An improved mechanism for detection of Network Wide Invariants due to policy conflicts

By

AWAIS BIN ASIF

A thesis submitted to the faculty of Electrical Engineering Department Military College of Signals, National University of Sciences and Technology, Rawalpindi in partial fulfillment of the requirements for the degree of MS in Electrical Engineering

August 2019

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS/MPhil thesis written by Mr **NS Awais Bin Asif,** Registration No. **00000118613** of **Military College of Signals** has been vetted by undersigned, found complete in all respect as per NUST Statutes/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial, fulfillment for award of MS/MPhil degree. It is further certified that necessary amendments as pointed out by GEC members of the student have been also incorporated in the said thesis.

Signature: _____

Name of Supervisor: Asst Prof Muhammad Imran, PhD

Date: _____

Signature (HoD): _____

Date: _____

Signature (Dean): _____

Date: _____

ABSTRACT

Software Defined Network (SDN) is new paradigm which decouples control plane and management plane from data plane which is obtained from forwarding devices. Control plane and management planes are implemented in centralized entity called controller. Data plane encompass the network devices mainly of switches which are controlled directly by the controller in order to install the forwarding rules called flow rules. Control plane in SDN has a single or multiple software application-based controllers. It communicates and commands the forwarding devices using the OpenFlow protocol. Application plane contains different applications that interact with controller to operate and manage network. Due to centralized architecture, SDN has many advantages, for example, it makes easy to control and manage the network. Second, it allows to modify the control protocol without making modifications in the forwarding devices, i.e. it allows the network to evolve the network without involving the vendors to update the hardware of the forwarding devices. Despite of numerous advantages, SDN is more prone to logical errors like loops, black holes, reachability problem, ACL policies violation and ACL policy conflicts etc. These logical problems are called network wide invariants. In this research work we proposed a technique to detect to network wide invariants due to ACL polices conflicts. Proposed method detects conflicts in ACL polices and check if there is an overlap in ACL polices and present ACL polices in a form which is much easier to travers which consequently improves the performance of network.

DEDICATION

With the blessing of Almighty ALLAH I am able to complete this thesis and I would like to dedicate this work to my

TEACHERS and FRIENDS

For their remarkable support during the whole duration of this Masters degree.

ACKNOWLEDGEMENT

All commendation because of Almighty Allah who showered his blessing and able me to complete such exploration work.

I would like to take this opportunity to special thanks for an amazing support that my supervisor Dr. Muhammad Imran and co-supervisor Dr. Nadir Shah have been, throughout the course of my thesis. He helped me a lot to able me in completing my MS Degree, he was all time readily available for my guidance and to help me throughout my Research work.

With sincere gratitude and bundle of prayers for Him!

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

SDN: Software Define Networking

ACL: Access Control List

HSA: Header Space Analysis

RBAC: Role Based Access Control

ORBAC: Organization Based Access Control

SAT: Separating Axis Theorem.

SW: Switches

Chapter 1

INTRODUCTION

1.1 Introduction of SDN

SDN [1] is an emerging paradigm that decouple control plane and management plane from data plane. Control plane and management planes are implemented in centralized entity called controller and data plane is implemented in forwarding devices called switches. Controller manages whole network therefore network appears as a single forwarding device to controller. SDN allows vendor independent control for network management via central controller [1],[2]. Controller installs forwarding rules called flow rules in forwarding devices to deliver packets to host. Forwarding devices observe and forward network traffic on the basis of installed flow rules by the controller. This separation of control and data plane gives many advantages over traditional network like network management and implementation of network security policies e.g. ACL policies, load balancing etc. SDN controller gives overall centralized view of organization's network to network administrator, which ease network administrator to manage network. SDN allows more control to the control plane through centralized device controller, where as in traditional network, forwarding devices have tight coupling of control plane and data plane that makes a close and vendor dependent architecture.

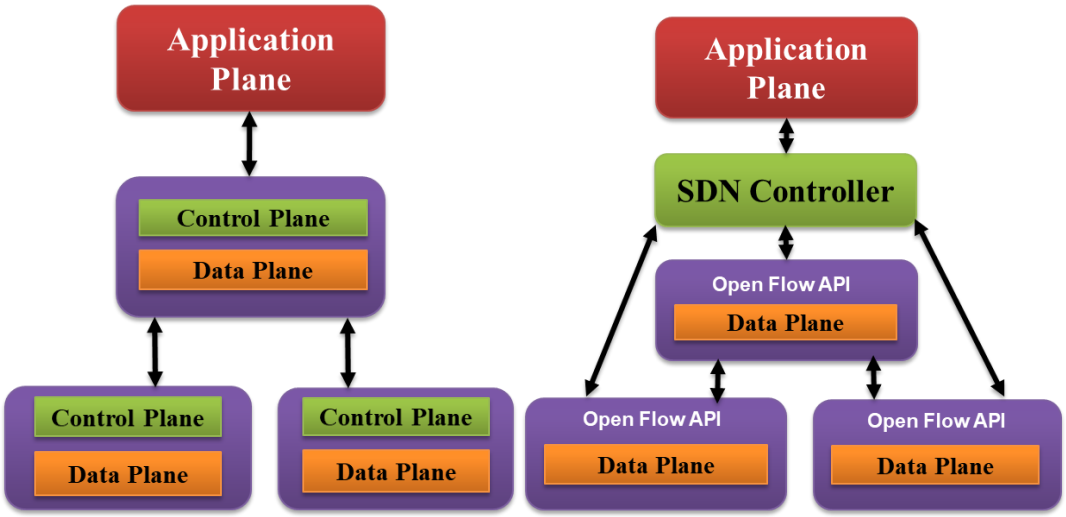


Figure 1.1: Traditional Network vs SDN

Fig. 1.1 illustrates difference between traditional network and SDN. In traditional network control plane and data plane are coupled in forwarding devices. Where as in SDN control plane is decoupled from data plane and implemented in centralized device called controller. Controller have overall view of complete network through open flow API, which allows network administrator to manage and control network. Controller installs forwarding rule in forwarding devices. For example, if a packet arrives at switch from host 1 for destination host 2, if there is no flow rule installed in switch for destination, switch forwards this packet to controller, controller passes this packet through ACL polices and install flow rules in all switches along the path from host 1 to host 2. Next time when packet arrive from host 1 for host 2 switches have already installed flow rules for host 2, switch directs the packet on the specified path as mentioned in installed flow rules. As explained before flow rules are installed in switches according to specified network policies called ACL polices by network administrators. To express these ACL policies network devices need to configure using vendor specific commands. In modern network there is a lot of dynamic changes which need to consider

while writing ACL policies. Implementing ACL policies in such a dynamic network is very challenging [2].

In traditional network control plane (make decisions to handle the traffic) and data plane (forward traffic to destination according to decision made by control plane) are embedding into single network device, which makes the job harder. Because in tradition network control plane is not flexible for innovation and evolution. SDN [4],[5] allows us to address these problems and limitations of traditional networks. It separates the integration of control plane and data plane from the network devices and implement it in controller and data plane is implemented in switches, making switch a simple forwarding device. This separation of control plane and data plane makes policy enforcement and network configuration flexible and manageable [6].

1.2 SDN Architecture

Fig. 1.2. and Fig. 1.3. illustrate architecture of SDN, it contains 3 planes Application plane, Control plane and Data plane. The separation of control plane and data plane is considered as a programming interface using Open flow [7],[8] API as shown in Fig. 2. Open flow switch contains forwarding table which contain flow rules written by controller. These flow rules perform certain task like forwarding or dropping traffic. Switches match the traffic with flow rules installed by controller and make decisions whether packet should be forwarded or dropped. SDN architecture is defined as four pillars

1. Control plane and Data plane are decoupled. Control plane is implemented in controller making network devices simple forwarding devices.

