Multi-cloud Connectivity Provisioning with Security Attributes

by

Liran Tao

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2023

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

In this thesis, we detail the following work: (i) the design and development of graph-based cloud management software with multiple microservices, including one that provides easy connectivity provisioning in the multi-cloud environment, and (ii) a detailed analysis of security attributes in the multi-cloud environment.

Acknowledgements

I would like to thank my supervisor, Professor Mahesh V. Tripunitara, for his vision in my research project and for his constant guidance and support during my Master's research work.

I would like to thank Rahul Punchhi, Alireza Lotfi, and Jialing Song, who contributed to the work of the graph-based cloud management system mentioned in this thesis.

Portions of the work in this thesis were carried out under the auspices of a Mitacs project in which the industry partners were Nelu Mihai and mosaixsoft.ca.

Dedication

This is dedicated to my parents Wenli and Hangang, and my long-time partner Jiahao.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Recent years have witnessed exponential growth in the adoption of cloud services. From SMEs to tech giants and governments, cloud service has become an integral part of supporting organizations' daily operations. Among all the cloud service providers (CSPs), Amazon Web Service (AWS), Google Cloud Platform (GCP), and Microsoft Azure (Azure) are some of the biggest players in the cloud computing market.

The idea of multi-cloud is associated with utilizing services from multiple CSPs simultaneously for a system or an application. It is estimated that more than 85% of the organizations have shifted their operations in multi-cloud environments [15]. Multi-cloud paradigm offers extra benefits to users and enterprises such as mitigating the risk of vendor lock-in, optimizing performances and costs, ensuring high availability, ensuring data sovereignty, and offering services that are particular to one CSP but not offered elsewhere [27][17][26][30].

Due to these reasons, given a distributed application, an enterprise may want some of its components on one cloud and some other components on other clouds as a result of business decisions such as maximizing the benefits of services provided by different clouds. In this scenario, provisioning connectivity to interconnect these clouds, which requires expert knowledge of multiple clouds involved, becomes an essential part of managing and maintaining such distributed applications. Also, under certain scenarios, enterprises may want to migrate applications from one cloud to another cloud due to economical or compliance reasons. One of the biggest challenges of managing multi-cloud applications or cloud migrations is to achieve consistent security objectives across all clouds involved, but this is often difficult to achieve because the security features offered by different CSPs differ in their syntax and semantics.

This research work aims at exploring the provision of connectivity with security attributes in the multi-cloud setting. Specifically, in chapter 2, we formalize the syntax and semantics of security features offered by each CSP and try to address the following questions:

1. Understand the expressive power of the portion of the security configuration that pertains to connectivity, i.e., given multiple CSPs, does one CSP's security language syntax offer more than that of other CSPs'?
2. Given a security configuration for one CSP, does there exist an equivalent security configuration in another CSP? If not, what is the tightest/best security configuration in another CSP?
3. What is an efficient algorithm to translate one CSP's security configuration to the best security configuration in another CSP?

In chapter 3, we describe the design and development of graph-based software that helps the management of multi-cloud applications and the provision of connectivity in multi-cloud environments.

1.2 Definition

Definition 1 *A cloud is one of the cloud service providers (CSPs), namely Amazon Web Service (AWS), Google Cloud Platform (GCP), or Microsoft Azure (Azure).*

Definition 2 *A subnet is a segmented piece of a larger network, i.e., a network inside a larger network.*

Definition 3 *A private network is a computer network that uses a private address space of IP addresses.*

A private address space is a subset of the following address ranges:

- 10.0.0.0 - 10.255.255.255
- 172.16.0.0 - 172.31.255.255
- 192.168.0.0 - 192.168.255.255

Resources such as computers inside a private network will be assigned private IP addresses.

Definition 4 *A private IP address is an IP address that is within the private address space.*

Definition 5 *A CIDR (Classless Inter-Domain Routing) Block defines a range of consecutive IPv4 addresses. A CIDR block is of the form a.b.c.d/n such that a, b, c, d each is a number of up to three digits, 0-255, and n is a number from 0-32 which represents the network mask.*

Definition 6 *Two CIDR blocks, a.b.c.d/n and e.f.g.h/m are Non-overlapping if and only if the conjunction of two sets of IP addresses defined by the two CIDR blocks is an empty set.*

Definition 7 *A Virtual Private Cloud (VPC) is a private network that is hosted on a cloud. Formally, a VPC is defined as a 2-tuple:*

$$VPC = \langle cloud, cidrBlock \rangle$$

A VPC resembles a private network whose CIDR block defines the private address space within the VPC. A VPC may be partitioned into smaller non-overlapping subnets.

Definition 8 *A Virtual Machine (VM) is a computer resource that is defined by a 3-tuple, namely*

$\langle$`ipAddress, subnet, securityAttribute?`$\rangle$*, where ipAddress* $\in$ *subnet and ? means optional.*

A VM may be *deployed* in a cloud. What this means is that an instance of a computing resource is reserved within a cloud. A deployed VM can also be attached to a subnet of a VPC and therefore, a VM that is attached to a VPC can have a private IP address that is within the VPC CIDR block. A VM is sometimes referred as an *instance* in the cloud.

Definition 9 *An IP Packet, pk_{ip}, is a 7-tuple:*

$$(ip_{src}, ip_{dest}, protocol, port_{src}?, port_{dest}?, type_{icmp}?, code_{icmp}?)$$

where ? signifies optional component.

Definition 10 *An instance of connectivity between two VPCs is a function of exactly two VPCs, which may not be distinct from one another. We say that there exists an instance of connectivity between two VPCs if and only if, without additional security attributes, any VMs in one VPC is able to send and receive ip packet to and from any VMs in the other VPC.*

Definition 11 *An instance of connectivity between two VMs is a function of exactly two VMs, which may not be distinct from one another. There exists an instance of connectivity between two VMs if and only if one VM is able to send and receive ip packet to and from the other VM via the private IP address of the other VM.*

Definition 12 *A stateful security configuration is one such that if an ip packet pk_{ip} is allowed or denied by the stateful security configuration in one direction, then the packet pk_{res} generated in response to pk_{ip} is allowed in the opposite direction regardless of the security configuration in the opposite direction.*

Definition 13 *A stateless security configuration is one such that the packets pk_{res} generated in response to an allowed ip packet pk_{ip} in one direction are subject to the security configuration for pk_{res} in the opposite direction.*

1.3 Related Work

Over the last decade, many studies explored the multi-cloud paradigm and proposed applications based on multiple clouds, especially for the ones that pertain to using multiple clouds to preserve network or data security and privacy. For example, DepSky is a multi-cloud storage system that addresses the risks that are otherwise common in single-cloud storage, namely, the loss of availability of data, data loss, data corruption, and the loss of privacy [14]. It achieves these by combining Byzantine quorum system protocols, cryptography, secret sharing, and erasure code. In addition, it reduces the coupling between users and CSPs so that vendor lock-in can be eliminated. To prevent the loss of data availability, DepSky replicates data and stores them in multiple clouds such that in the event of a subset of CSPs' service outage, the data remains accessible. DepSky also uses erasure codes to reduce the amount of replication needed, thus reducing storage costs. By using a set of efficient Byzantine quorum system protocols, DepSky offers protection against data loss and the corruption of data [14].

Stefanov & Shi proposed a multi-cloud protocol named Multi-Cloud Oblivious Storage (MCOS) which aimed at leveraging multiple clouds to protect data confidentiality and anonymity in the public cloud computing setting [28]. This protocol utilizes 2 non-colluding clouds, but it can be further expanded to include more CSPs [28]. In the 2-cloud oblivious storage implementation, the client shares a symmetric secret key with each cloud which will be used for encryption and decryption. The client initializes the storage by partitioning the original Oblivious Random Access Memory (ORAM) of size N into $O(\sqrt{N})$ partitions that are themselves ORAMs [28]. Each partition contains $\log N$ levels in a way that is similar to a binary tree data structure. Each level can contain data blocks or dummy (empty) blocks, and which level resides in which cloud can change as partitions are shuffled between clouds during a read or a write operation later on. Each data block is then placed randomly into a partition and assigned a random offset from the top level, i.e., the largest level, of that partition. To keep track the exact location of each data block, the client stores locally a position map that contains a tuple (*partition*, *level*, *offset*) for each data block. The tuple reveals information about the data block's exact location in a particular partition [28].

Another multi-cloud paradigm, Prio, proposed by Gibbs and Boneh, is a privacy-preserving system for collecting aggregate statistics [16]. Prio is proven to be robust, scalable, and can be deployed in the multi-cloud setting. Prio achieves the goal of *correctness*, *privacy*, *robustness*, and *efficiency*. Using Prio, if all clouds are honest, then the clouds collectively compute the correct aggregate statistics (*correctness*). If at least one cloud is honest, then Prio protects users' *privacy*. Malformed data from malicious clients have limited impact only (*robustness*). Furthermore, Prio is able to handle a high volume of user data (*efficiency*) [16].

We notice that these studies mainly focused on leveraging the multi-cloud paradigm to enforce

data confidentiality and preserve data privacy, while little attention was directed to achieving multi-cloud connectivity in general. Furthermore, these studies did not consider cloud-managed security solutions either, while this research work focuses on multi-cloud connectivity and security.

Recently, Yeganeh *et al.* conducted a comprehensive study on the characterization of multi-cloud connectivity involving major public cloud vendors. In this study, they compare three major multi-cloud connectivity paradigm, namely (1) transit provider-based best-effort public Internet (BEP), (2) third-party provider-based private (TPP) connectivity, and (3) CP-based private (CPP) connectivity [30]. The study found that CPP-based multi-cloud connectivity has low network latency and is more stable compared to BEP and TPP-based multi-cloud connectivity. It also showed that CPP-based connectivity has higher throughput and less variation [30]. These findings encouraged this research as it backed up the fact that cloud-managed multi-cloud connectivity provisioning, which will be covered in chapter 2, is indeed a feasible solution and exhibits certain benefits compared to other multi-cloud connectivity provisioning paradigms.

Chapter 2

Multi-cloud Security Attributes

Security is at the core of any cloud application, be it a single-cloud or a multi-cloud application. Even though VPCs provide network isolation, the need for Internet access for resources inside VPCs/VNets might expose the resources to malicious network traffic. Furthermore, poorly implemented security configurations might also impose a false sense of protection. It is therefore of paramount importance to leverage the cloud-managed security solution of each cloud properly to provide confidentiality, integrity, and availability for VPC networks in multi-cloud environments.

In this chapter, we examine the security attributes of clouds that pertain to connectivity. In section 2.1, we give an overview of various connectivity-related network security features offered by different cloud service providers (CSPs) Based on these features, section 2.2 formally defines the syntax and semantics of cloud security pertaining to connectivity that set the tone for the following sections. Section 2.3 presents formal proofs of equivalence for connectivity security attributes between clouds. In section 2.4, we explore and expand our understanding of the expressive power problem for the portion of the security configuration that pertains to connectivity. Finally, in section 2.5, we provide efficient algorithms that facilitate the migration from one cloud to another.

2.1 Cloud Securities for Connectivity

In this section, we give an overview of the kinds of security features offered by CSPs, namely AWS, GCP, and Azure.

Table 2.1: AWS Security Group Configuration Example

Inbound rules				
Type	Protocol	Port range	Source	Description (optional)
SSH	TCP	22	1.1.1.1/32	Allow SSH connections from IPv4 address 1.1.1.1 only.
HTTPS	TCP	443	0.0.0.0/0	Allow all incoming traffic to port 443
Custom TCP	TCP	5432	10.0.0.0/24	Allow traffic from private subnet with CIDR block 10.0.0.0/24 to port 5432
Outbound rules				
Type	Protocol	Port range	Destination	Description (optional)
All traffic	All	All	0.0.0.0/0	All outbound traffic

2.1.1 Security features in AWS VPCs

Security groups in AWS VPC allow ingress or egress traffic specified by the inbound and outbound rules at the resource level [4]. An EC2 instance[1], for example, can be associated with *one or more security groups*, and the inbound and outbound network traffic of the EC2 instance is controlled by the rules defined within these security groups. Every resource in an AWS VPC must be associated with one or more security groups. The security group supports allow rules only, i.e., the security group employs a *default deny* strategy. Table 2.1 presents an example of a security group configuration in AWS which allows SSH connection from IP address 1.1.1.1, port 443 that opens to the public internet, and connections to port 5432 over TCP protocol from any resource within the private subnet 10.0.0.0/24. This security group also allows any outbound traffic.

Each VPC in AWS has a default security group with the following security rules:

- Allow all inbound traffic from resources that are assigned to the same security group
- Allow all outbound IPv4 and IPv6 traffic.

Security groups in AWS are **stateful** 12.

Network access control lists (NACL) allow or deny ingress or egress traffic specified by the inbound and outbound rules at the subnet level, i.e., NACL rules apply to all resources inside the subnet to which NACL is attached. Table 2.2 shows an example of an NACL configuration in

[1] Amazon Elastic Compute Cloud, also known as EC2, is AWS's version of the VM8

Table 2.2: AWS Network Access Control List (NACL) Configuration Example

Inbound rules					
Rule number	Type	Protocol	Port range	Source	Allow/Deny
100	HTTPS	TCP	443	0.0.0.0/0	Allow
*	All traffic	All	All	0.0.0.0/0	Deny
Outbound rules					
Rule number	Type	Protocol	Port range	Destination	Allow/Deny
100	All traffic	All	All	0.0.0.0/0	Allow
*	All traffic	All	All	0.0.0.0/0	Deny

Table 2.3: AWS VPC Security Group and Network Access Control List Comparison

	Security Groups	Network ACL
Scope	Instance-level	Subnet-level
Filtering mechanism	Stateful	Stateless
Rule type	Supports allow only (default deny)	Supports allow rules and deny rules
Precedence	Evaluate all rules before deciding whether to allow traffic	Rules are evaluated in order, starting with the lowest numbered rule.
Occurance	One or more security group can be applied to an instance	One and only one NACL can be applied to a subnet

AWS which allows inbound traffic to port 443 from any IPv4 source and all outbound traffic to any IPv4 destination. Rule numbers in each NACL is a unique number from 1 to 32766. The rules are evaluated in order, starting from the lowest numbered rule [2]. If the traffic match a rule during the evaluation, the rule is applied to the traffic and the evaluation stops. Note that the rule for which the rule number is represented by an asterisk means that if a packet does not match any of the other numbered rule, then this rule gets applied.

Each VPC in AWS has a default NACL that is configured as follows [2]:

- Allow all inbound traffic to flow into the subnets with which it is associated.
- Allow all outbound traffic to flow out of the subnets with which it is associated.

It is worth noting that a customized NACL, i.e., a NACL that is not a default NACL in a VPC, denies all inbound and outbound traffic by default [2].

NACLs in AWS are **stateless**.

Table 2.3 explains the difference between security groups and NACLs. A *stateful* packet filtering mechanism implies that traffic generated in response to allowed traffic in one direction is

allowed to flow in the opposite direction, regardless of the rules. For instance, a security group with an inbound rule that allows incoming traffic to port 80 also allows the return traffic even if the outbound rules does not explicitly specify the corresponding rule entry. In contrast, a NACL, which adopts a *stateless* filtering mechanism, requires that the return traffic be explicitly specified by the inbound/outbound rules, e.g., if the inbound rule for a given NACL specifies that ICMP traffic is allowed from an IP range, the outbound rule must also specify that the ICMP traffic destined for that IP range is allowed. It is suggested by the AWS technical documentation that security groups should be used as the primary measure for network access to VPCs for its versatility such as being stateful and the ability to be referenced by other security group rules [3]. It is worth noting that, with the presence of security groups, network ACL can be properly configured to provide an additional layer of defense if security group rules are too permissive. The use of a combination of properly configured security groups and NACL offers defense-in-depth protection against malicious network traffic [3].

2.1.2 Security features in Azure VPCs

Network security groups (NSGs) in Azure allow or deny ingress or egress traffic specified by the inbound or outbound rules. Differing from the security group in AWS which applies only at the instance level, network security groups in Azure offer the capability to be applied to both instances and subnets.

Rules in the NSG are defined by the priority which determines the order of evaluation and the five-tuple, i.e., (Source, Source Port, Destination, Destination Port, Protocol), and the rules are *stateful* [8]. Table 2.4 shows an example of the NSG configuration such that inbound traffic to any resources within CIDR block 10.0.0.0/24 over HTTPS is allowed. This NSG is also configured to allow internal communication between resources inside the virtual network. Metadata such as VirtualNetwork and Internet are service tags that essentially represent a group of resources, e.g., the Internet represents all IP addresses while VirtualNetwork represents all resources within the virtual network to which the NSG is attached [8].

Both a VM and a subnet in Azure can be associated with zero or one NSG. A NSG can be associated with as many subnets and VMs as possible [13].

NSGs in Azure are **stateful** 12.

2.1.3 Security features in GCP VPCs

Firewall rules in GCP are *stateful rules* that apply to a given project and VPC network [20]. While GCP firewall rules are defined at the network level, connections are allowed or denied on a per-instance basis within the protected network [20]. Even though GCP does not have AWS

Table 2.4: Azure Network Security Group Configuration Example

Inbound rules					
Priority	Port	Protocol	Source	Destination	Action
100	443	TCP	0.0.0.0/0	10.0.0.0/24	Allow
65000	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65500	Any	Any	Any	Any	Deny
Outbound rules					
Priority	Port	Protocol	Source	Destination	Action
65000	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	Any	Any	Any	Internet	Allow
65500	Any	Any	Any	Any	Deny

security groups equivalent that directly applies to instances and provides instance-level access control over traffic, the same can be achieved by using a so-called *network tags* as the target of the rule in GCP firewalls. A *network tag* is an optional attribute that is attached to an instance and thus the instance can be referred to by it. Hence, firewall rules can also be applied at the instance level by utilizing the network tags. Each firewall rule can either allow or deny ingress or egress traffic. The rules are defined by the following components [20]:

- **Direction**: ingress/egress from the perspective of the target
- **Priority**: a numerical integer number from 0 to 65535 which determines whether the rule is applied. Low numerical number implies high priority and high numerical number implies low priority. The default priority is 1000. Unlike AWS NACL and Azure NSG, GCP Firewalls allow two rules to have the same priority. When this happens, the rule with a deny action takes precedence with the other with an allow action [20].
- **Action**: is either allow or deny. An allow action permits connections that match the other specified components whereas a deny action blocks connections that match the other specified components.
- **Target** defines the instance or network to which the rule applies.
- **Source or destination** filter for packet characteristics
- **Protocol** and destination port

Table 2.5 shows an example of the GCP firewall configuration. This configuration allows the incoming traffic from the public internet to establish a connection over HTTP (Port 80 over TCP) or HTTPS (Port 443 over TCP) It also allow traffic from IPv4 address 1.1.1.1 to establish an SSH connection (Port 22 over TCP) with VMs whose network tag is host-vm.

Table 2.5: GCP Firewall Configuration Example

Inbound rules						
Type	Priority	Source ranges	Action	Protocol and ports	Targets	Network
Ingress	1000	0.0.0.0/0	tcp:80, 443	Allow	Apply to all	default
Ingress	1000	1.1.1.1/32	tcp:22	Allow	host-vm	default
Ingress	65535	0.0.0.0/0	all	Deny	Apply to all	Default
Egress	65534	0.0.0.0/0	all	Allow	Apply to all	Default
Egress	65535	0.0.0.0/0	all	Deny	Apply to all	Default

2.2 Cloud Security Syntax and Semantics Pertaining to Connectivity

In this section, we formalize the syntax and the semantics of the security features pertaining to multi-cloud connectivity that we explored in section 2.1. Specifically, we examine the following: AWS security groups, AWS network access control lists (NACLs), GCP firewall, and Azure network security groups (NSGs) We discuss the rule configuration in relation to ip packets. We acknowledge that an ip packet has attributes that are optional, for example, an ip packet whose ip protocol is icmp contains optional attributes *type* and *code* instaed of *source port* and *destination port*. For the purpose of this analysis, we constrain the ip packets of our interest to be ones that contain the following attributes only: (ip_{src}, ip_{dest}, *protocol*, $port_{src}$, $port_{dest}$). Consequently, when we specify the syntax and the semantics of these cloud security features, we include only attributes pertaining to port numbers, i.e., source port and destination port, and ignore the icmp type and code attributes.

2.2.1 AWS Security Group Abstract Syntax

We formalize the policy language in a simplified abstract syntax based on the set of APIs for AWS security groups [2][3] and NACLs [4]. An AWS security group is a set of IpPermissions, an IpPermission is a 4-tuple as described in Table 2.6, and an IpRange is a set of ip addresses.

In the language we describe in Table 2.6, a *security group* is formally defined as a set of *IpPermissions*. Each IpPermission is a 4-tuple ⟨*Direction, IpProtocol, PortRange, IpRange*⟩. Since

[2]https://docs.aws.amazon.com/AWSEC2/latest/APIReference/API_AuthorizeSecurityGroupIngress.html
[3]https://docs.aws.amazon.com/AWSEC2/latest/APIReference/API_AuthorizeSecurityGroupEgress.html
[4]https://docs.aws.amazon.com/AWSEC2/latest/APIReference/API_CreateNetworkAclEntry.html

SecurityGroup	::=	a `set` of IpPermission*s*
IpPermission	::=	`tuple:`⟨Direction, IpProtocol, PortRange, IpRanges⟩
Direction	::=	*ingress* | *egress*
IpProtocol	::=	`int` $\in [0, 255]$
PortRange	::=	[FromPort, ToPort] $\subseteq [0, 65535]$
FromPort	::=	`int` $\in [0, 65535]$
ToPort	::=	`int` $\in [0, 65535]$
IpRanges	::=	$\{a.b.c.d/n : a, b, c, d \in [0, 255]$ *and* $n \in [0, 32]\}$

Table 2.6: AWS Security Group Abstract Syntax

each security group adopts a default-deny strategy, the IpPermission in the security group overrides the default deny rules to allow traffic based on the attributes specified in the 4-tuple. The *Direction* construct is a constant that states the direction of the traffic on which the ip permission is enforced. The *IpProtocol* construct is an integer that specifies the protocol of the traffic to which the ip permission is applicable. The *PortRange* constructs is a set defined by the set [*FromPort, ToPort*], which is a subset of [0, 65535], that together defines the range of ports that specific ip permission applies. Finally, the *IpRange* construct is a set of ip addresses in CIDR notation 5 that defines the allowed source ip ranges for ingress traffic or the allowed destination ip ranges for egress traffic.

