number of d features for the services (the number of possible categories each service may belong to), a total number of i iterations. The output is K segments. Consequently, this algorithm's time complexity is bounded by $O(K * i * d * VS)$. In practice, the algorithm is fast in finding the K segments according to [8]. We show in the evaluation section 3.7.3 an experiment of how much time is required to establish the segments using the proposed segmentation algorithm. In the next section, we will investigate when the algorithm should stop and the criteria that we should consider when calculating the segmentation cost.

3.5.2 Segments Analysis

In the previous section, we explained how to establish the segments and assign services to the segments. This section analyzes the segments based on the cost function and how we choose the best *K segments* that yield the optimum cost function (minimum segment's connectivity). In the K-means clustering algorithm, the cost function is mainly defined by the average error between the points and the centroids, as shown in the equation. 3.8.

$$Cost(VS, \mu) = \sum_{i} ||vs_i - \mu_{p_i}||^2 \tag{3.8}$$

However, K-means' standard error function (usually the Euclidean distance) does not reflect the actual loss/gain from the segmentation approach we proposed in this research. Also, it does not take into account the segment separation requirement as defined in the previous section. Therefore, the cost function should consider the penalty of adding the number of *DFW rules* to isolate the segments and control the inter-segment reachability.

We utilize the Bayesian Information Criterion (BIC) function to improve the cost function and include the penalty parameter. The BIC is an index to assist in quantifying and choosing the least complex probability model among multiple options [120]. The BIC ignores the prior probability model and measures the efficiency of different models at predicting the results. The efficiency is calculated by establishing an index of each model's parameters using the likelihood function and then implementing a penalty function for models with more parameters. These parameters contribute to the model complexity. The model complexity is used to represent the penalty function and is composed of the segment separation parameter. Using the BIC, We want to know which segmentation model yields the best cost function, considering the segment separation as defined earlier. The new cost function is calculated using the following equation:

$$Cost(VS, \mu) = log\left[\frac{1}{s_d d}\sum_i ||vs_i - \mu_{p_i}||^2\right] + K\frac{log(s_d)}{s_d} \tag{3.9}$$

where d is the number of features (services categories), K is the number of segments, and s_d is the segment separation variable.

To show an example of how the segmentation approach finds the best number of segments, we show a case for a fixed number of services, but with randomly changing topology and service categories. Consider we have ten services ranked into the pre-defined categories as shown in Table 3.5:

Note the connectivity between the services in the above table. This connection represents the actual communication between the vulnerable services. It does not reflect the placed *DFW rules* that we will utilize later to limit the attacker's reachability from one node to another. Since we know the type of services in the system, we can initialize the centroids according to the type of the available services, i.e., we may initialize the centroids to represent *http, SQL, and FTP* services. This assignment of K is not finalized as we will calculate the error to determine the best K later on, according to equation 3.9. We will assume the data converges from the first iteration for simplicity, and the average of the centroids does not change. The output of the segmentation process and assignment of the services to the segments is shown in Figure 3.6. This figure shows the relationship between the services in the seg-

Table 3.5: Example of 10 Services Ranked into the Categories According to Actual Traffic the Service Provides.

Service	**Connected to**	**Ranking**	**CVSS Score**
http_1	http_3, FTP_1	{1,2,3,8,8,7,8,8}	0.75
http_2	FTP_2, SQL_2	{3,1,2,8,7,4,5,8}	0.75
http_3	http_1, Mail_1	{4,3,1,7,2,3,3,8}	0.75
http_4	SQL_1, SQL_2	{8,8,7,2,3,5,6,1}	0.75
SQL_1	FTP_2, http_4	{8,2,1,8,6,3,7,7}	0.65
SQL_2	http_2, Mail_1, LDAP_1	{8,1,2,8,3,4,5,8}	0.6
FTP_1	http_1	{3,1,2,8,8,5,4,6}	0.7
FTP_2	http_2, SQL_1	{5,3,2,6,4,1,2,8}	0.7
LDAP_1	SQL_2	{8,8,8,2,1,3,7,4}	0.7
Mail_1	SQL_2	{8,2,3,8,7,1,4,5}	0.6

ments, where the **red** lines represent the inter-segment connection and the **blue** lines indicate the intra-segment connection. These rules reflect the original connectivity between the services, as shown in Table 3.5. Moreover, a link from http_1 to http_3, for example, indicates a pre-condition to exploit the service http_3 from the service http_1. In other words, the post-condition of compromising service http_1 becomes a pre-condition to exploit service http_3. Similarly, the pre-condition to exploit Mail_1 in segment_3 is to compromise service http_3 in segment_2. Figure 3.6 also indicates that the segmentation process preserves the original connectivity between the services such that the end-to-end service delivery is maintained to the users.

The next step is to calculate the error (segmentation cost) according to equation 3.9. This equation will help us calculate the optimal number of segments. To do this,

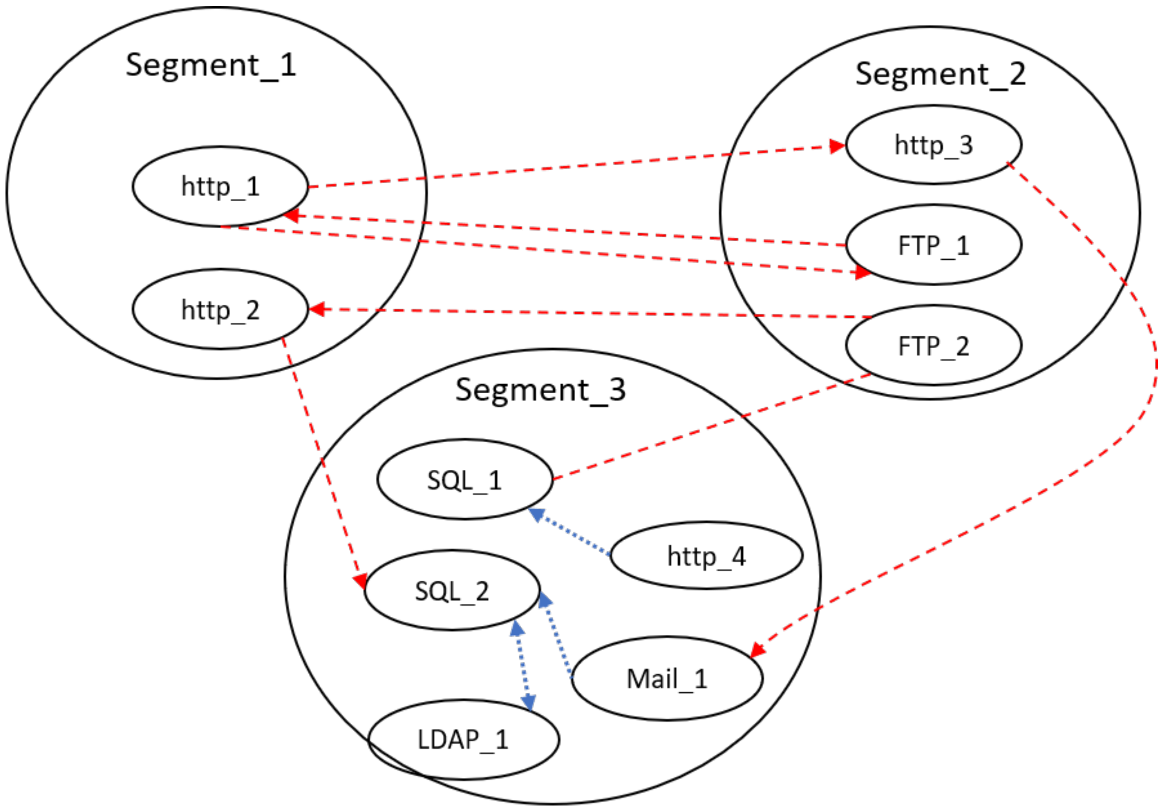


Figure 3.6: The Output of the Segmentation Process with *Dfw Rules* Highlighted in Red.

we need to examine the effect of the segment separation (*DFW rules*) variable that is calculated according to equation 3.4. To get the number of optimal segments, we test two cases as follows;

- Case 1: We perform fine-tuning of the parameters K & s_d. The cost function will result in a decreasing curve, as shown in Figure 3.7. We rely on the elbow approach to choose the best number of segments, where the error curve starts to take an elbow shape. This approach is well known in the literature for selecting the best K option. Figure 3.7 shows the testing of changing the *DFW rules* in the blue line, where the best number of K segments is four.

- Case 2: Let's examine another situation where the services have different features (categories) from the one shown in Table 3.5. The error (distance between the services and the centroid) is going to be changed, and we notice in the red line in Figure 3.7 that it is increasing for a bit before it starts to decrease and form the elbow shape. We also change the segment separation variable's value similarly to the first case. The best optimum number of segments here is six, as indicated by the red line. The two cases show that the type of services, their properties, and the *DFW rules* affect the number of optimal segments.

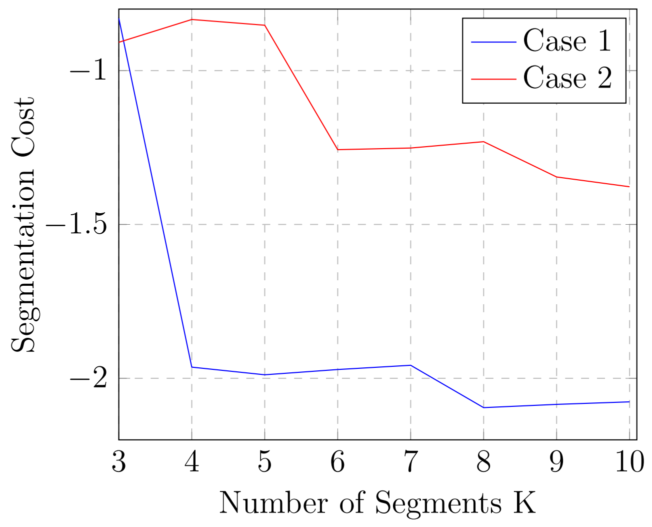


Figure 3.7: The Segmentation Cost Drops as the Number of Segments Increases until an Elbow Shape Is Created.

Special Cases:

The previously described example illustrates when the system is running with different types of services, with a similarity between them. However, we may have a system running services of the same type (all web servers, for instance) or a system running different types as such there is no relationship between the services at all. We analyze the two cases as follows:

1. System running different services:

   Consider we have a system running 2 services, an *http* server and a *mail* server. The ranking values of the *http* server will be $[1, 2, 3, 8, 8, 7, 6, 5]$ whereas the mail server will be $[6, 2, 2, 7, 8, 1, 2, 3]$. If we extend the number of services to 100, we will have a low similarity score (closer to -1). One solution to this case is to adjust the segments' initial centroids using the K-means++ algorithm [9]. The steps to choose the centroids are:

   (a) Choose one of the services at random as an initial centroid μ_{init}.

   (b) Calculate the distance $d(\mu_{init}, VS)$, from the initial centroid to all other services VS.

   (c) Choose the next centroid from the remaining data points (services) such that the probability of the remaining data points is proportional to $d(\mu_{init}, VS)^2$

   (d) Repeat the steps until all centroids have been assigned.

   The K-means++ algorithm has been proven to have a complexity of $O(logK)$ [9]. This approach will solve the problem of having services of different types and avoids creating a segment for every service individually.

2. System running similar services: Consider we have a system that runs only Web servers. In this case, the similarity measure will be one, and the distance between the services is zero. This scenario will make the segmentation process difficult as all the services will be added to the same segment based on the similarity score indicator. To overcome this issue, we can separate the connected services by placing them into different segments to minimize the connectivity between the services within the same segment. The purpose of separating the connected services is to control the reachability from one service to another by

utilizing the *DFW rule* functionality. The process of separating the services into the segment should consider the optimal number of segments and the cost of adding *DFW rules* as described earlier.

3.6 Scalable Attack Graph Generation

In the previous sections, we presented our approach for establishing segments and obtaining the optimal number of segments. In this section, we show our algorithm for generating the attack graph (AG). The process to get the scalable attack graph includes generating a sub-AG for every segment and examining the connectivity between the segments. If two or more segments are connected through the *DFW rule*, we need to merge the sub-AGs of the connected segments. The final graph is called the Composite Attack Graph (CAG), which has segments as nodes and *DFW rules* as edges connecting the segments based on the service connectivity that may lead to pre or post-conditions to exploit the vulnerabilities. Procedure *Attack Graph Generation* in Algorithm 4 shows these steps, where the input to the Algorithm is the established segments, the vulnerable services, and the list of *DFW rules* (R). After analyzing this algorithm, the complexity of generating the AG becomes $O((\frac{N}{K})^2)$ because the total AG's cost is divided by the total number of segments.

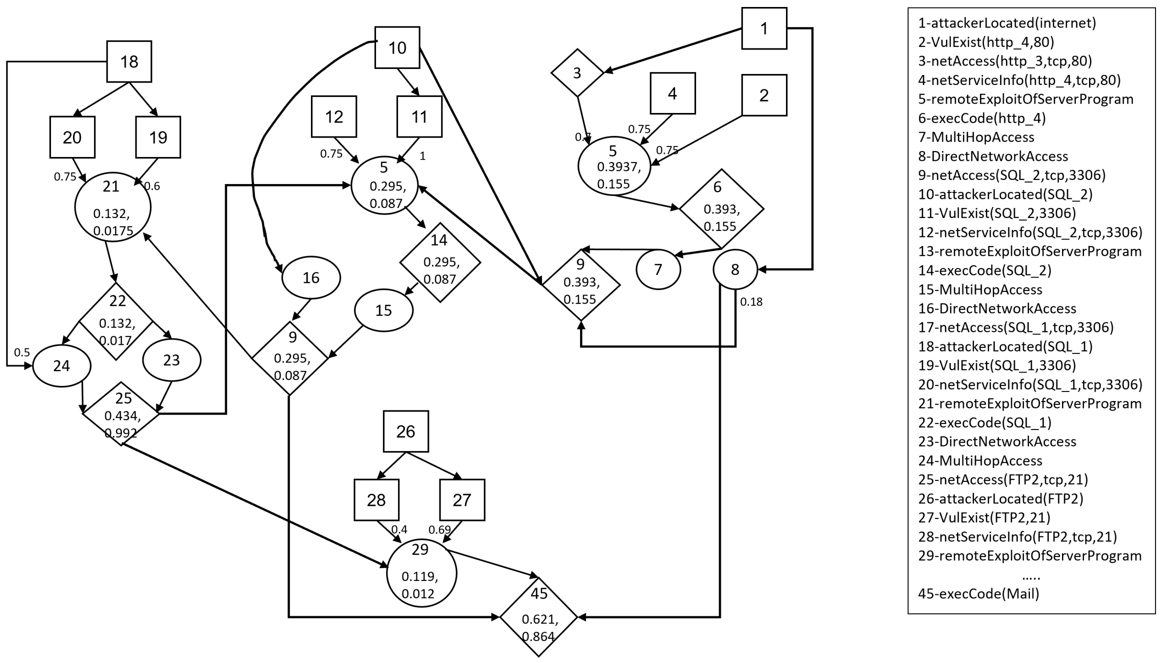
Definition 3 Composite Attack Graph *(CAG) is a tuple CAG=*$\{K, E, N\}$.

- *K denotes the set of all segments. Each segment has a sub attack graph (sub-AG), i.e., sub-AG*$_1$*, sub-AG*$_2$ $\in K$*. The sub-AG represents the vulnerable services* $(vs_1 \dots vs_N)$ *and the connectivity relationship between the vulnerable services within the same segment.*

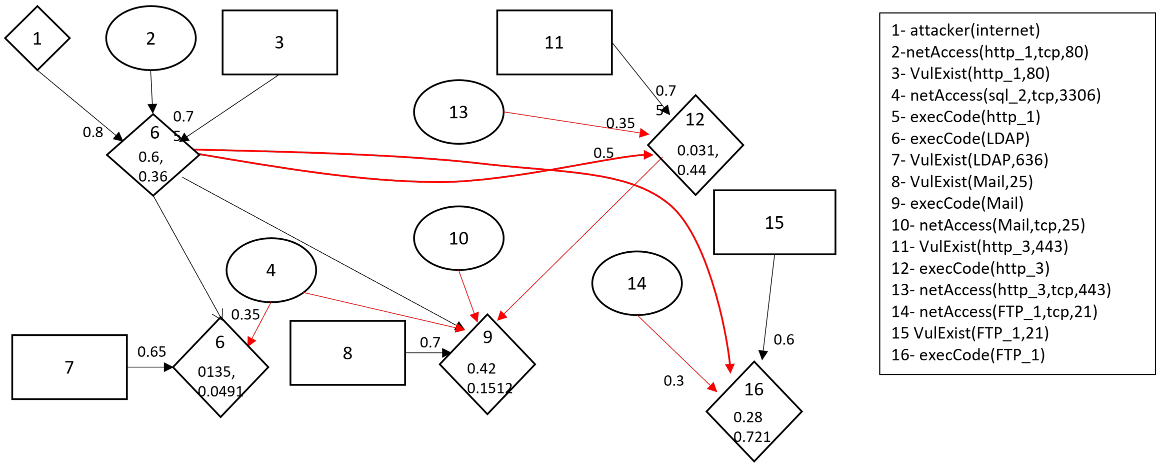
- *N denotes the set of all nodes present in the CAG. A node is an individual segment* p *that can be denoted by* N^p*. The nodes can be conjunct nodes* N_C^p, *disjunct node N_D^p or root node N_R^p. A link from segment p_i to segment p_j indicates reachability to a vulnerability in the target segment p_j. In other words, this link is the result of exploiting segment p_i, which we call post-condition that is needed for the attacker to reach and exploit p_j. Hence the post-condition from p_i becomes a pre-condition p_j.*

- *$E \subseteq E_{pre}^K \cup E_{post}^K$ is the edges present across all segments and corresponds to the DFW rules. If an edge from segment p_i creates a post-condition in segment p_j, we denote the post-condition edge using $E_{post}^{p_j} = N_C^{p_i} \times N_D^{p_j}$.*

We showed how to establish the segments and how to establish an AG for each one. Yet, there will be a connecting link from one segment to the other, as shown by the red links in Figure 3.6. This communication link can be a directional or bi-directional link. We need to establish the DFW rule that will prevent the attacker from exploiting this communication link for every case. Placing a DFW rule can also be for the directional link or bi-directional communication. For example, the administrator may only place a DFW rule to allow communication from *http_2* to *SQL_2*. Controlling such reachability at the granularity of communication between services allows us to separate the sub-AG and enhance the AG generation's scalability. We can also add one or multiple *DFW rules* for one communication link. The difference will be the direction of the communication and the specified port number of the running services. Note that a bi-directional communication link might create a cycle in the graph. To break the cycle, we need to place a *DFW rule* that manages both communication directions. To show the resulting AG of the example in Figure 3.6, We provide Figure 3.8 (a), which shows the AG for the system without considering any segmentation. It is noticed that the graph is huge and complicated to analyze. The graph has more than 40 nodes and 100 edges. To enhance the visibility of the Figure, we highlighted the Figure with colored boxes. Once our segmentation approach is applied, Figure 3.8 (b) below shows the resulting AG after using the segmentation approach. The **red** arrows indicated the placed *DFW rule* that controls the reachability from one node in one segment to the other. These firewall rules also indicate the pre-conditions for the attack to be successful and for the attacker to transition from one segment to another. It can be noticed that the provided Figure is cycle-free. The difference between the two figures is the massive number of nodes present in the top Figure due to the nodes' available reachability. By utilizing the proposed segmentation approach in this research, we can develop a scalable graph representing the vulnerability dependency between the vulnerable services and showing how an attacker can exploit one node and transition to multiple other nodes. Another advantage of reducing the AG's size is reducing the overall security risk. As the graph size decreases, the administrator will be able to effectively identify the critical attack paths based on the number of connected nodes or using other attack path identification methods and effectively securing the attack path [110, 88].


1-attackerLocated(internet)
2-VulExist(http_4,80)
3-netAccess(http_3,tcp,80)
4-netServiceInfo(http_4,tcp,80)
5-remoteExploitOfServerProgram
6-execCode(http_4)
7-MultiHopAccess
8-DirectNetworkAccess
9-netAccess(SQL_2,tcp,3306)
10-attackerLocated(SQL_2)
11-VulExist(SQL_2,3306)
12-netServiceInfo(SQL_2,tcp,3306)
13-remoteExploitOfServerProgram
14-execCode(SQL_2)
15-MultiHopAccess
16-DirectNetworkAccess
17-netAccess(SQL_1,tcp,3306)
18-attackerLocated(SQL_1)
19-VulExist(SQL_1,3306)
20-netServiceInfo(SQL_1,tcp,3306)
21-remoteExploitOfServerProgram
22-execCode(SQL_1)
23-DirectNetworkAccess
24-MultiHopAccess
25-netAccess(FTP2,tcp,21)
26-attackerLocated(FTP2)
27-VulExist(FTP2,21)
28-netServiceInfo(FTP2,tcp,21)
29-remoteExploitOfServerProgram
.....
45-execCode(Mail)

(a) Before Segmentation



(b) After Segmentation

Figure 3.8: The Output of the Attack Graph (a) Before & (B) After Applying The Segmentation Approach. The Cumulative Exploitation and Risk Probability Values Are Shown in The Graphs, Where The Segmentation Approach Reduces The Cumulative Risk Value.

3.6.1 AG Validation:

The validation of the generated AG can be achieved using a risk-based calculation that utilizes the attack paths within the AG to measure the security state. For this purpose, we utilize the work by Chung *et al.* [36] that derives probability-based scoring to compute the cumulative risk probability of each node in the AG.

The priori risk probability for the root nodes of the graph is denoted using G_V. Usually, this probability is in assigned a high probability, e.g., from 0.7 to 1. Also, in the segmented AG, the probability of the priori nodes from one segment to another has the same value range due to the complexity the *DFW rule* adds to exploit nodes between segments. For internal exploitation nodes, each attack-step node, $e \in N_C$, has a *probability of vulnerability exploitation* denoted as $G_M[e]$ that is assigned according to the *Base Score (BS)* from the *Common Vulnerability Scoring System (CVSS)* that is obtained from the NVD database [109]. For the presented vulnerable services, we provide their corresponding CVSS score in Table 3.5. The base score value ranges from 0 to 10. In our AG, each internal node is assigned a BS value divided by 10 such that:

$$G_M[e] = BS(e)/10, \ \forall e \in N_C \tag{3.10}$$

The risk probability of AG nodes are determined based on the relationships with its predecessors. This relationship is calculated using the conditional probability. We use the CVSS score to calculates the conditional probabilities as follows:

- For any attack-step node $n \in N_C$ with immediate predecessors set $W = Parent(n)$:

$$Pr(n|W) = G_M[n] \times \Pi_{p \in W} Pr(p|W) \tag{3.11}$$

  For example, nodes $2, 4$ & 3 are predecessors to node 5. Hence, $Pr(n|W)$ for node $5 = 0.3937$.

- For any privilege node $n \in N_D$ with immediate predecessors set $W = Parent(n)$:

$$Pr(n|W) = 1 - \Pi_{p \in W}(1 - Pr(p|W) \tag{3.12}$$

For example, nodes 7 & 8 are predecessors for node 9. Hence, $Pr(n|W) = 0.502$.

After calculating the conditional probabilities for all the internal nodes in the AG and CAG, the risk values from all the predecessors can be merged to get the cumulative risk probability for each nodes as follows:

- For any attack-step node $n \in N_C$ with immediate predecessors set $W = Parent(n)$:

$$Pr(n) = Pr(n|W) \times \Pi_{p \in W} Pr(p) \tag{3.13}$$

For example, nodes $2, 4$ & 3 are predecessors to node 5. Hence, $Pr(n)$ for node $5 = 0.155$.

- For any privilege node $n \in N_D$ with immediate predecessors set $W = Parent(n)$:

$$Pr(n) = 1 - \Pi_{p \in W}(1 - Pr(p)) \tag{3.14}$$

For example, nodes 7 & 8 are predecessors for node 9. Hence, $Pr(n|W) = 0.845$.

Using these equations, we show on Figure 3.8 that the total risk value in the first case is ~ 0.86 and it decreases to ~ 0.721 in the segmented graph. The reason for the reduction is due to the fact that the segmented AG has less number of attack paths due to the utilization of the *DFW rules*. The user can benefit from this approach in order to validate the output of the AG and whether or not the segmentation approach has benefited the risk assessment of the system.

3.6.2 AG Update:

In a large cloud system such as the data-center network, services are added and removed rapidly. The change in the services will require reanalyzing the AG to ensure an accurate representation of the security state. To update the AG, we first need to examine which segment this new service should include. According to the compactness measurement, this process is simply 'classifying' the service to place it in the most similar segment. We may do so by calculating the distance from the new service to all existing centroids. The nearest centroid to the new service will be chosen as a segment for the service. After classification, we examine the effects of the service on its local segment and the other segments. If the service is connected to a service within the same segment, we recompute the segment's sub-AG. If the service is connected through the *DFW rule* to other segments, we merge the sub-AGs of those segments. This ensures that we account for the local and global effect of the service on the attack graph and the system's overall security state. Procedure *Update AG* in Algorithm 4 shows how the update process occur.