2. Forwarding decision depends on flow rules installed by controller in switches. Flow rule define the behavior of packets from source to destination. All packets of flow have same forwarding behavior at forwarding device. [9],[10]
3. Control logic is moved to external device called controller or NOS.
4. Network is programable through network application running on top of controller in application layer.

Following are the layers of SDN architecture as shown in Fig. 1.2.

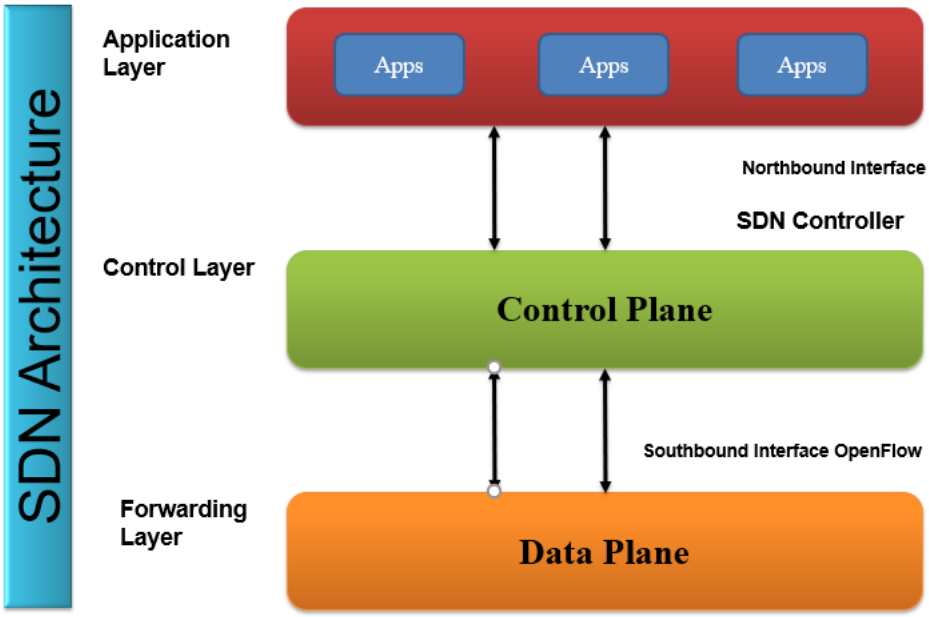


Figure 1.2: SDN Architecture

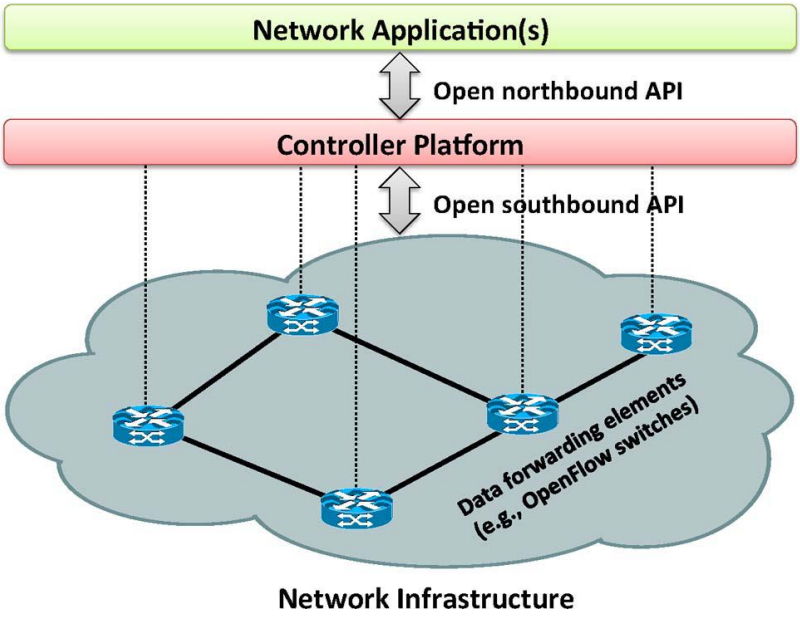


Figure 1.3: SDN Architecture [14]

1.2.1 Data Plane

Data plane encompass the network devices mainly switches. These switches are called forwarding devices in SDN. These are directly managed by the controller in order to install the flow rules. Forwarding devices contain forwarding table in which flow rules are written by the controller. Forwarding devices guides the traffic according to flow rules installed in forwarding table. When a packet is arrived, it matches destination of packet with all the flow rules mentioned in forwarding table, if it matched with flow rule forwarding device forward the packet along the path mentioned in flow rule. If packet does not match with any flow rule, this packet is forwarded to controller. Controller then passes this packet through ACL polices and install flow rules in switches as explained below.

1.2.2 Control Plane:

Control plane in SDN has a signal or multiple software application-based

controllers. In traditional network control plane is integrated with data plane whereas in SDN control plane is separated from data plane. Controller communicates and manage the forwarding devices using the OpenFlow protocol. Main task of controller is to install flow rules in forwarding devices. If switch receives a packet and it does not have flow rule for specific destination it sends packet to controller. Controller have overall view of whole network. Controller computes the path for destination by passing the packet through ACL polices mentioned by network administrator and install flow rule in switches. Switch will use these flow rules for any further communication.

1.2.3 Application/Management Plane:

Applications that interact with controller to make the decision abstractly for user in senesce that what and how efficiently to serve? For example, there are many applications in SDN like firewall, load balancer etc. Application plane is the interface for the administrators to develop applications and customize behavior of network. This entity makes network programable and flexible for developers.

1.2 Problem Statement

Network policies like ACL policies in SDN is implemented in management plane. Using ACL polices controller installs flow rule in forwarding devices (Switches). Forwarding devices uses these flow rules to guide network traffic to the destination. SDN allows multiple network applications to specify ACL polices simultaneously on controller [11], which may lead to conflicts or overlap [11] - [13]. These conflicts and overlap may degrade performance of network quit significantly. Consider an open flow switch having two flow rules

with destination IP 11.1.0.0/16 and 12.1.0.0/16 respectively. If a new flow rules is installed with destination IP 11.0.0.0/8, it may affect the packet belonging to 11.1.0.0/16 range. However newly installed rule will not affect the packets outside the range 11.0.0.0/8 and packets belonging to range 12.1.0.0/16. If first flow rule allow packet for destination 11.1.0.0/16 and second flow rule deny packets for destination 11.0.0.0/8. In this case there is conflict in policy, which cause network wide invariant. Let's take an example from Stanford backbone network traces.

Let P1 and P2 are two ACL polices,

P1 = access-list 151 permit tcp 171.64.24.128 0.0.0.127 any

P2 = access-list 151 deny tcp any 171.64.250.0 0.0.0.128

If packet belong to source IP address range 171.64.24.128-171.64.24.128.127 and destination IP address range 171.64.250.0-171.64.250.128, then this range matches with both polices P1 and P2. Where P1 allows this packet and P2 deny this packet, which result in conflict in these two polices. Let's take another example from Stanford backbone network,

let P1 and P2 are two polices,

access-list 151 permit tcp any 171.64.250.28 0.0.0.1

access-list 151 permit tcp any 171.64.250.0 0.0.0.31

In this example if packet belong to source IP address range 0.0.0.0-255.255.255.255 and destination IP address 171.64.250.28-171.64.250.29, then this packet matches with both polices P1 and P2, where both polices are allowing this packet. In this case there is redundancy in ACL polices, which

takes extra time to pass through it.

Our proposed technique addresses these problems and give a solution which detect conflicts and overlaps in ACL policies and specify overlapped ACL policies in the form such that traversal through ACL polices will be much easier and traversal time will be improved.

1.4 Objectives

Following are the objectives of this research.

1. To propose a technique which detects network wide invariants due to ACL policy conflicts.
2. To propose a method to find overlap ACL policies and represent in form which is easy to traverse.