2.2.2 AWS Network Access Control List (NACL) Abstract Syntax

NACL	::=	a `sequence` of Rule*s*
Rule	::=	`tuple:`⟨Egress, RuleAction, RuleNumber, CidrBlock, Protocol, PortRange⟩
Egress	::=	*true* | *false*
RuleAction	::=	*allow* | *deny*
RuleNumber	::=	`int` $\in [1, 32766]$
CidrBlock	::=	$a.b.c.d/n : a, b, c, d \in [0, 255]$ *and* $n \in [0, 32]$
Protocol	::=	`int:` $\in [0, 255]$
PortRange	::=	[FromPort, ToPort] $\subseteq [0, 65535]$
FromPort	::=	`int` $\in [0, 65535]$
ToPort	::=	`int` $\in [0, 65535]$

Table 2.7: AWS NACL Abstract Syntax

Table 2.7 presents the abstract syntax for AWS NACL. A *NetworkAccessControlList* is defined as a sequence of *Rules*. A *Rule* is a 6-tuple ⟨*Egress, RuleAction, RueNumber, CiderBlock, Protocol, PortRange*⟩. The *Egress* construct is a boolean for which true means the rule targets outbound traffic and false indicates that the rule is for inbound traffic. The *RuleAction* construct is a constant that decides whether the traffic to which the rule is applicable should be allowed or denied. The *RuleNumber* construct is an integer that decides the relative order for which the rule will be evaluated. The rule number in an AWS NACL must be unique. Whenever the subnet receives network traffic, be it inbound or outbound, the rules in NACL will be evaluated according to the rule number in ascending order, i.e., the rule with the lowset rule number in the NACL will be evaluated first. The evaluation stops as soon as there is a match and subsequently the matching rule will be applied to the traffic. The *CidrBlock* construct is a set of ip addresses in CIDR notation that defines the allowed or denied traffic based on the source ip ranges if it is an ingress rule, or, destination ip ranges if it is an egress rule. The *Protocol* construct is an integer that specifies the protocol number [5] of the traffic to which the ip permission is applicable, e.g., 6 is for TCP and 17 is for UDP etc. The *PortRange* construct is a set [*FromPort, ToPort*] which together define the range of ports that the rule applies.

Formally, the mapping from an AWS subnet to an AWS NACL is a function such that each subnet must be associated with an NACL. An NACL can be associated with multiple subnets. However, a subnet can be associated with only one network ACL at a time. When a new network ACL with is associated with a subnet, the previous association is removed.

2.2.3 AWS Security Semantics

We define the term/phrase "a set of security groups $\{G_1, \ldots, G_k\}$ allows an ip packet to arrive at a VM 8" as the situation that either (1) there exists an ip permission P_i in $\{G_1, \ldots, G_k\}$ that permits some egress traffic for which the ip packet matches the statefully-added ingress ip permisson, or, (2) there is a matching ingress ip permission P_j in $\{G_1, \ldots, G_k\}$ for the ip packet, but not both.

> Formally, given (i) a set of AWS security groups $\{G_1, \ldots, G_k\}$ for some $k \geq 1$, (ii) a subnet S, (iii) a VM 8, $M = \langle ip_m, S, \{G_1, \ldots, G_k\}\rangle$ (iv) an ip packet pk_{ip} whose destination ip address is $pk_{ip}.ip_{dest}$ such that $pk_{ip}.ip_{dest} = ip_m$, and (v) a set of Ip-Permissions $\{P_1, \ldots, P_n\}$ for some $n \geq 1$ in $\{G_1, \ldots, G_k\}$, we define the term/phrase "There exists an ip permission P_i in a set of security groups $\{G_1, \ldots, G_k\}$ that permits some egress traffic for which the ip packet matches the statefully-added ingress ip permission" as the situation that $((P_i \in \{P_1 \ldots P_n\}) \wedge (P_i.direction = egress) \wedge (pk_{ip}.ip_{src} \in P_i.IpRanges) \wedge (pk_{ip}.protocol = P_i.IpProtocol) \wedge (pk_{ip}.port_{src} \in [P_i.FromPort, P_i.ToPort]))$ is valid.

[5] http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml

Given (i) a set of AWS security groups $\{G_1, \ldots, G_k\}$ for some $k \geq 1$, (ii) a subnet S, (iii) a VM 8, $M = \langle ip_m, S, \{G_1, \ldots, G_k\}\rangle$ (iv) an ip packet pk_{ip} whose destination ip address is $pk_{ip}.ip_{dest}$ such that $pk_{ip}.ip_{dest} = ip_m$, and (v) a set of IpPermissions $\{P_1, \ldots, P_n\}$ for some $n \geq 1$ in $\{G_1, \ldots, G_k\}$, we define the term/phrase "There exists a matching ingress ip permission P_j in a set of security groups $\{G_1, \ldots, G_k\}$ for the ip packet" as the situation that $((P_i \in \{P_1 \ldots P_n\}) \wedge (P_j.direction = ingress) \wedge (pk_{ip}.ip_{src} \in P_j.IpRanges) \wedge (pk_{ip}.protocol = P_j.IpProtocol) \wedge (pk_{ip}.port_{dest} \in [P_j.FromPort, P_j.ToPort]))$ is valid.

We define the term/phrase "a set of security groups $\{G_1, \ldots, G_k\}$ allows an ip packet to leave a VM 8" as the situation that either (1) there exists an ip permission P_i in $\{G_1, \ldots, G_k\}$ that permits some ingress traffic for which the ip packet matches the statefully-added egress ip permission, or, (2) there is a matching egress ip permission P_j in $\{G_1, \ldots, G_k\}$ for the ip packet.

Given (i) a set of AWS security groups $\{G_1, \ldots, G_k\}$ for some $k \geq 1$, (ii) a subnet S, (iii) a VM 8, $M = \langle ip_m, S, \{G_1, \ldots, G_k\}\rangle$, (iv) an ip packet pk_{ip} whose source ip address is $pk_{ip}.ip_{src}$ such that $pk_{ip}.ip_{src} = ip_m$, and (v) a set of ip permissions $\{P_1, \ldots, P_n\}$ for some $n \geq 1$ in $\{G_1, \ldots, G_k\}$, we define the term/phrase "There exists an ip permission P_i in a set of security groups $\{G_1, \ldots, G_k\}$ that permits some ingress traffic for which the ip packet matches the statefully-added egress ip permission" as the situation that $((P_i \in \{P_1 \ldots P_n\}) \wedge (P_i.direction = ingress) \wedge (pk_{ip}.ip_{dest} \in P_i.IpRanges) \wedge (pk_{ip}.protocol = P_i.IpProtocol) \wedge (pk_{ip}.port_{src} \in [P_i.FromPort, P_i.ToPort]))$ is valid.

Given (i) a set of AWS security groups $\{G_1, \ldots, G_k\}$ for some $k \geq 1$, (ii) a subnet S, (iii) a VM, $M = \langle ip_m, S, \{G_1, \ldots, G_k\}\rangle$, (iv) an ip packet pk_{ip} whose source ip address is $pk_{ip}.ip_{src}$ such that $pk_{ip}.ip_{src} = ip_m$, and (v) a set of ip permissions $\{P_1, \ldots, P_n\}$ for some $n \geq 1$ in $\{G_1, \ldots, G_k\}$, we define the term/phrase "There exists a matching egress ip permission P_j in a set of security groups $\{G_1, \ldots, G_k\}$ for the ip packet" as the situation that $((P_j \in \{P_1 \ldots P_n\}) \wedge (P_j.direction = egress) \wedge (pk_{ip}.ip_{dest} \in P_j.IpRanges) \wedge (pk_{ip}.protocol = P_j.IpProtocol) \wedge (pk_{ip}.port_{dest} \in [P_j.FromPort, P_j.ToPort]))$ is valid.

We define the term/phrase "NACL allows an ip packet to enter the subnet to which the NACL is bound" as the situation that there exists an allow-ingress rule R_i in the NACL that matches the ip packet that is of higher priority than every deny-ingress rule that matches the same ip packet. Note that there must exist at least one deny-ingress rule that matches the ip packet, i.e., the default deny-ingress rule.

Given a subnet S to which a NACL N is bound, an ip packet pk_{ip} whose source ip address is $pk_{ip}.ip_{src}$ such that $pk_{ip}.ip_{src} \in S$, the AWS NACL syntax 2.7, and

a sequence of rules $\langle R_1, \ldots, R_n \rangle$ for some $n \geq 1$ in N, we define the term/phrase "an allow-ingress rule R_i matches the ip packet" as the situation that $\neg(R_i.Egress) \wedge (R_i.RuleAction = allow) \wedge (pk_{ip}.ip_{src} \in R_i.CidrBlock) \wedge (pk_{ip}.protocol = R_i.Protocol) \wedge (pk_{ip}.port_{dest} \in R_i.PortRange))$

Given a subnet S to which a NACL N is bound, an ip packet pk_{ip} whose source ip address is $pk_{ip}.ip_{src}$ such that $pk_{ip}.ip_{src} \in S$, the AWS NACL syntax 2.7, and a sequence of rules $\langle R_1, \ldots, R_n \rangle$ for some $n \geq 1$ in N, we define the term/phrase "a deny-ingress rule R_j in the NACL that matches the ip packet" as the situation that $\neg(R_j.Egress) \wedge (R_j.RuleAction = deny) \wedge (pk_{ip}.ip_{src} \in R_i.CidrBlock) \wedge (pk_{ip}.protocol = R_i.Protocol) \wedge (pk_{ip}.port_{dest} \in R_i.PortRange)$

Given a subnet S to which a NACL N is bound, an ip packet pk_{ip} whose source ip address is $pk_{ip}.ip_{src}$ such that $pk_{ip}.ip_{src} \in S$, the AWS NACL syntax 2.7, and a sequence of rules $\langle R_1, \ldots, R_n \rangle$ for some $n \geq 1$ in N, an allow-ingress rule R_i such that $R_i \in \{R_1, \ldots, R_n\}$ that matches the ip packet, and a deny-ingress rule R_j such that $R_j \in \{R_1, \ldots, R_n\}$ that matches the ip packet, we define the term/phrase "an allow-ingress rule R_i that matches the ip packet that is of higher priority than any deny-ingress rule R_j that matches the ip pakcet" as the situation that $(R_i.RuleNumber) < R_j.RuleNumber)$

We define the term/phrase "NACL allows an ip packet to leave the subnet to which the NACL is bound" as the situation that there exists an allow-egress rule R_i in the NACL that matches the ip packet that is of higher priority than every deny-egress rule that matches the ip packet. Note that there must exist at least one deny-egress rule that matches the ip packet, i.e., the default deny-egress rule.

Given a subnet S to which a NACL N is bound, an ip packet pk_{ip} whose source ip address is $pk_{ip}.ip_{src}$ such that $pk_{ip}.ip_{src} \in S$, the AWS NACL syntax 2.7, and a sequence of rules $\langle R_1, \ldots, R_n \rangle$ for some $n \geq 1$ in N, we define the term/phrase "an allow-egress rule R_i matches the ip packet" as the situation that $(R_i.Egress) \wedge (R_i.RuleAction = allow) \wedge (pk_{ip}.ip_{dest} \in R_i.CidrBlock) \wedge (pk_{ip}.protocol = R_i.Protocol) \wedge (pk_{ip}.port_{dest} \in R_i.PortRange)$

Given a subnet S to which a NACL N is bound, an ip packet pk_{ip} whose source ip address is $pk_{ip}.ip_{src}$ such that $pk_{ip}.ip_{src} \in S$, the AWS NACL syntax 2.7, and a sequence of rules $\langle R_1, \ldots, R_n \rangle$ for some $n \geq 1$ in N, we define the term/phrase "a deny-egress rule R_j in the NACL that matches the ip packet" as the situation that $(1 \leq j \leq n) \wedge \neg(i = j) \wedge (R_j.Egress) \wedge (R_j.RuleAction = deny) \wedge (pk_{ip}.ip_{src} \in S) \wedge (pk_{ip}.ip_{dest} \in R_i.CidrBlock) \wedge (pk_{ip}.protocol = R_i.Protocol) \wedge (pk_{ip}.port_{dest} \in R_i.PortRange)$

Given a subnet S to which a NACL N is bound, an ip packet pk_{ip} whose source ip address is $pk_{ip}.ip_{src}$ such that $pk_{ip}.ip_{src} \in S$, the AWS NACL syntax 2.7, and a sequence of rules $\langle R_1, \ldots, R_n \rangle$ for some $n \geq 1$ in N, an allow-egress rule R_i such that $R_i \in \{R_1, \ldots, R_n\}$ that matches the ip packet, and a deny-egress rule R_j such that $R_j \in \{R_1, \ldots, R_n\}$ that matches the ip packet, we define the term/phrase "an allow-egress rule R_i that matches the ip packet that is of higher priority than any deny-egress rule R_j that matches the ip pakcet" as the situation that $(R_i.RuleNumber) < R_j.RuleNumber)$

Given the above definitions, and the facts that: (1) every VM has at least one security group bound to it, and, (2) every subnet has exactly one NACL bound to it, we specify the semantics of a set of security groups, $\{G_1, \ldots, G_k\}$ for some $k \geq 1$ and a NACL N as follows:

Given a subnet S, a set of security groups $\{G_1, \ldots, G_k\}$ for some $k \geq 1$, a NACL N that is bound to S, a VM M such that $M = \langle ip_m, S, \{G_1, \ldots, G_k\} \rangle$. Then, an ip packet is allowed to arrive at the VM if and only if (i) the NACL N allows the IP packet to enter the subnet S to which N is bound, and, (ii) $G_1, \ldots, G_k$ allow the ip packet to arrive at M.

And, an ip packet sent by M is allowed to exit AWS if and only if (i) $G_1, \ldots, G_k$ allow the ip packet to leave M, and, (ii) the NACL N allows the ip packet to leave the S to which the N is bound.

2.2.4 GCP Firewall Abstract Syntax

We formalize the policy language in a simplified abstract syntax based on the set of APIs for GCP firewalls. In this syntax, ? denotes optional elements and * denotes list-valued elements.

Table 2.8 presents the abstract syntax of the GCP Firewall. A *Firewall* is a sequence of *Rules*. A *Rule* is a 5-tuple, (*Direction, Priority, Action, Target, Source or Destination*) *Direction* is either 'ingress' or 'egress' which defines the direction of the flow of the traffic. *Priority* is an integer from 0 to 65535 that decides whether the rule is applied. Rules are evaluated starting from the lowest numbered priority. Where two rules have the same priority, the deny rule takes precedence over the allow rule. The *Action* construct is a set of *Allow* constructs or a set of *Deny* constructs but not both. *Allow* is a 2-tuple $\langle Protocol, Range \rangle$ that defines the ports over a specific protocol to which the traffics are permitted. Similarly, the *Deny* construct is a 2-tuple $\langle Protocol, Range \rangle$ that determines the ports over a specific protocol to which traffic is blocked. The *Target* construct is a set of ip addresses in CIDR notation that defines the target instances to which the rules are applied. The *Source* construct is a set of ip addresses in CIDR notation that defines the source

Firewall	::=	a `sequence` of Rule$s$
Rule	::=	`tuple:` ⟨Direction, Priority, Action, Target, Source | Destination⟩
Direction	::=	*ingress* | *egress*
Priority	::=	`int:` $\in [0, 65535]$
Action	::=	a `set` of Allow$s$ | a `set` of Deny$s$
Allow	::=	`tuple:` ⟨Protocol, PortRanges?⟩
Deny	::=	`tuple:` ⟨Protocol, PortRanges?⟩
Protocol	::=	`int:` $[0, 255]$
PortRanges	::=	`set` $\subseteq [0, 65535]$
Target	::=	$\{a.b.c.d/n : a, b, c, d \in [0, 255]$ *and* $n \in [0, 32]\}$
Source	::=	$\{a.b.c.d/n : a, b, c, d \in [0, 255]$ *and* $n \in [0, 32]\}$
Destination	::=	$\{a.b.c.d/n : a, b, c, d \in [0, 255]$ *and* $n \in [0, 32]\}$

Table 2.8: GCP Firewall Abstract Syntax

of the *ingress* traffic to which the rules are applicable. Similarly, the *Destination* construct is a set of ip addresses in CIDR notation that defines the destination of the *egress* traffic to which the rules are applicable.

2.2.5 GCP Security Semantics

We define the term/phrase "a GCP firewall F allows an ip packet pk_{ip} to arrive at a VM" as the situation that either (1) there exists an allow-egress rule R_i in F for which the ip packet pk_{ip} matches the statefully-added ingress rule, and such allow-egress rule which matches the ip packet pk_x is of higher priority than every deny-egress rule that matches ip packet pk_x, or, (2) there is a matching allow-ingress rule R_i in F for the ip packet pk_{ip} that is of higher priority than every deny-ingress rule that matches the ip packet pk_{ip}, but not both. Note that there must exist at least one deny-egress rule that matches the ip packet pk_x, i.e., the default deny-egress rule.

> Given (i) a subnet S to which a GCP firewall F is bound, (ii) a VM 8, $M = \langle ip_m, S, F\rangle$ (iii) an ip packet pk_{ip} whose destination ip address is $pk_{ip}.ip_{dest}$ such that $pk_{ip}.ip_{dest} = ip_m$, and (iv) a sequence of Rules $\langle R_1, \ldots, R_n\rangle$ for some $n \geq 1$ in F, we define the term/phrase "there exists an allow-egress rule R_i in F for which the ip packet pk_{ip} matches the statefully-added ingress rule" as the situation that $(R_i \in \langle R_1 \ldots R_n\rangle) \wedge (R_i.Direction = egress) \wedge (R_i.Action.Allow.Protocol = pk_{ip}.protocol) \wedge (pk_{ip}.port_{src} \in R_i.Action.Allow.PortRange) \wedge (pk_{ip}.ip_{src} \in R_i.Destination)$ is valid.
>
> Given (i) a subnet S to which a GCP firewall F is bound, (ii) a VM 8, $M = \langle ip_m, S, F\rangle$ (iii) an ip packet pk_{ip} whose destination ip address is $pk_{ip}.ip_{dest}$ such that $pk_{ip}.ip_{dest} =$

ip_m, and (iv) a sequence of Rules $\langle R_1, \ldots, R_n \rangle$ for some $n \geq 1$ in F, we define the term/phrase "an allow-egress rule R_i matches an ip packet pk_x is of higher priority than any deny-egress rule R_j that matches the ip packet pk_x" as the situation that $(R_i.Priority < R_j.Priority)$ is valid.

Given (i) a subnet S to which a GCP firewall F is bound, (ii) a VM 8, $M = \langle ip_m, S, F \rangle$ (iii) an ip packet pk_{ip} whose destination ip address is $pk_{ip}.ip_{dest}$ such that $pk_{ip}.ip_{dest} = ip_m$, and (iv) a sequence of Rules $\langle R_1, \ldots, R_n \rangle$ for some $n \geq 1$ in F, we define the term/phrase "there is a matching allow-ingress rule R_i in F for the ip packet pk_{ip}" as the situation that $(R_i \in \langle R_1 \ldots R_n \rangle) \wedge (R_i.Direction = ingress) \wedge (R_i.Action.Allow.Protocol = pk_{ip}.protocol) \wedge (pk_{ip}.port_{dest} \in R_i.Action.Allow.PortRange) \wedge (pk_{ip}.ip_{src} \in R_i.Source)$ is valid.

We define the term/phrase "a GCP firewall F allows an ip packet pk_{ip} to leave the subnet S to which the firewall is bound" as the situation that either (1) there exists an allow-ingress rule R_i in F for which the ip packet pk_{ip} matches the statefully-added egress rule, and such allow-ingress rule which matches the ip packet pk_x is of higher priority than every deny-ingress rule that matches ip packet pk_x, or, (2) there is a matching allow-egress rule R_i in F for the ip packet pk_{ip} that is of higher priority than every deny-egress rule that matches the ip packet pk_{ip}, but not both. Note that there must exist at least one deny-ingress rule that matches the ip packet pk_x, i.e., the default deny-ingress rule.

Given (i) a subnet S to which a GCP firewall F is bound, (ii) a VM 8, $M = \langle ip_m, S, F \rangle$ (iii) an ip packet pk_{ip} whose source ip address is $pk_{ip}.ip_{src}$ such that $pk_{ip}.ip_{src} = ip_m$, and (iv) a sequence of Rules $\langle R_1, \ldots, R_n \rangle$ for some $n \geq 1$ in F, we define the term/phrase "there exists an allow-ingress rule R_i in F for which the ip packet pk_{ip} matches the statefully-added egress rule" as the situation that $(R_i \in \langle R_1 \ldots R_n \rangle) \wedge (R_i.Direction = ingress) \wedge (R_i.Action.Allow.Protocol = pk_{ip}.protocol) \wedge (pk_{ip}.port_{src} \in R_i.Action.Allow.PortRange) \wedge (pk_{ip}.ip_{dest} \in R_i.Source)$ is valid.

Given (i) a subnet S to which a GCP firewall F is bound, (ii) a VM 8, $M = \langle ip_m, S, F \rangle$ (iii) an ip packet pk_{ip} whose source ip address is $pk_{ip}.ip_{src}$ such that $pk_{ip}.ip_{src} = ip_m$, and (iv) a sequence of Rules $\langle R_1, \ldots, R_n \rangle$ for some $n \geq 1$ in F, we define the term/phrase "an allow-ingress rule R_i matches an ip packet pk_x is of higher priority than any deny-ingress rule R_j that matches the ip packet pk_x" as the situation that $R_i.Priority < R_j.Priority$ is valid.

Given (i) a subnet S to which a GCP firewall F is bound, (ii) a VM 8, $M = \langle ip_m, S, F \rangle$ (iii) an ip packet pk_{ip} whose source ip address is $pk_{ip}.ip_{src}$ such that $pk_{ip}.ip_{src} = ip_m$, and (iv) a sequence of Rules $\langle R_1, \ldots, R_n \rangle$ for some $n \geq 1$ in F, we define the term/phrase "there is a matching allow-egress rule R_i in F for the

ip packet pk_{ip}" as the situation that $(R_i \in \langle R_1 \ldots R_n \rangle) \wedge (R_i.Direction = egress) \wedge (R_i.Action.Allow.Protocol = pk_{ip}.protocol) \wedge (pk_{ip}.port_{dest} \in R_i.Action.Allow.PortRange) \wedge (pk_{ip}.ip_{dest} \in R_i.Destination)$ is valid.

Given the above definitions, we specify the semantics of the GCP firewall as follows:

> Given (i) a subnet S in GCP to which a GCP firewall F is bound, (ii) a VM 8, $M = \langle ip_m, S, F \rangle$ (iii) a sequence of Rules $\langle R_1, \ldots, R_n \rangle$ for some $n \geq 1$ in F, an ip packet pk_{ip} is allowed to arrive at the VM M if and only if F allows an ip packet pk_{ip} to arrive at M.
>
> And, an ip packet pk_{ip} is allowed to exit GCP if and only if F allows pk_{ip} to leave the subnet S to which F is bound.

2.2.6 Azure Network Security Group Abstract Syntax

We formalize the policy language in a simplified abstract syntax based on the set of APIs for Azure Network Security Groups [6]. In this syntax, * denotes list valued elements. An Azure network security group is a sequence of Rules, a Rule is an 8-tuple as described in Table 2.9.