AG Cycle Detection and Removal:

During AG computation and generation, a situation where a *cycle* appears in the graph, where there are multiple reasons for AG cycles. One of the most popular is the *redundant* post-conditions of vulnerabilities. According to [111, 28, 62], cycle detection and *pruning* is part of minimizing the AG size (in terms of nodes and edges) to provide a better analysis, ensure graph correctness, and most importantly, produce more *scalable* AG. Because of vulnerability dependency and post-conditions redundancy, the resulted AG may have a cycle in which the cyber adversary may transition from the current node back to a previously exploited node. More specifically, the interconnections between an organization's networking devices will lead to connectivity dependency and will not treat those connections independently [62]. In a real-life scenario, attackers will not go back to exploit a node they already visited and exploited. This will only increase the chance of getting detected by the security administrator and require additional effort, which the attacker would want to exploit other non-previously visited nodes. Consequently, the attack path that leads to exploiting a node that has already been exploited should be removed. The cycle detection and removal are shown in Algorithm 5. This algorithm checks the cycle's existence at each level of the AG generation process. Lines 31-43 check the vertices of each segment (sub-AG, line 35). The vertex is checked for cycle using the procedure *isCycleUtil* (line 1-19). The data structures *visited[]* and *recStack[]* check all the children nodes of the current node using depth-first search (DFS) algorithm. If the cycle is detected in the sub-AG, the edge causing the cycle is removed - line 13. In connections across two segments, which can lead to a cycle, the distributed firewall (DFW) checks the flow rules between two segments. If the rules have similar actions (except *DENY* rule) - lines 23 and 24, a *DENY* rule - line 25 is added to prevent such rules, creating bi-directional links, which, in effect, removes the cycles created by such inter-segment rule dependencies.

Algorithm 2 Segmentation and Scalable AG Generation.

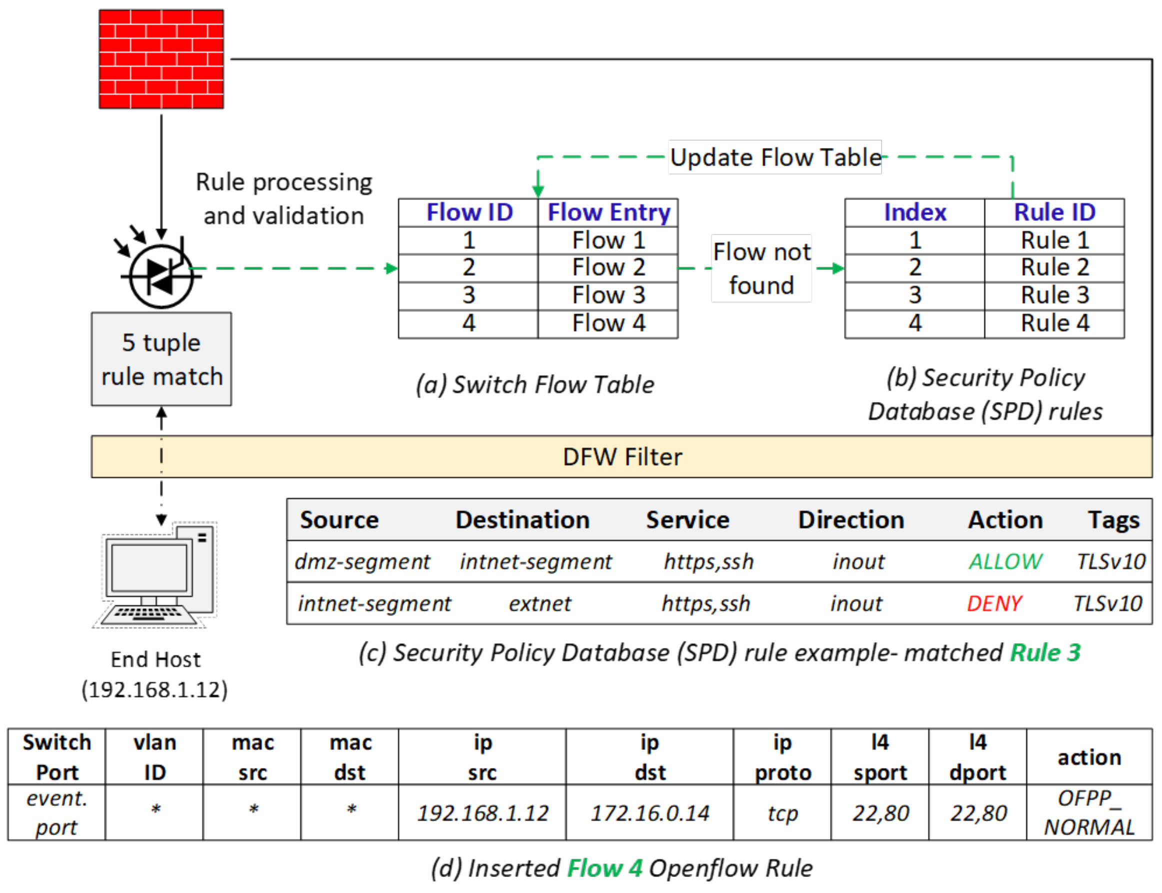
```
1: Input: Segments, VS, R
2: Output: CAG
3: procedure Attack Graph Generation
4:     for all Segments do
5:         Compute sub-AG for $segment_i$
6:         run Detect-Cycle()
7:     for all subAGs do
8:         if connected ($subAG_i$, $subAG_j$, $R$) then
9:             merge ($subAG_i$, $subAG_j$)
10: procedure Update AG
11:     $r \leftarrow$ new DFW rule
12:     $vs \leftarrow$ new vulnerable services
13:     if ($r \cap vs = \phi$) then
14:         continue
15:     else if ($vs_i$ & $vs_j \in segment_i$) then
16:         new_attack_path($vs_i, vs_j, r$)
17:     else
18:         get sub-AG of $vs_i$ & sub-AG of $vs_j$
19:         merge sub-AGs
```

Algorithm 3 Attack Graph Cycle Detection and Removal

```
1: procedure IsCycleUtil(i, visited, recStack)
2:     if (recStack[i]) then
3:         return true
4:     if (visited[i]) then
5:         return false
6:     visited[i] = true                                   ▷ Mark node visited
7:     recStack[i] = false
8:     children = getAdjList(i)
9:     for c ∈ children do                                 ▷ Loop over adj list of node
10:        if IsCycleUtil(c, visited, recStack) then
11:            removeEdge(c,i)                             ▷ Remove i from adjList of c
12:            return true                                 ▷ Back edge removed
13:    recStack[i] = false
14:    return false                                        ▷ Cycle not found
15: procedure CheckCross-Edge(dfw-rules)
16:    for $r_i$ ∈ dfw-rules do
17:        for $r_j$ ∈ dfw-rules do                        ▷ $i \neq j$
18:            if ($r_i$.act equals $r_j$.act) and ($r_i$.act=FWD or NAT) then   ▷ Checking
   for cycle
19:                if ($r_i$.src ∈ $r_j$.dst) and ($r_j$.dst ∈ $r_i$.src) then
20:                    $r_j$.append($r_j$.dst, $r_i$.src, act=DENY)
```

```
21: procedure DETECT-CYCLE (SUB-AG[], DFW-RULES[])
22:     bool visited[]
23:     bool recStack[]
24:     for s $\in$ Sub-AG[] do
25:         while i $\in$ s.V do                              ▷ Loop over Sub-AG nodes
26:             if IsCycleUtil(i, visited, recStack) then
27:                 return true                                 ▷ Cycle detected
28:         return false
29:     Check-Cross-Edge(dfw-rules)                         ▷ Cross Sub-AG edges
```

3.6.3 DFW Dynamic Traffic Match and Flow Update


Update Flow Table
Rule processing and validation
Flow not found
5 tuple rule match
(a) Switch Flow Table
(b) Security Policy Database (SPD) rules
DFW Filter
End Host (192.168.1.12)
(c) Security Policy Database (SPD) rule example- matched Rule 3
(d) Inserted Flow 4 Openflow Rule

Flow ID	Flow Entry
1	Flow 1
2	Flow 2
3	Flow 3
4	Flow 4

Index	Rule ID
1	Rule 1
2	Rule 2
3	Rule 3
4	Rule 4

Source	Destination	Service	Direction	Action	Tags
dmz-segment	*intnet-segment*	*https,ssh*	*inout*	*ALLOW*	*TLSv10*
intnet-segment	*extnet*	*https,ssh*	*inout*	*DENY*	*TLSv10*

Switch Port	vlan ID	mac src	mac dst	ip src	ip dst	ip proto	l4 sport	l4 dport	action
event. port	*	*	*	*192.168.1.12*	*172.16.0.14*	*tcp*	*22,80*	*22,80*	*OFPP_ NORMAL*

Figure 3.9: Distributed Firewall (DFW) Security Policy Rule Match and Flow Table Update. The Flow Table Is Dynamically Updated Based on Security Policy Present in Security Policy Database (SPD).

The DFW utilizes OpenFlow and REST API network to match the traffic based on five tuples, i.e., {srcip, dstip, sport, dstport, protocol}. The process of how DFW match and rule update process is shown in Figure 3.9.

- **Step 1:** The *end-host* (192.168.1.12) from *intranet-segment*, attempts to send *http* traffic to port *80* and *ssh* traffic to port *22* of host (172.16.0.14) situated in another segment *dmz-segment*.

- **Step 2:** Initially, when the flow table is checked using *table_lookup*, there is no rule present for the matching traffic rule. The flow table only has rules with *Flow ID* {*1-3*} - Figure 3.9 (a). The packet is sent to the controller using *action=OFPP_CONTROLLER*.

- **Step 3:** The controller checks the security policies defined by the *Security Policy Database* (SPD) rules present in the application plane, using northbound REST API. The traffic pattern matches the *Rule ID* {*3*} - Figure 3.9 (b), (c). The action defined in the SPD for this traffic is *ALLOW*.

- **Step 4:** The flow table is updated with a new OpenFlow rule - *Flow ID* {*4*}. The fields corresponding to layers 3,4 are updated, and layer 2 fields are wild-carded - Figure 3.9 (d). Thus, communication is enabled between two hosts. If there is no match for the traffic in either the flow table or SPD, the traffic is discarded based on white-listing policy.

3.6.4 Scalable Attack Graph Generation Cost Analysis

We consider the mapping between the physical network and virtual network shown in Figure 3.10 (a). The physical topology consists of two segments, i.e., *Segment 1* and *Segment 2*, with $VM_1, VM_2 \in Segment1$ and $VM_3 \in Segment2$. Each VM consists of several services such as apache2, MySQL, etc. The connectivity relation between the VMs is used to determine the AG for the entire network. For instance, if the firewall rules are defined between VMs and segments in a coarse-grained manner, the AG will be huge as shown in *Before DFW* case in the above Figure 3.10. According to the white-listing policy, the traffic across each segment might be limited. Whereas, if we enforce the white-listing policy at segment and service (SSH, MySQL) level as shown in Figure 3.9 (c), the attack graph generated by incorporating the traffic can be finite for security analysis. Once the DFW is enforced at different levels of the network, i.e., at the granularity of per-VM, per-segment, or an entire network, we obtain a sparse AG, as shown in *After DFW* in Figure 3.10 (a). We define the *Incidence* and *Laplacian* matrices for the attack graph G below:

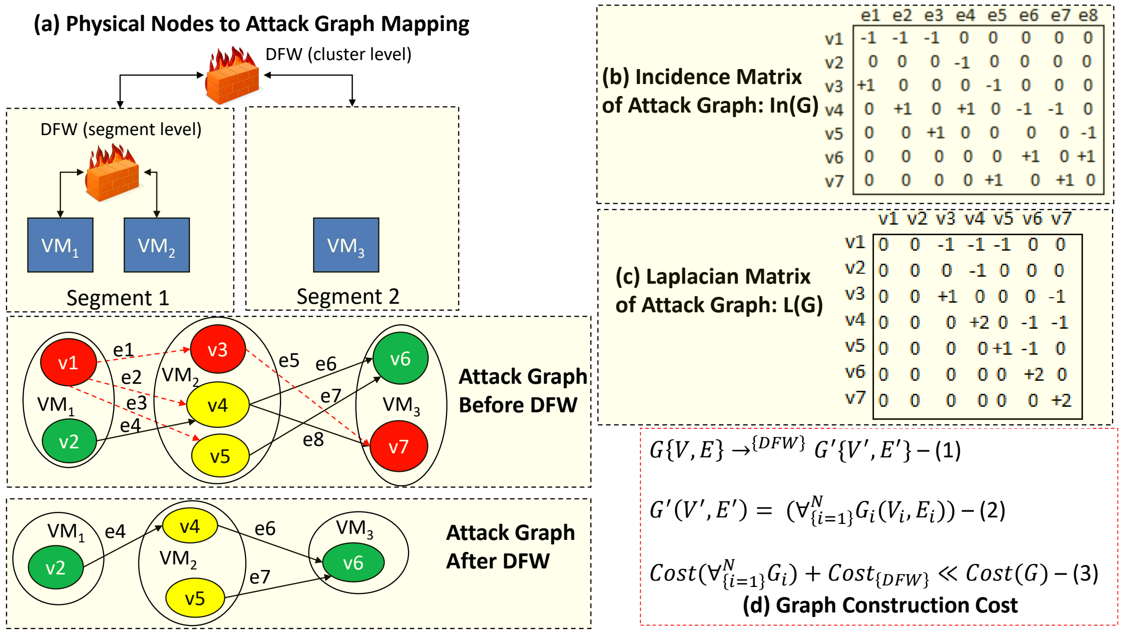
Definition 4 *Incidence Matrix: The incidence matrix In(G) of graph G{V,E} is a* $|V| \times |E|$ *matrix, as shown in the Figure 3.10 (b), with one row for each node and one column for each edge. For each edge* $e(i, j) \in E$*, column entry e of In(G) is zero, except for* i^{th} *and* j^{th} *entries, which are +1 and -1, respectively (if there is an edge from i to j, the value is +1, whereas it is -1 if there is an edge from j to i in the graph, the value is zero if there is no edge e(i,j)).*

Definition 5 *Laplacian Matrix: The Laplacian matrix L(G) of graph G{V,E} as shown in the Figure 3.10 (c), is a* $|V| \times |V|$ *symmetric matrix, with one row and column for each node. It is defined by*

- *L(G) (i,i): is the degree of node I (number of incident edges).*
- *L(G) (i,j): -1 if* $i \neq j$ *and there is an edge (i,j).*
- *L(G) (i,j): 0 otherwise.*

The application of DFW at different levels of the physical and logical network increases graph sparsity. The aggregated graph has reduced state space compared to the original AG.

3.6.5 Sparse Graph Connectivity Using DFW


(a) Physical Nodes to Attack Graph Mapping
DFW (cluster level)
DFW (segment level)
VM$_1$
VM$_2$
VM$_3$
Segment 1
Segment 2
v1
v2
v3
v4
v5
v6
v7
e1
e2
e3
e4
e5
e6
e7
e8
Attack Graph Before DFW
Attack Graph After DFW
(b) Incidence Matrix of Attack Graph: In(G)
(c) Laplacian Matrix of Attack Graph: L(G)
(d) Graph Construction Cost

$$\begin{array}{c|cccccccc} & e1 & e2 & e3 & e4 & e5 & e6 & e7 & e8 \\ \hline v1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ v2 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ v3 & +1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ v4 & 0 & +1 & 0 & +1 & 0 & -1 & -1 & 0 \\ v5 & 0 & 0 & +1 & 0 & 0 & 0 & 0 & -1 \\ v6 & 0 & 0 & 0 & 0 & 0 & +1 & 0 & +1 \\ v7 & 0 & 0 & 0 & 0 & +1 & 0 & +1 & 0 \end{array}$$

$$\begin{array}{c|ccccccc} & v1 & v2 & v3 & v4 & v5 & v6 & v7 \\ \hline v1 & 0 & 0 & -1 & -1 & -1 & 0 & 0 \\ v2 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ v3 & 0 & 0 & +1 & 0 & 0 & 0 & -1 \\ v4 & 0 & 0 & 0 & +2 & 0 & -1 & -1 \\ v5 & 0 & 0 & 0 & 0 & +1 & -1 & 0 \\ v6 & 0 & 0 & 0 & 0 & 0 & +2 & 0 \\ v7 & 0 & 0 & 0 & 0 & 0 & 0 & +2 \end{array}$$

$$G\{V,E\} \rightarrow^{\{DFW\}} G'\{V',E'\} - (1)$$

$$G'(V',E') = (\forall_{\{i=1\}}^{N} G_i(V_i,E_i)) - (2)$$

$$Cost(\forall_{\{i=1\}}^{N} G_i) + Cost_{\{DFW\}} \ll Cost(G) - (3)$$

Figure 3.10: Distributed Firewall-based Multi-level AG Generation. Vulnerabilities Have Been Color-coded Based on Their Severity. The Red Color: High-severity, Yellow Color: Medium Severity, Green Color: Low Severity. After DFW-based Optimal Segmentation, High Severity Vulnerabilities Are Blocked, as Presented in the Figure.

The incidence graph In(G) and laplacian graph L(G) have the following properties.

- L(G) is *symmetric*, i.e., *eigenvalues* of L(G) are real and its *eigenvectors* are *real* and *orthogonal*. For example, let $e = [1, ..., 1]^T$ be a *column vector*. Then $L(G) \times e = 0$.

- Matrices are independent of signs chosen for each column of In(G), $In(G) \times In(G)^T = L(G)$.

- Let $L(G) \times v = \lambda \times v$ and $\lambda \neq 0$, where v is eigenvector and λ is eigenvalue of

L(G),

$$\begin{aligned} \lambda &= ||In(G)^T - v||^2/||v||^2 \\ \lambda &= \frac{\sum_{e(i,j)\in E}(v(i)-v(j))^2}{\sum_i v(i)^2} \end{aligned} \tag{3.15}$$

- Eigenvalues of L(G) are non-negative, i.e., $0 = \lambda_1 \leq \lambda_2 ... \leq \lambda_n$.
- The number of connected components of G is equal to number of λ_i equal to 0. In particular, $\lambda_2 \neq 0$ *if & only if* G is connected.

Using the properties defined above, we check the *algebraic connectivity* of two graphs G and G', which can be compared to checking the density reduction. The graph $G'\{V', E'\}$ obtained in the case of *After DFW* scenario, is composed of sub attack graphs (sub-AGs), $G_1, G_2 ..., G_n$, i.e., $G'\{V', E'\} = \cup_{i=1}^{N} G_i$, as shown in Figure 3.10 (d). Since $G'\{V', E'\}$ is obtained from $G\{V, E\}$ after collapsing vertices and edges at different layers using a multi-level DFW, it naturally follows that G' is a subgraph of G, i.e., $G' \subseteq G$. We utilize an important corollary from spectral bisection algorithm [138], and the properties of laplacian matrix discussed in this subsection to derive the equation $\lambda_2(L(G')) \leq \lambda_2(L(G))$.

Result: $G'\{V', E'\} \subseteq G\{V, E\} \rightarrow \lambda_2(L(G')) \leq \lambda_2(L(G))$, i.e., on application of DFW, the algebraic connectivity, and in effect, density of the AG reduces. Thus, our approach, helps in creating *scalable AGs* (CAG) in a multi-tenant cloud network.

Cost Analysis: *Upper bound* on the cost can be obtained by considering that graph G{V,E} is *fully connected*, in which case, the *micro-segmentation* will not be able to achieve noticeable benefits. The cost of generating the full AG in the absence of DFW, Cost(G), is much higher than using DFW. However, the goal of *micro-segmentation* is to ensure that the graph is sparsely connected based on the white-listing approach.

Consequently, $Cost(G') = \forall_{i=1}^{N} Cost(G_i) + Cost(DFW)$ - Figure 3.10 (d) and $Cost(G') << Cost(G)$ since the effort for generation of graphs $G_1, .., G_i$ is computed in parallel with the help of SDN controller. The only additional efforts $Cost(DFW)$ is needed for checking *DFW rules*, and maintaining synchronization between different DFW agents present on individual *segments*.

3.7 Experimental Results

In this chapter, we provide evidence of the viability of our approach through a series of experiments. Our goal is to prove the scalability and effectiveness of the proposed S3 framework. We conducted the first experiment to test the number of vulnerabilities on the AG size (section 3.7.2). The second evaluation experiment is to calculate the AG generation time as the number of services increases, and the number of segments in the system also increases (Table 3.7 in section 3.7.3). As services are added and removed to a dynamic system such as the data center network, measuring the effect of adding new services is essential to understanding the security situation better. We measured the impact of the services on AG generation time and updated it according to Algorithm 4 (section 3.7.3). It is essential to check the SDN controller overhead that is induced by the AG computing. Thus, we examined this overhead to see how much the end-to-end throughput is affected when the AG is computed. Our evaluation showed that the maximum overhead does not outpace 12% (section 3.7.5).

3.7.1 Experimental Setup

We created the system shown in Figure 3.2 to implement the experiments. We utilized an OpenStack-based cloud network comprising two Dell R620 servers and two Dell R710 servers, all hosted in the data center. Each Dell server has about 128 GB of RAM and 16 core CPUs. In addition to the components in table 3.6, we used the latest version of Open vSwitch (OVS 2.13.90) as OpenFlow switches, and they are connected to containers in the data plane. The details of software components and implementation framework used for experimental analysis have been provided in Table 3.6.

Table 3.6: S3 Components Used in The Experimental Evaluation.

Component	**Version/LOC**	**Language/Framework**
SDN Controller	POX controller	Python 2.7
Vulnerability Scanner	Nessus V. 8.8.0	Attack Scripting Language (NASL).
Attack Graph Generator	MulVal V. 1.1	Datalog Modeling
Security Policy Database	500	Python & MONGO_DB
Data-Plane	Variable	Containers

3.7.2 Attack Graph Scalability Evaluation

Besides the impact of the vulnerability dependency created due to the interconnection between the vulnerable services, it is crucial to study how many vulnerabilities in the system affect the AG. In this experiment, we want to study the number of vulnerabilities on the AG scalability. To show the scalability of the S3 framework, we simulated a system with an increasing number of vulnerabilities. Figure 3.11 shows the experimental data where we emphasize the relationship between the number of vulnerabilities and the size of the resulted AG in terms of nodes and edges, where the x-axis shows the total number of vulnerabilities in the entire system. The y-axis

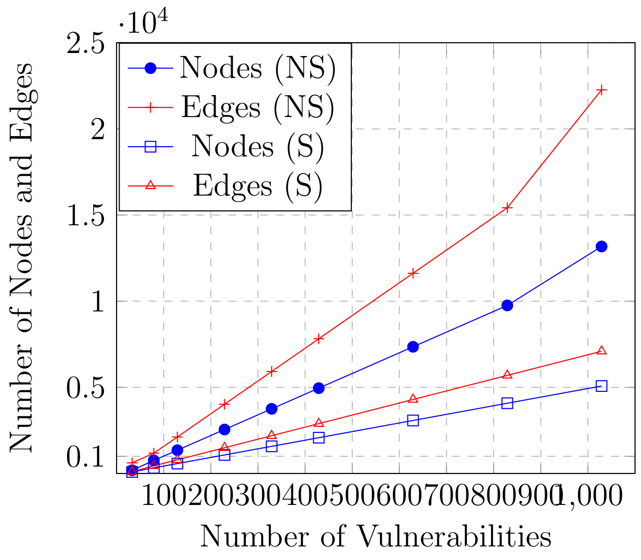


Figure 3.11: Comparison for the Number of Nodes (Blue) and Edges (Red) with No Segmentation (NS) and After Using S3 Segmentation (S).

shows the number of nodes and edges in the AG. The nodes and edges without segmentation (Nodes (NS), and Edges (NS)) in the AG are equivalent to MulVAL's [115] approach), respectively. The blue and red lines show the AG's number of nodes and edges respectively. The total number of nodes and edges before using S3, i.e., no-segmentation (NS) when the system has over 1000 vulnerabilities is about *13k* nodes and *22k* edges. This is due to the absence of the *DFW rules* affecting the reachability between the individual components in the system. After using S3, i.e., using segmentation (S) where the DFW is enforcing the exact reachability information, the number of nodes drops to about *5k*, and the number of edges is *7k*, respectively (Figure 3.11). This is a significant reduction compared to an AG without any *DFW rules*, which shows how valid the proposed micro-segmentation DFW-based approach is, especially for large cloud systems.