1.5 Thesis Organization

Thesis organization is as follows:

- Chapter 1: In this chapter Introduction to SDN is explained and brief comparison between traditional IP address and SDN along with the architecture SDN and advantages of SDN over traditional network.
- Chapter 2: This chapter describes literature review.
- Chapter 3: This chapter descries the proposed methodology.
- Chapter 4: In this chapter simulations results are discussed.
- Chapter 5: This chapter concludes all the work done and its future perspectives to extend this work.

1.6 Summary

This chapter covers the main aspects of research topic. A detailed overview about architecture and introduction of SDN is given it also compares traditional network with SDN and describes the need of SDN in modern networks.

Chapter 2

LITERATURE REVIEW

As explained above, the network wide invariants can arise in SDN due to incorrect behavior of different modules at control, data and management planes, and due to different ways of interaction between any of two planes among these three planes. In this section, we explain the existing approaches that detect the network wide invariants produced due incorrect operation SDN modules.

2.1 Veriflow:

Veriflow [14] attempts to detect the network wide invariants (like loops, reachability, black holes, etc.) at real time. Veriflow works as a layer between controller and data plane, as shown in Fig. 2.1. When controller generates the flow rules for an arriving packet, the controller passes these flow rules through Veriflow and ask that whether the flow rules cause a particular network wide invariant or not. If the network wide invariant is created, then Veriflow either generates alarm to inform the network operator and let the flow rules to be installed at data plane or drop the flow rules by avoiding the installation of flow rules at the data plane. Otherwise, Veriflow sends the flow rules to the data plane to be installed. As given in Fig. 2.2, Veriflow has three modules as follows:

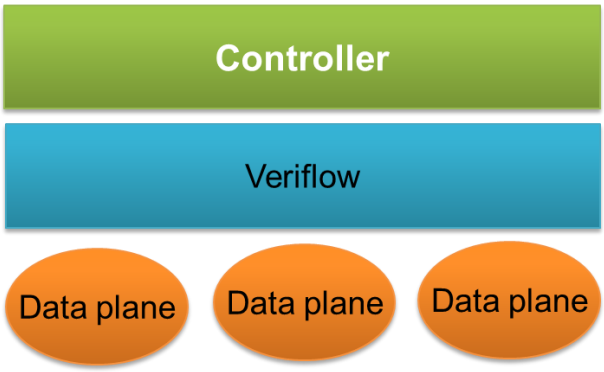


Figure 2.1: Layer of Veriflow between controller and Data Plane

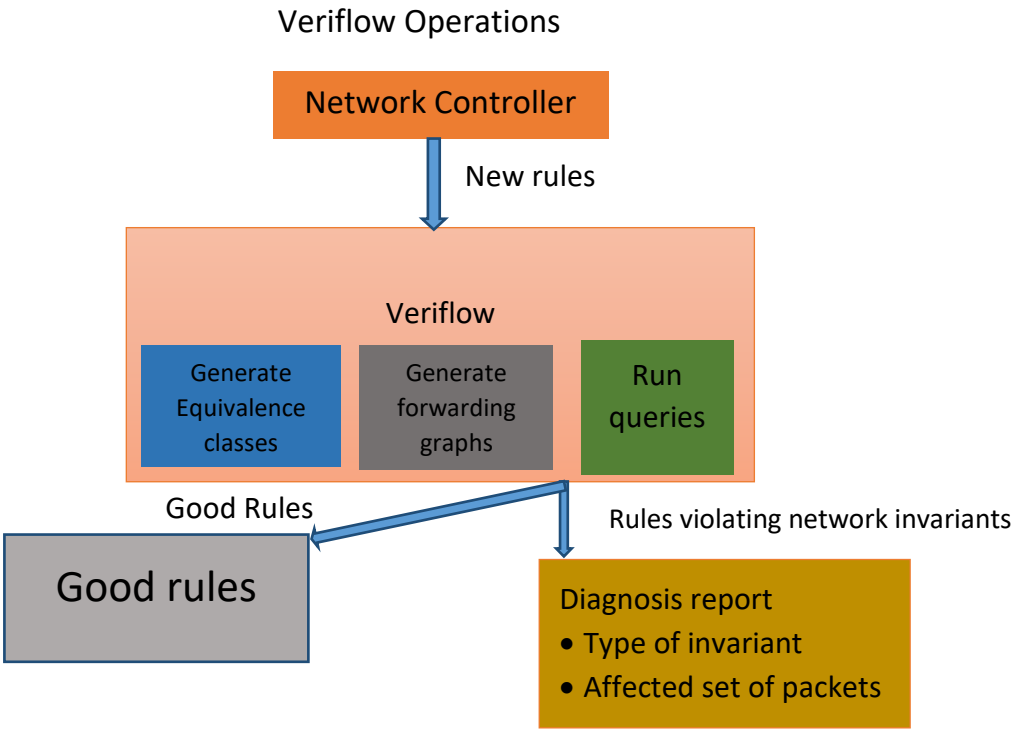


Figure 2.2: Model of Veriflow

2.1.1 Generate equivalence classes:

In this module, Veriflow uses tries [15] structure to store the forwarding rules matched to a set of packets. In hierarchy of tries, each level has a bit equivalent to packet header. Tries from top to bottom of path stores the flow rule's matched to packets header fields. It also stores the information of

device for which packets are located. At time of generation of a new forwarding rule, tries are searched level by level to select specific leaf that contains forwarding rule.

2.1.2 Generate forwarding graph:

A forwarding graph is generated by Veriflow which has equivalent classes as nodes. In the graph, directed edges represent the decision to forward data from equivalence class to device. To generate forwarding graph, it involves two times traversing of tries to get the classes of similar packets and devices. When an equivalence class is modified, Veriflow maintains record of altered class and invariant.

2.1.3 Run queries:

Veriflow checks the flow rules for different types of network wide invariant and the operator has to specify which type(s) of invariant are to be checked. For example, to verify reachability, the reachability verification function takes the directed edge graph and traverse this. This procedure applies depth first methodology to determine whether a packet shall reach to its destination or not. Similarity, to check consistency, Veriflow traverse the forwarding graph between two devices e.g. from R1 to R2, if packets does not reach same destination then Veriflow indicate bugs. However, Veriflow does not automatically localize that the network wide invariant is occurred due to conflict among ACL policies. Moreover, it does not attempt to reduce the number of ACL policies that are overlapping and causing redundant ACL policies. This leads to longer delay at controller to pass the packet through more number of ACL policies.