NetworkSecurityGroup	::=	a `sequence` of Rule*s*
Rule	::=	`tuple:` ⟨Direction, Priority, Access, Protocol, SourceIpRange, SourcePortRange, DestinationIpRange, DestinationPortRange⟩
Direction	::=	*inbound* | *outbound*
Priority	::=	`int:` $\in [100, 4096]$
Access	::=	*allow* | *deny*
Protocol	::=	`int:` $[0, 255]$
SourceIpRange	::=	$\{a.b.c.d/n : a, b, c, d \in [0, 255]$ *and* $n \in [0, 32]\}$
SourcePortRange	::=	`set` $\subseteq [0, 65535]$
DestinationIpRange	::=	$\{a.b.c.d/n : a, b, c, d \in [0, 255]$ *and* $n \in [0, 32]\}$
DestinationPortRange	::=	`set` $\subseteq [0, 65535]$

Table 2.9: Azure Network Security Group Abstract Syntax

[6] https://learn.microsoft.com/en-us/rest/api/virtualnetwork/network-security-groups

2.2.7 Azure Security Semantics

We define the term/phrase "an Azure network security group G allows an ip packet pk_{ip} to reach a VM" as the situation that: (1) there exists an allow-egress rule R_i in G that permits some egress traffic for which pk_{ip} matches the statefully-added ingress rule, and such allow-egress rule which matches the ip packet pk_x is of higher priority than any deny-egress rule that matches pk_x or, (2) there exists a matching allow-ingress rule R_i in G for pk_{ip} that is of higher priority than any deny-ingress rule that matches pk_{ip}, but not both.

> Given (i) an Azure network security groups G, (ii) a subnet S, (iii) a VM 8, $M = \langle ip_m, S, G\rangle$ (iv) an ip packet pk_{ip} whose destination ip address is $pk_{ip}.ip_{dest}$ such that $pk_{ip}.ip_{dest} = ip_m$, and (v) a sequence of Rules $\langle R_1, \ldots, R_n\rangle$ for some $n \geq 1$ in G, we define the term/phrase "There exists an allow-egress rule R_i in G that permits some egress traffic for which pk_{ip} matches the statefully-added ingress rule" as the situation that $((R_i \in \{P_1 \ldots P_n\}) \wedge (R_i.Direction = outbound) \wedge (R_i.Access = allow) \wedge (pk_{ip}.protocol = R_i.Protocol) \wedge (pk_{ip}.ip_{src} \in R_i.DestinationIpRange) \wedge (pk_{ip}.port_{src} \in R_i.DestinationPortRange)) \wedge (pk_{ip}.ip_{dest} \in R_i.SourceIpRange) \wedge (pk_{ip}.port_{dest} \in R_i.SourcePortRange))$ is valid.
>
> Given (i) an Azure network security groups G, (ii) a subnet S, (iii) a VM 8, $M = \langle ip_m, S, G\rangle$ (iv) an ip packet pk_{ip} whose destination ip address is $pk_{ip}.ip_{dest}$ such that $pk_{ip}.ip_{dest} = ip_m$, and (v) a sequence of Rules $\langle R_1, \ldots, R_n\rangle$ for some $n \geq 1$ in G, we define the term/phrase "There exists a matching allow-ingress rule R_i in G for pk_{ip}" as the situation that $((R_i \in \{P_1 \ldots P_n\}) \wedge (R_i.Direction = inbound) \wedge (R_i.Access = allow) \wedge (pk_{ip}.protocol = R_i.Protocol) \wedge (pk_{ip}.ip_{src} \in R_i.SourceIpRange) \wedge (pk_{ip}.port_{src} \in R_i.SourcePortRange)) \wedge (pk_{ip}.ip_{dest} \in R_i.DestinationIpRange) \wedge (pk_{ip}.port_{dest} \in R_i.DestinationPortRange))$ is valid.

We define the term/phrase "an Azure network security group G allows an ip packet pk_{ip} to leave a VM" as the situation that: (1) there exists an allow-ingress rule R_i in G that permits some ingress traffic for which the pk_{ip} matches the statefully-added egress rule, and such allow-ingress rule which matches the ip packet pk_x is of higher priority than every deny-ingress rule that matches pk_x or, (2) there exists a matching allow-egress rule R_i in G for pk_{ip} that is of higher priority than every deny-egress rule that matches pk_{ip}, but not both.

> Given (i) an Azure network security groups G, (ii) a subnet S, (iii) a VM 8, $M = \langle ip_m, S, G\rangle$ (iv) an ip packet pk_{ip} whose source ip address is $pk_{ip}.ip_{src}$ such that $pk_{ip}.ip_{src} = ip_m$, and (v) a sequence of Rules $\{R_1, \ldots, R_n\}$ for some $n \geq 1$ in G, we define the term/phrase "There exists an allow-ingress rule R_i in G that permits some ingress traffic for which pk_{ip} matches the statefully-added egress rule" as the situation that $((R_i \in \{P_1 \ldots P_n\}) \wedge (R_i.Direction = inbound) \wedge (R_i.Access =$

$allow) \wedge (pk_{ip}.protocol = R_i.Protocol) \wedge (pk_{ip}.ip_{src} \in R_i.DestinationIpRange) \wedge (pk_{ip}.port_{src} \in R_i.DestinationPortRange)) \wedge (pk_{ip}.ip_{dest} \in R_i.SourceIpRange) \wedge (pk_{ip}.port_{dest} \in R_i.SourcePortRange))$ is valid.

Given (i) an Azure network security groups G, (ii) a subnet S, (iii) a VM 8, $M = \langle ip_m, S, G\rangle$ (iv) an ip packet pk_{ip} whose source ip address is $pk_{ip}.ip_{src}$ such that $pk_{ip}.ip_{src} = ip_m$, and (v) a sequence of Rules $\{R_1, \ldots, R_n\}$ for some $n \geq 1$ in G, we define the term/phrase "There exists a matching allow-egress rule R_i in G for pk_{ip}" as the situation that $((R_i \in \{P_1 \ldots P_n\}) \wedge (R_i.Direction = outbound) \wedge (R_i.Access = allow) \wedge (pk_{ip}.protocol = R_i.Protocol) \wedge (pk_{ip}.ip_{src} \in R_i.SourceIpRange) \wedge (pk_{ip}.port_{src} \in R_i.SourcePortRange)) \wedge (pk_{ip}.ip_{dest} \in R_i.DestinationIpRange) \wedge (pk_{ip}.port_{dest} \in R_i.DestinationPortRange))$ is valid.

We define the term/phrase "an Azure network security group G allows an ip packet pk_{ip} to enter the subnet to which G is bound" as the situation that: (1) there exists an allow-egress rule R_i in G that permits some egress traffic for which the pk_{ip} matches the statefully-added ingress rule, and such allow-egress rule which matches pk_x is of higher priority than every deny-egress rule that matches pk_x or, (2) there exists a matching allow-ingress rule R_i in G for pk_{ip} that is of higher priority than every deny-ingress rule that matches pk_{ip}, but not both.

Given (i) a subnet S to which an Azure network security group G is bound, (ii) a VM 8, $M = \langle ip_m, S, G\rangle$ (iii) an ip packet pk_{ip} whose destination ip address is $pk_{ip}.ip_{dest}$ such that $pk_{ip}.ip_{dest} = ip_m$, and (iv) a sequence of Rules $\langle R_1, \ldots, R_n\rangle$ for some $n \geq 1$ in F, we define the term/phrase "there exists an allow-ingress rule R_i in F for which the ip packet pk_{ip} matches the statefully-added egress rule" as the situation that $(R_i \in \langle R_1 \ldots R_n\rangle) \wedge (R_i.Direction = ingress) \wedge (R_i.Action.Allow.Protocol = pk_{ip}.protocol) \wedge (pk_{ip}.port_{src} \in R_i.Action.Allow.PortRange) \wedge (pk_{ip}.ip_{dest} \in R_i.Source)$ is valid.

Given (i) a subnet S to which an Azure network security group G is bound, (ii) a VM 8, $M = \langle ip_m, S, G\rangle$ (iii) an ip packet pk_{ip} whose destination ip address is $pk_{ip}.ip_{dest}$ such that $pk_{ip}.ip_{dest} = ip_m$, and (iv) a sequence of Rules $\langle R_1, \ldots, R_n\rangle$ for some $n \geq 1$ in G, we define the term/phrase "There exists a matching allow-ingress rule R_i in G for pk_{ip}" as the situation that $((R_i \in \{P_1 \ldots P_n\}) \wedge (R_i.Direction = inbound) \wedge (R_i.Access = allow) \wedge (pk_{ip}.protocol = R_i.Protocol) \wedge (pk_{ip}.ip_{src} \in R_i.SourceIpRange) \wedge (pk_{ip}.port_{src} \in R_i.SourcePortRange)) \wedge (pk_{ip}.ip_{dest} \in R_i.DestinationIpRange) \wedge (pk_{ip}.port_{dest} \in R_i.DestinationPortRange))$ is valid.

We define the term/phrase "an Azure network security group G allows an ip packet pk_{ip} to leave the subnet to which G is bound" as the situation that: (1) there exists an allow-ingress rule R_i in G that permits some ingress traffic for which pk_{ip} matches the statefully-added egress rule, and

such allow-ingress rule which matches pk_x is of higher priority than every deny-ingress rule that matches pk_x or, (2) there exists a matching allow-egress rule R_i in G for pk_{ip} that is of higher priority than every deny-egress rule that matches pk_{ip}, but not both.

> Given (i) a subnet S to which an Azure network security group G is bound, (ii) a VM 8, $M = \langle ip_m, S, G \rangle$ (iii) an ip packet pk_{ip} whose source ip address is $pk_{ip}.ip_{src}$ such that $pk_{ip}.ip_{src} = ip_m$, and (iv) a sequence of Rules $\{R_1, \ldots, R_n\}$ for some $n \geq 1$ in G, we define the term/phrase "There exists an allow-ingress rule R_i in G that permits some ingress traffic for which pk_{ip} matches the statefully-added egress rule" as the situation that $((R_i \in \{P_1 \ldots P_n\}) \wedge (R_i.Direction = inbound) \wedge (R_i.Access = allow) \wedge (pk_{ip}.protocol = R_i.Protocol) \wedge (pk_{ip}.ip_{src} \in R_i.DestinationIpRange) \wedge (pk_{ip}.port_{src} \in R_i.DestinationPortRange)) \wedge (pk_{ip}.ip_{dest} \in R_i.SourceIpRange) \wedge (pk_{ip}.port_{dest} \in R_i.SourcePortRange))$ is valid.
>
> Given (i) a subnet S to which an Azure network security group G is bound, (ii) a VM 8, $M = \langle ip_m, S, G \rangle$ (iii) an ip packet pk_{ip} whose source ip address is $pk_{ip}.ip_{src}$ such that $pk_{ip}.ip_{src} = ip_m$, and (iv) a sequence of Rules $\{R_1, \ldots, R_n\}$ for some $n \geq 1$ in G, we define the term/phrase "There exists a matching allow-egress rule R_i in G for pk_{ip}" as the situation that $((R_i \in \{P_1 \ldots P_n\}) \wedge (R_i.Direction = outbound) \wedge (R_i.Access = allow) \wedge (pk_{ip}.protocol = R_i.Protocol) \wedge (pk_{ip}.ip_{src} \in R_i.SourceIpRange) \wedge (pk_{ip}.port_{src} \in R_i.SourcePortRange)) \wedge (pk_{ip}.ip_{dest} \in R_i.DestinationIpRange) \wedge (pk_{ip}.port_{dest} \in R_i.DestinationPortRange))$ is valid.

Given the above definition, we specify the semantics of the Azure network security group as follows:

> Given two network security groups G_i and G_j that bounds to a subnet S and a VM 8 M respectively such that $M = \langle ip_{vm}, S, G_j \rangle$, an ip packet pk_{ip} is allowed to arrive at M if and only if (i) G_i allows pk_{ip} to enter S, and, (ii) G_j allows pk_{ip} to reach M.
>
> And, an ip packet pk_{ip} sent by M is allowed to exit Azure if and only if (i) G_j allows pk_{ip} to leave M, and, (ii) G_i allows pk_{ip} to leave S.

2.3 Multi-cloud Connectivity Security Attributes Equivalence and Tightness

In this section, we discuss the equivalence problem of AWS, GCP, and Azure network security based on the semantics presented in section 2.2. Specifically, we try to answer the following question – given one security configuration in one of the clouds, does there exists an equivalent security configuration in other clouds that achieve the same security objectives? If not, what is

the tightest security configuration can other clouds offer? For example, given an AWS subnet, an AWS security group, and an NACL configuration that allows an ip packet pk_{ip} to arrive at a VM in AWS, does there exists a configuration in GCP that also allows pk_{ip} to arrive at a VM in GCP? If this is not possible, what is the best security configuration in GCP one can implement to cover as many security objectives, that are achieved by the AWS security configuration, as possible? An important aspect that will be useful in the discussion of this section is that: In AWS, security groups are stateful while NACLs are stateless. In GCP and in Azure, both GCP firewall rules and Azure NSG are stateful.

We first acknowledge that AWS supports the concept of ICMP type and code in both security groups and NACLs which Azure and GCP do not support such granularity. In this regard, we claim that there exists a least common denominator problem between the network security features offered by the three cloud computing vendors, i.e., there exists a security feature that AWS supports but Azure and GCP do not support. We also claim that AWS is more 'expressive' in its network security features than GCP and Azure due to its additional support for ICMP type and ICMP code. However, for the purpose of our analysis in this section, we refrain ourselves from considering this aspect and consider only ip packets with protocols that support ports.

Given a subnet S_c that belongs to one of the clouds, another subnet S_o such that $S_c \cap S_o = \emptyset$, two VMs M_{m1} and M_{m2}, we are interested in the following scenarios: (i) two VMs resides in the same subnet S_c and, (ii) M_1 resides in S_c and M_2 resides in S_o.

2.3.1 Security Within a Subnet

In this subsection, we discuss the equivalence relation of the security features between any of the two clouds.

2.3.1.1 AWS vs. GCP

Theorem 1 *For every AWS security configuration, there exists an equivalent GCP security configuration.*

We prove this theorem by case analysis. Given an ip packet pk_{ip}, a subnet S_{aws} to which a NACL N is bound, a set of ip permissions $\mathcal{P}_1$ in a set of security groups $\mathcal{G}_1$, another set of ip permissions $\mathcal{P}_2$ in a set of security groups $\mathcal{G}_2$, a VM $M_{m1} = \langle ip_{m1}, S_{aws}, \mathcal{G}_1 \rangle$, and a VM $M_2 = \langle ip_{m2}, S_{aws}, \mathcal{G}_2 \rangle$, there are six cases arising from the AWS security semantics: (1) $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is allowed to leave M_{m1} and allowed to arrive at M_{in}, (2) $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is allowed to leave M_{m1} and denied to arrive at M_{m2}, (3) $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is denied to leave M_{m1}, (4) $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ and is allowed to leave M_{m2} and allowed to

arrive at M_{m1}, (5) $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ and is allowed to leave M_{m2} and denied to arrive at M_{m1}, and, (6) $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ and is denied to leave M_{m2}.

Given a GCP subnet S_{gcp} to which a firewall F is bound, a sequence of rules $\langle R_1, \ldots, R_n \rangle$ in F for some $n \geq 1$, a VM $M_{m1} = \langle ip_{m1}, S_{gcp}, F \rangle$, and a VM $M_{m2} = \langle ip_{m2}, S_{gcp}, F \rangle$, by constructing equivalent GCP security configurations, we perform the case analysis as follows:

Case 1 – Given $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is allowed to leave M_{m1} and allowed to arrive at M_{in}, we can construct the following security configuration in GCP to achieve the same objective: (i) a matching allow-egress rule R_i in F for ip_{pk} that is of higher priority than every deny-egress rule in F that matches pk_{ip}, and, (ii) a matching allow-ingress rule R_j in F for ip_{pk} that is of higher priority than every deny-ingress rule in F that matches pk_{ip}.

Case 2 – Given $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is allowed to leave M_{m1} and denied to arrive at M_{m2}, we can construct the following security configuration in GCP to achieve the same objective: (i) a matching allow-egress rule R_i in F for ip_{pk} that is of higher priority than every deny-egress rule in F that matches pk_{ip}, and, (ii) a matching deny-ingress rule R_j in F for ip_{pk} that is of higher priority than every allow-ingress rule in F that matches pk_{ip}.

Case 3 – Given $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is denied to leave M_{m1}, we can construct the following security configuration in GCP to achieve the same objective: (i) a matching deny-egress rule R_i in F for ip_{pk} that is of higher priority than every allow-egress rule in F that matches pk_{ip}

Case 4 – Given $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ and is allowed to leave M_{m2} and allowed to arrive at M_{m1}, we can construct the following security configuration in GCP to achieve the same objective: (i) a matching allow-egress rule R_i in F for ip_{pk} that is of higher priority than every deny-egress rule in F that matches pk_{ip}, and, (ii) a matching allow-ingress rule R_j in F for ip_{pk} that is of higher priority than every deny-ingress rule in F that matches pk_{ip}.

Case 5 – Given $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ and is allowed to leave M_{m2} and denied to arrive at M_{m1}, we can construct the following security configuration in GCP to achieve the same objective: (i) a matching allow-egress rule R_i in F for ip_{pk} that is of higher priority than every deny-egress rule in F that matches pk_{ip}, and, (ii) a matching deny-ingress rule R_j in F for ip_{pk} that is of higher priority than every allow-ingress rule in F that matches pk_{ip}.

Case 6 – Given $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ and is denied to leave M_{m2}, we can construct the following security configuration in GCP to achieve the same objective: (i) a matching deny-egress rule R_i in F for ip_{pk} that is of higher priority than every allow-egress rule in F that matches pk_{ip}

Thus, we have proved that for every AWS security configuration, there exists an equivalent GCP security configuration.

Theorem 2 *For every GCP security configuration, there exists an equivalent AWS security configuration.*

We prove this theorem by case analysis. Given an ip pakcet pk_{ip}, a GCP subnet S_{gcp} to which a firewall F is bound, a sequence of rules $\langle R_1, \ldots, R_n \rangle$ in F for some $n \geq 1$, a VM $M_{m1} = \langle ip_{m1}, S_{gcp}, F \rangle$, and a VM $M_{m2} = \langle ip_{m2}, S_{gcp}, F \rangle$, there are 6 cases arising from the GCP security semantics: (1) $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is allowed to leave M_{m1} and allowed to arrive at M_{in}, (2) $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is allowed to leave M_{m1} and denied to arrive at M_{m2}, (3) $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is denied to leave M_{m1}, (4) $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ which is allowed to leave M_{m2} and allowed to arrive at M_{m1}, (5) $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ which is allowed to leave M_{m2} and denied to arrive at M_{m1}, and, (6) $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ which is denied to leave M_{m2}.

Given a subnet S_{aws} to which a NACL N is bound, a set of ip permissions $\mathcal{P}_1$ in a set of security groups $\mathcal{G}_1$, another set of ip permissions $\mathcal{P}_2$ in a set of security groups $\mathcal{G}_2$, a VM $M_{m1} = \langle ip_{m1}, S_{aws}, \mathcal{G}_1 \rangle$, and a VM $M_2 = \langle ip_{m2}, S_{aws}, \mathcal{G}_2 \rangle$, by constructing equivalent GCP security configurations, we perform the case analysis as follows:

Case 1 – Given $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is allowed to leave M_{m1} and allowed to arrive at M_{in}, we can construct the following security configuration in AWS to achieve the same objective: (i) a matching egress ip permission $P_i \in \mathcal{P}_1$ for pk_{ip}, (ii) a matching ingress ip permission $P_j \in \mathcal{P}_2$ for pk_{ip}.

Case 2 – Given $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is allowed to leave M_{m1} and denied to arrive at M_{m2}, we can construct the following security configuration in AWS to achieve the same objective: (i) a matching egress ip permission $P_i \in \mathcal{P}_1$ for pk_{ip}, (ii) no matching ingress ip permission exists for pk_{ip} for all ip permission $P_j \in \mathcal{P}_2$.

Case 3 – Given $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is denied to leave M_{m1}, we can construct the following security configuration in AWS to achieve the same objective: (i) no matching egress ip permission exists for pk_{ip} for all ip permission $P_i \in \mathcal{P}_1$.

Case 4 – Given $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ and is allowed to leave M_{m2} and allowed to arrive at M_{m1}, we can construct the following security configuration in AWS to achieve the same objective: (i) a matching ingress ip permission $P_i \in \mathcal{P}_1$ for pk_{ip}, (ii) a matching egress ip permission $P_j \in \mathcal{P}_2$ for pk_{ip}.

Case 5 – Given $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ and is allowed to leave M_{m2} and denied to arrive at M_{m1}, we can construct the following security configuration in AWS to achieve

the same objective: (i) a matching egress ip permission $P_j \in \mathcal{P}_2$ for pk_{ip}, (ii) no matching ingress ip permission exists for pk_{ip} for all ip permission $P_i \in \mathcal{P}_1$.

Case 6 – Given $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ and is denied to leave M_{m2}, we can construct the following security configuration in AWS to achieve the same objective: (i) no matching egress ip permission exists for pk_{ip} for all ip permission $P_j \in \mathcal{P}_2$.

Thus, we have proved that for every GCP security configuration, there exists an equivalent AWS security configuration.

2.3.1.2 AWS vs. Azure

Theorem 3 *For every AWS security configuration, there exists an equivalent Azure security configuration.*

We prove this theorem by case analysis. Given an ip packet pk_{ip}, a subnet S_{aws} to which a NACL N is bound, a set of ip permissions $\mathcal{P}_1$ in a set of security groups $\mathcal{G}_1$, another set of ip permissions $\mathcal{P}_2$ in a set of security groups $\mathcal{G}_2$, a VM $M_{m1} = \langle ip_{m1}, S_{aws}, \mathcal{G}_1 \rangle$, and a VM $M_2 = \langle ip_{m2}, S_{aws}, \mathcal{G}_2 \rangle$, there are six cases arising from the AWS security semantics: (1) $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is allowed to leave M_{m1} and allowed to arrive at M_{in}, (2) $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is allowed to leave M_{m1} and denied to arrive at M_{m2}, (3) $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is denied to leave M_{m1}, (4) $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ and is allowed to leave M_{m2} and allowed to arrive at M_{m1}, (5) $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ and is allowed to leave M_{m2} and denied to arrive at M_{m1}, and, (6) $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ and is denied to leave M_{m2}.

Given an Azure subnet S_{azure}, a sequence of rules $\langle R_{i^1}, \ldots, R_{i^n} \rangle$ in an Azure NSG G_1 for some $n \geq 1$, another sequence of rules $\langle R_{j^1}, \ldots, R_{j^n} \rangle$ in another Azure NSG G_2 for some $n \geq 1$, a VM $M_{m1} = \langle ip_{m1}, S_{azure}, G_1 \rangle$, and a VM $M_{m2} = \langle ip_{m2}, S_{azure}, G_2 \rangle$, by constructing equivalent Azure security configurations, we perform the case analysis as follows:

Case 1 – Given $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is allowed to leave M_{m1} and allowed to arrive at M_{in}, we can construct the following security configuration in GCP to achieve the same objective: (i) a matching allow-egress rule R_i in G_1 for ip_{pk} that is of higher priority than every deny-egress rule in G_1 that matches pk_{ip}, and, (ii) a matching allow-ingress rule R_j in G_2 for ip_{pk} that is of higher priority than every deny-ingress rule in G_2 that matches pk_{ip}.