3.7.3 Attack Graph Generation Time and Automatic Graph Update Evaluation

The scalability of AG is measured by the graph size and the time required to generate the graph. We created several test cases to test the time required to generate the AG when we have a different number of segments and a vulnerable services in each of those segments. The process of generating the AG has two parts: a) The pre-processing step, where the segments need to be established and the based on the proposed segmentation approach and Algorithm 1. b) The AG generation step: applying the Algorithm 4 to generate the graph.

Segments Preparation and Creation

To show the effectiveness of the proposed segmentation approach, we measure the time required to establish the segments based on multiple numbers of vulnerable network services in the system. The services are assumed to belong to different categories. Recall that the segmentation complexity depends on the number of services, number of clusters, the data dimensionality (services categories), and iterations. We present the result of our experiment in Figure 3.12, where we test the time (Y-axis) required to establish the segments (in the X-axis) for a different number of services. The number of services' categories is fixed to eight, as we discussed earlier. We perform parameter tuning on the number of segments and services to measure the segmentation time. We notice that the time in all of the cases is not significant, which indicates the effectiveness of the proposed segmentation approach. The number of iterations to establish the segments ranges between 2 minimally (50-100 cases) to 11 iterations maximally (200-300 cases), with a 0.001 tolerance rate for the centroids convergence. We compare our approach with the work of Chen *et al.* [29], where they achieved a segmentation time of 8 nodes in ~ 12 seconds. Our approach generates 10 segments

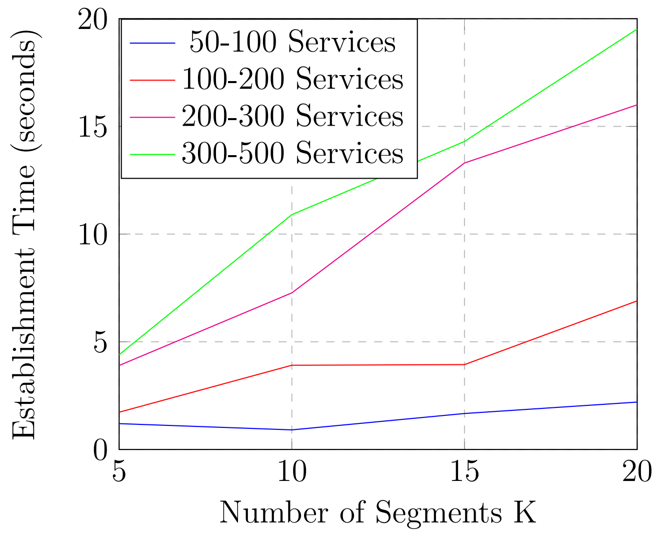


Figure 3.12: Segments Establishment Time (in Seconds) Using the S3 Approach for Different Number of Network Services and Different Segments.

for 50-100 services in less than 5 seconds. Therefore, our approach is more effective as it does not require generating an entire AG before implementing the segmentation in [29] approach.

3.7.4 AG Generation Time

After establishing the segments and assigning the services to them, the next step is to measure AG's generation time in the system. In the first test case, we are testing how much time is needed to generate an AG for a system with 50-100 services with various vulnerabilities on those services. This resulted in 5 segments, as can be seen in the experimental analysis. Moreover, we measure the graph density of the resulted AG using the formula in equation 3.16:

$$Density = \frac{\mid E \mid}{\mid V \mid (\mid V \mid -1)}, \tag{3.16}$$

where $\mid E \mid$ is the total number of edges for the AG, and $\mid V \mid$ is the total number of

Table 3.7: Sub-AG Generation Time, Graph Density, and the Number of Nodes and Edges for Each Sub-AG When the Number of Services Is Increasing.

# Services	**50-100 Services**				**100-200 Services**			
#Segments	5	10	15	20	5	10	15	20
Time (seconds)	2.22	3.88	5.925	8.22	2.386	4.93	7.2112	10.229
# Edges	6552	12186	18990	27450	14400	28494	40698	52956
# Nodes	5829	10842	16895	24420	12805	25338	36191	47092
Density	19.3E-05	10.4E-05	6.6E-05	4.6E-05	8.8E-05	4.44E-05	3.1E-05	2.4E-05
# Services	**200-300 Services**				**300-500 Services**			
#Segments	5	10	15	20	5	10	15	20
Time (seconds)	3.56	7.15	10.6	13.96	6.46	11.05	15.91	19.7
# Edges	18819	44100	63918	88065	34242	65922	93117	128580
# Nodes	18101	39210	57951	79668	32533	60698	85623	116304
Density	5.7E-05	2.9E-05	1.9E-05	1.4E-05	3.2E-05	1.8E-05	1.3E-05	9.5E-06

nodes or vertices in the AG. The results in the table show a scalable AG generation time. For instance, in the last case in Table 3.7 where the system has 300-500 services,

Table 3.8: Mean and Standard Deviation for the AG Generation Time for the Displayed Number of Segments in Table 3.7.

#Segments	5	10	15	20
Mean time (Seconds)	3.66	6.75	9.91	13.03
standard deviation	1.96	3.17	4.46	5.04

and it is divided based on the segmentation approach into 20 segments, the AG generation time is about 20 seconds, which is a reasonable time for such a large system. In Table 3.8, we show the average time for the AG generation and the standard deviation for the 5, 10, 15, and 20 segments cases, respectively. A closer look into the AG generation time and the segments' establishment time indicates that the proposed segmentation approach using the S3 framework can generate a scalable AG within a reasonable time. Furthermore, S3 proves that dividing the services into segments and applying the AG into those segments is much more efficient than establishing an AG for the system when the services are not segmented, as we show below.

AG Update Time:

To measure updating the AG when a new service is added, we implemented an experiment to test this change in the system. Table 3.9 shows the number of newly added services and the corresponding AG generation time after the update. When ten new services are added, the AG generation time takes about 3 seconds. If the number of new incoming services increases to 50 services, the AG time takes about 4 seconds. This experiment is implemented when the system has 50-100 services, and these services were present amongst 10 segments. The evaluation proves that an update to the system configuration will not impact the AG generation time, and it shows how S3 becomes interactive in a real-time fashion.

Table 3.9: The AG Generation Time After a Service Update

#New Services	AG Update Time
10	3.02
20	3.2
30	3.5
40	3.64
50	3.88

AG Generation Time without Using S3:

To prove the effectiveness of our DFW-based segmentation approach, S3, we conducted additional experiments to examine the generation time by not considering the segmentation and using a *Firewall* (centralized one) and segmentation by DFW. Table 3.10 shows the AG generation time with and without segmentation for the specified number of hosts. The vulnerable services are simulated to give the displayed number of segments. The results when using DFW are significantly better than when not using segmentation and using a centralized firewall. This is due to the absence of east/west traffic among running services, which did not specify reachability information between running services. As a result, AG is computed centrally and resulted in significant performance improvement compared to traditional generating an AG.

Table 3.10: Sub-AG Scalability Generation Time by Using a Firewall, and Both with and Without Segmentation. The Segmentation Approach Proved to Be Effective in Reducing the Generation Time of AG.

# Services	**# Segments**	**Generation Time without Segmentation (seconds)**	**Generation time using Segmentation (seconds)**
750	5	3.51	0.872
1450	10	17.344	2.083
2490	15	4980	4.468
3360	20	6720	10.027

3.7.5 SDN Controller Overhead

It is crucial to check the communication overhead induced by using our approach for AG generation and computation on the node that is computing the AG. We experimented to ensure the AG module is not inundating the computing node and to ensure the available throughput is not decreasing significantly. Our proposed Algorithm 4 computes all sub-AGs for each generated segment, and our goal is to test the effect of these operations on the communication throughput.

The evaluation of the controller overhead is conducted by testing a different number of services using the same number of segments and with a different number of segments using the same number of services. Figure 3.13 shows the first case where the system has a range of services in every case starting from 10 services to 50 services, and the number of segments in each case is 5. The total number of vulnerabilities in each case is ~ 4000 vulnerabilities. We measure the throughput using the *iperf* tool, which measures and weighs out the end-to-end bandwidth. A comparison between

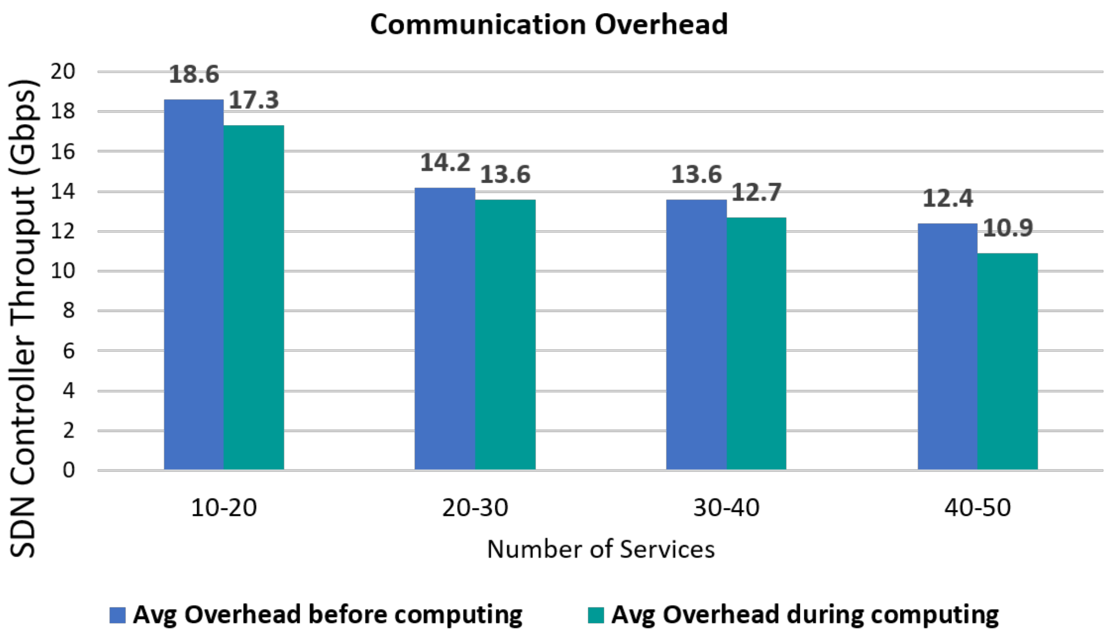


Figure 3.13: SDN Controller Overhead When the Number of Services is Different in the 5 Segments Test Case.

the communication throughput is shown in the Figure before the AG computation (the blue bar) and during the computation process (the green bar). The results are an average of *three runs*. It is noticed that the worst-case overhead does not exceed $\sim$ 12% in the *40-50* services case. Figure 3.14 shows the experiment results for changing the number of segments and fixing the number of services to *100*. We also note that the overhead induced by the AG computation does not exceed $\sim$ 12 in the *15 segments* case. The throughput's decline is justified by the overhead the AG generation module is causing and affecting the computing node. To generate the full AG, the following procedures will occur: AG computation, cycle-detection, and merging of the sub-AGs for each segment into the full AG. These experiments prove that our approach does not have a significant impact on the SDN controller in such a large-scale system.

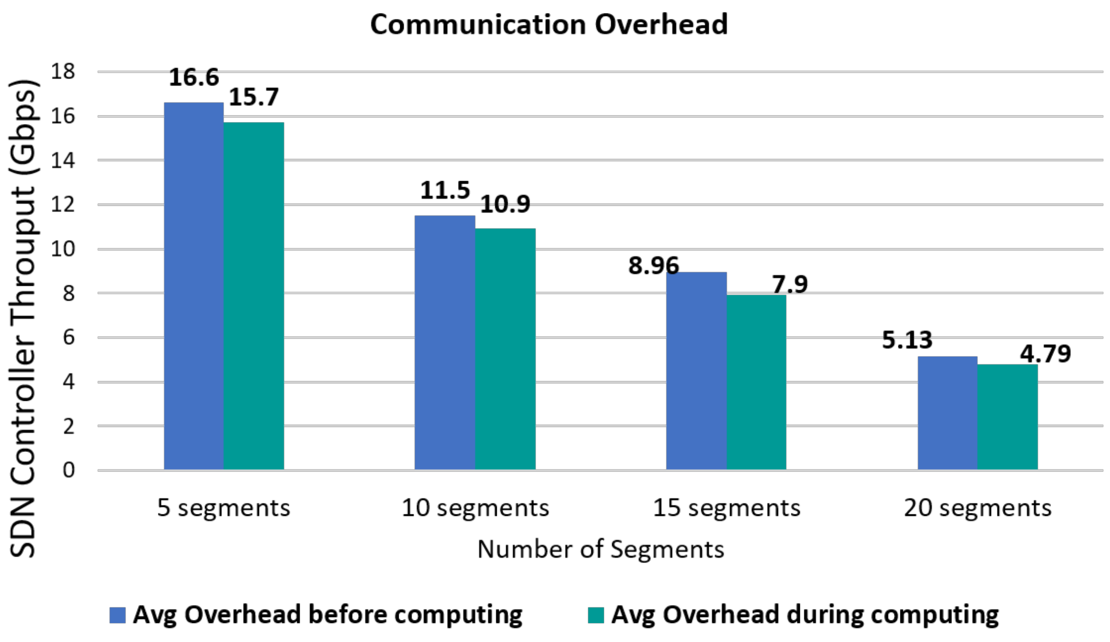


Figure 3.14: SDN Controller Overhead When the Number of Segments Is Different Using 100 Services in Each Segment.

3.7.6 Cycle Detection Time

We utilized the cycle detection and removal Algorithm 5 to experiment to check the time required to detect and remove cycles in the system using S3. The cycle detection time for 100 hosts is around 52 ms. The cycle detection time for 300 hosts is approximately 163 ms. As shown from Algorithm 4, after computing each sub-AG, the procedure of *Detect-Cycle()* is called where it traverses the output of the sub-AG and examines nodes and edges of the graph. The implementation of the cycle detection and removal is done by analyzing the *DFW rules* present on the OpenFlow switches and the information we have from the AG generation module. The experimental results show that cycle detection time scales linearly with the number of hosts. This is also in agreement with the complexity analysis for the algorithm, as discussed in Section 3.6. Research work for cycle detection in the context of the

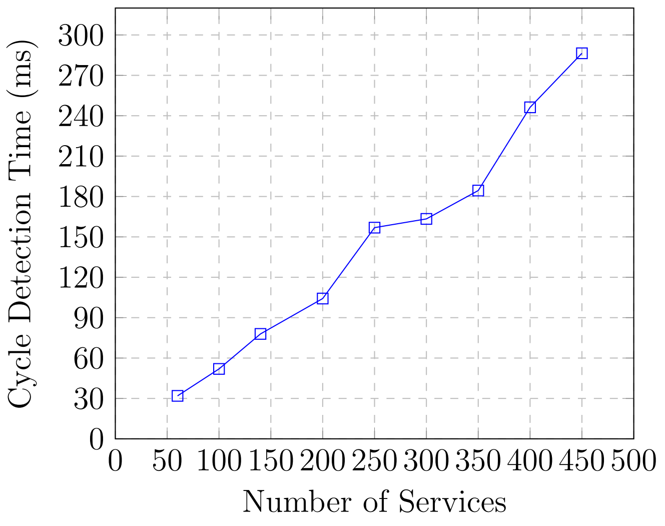


Figure 3.15: S3 Cycle Detection Time With a Linear Scaling.

attack graph, conducted by Homer *et al.* [62] was able to identify cycles in ∼150 *ms* for ten hosts and 46 vulnerabilities. In comparison, in our work, we can detect cycles for 200 services in ∼150 *ms*, which reinforces the scalability of our proposed solution is in terms of AG generation and graph correctness.

3.8 Conclusion and Discussion

The significant increase of cyber attacks and threats require robust, scalable, and efficient security analysis approaches to help system administrators defend the system. A fundamental approach to model the security dependencies in the system is through attack graphs (AGs). This chapter proposed a scalable security state framework (S3) that divides the network into smaller components and models each component's vulnerabilities by enforcing granular security policies. The proposed *microsegmentation* approach can establish scalable AG for an extensive system such as the data center network. In effect, S3 reduces the number of critical security states and the AG generation time, as shown in section 3.7.3. To present an efficient approach to obtaining the best number of segments, we proposed a segmentation algorithm derived from the K-means clustering approach. The algorithm divides the services into segments according to their similarity and separates them using the *DFW rules*.

Furthermore, we introduce a novel cycle detection Algorithm 3.4.2, which can identify and resolve cyclic dependencies between attack graph nodes. The resulting loop-free attack graph allows scalable security analysis over an extensive cloud network. In this work, we do not consider the case of policy conflict between the resulted micro-segmentation approach or the security policy database that may accidentally allow attackers to access sensitive parts of the system, even if there is no vulnerability. Also, we did not compare our segmentation algorithm with other graph-based segmentation and clustering algorithms. In the future work, we plan to investigate these problems and examine their effects on proactive cybersecurity defense mechanisms.

Segment Validation and Segmentation Heuristics: We utilized a *Segmentation Index* based sub-AG (segment), that has a validation heuristic approach. The algorithm provides information about each segment's appropriate size, such that the complexity concerns for AG generation are addressed. Still, each segment is highly *cohesive* (has the same type of services and vulnerabilities). This will help in the application of security patches to the full segment. There are other segmentation heuristics, classified under *graph clustering* algorithms, e.g., *k-spanning tree*, which creates k-groups of non-overlapping vertices, *shared nearest neighbor* (SNN) graph. We plan to compare the optimal segmentation heuristic discussed in Section 3.5.1 with other state-of-the-art graph segmentation heuristics in future work.

Attack Graph Evaluation: In this chapter, we discussed a scalable approach for generating the AG. There are other applications of AG for security analysis after it is generated. For example, AG is widely used for moving target defense (MTD) [133], a game theory-based approach deployed by the defender to increase the benefit of the security system and to make the defense mechanism proactive. Moreover, AG can be used to explore zero-day vulnerabilities if we deploy a graph neural network-based analysis model on the graph to either perform node or link prediction or graph ranking methods as discussed in [100, 98]. Security policy conflict [66, 124, 31, 34] handling, however, is another area of research that will be considered as a part of future work.

Stateful Distributed Firewall-based Segmentation: In this work, we proposed the use of stateless *DFW*. However, it is essential to account for a stateful-based firewall [32] for micro-segmentation-based security analysis. A stateful firewall should provide an understanding of intent-based policies and specifying access control list policies. Moreover, DFW scalability should be considered to achieve optimal and efficient security monitoring for system services. In the future, we plan on studying the effect of stateful distributed firewall functionality on the AG generation and how a better security scenario analysis can be achieved by considering the stateful capability to enhance the large-scale data center network.

Chapter 4

INTENT-DRIVEN SECURITY POLICY MANAGEMENT FOR SOFTWARE-DEFINED SYSTEMS

4.1 Abstract

Different network controllers are utilized in a multi-domain software-defined systems (SDx) to manage the networking resources. However, these controllers operate using a different high-level language (intent). Thus, the admin needs to perform cross-layer translation from the user requirements to the underlying network controller format, increasing human-in-the-loop overhead. There are two primary security and management challenges involved in managing multi-domain controllers. The first challenge is how to design an SDN controller language that can effectively convert human-specified networking policies at the control plane into the network flow rules level at the data plane. The second challenge is how to reduce the complexity of network flow rules conflict checking at the data plane. To address these challenges, This chapter present a new intent-based security policy enforcement solution called INTPOL. First, INTPOL provides a unified intent rules that abstracts the network admin from the underlying network controller's format. Second, INTPOL develops a networking service solution to use a bounded formal model for network service compliance checking that significantly reduces the complexity of flow rules conflicts checking at the data plane level. Finally, INTPOL is expendable from a single SDN domain to multiple SDN domains and hybrid networks by applying network service function chaining (SFC) for inter-domain policy management.

4.2 Introduction

In a Software-Defined Networking (SDN) environment, the SDN control plane manages a global network view. As pointed out in [136, 23, 60, 163, 59], in a multi-SDN domain networking environment, policy misconfigurations in middleboxes (network functions) is a common cause of middleboxes failure. The papers also provide details that 67.3% of firewalls, 63.2% of proxies, and 54.5% of intrusion detection systems (IDS) network functions fail due to misconfigurations. The networks' safety and security properties (policies) are also called *Invariants* [99, 93]. These network policies need to be verified and checked for policy conflicts such as overlapping between the flow rules' headers in a scalable fashion to ensure security, safety, and smooth functioning of the network functions. To illustrate the described problem, we present the existing network flow rule generation and management in Figure 4.1. The SDN control plane is the interface between the application plane and the data plane. The user specifies a high-level intent at the application plane that reflects their network/security requirements. The intent is defined by the user's desired security goal or business requirement [117]. Our focus in this work is on application plane intents translated into control plane policies. Thus, this chapter focuses on SDN-based networks, and the proposed framework should not be considered technology-neutral.

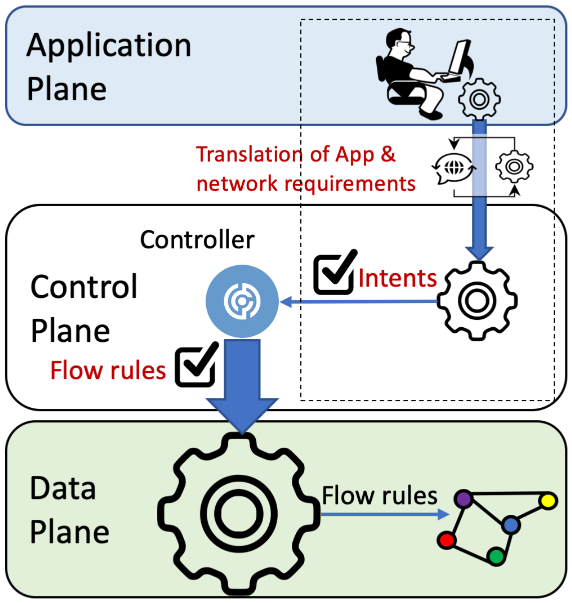


Figure 4.1: SDN Control Flow for multi-Tiered Network Policy Checking Using INT-POL Framework.

This chapter covers a broader range of conflicts without increasing the policy conflict-checking space. The reconciliation mechanism used in this research optimizes the policy space, which leads to compact policy representation. No exiting work provides a framework for policy checking at the application plane, i.e., analyzing the policy requirement when the user inserts a policy at the management interface to facilitate the data plane conflict checking. Each application plane policy can generate multiple flow rules (at the data plane) that conflict with each other. We build an abstracted bounded formal model and potential conflict-inducing queries to analyze potential conflicts before inserting the user requirements in the form of OpenFlow rules at the data plane (proactive approach of conflict checking). If the higher-level user requirements are not checked, the conflict-checking complexity for flow rules on the data plane will rise dramatically. For example, our evaluation of the Stanford topology in section 4.6.3 reveals that for every 20 created intents, there are 413 conflicting flow rules, a tenfold increase. Therefore, the fundamental issue is that if policy conflicts exist at the application plane level and are not analyzed until they get translated into flow rules in the data plane, the policy conflict checking and management overhead in the data plane increases significantly.

Our goal is to address two questions in this chapter: First, *how to design a human to SDN controller language to effectively translate human-specified networking policies into network flow-level rules.* The framework can be applied to SDN and hybrid networks that comprise SDN and traditional networking components. Second, *how to reduce the complexity of the policies and flow rules conflict checking*? We propose a novel approach to address the above problems by creating a new intent-driven policy framework, *INTPOL*. This framework allows the network administrator to express security policies at the application plane level. The policy designers can create network management and operational policies while remaining abstracted from the underlying SDN controller.

The security policies specified in the application plane are parsed for predicates of the INTPOL framework. A bounded model representation of state changes of the network packet is created using NuSMV [37] based framework, which captures the state changes of the packet as it transitions between network functions and hosts. The INTPOL intent rules translates the security policies into the REST API call format of the corresponding SDN controller. Realizing network policies at the data plane level introduces many more flow rules. Detection and resolution of conflicts amongst those flow rules at the data plane level can impact network services' performance. This problem can be dramatically amplified when expanding SDN systems into multiple SDN domains, where inter-domain networks consist of SDN and traditional routing devices.

The key technical novelty of INTPOL lies in how to ensure consistent behavior in the network, which is considered a network-wide invariant verification issue [161, 20]. Existing solutions such as the work conducted by Yuan *et al.* [158] provides a scalable formal solution for network policy verification using optimizations like model pre-computation and query containment. The issue with this research work, however, is that formal models like Linear Temporal Logic (LTL) and Computational Tree Logic (CTL) are PSPACE complete in the worst case [12], which limit the scalability of the model. The NetSMC approach [158] cannot express arbitrary path quantification scenarios, e.g., a packet will be delivered in the future, whereas INTPOL handles such policy scenarios. INTPOL models the problem of verifying network policies using a Bounded Model Checking (BMC) [40] to check for the existence of network policy violation within the bounds placed on the network, e.g., all paths up to length K. This approach ensures sufficient coverage for checking policy violation issues in the network and reduces the space complexity of network policy verification from PSPACE in the worst case to linear time (linear in the scale of the number of network states).