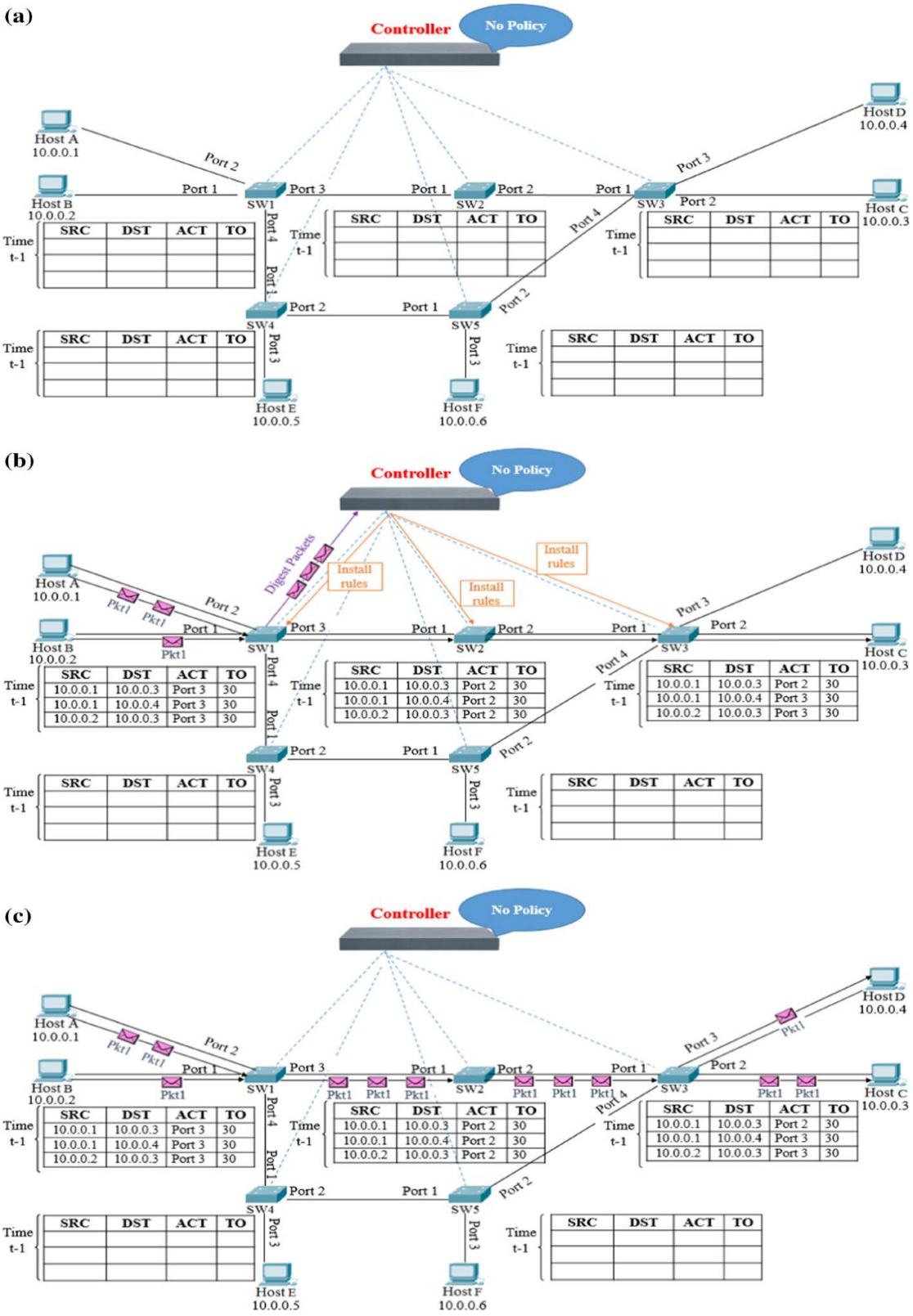
2.2 Network Debugger (NDB):

NDB [16] is a network debugging tool using the software based investigation of SDN applications. NDB use the breakpoints and back tracing the packets when error condition or network wide rule violation occur in network. GNU Debugger [17] and NDB are of similar nature debugger that perform the debugging using the breakpoint and back packet tracing methods for the sequence of events that cause an error. NDB can identify error conditions but it does not identify neither conflict in ACL policies nor overlap in ACL policies.

2.3 Automatic installation of rule in case of policy change:

Mudassar et al. proposed a technique [18] which identify change in ACL policy and install rules according to new policies and overwrite the previous one. Consider a scenario in Fig. 2.3, where host A wants to communicate with host C. Initially there are no flow rules installed in all the switches along the path from host A to host C. When a packet arrives at switch1 from host A for destination host C, there is no flow rule installed in flow table of switch1 for destination host C. Switch 1 will send packet to controller. Controller passes this packet through ACL policy and installs flow rules in all switches along the path from host A to host C. When subsequent packets are sent by host A, packets are successfully forwarded to the destination host C. After some time new policy is introduced at controller, which says host A can communicate host C but packet should not pass through switch 2. In this case there is conflict in flow rules, because switch follow the installed rules in switches. This problem is addressed in this paper and they proposed a method to automatically install flow rule in switches in case of change in policy at controller and overwrite pervious rules. This technique detect change in ACL

policy and install rule accordingly but does not detect conflict in ACL polices. Moreover, it does not detect overlap in ACL polices.



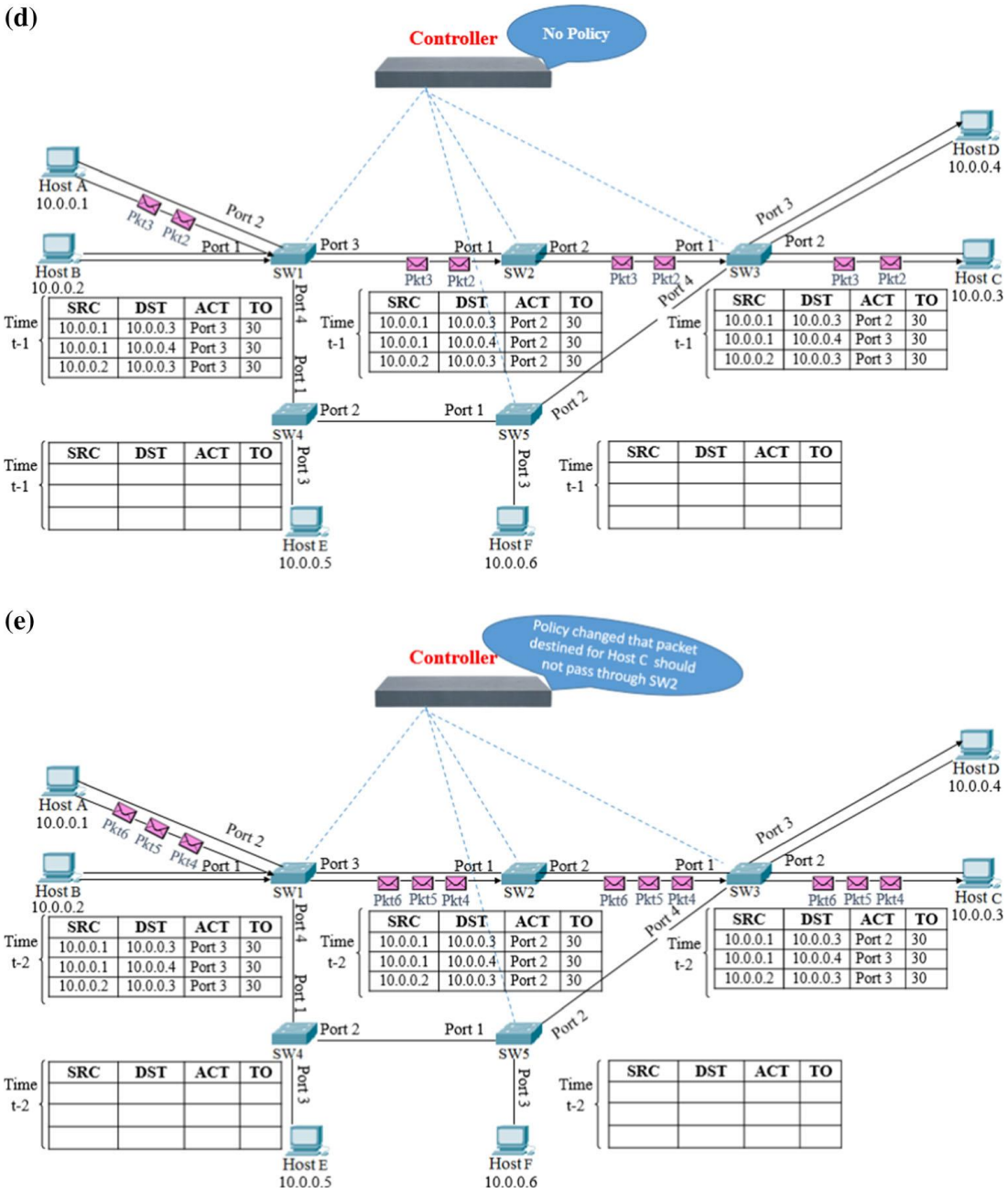


Figure 2.3: a) Shows empty forwarding table. b) flow rule installed by controller along path SW1-Sw2-SW3. c) communication from host A to host C and host D d) communication from host A to host C. e) policy changed but switches flow the previously installed flow rules. [18]

2.4 Header Space Analysis (HSA):

HSA [19] analyze network to detect network wide invariants i.e. loops, black hole and reachability. HSA is a framework which allow network administrator to statically analyze network. HSA addresses the problem to detect loops,

black holes and reachability access between two hosts. It checks whether host, user or traffic is isolated or belong to specific group. For example, is host B is reachable from host A? HSA forms a header space of L bits where each packet is represented as $[0,1]^L$ where L represents header length. HSA model all middle boxes like switches and router as transfer function Ψ as shown in Fig. 2.4, If packet traverse through these transfer functions, it can be traced in geometric space. HSA detects different network wide invariants like reachability analysis, loop detection and black hole. HSA detect loop by injecting a test packet in network, if packet arrives back at injection port then there is loop in network. This process is repeated for every node in network until all nodes are verified. For reachability analysis it generates reachability function R which contains transfer function of all switches along the path from host A to host B. Using this reachability function, host A and host B is analyzed to check whether host B is reachable from host A or not. This technique detects network wide invariants but it does not specify network wide invariants due to ACL policies.