Case 2 – Given $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is allowed to leave M_{m1} and denied to arrive at M_{m2}, we can construct the following security configuration in GCP to achieve the same objective: (i) a matching allow-egress rule R_i in G_1 for ip_{pk} that is of higher priority

than every deny-egress rule in G_1 that matches pk_{ip}, and, (ii) a matching deny-ingress rule R_j in G_2 for ip_{pk} that is of higher priority than every allow-ingress rule in G_2 that matches pk_{ip}.

Case 3 – Given $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is denied to leave M_{m1}, we can construct the following security configuration in GCP to achieve the same objective: (i) a matching deny-egress rule R_i in G_1 for ip_{pk} that is of higher priority than every allow-egress rule in G_1 that matches pk_{ip}

Case 4 – Given $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ and is allowed to leave M_{m2} and allowed to arrive at M_{m1}, we can construct the following security configuration in GCP to achieve the same objective: (i) a matching allow-egress rule R_i in G_2 for ip_{pk} that is of higher priority than every deny-egress rule in G_2 that matches pk_{ip}, and, (ii) a matching allow-ingress rule R_j in G_1 for ip_{pk} that is of higher priority than every deny-ingress rule in G_1 that matches pk_{ip}.

Case 5 – Given $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ and is allowed to leave M_{m2} and denied to arrive at M_{m1}, we can construct the following security configuration in GCP to achieve the same objective: (i) a matching allow-egress rule R_i in G_2 for ip_{pk} that is of higher priority than every deny-egress rule in G_2 that matches pk_{ip}, and, (ii) a matching deny-ingress rule R_j in G_1 for ip_{pk} that is of higher priority than every allow-ingress rule in G_1 that matches pk_{ip}.

Case 6 – Given $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ and is denied to leave M_{m2}, we can construct the following security configuration in GCP to achieve the same objective: (i) a matching deny-egress rule R_i in G_2 for ip_{pk} that is of higher priority than every allow-egress rule in G_2 that matches pk_{ip}

Thus, we have proved that for every AWS security configuration, there exists an equivalent Azure security configuration.

Theorem 4 *For every Azure security configuration, there exists an equivalent AWS security configuration.*

We prove this theorem by case analysis. Given an ip packet pk_{ip}, an Azure subnet S_{azure}, a sequence of rules $\langle R_{i^1}, \ldots, R_{i^n} \rangle$ in an Azure NSG G_1 for some $n \geq 1$, another sequence of rules $\langle R_{j^1}, \ldots, R_{j^n} \rangle$ in another Azure NSG G_2 for some $n \geq 1$, a VM $M_{m1} = \langle ip_{m1}, S_{azure}, G_1 \rangle$, and a VM $M_{m2} = \langle ip_{m2}, S_{azure}, G_2 \rangle$, there are six cases arising from the Azure security semantics: (1) $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is allowed to leave M_{m1} and allowed to arrive at M_{in}, (2) $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is allowed to leave M_{m1} and denied to arrive at M_{m2}, (3) $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is denied to leave M_{m1}, (4) $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ and is allowed to leave M_{m2} and allowed to arrive at M_{m1}, (5) $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ and is allowed to leave M_{m2} and denied to arrive at M_{m1}, and, (6) $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ and is denied to leave M_{m2}.

Given a subnet S_{aws} to which a NACL N is bound, a set of ip permissions $\mathcal{P}_1$ in a set of security groups $\mathcal{G}_1$, another set of ip permissions $\mathcal{P}_2$ in a set of security groups $\mathcal{G}_2$, a VM $M_{m1} = \langle ip_{m1}, S_{aws}, \mathcal{G}_1 \rangle$, and a VM $M_2 = \langle ip_{m2}, S_{aws}, \mathcal{G}_2 \rangle$, by constructing equivalent Azure security configurations, we perform the case analysis as follows:

Case 1 – Given $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is allowed to leave M_{m1} and allowed to arrive at M_{in}, we can construct the following security configuration in AWS to achieve the same objective: (i) a matching egress ip permission $P_i \in \mathcal{P}_1$ for pk_{ip}, (ii) a matching ingress ip permission $P_j \in \mathcal{P}_2$ for pk_{ip}.

Case 2 – Given $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is allowed to leave M_{m1} and denied to arrive at M_{m2}, we can construct the following security configuration in AWS to achieve the same objective: (i) a matching egress ip permission $P_i \in \mathcal{P}_1$ for pk_{ip}, (ii) no matching ingress ip permission exists for pk_{ip} for all ip permission $P_j \in \mathcal{P}_2$.

Case 3 – Given $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is denied to leave M_{m1}, we can construct the following security configuration in AWS to achieve the same objective: (i) no matching egress ip permission exists for pk_{ip} for all ip permission $P_i \in \mathcal{P}_1$.

Case 4 – Given $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ and is allowed to leave M_{m2} and allowed to arrive at M_{m1}, we can construct the following security configuration in AWS to achieve the same objective: (i) a matching ingress ip permission $P_i \in \mathcal{P}_1$ for pk_{ip}, (ii) a matching egress ip permission $P_j \in \mathcal{P}_2$ for pk_{ip}.

Case 5 – Given $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ and is allowed to leave M_{m2} and denied to arrive at M_{m1}, we can construct the following security configuration in AWS to achieve the same objective: (i) a matching egress ip permission $P_j \in \mathcal{P}_2$ for pk_{ip}, (ii) no matching ingress ip permission exists for pk_{ip} for all ip permission $P_i \in \mathcal{P}_1$.

Case 6 – Given $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ and is denied to leave M_{m2}, we can construct the following security configuration in AWS to achieve the same objective: (i) no matching egress ip permission exists for pk_{ip} for all ip permission $P_j \in \mathcal{P}_2$.

Thus, we have proved that for every Azure security configuration, there exists an equivalent AWS security configuration.

2.3.1.3 Azure vs. GCP

Theorem 5 *For every Azure security configuration, there exists an equivalent GCP security configuration.*

We prove this theorem by case analysis. Given an ip packet pk_{ip}, an Azure subnet S_{azure}, a sequence of rules $\langle R_{i^1}, \ldots, R_{i^n} \rangle$ in an Azure NSG G_1 for some $n \geq 1$, another sequence of rules

$\langle R_{j^1}, \ldots, R_{j^n} \rangle$ in another Azure NSG G_2 for some $n \geq 1$, a VM $M_{m1} = \langle ip_{m1}, S_{azure}, G_1 \rangle$, and a VM $M_{m2} = \langle ip_{m2}, S_{azure}, G_2 \rangle$, there are six cases arising from the Azure security semantics: (1) $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is allowed to leave M_{m1} and allowed to arrive at M_{in}, (2) $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is allowed to leave M_{m1} and denied to arrive at M_{m2}, (3) $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is denied to leave M_{m1}, (4) $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ and is allowed to leave M_{m2} and allowed to arrive at M_{m1}, (5) $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ and is allowed to leave M_{m2} and denied to arrive at M_{m1}, and, (6) $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ and is denied to leave M_{m2}.

Given a GCP subnet S_{gcp} to which a firewall F is bound, a sequence of rules $\langle R_1, \ldots, R_n \rangle$ in F for some $n \geq 1$, a VM $M_{m1} = \langle ip_{m1}, S_{gcp}, F \rangle$, and a VM $M_{m2} = \langle ip_{m2}, S_{gcp}, F \rangle$, by constructing equivalent GCP security configurations, we perform the case analysis as follows:

Case 1 – Given $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is allowed to leave M_{m1} and allowed to arrive at M_{in}, we can construct the following security configuration in GCP to achieve the same objective: (i) a matching allow-egress rule R_i in F for ip_{pk} that is of higher priority than every deny-egress rule in F that matches pk_{ip}, and, (ii) a matching allow-ingress rule R_j in F for ip_{pk} that is of higher priority than every deny-ingress rule in F that matches pk_{ip}.

Case 2 – Given $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is allowed to leave M_{m1} and denied to arrive at M_{m2}, we can construct the following security configuration in GCP to achieve the same objective: (i) a matching allow-egress rule R_i in F for ip_{pk} that is of higher priority than every deny-egress rule in F that matches pk_{ip}, and, (ii) a matching deny-ingress rule R_j in F for ip_{pk} that is of higher priority than every allow-ingress rule in F that matches pk_{ip}.

Case 3 – Given $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is denied to leave M_{m1}, we can construct the following security configuration in GCP to achieve the same objective: (i) a matching deny-egress rule R_i in F for ip_{pk} that is of higher priority than every allow-egress rule in F that matches pk_{ip}

Case 4 – Given $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ and is allowed to leave M_{m2} and allowed to arrive at M_{m1}, we can construct the following security configuration in GCP to achieve the same objective: (i) a matching allow-egress rule R_i in F for ip_{pk} that is of higher priority than every deny-egress rule in F that matches pk_{ip}, and, (ii) a matching allow-ingress rule R_j in F for ip_{pk} that is of higher priority than every deny-ingress rule in F that matches pk_{ip}.

Case 5 – Given $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ and is allowed to leave M_{m2} and denied to arrive at M_{m1}, we can construct the following security configuration in GCP to achieve the same objective: (i) a matching allow-egress rule R_i in F for ip_{pk} that is of higher priority than every deny-egress rule in F that matches pk_{ip}, and, (ii) a matching deny-ingress rule R_j in F for ip_{pk} that is of higher priority than every allow-ingress rule in F that matches pk_{ip}.

Case 6 – Given $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ and is denied to leave M_{m2}, we can construct the following security configuration in GCP to achieve the same objective: (i) a matching deny-egress rule R_i in F for ip_{pk} that is of higher priority than every allow-egress rule in F that matches pk_{ip}

Thus, we have proved that for every AWS security configuration, there exists an equivalent GCP security configuration.

Theorem 6 *For every GCP security configuration, there exists an equivalent Azure security configuration.*

We prove this theorem by case analysis. Given an ip pakcet pk_{ip}, a GCP subnet S_{gcp} to which a firewall F is bound, a sequence of rules $\langle R_1, \ldots, R_n \rangle$ in F for some $n \geq 1$, a VM $M_{m1} = \langle ip_{m1}, S_{gcp}, F \rangle$, and a VM $M_{m2} = \langle ip_{m2}, S_{gcp}, F \rangle$, there are 6 cases arising from the GCP security semantics: (1) $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is allowed to leave M_{m1} and allowed to arrive at M_{in}, (2) $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is allowed to leave M_{m1} and denied to arrive at M_{m2}, (3) $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is denied to leave M_{m1}, (4) $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ which is allowed to leave M_{m2} and allowed to arrive at M_{m1}, (5) $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ which is allowed to leave M_{m2} and denied to arrive at M_{m1}, and, (6) $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ which is denied to leave M_{m2}.

Given an Azure subnet S_{azure}, a sequence of rules $\langle R_{i^1}, \ldots, R_{i^n} \rangle$ in an Azure NSG G_1 for some $n \geq 1$, another sequence of rules $\langle R_{j^1}, \ldots, R_{j^n} \rangle$ in another Azure NSG G_2 for some $n \geq 1$, a VM $M_{m1} = \langle ip_{m1}, S_{azure}, G_1 \rangle$, and a VM $M_{m2} = \langle ip_{m2}, S_{azure}, G_2 \rangle$, by constructing equivalent Azure security configurations, we perform the case analysis as follows:

Case 1 – Given $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is allowed to leave M_{m1} and allowed to arrive at M_{in}, we can construct the following security configuration in GCP to achieve the same objective: (i) a matching allow-egress rule R_i in G_1 for ip_{pk} that is of higher priority than every deny-egress rule in G_1 that matches pk_{ip}, and, (ii) a matching allow-ingress rule R_j in G_2 for ip_{pk} that is of higher priority than every deny-ingress rule in G_2 that matches pk_{ip}.

Case 2 – Given $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is allowed to leave M_{m1} and denied to arrive at M_{m2}, we can construct the following security configuration in GCP to achieve the same objective: (i) a matching allow-egress rule R_i in G_1 for ip_{pk} that is of higher priority than every deny-egress rule in G_1 that matches pk_{ip}, and, (ii) a matching deny-ingress rule R_j in G_2 for ip_{pk} that is of higher priority than every allow-ingress rule in G_2 that matches pk_{ip}.

Case 3 – Given $pk_{ip} = \langle ip_{m1}, ip_{m2}, protocol_{ip}, port_{m1}, port_{m2} \rangle$ which is denied to leave M_{m1}, we can construct the following security configuration in GCP to achieve the same objective: (i) a matching deny-egress rule R_i in G_1 for ip_{pk} that is of higher priority than every allow-egress rule in G_1 that matches pk_{ip}

Case 4 – Given $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ and is allowed to leave M_{m2} and allowed to arrive at M_{m1}, we can construct the following security configuration in GCP to achieve the same objective: (i) a matching allow-egress rule R_i in G_2 for ip_{pk} that is of higher priority than every deny-egress rule in G_2 that matches pk_{ip}, and, (ii) a matching allow-ingress rule R_j in G_1 for ip_{pk} that is of higher priority than every deny-ingress rule in G_1 that matches pk_{ip}.

Case 5 – Given $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ and is allowed to leave M_{m2} and denied to arrive at M_{m1}, we can construct the following security configuration in GCP to achieve the same objective: (i) a matching allow-egress rule R_i in G_2 for ip_{pk} that is of higher priority than every deny-egress rule in G_2 that matches pk_{ip}, and, (ii) a matching deny-ingress rule R_j in G_1 for ip_{pk} that is of higher priority than every allow-ingress rule in G_1 that matches pk_{ip}.

Case 6 – Given $pk_{ip} = \langle ip_{m2}, ip_{m1}, protocol_{ip}, port_{m2}, port_{m1} \rangle$ and is denied to leave M_{m2}, we can construct the following security configuration in GCP to achieve the same objective: (i) a matching deny-egress rule R_i in G_2 for ip_{pk} that is of higher priority than every allow-egress rule in G_2 that matches pk_{ip}

Thus, we have proved that for every GCP security configuration, there exists an equivalent Azure security configuration.

2.3.2 Secturity Outside a Subnet

2.3.2.1 AWS vs. GCP

Theorem 7 *The networking security feature in GCP is not as expressive as the one in AWS. That is, there exists an AWS security configuration, for which no equivalent GCP security configuration exists.*

We prove this theorem by contradiction. We construct an AWS security configuration such that no equivalent GCP security configuration exists as follows:

Given an AWS subnet S_{aws}, a set of ip permissions $\{P_1, \ldots, P_n\}$ for some $n \geq 1$ in security group $\{G_1, \ldots, G_k\}$ for some $k \geq 1$, a VM, $M_{aws} = \langle ip_m, S_{aws}, \{G_1, \ldots, G_k\} \rangle$, a NACL N, with a sequence of rules $\langle R_1, \ldots, R_m \rangle$ for some $m \geq 1$, to which S is bound, an ip packet $pk_{in} = \langle ip_x, ip_y, protocol, port_x, port_y \rangle$ such that $ip_y = ip_m$, and an ip packet generated in response to pk_{in}, i.e., $pk_{res} = \langle ip_y, ip_x, protocol, port_y, port_x \rangle$, suppose the following security configuration exists:

There exists an ip permission P_i in $\{P_1, \ldots, P_n\}$ such that pk_{in} is allowed by P_i, i.e.,

$$\exists P_i \cdot (P_i \in \{G_1, \ldots, G_n\}) \wedge (P.direction = ingress) \wedge (pk_{in}.ip_x \in P_i.IpRanges) \wedge (pk_{in}.protocol = P_i.IpProtocol) \wedge (pk_{in}.port_y \in [P_i.FromPort, P_i.ToPort])$$

and,

there exists an allow-ingress rule R_j in $\langle R_1, \ldots, R_m \rangle$ such that pk_{in} is allowed by R_j to enter S_{aws}, i.e.,

$$\exists R_j \cdot (R_j \in \langle R_1, \ldots, R_m \rangle) \wedge (\neg R_j.Egress) \wedge (R_j.RuleAction = allow) \wedge (pk_{in}.ip_x \in R_j.CidrBlock) \wedge (pk_{in}.protocol = R_j.Protocol) \wedge (pk_{in}.port_y \in R_j.PortRange)$$

and that R_j has higher priority than any deny-ingress rule that matches pk_{in}, and

there exists a deny-egress rule R_k in $\langle R_1, \ldots, R_m \rangle$ such that pk_{res} is denied by R_k to leave S_{aws}, i.e.,

$$\exists R_k \cdot (R_k \in \langle R_1, \ldots, R_m \rangle) \wedge (R_k.Egress) \wedge (R_k.RuleAction = deny) \wedge (pk_{res}.ip_x \in R_k.CidrBlock) \wedge (pk_{res}.protocol = R_k.Protocol) \wedge (pk_{res}.port_x \in R_k.PortRange)$$

and that R_k has higher priority than any allow-egress rule that matches pk_{res}.

With the above security configuration, the following security objective is achieved: (i) pk_{in} is allowed to enter S_{aws}, (ii), pk_{in} is allowed to arrive at the VM M_{aws}, and (iii) pk_{in}'s response packet pk_{res} is denied to leave AWS.

Given a GCP subnet S_{gcp} to which the GCP firewall F is bound, a sequence of rules $\langle R_1, \ldots, R_n \rangle$ for some $n \geq 1$ in F, a VM, $M_{gcp} = \langle ip_m, S_{gcp}, F \rangle$, an ip packet $pk_{in} = \langle ip_x, ip_y, protocol, port_x, port_y \rangle$ such that $ip_y = ip_m$, and an ip packet generated in response to pk_{in}, i.e., $pk_{res} = \langle ip_y, ip_x, protocol, port_y, port_x \rangle$, assume there exists a security configuration that achieves the same security objective as that in AWS, i.e., (i) pk_{in} is allowed to arrive at M_{gcp}, and (ii) pk_{res} is denied to leave GCP. Suppose our assumption is true, i.e., there exists a GCP configuration that allows pk_{in} to arrive at M_{gcp}, then there either (i) exists an allow-egress rule R_i for which pk_{in} matches the statefully-added ingress rule, and such allow-egress rule which matches the ip packet pk_{res} is of higher priority than every deny-egress rule that matches pk_{res}, or, (ii) there exists a matching allow-ingress rule R_i in F for pk_{in} that is of higher priority than any deny-ingress rule that matches pk_{in}.

If (i) is true, then it is a contradiction to that pk_{res} will be denied to leave GCP because there exists an allow-egress rule R_i which matches the ip packet pk_{res} that is of higher priority than every deny-egress rule that matches pk_{res}. If (ii) is true, then there is also a contradiction because the firewall rule in GCP is stateful, i.e., if there exists an allow-ingress firewall rule R_i in F that matches pk_{in} such that pk_{in} is allowed to arrive at M_{gcp} and that R_i has higher priority than any deny-ingress rule that matches pk_{in}, then its response packet pk_{res} will also be allowed to leave GCP.

Therefore, we proved that there exists an AWS security configuration, for which no equivalent GCP security configuration exists.

Theorem 8 *For every GCP security configuration, there exists an equivalent AWS security configuration.*

We prove this theorem by case analysis. Given an ip packet pk_{ip}, a subnet S_{gcp} to which a firewall F is bound, a sequence of rules $\langle R_1, \ldots, R_n \rangle$ in F for some $n \geq 1$, a VM $M_{in} = \langle ip_{in}, S_{gcp}, F \rangle$, and a VM $M_o = \langle ip_o, S_o \rangle$ such that $ip_o \notin S_{gcp}$ and $S_{gcp} \cap S_o = \emptyset$, there are four cases arising from the GCP security semantics: (1) $pk_{ip} = \langle ip_o, ip_{in}, protocol_{ip}, port_x, port_y \rangle$ and is allowed to arrive at M_{in}, (2) $pk_{ip} = \langle ip_o, ip_{in}, protocol_{ip}, port_x, port_y \rangle$ and is denied to arrive at M_{in}, (3) $pk_{ip} = \langle ip_{in}, ip_o, protocol_{ip}, port_x, port_y \rangle$ and is allowed to leave S_{gcp}, and, (4) $pk_{ip} = \langle ip_{in}, ip_o, protocol_{ip}, port_x, port_y \rangle$ and is denied to leave S_{gcp}.

Suppose we are given the following AWS setting: a subnet S_{aws} to which a NACL N is bound, a set of ip permissions $\{P_1, \ldots, P_n\}$ for some $n \geq 1$ in a set of security groups $\{G_1, \ldots, G_k\}$ for some $k \geq 1$, a sequence of rules $\langle R_1, \ldots, R_m \rangle$ in N for some $m \geq 1$, a VM $M_{in} = \langle ip_{in}, S_{aws}, \{G_1 \ldots, G_k\} \rangle$, and a VM $M_o = \langle ip_o, S_o \rangle$ such that $ip_o \notin S_{aws}$ and $S_{aws} \cap S_o = \emptyset$. By constructing equivalent AWS security configurations, we perform the case analysis as follows:

Case 1 – Given $pk_{ip} = \langle ip_o, ip_{in}, protocol_{ip}, port_x, port_y \rangle$ and is allowed to arrive at M_{in}, We can construct the following security configuration in AWS to achieve the same objective: (i) a matching allow-ingress NACL rule $R_i \in N$ for pk_{ip} that is of higher priority than any deny-ingress rule that matches pk_{ip}, and, (ii) a matching allow-egress NACL rule $R_j \in N$ for the packet generated in response to pk_{ip}, i.e., $pk_{res} = \langle ip_{in}, ip_o, protocol_{ip}, port_y, port_x \rangle$ that is of higher priority than any deny-egress rule that matches pk_{res} and, (iii) a matching ingress ip permission $P_j \in \{P_1, \ldots, P_n\}$ for pk_{ip}.

Case 2 – Given $pk_{ip} = \langle ip_o, ip_{in}, protocol_{ip}, port_x, port_y \rangle$ and is denied to arrive at M_{in}, we can construct the following security configuration in AWS to achieve the same objective: a matching deny-ingress NACL rule $R_i \in N$ for pk_{ip} that is of higher priority than any allow-ingress rule that matches pk_{ip}

Case 3 – Given $pk_{ip} = \langle ip_{in}, ip_o, protocol_{ip}, port_x, port_y \rangle$ and is allowed to leave S_{gcp}, we can construct the following security configuration in AWS to achieve the same objective: (i) a matching allow-egress NACL rule $R_i \in N$ for pk_{ip} that is of higher priority than any deny-egress rule that matches pk_{ip}, and, (ii) a matching allow-ingress NACL rule $R_j \in N$ for the packet generated in response to pk_{ip}, i.e., $pk_{ip} = \langle ip_o, ip_{in}, protocol_{ip}, port_x, port_y \rangle$ that is of higher priority than any deny-ingress rule that matches pk_{res} and, (iii) an matching egress ip permission $P_j \in \{P_1, \ldots, P_n\}$ for pk_{ip}.

Case 4 – Given $pk_{ip} = \langle ip_{in}, ip_o, protocol_{ip}, port_x, port_y \rangle$ and is denied to leave S_{gcp}, we can construct the following security configuration in AWS to achieve the same objective: a matching deny-egress NACL rule $R_i \in N$ for pk_{ip} that is of higher priority than any allow-egress rule that matches pk_{ip}.