4.2.1 Chapter Contributions

The contributions of this chapter are summarized as follows:

- We introduce INTPOL, a new intent-based framework for translating network policy requirements into a unified format. The provided intent rules in INTPOL will abstract network administrators from underlying network controllers' policy specification semantics. The framework allows easy expression of complex scenarios such as service function chaining (SFC) (Section 4.4.5) and hybrid networks comprising traditional and open-flow networks (Section 4.4.5). The INTPOL scales well compared to existing research works in terms of policy composition (Section 4.6.2). INTPOL also utilizes bounded model checking (BMC) and application layer policy optimization (section 4.4.6) to achieve scalable policy composition and conflict checking.

- We utilize two optimization approaches (section 4.4.6) to pre-compute the application plane policies. The pre-computation of packet traversal through the SFC paths will reduce the data plane conflict checking. We demonstrate how the application plane verification and pre-processing using INTPOL will reduce the data plane conflict checking time. We utilize a large-scale network dataset, *Stanford Topology* [81], to show that conflict overhead at the application plane is minimal $\sim$ 250-800x times lower than policy conflict checking at the data plane as described in section 4.6.3.

The rest of the chapter is organized as follows: section 4.3 discusses the related work to INTPOL, focusing on intent-based policy configuration and policy conflict checking. We present INTPOL system model and the intent management of different network functions in section 4.4. The implementation details of INTPOL are presented in section 4.5. We evaluate the INTPOL framework and conduct performance evaluation in section 4.6. Finally, we conclude the chapter and discuss future work in section 4.7.

4.3 Related Work

4.3.1 Intent-based Policy Configuration

Several research works introduced intent-based policy expression and management approaches for network policy conflict. PGA [125] proposed the concept of end-point-group (EPG) to abstract the policy specification from the underlying physical infrastructure. A high-level tagging is used to define the endpoints instead of low-level addresses such as IP addresses. This abstraction approach exploits the graphs feature to resolve and detect policy conflicts. Also, PGA analyzes the Access Control

List (ACL) policies and converts them into a graph structure to find conflicts between ACL policies. The graph structure input is the possible communication between the network endpoints and the required service function chain for every communication. The policy checking method for data plane flow rules presented in this chapter covers a broader range of conflicts. In contrast, the examples presented in the PGA paper (the load balancer & byte counter) cover one class of conflicts, the polymorphism type (we emphasize the different conflicts later in the chapter). Furthermore, the PGA approach does not include cases that include security policies with the same actions and matching packet header as we do in the presented conflict-checking approach. The reconciliation of these redundant rules can optimize the speed of the end-to-end packet processing, unlike PGA's approach that increases the policy checking space.

JANUS [1] builds upon the policy graph abstraction (PGA) framework proposed by Prakash *et al.* [125], which represents dynamic temporal policies and QoS policies in an intent-based language. Janus also aims to maximize the number of configured policies by utilizing heuristic algorithms.

Han *et al.* [58] presented a framework for providing an interface to add intents by the user and translate them into network policy. The authors do not consider the scenario of multiple controllers running in the environment, nor do they consider the added intents' conflicts. Jacobs *et al.* [72] discussed how AI can be utilized to allow networks to be more intelligent. They showcased how intent-based networking (IBN) can translate high-level policies without the overhead of translating them into network flows. Most existing works lack a framework that can interpret diverse network requirements and perform end-to-end network property verification using a unified language.

We compared the scope of policy composition approach used in INTPOL with similar network modeling research works such as Frentic [54], Pyretic [127], and Covisor [75]. These works do not efficiently model hybrid environments (mixture of SDN and non-SDN environments). Covisor uses a clustering mechanism, which violates the least privilege security requirement principle. The controller in one domain can access the resources of another domain. INTPOL uses a unified model for ensuring resource isolation in a hybrid environment (see Section 4.4.5). Furthermore, INTPOL provides a scalable formal approach to allow network operators to specify network intents at the application plane. This abstracts the network operator from the policy configuration details for each network controller.

4.3.2 Network Policy Conflict Checking

Several existing solutions for network policy checking and invariant verification, such as [84, 14, 119, 139, 72, 7]. One of the earlier research work for network invariant verification is Veriflow [84]. Veriflow aims to check the network invariant in real-time with the change in the network state. Thus, Veriflow only checks the packet reachability (as one type of invariant) between two machines after the flow has been deployed [14].

Stateful network function verification has been studied by researchers as well in [147, 158]. The stateful network function verification goal is to enhance network invariant property checking by incorporating stateful network functions such as the stateful firewall. NetSMC [158] uses an existential first-order logic and query containment to provide stateful network verification. NetSMC framework uses image pre-computation to help provide a scalable formal model. The formal model for network invariant verification suffers from the number of verification states' scalability challenges. We utilize a bounded model checking approach to address the scalability limitation inherent in existing research works and design INTPOL to scale linearly in the number of model states. Additionally, the authors described how the NetSMC model is limited since it cannot handle arbitrary path quantification, e.g., in some future time packet from host h1 to h2 is delivered. As described in the next section, we use temporal logic to quantify policies for such scenarios. The authors in Epinoia [147] use bit-vector encoding and path segmentation for optimizing the intent checking procedure. We utilized bounded model checking to optimize the network verification. Additionally, we utilized policy conflict checking at the two-levels application plane and data plane to optimize the performance of conflict checking. Another key difference between INTPOL and Epinoia is that Epinoia utilized a traditional network setup (Internet Topology Zoo) for evaluating its reachability. In contrast, we have assessed a hybrid network setup containing both SDN and conventional network components. In addition, INTPOL uses LTL for expressing network behavior since complex network functions can be expressed using network invariant provided by LTL, compared to the limited expressive capability afforded by SMT solvers used in Epinoia. One main difference between LTL and SMT solvers is that the value of an LTL formula at a particular time instant is logically related to previous time instants [79]. This models the network behavior more closely since the network packets' behavior is highly dependent on all the network functions that apply to incoming network traffic. A recent study by Varadharajan *et al.* [140] proposed a new application that handles the policies between different SDN domains to address multi-domain SDN policy management. The paper uses a policy handle and policy token. The handle shows the visited autonomous system (AS) by the flow and the packets. The evaluation of the paper offers efficient network communication in terms of throughput and delays. However, one key feature of that work is missing: verifying the policies effectively and ensuring end-to-end reachability without conflict between the policies, as we do in this chapter.

4.4 INTPOL System and Model Descriptions

In this section, we first describe the design of the INTPOL framework. We then follow with examples to cover different security policies like Firewall and SFC. Next, we describe the INTPOL scalable policy checking framework; Furthermore, we present the intent conflict checking at various SDN infrastructure levels, showing how the presented framework can easily be expanded to hybrid environments (containing both SDN and traditional network elements). Finally, We discuss Service Function Chaining (SFC) as an example to explain how INTPOL can handle multiple network functions and complex packet processing scenarios.

4.4.1 INTPOL Framework

Intent refers to application plane policies used for expressing specific actions for a target network. The intents can be used for expressing routing, packet forwarding, and monitoring decisions in a network. The purpose of employing an intent is to achieve a certain business goal that the user has expressed. The intents are compiled into network policies, e.g., routing intent can be compiled into routing policies, endpoint intent can be compiled into several endpoint policies, each of which covers partial flow space of the intent [91, 99]. The intent-based networking developed primarily for the self-driving network such as SDN networks [117]. The ONOS SDN controller is one of the famous examples that provides intent-based networking [10].

We present Figure 4.2 that shows the end-to-end handling of network intents at the application plane. The network operator specifies some intents at the application plane, as can be seen from the block *Network Intents* in Figure 4.2.

- Traffic is allowed between hosts h2 and h3 (h2 $\triangleright$ h3).
- Traffic is allowed between hosts h3 and h4 (h2 $\triangleright$ h4).

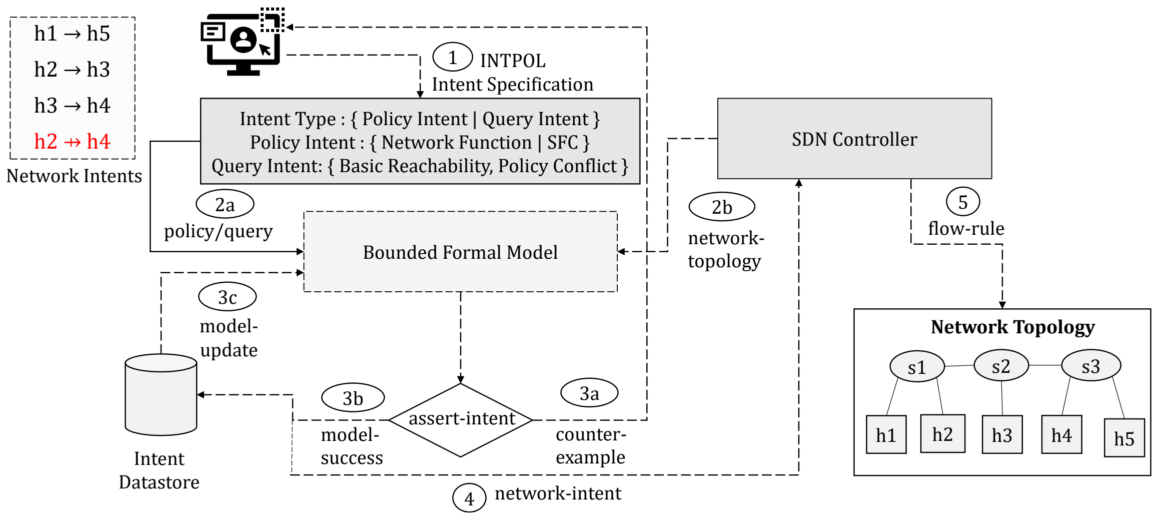


Figure 4.2: Intent Specification and Formal Modeling. The Users Can Specify Policy and Query Intents at the Application Plane.

- Traffic originating from host h2 should never reach host h4 (h2 $\not\to$ h4).

On the one hand, The *Policy Intent* is used to perform network updates like inserting a rule to block particular network traffic (Firewall) or inspecting suspicious traffic (Intrusion Prevention System). On the other hand, the *Query Intent* can help the network operator assert the safety, auditing access control, and service availability in the network. When the user specifies a network intent (1), the intent is converted into a formal logic formula (2a). The Bounded Formal Model utilizes the network topology information extracted from the SDN controller (2b). The intents present in the Intent Datastore are fetched to construct/update a bounded Linear Temporal Logic (LTL) model.

The model is evaluated for some basic safety properties, such as packet reachability, which conflicts with the existing intents in an automated fashion. Suppose the intent violates any network safety properties. In that case, the user will respond with a counterexample from the model to show which state(s) in the network are violating the network properties (3a). If the intent does not violate any network safety or reachability properties, it is stored in the Intent Datastore (3b), and the bounded model is updated with a new intent (3c). The intent is passed to the SDN controller (4) to create an SDN controller-specific intent [122, 10]. The SDN controller installs flow rule (5) to realize the user intent at the data plane level. Due to space limitations, we refer the reader to the following reference for more details on how the SDN controller establishes flow rules from the intents [126, 42, 10].

Definition 6 ***Security Policy*** *is a rule from the ruleset of the entire network R, i.e., $r_i \subseteq R$, where $R = \{\forall_{i=1}^{m} r_i\}$. Furthermore, each rule r_i, can be decomposed based on packet match condition, and actions $r_i = \{m_i, a_i\}$.*

The match field of the packet header's values can be further classified into individual headers that are part of the packet. The packet match m_i consists of physical port of incoming traffic δ_i, source and destination hardware address, α_{si}, α_{di}, source and destination IP address, β_{si}, β_{di}, source and destination port addresses, γ_{si}, γ_{di}, protocol δ_i, priority value ζ_i, for a given virtual network function. For instance, a stateless Firewall (iptables) allows assignment of rule priority besides the packet header entries.

Consider the policy $\{h1 \rightarrow h5, ALLOW\}$ defined at the application plane. We have the source and destination addresses β_{si} = h1.ip, β_{di} = h5.ip for the rule r_i, and $a_i = ALLOW$. Similarly, we can use the application layer intents to define other policies such that we specify the layers 2-4 source and destination addresses and action to allow or deny the network traffic.

4.4.2 Policy Conflict Detection

Existing intent-based networking approaches [74, 122] require the network admin to do a cross-layer translation from the high level-intents and network requirements into the appropriate controller's format. Based on the translation results, the controller can then generate the underlying network flow rules. However, the translated intents or the generated flow rules can have policy conflict [124, 66]. The policy conflict can occur for different reasons, such as a) policy inconsistencies due to service function chaining processing where multiple flow tables handling the same flow(s) might have conflicting actions; b) SDN controllers or virtual private network (VPN) implementations that modify header content could result in flow rules being inadvertently being applied to specific flows; c) flow rules injected by different applications using the northbound APIs (between the control and application planes) can have conflicting actions for the same flow; d) matching on different OSI layer addresses resulting in different actions; and e) administrator error where they insert or update a flow rule that conflict with the existing flow rules in the data plane flow table(s). The error can be because of the address space overlap between the new and the old flow rules, or it can be a new action for the flow rules, such as allowing access to the database server from the public network. In contrast, the previously installed flow rule prevents such action. While this list is not exhaustive, it demonstrates how common policy conflicts may be in SDN-based cloud infrastructures. The current research works [124, 66] primarily focus on flow-rule level conflict checking to ensure that the requested network resources allocation does not compromise existing flow and security policies. Moreover, the class of policy conflicts checked in existing research works, PGA & JANUS [125, 1], only covers rules with matching header and distinctive actions.

Policy Conflict Detection in INTPOL

We introduce two-level conflict analysis in this research work. The conflict checking at the application plane using a bounded model checking (BMC) approach results in a significant reduction of policy conflict detection time at the data plane. When an intent is added at the application plane level, and policy optimizations are performed to eliminate redundant policies, it is translated into the controller-specific intent command. The SDN controller, e.g., ONOS [16], allows specifying a *HostToHost* intent, which allows communication between network hosts, and *PointToPoint* intent which allows traffic to pass through two switches' ports. If we consider a host intent, e.g., *add-host-intent h1 h5*, based on the example network in Figure 4.2, ONOS controller *a)* identifies the path between hosts, i.e., h1-s1-s2-s3-s5, and *b)* generates Openflow rules to establish communication along the path. Additionally, the controller marks *appId* in the flow rule as *org.onos.intent* to show that the flow rule was added using the intent module. It is noteworthy that there will be five flow rules installed in Figure 4.2 for a single host intent. If there are conflicts amongst intents at the application plane, it will, in turn, generate flow rules which will conflict with each other.

```
Intent-Type :: Policy-Intent| Query-Intent
Policy-Intent :: Network-Function| SFC
SFC :: Network-Function, NW-Function[*]
Network-Function :: Firewall| IDS| Routing | LB| DPI
Firewall :: Header, Action
Header :: Hw-Src, Hw-Dst, Src-IP, Dst-IP,
          Src-Port, Dst-Port, Proto
Action :: Forward| Drop| Modify
Query-Intent :: Check-Conflict| Check-Loop| Check-Reachablity
```

Figure 4.3: INTPOL Intent Rules Describing Different Kind of Intents Where the User Can Specify at the Application Plane.

INTPOL Intent Rules Description

Next, we describe the intent rules for the INTPOL framework. Figure 4.11 shows the step-wise flow for interpreting the user requirements at the application plane and translating them to the corresponding bounded model program to check the necessary network policies. If we consider the steps for handling the user intents in Figure 4.2, then the steps *1-3* that is related to the application of the INTPOL framework are implemented. If an invariant in the user's policies are violated, the network admin is notified as shown in step *3a*. We illustrate the example process of translating the policy and query intents into a formal model using the Firewall as a sample network function. Consider the network topology described in Figure 4.2. In the rules example, we have a high-level network intents for hosts, $h1 \rightarrow h5$, $h2 \rightarrow h3$, $h3 \rightarrow h4$, and rules denying traffic between hosts $h2$ and $h4$. Each network host is connected to the network switch using a *layer 2* switch port. Similarly, network switches are connected to the network controllers (ONOS, ODL) using a layer-2 port. The bounded formal model accepts these requirements. The network topology information fetched from the SDN controller creates a model of packet propagation and changes in the packet state as the packet traverses along different paths from the source to the destination in the network. Linear Temporal Logic (LTL) [12], a form of model checking technique provided by NuSMV [37], in particular, characterizes the linear path induced by the Finite State Machine (FSM) of the network states [12]. The following section showcases how our model utilizes the bounds based on the network policies and topology to reduce the number of model checking states at the application plane. Then, we present the overall complexity of handling policy conflicts in the data plane.

4.4.3 Programming Network Functions (NFs)

We consider an example of programming an individual network function in the INTPOL framework. Consider the Deep Packet Inspection (DPI) as an example, derived from the Open source DPI tool nDPI [46]. The tool utilizes open-source libraries to inspect the packet header and payload.

```
# Protocols
# Format:
tcp:81,tcp:8181@HTTP
udp:5061-5062@SIP
tcp:860,udp:860,tcp:3260,udp:3260@iSCSI
tcp:3000@ntop
# Subprotocols
# Format:
host:"googlesyndacation.com"@Google
host:"venere.com"@Venere
host:"kataweb.it",host:"repubblica.it"
         @Repubblica
```

Figure 4.4: Data Collected by nDPI that Identifies Main Format, Protocol, and Information Related to Subprotocol.

The traffic collected using nDPI, shows patterns `<tcp|udp>:<port>,...@<proto>`, for the main protocol. For example network traffic over ports 81, 8181 is tagged as HTTP as shown in Figure 4.4. The `host:"<value>",...,@<subproto>` pattern shows host, and subprotocol information. If the host's DNS information is present in the traffic, the nDPI identifies host, and subprotocol, e.g., "venere.com", and

"Google", as domain provider.

```
Network−Function :: DPI
DPI :: Proto, Subproto
Proto :: <tcp|udp>:<port>,
         <tcp|udp>:<port> [∗] @<proto>
Subproto :: host:"value" [∗] @<subproto>
```

Figure 4.5: Expression of DPI Function from OpenDPI As an INTPOL Network Function

TThe INTPOL framework can be used to express the corresponding network function (NF) based on the ruleset of DPI (see Figure 4.5). The primary network function of DPI can be classified into protocol and sub-protocol. The protocol and sub-protocols can be expressed into protocol type (TCP or UDP), port, host website accessed, and the medium through which the query took place (e.g., Google).

4.4.4 INTPOL Model Checking Framework

Bounded Model Checking

Formal models, such as LTL, suffer from scalability challenges. The total number of model states can be as large as 10^{20} for some models [18]. Bounded model checking (BMC) [40] checks the state space for a counterexample using a user-specified bound k. BMC builds a boolean formula for each value of K that is satisfiable if a counterexample of length K exists. For a given transition system with 's' states, the model can be expressed using $k \times s$ variables. Once the bounded model is created, SAT [143] is used to check the boolean formula's satisfiability. The bounded model checking formula's completeness is established using completeness threshold, liveliness property, and induction tests, as Biere *et al.* [18] discussed above.

Establishing Threshold for Bounded Model Checking: The bound K on a model of the network M can be specified in terms of *reachablity diameter* rd(M), i.e., the minimum number of steps required to reach all reachable states. Another possible mechanism is *recurrence diameter* rdr(M) to utilize the minimum number of steps for reaching all reachable states. In our example 4.8, *rdr(M) = 5*, i.e., distance from host *h1* to *h5*, and *rd(M) = 11*, which indicates the number of steps needed to perform breadth first search over the network. These thresholds are utilized as a baseline to define bounds on the model during the empirical evaluation.

Formal Model for Network Verification

Network Invariants is defined as the network properties desired for optimal functioning and security of the network. For instance, virtual network isolation, absence of forwarding loops in the network, end-to-end packet reachability (absence of black holes), etc. Network verification is achieved by expressing network invariants based on the current topology configuration, traffic management rules, and high-level network requirements. We use temporal logic-based (LTL) network verification [12] to check if the underlying network meets high-level network requirements. An LTL invariant is evaluated along the linear path. If the invariant state holds for all the paths starting in a given state, we consider the invariant to be true.

Table 4.1 describes the formal semantics used for LTL. We can describe the network invariants, such as global reachability $G\ p$, conditional packet processing $p \cup q$, and white-listing policy violations for the underlying network using the queries created using the LTL model checking rules (invariants). The invariants serve as *Query Intent* in our system. The next section discusses how service function chain intent and network function intents are represented in the INTPOL system.

Table 4.1: Formal Semantics of LTL, Which Is Used for Expressing Network Invariants

LTL Rule	**Rule Interpretation**
F p (in the future p)	Condition p holds in one of the future time instants.
G p (globally p)	A certain condition p holds globally in all future time instants.
p U q (p until q)	Condition p holds until a state is reached where condition q holds.
X p (next p)	Starting condition p is true in next state.

4.4.5 Intent Handling in INTPOL

We discuss how an intent submitted by the user, as described in Figure 4.11, is handled by the INTPOL system. We describe scenarios - SFC Intent, Network Function Intent such as Firewall (FW) that can be defined individually or as part of SFC Intent. This approach allows the user to check end-to-end reachability and policy conflict checking with the help of illustrative examples. Also, we provide a use case that shows how multi-domain SDN scenario verification is achieved.

Service Function Chaining Intent

We consider Service Function Chain (SFC), as shown in Figure 4.6. The example describes the traffic processing between different end-point groups (EPGs[1-4]). Also, The network gateway (NAT) in the example acts as a traffic classifier. There are

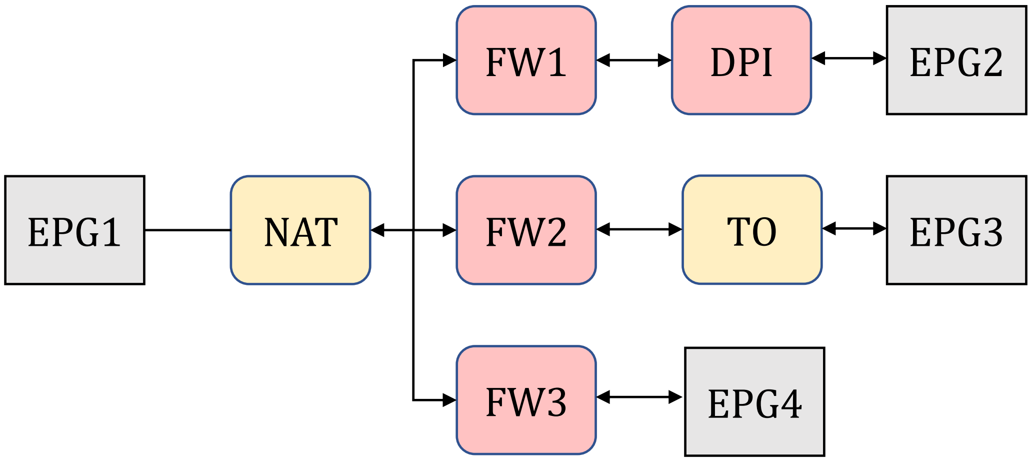


Figure 4.6: A Service Function Chain (SFC) Scenario with Multiple Network Functions (NFs).

three separate service chains in this example. The HTTP traffic is classified at the NAT gateway and follows the corresponding service chain path, with Deep-Packet Inspection (DPI), i.e., SFC1: EPG1 → NAT → FW1 → DPI → EGP2. If the traffic is meant for video streaming services, it follows an alternate path with Traffic Optimizer (TO), i.e., SFC2: EPG1 → NAT → FW2 → TO → EGP3. All other traffic follows SFC3: EPG1 → NAT → FW3 → EGP4. We illustrate the use of INTPOL intent rules, as described in Figure 4.7, to represent the individual service chains. We consider the network is following a white-listing approach for stronger security, and the traffic is only allowed between EPG1 as the source and EPG[2-4] as the destinations. Therefore, the traffic between EPG [2-4] is blocked as part of the network policy.

The example in Figure 4.7 illustrates the implementation of complex network service chains. Variables *dst* and *sf* in lines 1-2 are used to define packet destination and service function, respectively. The formal model of the packet propagation through a chain of different network functions is described in lines 5-14. For instance, if the packet service function is NAT and the destination is EPG2, it indicates the packet

```
MODULE main
VAR dst: {EPG1, EPG2, EPG3, EPG4}
     sf: {NAT, FW1, FW2, FW3, DPI, TO}
ASSIGN
next(sf) := case
     sf = NAT & (dst=EPG2) : FW1;
     sf = FW1 : DPI;
     sf = DPI : EPG2;
     sf = NAT & (dst=EPG3) : FW2;
     sf = FW2 : TO;
     sf = TO : EPG3;
     sf = NAT & (dst=EPG4) : FW3;
     sf = FW3 : EPG4;
     TRUE: NAT;
 esac;
 next(dst) := dst;
 INIT sf = NAT;
```

Figure 4.7: Example Usage of LTL-based Model Checking Framework for Implementing Three Separate Service Function Chains.

belongs to the SFC1, and the packet is forwarded to FW1, DPI, and finally, EPG2, shown in lines 6-8. Similarly, lines 9-11 represent the implementation of SFC2, and lines 12 and 13 are the implementation of SFC3.