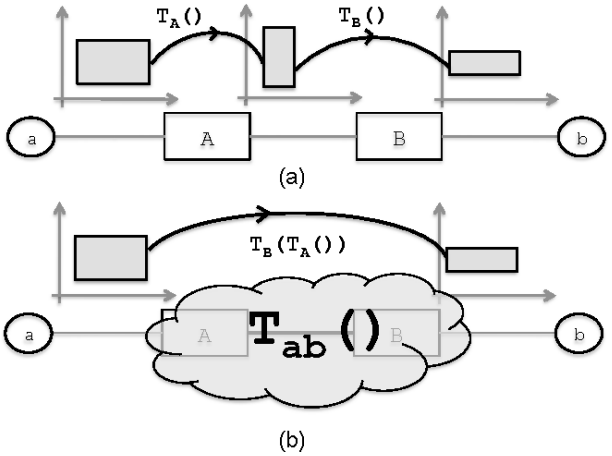


Figure 2.4: HSA Transfer functions for intermediate devices [19]

2.5 PGA: Using Graphs to Express and Automatically Reconcile Network Policies:

PGA [20] automatically compile conflict free policy. PGA is a layer between application plane and controller as shown in Fig. 2.5. PGA architecture consists of two layers Graph Editor and Graph Composer as shown in Fig. 2.6. User/ network administrator define network policies in the form of graph using graph editor layer and these policies are submitted to graph composer layer. Graph composer automatically resolves conflict between policies and generate conflict free graph. For example, an organization wants to implement Customer Relation Management (CRM) for their customers in front office. According to network policy “P1” only marketing employ can communicate CRM server on port 7000 using load balancing service. Whereas there is another policy “P2” which says that employees of organization have restricted access to CRM server through TCP port 80, 334 and 7000 and traffic should go through firewall. There is a need to combine these two policies which should full fill requirements of both policies “P1” and “P2”. In PGA, network administrator create graph based on network policy and submit this graph to PGA graph composer. Graph composer find conflicts among all policies and give suggestions to network administrator and generate conflict free/ error free graph. PGA resolve conflicts but it is at abstract level and it very slow process. It requires days or weeks to complete the whole process. However, PGA avoid conflicts at abstract level but it does not find overlap.

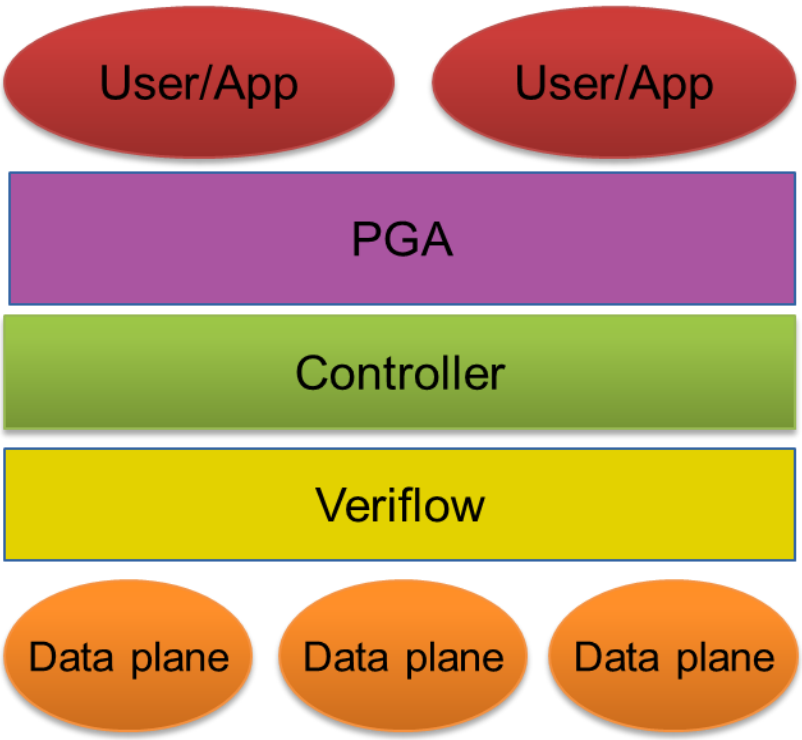


Figure 2.5: Layered model of PGA

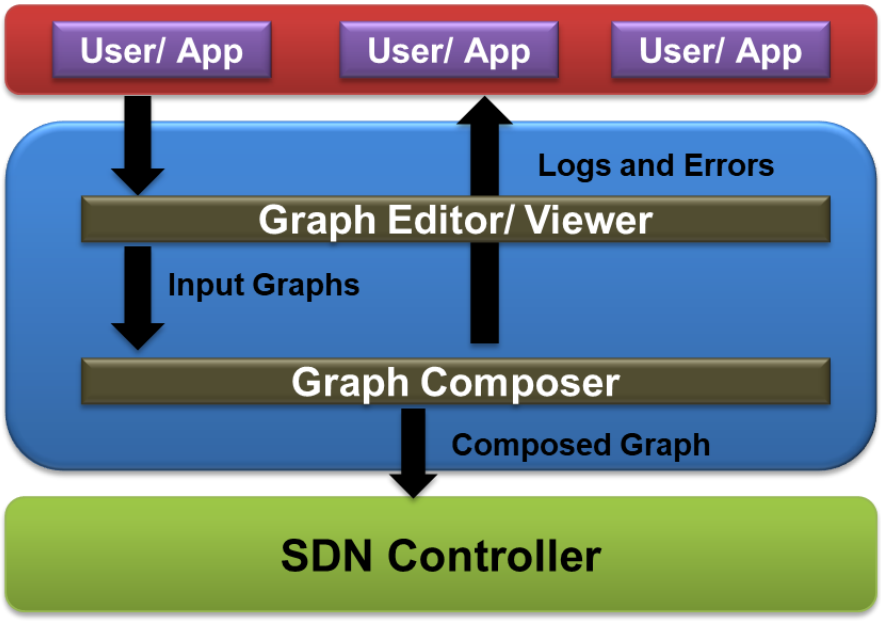


Figure 2.6: PGA Architecture

2.6 No Bugs In Controller Execution (NICE):

NICE [21] is a method to test open flow applications for bugs and network wide invariants in SDN by using model checking and symbolic execution. NICE work flow is shown in Fig 2.7. Consider a scenario in Fig. 2.8, where host A and host B want to communicate through intermediate switches SW1, SW2 and SW3. When packet arrives at SW1, it does not contain flow rules for arrived packet, it sends digest packet to controller. Controller passes this packet through ACL policies and decide whether packet should be permitted or denied. If packet is permitted controller install forwarding rules along the path from host A to host B i.e. SW1, SW2 and SW3. Because of some software error at controller flow rule is not installed or delayed at SW3 by controller. When packet arrives at SW3 it does not have installed flow rules, SW3 send digest packet to controller to request for installation of flow rule. Controller assumes it has already installed flow rule. This is an error in controller where controller is working fine but due to some error flow rules are not installed in forwarding devices. NICE test SDN application by sending stream of packets, covering all possible events to detect these kind of network invariants. NICE detect network invariants like forwarding loops or black holes but it cannot detect network invariants due to conflict in policies.