Thus, we have proved that for every GCP security configuration, there exists an equivalent AWS security configuration.

2.3.2.2 AWS vs. Azure

Theorem 9 *The networking security feature in Azure is not as expressive as the one in AWS. That is, there exists an AWS security configuration, for which no equivalent Azure security configuration exists.*

We prove this theorem by contradiction. We construct an AWS security configuration such that no equivalent Azure security configuration exists as follows:

Given an AWS subnet S_{aws}, a set of ip permissions $\{P_1, \ldots, P_n\}$ for some $n \geq 1$ in security group $\{G_1, \ldots, G_k\}$ for some $k \geq 1$, a VM, $M_{aws} = \langle ip_m, S_{aws}, \{G_1, \ldots, G_k\}\rangle$, a NACL N, with a sequence of rules $\langle R_1, \ldots, R_m\rangle$ for some $m \geq 1$, to which S is bound, an ip packet $pk_{in} = \langle ip_x, ip_y, protocol, port_x, port_y\rangle$ such that $ip_y = ip_m$, and an ip packet generated in response to pk_{in}, i.e., $pk_{res} = \langle ip_y, ip_x, protocol, port_y, port_x\rangle$, suppose the following security configuration exists:

There exists an ip permission P_i in $\{P_1, \ldots, P_n\}$ such that pk_{in} is allowed by P_i, i.e.,

$$\exists P_i \cdot (P_i \in \{G_1, \ldots, G_n\}) \wedge (P.direction = ingress) \wedge (pk_{in}.ip_x \in P_i.IpRanges) \wedge (pk_{in}.protocol = P_i.IpProtocol) \wedge (pk_{in}.port_y \in [P_i.FromPort, P_i.ToPort])$$

and,

there exists an allow-ingress rule R_j in $\langle R_1, \ldots, R_m\rangle$ such that pk_{in} is allowed by R_j to enter S_{aws}, i.e.,

$$\exists R_j \cdot (R_j \in \langle R_1, \ldots, R_m\rangle) \wedge (\neg R_j.Egress) \wedge (R_j.RuleAction = allow) \wedge (pk_{in}.ip_x \in R_j.CidrBlock) \wedge (pk_{in}.protocol = R_j.Protocol) \wedge (pk_{in}.port_y \in R_j.PortRange)$$

and that R_j has higher priority than any deny-ingress rule that matches pk_{in}, and

there exists a deny-egress rule R_k in $\langle R_1, \ldots, R_m\rangle$ such that pk_{res} is denied by R_k to leave S_{aws}, i.e.,

$$\exists R_k \cdot (R_k \in \langle R_1, \ldots, R_m\rangle) \wedge (R_k.Egress) \wedge (R_k.RuleAction = deny) \wedge (pk_{res}.ip_x \in R_k.CidrBlock) \wedge (pk_{res}.protocol = R_k.Protocol) \wedge (pk_{res}.port_x \in R_k.PortRange)$$

and that R_k has higher priority than any allow-egress rule that matches pk_{res}.

With the above security configuration, the following security objective is achieved: (i) pk_{in} is allowed to enter S_{aws}, (ii), pk_{in} is allowed to arrive at the VM M_{aws}, and (iii) pk_{in}'s response packet pk_{res} is denied to leave AWS.

Given an Azure subnet S_{azure} to which an NSG G_i with a sequence of rules $\{R_i^1, \ldots, R_i^n\}$ for some $n \geq 1$ is bound, a VM, $M_{azure} = \langle ip_m, S_{azure}, G_j \rangle$ to which another NSG G_j with a sequence of rules $\{R_j^1, \ldots, R_j^k\}$ for some $k \geq 1$ is bound, an ip packet $pk_{in} = \langle ip_x, ip_y, protocol, port_x, port_y \rangle$ such that $ip_y = ip_m$, and an ip packet generated in response to pk_{in}, i.e., $pk_{res} = \langle ip_y, ip_x, protocol, port_y, port_x \rangle$, suppose there exists a configuration that achieves the same security objective as that in AWS, i.e., allows $pk_{ingress}$ to arrive at M_{azure} and denies pk_{res} to leave Azure. Since there exists a configuration that allows pk_{in} to arrive at M_{azure}, by the semantics of Azure networking security, (i) G_i allows pk_{in} to enter S_{azure}, and, (ii) G_j allows pk_{in} to reach M_{azure} which means the response packet pk_{res} generated in response to pk_{in} is also allowed to leave Azure due to the statefulness of NSG rules. This is a contradiction to the assumption that the configuration denies pk_{res} to leave Azure.

Therefore, we have proven that there exists an AWS security configuration, for which no equivalent Azure security configuration exists.

Theorem 10 *For every Azure security configuration, there exists an equivalent AWS security configuration.*

We prove this theorem by case analysis. Given an ip packet pk_{ip}, a subnet S_{azure} to which a NSG G_i is bound, a sequence of rules $\langle R_1, \ldots, R_n \rangle$ in G_i for some $n \geq 1$, another NSG G_j with a sequence of rules $\langle R_1, \ldots, R_m \rangle$ for some $m \geq 1$, a VM $M_{in} = \langle ip_{in}, S_{azure}, G_j \rangle$, and a VM $M_o = \langle ip_o, S_o \rangle$ such that $ip_o \notin S_{azure}$ and $S_{azure} \cap S_o = \emptyset$, there are four cases arising from the Azure security semantics: (1) $pk_{ip} = \langle ip_o, ip_{in}, protocol_{ip}, port_x, port_y \rangle$ which is allowed to enter S_{azure} and is allowed to arrive at M_{in}, (2) $pk_{ip} = \langle ip_o, ip_{in}, protocol_{ip}, port_x, port_y \rangle$ which is allowed to enter S_{azure} and is denied to arrive at M_{in} (3) $pk_{ip} = \langle ip_o, ip_{in}, protocol_{ip}, port_x, port_y \rangle$ which is denied to enter S_{azure}. (4) $pk_{ip} = \langle ip_{in}, ip_o, protocol_{ip}, port_x, port_y \rangle$ which is allowed to leave M_{in} and is allowed to leave S_{azure}. (5) $pk_{ip} = \langle ip_{in}, ip_o, protocol_{ip}, port_x, port_y \rangle$ which is allowed to leave M_{in} and is denied to leave S_{azure}. (6) $pk_{ip} = \langle ip_{in}, ip_o, protocol_{ip}, port_x, port_y \rangle$ which is denied to leave M_{in}.

Suppose we are given the following AWS setting: a subnet S_{aws} to which a NACL N is bound, a set of ip permissions $\{P_1, \ldots, P_n\}$ for some $n \geq 1$ in a set of security groups $\{G_1, \ldots, G_k\}$ for some $k \geq 1$, a sequence of rules $\langle R_1, \ldots, R_m \rangle$ in N for some $m \geq 1$, a VM $M_{in} = \langle ip_{in}, S_{aws}, \{G_1 \ldots, G_k\} \rangle$, and a VM $M_o = \langle ip_o, S_o \rangle$ such that $ip_o \notin S_{aws}$ and $S_{aws} \cap S_o = \emptyset$. By constructing equivalent AWS security configurations, we perform the case analysis as follows:

Case 1 – Given $pk_{ip} = \langle ip_o, ip_{in}, protocol_{ip}, port_x, port_y \rangle$ which is allowed to enter S_{azure} and is allowed to arrive at M_{in}, we can construct the following security configuration in AWS to achieve the same objective: (i) a matching allow-ingress NACL rule $R_i \in N$ for pk_{ip} that is of higher priority than any deny-ingress rule that matches pk_{ip}, and (ii) a matching allow-egress NACL rule $R_j \in N$ for the packet generated in response to pk_{ip}, i.e., $pk_{res} = \langle ip_{in}, ip_o, protocol_{ip}, port_y, port_x \rangle$ that

is of higher priority than any deny-egress rule that matches pk_{res} and, (iii) an matching ingress ip permission $P_j \in \{P_1, \ldots, P_n\}$ for pk_{ip}.

Case 2 – Given $pk_{ip} = \langle ip_o, ip_{in}, protocol_{ip}, port_x, port_y \rangle$ which is allowed to enter S_{azure} and is denied to arrive at M_{in}, we can construct the following security configuration in AWS to achieve the same objective: (i) a matching allow-ingress NACL rule $R_i \in N$ for pk_{ip} that is of higher priority than any deny-ingress rule that matches pk_{ip}, and (ii) a matching allow-egress NACL rule $R_j \in N$ for the packet generated in response to pk_{ip}, i.e., $pk_{res} = \langle ip_{in}, ip_o, protocol_{ip}, port_y, port_x \rangle$ that is of higher priority than any deny-egress rule that matches pk_{res} and, (iii) no matching ingress ip permission exists for pk_{ip} for all ip permission $P_j \in \{P_1, \ldots, P_n\}$ for $1 \leq j \leq n$.

Case 3 – Given $pk_{ip} = \langle ip_o, ip_{in}, protocol_{ip}, port_x, port_y \rangle$ which is denied to enter S_{azure}, we can construct the following security configuration in AWS to achieve the same objective: a matching deny-ingress NACL rule $R_i \in N$ for pk_{ip} that is of higher priority than any allow-ingress rule that matches pk_{ip},

Case 4 – Given $pk_{ip} = \langle ip_{in}, ip_o, protocol_{ip}, port_x, port_y \rangle$ which is allowed to leave M_{in} and is allowed to leave S_{azure}, we can construct the following security configuration in AWS to achieve the same objective: (i) a matching allow-egress NACL rule $R_i \in N$ for pk_{ip} that is of higher priority than any deny-egress rule that matches pk_{ip}, (ii) a matching allow-ingress NACL rule $R_j \in N$ for the packet generated in response to pk_{ip}, i.e., $pk_{ip} = \langle ip_o, ip_{in}, protocol_{ip}, port_x, port_y \rangle$ that is of higher priority than any deny-ingress rule that matches pk_{res} and, (iii) a matching egress ip permission $P_j \in \{P_1, \ldots, P_n\}$ for pk_{ip}.

Case 5 – Given $pk_{ip} = \langle ip_{in}, ip_o, protocol_{ip}, port_x, port_y \rangle$ which is allowed to leave M_{in} and is denied to leave S_{azure}, we can construct the following security configuration in AWS to achieve the same objective: (i) a matching egress ip permission $P_j \in \{P_1, \ldots, P_n\}$ for pk_{ip}, (ii) a matching deny-egress NACL rule $R_i \in N$ for pk_{ip} that is of higher priority than any allow-egress rule that matches pk_{ip}.

Case 6 – Given $pk_{ip} = \langle ip_{in}, ip_o, protocol_{ip}, port_x, port_y \rangle$ which is denied to leave M_{in}, we can construct the following security configuration in AWS to achieve the same objective: (i) no matching egress ip permission exists for pk_{ip} for all ip permission $P_j \in \{P_1, \ldots, P_n\}$ for $1 \leq j \leq n$.

Thus, we have proved that for every Azure security configuration, there exists an equivalent AWS security configuration.

2.3.2.3 Azure vs. GCP

Theorem 11 *For every GCP security configuration, there exists an equivalent Azure security configuration.*

We prove this theorem by case analysis. Given an ip packet pk_{ip}, a subnet S_{gcp} to which a firewall F is bound, a sequence of rules $\langle R_1, \ldots, R_n \rangle$ in F for some $n \geq 1$, a VM $M_{in} = \langle ip_{in}, S_{gcp}, F \rangle$, and a VM $M_o = \langle ip_o, S_o \rangle$ such that $ip_o \notin S_{gcp}$ and $S_{gcp} \cap S_o = \emptyset$, there are four cases arising from the GCP security semantics: (1) $pk_{ip} = \langle ip_o, ip_{in}, protocol_{ip}, port_x, port_y \rangle$ and is allowed to arrive at M_{in}, (2) $pk_{ip} = \langle ip_o, ip_{in}, protocol_{ip}, port_x, port_y \rangle$ and is denied to arrive at M_{in}, (3) $pk_{ip} = \langle ip_{in}, ip_o, protocol_{ip}, port_x, port_y \rangle$ and is allowed to leave S_{gcp}, and, (4) $pk_{ip} = \langle ip_{in}, ip_o, protocol_{ip}, port_x, port_y \rangle$ and is denied to leave S_{gcp}.

Suppose we are given the following Azure setting: a subnet S_{azure} to which a NSG G_i is bound, a sequence of rules $\langle R_1, \ldots, R_n \rangle$ in G_i for some $n \geq 1$, another NSG G_j with a sequence of rules $\langle R_1, \ldots, R_m \rangle$ for some $m \geq 1$, a VM $M_{in} = \langle ip_{in}, S_{azure}, G_j \rangle$, and a VM $M_o = \langle ip_o, S_o \rangle$ such that $ip_o \notin S_{azure}$ and $S_{azure} \cap S_o = \emptyset$. By constructing equivalent Azure security configurations, we perform the case analysis as follows:

Case 1 – Given $pk_{ip} = \langle ip_o, ip_{in}, protocol_{ip}, port_x, port_y \rangle$ and is allowed to arrive at M_{in}, We can construct the following security configuration in Azure to achieve the same objective: (i) a matching allow-ingress rule $R_f \in G_i$ for pk_{ip} that is of higher priority than any deny-ingress rule in G_i that matches pk_{ip}, and, (ii) a matching allow-ingress rule $R_g \in G_j$ for pk_{ip} that is of higher priority than any deny-ingress rule G_j that matches pk_{ip}.

Case 2 – Given $pk_{ip} = \langle ip_o, ip_{in}, protocol_{ip}, port_x, port_y \rangle$ and is denied to arrive at M_{in}, we can construct the following security configuration in Azure to achieve the same objective: (i) a matching allow-ingress rule $R_f \in G_i$ for pk_{ip} that is of higher priority than any deny-ingress rule in G_i that matches pk_{ip}, and, (ii) a matching deny-ingress rule $R_g \in G_j$ for pk_{ip} that is of higher priority than any allow-ingress rule in G_j that matches pk_{ip}.

Case 3 – Given $pk_{ip} = \langle ip_{in}, ip_o, protocol_{ip}, port_x, port_y \rangle$ and is allowed to leave S_{gcp}, we can construct the following security configuration in Azure to achieve the same objective: (i) a matching allow-egress rule $R_f \in G_i$ for pk_{ip} that is of higher priority than any deny-egress rule in G_i that matches pk_{ip}, and, (ii) a matching allow-egress rule $R_g \in G_j$ for pk_{ip} that is of higher priority than any deny-egress rule G_j that matches pk_{ip}.

Case 4 – Given $pk_{ip} = \langle ip_{in}, ip_o, protocol_{ip}, port_x, port_y \rangle$ and is denied to leave S_{gcp}, we can construct the following security configuration in Azure to achieve the same objective: (i) a matching deny-egress rule $R_f \in G_i$ for pk_{ip} that is of higher priority than any deny-ingress rule in G_i that matches pk_{ip}

Thus, we have proved that for every GCP security configuration, there exists an equivalent Azure security configuration.

Theorem 12 *For every Azure security configuration, there exists an equivalent GCP security configuration.*

We prove this theorem by case analysis. Given an ip packet pk_{ip}, a subnet S_{azure} to which a NSG G_i is bound, a sequence of rules $\langle R_1, \ldots, R_n \rangle$ in G_i for some $n \geq 1$, another NSG G_j with a sequence of rules $\langle R_1, \ldots, R_m \rangle$ for some $m \geq 1$, a VM $M_{in} = \langle ip_{in}, S_{azure}, G_j \rangle$, and a VM $M_o = \langle ip_o, S_o \rangle$ such that $ip_o \notin S_{azure}$ and $S_{azure} \cap S_o = \emptyset$, there are four cases arising from the Azure security semantics: (1) $pk_{ip} = \langle ip_o, ip_{in}, protocol_{ip}, port_x, port_y \rangle$ which is allowed to enter S_{azure} and is allowed to arrive at M_{in}, (2) $pk_{ip} = \langle ip_o, ip_{in}, protocol_{ip}, port_x, port_y \rangle$ which is allowed to enter S_{azure} and is denied to arrive at M_{in} (3) $pk_{ip} = \langle ip_o, ip_{in}, protocol_{ip}, port_x, port_y \rangle$ which is denied to enter S_{azure}. (4) $pk_{ip} = \langle ip_{in}, ip_o, protocol_{ip}, port_x, port_y \rangle$ which is allowed to leave M_{in} and is allowed to leave S_{azure}. (5) $pk_{ip} = \langle ip_{in}, ip_o, protocol_{ip}, port_x, port_y \rangle$ which is allowed to leave M_{in} and is denied to leave S_{azure}. (6) $pk_{ip} = \langle ip_{in}, ip_o, protocol_{ip}, port_x, port_y \rangle$ which is denied to leave M_{in}.

Suppose we are given the following GCP setting: a subnet S_{gcp} to which a firewall F is bound, a sequence of rules $\langle R_1, \ldots, R_n \rangle$ in F for some $n \geq 1$, a VM $M_{in} = \langle ip_{in}, S_{gcp}, F \rangle$, and a VM $M_o = \langle ip_o, S_o \rangle$ such that $ip_o \notin S_{gcp}$ and $S_{gcp} \cap S_o = \emptyset$. By constructing equivalent GCP security configurations, we perform the case analysis as follows:

Case 1 – Given $pk_{ip} = \langle ip_o, ip_{in}, protocol_{ip}, port_x, port_y \rangle$ which is allowed to enter S_{azure} and is allowed to arrive at M_{in}, we can construct the following security configuration in GCP to achieve the same objective: (i) a matching allow-ingress rule R_i in F for ip_{pk} that is of higher priority than every deny-ingress rule in F that matches pk_{ip}.

Case 2 – Given $pk_{ip} = \langle ip_o, ip_{in}, protocol_{ip}, port_x, port_y \rangle$ which is allowed to enter S_{azure} and is denied to arrive at M_{in}, we can construct the following security configuration in AWS to achieve the same objective: (i) a matching deny-ingress rule R_i in F for ip_{pk} that is of higher priority than every allow-ingress rule in F that matches pk_{ip}.

Case 3 – Given $pk_{ip} = \langle ip_o, ip_{in}, protocol_{ip}, port_x, port_y \rangle$ which is denied to enter S_{azure}, we can construct the following security configuration in AWS to achieve the same objective: (i) a matching deny-ingress rule R_i in F for ip_{pk} that is of higher priority than every allow-ingress rule in F that matches pk_{ip}.

Case 4 – Given $pk_{ip} = \langle ip_{in}, ip_o, protocol_{ip}, port_x, port_y \rangle$ which is allowed to leave M_{in} and is allowed to leave S_{azure}, we can construct the following security configuration in AWS to achieve the same objective: (i) a matching allow-egress rule R_i in F for ip_{pk} that is of higher priority than every deny-egress rule in F that matches pk_{ip}.

Case 5 – Given $pk_{ip} = \langle ip_{in}, ip_o, protocol_{ip}, port_x, port_y \rangle$ which is allowed to leave M_{in} and is denied to leave S_{azure}, we can construct the following security configuration in AWS to achieve the same objective: (i) a matching deny-egress rule R_i in F for ip_{pk} that is of higher priority than every allow-egress rule in F that matches pk_{ip}.

Case 6 – Given $pk_{ip} = \langle ip_{in}, ip_o, protocol_{ip}, port_x, port_y \rangle$ which is denied to leave M_{in}, we can construct the following security configuration in AWS to achieve the same objective: (i) a matching deny-egress rule R_i in F for ip_{pk} that is of higher priority than every allow-egress rule in F that matches pk_{ip}.

Thus, we have proved that for every Azure security configuration, there exists an equivalent GCP security configuration.

2.4 The Expressive Poweras Problem in Multi-cloud Security Attributes

In this section, we briefly discuss the expressive power problem pertaining to multi-cloud security attributes. Specifically, we try to answer the following question – given two clouds, C_1 and C_2, does C_1's security syntax offer more than that of C_2's?

With the help of proofs presented in section 2.3, the answer to this question is straightforward. We concluded that *AWS's security syntax is more "expressive" than Azure's and GCP's, and thus, AWS's security syntax is more expressive than Azure's and GCP's under the assumption that the end-to-end connectivity involves one end, e.g., a VM, inside the cloud's subnet, and the other end outside the cloud's subnet.* The reasons are as follows:

1. **Theorem 7**: There exists an AWS security configuration for which no equivalent GCP configuration exists, and,
2. **Theorem 9**: There exists an AWS security configuration for which no equivalent Azure configuration exists.

Under the same assumption, and based on theorem 11 and theorem 12, we concluded that GCP and Azure's security syntax are equivalent, and thus, are equally expressive.

From theorem 1 to 6, we conclude that *under the assumption that the end-to-end connectivity involves both ends within the same subnet, AWS, GCP, and Azure's security syntax are equivalent, and thus, are equally expressive*.

2.5 Efficient Algorithm for Cloud Security Migration

In this section, we propose six efficient algorithms for translating a given cloud security configuration to the security configuration in a different cloud. Specifically, the proposed algorithm does the following: Given a cloud configuration C_A in cloud A as an input, the algorithm outputs the "tightest" equivalent security configuration C_B in cloud B as output. By "tightest", we mean a

security configuration that allows whatever traffic, in the form of ip packets, that C_A allows while denying as many ip packets that C_A denies.

We introduced the notion of "tightest" in this context to incorporate the notion of *statefulness* vs. *statelessnes*. This approach maximizes the availability of network traffic while trade-off is the confidentiality.

2.5.1 Migrating AWS Security Configuration to Azure

Given the following AWS setting and security configuration:

1. an AWS subnet S_{aws},
2. a set of security groups $\{G_1, \ldots, G_k\}$ for some $k \geq 1$,
3. a VM, $M_{aws} = \langle ip_{aws_m}, S_{aws}, \{G_1, \ldots, G_k\} \rangle$,
4. a NACL N, with a sequence of rules $\langle R_1, \ldots, R_m \rangle$ for some $m \geq 1$, to which S is bound

and the following Azure setting:

1. an Azure subnet S_{azure},
2. a VM, $M_{azure} = \langle ip_{azure_m}, S_{azure} \rangle$,

we propose the following algorithm 1 to efficiently migrate the AWS security configuration to Azure by preserving availability over security.