Network Function Intent

We consider an individual network function *Firewall (FW)* to check how the rules of network functions can be expressed using the INTPOL framework. Consider two invariants I1, and I2, that can be expressed using LTL equations (1) & (2) below:

I1: Traffic sent by host h1 should eventually reach host h5.

$$\forall p \in Packet : G(send(h1, p) \cap any(p)) \rightarrow F(recv(h5, p)) \tag{4.1}$$

I2: any traffic sent by h2 should not reach h4.

$$\forall p \in Packet : G(send(h2, p) \cap any(p)) \rightarrow G(\neg recv(h4, p)) \tag{4.2}$$

Equation (1) checks the reachability property between h1 and h5, and equation (2) checks the firewall rules between hosts h2 and h4. If any network state along the patch h1-h5 violates the network invariant, it will be produced as a counterexample of the model. Appropriately, we can verify higher-level network intents using model checking based on the LTL formal semantics presented earlier.

```
MODULE main
   VAR switch: {s1,s2,s3};
        src: {h1,h2,h3,h4,h5};
        dst: {h1,h2,h3,h4,h5};
   ASSIGN
   next(switch):= case
    switch = s1 & (dst !=h1 | dst !=h2):s2;
    switch = s2 & (dst !=h3): {s1,s3};
    switch = s3 & (dst !=h4 | dst !=h5):s2;
    TRUE: s1;
   esac;
   next(src):= src;
   next(dst):= dst;
INIT switch = s1;

check_ltl_bmc -k 5 "G (src=h1->F(dst=h5))"
check_ltl_bmc -k 5 "G (src=h2->G(!dst=h4))"
```

Figure 4.8: An Example of a Bounded LTL Model Utilizes Network Topology from the SDN Controller to Create a Model Specification. The Last Line Represents a Query Intent to Check If Any Packet Starting from (src=h2) Can Eventually Reach (dst=h4)

We consider an example of modeling network intents from Figure 4.2 using a bounded LTL model. Since the intents are representative of the Firewall rules, a type of *Policy Intent*, we model the packet header using the example described in

Figure 4.8. The variables *switch*, *src*, and *dst* in the *VAR* section - lines *2-4* represent the scope of values for switches, source, and destination addresses, respectively. We use these variables to represent the values for this example, but depending on the type of intent, the values can take numeric range, e.g., *src=192.168.1.0/24*, *switch =of:00000001*. The *ASSIGN* section lines *5-13* check the next state transition of a network packet. When the packet is located at switch s1, if the packet's destination address is not h1 or h2, i.e., (dst!=h1 — dst!=h2) is forwarded to switch s2. Alternatively, we can consider the next transition of state *switch* for this packet to be s2. Similarly, based on the packet header match conditions, the state transition of the packet is determined in the program. The block *INIT* - line *14* is used to specify the packet's starting state.

Case Study: Conflict Checking in Hybrid-SDN Networks

SDN domain can be combined with traditional networking such that the traditional network functions, e.g., BGP routers, establish multi-domain communication. This type of networking architecture is called hybrid-SDN [5, 128]. Figure 4.9 shows an example of the hybrid-SDN networking architecture. Some components are Software-Defined, e.g., Switches (s1-5) and (s11) are managed by ONOS-01, whereas switches (s6-10) are controlled by ONOS-02. The hybrid environment presents challenges such as certain parts of the network being managed using Commercial Off the Shelf (COTS) network management applications as discussed by Vissicchio et al. [142]. It is challenging to address some research questions like a) how will route handling decisions by BGP be in sync with sub-network managed using SDN? b) can the disconnection between SDN and non-SDN environments introduce security loopholes? c) How will different parts of the hybrid network ensure network sync to guarantee flow propagation, packet delivery, and optimal network performance?

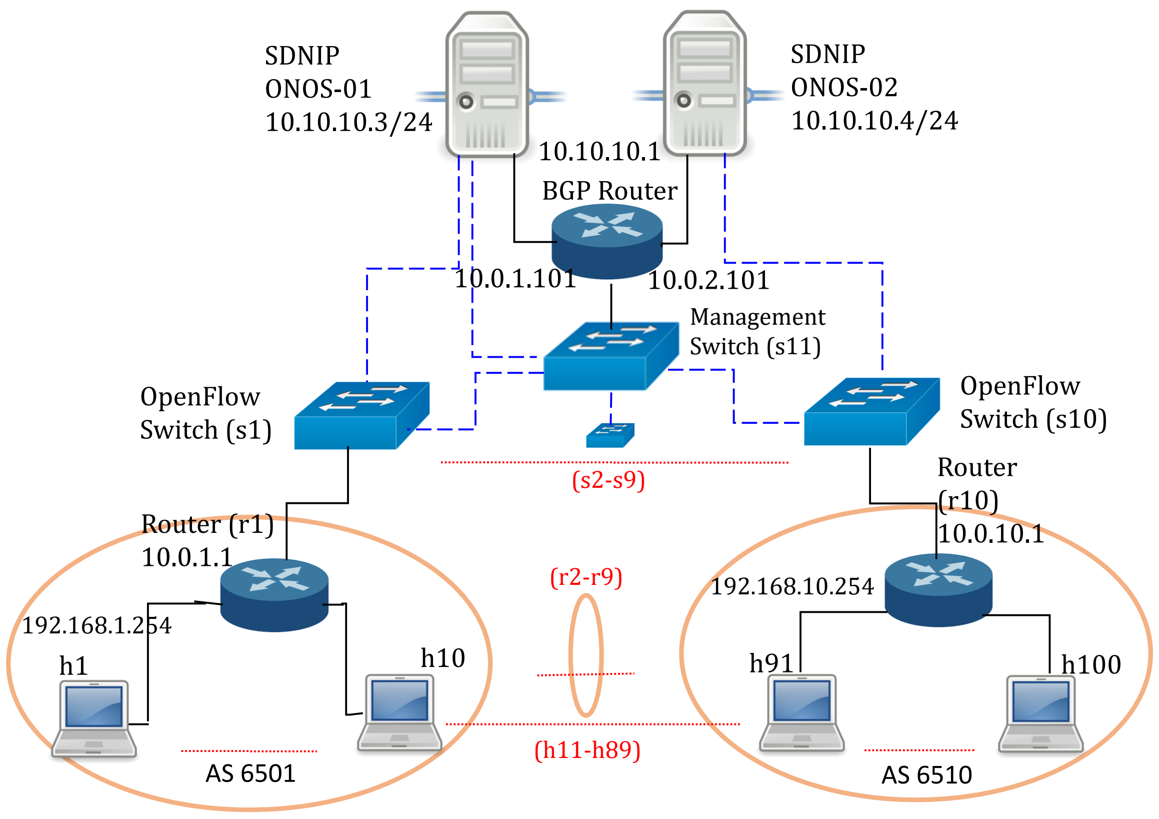


Figure 4.9: A Hybrid Network Scenario With Network Components Based on Traditional Networking (BGP Routing) and Openflow Network (Highlighted in Blue).

We expand the INTPOL framework to provide end-to-end policy verification in a hybrid network scenario with Border Gateway Protocol (BGP) for communication across multiple domains, as provided in Figure 4.9. The BGP-based routing technology role is to enable communication between the network managed by ONOS-01 and ONOS-02. Moreover, the end hosts (h1-h100) in each network are connected to the SDN networking using routers r1-r10. Hence, the Openflow network can only see the packets coming from routers. This also shows the BGP router is executing control and data plane functionality. The routers apply NAT functionality to masquerade the local IP address into the public domain IP to enable BGP communication.

We represent the network elements in the hybrid-SDN as network functions. The network functions are tested to verify how the rules can be expressed using the bounded model checking approach. For example, if we want to check for network reachability in a hybrid environment to guarantee packet delivery, we can express the hybrid network model using the INTPOL framework. The network reachability invariant *I3* can be expressed as:

I3: Network flow sent by host h1 should eventually reach host h100.

$$\forall p \in Packet : G(send(h1, p) \cap any(p)) \rightarrow F(recv(h100, p)) \tag{4.3}$$

Where the equation is used to check for reachability property between *h1* and *h100*. Suppose there is any network state along the path h1-h100 that violates the network invariant. It will be produced as a counterexample of the model, and the high-level network intent is verified using the bounded checking model.

```
MODULE main
    VAR NF: {s1 ,s2 ,sm, bgp1 , bgp2 };
      src: {h1 ,h20 };
          dst: {h1 ,h20 };
ASSIGN
   next(NF):= case
     NF = bgp1 & dst !=h1  : s1;
     NF = s1 & dst !=h1: sm;
     NF = s1 & dst =h1: bgp1;
     NF = sm & dst =h1:   s1;
     NF = sm & (dst =h1|dst = h20):{s1 ,s2 };
     NF = bgp2 & dst !=h20: s2;
     NF = s2 & dst =h20:   bgp2;
     NF = s2 & dst !=h20  : sm;
     TRUE: s1;
   esac;
   next(src):= src;
   next(dst):= dst;
INIT NF = s1;
check_ltl_bmc -k 5 "G(src=h1->F␣(dst=h20))"
```

Figure 4.10: An Example of a Bounded LTL Model Utilizes Network Topology from the Hybrid-SDN Example in Figure 4.9 to Create a Model Specification. The Last Line Represents a Query to Check If Any Packet Starting from (src=h1) Can Eventually Reach (dst=h20). The Example Is Shortened to Simplify the Presentation, but It Can Be Extended Between Hosts h1 & h100.

Figure 4.10 provides an example of modeling network intents that represent hybrid-SDN connectivity. The example shows the connectivity verification between the source host *h1* under the BGP router *bgp1* and the destination host *h2* under BGP router *bgp2*. Both routers are assumed to be connected to the management switch *s11* as shown in Figure 4.9. The variables NF, src, and dst in the *VAR* section represent the network function (switches and routers), the source node, and the destination node, respectively. We used variables to represent values in this example, but numerical values can also be assigned, such as an *IP address*. The *ASSIGN* section lines *5-18* check the next state transition of a network packet. When the packet is located at router bgp1, it will be transmitted to the switch s1 if the destination is not *h1*. Similarly, based on the packet header match conditions, the state transition of the packet is determined in the program. The section *INIT* (line 21) is used to specify the packet's starting state. In the evaluation section 4.6.1, we show the performance of the INTPOL LTL-BMC model, checking for inter-domain and intra-domain communication.

4.4.6 Application Layer Packet Caching and Policy Composition

The use of INTPOL intent rules creates a formal model for packet processing across all the network functions and flow rules. We create a multi-level packet processing model by first processing SFC dependencies and individual network function level dependencies. As a result, it is possible to pre-compute the paths of all packets across the network that traverses several network functions. To facilitate the two-level checking in INTPOL, we perform two approaches at the application plane layers 1) match and action caching and 2) policy composition to reduce the conflict checking overhead at the data plane.

Packet Match/Action Caching

We define a few notations P - set of all packets and R - set of all rules present in the network functions. The goal of the policy optimization at the application plane is to pre-compute the path of the packet(s) along the SFC. When the packet arrives at a network function, two functions are activated: cacheMatch (P) and cacheAction (P). They check the result of the packet transition along the path in the SFC that is defined for the packet. If a previous packet transitioned along the service function chain, the *match, action* values for the packet are cached at the application plane. For instance, if $p \in P$ matches the rule of FW1 network function r_{FW1}, we check the result of a similar packet along the path $FW1 \rightarrow DPI \rightarrow EPG$ and insert the rule into the switch flow table at the data plane.

Policy Composition

The role of the policy composition is to check all the rules having overlapping header space and common actions, i.e., $\forall r \in SFC$, where each rule is defined by *match, action* pair, or $r = \{m, a\}$. Consider two rules r_i, r_j, $m_i \subset m_j$, and $a_i == a_j$, we can create a new rule that combine the two rules by $r_k = r_i \cup r_j$. Many policies in the application plane are a subset of one another, which increases the packet processing overhead at the data plane since the SFC checks redundant policies. We can compose such policies at the application plane, which will reduce the packet checking overhead at the data plane. If $a_i \neq a_j$, we can create rule r_k, where $m_k = m_i \cup m_j$, and $a_k = \{a_i, a_j\}$. This approach shows the case where packet headers (match fields) have overlap but have different actions. Such policies can be abstracted to apply a set of grouped actions to the packet instead of sequentially. The joint application of actions on the packet header reduces the packet processing overhead once the policies

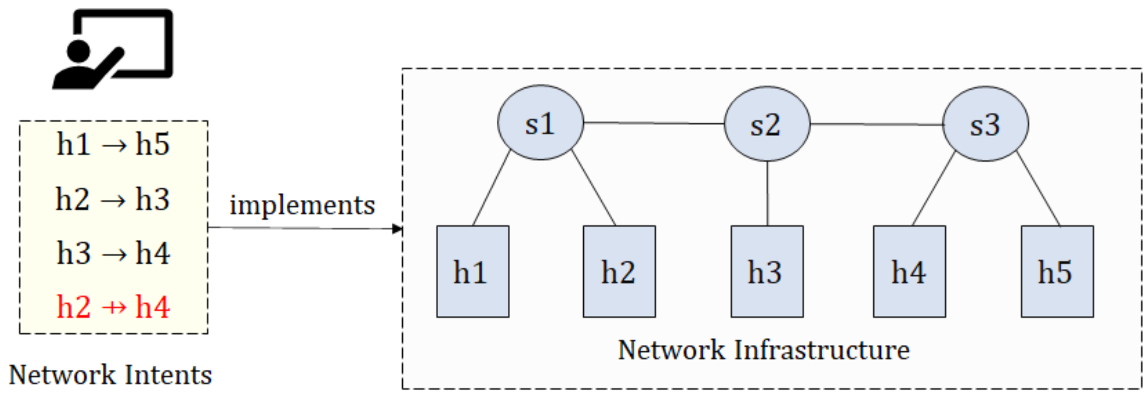
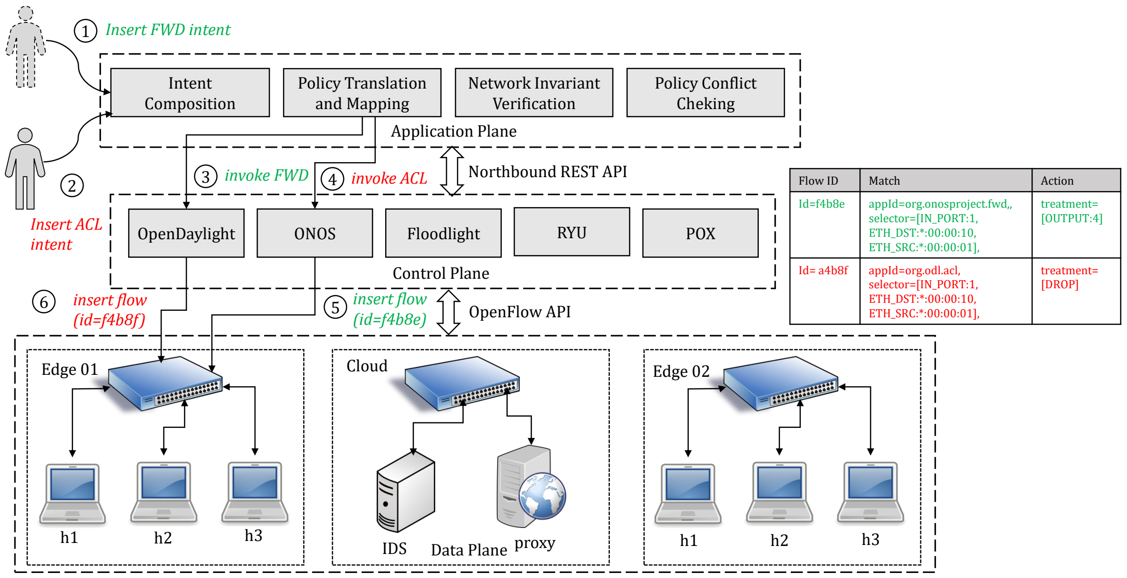


Figure 4.11: Example of Network Intents Expressed at The Application Plane.

are applied to the data plane in flow rules.

4.4.7 Policy Conflict Detection

We illustrate the problem that can exist because of a mismatch between high-level network security and orchestration requirements (intents). In the example Figure 4.11, we have high-level network intents for hosts, $h2 \rightarrow h3$, $h3 \rightarrow h4$. This creates OpenFlow rules at switches $s1, s2, s3$. The combination of these rules can cause a violation of security requirements. We can have $h2 \rightarrow h3 \cup h3 \rightarrow h4 = h2 \rightarrow h4$. The problem of identifying such network anomalies can become quite involved in a network consisting of thousands of sub-networks, hosts, and switches. Moreover, the example above describes a case of a simple access control list (ACL) intent expressed at the application plane. The advent of network function virtualization (NFV) [106] has allowed the creation of network functions such as load balancer, intrusion detection system (IDS), and deep-packet inspection (DPI) as part of a programmable network. Identifying policy inconsistencies across different network protocol stack layers and network paths becomes a challenging task. Existing rule-conflict detection mechanisms [124, 66] focus exclusively on OpenFlow rule conflicts at the data-plane



Flow ID	Match	Action
Id=f4b8e	appId=org.onosproject.fwd,, selector=[IN_PORT:1, ETH_DST:*:00:00:10, ETH_SRC:*:00:00:01],	treatment= [OUTPUT:4]
Id= a4b8f	appId=org.odl.acl, selector=[IN_PORT:1, ETH_DST:*:00:00:10, ETH_SRC:*:00:00:01],	treatment= [DROP]

Figure 4.12: Example of Policy Translation and Flow Rule Insertion Using Two Different Northbound REST-based Controller Modules (e.g., ODL, ONOS).

level. Conflict detection and resolution mechanism at the switch level can introduce unnecessary read-write latency issues and interrupt the normal functioning of the network. Symbolic model checking (SMC) can express the network properties at a higher level of abstraction. The model checkers such as NuSMV [38] allow the granular representation of network security and end-to-end connectivity properties.

4.4.8 Motivation

We consider the example of networking modules *Forwarding* present in the SDN controller ONOS, and Access Control List (ACL), present in the SDN controller OpenDaylight (ODL) to highlight the problem associated security and network management policies inserted in an SDN managed data-centric network. In this example network present in the Figure 4.12, we have OpenFlow switch *s1*, connected to two

SDN controllers ONOS, and OpenDaylight.

Step 1 - network admin inserts an intent to allow packet forwarding along ports of switch *s1* (s1-h1, s1-h2, s1-h3, s1-s2).

Step 2 - during a similar time frame, another network admin inserts an *Access Control List* (ACL) Intent.

Step 3 - the northbound REST API present at the application plane invokes ONOS module *org.onosproject.fwd()*.

Step 4 - OpenDaylight module *org.odl.acl()* module is invoked by REST API call from another session. Since there is no inherent mechanism present as part of OpenFlow framework which checks the dependencies between objectives of two security policies, we can have conflicting flow rules. As shown in table present in Figure 4.12, the Flow ID *f4b8e* is inserted in switch s1 in Step 5 by ONOS controller. This flow allows traffic entering port 1 of s1, with fields {ETH_DST = *:00:00:10, ETH_SRC = *.00:00:01} to be forwarded. On the other hand the Flow ID *f4b8e* inserted in s1 by ODL controller in Step 6 blocks the traffic matching the same criteria.

4.5 System Architecture and Implementation

4.5.1 Experimental Setup

We run INTPOL on an OpenStack-based (Openstack Victoria) cloud network comprising two Dell R620 servers and two Dell R710 servers, all hosted in the data center. Each Dell server has about 128 GB of RAM and a 16 core CPU. The SDN controllers ONOS & Opendaylight-Carbon provided network management and orchestration in our framework. For the network topology, we used Open Virtual Switch (OVS 2.13) for the data plane switches and utilized Docker container[1] to add hosts

[1]https://www.docker.com/

into the topology. Furthermore, we show the utilized system's components description in Table 4.2.

Table 4.2: INTPOL Components Used in the Implementation

Component	**LOC/Version**	**Language / Framework**
SDN Controller	OpenDaylight Carbon, ONOS	Java, REST APIs
Intent Specification	500	python with Flask APIs
Bounded Formal Model	2.6.0	python, NuSMV
Intent Datastore	3.30.1	SQLite
Network Topology	Variable	Docker Containers

4.5.2 INTPOL Implementation

We show Figure 4.13 that describes the dataflow in our INTPOL framework and its different components. INTPOL has an *INTENT checking*, *Policy Composition*, *Policy Conflict Detection and Resolution*, and *Intent Processing* modules. *Our INTENT Checking* module is responsible for determining the intent type, i.e., a network intent or query intent. We utilize the administrative panel's intent to create a formal network infrastructure and policies model. The network policies are checked for the type of intent. If the intent is a *Network Intent* (network function rule or service function chain requirement), the intent is added to the existing state transition system defined for the formal model. Alternatively, if the type of intent is *Query Intent*, it is used to create the LTL queries for checking the network invariants. For instance, if the query asks about the possible path between two network hosts, we create a query to check if a packet starting from the source address finally reaches the destination, as explained in the previous section. We also use the network topology information to place the bounds on the formal model. The value of the bound depends on the network diameter for our model. If the invariant results in a counterexample (violation of network policy), the network admin is informed about the violation.

The *Policy Composition* module receives the high-level requirements from the administrator and compose an appropriate policy based on the target network's SDN controller to account for the syntax requirements (see Figure 3). Additionally, the SDN controller is queried to extract information about network topologies, such as connectivity between hosts and switches. We formulate a formal model using the transition system and the topology information. These requirements are then submitted to the *INTENT Processing* that will check if the intent is appropriate, send it to the SDN controller for compilation and installation, and finally report back to the administrator whether the intent is installed or not. Based on the Policy Composition, we check using the formal model approach (BMC model) for the conflict in the policy, and later on in the *Flow Rule Conflict Checker*.

Algorithm 4 describes the processing of intents at the application plane and the generation of the formal model. The rules are sent to `REQUIREMENT-PARSER (R)` function, which parses the submitted network intents. If the type of intent is *Service Function Chain* - lines 25-27, the intent passes to the `SERVICE-FUNCTION-CHAIN (SF-List)` procedure. The source and destination network functions are identified - on lines 13-14, and the model is updated with state-transition corresponding to the path between the network functions. Similarly, if the type of intent is an individual *Network Function*, then a call to `NETWORK-FUNCTION (r)` checks the matching criteria for a rule (firewall rule, IDS rule) and corresponding action - lines 19-20. The formal model is updated with the values of the header match and related action - line 21. Suppose the type of intent is *Query Intent*. In that case, the call to `NETWORK INVARIANT (r)` procedure is invoked for processing of the submitted query, e.g., checking end-to-end packet reachability, application plane conflict check - lines 4-10. If the network invariant is satisfied, the rule is kept in the list of rules that a call to `RULE-CONFLICT-CHECKING (R)` in algorithm 5 will further analyze. If the invariant fails, the admin is notified - line 32, and the affected rules (formal model states presenting counterexamples) are removed from the set of non-conflicting rules.

The implementation goal of INTPOL is to reduce the data plane conflict-checking results. If the application plane verification procedures do not find any violations, we introduce an approach to cash the packet traversal in the SFC path. The procedure *Packet Caching* in Algorithm 4 examines the packets in the SFC path. If a packet belongs to the matching cache, we know it has been seen before, and we obtain the packet's action. The traffic header of the incoming network traffic P is examined. If there is a match, the data plane is populated with the corresponding flow rule instead of reviewing the entire chain of network function states from the source to destination lines 39-42. If there is no match, we check the packet processing for the $\{header, action\}$ pair along SFC link lines 42-45. This helps in faster packet processing at the application layer.