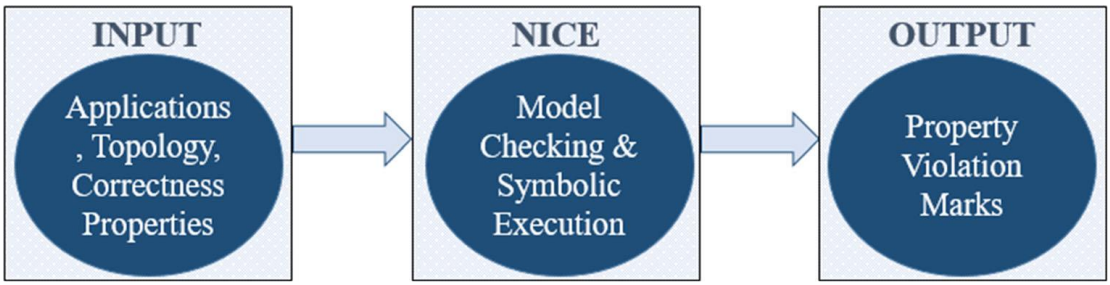


Figure 2.7: Nice working flow.[21]

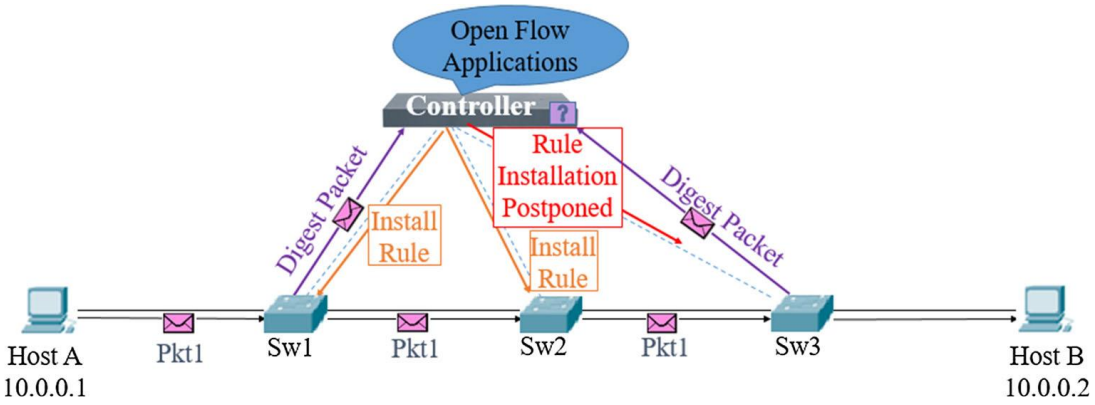


Figure 2.8: Scenario in which rules are not installed by controller [21]

2.7 Dynamic configuration of firewall using algebra of filtering rules:

Vladimir Zaborovsky et al. [22] suggest an approach that constructs an algebra of filtering rules to configure firewall dynamically in a way such that filtering rules are optimized and aggregated to remove redundant filtering rules. This makes easy to both aggregate and control the configuration firewall. The proposed approach consists of five main components: network monitory, ACL policy description module, firewall rules generator, information of resource model and algebra of filtering rules. Network monitor manages the whole system it keeps information of user activity, shared hardware resource and network state. Access policy description module specify ACL polices for firewall configuration like Mr. X should work with YouTube without refinement of nature of Mr. X work [23]. ACL policy is described using notions of subject, object and action. In above example subject is “Mr. X”, object is “YouTube”, action is “read” and decision is “prohibited”. This notion of subject, object and action are not enough to represent real work complex polices. There are some new approaches like RBAC [24] and OrBAC [25] have been developed to represent more complex polices. RBAC uses the notion of role and it replaces the notion of subject. Role may be role of network administrator or a

role of unprivileged user in network. RBAC represent ACL policies using the notion of role. OrBAC extends the traditional RBAC and brings new notion of activity, view and abstract context. Activity replaces the notion of action i.e. read and write and view replaces the notion of object. Using notion of activity, view and abstract context OrBAC represents ACL at abstract level. For example user is prohibited to access entertainment resources. In this abstract rule user is “role”, read is “activity” and entertainment resources is “view”. Firewall rule generator execute ACL policies using main function access control device (ACD). ACD decide whether a subject is permitted to perform an action or not. ACD is configure by the access rules specified by Access policy description module using hardware specific language.

In large distributed network huge number of ACL rules are implemented to restrict unauthorized access. These rules are generated from ACL policies, whereas implementation of these rules may rise errors. Proposed approach describes, test and verify ACL policies, using algebraic technique such that ACL policies are easily implemented in firewall. Author used concept of Ring [26] to define rules for firewall, where in algebra Ring is the set of elements with operators + and *. In simple words we can apply addition and multiplication on elements of Ring to aggregate rules. In this technique elements of ring are firewall rules generated from ACL policies and operator are union and intersection. Elements of ring are represented in space, representing filtering rules and forbidden areas that are incorrect from the point of view of ip network functionality. Using ring operator’s addition and multiplication rules for firewall are optimized and verified.

2.8 Automatic configuration of ACL policy in case of topology change in Hybrid SDN:

Rashid et al. [27] proposed a technique to automatically configure ACL policy in case of topology change in Hybrid SDN. SDN is not widely adopted by organizations due to budget constraint because organizations are reluctant to invest huge budget in SDN to deploy SDN device from scratch. So, organization are moving toward hybrid SDN in which SDN devices are deployed in parallel with traditional networking devices. In real time network addition or removal of links occurs frequently, which changes the topology. Changes in topology may affect network policies and hence it may affect network performance [28] and organization's security. Suppose there is organization's Hybrid SDN network as shown in Fig. 2.9. Company have two front offices site say A and B having front offices AF1-AF2 and BF1-BF2 and two data centers for each front office say AD1-AD2 and BD1-BD2 respectively.

2.8.1 Case1:

Company has ACL policy say p1 which states that front offices AF1-AF2 can communicate with data center AD1-AD2 and front office BF1-BF2 can communicate with data center BD1-BD2 and AF1-AF2 cannot communicate with data center BD1-BD2 similarly front office BF1-BF2 cannot communicate with data center AD1-AD2. This policy is implemented at interface i2.1 and i3.1 of router R2 and R3 respectively. If AF1-AF2 wants to communicate with data center BD1-BD2, packets will be drop at interface at i3.1. If front office BF1-BF2 wants to communicate with data center AD1-AD2, packets will be

drop at interface i2.1 as shown in Fig. 2.9a.

2.8.2 Case 2:

Later on, it decided by network administrator to add new link between router R2 and R3 as shown in Fig. 2.9b. Now front office AF1-AF2 have new link to communicate with data center BD1-BD2 through link R1, R2 and R3. Similarly, front office BF1-BF2 can also communicate with data center AD1-AD2 through link R4, R3 and R2. In this case ACL policy implemented on interface i2.1 and i3.1 is bypassed by new link established between R2 and R3. This problem can be overcome by manual configuration by network administrator, but it is very difficult task to detect these scenarios in large scale network.