Algorithm 1 Migrating AWS Security Configuration to Azure - Part 1

```
1:  function CreateAzureVmNsgFromAwsSg(securityGroups[])
2:      azureVmNsg = []
3:      priority = 100
4:      for each s ∈ awsSecurityGroups do:
5:          for each IpPermission ∈ s do:
6:              initialize a new nsgRule
7:              nsgRule.Direction = IpPermission.Direction == ingress ? inbound : outbound
8:              nsgRule.Priority = priority
9:              nsgRule.Access = "allow"
10:             nsgRule.Protocol = IpPermission.IpProtocol
11:             if nsgRule.Direction == inbound then
12:                 nsgRule.SourceIpRange = IpPermission.IpRanges
13:                 nsgRule.SourcePortRange = any
14:                 nsgRule.DestinationIpRange = any
15:                 nsgRule.DestinationPortRange = IpPermission.PortRange
16:             else
17:                 nsgRule.SourceIpRange = any
18:                 nsgRule.SourcePortRange = IpPermission.PortRange
19:                 nsgRule.DestinationIpRange = IpPermission.IpRanges
20:                 nsgRule.DestinationPortRange = any
21:             end if
22:             azureVmNsg.append(nsgRule)
23:             priority = priority + 1
24:         end for
25:     end for
26:     return azureVmNsg
27: end function
```

Algorithm 1 Migrating AWS Security Configuration to Azure - Part 2

```
1: function CreateAzureSubnetNsgFromAwsNacl(awsNacl[])
2:     sortedAwsNacl = awsNacl sorted by priority in ascending order
3:     sortedAwsNaclIngress, sortedAwsNaclEgress = []
4:     azureSubnetNsg = []
5:     priority = 100
6:     for each rule ∈ awsNacl do:
7:         initialize a new nsgRule
8:         nsgRule.Direction = rule.Egress ? inbound : outbound
9:         nsgRule.Priority = priority
10:        nsgRule.Access = rule.RuleAction
11:        nsgRule.Protocol = rule.Protocol
12:        if nsgRule.Direction == inbound then
13:            nsgRule.SourceIpRange = rule.CidrBlock
14:            nsgRule.SourcePortRange = any
15:            nsgRule.DestinationIpRange = any
16:            nsgRule.DestinationPortRange = rule.PortRange
17:        else
18:            nsgRule.SourceIpRange = any
19:            nsgRule.SourcePortRange = any
20:            nsgRule.DestinationIpRange = rule.CidrBlock
21:            nsgRule.DestinationPortRange = rule.PortRange
22:        end if
23:        azureSubnetNsg.append(nsgRule)
24:        priority = priority + 1
25:    end for
26:    return azureSubnetNsg
27: end function
```

The proposed algorithm consists of two parts (functions), namely 1) *CreateAzureVmNsgFromAwsSg*, and, 2) *CreateAzureSubnetNsgFromAwsNacl*.

The function *CreateAzureVmNsgFromAwsSg* takes as an input a set of AWS Security Groups and outputs an Azure Network Security Group that should be attached to the azure VM M_{azure}. The function *CreateAzureSubnetNsgFromAwsNacl* takes as an input an AWS NACL and outputs an Azure Network Security Group that should be attached to the Azure subnet S_{azure}.

2.5.2 Migrating AWS Security Configuration to GCP

Given the following AWS setting and security configuration:

1. an AWS subnet S_{aws},
2. a set of security groups $\{G_1, \ldots, G_k\}$ for some $k \geq 1$,
3. a VM, $M_{aws} = \langle ip_{aws_m}, S_{aws}, \{G_1, \ldots, G_k\} \rangle$,
4. a NACL N, with a sequence of rules $\langle R_1, \ldots, R_m \rangle$ for some $m \geq 1$, to which S is bound

and the following GCP setting:

1. a GCP subnet S_{gcp},
2. a VM, $M_{gcp} = \langle ip_{gcp_m}, S_{gcp} \rangle$,

we propose the following algorithm 2 to efficiently migrate the AWS security configuration to GCP by preserving availability over security.

Algorithm 2 Migrating AWS Security Configuration to GCP - Part 1

```
1:  function CreateGcpFirewallFromAws(securityGroups[], awsNacl[])
2:      sortedAwsNacl = awsNacl sorted by priority in ascending order
3:      sortedAwsNaclIngress, sortedAwsNaclEgress = []
4:      gcpFirewall = []
5:      for each rule ∈ sortedAwsNacl do
6:          if rule.Egress = True then
7:              sortedAwsNaclIngress.append(rule)
8:          else
9:              sortedAwsNACLEgress.append(rule)
10:         end if
11:     end for
12:     for each s ∈ awsSecurityGroups do:
13:         for each IpPermission ∈ s do:
14:             if IpPermission.Direction = ingress then
15:                 gcpRule = createGcpRule(IpPermission, sortedAwsNaclIngress)
16:             else
17:                 gcpRule = createGcpRule(IpPermission, sortedAwsNaclEgress)
18:             end if
19:             if gcpRule ≠ ∅ then
20:                 gcpFirewall.append(gcpRule)
21:             end if
22:         end for
23:     end for
24:     return gcpFirewall
25: end function
```

Algorithm 2 Migrating AWS Security Configuration to GCP - Part 2

```
1: function CreateGcpRule(p, sortedAwsNacl)
2:     initialize gcpRule = ∅
3:     for each r ∈ sortedAwsNacl do
4:         if (p.IpProtocol == r.Protocol) ∧ (p.PortRange ∩ r.PortRange) ≠ ∅∧ (p.IpRanges ∩
   r.CidrBlock) ≠ ∅ then
5:             if r.RuleAction == allow then
6:                 gcpRule.Direction = p.Direction
7:                 gcpRule.Priority = r.priority
8:                 gcpRule.Action.Allow = ⟨r.Protocol, (p.PortRange ∩ r.PortRange) ⟩
9:                 gcpRule.Source = p.IpRanges ∩ R.CidrBlock
10:                return gcpRule
11:            else
12:                return gcpRule
13:            end if
14:        end if
15:    end for
16:    return gcpRule
17: end function
```

The proposed algorithm consists of two parts (functions), namely 1) *CreateGcpFirewallFromAws*, and, 2) *CreateGcpRule*.

The function *CreateGcpFirewallFromAws* takes as an input a set of AWS Security Groups and an AWS NACL and outputs a GCP Firewall that should be attached to the GCP VM M_{gcp}. The function *CreateGcpRule* is a helper function that takes as inputs an ip permission and either of the i) sorted ingress AWS NACL, or, ii) sorted egress AWS NACL, and outputs a GCP firewall rule.

2.5.3 Migrating Azure Security Configuration to AWS

Given the following Azure setting and security configuration:

1. an Azure subnet S_{azure},
2. a VM, $M_{azure} = \langle ip_{azure_m}, S_{azure}, NSG_{vm} \rangle$,
3. an Azure network security group NSG_{vm}
4. an Azure network security group NSG_{subnet} that is attached to S_{azure}

and the following AWS setting:

1. an AWS subnet S_{aws},
2. a VM, $M_{aws} = \langle ip_{aws_m}, S_{aws} \rangle$,

we propose the following algorithm 3 to efficiently migrate the Azure security configuration to AWS:

Algorithm 3 Migrating Azure Security Configuration to AWS - Part 1

```
1:  function CreateAwsSgFromAzureNsg(azureNsg_vm[])
2:      awsSecurityGroup ← []
3:      for each rule ∈ azureNsg_vm do:
4:          if rule.Access == allow then
5:              initialize a new IpPermisson p
6:              p.Direction ← rule.Direction == inbound ? ingress : egress
7:              p.IpProtocol ← rule.Protocol
8:              if p.Direction == ingress then
9:                  p.PortRange ← rule.DestinationPortRange
10:                 p.IpRange ← rule.SourceIpRange
11:             else
12:                 p.PortRange ← rule.DestinationPortRange
13:                 p.IpRange ← rule.DestinationIpRange
14:             end if
15:             awsSecurityGroup.append(p)
16:         end if
17:     end for
18:     return awsSecurityGroup
19: end function
```

Algorithm 3 Migrating Azure Security Configuration to AWS - Part 2

```
1: function CreateAwsNaclFromAzureNsg(azureNsg_subnet[])
2:     awsNacl ← []
3:     ruleNumber = 1
4:     for each rule ∈ azureNsg_subnet do:
5:         if rule.Access == allow then
6:             initialize new rules r_ingress and r_egress
7:             if rule.Direction == inbound then
8:                 r_ingress.Egress ← False
9:                 r_ingress.RuleAction ← allow
10:                r_ingress.RuleNumber ← ruleNumber
11:                r_ingress.Protocol ← rule.Protocol
12:                r_ingress.CidrBlock ← rule.SourceIpRange
13:                r_ingress.PortRange ← rule.DestinationPortRange
14:                r_egress.Egress ← True
15:                r_egress.RuleAction ← allow
16:                r_egress.RuleNumber ← ruleNumber
17:                r_egress.Protocol ← rule.Protocol
18:                r_egress.CidrBlock ← rule.SourceIpRange
19:                r_egress.PortRange ← rule.SourcePortRange
20:            else
21:                r_egress.Egress ← True
22:                r_egress.RuleAction ← allow
23:                r_egress.RuleNumber ← ruleNumber
24:                r_egress.Protocol ← rule.Protocol
25:                r_egress.CidrBlock ← rule.DestinationIpRange
26:                r_egress.PortRange ← rule.DestinationPortRange
27:                r_ingress.Egress ← False
28:                r_ingress.RuleAction ← allow
29:                r_ingress.RuleNumber ← ruleNumber
30:                r_ingress.Protocol ← rule.Protocol
31:                r_ingress.CidrBlock ← rule.DestinationIpRange
32:                r_ingress.PortRange ← rule.SourcePortRange
33:            end if
34:            ruleNumber++
35:            awsNacl.append(r_ingress, r_egress)
36:        end if
37:    end for
38:    return awsNacl
39: end function
```

The proposed algorithm consists of two parts (functions), namely 1) *CreateAwsSgFromAzureNsg*, and, 2) *CreateAwsNaclFromAzureNsg*.

The function *CreateAwsSgFromAzureNsg* takes as input an Azure Network Security Groups that is attached to a VM and outputs an AWS Security Group. The function *CreateAwsNaclFromAzureNsg* takes as input an Azure Network Security Groups that is attached to a subnet and outputs an AWS NACL.

2.5.4 Migrating Azure Security Configuration to GCP

Given the following Azure setting and security configuration:

1. an Azure subnet S_{azure},
2. a VM, $M_{azure} = \langle ip_{azure_m}, S_{azure}, NSG_{vm} \rangle$,
3. an Azure network security group NSG_{vm} sorted by priority in ascending order
4. an Azure network security group NSG_{subnet} sorted by priority in ascending order that is attached to S_{azure}

and the following GCP setting:

1. a GCP subnet S_{gcp},
2. a VM, $M_{gcp} = \langle ip_{gcp_m}, S_{gcp} \rangle$,

we propose the following algorithm 4 to efficiently migrate the Azure security configuration to GCP:

Algorithm 4 Migrating Azure Security Configuration to GCP - Part 1

```
1:  function CreateGcpFirewallFromAzureNsg(azureNsg_vm[], azureNsg_subnet[])
2:      azureNsg_subnetIngress, azureNsg_subnetEgress ← []
3:      gcpFirewall ← []
4:      for each rule ∈ azureNsg_subnet do
5:          if rule.Direction = inbound then
6:              azureNsg_subnetIngress.append(rule)
7:          else
8:              azureNsg_subnetEgress.append(rule)
9:          end if
10:     end for
11:     for each rule ∈ azureNsg_vm do:
12:         gcpRule ← ∅
13:         if rule.Direction = inbound then
14:             gcpRule = createGcpIngressRule(rule, azureNsg_subnetIngress)
15:         else
16:             gcpRule = createGcpRule(rule, azureNsg_subnetEgress)
17:         end if
18:         if gcpRule ≠ ∅ then
19:             gcpFirewall.append(gcpRule)
20:         end if
21:     end for
22:     return gcpFirewall
23: end function
```

Algorithm 4 Migrating Azure Security Configuration to GCP - Part 2

```
1: function CreateGcpIngressRule(p, azureNsg_subnet)
2:     initialize gcpRule = ∅
3:     for each r ∈ azureNsg_subnet do
4:         if (p.Protocol == r.Protocol) ∧ (p.DestinationPortRange ∩ r.DestinationPortRange)
   ≠ ∅∧ (p.SourceIpRange ∩ r.SourceIpRange) ≠ ∅ then
5:             if r.Access == allow and p.Access == allow then
6:                 gcpRule.Direction = p.Direction
7:                 gcpRule.Priority = r.priority
8:                 gcpRule.Action.Allow = ⟨r.Protocol, (p.DestinationPortRange ∩
   r.DestinationPortRange) ⟩
9:                 gcpRule.Source = p.SourceIpRange ∩ r.SourceIpRange
10:                return gcpRule
11:            else
12:                return gcpRule
13:            end if
14:        end if
15:    end for
16:    return gcpRule
17: end function
```

Algorithm 4 Migrating Azure Security Configuration to GCP - Part 3

```
1: function CreateGcpEgressRule(p, azureNsg_subnet)
2:     initialize gcpRule = ∅
3:     for each r ∈ azureNsg_subnet do
4:         if (p.Protocol == r.Protocol) ∧ (p.DestinationPortRange ∩ r.DestinationPortRange)
   ≠ ∅∧ (p.DestinationIpRange ∩ r.DestinationIpRange) ≠ ∅ then
5:             if r.Access == allow and p.Access == allow then
6:                 gcpRule.Direction = p.Direction
7:                 gcpRule.Priority = r.priority
8:                 gcpRule.Action.Allow = ⟨r.Protocol, (p.DestinationPortRange ∩
   r.DestinationPortRange) ⟩
9:                 gcpRule.Source = p.DestinationIpRange ∩ r.DestinationIpRange
10:                return gcpRule
11:            else
12:                return gcpRule
13:            end if
14:        end if
15:    end for
16:    return gcpRule
17: end function
```

The proposed algorithm consists of three parts (functions), namely 1) *CreateGcpFirewallFromAzureNsg*, 2) *CreateGcpIngressRule*, and, 3) *CreateGcpEgressRule*

The function *CreateGcpFirewallFromAzureNsg* takes as input an Azure NSG that is attached to a VM and another Azure NSG that is attached to a subnet and outputs an equivalent GCP firewall. Both functions *CreateGcpIngressRule* and *CreateGcpEgressRule* are helper functions that take as inputs an NSG_{vm} rule and and NSG_{subnet} and output a GCP firewall rule.

2.5.5 Migrating GCP Security Configuration to AWS

Given the following GCP setting and security configuration:

1. a GCP subnet S_{gcp},
2. a GCP VM, $M_{gcp} = \langle ip_{gcp_m}, S_{gcp}, F \rangle$,
3. a GCP firewall F that is attached to S_{gcp}

and the following AWS setting:

1. an AWS subnet S_{aws},
2. a VM, $M_{aws} = \langle ip_{aws_m}, S_{aws} \rangle$,

we propose the following algorithm 5 to efficiently migrate the GCP security configuration to AWS:

Algorithm 5 Migrating GCP Security Configuration to AWS - Part 1

```
1:  function CreateAwsSgFromGcpFirewall(gcpFirewall[])
2:      awsSecurityGroup ← []
3:      for each rule ∈ gcpFirewall do
4:          if rule.Action.Allow != ∅ then
5:              initialize a new IpPermisson p
6:              p.Direction ← rule.Direction
7:              p.IpProtocol ← rule.Action.Allow.Protocol
8:              if p.Direction == ingress then
9:                  p.PortRange ← rule.PortRange
10:                 p.IpRange ← rule.Source
11:             else
12:                 p.PortRange ← rule.PortRange
13:                 p.IpRange ← rule.Destination
14:             end if
15:             awsSecurityGroup.append(p)
16:         end if
17:     end for
18:     return awsSecurityGroup
19: end function
```

Algorithm 5 Migrating GCP Security Configuration to AWS - Part 2

```
1: function CreateAwsNaclFromGcpFirewall(gcpFirewall[])
2:     awsNacl ← []
3:     ruleNumber = 1
4:     for each rule ∈ gcpFirewall do:
5:         initialize new rules r_ingress and r_egress
6:         if rule.Action.Allow ≠ ∅ then
7:             if rule.Direction == ingress then
8:                 r_ingress.Egress ← False
9:                 r_ingress.RuleAcion ← allow
10:                r_ingress.RuleNumber ← ruleNumber
11:                r_ingress.Protocol ← rule.Protocol
12:                r_ingress.CidrBlock ← rule.Source
13:                r_ingress.PortRange ← rule.PortRange
14:                r_egress.Egress ← True
15:                r_egress.RuleAcion ← allow
16:                r_egress.RuleNumber ← ruleNumber
17:                r_egress.Protocol ← rule.Protocol
18:                r_egress.CidrBlock ← rule.Source
19:                r_egress.PortRange ← *
20:            else
21:                r_egress.Egress ← True
22:                r_egress.RuleAcion ← allow
23:                r_egress.RuleNumber ← ruleNumber
24:                r_egress.Protocol ← rule.Protocol
25:                r_egress.CidrBlock ← rule.Destination
26:                r_egress.PortRange ← rule.PortRange
27:                r_ingress.Egress ← False
28:                r_ingress.RuleAcion ← allow
29:                r_ingress.RuleNumber ← ruleNumber
30:                r_ingress.Protocol ← rule.Protocol
31:                r_ingress.CidrBlock ← rule.Destination
32:                r_ingress.PortRange ← *
33:            end if
34:            ruleNumber++
35:            awsNacl.append(r_ingress, r_egress)
36:        end if
37:    end for
38:    return awsNacl
39: end function
```

The proposed algorithm consists of two parts (functions), namely 1) *CreateAwsSgFromGcpFirewall*, and, 2) *CreateAwsNaclFromGcpFirewall*.

The function *CreateAwsSgFromGcpFirewall* takes as input a GCP firewall that is attached to all instances within S_{gcp} and outputs an AWS Security Group. The function *CreateAwsNaclFromGcpFirewall* takes as input the same GCP firewall that is attached to all instances within S_{gcp} and outputs an AWS NACL.

2.5.6 Migrating GCP Security Configuration to Azure

Given the following GCP setting and security configuration:

1. a GCP subnet S_{gcp},
2. a GCP VM, $M_{gcp} = \langle ip_{gcp_m}, S_{gcp}, F \rangle$,
3. a GCP firewall F that is attached to S_{gcp}

and the following Azure setting:

1. an Azure subnet S_{azure},
2. a VM, $M_{azure} = \langle ip_{azure_m}, S_{azure} \rangle$,

we propose the following algorithm 6 to efficiently migrate the GCP security configuration to Azure:

Algorithm 6 Migrating GCP Security Configuration to Azure - Part 1

```
1:  function CreateAzureNsgFromGCP(gcpFirewall[])
2:      azureVmNsg = []
3:      priority = 100
4:      for each rule ∈ gcpFirewall do:
5:          if rule.Action.Allow ≠ ∅ then
6:              initialize a new nsgRule
7:              nsgRule.Direction ← rule.Direction == ingress ? inbound : outbound
8:              nsgRule.Priority = priority
9:              nsgRule.Access = allow
10:             nsgRule.Protocol = rule.Protocol
11:             if nsgRule.Direction == inbound then
12:                 nsgRule.SourceIpRange = rule.Source
13:                 nsgRule.SourcePortRange = any
14:                 nsgRule.DestinationIpRange = any
15:                 nsgRule.DestinationPortRange = rule.PortRange
16:             else
17:                 nsgRule.SourceIpRange = any
18:                 nsgRule.SourcePortRange = any
19:                 nsgRule.DestinationIpRange = rule.Destination
20:                 nsgRule.DestinationPortRange = rule.PortRange
21:             end if
22:             azureSubnetNsg.append(nsgRule)
23:             priority = priority + 1
24:         end if
25:     end for
26:     return azureVmNsg
27: end function
```

The proposed algorithm consists of one function only, namely *CreateAzureNsgFromGCP*.

The function *CreateAzureNsgFromGCP* takes as input a GCP firewall that is attached to all instances within S_{gcp} and outputs an Azure Network Security Group. Note that, the output Azure NSG can be applied at both the VM as well as the subnet levels.

Chapter 3

Provisioning Connectivity in Multi-cloud Environment

In this chapter, the discussion will be centered around provisioning connectivity in multi-cloud environments. In section 3.1, we formally define the term connectivity in the multi-cloud setting. In section 3.2, we summarize ways to establish connectivity between different clouds 1 via cloud-managed solutions. In section 3.3, we propose a graph-based system that consists of multiple microservices for easy management of multi-cloud applications and provide the design and implementation details for this software. Finally, in section 3.4, we provide the design and implementation details of the connectivity microservice that integrates the heterogeneous and complex cloud-native APIs to provide a unified and simplified set of APIs.

3.1 Definition of Connectivity in Multi-cloud Environment

Given two virtual machines (VMs) 8 M_1 and M_2 which are defined as follows:

$$M_1 = \langle ip_1, S_1, securityAttribute_1 \rangle$$

$$M_2 = \langle ip_2, S_2, securityAttribute_2 \rangle$$

such that $ip_1 \in S_1$, $ip_2 \in S_2$, and $S_1 \cap S_2 = \emptyset$

suppose M_1 and M_2 reside in two VPCs $V_1 = \langle cloud_1, cidrBlock_1 \rangle$ and $V_2 = \langle cloud_2, cidrBlock_2 \rangle$ with *non-overlapping CIDR blocks*, i.e., $S_1 \in cidrBlock_1$, $S_2 \in cidrBlock_2$, and $cidrBlock_1 \cap cidrBlock_2 = \emptyset$. We formally define the term "an instance of connectivity between two VMs" 11 as follows:

We say that there exists *an instance of connectivity between M_1 and M_2* if and only if i). an ip packet 9 pk_{ip1}, whose $ip_{src} = ip_1$ and $ip_{dest} = ip_2$, is allowed to leave M_1 and allowed to arrive at M_2, and, ii). an ip packet 9 pk_{ip2}, whose $ip_{src} = ip_2$ and $ip_{dest} = ip_1$, is allowed to leave M_2 and allowed to arrive at M_1.

By definition 7, resources in V_1 are logically separated from resources in V_2. Thus, the above definition has two implicit implications: i). there exists *an instance of connectivity between V_1 and V_2*, and, ii). there exists no security configuration that would otherwise:

- block or deny pk_1 from leaving M_1, S_1, and V_1,
- block or deny pk_1 from arriving at M_2, S_2, and V_2,
- block or deny pk_2 from leaving M_2, S_2, and V_2,
- block or deny pk_2 from arriving at M_1, S_1, and V_1,

In this chapter, we will focus on implication i), i.e., how to establish an instance of connectivity between two VPCs so that we can then provision connectivity between VMs. We have already directed our attention to multi-cloud security attributes in chapter 2 where concerns pertaining to implication ii) were addressed.

From implication i), it is evident that in order to create an instance of connectivity between M_1 and M_2, we need to provision an instance of connectivity between V_1 and V_2, and there are mainly two ways to provision an instance of connectivity between two VPCs, namely i). via the graphic user interface (GUI) console provided by each cloud 1, or, ii). via the set of application programming interfaces (APIs) or software development kits (SDKs) provided by each cloud.

In the next subsection 3.2, we discuss the detailed procedure of provisioning connectivity between two VPCs using only the GUI consoles of the clouds 1 and see why doing so is non-ideal for complex multi-cloud systems involving provisioning a large number of instances of connectivity.

3.2 Provisioning Instances of Connectivity between VPCs

Based on our definition for clouds 1, in table 3.1 we provide a concise summary of the steps and resources necessary for establishing the connectivity between any two VPCs in the same or different cloud. Based on this table, we discuss the detailed procedure of provisioning connectivity between two VPCs using only the GUI consoles of the clouds. While the details provided in this subsection are accurate at the time of this thesis submission, we acknowledge that they may change over time at the sole discretion of the CSPs.