Algorithm 4 Model Generation and Verification

```
1: procedure Network-Invariant (r)
2:     add-ltl-spec (r)
3:     K ← network diameter
4:     add-model-bounds (K)
5:     if network-invariant-violation (r) then
6:         return False
7:     else
8:         return True
9: procedure Service-Function-Chain (SF-List)
10:     for i ∈ range (SF-List[1,n]) do
11:         Extract-Path ($nf_i$, $nf_{i-1}$)
12:         add-src ($nf_i$), add-dst ($nf_{i-1}$)
13:         add-state-transition ($nf_i$, $nf_{i-1}$)
14: procedure Network-Function (r)
15:     header, action ← {match, action} ∈ r
16:     add-src (header.src), add-dst (header.dst)
17:     add-state-transition (header, action)
```

```
18: procedure REQUIREMENT-PARSER (R)
19:     for r ∈ R do
20:         if r.type ∈ Service-Function-Chain then
21:             SF-List ← r.extract()
22:             call SFC-Create (SF-List)
23:         else if r.type ∈ Network-Function then
24:             call Network-Function (r)
25:         else if r.type ∈ Query-Intent then
26:         else if call Network-Invariant (r) == False then
27:             send ($r_c$ ∈ R to admin)                    ▷ Rules in violation
28:     call RULE-CONFLICT-CHECKING (R)
29: procedure PACKET CACHING (P)
30:     for p ∈ P do
31:         if p[header] ∈ cacheMatch (P) then
32:             p[header] ← checkMatch (P)
33:             p[action] ← cacheAction (P)
34:         else
35:             checkLinkSFC (p[header])
36:             checkLinkSFC (p[action])
```

Algorithm 5 Flow Rule Conflict Checking Algorithm

```
1: procedure RULE-CONFLICT-CHECKING (R)
2:     $R \leftarrow$ current flow rules
3:     $R = \{match(R), A(R)\}$
4:     C $\leftarrow$ Conflict Set
5:     for $i \in \{1,n\}$ do
6:         for $j \in \{1,n\}$ do
7:             if $match(R_i) \subseteq match(R_j)$ OR $match(R_j) \subseteq match(R_i)$ AND $action(R_i) == action(R_j)$ then
8:                 C.add(Inheritance)
9:             else if $match(R_i) \subseteq match(R_j)$ OR $match(R_j) \subseteq match(R_i)$ AND $action(R_i) \neq action(R_j)$ then
10:                C.add(Polymorphism)
11:            else if $match(R_i) \cap match(R_j) \neq \emptyset$ AND $action(R_i) == action(R_j)$ then
12:                C.add(Aggregation)
13:            else if $match(R_i) \cap match(R_j) \neq \emptyset$ AND $action(R_i) \neq action(R_j)$ then
14:                C.add(Composition)
```

In the Algorithm 5, we check the network policies for overlapping header and action pairs. We classify policies into categories *Inheritance*, *Polymorphism*, *Aggregation*, and *Composition*. This helps in identifying conflicting policies in a service chain. If the header of one policy is a subset of another policy, as described in Section 3.4.2, the rules can be merged. The policy conflict cases *Inheritance* and *Polymorphism* - lines 7-10 cover these scenarios. The policy optimization by merging such overlapping policies will help reduce the overhead of flow rule conflict checking at the data plane. The conflicts *Aggregation*, and *Composition* - lines 11-15, can also pose issues such as security violations and packet processing overhead. We use the knowledge of domain experts to resolve conflicts of this nature and ensure there are no sub-optimal policies or security loopholes in the application plane. Consider two rules, rule 1 - {permit icmp}, rule 2 - {deny 10.0.0.0/8}. These rules have overlapping header space but different actions. These rules will conflict under the class *Polymorphism*. Conflict resolution can be performed using rule priority. Consider the case where 10.0.0.0/8 is a production network (security-critical). We can assign high priority to rule 2 to resolve the policy conflict. On the contrary, if the ICMP traffic is important for the business operation, rule 1 will be assigned a higher priority to resolve conflict. The policy intents are compiled into a bounded formal model (Section III-B) using the rules described in Figure 3.

4.5.3 Intent Processing Module

SDN controller's intent framework allows the users to specify their networking and security policies. Figure 4.13 shows INTPOL data flow to process the intents and install the flow rules. Consider an intent that specifies the reachability requirement between host h1 and host h5. This Intent(s) is sent asynchronously to SDN controller's *Compiling* stage. The *Compiling* stage performs various checks on the incoming intent, and if the compilation is successful, a list of installable intents is returned. The intents need to be validated for their feasibility and connectivity with regards to the given network topology, network criteria, resource availability, etc. The compilation process computes a primary shortest path and a backup path between the two given hosts. Here, the shortest path from host h1 to host h5 would be h1-s1-s2-s3-h5, given that all the links have the same weights. Suppose a node specified in the user

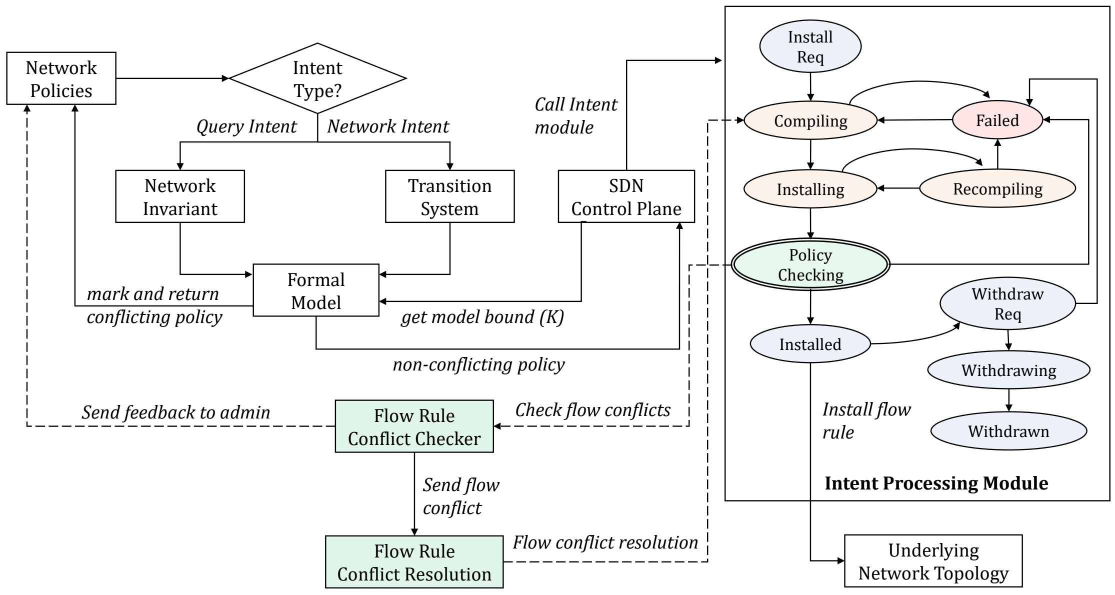


Figure 4.13: INTPOL Data Flow Diagram Describing Multi-level Network Policy Processing. The Formal Model Analyzes the Policies at the Application Plane, and the Policy Conflict Checker Checks the Conflicting Flow Rules at the Control Plane.

intent cannot be reached by the other node specified due to lack of connectivity, link failure, resource unavailability, or any other reason. In that case, that intent fails the compilation. We cannot say if this failure is due to some temporary network failure or some temporary event. Hence, the failing intent is kept in a *Compiling* state. If the compilation fails, the intent's state is assigned to a *Failed* state. In the event of network topology change, or link re-association, failed intents are considered again for compilation. In the event of topology change, if the network connectivity is regained, then compilation may succeed. The installable intents are sent from *Compiling* state to *Installing* state. If the installation fails for any reason, then it goes to *Recompiling* state. These intents are kept in the temporary state of recompilation for a short time before sending them to the failed state. However, the intent request may not necessarily fail due to the policy composition. Other external factors like reconfiguration could also cause the intent to fail. In that case, the intent is only sent to the failed state instead of notifying the network administrator. Once the intents pass, the *Installing* state is also analyzed for *Policy Conflict* as described in the following subsection, and in the event of no conflicting policies, the state transitions to *Installed Intent.* These intents are installed as flow rules in the underlying OpenFlow network. The intents that are successfully installed can be sent to the withdraw state if the admin wishes to remove them from the system. The withdraw request is processed, and the associated flow rules, links, and devices are unlinked from the intent. During this stage, the intent is kept in a temporary state of ”withdrawing” before it is withdrawn.

4.6 Performance Evaluation

This section presents the evaluation of the INTPOL framework using different application scenarios and network setups. First, we offer a case study to show how INTPOL performs in a service function chaining (SFC) scenario, comprising inter-domain communication in a hybrid-SDN environment and intra-domain policies within the same SDN domain. Second, we utilize the Stanford topology to apply the INTPOL approach to a large network scenario [81]. The topology comprises 14 operational zones (OZ), Cisco routers connected via 10 Ethernet switches to 2 backbone Cisco routers. The topology has 757,000 forwarding entries, 100+ VLANs, and 1500 ACL rules. Finally, we use policy conflict checking and the network traffic reachability as an invariants to assess the INTPOL framework's scalability on a large scale network with an increase in the number of intents.

4.6.1 Hybrid Network Scenario

Inter-Domain Communication

The overall network described in Figure 4.9 can be represented in the INTPOL framework using two-level of abstractions, i.e., SFC intents and Network Function (NF) intent. We consider the inter-domain network a special case of Service Function Chain (SFC). The routers (r1-10) can be interpreted as NAT providers, and switches (s1-10) perform packet switching and basic traffic filtering, which helps in reducing the overall overhead of checking network policies. The end-hosts can be represented as an abstracted group, i.e., $\{h1, .., h10\} \in AS6501$, {ONOS-01, ONOS-02} $\in AS6500$, and so forth. We consider the reachability between the hosts and the policy conflict(s) as the network invariant in this scenario.

The inter-domain level is the autonomous system (AS) level, which has the SDN controllers and the edge routers *r1-r10.* We consider this as network function NAT within each AS. The administrator requirements are specified such that the end-hosts (*h1-h100*) can reach each other, or some of them should never be communicating with any other network function. In this scenario, we note two policy checking levels: the SDN controller level and the individual traditional routing domain's level. On the one hand, The SDN level domain checks for the AS and handles the OpenFlow rules between ASes. On the other hand, each AS (*r1-r10*) checks the inter-domain invariant. This allows the admin to ensure end-to-end packet reachability within and between domains. Consequently, the evaluation goal of this scenario is to measure the performance of network invariant checking between the domains.

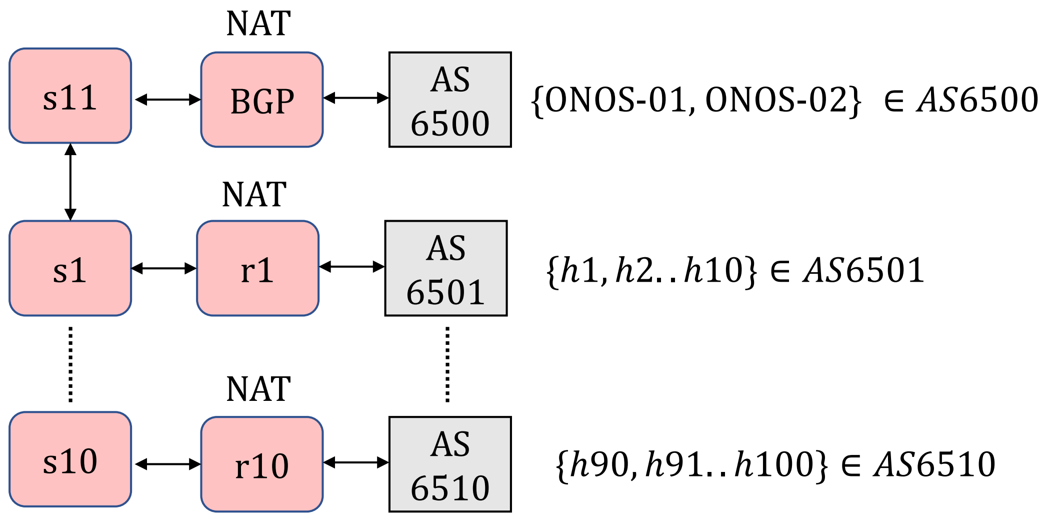


Figure 4.14: An Encapsulated Representation of Hybrid Network Scenario as a Special Case of Service Function Chaining (SFC).

We performed an experimental evaluation using LTL full-scale model checking, and LTL bounded model checking, i.e., LTL-BMC, in a multi-domain scenario described in Figure 4.14. We observed that performing network invariant checking by abstracting the packet processing across domains to a particular SFC case allows inter-domain packet switching and routing policies to be analyzed quickly and efficiently. Each network domain is comprised of ten hosts. We incremented the number of domains from 2-to 10 to observe the time required for LTL and LTL-BMC model construction and packet reachability property checking, as shown in Figure 4.15.

The value of the network bound K was selected based on the diameter of the current network, i.e., if the maximum path length between two hosts is 10, we utilized $K=10$. We can use the bounds based on the policy composition time and the number of policies in the network. The path length for a large network can be huge, but the network diameter-based bound helps provide appropriate coverage for the policy composition. Our inter-domain scenario's results are shown in Figures 4.15, 4.16.

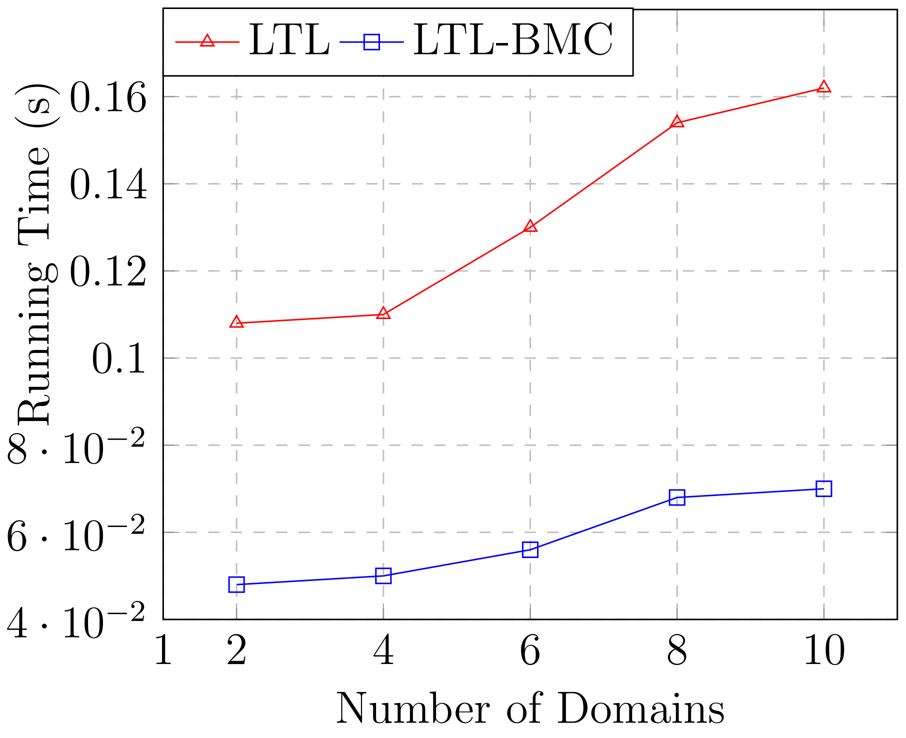


Figure 4.15: The Experimental Analysis of INTPOL in Inter-domain Hybrid Network. The LTL-BMC Model Verification Time Scales Well as the Number of Domains Increase in the Network.

The invariant checking using BMC for #domains=2 finished in 0.048s, whereas LTL required 0.108s. We observed a similar trend as we increased the number of domains to 10. For #domains=10, BMC required 0.07s, whereas the LTL finished in 0.016s. Overall, the BMC was $> 2x$ faster invariant checking than the LTL framework. This shows that BMC scales well in a multi-domain scenario. The INTPOL framework is generalizable to incorporate multiple types of network setups. Furthermore, The BMC model has a significantly less number of Binary Decision Diagram (BDD) nodes in comparison to the LTL model (Figure 4.16). For #domains=10, the BDD has 345 nodes, whereas the LTL model has 964 nodes. The increase in the LTL nodes limits the scalable network verification for large-scale models instead of the BDD model.

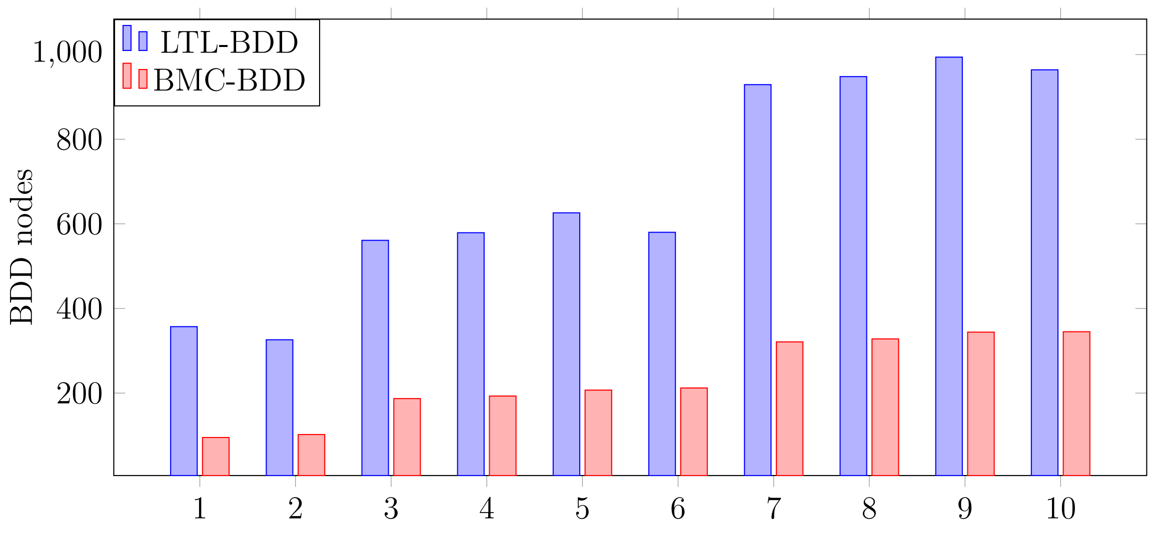


Figure 4.16: The Experimental Analysis of INTPOL in Inter-domain Hybrid Network Scenario. The BDD Nodes LTL-BMC Model Scales Well as the Number of Domains Increases in the Network.

Intra-Network Communication

The hosts h1-h10 utilize the traditional routing for communicating with each other, making the SDN network oblivious to the traffic managed by routers r1. Suppose we have a distributed geographical network, e.g., 192.168.1.0/24 is managed locally by r1-5. Each router connects to a switch, e.g., r1-s1, and all switches s1-10 connect to a management switch s11, which is connected to the *BGP* router (s11-bgp). The BGP router connects to ONOS-01, and similarly, 192.168.10.0/24 is managed by r6-10, which is connected to ONOS-02. We utilize the SDNIP application [94] to allow ONOS to run and communicate with the BGP router through iBGP protocol and communicate with routers *r1-r10* using eBGP protocol. Due to space limitations, we refer the reader to SDNIP details in [94] for details about connecting inter-domains with the controller. Next, we check the scalability of INTPOL by increasing the number of hosts within one domain. We consider switching and access control functions

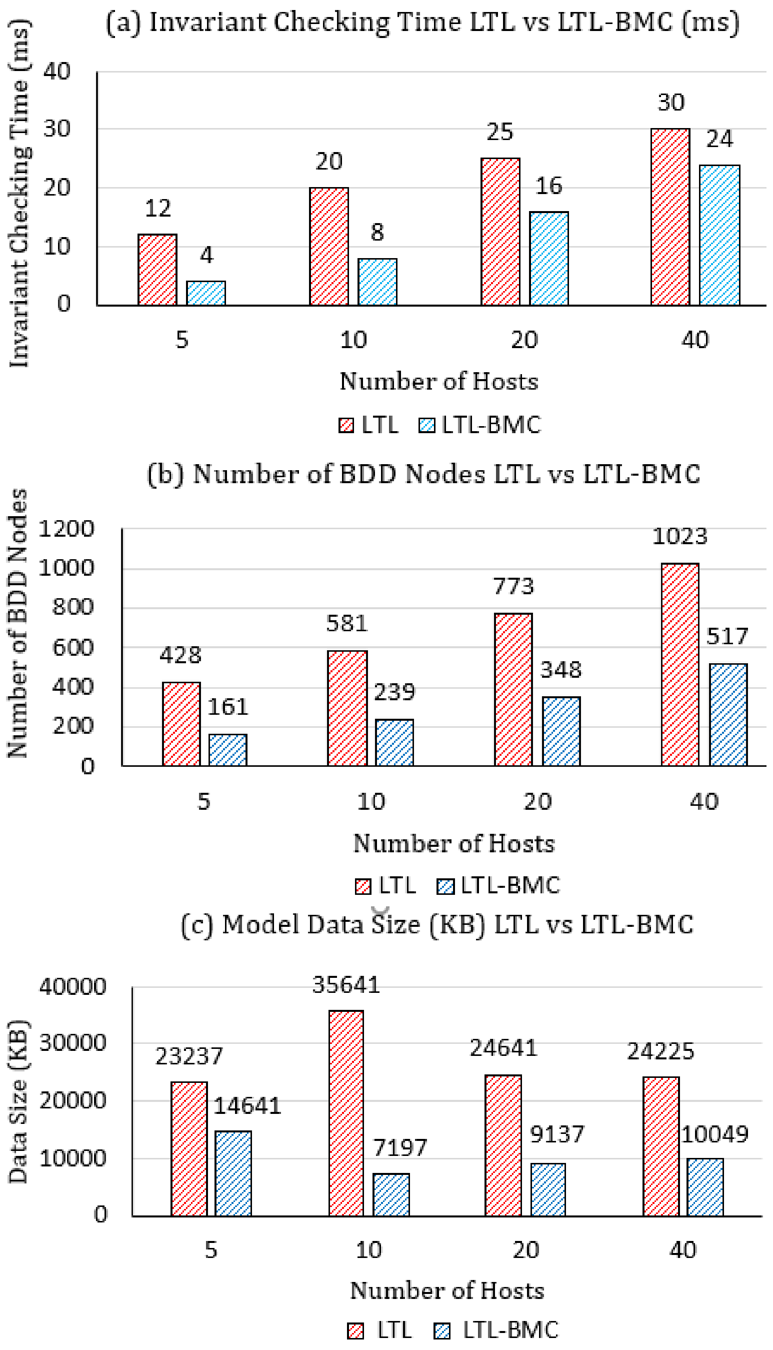


Figure 4.17: INTPOL Model Checking Framework Evaluation in a Single Domain Environment. As We Increase the Number of Hosts in Domain As6501, the State Nodes, Data Size of the Model, and Time of Model Checking Are Reduced.

within AS6501 to be managed as part of the routing domain of router r1.

To evaluate the scalability of our proposed INTPOL framework within a single domain AS6501, we experimented by measuring the performance of the BMC model compared to the LTL model. Like the inter-domain scenario, we have evaluated using the NuSMV program the number of Binary Decision Diagram (BDD) nodes, the data size, and the time of both the LTL model and the BMC model. Figure 4.17 shows the comparison between the LTL and the BMC model in four different scenarios, where the number of hosts in the network initially is set to 5 and gradually increases up to 40 hosts. The time required for checking invariant (end-to-end reachability) in the case of LTL is 12 (ms) for five hosts, whereas, for LTL-BMC, it is 4(ms). The overall time of LTL-BMC is less compared to LTL as the number of hosts is increased in the model- Figure 4.17(a). The network width is used as a model bound for LTL-BMC; the state space for invariant checking is smaller. It can also be noticed that the number of BDD nodes in the BMC model is much less than in the LTL model - Figure 4.17(b). Moreover, the data size for the LTL model is more extensive in all cases. The BMC model reduced the data size with a reduction of 37% in the five hosts case and 60% in the 40 hosts case - Figure 4.17(c). The main reason for the reduction is that we simulate and account for all the states generated by the model in the LTL model. In contrast, in the BMC model, we set k=10, which means the model is simulated for several steps to find the counterexample. Therefore, this approach helps the model scale well as the network hosts increase within a single domain.

4.6.2 Composition Time Analysis

To show the effectiveness of INTPOL policy composition time, we compared our approach with some related works similar to ours. In our research, we consider model generation as the composition. The analogous process in PGA [125] is a D3 graph composition. Additionally, we also compare our composition time against research works that conducted performance analysis for Firewall rules such FIREMAN [156], [57]. PGA considers access control list (ACL) and service chaining policies. FIREMAN uses ACL policies for model composition. Covisor uses stateless applications such as routing, traffic monitoring, and load balancing. Halle *et al.* [57] uses distributed firewall anomaly detection. These research works use layers 2-4 policies or service chaining policies. INTPOL provides coverage for all the scenarios covered in these research works (see Figure 4.11). It is noticed that the LTL-based model checking for the Firewall proposed by Halle *et al.* [57] conducted performance analysis for 100-1000 Firewall rules and their algorithm used the full-scale model checking. The runtime for the policy composition is $\sim$ 8s for 500 rules and nearly 1000s for 1000 rules (see Table 4.3). This abnormality omits any abstraction, whereas the abstractions created using INTPOL intent rules and conducting the model checking directly on the data plane's Firewall rules enhance the scalability of the model checking and reduce the policy composition time. Using the full-scale model checking induces a high-performance penalty as the network grows. PGA [125] uses an eager graph composition based on end-point groups (EPGs). Although it is difficult to determine the exact number of rules, we estimate the number of rules based on the graph edges in PGA's graph. PGA conducted graph composition analysis for 500-3000 rules. The composition time is $\sim$ 18s for 500 rules, the 30s for 1000 rules, and $\sim$ 70s for 3000 rules. FIREMAN [156] performs symbolic model checking along all paths of an Access Control List (ACL). The research work uses ACL rules for composition analysis. FIREMAN conducted an empirical evaluation for 200-800 ACL rules. The model composition takes 0.1s for 500 rules, $\sim$ 0.15s for 1000 rules, and 0.2s for 2000 rules. The performance is better than the LTL model checking for Firewall conducted

Table 4.3: Policy Composition Time (s) Analysis Between INTPOL, Fireman [156], PGA [125], Distributed LTL Approach [57] and Covisor [75].