2.8.3 Case3:

Suppose, the problem mentioned in case 2 is detected and resolved manually by network administrator. Now policies are implemented on interface i2.2 and i3.2. All packets from front office AF1-AF2 are dropped at interface i2.2 and all packets from front office BF1-BF2 are dropped at interface i3.2. Suppose link between front offices i.e. f1 is down as shown in Fig. 2.9c. In this case front offices cannot communicate directly through link f1 but there is alternative link available for front office AF1-AF2 to communicate with front office BF1-BF2 through link R1, R2, R3 and R4, but still front offices cannot communicate with other because of police implemented on interface i2.2 and i2.3 to drop all packets from front offices BF1-BF2 and AF1-AF2 respectively as shown in Fig. 2.9d.

Author proposed an improve mechanism to automatically configure network

devices if there is change in topology and to use an alternative path available in case of Hybid SDN. This method automatically configures ACL policies but it does not detect conflicts in policies.

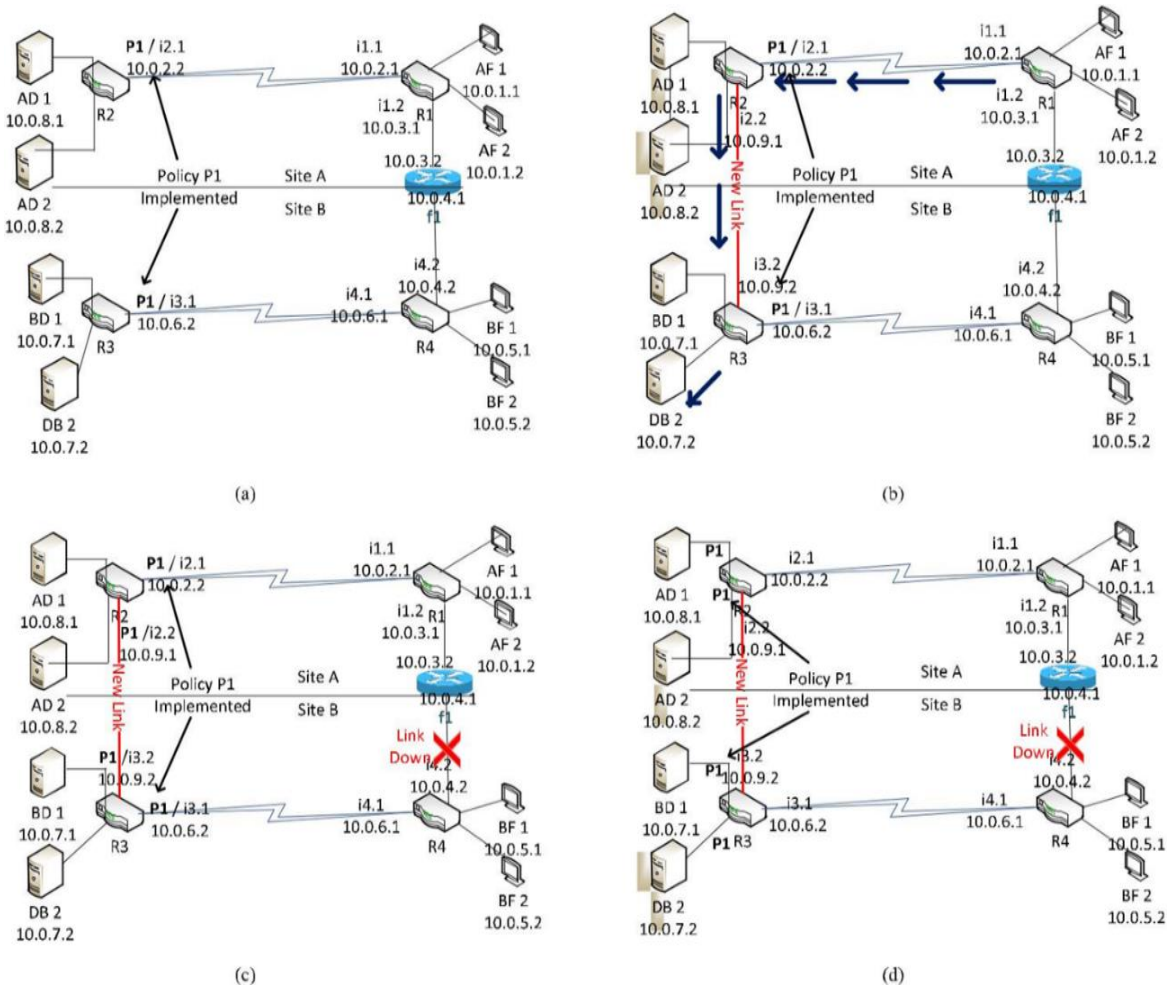


Figure 2.9: Hybrid SDN network for organization for Case 1, case 2, case 3 and case 4.[27]

Techniques discussed so far lack detection of conflicts in ACL polices, detection of network wide invariants due to conflicts in ACL polices and detection of redundant ACL polices. In this thesis a new technique will be proposed to detect network wide invariants due to conflicts in ACL polices and merge the redundant ACL polices.

Chapter 3

SYSTEM MODEL AND PROPOSED SOLUTION

In order to address the problems mentioned in chapter 1, we proposed a method to detect conflicts and overlap in ACL polices and represent ACL policies in such a way which is easy to traverse. Our proposed method will eventually improve network performance. We used the concept of set theory to represent ACL policy in Ring and then we represent these polices in 3D shape. Then we apply separating axis theorem to check if shapes are overlapping. If overlapping polices have different access, its mean overlapping ACL polices have conflicts. If no conflict found then overlapping polices are passed through different cases of merging as explained later in this section, if it matches with any case ACL polices are merged to remove redundant polices. Our proposed method's work flow is shown in Fig. 3.1. In the form of flow chart