VPC1	VPC2	VPC1 Resources	VPC2 Resources
AWS	AWS	VPC Peering Connection, Route Table[6]	VPC Peering Connection, Route Table[6]
Azure	Azure	Azure Virtual Network Peering[12]	Azure Virtual Network Peering[12]
AWS	GCP	Internet Gateway, Customer Gateway, Virtual Private Gateway, S2S VPN Connection, Route Table[18]	Cloud Router, HA VPN Gateway, VPN tunnels, Peer VPN gateway, Firewall Rules[18]
AWS	Azure	Customer Gateway, Virtual Private Gateway, S2S VPN Connection, Route Table[5]	Virtual Network Gateway, Local Network Gateway, Connection[9]
Azure	GCP	Virtual Network Gateway, Local Network Gateway, Connection[9]	Classic VPN Gateway, VPN Tunnel, Forwarding Rules, Global Route[19]

Table 3.1: Summary of Resources for Provisioning Connectivity Between Two VPCs

3.2.1 Provisioning Connectivity: (AWS,AWS)

Suppose we have 2 VPCs, V_1 and V_2, that are both hosted on AWS under the same account and region. To provision an instance of connectivity between V_1 and V_2, we need to create a *VPC Peering* as follows [6]:

1. On the Amazon VPC console, choose **Peering connections**
2. Pick V_1 as *the requester VPC*, and V_2 as the *the accepter VPC* or vice versa.
3. Choose **Create Peering Connection**.
4. Selet the VPC peering connection created in the last setp, and choose **Action, Accept Request**
5. Edit the *main route table* for both V_1 and V_2: for each route table, add a route such that the **Destination** contains the other VPC's CIDR block range and the VPC peering created in step 3 is chosen as **the target**.

3.2.2 Provisioning Connectivity: (Azure, Azure)

Suppose we have 2 VNets, V_1 and V_2, that are both hosted on Azure [1]. To provision the connectivity between V_1 and V_2, we need to create an *Azure Virtual Network Peeering* as follows [10]:

1. On one of the VNet console, choose **Peerings** and then **Add** a peering.
2. On the **Add peering** page, under **This virtual network**:

[1] Virtual Networks (VNets) are Azure's version of VPCs

- Add a custom Peering link names
- *Allow* traffic to remote virtual network
- *Allow* traffic forwarded from remote virtual network

3. On the **Add peering** page, under **Remote virtual network**:
   - Add a custom Peering link names
   - Add the name of the other VNet as the **Virtual network**
   - *Allow* traffic to remote virtual network
   - *Allow* traffic forwarded from remote virtual network
4. After completing step 2 and 3, click on **Add**.

3.2.3 Provisioning Connectivity: (AWS, GCP)

Suppose we have 2 VPCs, V_1 and V_2, such that V_1 is hosted on AWS and V_2 is hosted on GCP. To provision the connectivity between V_1 and V_2, we need to create the following resources on both AWS and GCP as follows: [18]:

1. On the GCP console, under Hybrid Connectivity, create a **HA VPN gateway** with a custom name.
2. On the GCP console, under Hybrid Connectivity, create a **Cloud Router** with a custom route name and ASN (autonomous system number). The ASN number can be any private ASN in the range 64512-65534 or 4200000000-4294967294. Upon completing this step, a VPN gateway with two interfaces will be created. Each interface will have an external address which we will use in the following step.
3. On the AWS console, under VPC service, create a **Customer Gateways** such that the **BGP ASN** is set to the ASN specified in step 1, and the **IP address** is specified as one of the external address created in step 2.
4. On the AWS console, under VPC service, create a **Virtual Private Gateway** with custom name and attach it to the AWS VPC network.
5. On the AWS console, under VPC service, create a VPN connection, i.e., Site-to-Site (S2S) VPN connection, as follows:
   - Select **Virtual private gateway** as the Target gateway type
   - Select the virtual private gateway created in step 4 as the **Virtual private gateway**

- Select the customer gateway created in step 3 under **Customer gateway ID** as the *Existing* customer gateway.
- Select **Dynamic (requires BGP)** as the **Routing options**
- Input the CIDR block of the AWS VPC as the **Local IPv4 network CIDR**
- Input the CIDR block of the GCP VPC as the **Remote IPv4 network CIDR**

6. On the AWS console, under VPC service, download the configuration file from the S2S connection created in step 5.
7. On the GCP console, under Hybrid Connectivity, create a **Peer VPN Gateway** with a custom name and choose **two interfaces**. For each interface, provide one of the **Outside IP addresses** from file downloaded in step 6 as the **Interface IP address**.
8. On the GCP console, under Hybrid Connectivity, create two VPN tunnels as follows:
   - Select *On-prem or Non Google Cloud* as the **Peer VPN gateway**
   - Under **Peer VPN gateway name**, select the peer vpn gateway created in step 7 as the
   - Under **High availability**, select *Create a pair of VPN tunnels*
   - Under **Cloud Router**, select the cloud router created in step 2.
   - Under VPN tunnels, create 2 VPN tunnels by selecting the IP address used in step 3 as the **Associated Cloud VPN gateway interface**. Select one of the interface from the peer vpn gateway created in step 7 as the **Associated peer VPN gateway interface**. Input the pre-shared keys from the file downloaded in step 6 as the **IKE pre-shared key**
   - **Configure BGP Session** for each tunnel by providing the ASN of AWS Virtual Private Gateway created in step 4 as the **Peer ASN**. Under **Allocate BGP IPv4 address**, select *Manually* and input the following information from the configuration file downloaded in step 6: *Inside IP address of Customer Gateway* as the **Cloud Router BGP IPv4 address** and *Inside IP address of Virtual Private Gateway* as the **BGP peer IPv4 address**.

3.2.4 Provisioning Connectivity: (AWS, Azure)

Suppose we have 2 VPCs, V_1 and V_2, such that V_1 is hosted on AWS and V_2 is hosted on Azure. To provision the connectivity between V_1 and V_2, we need to create the following resources on both AWS and Azure as follows[11]:

1. On the Azure console, create a **Virtual Network Gateway** as follows:

- Select *VPN* as the **Gateway type**.
- Select *Route-based* as the **VPN type**.
- Choose one of the option from the list for **SKU** and **Generation**. These options determines the types of workloads, throughput, features, and SLAs [7].
- Select the Azure VPC as the **Virtual network**.
- Input a subnet range that lies within the Vnet CIDR block for the **Gateway subnet address range** [7].
- Create a new Public IP address for the virtual network gateway.

2. On the AWS console, under VPC service, create a **Customer Gateway** such that the IP address corresponds to the public IP address created in step 1.
3. On the AWS console, under VPC service, create a **Virtual Private Gateway** with custom name and attach it to the AWS VPC network.
4. On the AWS console, under VPC service, create a VPN connection, i.e., Site-to-Site (S2S) VPN connection, as follows:
   - Select **Virtual private gateway** as the Target gateway type
   - Select the virtual private gateway created in step 4 as the **Virtual private gateway**
   - Select the customer gateway created in step 3 under **Customer gateway ID** as the *Existing* customer gateway.
   - Select **Static** as the **Routing options**.
   - Input the Azure Vnet CIDR block as the **Static IP Prefixes**.
   - Select **IPv4** as the **Tunnel inside ip Version**.
5. On the AWS console, under VPC service, download the configuration file from the S2S connection created in step 4.
6. On the Azure console, create a **Local Network Gateway** as follows:
   - Specify the *public IP of Virtual Private Gateway* from the configuration file downloaded in step 5 as the **IP Address**.
   - Specify the AWS VPC CIDR block as the **Address Space**.
7. On the Azure console, under the Virtual Network Gateway, add a new **Connection** as follows:
   - Select *Site-to-site (IPsec)* as the **Connection Type**.

- Select the virtual network gateway created in step 1 as the **Virtual network gateway**.
- Select the local network gateway created in step 6 as the **Local network gateway**.
- Input the one of the pre-shared key from the configuration file downloaded in step 5 as the **Shared key (PSK)**.
- Select *IKEv2* as the **IKE Protocol**.

8. (Optional) To achieve high-availability, repeat step 6 and 7 to create a connection that pairs with the second tunnel of AWS S2S Connection.

3.2.5 Provisioning Connectivity: (Azure, GCP)

Suppose we have 2 VPCs, V_1 and V_2, such that V_1 is hosted on Azure and V_2 is hosted on GCP. To provision the connectivity between V_1 and V_2, we need to create the following resources on both Azure and GCP as follows [21]:

1. On the Azure console, create a **Virtual Network Gateway** as follows:
   - Select *VPN* as the **Gateway type**.
   - Select *Route-based* as the **VPN type**.
   - Choose one of the option from the list for **SKU** and **Generation**. These options determines the types of workloads, throughput, features, and SLAs [7].
   - Select the Azure VPC as the **Virtual network**.
   - Input a subnet range that lies within the Vnet CIDR block for the **Gateway subnet address range** [7].
   - Create a new Public IP address for the virtual network gateway.
2. On the GCP console, under Hybrid Connectivity, create a **Classic VPN** as follows:
   - Under the Google Compute Engine VPN gateway, select the GCP VPC as **Network**.
   - Under the Google Compute Engine VPN gateway, select **Create IP Address** as for the **IP address**.
   - Under New Tunnel, input the IP address created in step 1 as the **Remote peer IP address**. This IP address will be used later when we configure Azure Local Network Gateway.
   - Under New Tunnel, click on **generate and copy** to generate an **IKE pre-shared key**. Note down the generated pre-shared key.

- Under New Tunnel, select *Route-based* as the **Routing options** and input the Azure VNet CIDR block as the **Remote network IP ranges**.

3. On the Azure console, create a **Local Network Gateway** as follows:
   - Specify the IP address created in step 2 as the **IP Address**.
   - Specify the GCP VPC CIDR block as the **Address Space**.
4. On the Azure console, under the Virtual Network Gateway, add a new **Connection** as follows:
   - Select *Site-to-site (IPsec)* as the **Connection Type**.
   - Select the virtual network gateway created in step 1 as the **Virtual network gateway**.
   - Select the local network gateway created in step 3 as the **Local network gateway**.
   - Input the pre-shared key copied in step 2 as the **Shared key (PSK)**.
   - Select *IKEv2* as the **IKE Protocol**.

As evident from the above, provisioning connectivity in the heterogeneous cloud environment can be tedious and complex especially when instances of connectivity need to be provisioned between each of the multiple clouds. However, before we address this issue and propose a solution in subsection 3.4, we will discuss in subsection 3.3 on how to effectively capture the multi-cloud environment using a graph-based cloud management system such that our proposed solution in subsection 3.4 can be incorporated into it.

3.3 Graph-based Cloud Management System

In this subsection, we propose a graph-based system that consists of multiple microservices for easy management of multi-cloud applications and provide the design and implementation details for this system.

The purpose of this system is to provide a GUI and a backend request dispatcher that enables the communication between a user, e.g., network admin, and the clouds such that the user will be able to manage multiple clouds using only this system. This system would also include multiple microservices to which the backend dispatcher would communicate to perform specific tasks.

3.3.1 System Architecture

Figure 3.1 illustrates the generic system architecture of the proposed system. In this system, the user would have direct access to the GUI. The user would then perform a set of tasks on the

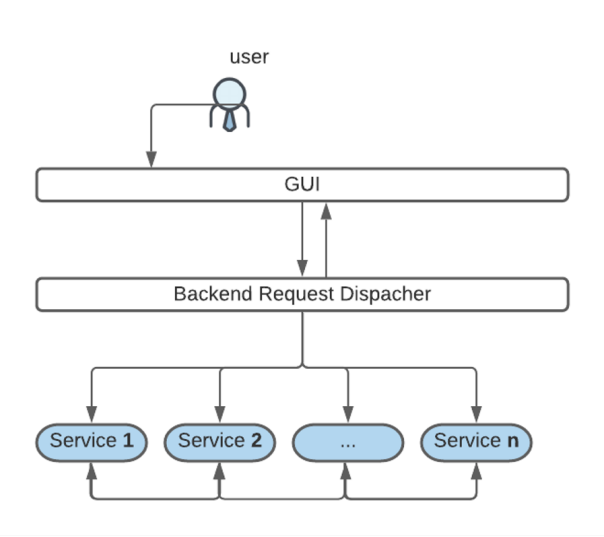

Figure 3.1: System Architecture for Proposed Cloud Management System

GUI where the GUI will, based on the tasks performed by the user, send requests to the *backend request dispatcher*.

The backend dispatcher acts as a centralized API gateway that direct the requests to different microservices. The benefit of having a backend request dispatcher is that it reduces coupling, the number of round trips for requests, the attack surface that would otherwise be exposed to the public, as well as cross-cutting concerns [22].

The microservices, or simply services, behind the backend request dispatcher each handles a unique set of tasks, for example, under the context of multi-cloud application management, these could be:

- a database service that reads or writes data pertaining to each cloud from or into a database
- an identity access management (IAM) service that grants or revokes users' access right to the clouds
- a connectivity service that connects network resources between different clouds
- ...

In our proposed system, we include the following microservices:

- **The graph database microservice** that leverages the graph-based database Neo4j to model multi-cloud environments.

- **The connectivity provisioning microservice** that provides a unified set of APIs to automate connectivity provisioning in multi-cloud environments.

3.3.2 The Graph Database Microservice

The graph database microservice is the knowledge-base that all layers of the system are allowed to read and optionally write. At the core, it is a database based on a popular NoSQL database known as Neo4j that stores data as a graph, i.e., graph data structure. In other words, all data within the system can be stored as either vertices or edges. The Neo4j database itself can be queried using the graph query language known as Cypher, which is a powerful, intuitive, graph-optimized query language [23]. However, the graph database microservice provides a simplified set of APIs for querying, writing, modifying, and deleting data from the Neo4j database.

There are some important properties of the graph database microservice:

1. Every vertex and edge has attributes. Attributes refer to name-value pairs.
2. A vertex can itself represent a graph at a "lower layer".
3. Similarly, an edge may represent a graph at a "lower layer."

With the help of the graph database microservice, we can model the multi-cloud environment by layers as follows:

- At the top layer, we would have nodes that represent each of the cloud 1. For example, figure 3.2 shows the modeling of the top-layer of a multi-cloud application that involves AWS, GCP, and Azure. Note that figure 3.2 shows no edge between each of the nodes because, in this illustration, there exists no network connectivity between each of the clouds.

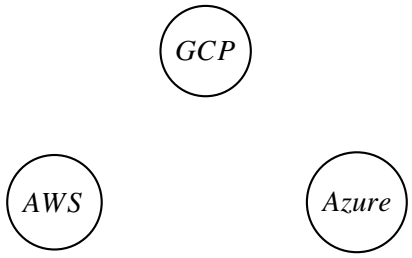

Figure 3.2: Modeling Multi-cloud Application: Cloud-layer

- At the second-to-top level, we would have nodes that represent the VPCs contained in the multi-cloud environment. For example, figure 3.3 shows 3 VPCs, each from one of the clouds 1, such that no instance of connectivity exists between each pair of VPCs.

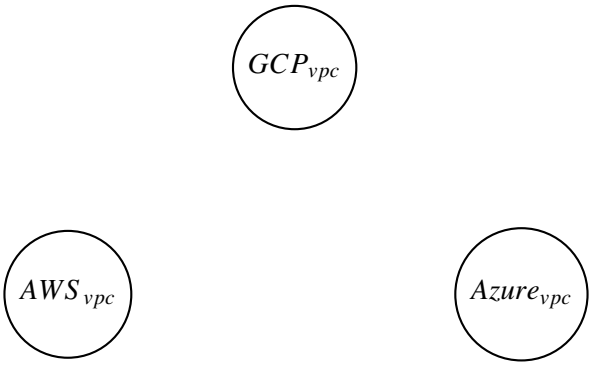

Figure 3.3: Modeling Multi-cloud Application – VPC-level without Connectivity

Of course, besides modeling the VPCs contained in the multi-cloud environment, we could also model instances of connectivity, if there are any, between VPCs using an edge. Figure 3.4 illustrates an example of such scenarios where there exists an instance of connectivity between AWS_{vpc} and GCP_{vpc} and an instance of connectivity between AWS_{vpc} and $Azure_{vpc}$, but no instance of connectivity exists between GCP_{vpc} and $Azure_{vpc}$. Formally, we could represent the graph as follows:

$$G = \langle V, E \rangle \text{ where } V = \{AWS_{vpc}, GCP_{vpc}, Azure_{vpc}\} \text{ and}$$
$$E = \{(AWS_{vpc}, GCP_{vpc}), (AWS_{vpc}, Azure_{vpc})\}$$

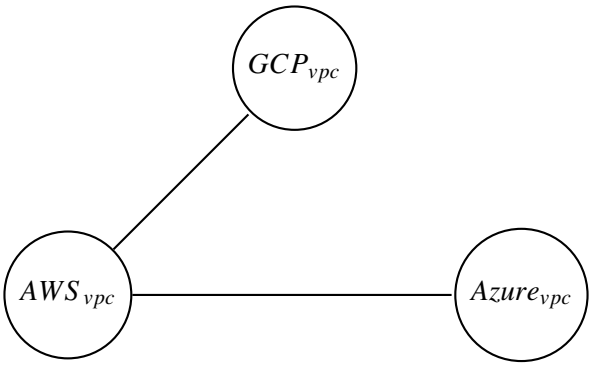

Figure 3.4: Modeling Multi-cloud Application – VPC-level with Partial Connectivity

Finally, we could also model the fully-connected topology. Figure 3.5 shows a fully connected multi-cloud network such that i). there exist two VPCs in each one of the clouds, and, ii). there exists an instance of connectivity between each pair of the VPCs. Formally, we can represent this network topology as follow: $G = \langle V, E \rangle$ such that for all $v_i, v_j \in V$, $(v_i, v_j) \in E$.

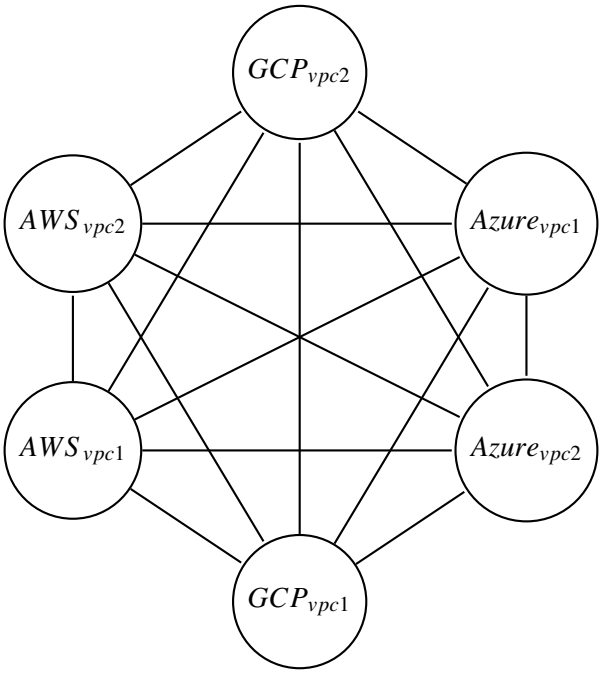

Figure 3.5: Modeling Multi-cloud Application – VPC-level with Full Mesh Connectivity

Suppose we have a multi-cloud application that has the same architecture that is shown in figure 3.5, i.e., i). there are two VPCs in each of the clouds, and, ii). there exists an instance of connectivity for each pair of the VPCs. Suppose also that we wanted to model this using the plain Cypher query language and give each edge a meaningful name to model the relationship between nodes [24], say "connectivity", then we would need the following script:

```
/* Firstly, we create 6 nodes representing 6 VPCs*/
CREATE (v:Vpc {name: 'AWS_VPC1'});
CREATE (v:Vpc {name: 'AWS_VPC2'});
CREATE (v:Vpc {name: 'GCP_VPC1'});
CREATE (v:Vpc {name: 'GCP_VPC2'});
CREATE (v:Vpc {name: 'AZURE_VPC1'});
CREATE (v:Vpc {name: 'AZURE_VPC2'});
/* Now we create pair-wise relationships */
MATCH(a:Vpc),(b:Vpc) WHERE a.name = 'AWS_VPC1' AND b.name = 'AWS_VPC2' CREATE
    (a)-[r:connectivity]->(b) RETURN type(r);
MATCH(a:Vpc),(b:Vpc) WHERE a.name = 'AWS_VPC1' AND b.name = 'GCP_VPC1' CREATE
    (a)-[r:connectivity]->(b) RETURN type(r);
```

```
11 MATCH(a:Vpc),(b:Vpc) WHERE a.name = 'AWS_VPC1' AND b.name = 'GCP_VPC2' CREATE
       (a)-[r:connectivity]->(b) RETURN type(r);
12 MATCH(a:Vpc),(b:Vpc) WHERE a.name = 'AWS_VPC1' AND b.name = 'AZURE_VPC1' CREATE
       (a)-[r:connectivity]->(b) RETURN type(r);
13 MATCH(a:Vpc),(b:Vpc) WHERE a.name = 'AWS_VPC1' AND b.name = 'AZURE_VPC2' CREATE
       (a)-[r:connectivity]->(b) RETURN type(r);
14 MATCH(a:Vpc),(b:Vpc) WHERE a.name = 'AWS_VPC2' AND b.name = 'GCP_VPC1' CREATE
       (a)-[r:connectivity]->(b) RETURN type(r);
15 MATCH(a:Vpc),(b:Vpc) WHERE a.name = 'AWS_VPC2' AND b.name = 'GCP_VPC2' CREATE
       (a)-[r:connectivity]->(b) RETURN type(r);
16 MATCH(a:Vpc),(b:Vpc) WHERE a.name = 'AWS_VPC2' AND b.name = 'AZURE_VPC1' CREATE
       (a)-[r:connectivity]->(b) RETURN type(r);
17 MATCH(a:Vpc),(b:Vpc) WHERE a.name = 'AWS_VPC2' AND b.name = 'AZURE_VPC2' CREATE
       (a)-[r:connectivity]->(b) RETURN type(r);
18 MATCH(a:Vpc),(b:Vpc) WHERE a.name = 'GCP_VPC1' AND b.name = 'GCP_VPC2' CREATE
       (a)-[r:connectivity]->(b) RETURN type(r);
19 MATCH(a:Vpc),(b:Vpc) WHERE a.name = 'GCP_VPC1' AND b.name = 'AZURE_VPC1' CREATE
       (a)-[r:connectivity]->(b) RETURN type(r);
20 MATCH(a:Vpc),(b:Vpc) WHERE a.name = 'GCP_VPC1' AND b.name = 'AZURE_VPC2' CREATE
       (a)-[r:connectivity]->(b) RETURN type(r);
21 MATCH(a:Vpc),(b:Vpc) WHERE a.name = 'GCP_VPC2' AND b.name = 'AZURE_VPC1' CREATE
       (a)-[r:connectivity]->(b) RETURN type(r);
22 MATCH(a:Vpc),(b:Vpc) WHERE a.name = 'GCP_VPC2' AND b.name = 'AZURE_VPC2' CREATE
       (a)-[r:connectivity]->(b) RETURN type(r);
23 MATCH(a:Vpc),(b:Vpc) WHERE a.name = 'AZURE_VPC1' AND b.name = 'AZURE_VPC2'
       CREATE (a)-[r:connectivity]->(b) RETURN type(r);
```

Figure 3.6 shows the graph rendered by the Neo4j graph database after running the above script in the Neo4j Browser [25]. However, given as input the number of nodes to be created, the number of queries to be written is at worst quadratic to the size of the input because it requires $\frac{n(n-1)}{2}$ edges to make a fully connected graph.