#Policies	**INTPOL**	**Fireman [156]**	**PGA [125]**	**Distributed LTL [57]**	**Covisor [75]**
500	0.01	0.1	18	8	0.05
1000	0.02	0.15	30	1000	0.1
1500	0.048	0.2	40	NA	0.13
2000	0.07	0.24	50	NA	0.15
2500	0.09	0.26	60	NA	0.18
3000	0.1	0.3	70	NA	0.25

by Halle *et al.* [57]. However, compact representation and bounded model checking can improve the model checking procedure as introduced in INTPOL. Covisor [75] utilizes symbolic model checking and parallel policy composition, which leads to improved policy composition 0.05s for 500 rules, $\sim$ 0.13s for 1500 rules, and $\sim$ 0.25s for 3000 rules. We observed that INTPOL performs the model generation for 500 rules in 0.02s and 1500 rules in 0.048s. The generation time for 3000 rules in the case of INTPOL is $\sim$ 0.1s. Although it is difficult to have a common baseline for comparing composition time with current research works due to a varying number of rules. We found the model representation is compact and scales well compared to similar research works, e.g., INTPOL takes 0.02s, compared to FIREMAN $\sim$ 0.15s for 1000 rules. Thus, INTPOL scales well, and the policy space increases compared to previous research work in policy composition.

4.6.3 Policy Conflict Checking for Large Network Scenario

In this section, our goal is to measure the scalability of INTPOL's conflict-checking models. For this purpose, the *Stanford topology* dataset [81] is utilized for the experimental analysis. The topology consists of 14 Operational Zones (OZ) routers,

Table 4.4: INTPOL Model Checking Framework Applied to Stanford Topology [81]. The Overhead of Generating a Model Using LTL-BMC is Lower Compared to LTL Full-Scale Framework.

#Intents	**Intent Gen. Time**	**LTL Nodes**	**LTL-BMC Nodes**
20	3.42	9768	3170
40	4.33	13936	3354
60	5.91	18713	3532
80	8.525	21610	4079
100	11.29	22939	4320

ten switches, and two backbone routers connected to OZs via switches. There are 1500 ACL rules and 757k forwarding entries in the simulated network. We aim to evaluate the performance of INTPOL in terms of policy translation scalability at the application plane and the data plan. Specifically, we present how the proposed BMC model will behave in a large network compared to the traditional LTL model for checking network invariants. Also, we evaluate the data plane conflict-checking scalability based on algorithm 5.

The Scalability of Conflict Checking at The Application Plane

Using the *Stanford Topology*, we start by analyzing the conflicts between the network policies at the application plane. The pre-specified policies, policies that specify reachability requirements between end-hosts and operational zones' routers, are converted into a formal model by checking the user requirements and network structure. The formal model checking occurs by converting these policies into network intents based on Algorithm 4. The network intents are checked for conflict, reachability, and end-to-end packet flow using network invariant property in the LTL & LTL-BMC formal models. The hosts in the *Stanford Topology* are selected to check for conflict-free policies between them. Each end-to-end packet reachability requirement between two randomly chosen hosts is inserted into the formal model as a network intent, and rules for checking conflict between user policies are added as a network invariant. For instance, when the number of intents (#Intents) is 20, we create 20 host-to-host reachability intents between randomly selected hosts in the network and check for a reachability conflict between them. The process we are conducting aims to place bounds on the (LTL-BMC) model such that the network diameter limits the conflict checking. The application plane conflict checking results are presented in Figure 4.18, where we show the number of intents and the equivalent number of flow rules generated by those intents in the *X-axis* and the conflict checking time on the *Y-axis*. Although the LTL & LTL-BMC models have a similar checking time, we can notice that when the number of intents and flow rules increases beyond a thousand, the LTL model time starts to rise significantly. The model's scalability can also be provided by measuring the resulting BDD nodes of each model. Table 4.4 shows the comparison between the nodes for the LTL & LTL-BMC models, and we notice how the number of BDD nodes in the LTL model is way higher than in the LTL-BMC model. The results in Table 4.4 are expected because we bound the diameter of node expansion by using the bounded model.

The Scalability of Conflict Checking at The Data Plane

We can observe from Table 4.4 & Figure 4.18 that the overhead associated with identifying conflicts at the application plane using the bounded model provides enormous advantages for early identification of conflicts, opportunities for traffic optimization, and policy composition at the application plane.

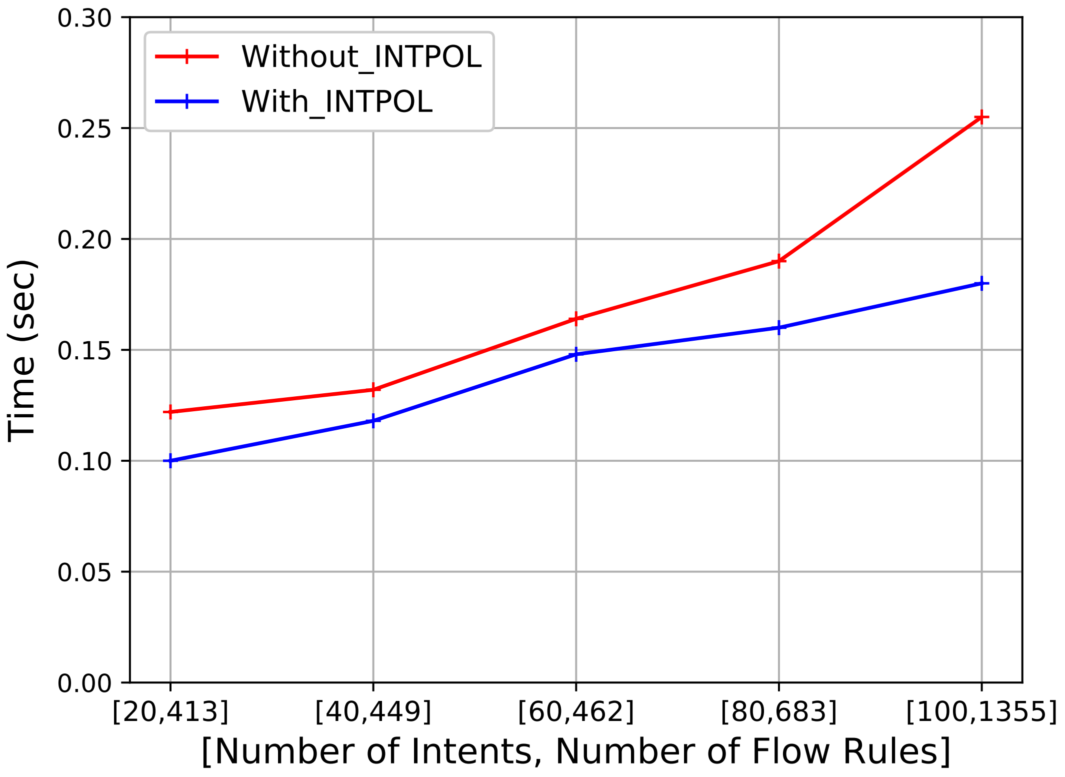


Figure 4.18: A Comparison Between The Application Plane Conflict Detection Time Using INTPOL Approach (Blue Line) and Without Using INTPOL (Red Line).

Considering the conflict at the data plane, we assume that the application plane does not handle conflicts (no formal model is present in the system). The intents are inserted into OpenFlow switches using the intent processing module described in Figure 4.6. We observe that the number of flow rules (#Flow Rules) generated for 20 intents is 413. This result is because, for realizing an intent, the SDN controller needs to add flows along the path between two hosts. For instance, in the case of the network example presented in Figure 4.2, if we add a host intent *(add-host-intent h1 h5)*, we need to insert four flow rules, rule 1 (flow between h1 and s1), rule 2 (flow between s1-s2), rule 3 (flow between s2-s3), and rule4 (flow between s3-h5). Thus in

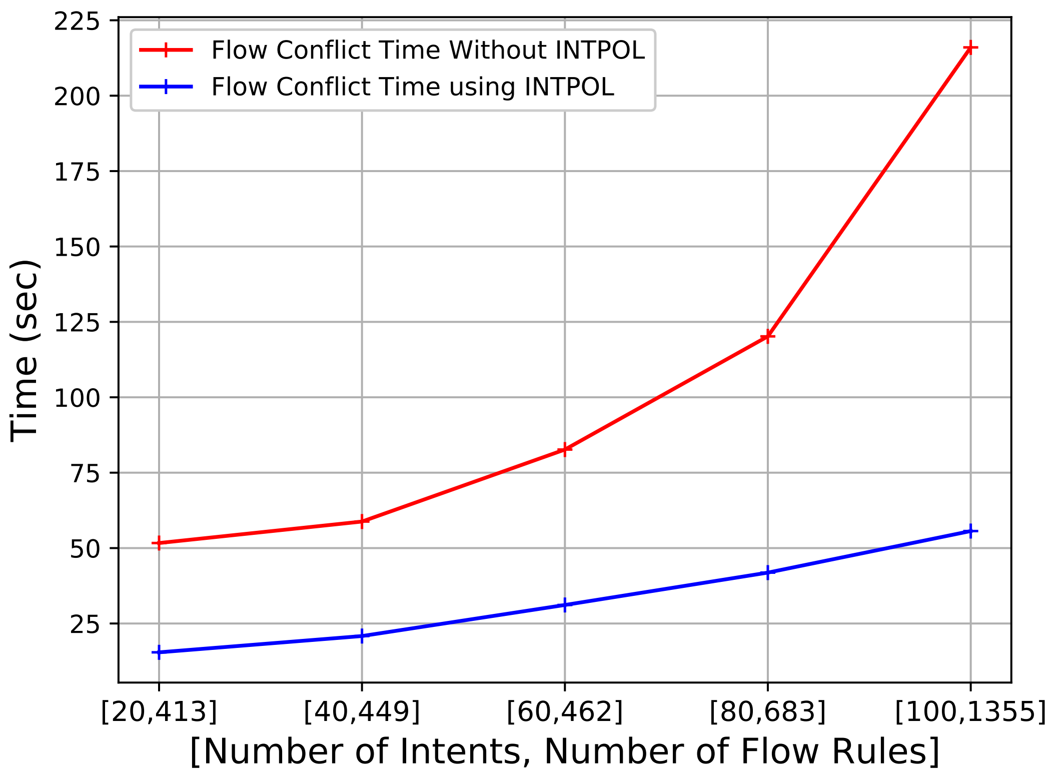


Figure 4.19: A Comparison Between The Flow Rule Conflict Detection Time With and Without Using INTPOL. The Data Plane Conflict Detection Time is Significantly Increased in The Absence of INTPOL Framework.

the case of hosts selected for a scalable topology such as described in our experiment, there can be multiple flow rules for each intent; these flow rules, in turn, conflict with each other.

The scalability of the INTPOL conflict checking experiment is presented in Figure 4.19. We can observe that conflict checking time is 51.68s for 413 flow rules while it only takes less than 5s for the same number of flow rules using the INTPOL approach. As we increase the number of intents from 20 to 100, the number of flow rules generated increases to 1355. The time required for checking conflicts among flow rules is also increased to 216.01s. This result is significantly greater than the time needed to check conflicts using the INTPOL framework because we perform the packet caching and policy composition at the application plane. INTPOL takes about 50s for conflict detection. Thus, if we identify and remove the conflicts between the intents at the application plane using the bounded formal model, the number of flow rules generated and the time needed to check the policy conflict will significantly decrease.

4.7 Conclusion and Discussion

In this chapter, we presented a multi-level network policy checking framework for SDN networks. INTPOL aims to reduce the complexity of network-side security policies and mission requirements in a multi-domain cloud environments by having a unified representation for the high-level application plane intents. Our framework utilizes bounded model checking (BMC) to limit policy conflicts checking at the application plane, which reduces the conflict detection time at the data plane. The presented solution can reduce the overhead of network policy conflict-checking significantly. We utilized case studies to show that INTPOL is generalized enough to handle scenarios such as Service Function Chaining (SFC), multiple network functions (Firewall, IDS, DPI), and hybrid networks involving traditional BGP routing and OpenFlow components.

This chapter has not covered the cases for stateful network functions. Some recent works such as Epinoia [147], and NetSMC [158] cover stateful network functions. The bounded formal model can provide coverage based on the network size/diameter. We have used popular means to establish the network bound (see Section III-D). This will cover a large set of policies. The framework scales well on a sizeable enterprise-grade network, as demonstrated by experiments performed on the Stanford topology, but fails to provide coverage for all cases of network policies. The policy composition used in some of the related works, such as Pyretic [127] and Covisor [75] approaches, identify parallelism opportunities that are not considered in INTPOL. We plan to explore complex network policies and network infrastructure, including stateful network functions with different cloud providers in the future.

Chapter 5

CONCLUSION AND FUTURE WORK

Cybersecurity is among the top concern for governments and privet sector around the world. The increasing number of cyber attacks threats posses a huge challenge on the IT administrators to ensure the confidentiality, integrity, and availability of the system services. The migration into cloud-based networks and the amount of automated services and applications makes the human-based analysis useless and weak. As human, we understand visual communication better than any other form of communication. Hence, the idea of Graphical Security Analysis (GrMS) and Attack Graph (AG) is introduced to fill the gap between human-based analysis and automated analysis. However, the AG by itself is not efficient in analyzing the large-scale systems such as data center networks, and it cannot provide an analysis of the high-level security policies that are implemented by the administrator.

This dissertation present a solutions to overcome the AG scalability issue and provide an optimization approach to solve the security policy conflict problem. Our goal is establish the foundation for scalable security state analysis for cloud-based systems, where the services are hosted on the cloud tenants and manual or human-based analysis is not possible. This will require the analysis approach to be fast, scalable, and readable to the administrator. The current problems with AG is that they are not scalable, difficult to generate, and hard to read. While AG deals with analyzing vulnerabilities at the lower level, the security policies that specify the functioning and security rules of the system at the high-level should also be analyzed. We explained how the high-level security policies can have conflict among them. We provided an approach that is based on the *Intents* to analyze the high-level security policies and detect the conflict early before it occur at the data plane level.

In Chapter 3, we presented *S3*, a novel framework to solve the AG scalability problem. S3 follows the *divide-and-conquer* concept to divide the large networking system into small manageable segments. Using the *Spearman correlation coefficient*, we form the segments based on the networking services' similarity and the distributed firewall (DFW) rules that separate the segments. Finally, we generate a sub-AG for every established segment and we merge all the sub-AGs to form the global AG. This way, we are able to reduce the complexity of the AG generation from $O(N^2)$ to $O((\frac{N}{K})^2)$, where N is the total number of vulnerable services and K is the total number of established segments. Moreover, our evaluation of the *S3* framework showed that we are able to reduce the AG generation time for a system with thousand of vulnerable services, reduce the AG density, maintain a conservative SDN controller consumption overhead, and detect cycles in the AG in a reasonable time.

In Chapter 4, we presented *INTPOL*, an intent-based framework that resolves high-level security requirements by utilizing the intents to translate the requirements into low-level flow rules. We showed in *INTPOL* how we utilize the *bounded model checking* (BMC) approach to confine the network verification within a certain bound. This way we are able to limit the conflict checking based on the number of domains or based on the maximum number a flow can traverse in the cloud system. *INTPOL* is evaluated and compared to other related work, where *INTPOL* is able to achieve a better composition, verification, and conflict resolution time. Furthermore, we were able to reduce the data plane conflict checking time and the number of conflicting rules. This resulted in helping the control plane to maintain a more stable, scalable, and efficient network management, which was an obstacle in the current literature that prevented the control plane from doing its main functionality and managing an efficient data plane.

5.1 Future Work

The work in this dissertation has paved the path for efficient and scalable cybersecurity analysis and modeling. After obtaining a scalable graph that contains a global view of the security situation, this graph can be used for further security automation processes. To achieve the optimal cybersecurity defense mechanism, we believe the introduction of Artificial Intelligence (AI-based) solutions are necessary due to the complexity and advanced computing capabilities in current systems. Cyber adversaries nowadays have access to super-computing and they are able to launch an automated cyber attacks towards any target. We have seen in past couple of years how *SolarWinds*, a major software provider, was hacked and the administrators did not know about the breach for a long time before it is discovered. The automated security analysis requires fetching the current security state in real-time in order to make accurate decision about deploying the appropriate defense mechanism.

Proactive Security

Proactive security is the key solution to ensure safety and accurate functioning of the system. This goal requires cyber-interaction between different system and defense components. The scalable AG generation presented in S3 framework can be utilized as an input to model a cybersecurity game between the attacker and the defender. This concept is known as Moving Target Defense (MTD) and it involves game-theory and AI-based planning and strategic movements. The scalable output from the AG can be used to make assumptions about the attacker's knowledge about the networking system. An advanced attacker with an advanced reconnaissance can obtain the same vulnerability information as the one obtained by the AG. Hence, the attacker has leverage over the defender and they may use this information to craft an advanced, dedicated attack to help them achieve their goal. Proactive security means the defender should predict the existence of these vulnerabilities before they occur or explored by the vulnerability scanning. One way to do that is by using the AG to build a new correlation models to predict *Zero-Day* vulnerability. Because the AG produces all the possible attack paths, combining multiple paths can lead to violating a security policy and result in a dangerous exploit.

Maintaining Scalable Security Analysis

The *INTPOL* work presented in Chapter 4 provided an approach to detect conflicts between security policies at the application plane before they are transmitted into the data plane. However, there is a lot of improvement opportunities in this domain with regards to maintaining a scalable security state in the cloud network. As mentioned earlier, considering the stateful network functions is a key to analyze the security policies since the dynamic cloud services and application are added and removed continuously. Also, these services are dependent upon the 'state' of the network flow. Therefore, a network verification and policy checking must consider the state of the network functions. In addition, the large cloud networks like data center networks have several controllers in the control domain. Unifying the application plane policy language between different cloud controllers will require involving a logic-based algorithms to enable reasoning between the input policies and the existing policies. This logic approach will be used next to help the network administrator find bugs in the network, discover black-holes or infinite loops, and build security policies that aid in achieving the proactive security objective. Because the security policies are important in specifying the access level, protect confidentiality, integrity, and availability of the resources, and help to enhance the system network performance.

REFERENCES

[1] A. Abhashkumar, J.-M. Kang, S. Banerjee, A. Akella, Y. Zhang, and W. Wu. Supporting diverse dynamic intent-based policies using janus. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, pages 296–309, 2017.

[2] M. Albanese, S. Jajodia, A. Pugliese, and V. Subrahmanian. Scalable analysis of attack scenarios. In *European Symposium on Research in Computer Security*, pages 416–433. Springer, 2011.

[3] M. Almukaynizi, E. Marin, E. Nunes, P. Shakarian, G. I. Simari, D. Kapoor, and T. Siedlecki. Darkmention: A deployed system to predict enterprise-targeted external cyberattacks. In *2018 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 31–36. IEEE, 2018.

[4] M. Almukaynizi, E. Nunes, K. Dharaiya, M. Senguttuvan, J. Shakarian, and P. Shakarian. Patch before exploited: An approach to identify targeted software vulnerabilities. In *AI in Cybersecurity*, pages 81–113. Springer, 2019.

[5] R. Amin, M. Reisslein, and N. Shah. Hybrid sdn networks: A survey of existing approaches. *IEEE Communications Surveys & Tutorials*, 20(4):3259–3306, 2018.

[6] P. Ammann, D. Wijesekera, and S. Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 217–224. ACM, 2002.

[7] M. T. Arashloo, Y. Koral, M. Greenberg, J. Rexford, and D. Walker. Snap: Stateful network-wide abstractions for packet processing. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 29–43, 2016.

[8] D. Arthur and S. Vassilvitskii. How slow is the k-means method? In *Proceedings of the twenty-second annual symposium on Computational geometry*, pages 144–153, 2006.

[9] D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006.

[10] Ayaka Koshibe. INTENT-based networking in ONOS. `https://wiki.onosproject.org/display/ONOS/Intent+Framework`, May, 24.2016 Accessed: March, 5,2021.

[11] D. Baca and K. Petersen. Prioritizing countermeasures through the countermeasure method for software security (cm-sec). In *International Conference on Product Focused Software Process Improvement*, pages 176–190. Springer, 2010.

[12] C. Baier and J.-P. Katoen. *Principles of model checking.* MIT press, 2008.

[13] F. Bannour, S. Souihi, and A. Mellouk. Distributed sdn control: Survey, taxonomy, and challenges. *IEEE Communications Surveys & Tutorials*, 20(1):333–354, 2017.

[14] R. Beckett, A. Gupta, R. Mahajan, and D. Walker. A general approach to network configuration verification. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 155–168, 2017.

[15] R. Beckett, X. K. Zou, S. Zhang, S. Malik, J. Rexford, and D. Walker. An assertion language for debugging sdn applications. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 91–96, 2014.

[16] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, et al. Onos: towards an open, distributed sdn os. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 1–6, 2014.

[17] S. Bhattacharya, S. Malhotra, and S. Ghsoh. A scalable representation towards attack graph generation. In *2008 1st International Conference on Information Technology*, pages 1–4. IEEE, 2008.

[18] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. 2003.

[19] P. Borril, M. Burgess, T. Craw, and M. Dvorkin. A promise theory perspective on data networks. *arXiv preprint arXiv:1405.2627*, 2014.

[20] D. Bringhenti, G. Marchetto, R. Sisto, S. Spinoso, F. Valenza, and J. Yusupov. Improving the formal verification of reachability policies in virtualized networks. *IEEE Transactions on Network and Service Management*, 18(1):713–728, 2020.

[21] L. Brunetta, M. Conforti, and G. Rinaldi. A branch-and-cut algorithm for the equicut problem. *Mathematical Programming*, 78(2):243–263, 1997.

[22] N. Cao, K. Lv, and C. Hu. An attack graph generation method based on parallel computing. In *International Conference on Science of Cyber Security*, pages 34–48. Springer, 2018.

[23] M. S. Castanho, C. K. Dominicini, M. Martinello, and M. A. Vieira. Chaining-box: A transparent service function chaining architecture leveraging bpf. *IEEE Transactions on Network and Service Management*, 2021.

[24] L. M. Castro and N. Paladi. Validation of sdn policies: a property-based testing perspective. *Procedia Computer Science*, 160:23–29, 2019.

[25] S. Ceri, G. Gottlob, and L. Tanca. What you always wanted to know about datalog (and never dared to ask). *IEEE transactions on knowledge and data engineering*, 1(1):146–166, 1989.

[26] R. Chandramouli and R. Chandramouli. Secure virtual network configuration for virtual machine (vm) protection. *NIST Special Publication*, 800:125B, 2016.

[27] S. Chauhan, M. Girvan, and E. Ott. Spectral properties of networks with community structure. *Physical Review E*, 80(5):056114, 2009.

[28] F. Chen, D. Liu, Y. Zhang, and J. Su. A scalable approach to analyzing network security using compact attack graphs. *Journal of Networks*, 5(5):543, 2010.

[29] Y. Chen, Z. Liu, Y. Liu, and C. Dong. Distributed attack modeling approach based on process mining and graph segmentation. *Entropy*, 22(9):1026, 2020.

[30] J. C. C. Chica, J. C. Imbachi, and J. F. B. Vega. Security in sdn: A comprehensive survey. *Journal of Network and Computer Applications*, 159:102595, 2020.

[31] A. Chowdhary, A. Alshamrani, and D. Huang. Supc: Sdn enabled universal policy checking in cloud network. In *2019 International Conference on Computing, Networking and Communications (ICNC)*, pages 572–576. IEEE, 2019.

[32] A. Chowdhary, D. Huang, A. Alshamrani, A. Sabur, M. Kang, A. Kim, and A. Velazquez. Sdfw: sdn-based stateful distributed firewall. *arXiv preprint arXiv:1811.00634*, 2018.

[33] A. Chowdhary, S. Pisharody, and D. Huang. Sdn based scalable mtd solution in cloud network. In *Proceedings of the 2016 ACM Workshop on Moving Target Defense*, pages 27–36. ACM, 2016.

[34] A. Chowdhary, A. Sabur, D. Huang, J. Kirby, and M. Kang. Object oriented policy conflict checking framework in cloud networks (oopc). *IEEE Transactions on Dependable and Secure Computing*, 2021.

[35] A. Chowdhary, A. Sabur, N. Vadnere, and D. Huang. Intent-driven security policy management for software-defined systems. *IEEE Transactions on Network and Service Management*, 2022.

[36] C.-J. Chung, P. Khatkar, T. Xing, J. Lee, and D. Huang. Nice: Network intrusion detection and countermeasure selection in virtual network systems. *IEEE transactions on dependable and secure computing*, 10(4):198–211, 2013.

[37] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. Nusmv 2: An opensource tool for symbolic model checking. In *International Conference on Computer Aided Verification*, pages 359–364. Springer, 2002.

[38] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. Nusmv: A new symbolic model verifier. In *International conference on computer aided verification*, pages 495–499. Springer, 1999.

[39] Cisco. Cisco Open SDN Controller. `https://www.cisco.com/c/en/us/products/cloud-systems-management/open-sdn-controller/index.html`, March 2020.

[40] E. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded model checking using satisfiability solving. *Formal methods in system design*, 19(1):7–34, 2001.

[41] J. Collings and J. Liu. An openflow-based prototype of sdn-oriented stateful hardware firewalls. In *2014 IEEE 22nd International Conference on Network Protocols*, pages 525–528. IEEE, 2014.

[42] D. Comer and A. Rastegatnia. Osdf: An intent-based software defined network programming framework. In *2018 IEEE 43rd Conference on Local Computer Networks (LCN)*, pages 527–535. IEEE, 2018.

[43] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2009.

[44] T. Dargahi, A. Caponi, M. Ambrosin, G. Bianchi, and M. Conti. A survey on the security of stateful sdn data planes. *IEEE Communications Surveys & Tutorials*, 19(3):1701–1725, 2017.

[45] R. Decker. Automating security policy enforcement with nsx service composer, 2015. Online: accessed 20 Jun 2020.

[46] L. Deri, M. Martinelli, T. Bujlow, and A. Cardigliano. ndpi: Open-source high-speed deep packet inspection. In *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 617–622. IEEE, 2014.

[47] L. Donetti and M. A. Munoz. Detecting network communities: a new systematic and efficient algorithm. *Journal of Statistical Mechanics: Theory and Experiment*, 2004(10):P10012, 2004.

[48] R. O. Duda, P. E. Hart, et al. *Pattern classification and scene analysis*, volume 3. Wiley New York, 1973.

[49] K. Durkota, V. Lisỳ, B. Bosanskỳ, and C. Kiekintveld. Optimal network security hardening using attack graph games. In *IJCAI*, pages 526–532, 2015.

[50] J. Erickson. Prediction: 80% Of Enterprise IT Will Move To The Cloud By 2025. url=Phttps://www.forbes.com/sites/oracle/2019/02/07/prediction-80-of-enterprise-it-will-move-to-the-cloud-by-2025/?sh=577772072a67, 2019. Online; accessed 11 Nov 2020.

[51] M. Farshbaf and M.-R. Feizi-Derakhshi. Multi-objective optimization of graph partitioning using genetic algorithms. In *2009 Third International Conference on Advanced Engineering Computing and Applications in Sciences*, pages 1–6. IEEE, 2009.

[52] S. K. Fayaz, T. Sharma, A. Fogel, R. Mahajan, T. Millstein, V. Sekar, and G. Varghese. Efficient network reachability analysis using a succinct control plane representation. In *12th USENIX Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 217–232, 2016.

[53] A. Fogel, S. Fung, L. Pedrosa, M. Walraed-Sullivan, R. Govindan, R. Mahajan, and T. Millstein. A general approach to network configuration analysis. In *12th USENIX Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, pages 469–483, 2015.

[54] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker. Frenetic: A network programming language. *ACM Sigplan Notices*, 46(9):279–291, 2011.

[55] D. Gorbatenko, A. Semenov, and S. Kochemazov. Unprovet: Using explicit constraint propagation to construct attack graphs. In *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1199–1204. IEEE, 2019.

[56] R. Greiner, R. Hayward, M. Jankowska, and M. Molloy. Finding optimal satisficing strategies for and-or trees. *Artificial Intelligence*, 170(1):19–58, 2006.

[57] S. Hallé, É. L. Ngoupé, R. Villemaire, and O. Cherkaoui. Distributed firewall anomaly detection through ltl model checking. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pages 194–201. IEEE, 2013.

[58] Y. Han, J. Li, D. Hoang, J.-H. Yoo, and J. W.-K. Hong. An intent-based network virtualization platform for sdn. In *2016 12th International Conference on Network and Service Management (CNSM)*, pages 353–358. IEEE, 2016.

[59] F. He and E. Oki. Main and secondary controller assignment with optimal priority policy against multiple failures. *IEEE Transactions on Network and Service Management*, 18(4):4391–4405, 2021.

[60] M. He, A. Varasteh, and W. Kellerer. Toward a flexible design of sdn dynamic control plane: An online optimization approach. *IEEE Transactions on Network and Service Management*, 16(4):1694–1708, 2019.

[61] Z. He, S. Deng, and X. Xu. Approximation algorithms for k-modes clustering. In *International Conference on Intelligent Computing*, pages 296–302. Springer, 2006.

[62] J. Homer, X. Ou, and D. Schmidt. A sound and practical approach to quantifying security risk in enterprise networks. *Kansas State University Technical Report*, pages 1–15, 2009.

[63] J. B. Hong and D. S. Kim. Performance analysis of scalable attack representation models. In *IFIP International Information Security Conference*, pages 330–343. Springer, 2013.

[64] J. B. Hong, D. S. Kim, C.-J. Chung, and D. Huang. A survey on the usability and practical applications of graphical security models. *Computer Science Review*, 26:1–16, 2017.

[65] J. B. Hong, D. S. Kim, and T. Takaoka. Scalable attack representation model using logic reduction techniques. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on*, pages 404–411. IEEE, 2013.

[66] H. Hu, W. Han, G.-J. Ahn, and Z. Zhao. Flowguard: building robust firewalls for software-defined networks. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 97–102. ACM, 2014.

[67] D. Huang, A. Chowdhary, and S. Pisharody. *Software-Defined Networking and Security: From Theory to Practice.* CRC Press, 2018.

[68] X. Huang, S. Cheng, K. Cao, P. Cong, T. Wei, and S. Hu. A survey of deployment solutions and optimization strategies for hybrid sdn networks. *IEEE Communications Surveys & Tutorials*, 21(2):1483–1507, 2018.

[69] M. Husák, J. Komárková, E. Bou-Harb, and P. Čeleda. Survey of attack projection, prediction, and forecasting in cyber security. *IEEE Communications Surveys & Tutorials*, 21(1):640–660, 2018.

[70] A. Ibrahim, S. Bozhinoski, and A. Pretschner. Attack graph generation for microservice architecture. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 1235–1242, 2019.

[71] K. Ingols, R. Lippmann, and K. Piwowarski. Practical attack graph generation for network defense. In *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*, pages 121–130. IEEE, 2006.

[72] A. S. Jacobs, R. J. Pfitscher, R. A. Ferreira, and L. Z. Granville. Refining network intents for self-driving networks. In *Proceedings of the Afternoon Workshop on Self-Driving Networks*, pages 15–21, 2018.

[73] S. Jajodia, S. Noel, and B. Oberry. Topological analysis of network attack vulnerability. In *Managing Cyber Threats*, pages 247–266. Springer, 2005.

[74] Y. Jang, S. P. Chung, B. D. Payne, and W. Lee. Gyrus: A framework for user-intent monitoring of text-based networked applications. In *NDSS*, 2014.

[75] X. Jin, J. Gossels, J. Rexford, and D. Walker. Covisor: A compositional hypervisor for software-defined networks. In *12th USENIX Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, pages 87–101, 2015.

[76] P. Johnson. With The Public Clouds Of Amazon, Microsoft And Google, Big Data Is The Proverbial Big Deal. url=https://www.forbes.com/sites/johnsonpierr/2017/06/15/with-the-public-clouds-of-amazon-microsoft-and-google-big-data-is-the-proverbial-big-deal/409452d02ac3, 2017. Online; accessed 3 Mar 2019.

[77] Jonathan Hart. Basic ONOS Tutorial. `https://wiki.onosproject.org/display/ONOS/SDN-IP`, December, 04.2014 Accessed: June, 5,2020.

[78] Juniper Networks. Software-Defined Networking, Network Management, and Operations. `https://www.juniper.net/us/en/products-services/management-operations-sdn/`, March, 2020 Accessed: May, 2,2020.

[79] M. M. P. Kallehbasti, M. G. Rossi, and L. Baresi. On how bit-vector logic can help verify ltl-based specifications. *IEEE Transactions on Software Engineering*, 2020.

[80] K. Kaynar and F. Sivrikaya. Distributed attack graph generation. *IEEE Transactions on Dependable and Secure Computing*, 13(5):519–532, 2016.

[81] P. Kazemian, G. Varghese, and N. McKeown. Header space analysis: Static checking for networks. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation ({NSDI} 12)*, pages 113–126, 2012.

[82] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(2):291–307, 1970.

[83] A. R. Khakpour and A. X. Liu. An information-theoretical approach to high-speed flow nature identification. *IEEE/ACM transactions on networking*, 21(4):1076–1089, 2012.

[84] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey. Veriflow: Verifying network-wide invariants in real time. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation ({NSDI} 13)*, pages 15–27, 2013.

[85] J. M. Kizza. Introduction to computer network vulnerabilities. In *Guide to Computer Network Security*, pages 87–103. Springer, 2015.

[86] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2014.

[87] Z. Latif, K. Sharif, F. Li, M. M. Karim, S. Biswas, M. Shahzad, and S. P. Mohanty. Dolphin: Dynamically optimized and load balanced path for inter-domain sdn communication. *IEEE Transactions on Network and Service Management*, 18(1):331–346, 2020.

[88] J. Lee, H. Lee, and H. P. In. Scalable attack graph for risk assessment. In *2009 International Conference on Information Networking*, pages 1–5. IEEE, 2009.

[89] S. Lee, S. Woo, J. Kim, V. Yegneswaran, P. Porras, and S. Shin. Audisdn: Automated detection of network policy inconsistencies in software-defined networks.

[90] V. Lempitsky, P. Kohli, C. Rother, and T. Sharp. Image segmentation with a bounding box prior. In *2009 IEEE 12th international conference on computer vision*, pages 277–284. IEEE, 2009.

[91] H. Li, K. Chen, T. Pan, Y. Zhou, K. Qian, K. Zheng, B. Liu, P. Zhang, Y. Tang, and C. Hu. Cora: Conflict razor for policies in sdn. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 423–431. IEEE, 2018.

[92] Y. Li and M. Chen. Software-defined network function virtualization: A survey. *IEEE Access*, 3:2542–2553, 2015.

[93] Y. Li, X. Yin, Z. Wang, J. Yao, X. Shi, J. Wu, H. Zhang, and Q. Wang. A survey on network verification and testing with formal methods: Approaches and challenges. *IEEE Communications Surveys & Tutorials*, 21(1):940–969, 2018.

[94] P. Lin, J. Hart, U. Krishnaswamy, T. Murakami, M. Kobayashi, A. Al-Shabibi, K.-C. Wang, and J. Bi. Seamless interworking of sdn and ip. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, pages 475–476, 2013.

[95] X. Liu. A network attack path prediction method using attack graph. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–8, 2020.

[96] Y. Liu and H. Man. Network vulnerability assessment using bayesian networks. In *Data Mining, Intrusion Detection, Information Assurance, and Data Networks Security 2005*, volume 5812, pages 61–71. International Society for Optics and Photonics, 2005.

[97] J. W. Lloyd. *Foundations of logic programming.* Springer Science & Business Media, 2012.

[98] W. W. Lo, S. Layeghy, M. Sarhan, M. Gallagher, and M. Portmann. E-graphsage: A graph neural network based intrusion detection system. *arXiv preprint arXiv:2103.16329*, 2021.

[99] C. Lorenz, V. Clemens, M. Schrötter, and B. Schnor. Continuous verification of network security compliance. *IEEE Transactions on Network and Service Management*, 2021.

[100] L. Lu, R. Safavi-Naini, M. Hagenbuchner, W. Susilo, J. Horton, S. L. Yong, and A. C. Tsoi. Ranking attack graphs with graph neural networks. In *International Conference on Information Security Practice and Experience*, pages 345–359. Springer, 2009.

[101] K. Madsen and H. Schjær-Jacobsen. Linearly constrained minimax optimization. *Mathematical Programming*, 14(1):208–223, 1978.

[102] R. McLean. A hacker gained access to 100 million capital one credit card applications and accounts. `https://www.cnn.com/2019/07/29/business/capital-one-data-breach/index.html`, July-2019. Online; accessed 17 Oct 2019.

[103] M. A. McQueen, W. F. Boyer, M. A. Flynn, and G. A. Beitel. Quantitative cyber risk reduction estimation methodology for a small scada control system. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)*, volume 9, pages 226–226. IEEE, 2006.

[104] J. Medved, R. Varga, A. Tkacik, and K. Gray. Opendaylight: Towards a model-driven sdn controller architecture. In *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, pages 1–6. IEEE, 2014.

[105] Metzler, Jim. Survey shows growing interest in sdn, where and how companies might deploy the tech, 2016. [Online; accessed 3 Mar 2019].

[106] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications surveys & tutorials*, 18(1):236–262, 2015.

[107] O. Mjihil, D. Huang, and A. Haqiq. Improving attack graph scalability for the cloud through sdn-based decomposition and parallel processing. In *International Symposium on Ubiquitous Networking*, pages 193–205. Springer, 2017.

[108] N. Mojidra. Stateful vs. Stateless Firewalls. url=https://www.cybrary.it/0p3n/stateful-vs-stateless-firewalls/, 2016. Online; accessed 20 Sep 2018.

[109] NIST. National Vulnerability Database. url=https://nvd.nist.gov/. Online; accessed 3 Mar 2019.

[110] S. Noel and S. Jajodia. Metrics suite for network attack graph analytics. In *Proceedings of the 9th Annual Cyber and Information Security Research Conference*, pages 5–8, 2014.

[111] S. Noel, S. Jajodia, B. O'Berry, and M. Jacobs. Efficient minimum-cost network hardening via exploit dependency graphs. In *19th Annual Computer Security Applications Conference, 2003. Proceedings.*, pages 86–95. IEEE, 2003.

[112] A. O'DRISCOLL. 25+ cyber security vulnerability statistics and facts of 2022. `https://www.comparitech.com/blog/information-security/cybersecurity-vulnerability-statistics/#:~:text=The%20number%20of%20new%20vulnerabilities,already%20seen%20an%20additional%20700.`, May-2022. Online; accessed 7 July 2022.

[113] ONOS Wiki. Basic ONOS Tutorial. `https://wiki.onosproject.org/display/ONOS/Basic+ONOS+Tutorial`, August 2020.

[114] X. Ou, W. F. Boyer, and M. A. McQueen. A scalable approach to attack graph generation. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 336–345. ACM, 2006.

[115] X. Ou, S. Govindavajhala, and A. W. Appel. Mulval: A logic-based network security analyzer. In *USENIX Security Symposium*, pages 8–8. Baltimore, MD, 2005.

[116] Palantir. Restricting SMB-based lateral movement in a Windows environment. url=https://blog.palantir.com/restricting-smb-based-lateral-movement-in-a-windows-environment-ed033b888721, 7-8-2020. Online; accessed 3 May 2021.

[117] L. Pang, C. Yang, D. Chen, Y. Song, and M. Guizani. A survey on intent-driven networks. *IEEE Access*, 8:22862–22873, 2020.

[118] R. S. Parpinelli, H. S. Lopes, and A. A. Freitas. Data mining with an ant colony optimization algorithm. *IEEE transactions on evolutionary computation*, 6(4):321–332, 2002.

[119] L. Pedrosa, R. Iyer, A. Zaostrovnykh, J. Fietz, and K. Argyraki. Automated synthesis of adversarial workloads for network functions. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 372–385, 2018.

[120] D. Pelleg, A. W. Moore, et al. X-means: Extending k-means with efficient estimation of the number of clusters. In *Icml*, volume 1, pages 727–734, 2000.

[121] J. G. V. Pena and W. E. Yu. Development of a distributed firewall using software defined networking technology. In *Information Science and Technology (ICIST), 2014 4th IEEE International Conference on*, pages 449–452. IEEE, 2014.

[122] M. Pham and D. B. Hoang. Sdn applications-the intent-based northbound interface realisation for extended applications. In *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, pages 372–377. IEEE, 2016.

[123] D. L. Pipkin. *Information security: protecting the global enterprise.* Prentice-Hall, Inc., 2000.

[124] S. Pisharody, J. Natarajan, A. Chowdhary, A. Alshalan, and D. Huang. Brew: A security policy analysis framework for distributed sdn-based cloud environments. *IEEE transactions on dependable and secure computing*, 2017.

[125] C. Prakash, J. Lee, Y. Turner, J.-M. Kang, A. Akella, S. Banerjee, C. Clark, Y. Ma, P. Sharma, and Y. Zhang. Pga: Using graphs to express and automatically reconcile network policies. *ACM SIGCOMM Computer Communication Review*, 45(4):29–42, 2015.

[126] A. Rafiq, M. Afaq, and W.-C. Song. Intent-based networking with proactive load distribution in data center using ibn manager and smart path manager. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–18, 2020.

[127] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker. Modular sdn programming with pyretic. *Technical Report of USENIX*, 2013.

[128] C. Ren, H. Li, Y. Li, Y. Wang, H. Xiang, et al. On efficient service function chaining in hybrid software defined networks. *IEEE Transactions on Network and Service Management*, 2021.

[129] S. Revathi and A. Malathi. A detailed analysis on nsl-kdd dataset using various machine learning techniques for intrusion detection. *International Journal of Engineering Research & Technology (IJERT)*, 2(12):1848–1853, 2013.

[130] A. Sabur, A. Chowdhary, D. Huang, M. Kang, A. Kim, and A. Velazquez. S3: A dfw-based scalable security state analysis framework for large-scale data center networks. In *22nd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2019)*, pages 473–485, 2019.

[131] D. Satasiya, R. Raviya, and H. Kumar. Enhanced sdn security using firewall in a distributed scenario. In *Advanced Communication Control and Computing Technologies (ICACCCT), 2016 International Conference on*, pages 588–592. IEEE, 2016.

[132] S. Sengupta, A. Chowdhary, D. Huang, and S. Kambhampati. Moving target defense for the placement of intrusion detection systems in the cloud. In *International Conference on Decision and Game Theory for Security*, pages 326–345. Springer, 2018.

[133] S. Sengupta, A. Chowdhary, A. Sabur, A. Alshamrani, D. Huang, and S. Kambhampati. A survey of moving target defenses for network security. *IEEE Communications Surveys & Tutorials*, 22(3):1909–1941, 2020.

[134] H.-W. Shen and X.-Q. Cheng. Spectral methods for the detection of network community structure: a comparative analysis. *Journal of Statistical Mechanics: Theory and Experiment*, 2010(10):P10020, 2010.

[135] J. Shen, J. Xia, S. Dong, X. Zhang, and K. Fu. Universal feature extraction for traffic identification of the target category. *PloS one*, 11(11):e0165993, 2016.

[136] J. Sherry, S. Ratnasamy, and J. S. At. A survey of enterprise middlebox deployments. 2012.

[137] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing. Automated generation and analysis of attack graphs. In *null*, page 273. IEEE, 2002.

[138] H. D. Simon. Partitioning of unstructured problems for parallel processing. *Computing systems in engineering*, 2(2):135–148, 1991.

[139] B. Tian, X. Zhang, E. Zhai, H. H. Liu, Q. Ye, C. Wang, X. Wu, Z. Ji, Y. Sang, M. Zhang, et al. Safely and automatically updating in-network acl configurations with intent language. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 214–226. 2019.

[140] V. Varadharajan, K. Karmakar, U. Tupakula, and M. Hitchens. A policy-based security architecture for software-defined networks. *IEEE Transactions on Information Forensics and Security*, 14(4):897–912, 2018.

[141] Y. Velner, K. Alpernas, A. Panda, A. Rabinovich, M. Sagiv, S. Shenker, and S. Shoham. Some complexity results for stateful network verification. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 811–830. Springer, 2016.

[142] S. Vissicchio, L. Vanbever, and O. Bonaventure. Opportunities and research challenges of hybrid software defined networks. *ACM SIGCOMM Computer Communication Review*, 44(2):70–75, 2014.

[143] Y. Vizel, G. Weissenbacher, and S. Malik. Boolean satisfiability solvers and their applications in model checking. *Proceedings of the IEEE*, 103(11):2021–2035, 2015.

[144] VMWare. Five critical requirements for internal firewalling in the data center why traditional perimeter firewalls are becoming obsolete for protecting east-west traffic. March 2020.

[145] R. Wallner and R. Cannistra. An sdn approach: quality of service using big switchs floodlight open-source controller. *Proceedings of the Asia-Pacific Advanced Network*, 35(14-19):10–7125, 2013.

[146] A. Wang, L. Jia, W. Zhou, Y. Ren, B. T. Loo, J. Rexford, V. Nigam, A. Scedrov, and C. Talcott. Fsr: Formal analysis and implementation toolkit for safe interdomain routing. *IEEE/ACM Transactions on Networking*, 20(6):1814–1827, 2012.

[147] H. Wang, P. Sharma, F. Ahmed, J.-M. Kang, C. Qian, and M. Yannakakis. Epinoia: Intent checker for stateful networks. In *2021 International Conference on Computer Communications and Networks (ICCCN)*, pages 1–9. IEEE, 2021.

[148] L. Wang, T. Islam, T. Long, A. Singhal, and S. Jajodia. An attack graph-based probabilistic security metric. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 283–296. Springer, 2008.

[149] Wikipedia contributors. Compact space — Wikipedia, the free encyclopedia, 2020. [Online; accessed 2-November-2020].

[150] A. Xie, Z. Cai, C. Tang, J. Hu, and Z. Chen. Evaluating network security with two-layer attack graphs. In *2009 Annual Computer Security Applications Conference*, pages 127–136. IEEE, 2009.

[151] D. Xu and Y. Tian. A comprehensive survey of clustering algorithms. *Annals of Data Science*, 2(2):165–193, 2015.

[152] S. H. Yeganeh, M. Eftekhar, Y. Ganjali, R. Keralapura, and A. Nucci. Cute: Traffic classification using terms. In *2012 21st International Conference on Computer Communications and Networks (ICCCN)*, pages 1–9. IEEE, 2012.

[153] S. Yi, Y. Peng, Q. Xiong, T. Wang, Z. Dai, H. Gao, J. Xu, J. Wang, and L. Xu. Overview on attack graph generation and visualization technology. In *2013 International Conference on Anti-Counterfeiting, Security and Identification (ASID)*, pages 1–6. IEEE, 2013.

[154] M. Yousefi, N. Mtetwa, Y. Zhang, and H. Tianfield. A reinforcement learning approach for attack graph analysis. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 212–217. IEEE, 2018.

[155] B.-t. YUAN, Z.-l. PAN, and S. Fan. A review on network attack graph technology. *DEStech Transactions on Engineering and Technology Research*, (ecar), 2018.

[156] L. Yuan, H. Chen, J. Mai, C.-N. Chuah, Z. Su, and P. Mohapatra. Fireman: A toolkit for firewall modeling and analysis. In *2006 IEEE Symposium on Security and Privacy (S&P'06)*, pages 15–pp. IEEE, 2006.

[157] R. Yuan, Z. Li, X. Guan, and L. Xu. An svm-based machine learning method for accurate internet traffic classification. *Information Systems Frontiers*, 12(2):149–156, 2010.

[158] Y. Yuan, S.-J. Moon, S. Uppal, L. Jia, and V. Sekar. Netsmc: A custom symbolic model checker for stateful network verification. In *17th USENIX Symposium on Networked Systems Design and Implementation ({NSDI} 20)*, pages 181–200, 2020.

[159] A. Zeineddine and W. El-Hajj. Stateful distributed firewall as a service in sdn. In *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, pages 212–216. IEEE, 2018.

[160] S. Zhang, X. Ou, and J. Homer. Effective network vulnerability assessment through model abstraction. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 17–34. Springer, 2011.

[161] Y. Zhao, P. Zhang, Y. Wang, and Y. Jin. Troubleshooting data plane with rule verification in software-defined networks. *IEEE Transactions on Network and Service Management*, 15(1):232–244, 2017.

[162] R. Zhuang, S. Zhang, S. A. DeLoach, X. Ou, and A. Singhal. Simulation-based approaches to studying effectiveness of moving-target network defense. In *National symposium on moving target research*, volume 246, 2012.

[163] M. Zoure, T. Ahmed, and L. Réveillére. Network services anomalies in nfv: Survey, taxonomy, and verification methods. *IEEE Transactions on Network and Service Management*, 2022.

APPENDIX A

A PERMISSION FROM CO-AUTHORS

The author of this dissertation states and confirms that he has obtained permission from all co-authors of the previously published work and that they have all granted him permission to include the research work in this dissertation.