3.4 Connectivity Provisioning Microservice

As evident from subsection 3.2, establishing connectivity between two given VPCs that are hosted on different clouds can be cumbersome and time consuming as it involves creating various resources on both cloud platforms and often requires an understanding for computer networking. The situation could be further exacerbated if we wish to establish a full-mesh network topology

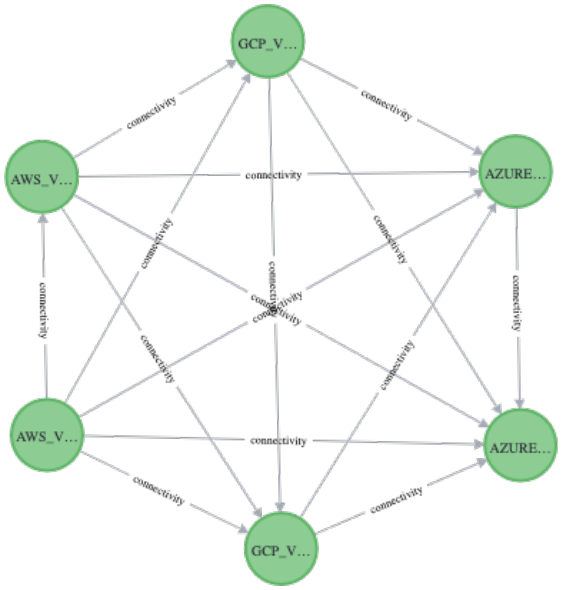

Figure 3.6: Creating an Example Graph using Cypher Language

between more than two VPCs. To eliminate the tedious work and potential misconfigurations, we propose a microservice that automates the establishment of connectivity in a multi-cloud environment by providing a suite of APIs. In the following subsections, we will present the design of the microservice and the APIs in datails.

The Connectivity Microservice

Overall, there are two main APIs for the connectivity microservice: the **Connect** API that creates the resources necessary on both clouds to connect a pair of VPCs, and the **Disconnect** API that deletes the resources created by the Connect API to disconnect two connected VPCs. The Disconnect API also provides *rollback* functionality if an error occured during the creation of relevant resources. These APIs can be modelled by the following Deterministic Finite Automaton (DFA) shown in Figure 3.7.

As can be observed from Figure 3.7, there are three final states, namely *failed*, *connected*, and *disconnected*. Upon receiving the Connect API call, the microservice transitions from the initial state to the *connecting* state where the connectivity microservice starts to create the necessary resources for the initialized connectivity between two VPCs that are specified in the API call. If an error is encountered during the connecting state, the service goes into the *failed* final state indicating that the establishment of connectivity has failed and the software stops. Once all resources

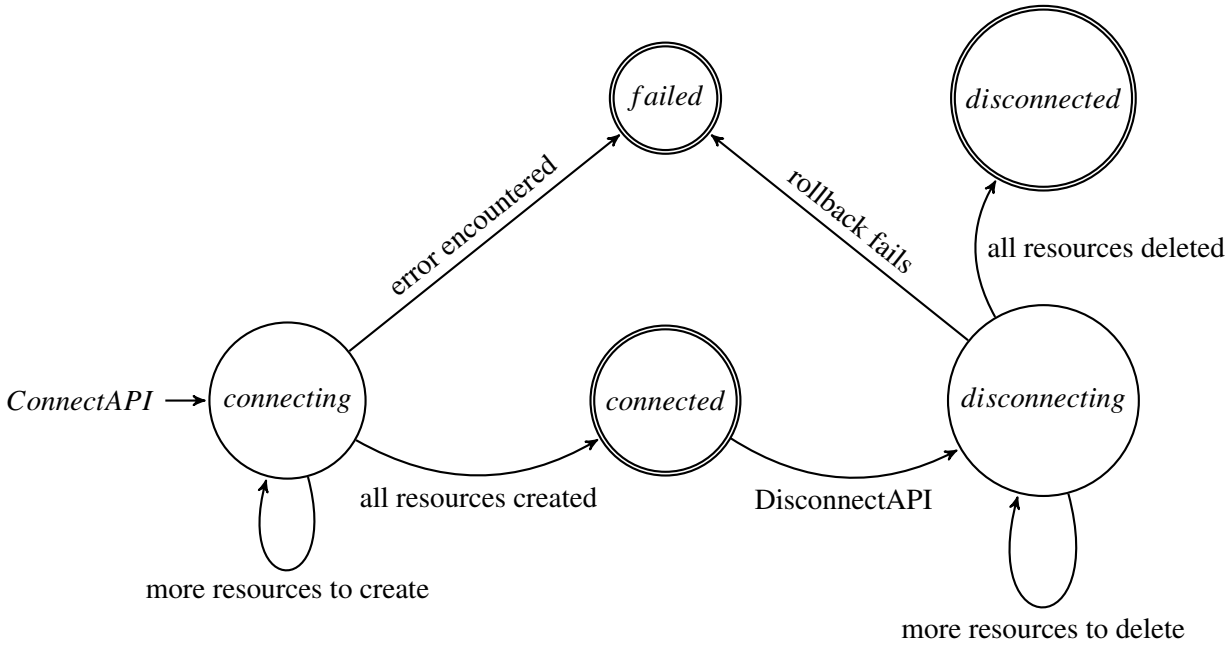

Figure 3.7: Connectivity Microservice DFA

have been created successfully, the software reaches the *connected* state and the software stops. When the *connected* final state is reached, an instance of connectivity is said to be established between the two VPCs. An instance of connectivity stay connected until a Disconnect API call is made such that the software transitions into the *disconnecting* state where the all resources created previously by the ConnectAPI call will be deleted. Should there be an error occured during the deletion process, the state will transition into the *failed* final state. Upon all resources being deleted successfully, the software transitions into the *disconnected* final state to signal that the connectivity instance between the two VPCs has been removed, i.e., no VM in one VPC is able to send and receive network traffic from VMs in the other VPC.

As evident from table 3.1, it is worth noting that for each instance of connectivity between two VPCs that is initialized by the connectivity microservice, the resources needed to create may not be the same. Furthermore, the resource creating process requires sequential ordering. For example, as described in Section 3.2, when establishing an instance of connectivity between an AWS VPC and a GCP VPC, one needs to create a Cloud Router in GCP before creating a customer gateway in AWS because the latter requires the IP address from the former.

Figure 3.8 presents a more fine-grained figurative description of the *Connect API* of the connectivity microservice, whereas the flowchart in figure 3.9 shows the details of *Disconnect API*

of the connectivity microservice. Each *helper* takes care of creating resources pertaining to the specific connection type. A connection type is an unordered pair defined by the clouds in which the two VPCs reside, for example $\{aws, gcp\}$ refers to a pair of VPCs such that one VPC belong to AWS and the other belongs to GCP.

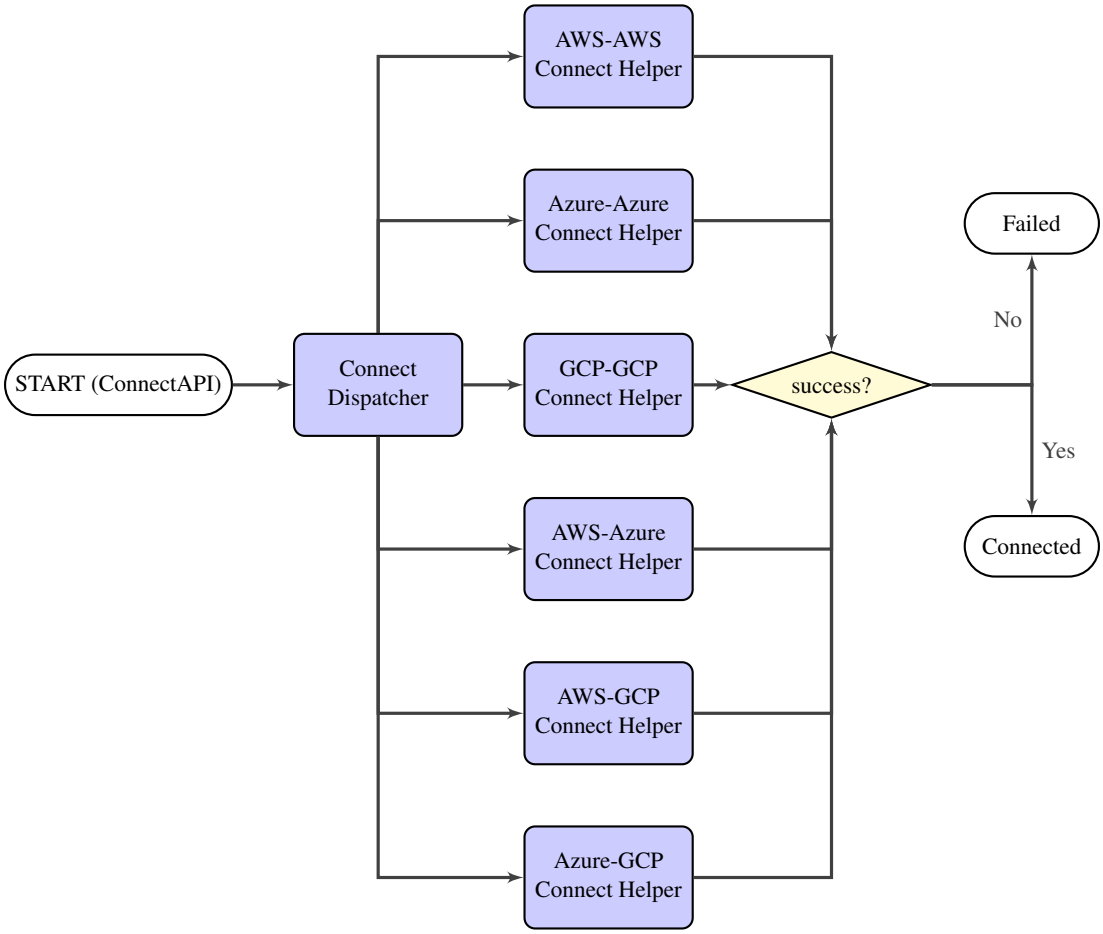

Figure 3.8: Connectivity Microservice Flowchart - Connect API

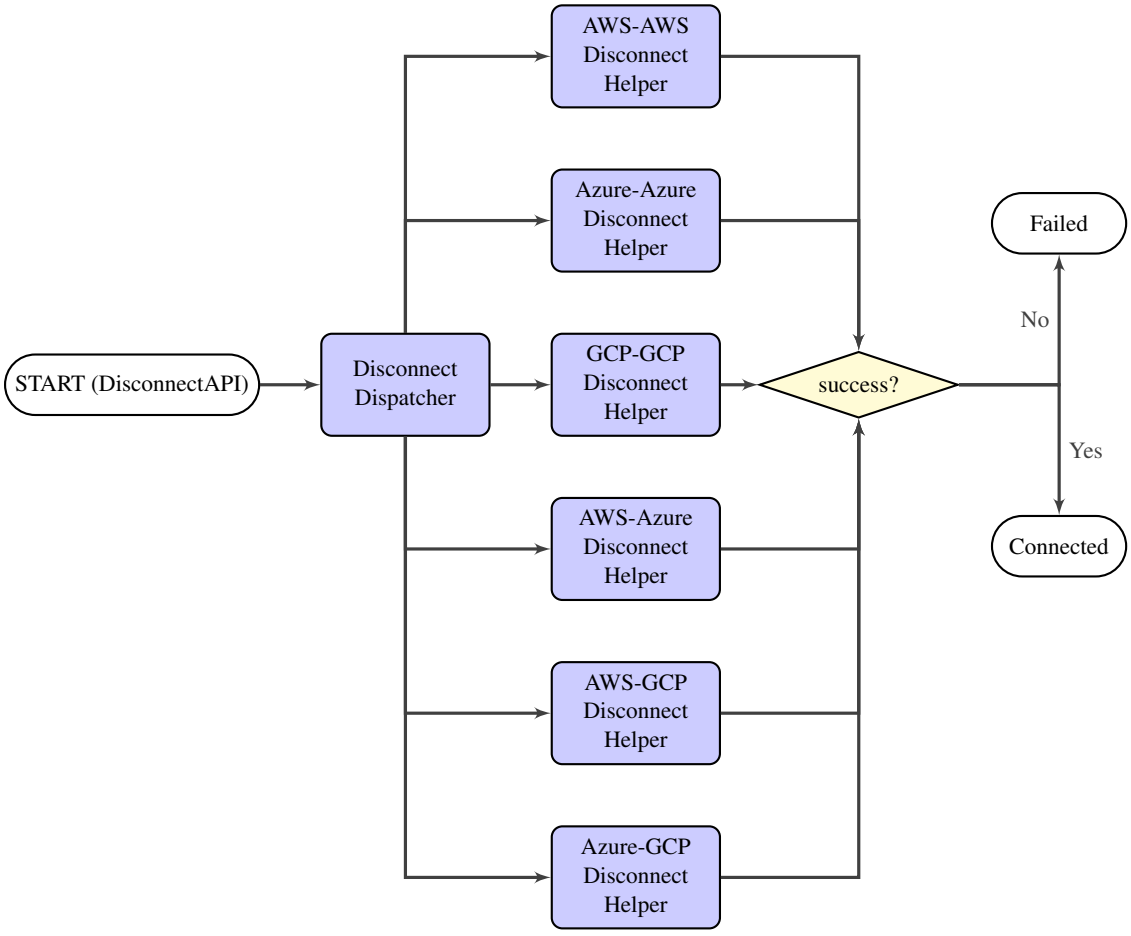

Figure 3.9: Connectivity Microservice Flowchart - Disconnect API

Chapter 4

Conclusion and Future Work

In this thesis, we explore the topic of provisioning connectivity with security attributes in multi-cloud environments.

In chapter 2, we examined the security aspects pertaining to multi-cloud connectivity. In section 2.1, we explored various connectivity-related network security features offered by different cloud service providers (CSPs) 1. Based on these features, section 2.2 gave formal definitions for the syntax and semantics of cloud security pertaining to connectivity. In section 2.3, we presented proofs of equivalence for the security attributes between clouds with the following findings – there exists an AWS security configuration such that no equivalent security configuration exists in neither GCP nor Azure. In section 2.4, we expanded our understanding of the expressive power problem for the portion of the security configuration that pertains to connectivity and came to the following conclusion:

1. Under the assumption that the end-to-end connectivity involves one end, e.g., a VM, inside the cloud's subnet, and the other end outside the cloud's subnet, AWS's security syntax is more "express" than Azure's and GCP's, and thus, AWS's security syntax is more expressive than Azure's and GCP's.
2. Under the assumption that the end-to-end connectivity involves one end, e.g., a VM, inside the cloud's subnet, and the other end outside the cloud's subnet, GCP and Azure's security syntax are equivalent, and thus, are equally expressive
3. Under the assumption that the end-to-end connectivity involves both ends within the same subnet, AWS, GCP, and Azure's security syntax are equivalent, and thus, are equally expressive.

Finally, in section 2.5, we provided six efficient algorithms that facilitate the migration from one cloud to another.

Chapter 3 focuses more on the practical aspects of this research, i.e., designing and developing graph-based multi-cloud management software in which one of the key features is to facilitate the provisioning of connectivity between clouds in the heterogeneous cloud environment. Section 3.1 gave a formal definition of the term connectivity in the multi-cloud setting. In section 3.2, we tabulated ways to provision connectivity between different clouds 1 via cloud-managed solutions. In section 3.3, we proposed a graph-based system that consists of multiple microservices for easy management of multi-cloud applications and provided details for the design and implementation of this software. Lastly, in section 3.4, we gave a deep dive into the design and implementation of the connectivity microservice that integrates the heterogeneous and complex cloud-native APIs to provide a unified and simplified set of APIs for provisioning connectivity in the multi-cloud environment.

We would like to list the following potential future works that we believe are worth exploring. Firstly, a lot of the work could be directed to the enhancement of the software mentioned in chapter 3. For example, the security attributes pertaining to connectivity provisioning that were discussed in chapter 2 have not yet been incorporated into the software. Also, another big component pertaining to cloud security is Identity Access Management (IAM) which we have not yet explored in this research work. Finally, besides security attributes, there are other important attributes pertaining to multi-cloud connectivity provisioning that are worth exploring, e.g., performance attributes such as throughput. We believe the result gathered from studying these attributes could potentially be incorporated into the aforementioned software such that the software will become more robust in provisioning multi-cloud connectivity.

References

[1] Amazon Virtual Private Cloud Documentation. https://docs.aws.amazon.com/vpc/, 2022.

[2] Infrastructure Security in Amazon VPC, 2022. URL: https://docs.aws.amazon.com/vpc/latest/userguide/vpc-network-acls.html.

[3] Infrastructure Security in Amazon VPC, 2022. URL: https://docs.aws.amazon.com/vpc/latest/userguide/infrastructure-security.html.

[4] Internetwork Traffic Privacy in Amazon VPC, 2022. URL: https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Security.html.

[5] Site-to-Site VPN Single and Multiple Connection Examples, 2022. URL: https://docs.aws.amazon.com/vpn/latest/s2svpn/Examples.html.

[6] Work with VPC Peering Connections. https://docs.aws.amazon.com/vpc/latest/peering/working-with-vpc-peering.html, 2022.

[7] About VPN Gateway Configuration Settings, 2022. URL: https://learn.microsoft.com/en-us/azure/vpn-gateway/vpn-gateway-about-vpn-gateway-settings.

[8] Azure Network Security Group, 2022. URL: https://learn.microsoft.com/en-us/azure/virtual-network/network-security-groups-overview.

[9] Connect On-premises Networks to Azure by Using Site-to-site VPN Gateways, 2022. URL: https://learn.microsoft.com/en-us/training/modules/connect-on-premises-network-with-vpn-gateway/2-connect-on-premises-networks-to-azure-using-site-to-site-vpn-gateways.

[10] Tutorial: Connect Virtual Networks with Virtual Network Peering Using the Azure Portal. https://learn.microsoft.com/en-us/azure/virtual-network/tutorial-connect-virtual-networks-portal, 2022.

[11] Tutorial: Create a site-to-site VPN connection in the Azure portal, 2022. URL: https://learn.microsoft.com/en-us/azure/vpn-gateway/tutorial-site-to-site-portall.

[12] Virtual Network Peering. https://learn.microsoft.com/en-us/azure/virtual-network/virtual-network-peering-overview, 2022.

[13] How Network Security Groups Filter Network Traffic, 2023. URL: https://learn.microsoft.com/en-us/azure/virtual-network/network-security-group-how-it-works.

[14] Alysson Bessani, Miguel Correia, Bruno Quaresma, Fernando André, and Paulo Sousa. Depsky: Dependable and secure storage in a cloud-of-clouds. *ACM Trans. Storage*, 9(4), nov 2013. URL: https://doi-org.proxy.lib.uwaterloo.ca/10.1145/2535929, doi:10.1145/2535929.

[15] Assembling Your Cloud Orchestra: a Field Guide to Multicloud Management, 2023. URL: https://www.ibm.com/thought-leadership/institute-business-value/report/multicloud.

[16] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 259–282, Boston, MA, March 2017. USENIX Association. URL: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/corrigan-gibbs.

[17] Nicolas Ferry, Alessandro Rossini, Franck Chauvel, Brice Morin, and Arnor Solberg. Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems. In *2013 IEEE Sixth International Conference on Cloud Computing*, pages 887–894, 2013. doi:10.1109/CLOUD.2013.133.

[18] Build HA VPN connections between Google Cloud and AWS, 2022. URL: https://cloud.google.com/architecture/build-ha-vpn-connections-google-cloud-aws.

[19] Create a Classic VPN Using Static Routing, 2022. URL: https://cloud.google.com/network-connectivity/docs/vpn/how-to/creating-static-vpns.

[20] GCP Firewalls, 2022. URL: https://cloud.google.com/vpc/docs/firewalls.

[21] Google Cloud VPN Interop Guide Using Cloud VPN With Microsoft Azure VPN Gateway, 2022. URL: https://cloud.google.com/files/CloudVPNGuide-UsingCloudVPNwithAzureVPN.pdf.

[22] The API Gateway Pattern Versus the Direct Client-to-microservice communication, 2022. URL: https://learn.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/direct-client-to-microservice-communication-versus-the-api-gateway-pattern.

[23] Neo4j Graph Data Platform, 2023. URL: https://neo4j.com/.

[24] Neo4j Relationships, 2023. URL: https://neo4j.com/docs/graphql-manual/current/type-definitions/relationships/.

[25] Neo4j Browser, 2023. URL: https://neo4j.com/docs/operations-manual/current/installation/neo4j-browser/.

[26] Dana Petcu. Multi-cloud: Expectations and current approaches. In *Proceedings of the 2013 International Workshop on Multi-Cloud Applications and Federated Clouds*, MultiCloud '13, page 1–6, New York, NY, USA, 2013. Association for Computing Machinery. URL: https://doi-org.proxy.lib.uwaterloo.ca/10.1145/2462326.2462328, doi:10.1145/2462326.2462328.

[27] Ansar Rafique, Dimitri Van Landuyt, Vincent Reniers, and Wouter Joosen. Towards an adaptive middleware for efficient multi-cloud data storage. In *Proceedings of the 4th Workshop on CrossCloud Infrastructures and Platforms*, Crosscloud'17, New York, NY, USA, 2017. Association for Computing Machinery. URL: https://doi-org.proxy.lib.uwaterloo.ca/10.1145/3069383.3069387, doi:10.1145/3069383.3069387.

[28] Emil Stefanov and Elaine Shi. Multi-cloud oblivious storage. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, page 247–258, New York, NY, USA, 2013. Association for Computing Machinery. URL: https://doi-org.proxy.lib.uwaterloo.ca/10.1145/2508859.2516673, doi:10.1145/2508859.2516673.

[29] Keith Stouffer, Timothy Zimmerman, CheeYee Tang, Michael Pease, Joshua Lubell, Jeffrey Cichonski, and John McCarthy. Cybersecurity framework version 1.1 manufacturing profile. *National Institute of Standards and Technology: Gaithersburg, MD, USA*, 2020.

[30] Bahador Yeganeh, Ramakrishnan Durairajan, Reza Rejaie, and Walter Willinger. A first comparative characterization of multi-cloud connectivity in today's internet. In Anna Sperotto, Alberto Dainotti, and Burkhard Stiller, editors, *Passive and Active Measurement*, pages 193–210, Cham, 2020. Springer International Publishing.