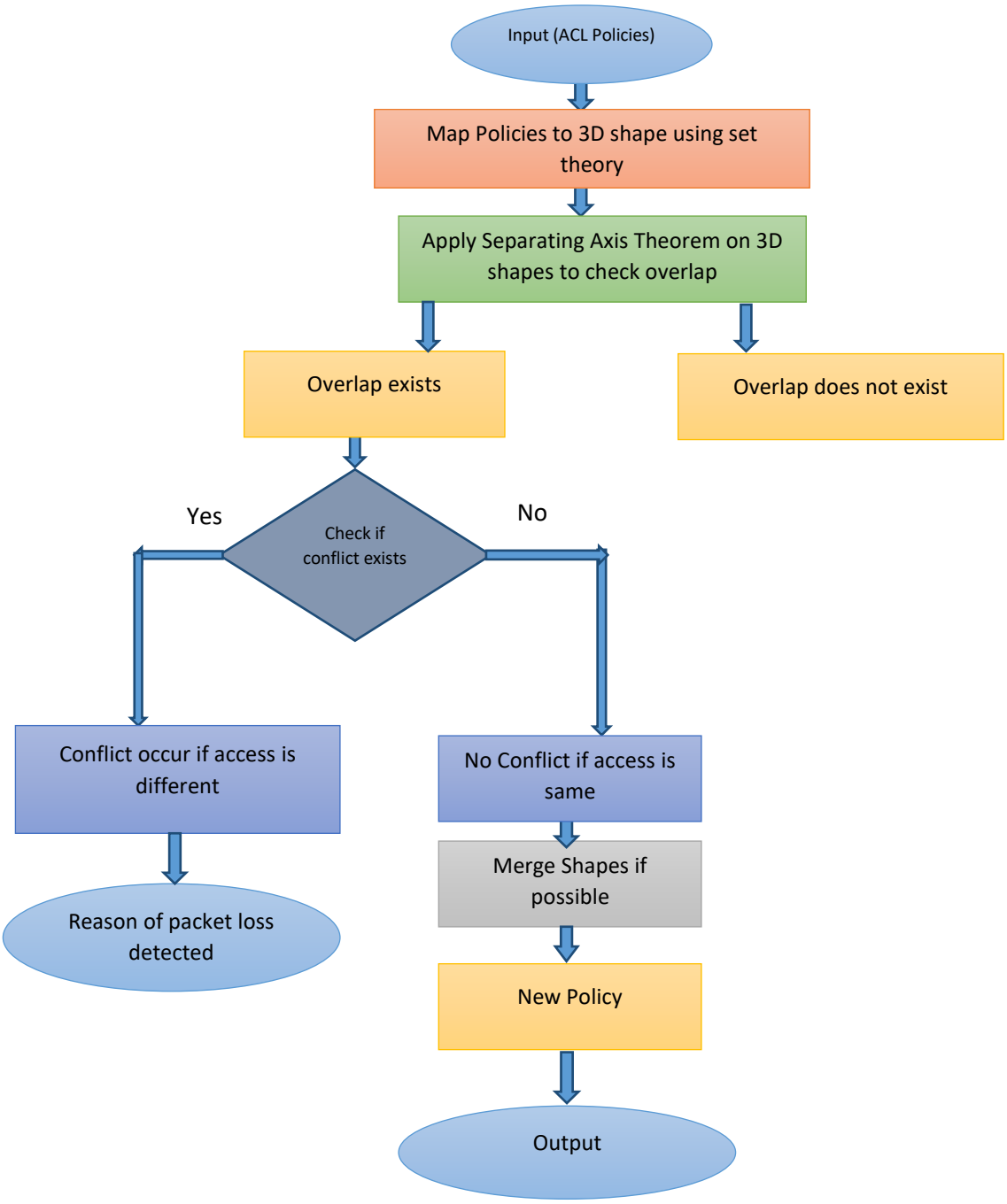


Figure 3.1: Flow chart of our proposed method

3.1 Set Theory:

To represent ACL policy we used concept of algebra of filtering rule R=<R,∑>. Where R is the set of filtering rules, ∑ is the set of possible operation over the set R. Combination of R and ∑ is known as ring [26]. Ring R has following

properties.

3.1.1 Properties:

1. Commutativity of addition: $\forall a,b \in R \;\; a + b = b + a$.

2. Associativity of addition: $\forall a,b,c \in R \;\; a + (b + c) = (a + b) + c$.

3. Zero element of addition: $\forall a \in R \; \exists 0 \in R: \;\; a + 0 = 0 + a = a$.

4. Inverse element of addition: $\forall a \in R \; \exists b \in R: \;\; a + b = b + a = 0$.

5. Associativity of multiplication: $\forall a,b,c \in R \quad a \times (b \times c) = (a \times b) \times c$

6. Distributivity: $\forall a, b, c \in R \; \{(b + c) \times a = b \times a + c \times a\}$.

$$\{a \times (b + c) = a \times b + a \times c\}$$

7. Identity element: $\forall a \in R \; \exists 1 \in R: \quad a \times 1 = 1 \times a = a$.

8. Commutativity of multiplication: $\forall a,b \in R \;\; a \times b = b \times a$.

Set of filtering rule R is defined as

$$R =< R, \sum >$$

$$R = \{r_j, j = 1, |R|\}$$

$$r_j = \{X, \ldots . X_n, Y_1, \ldots . Y_n, P_1, \ldots \ldots P_n, A_j\}$$

$$\sum = \{\Phi_1, \Phi_2\}$$

$$A_j = \{0,1\}$$

Where X represents pool of source IP address of ACL policy, Y represents pool of destination IP address of ACL policy

and A represents access of ACL policy. If access of ACL policy is permit value of A_j=1 otherwise A_j=0. Set ∑ define operations that are possible over filtering rule. Where Φ_1 is operation of addition and Φ_2 is operation of multiplication. Operation of addition is implemented using Union U and operation of multiplication is implemented using intersection.

3.1.2 Example:

Let's take an example of policy P1

P1= access-list 100 permit ip 172.17.0.0 0.0.255.255 172.27.16.32 0.0.0.31

$R =< R, \sum >$

$R = \{r_j, j = 1, |R|\}$

$r_1 = \{X_{a1}, X_{a2}, \dots \dots Y_{aN}, Y_{a1}, Y_{a2}, \dots \dots Y_{aN}, P_{a1}, P_{a2}, \dots \dots P_{aN}, A_j\}$

$r_2 = \{X_{b1}, X_{b2}, \dots \dots Y_{bN}, Y_{b1}, Y_{b2}, \dots \dots Y_{bN}, P_{b1}, P_{b2}, \dots \dots P_{bN}, A_j\}$

$\sum = \{\Phi_1, \Phi_2\}$

$A_j = \{0,1\}$

Pool of source IP address is 172.17.0.0 - 172.17.255.255

Pool of destination IP address is 172.27.16.32 - 172.27.16.63

A=1

$r_1 = \{172.17.0.0, 172.17.255.255, 172.27.16.32, 172.27.16.63, 1\}$

3.2 Representation of Policy in 3D:

ACL policy comprises of source IP, destination IP, port and protocol. We represent ACL policy in 3 dimensions, where source ip is on x-axis,

destination ip is on y-axis and port on z-axis. x-axis consists of pool of source ip address, y-axis consists of pool of destination ip address and z axis consists of range of port as shown in Fig. 3.2.

3.2.1 Example:

access-list 100 permit ip 172.17.0.0 0.0.255.255 172.27.16.32 0.0.0.31 range 5000 5050

x-axis: 172.17.0.0 – 172.17.255.255

y-axis: 172.27.16.32 – 172.27.16.63

z-axis: 5000 to 5050

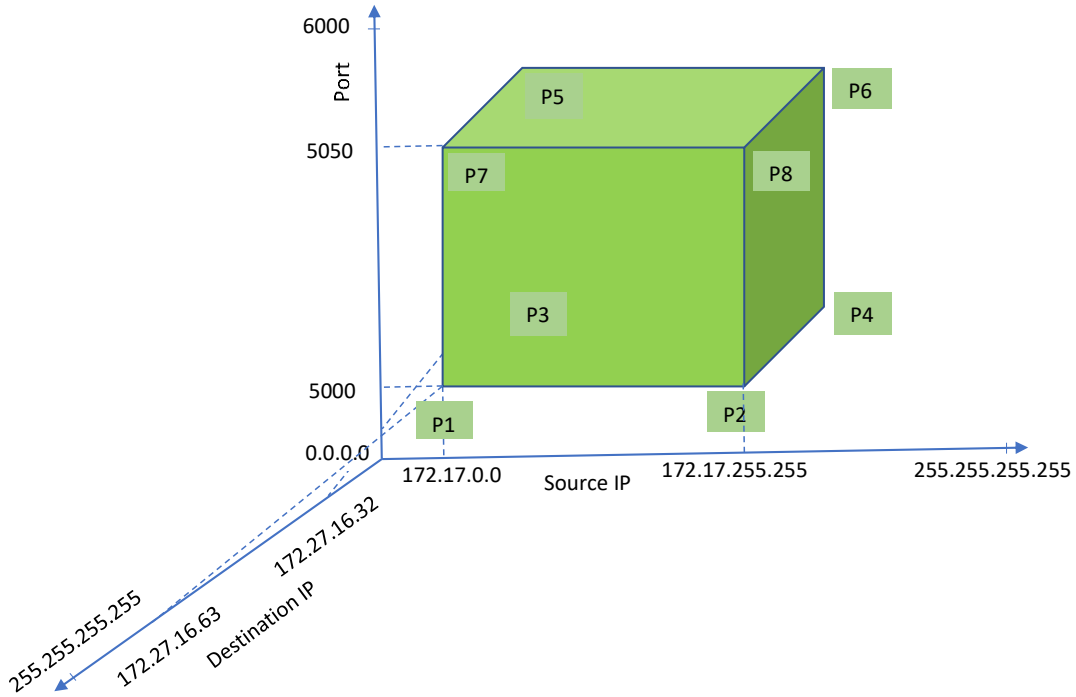


Figure 3.2: 3D Representation of ACL policy

3.3 Separating Axis Theorem:

Separating axis theorem is used to check collision between two polygons. Theorem states if you are able to draw a line between polygons then they are colliding with each other.

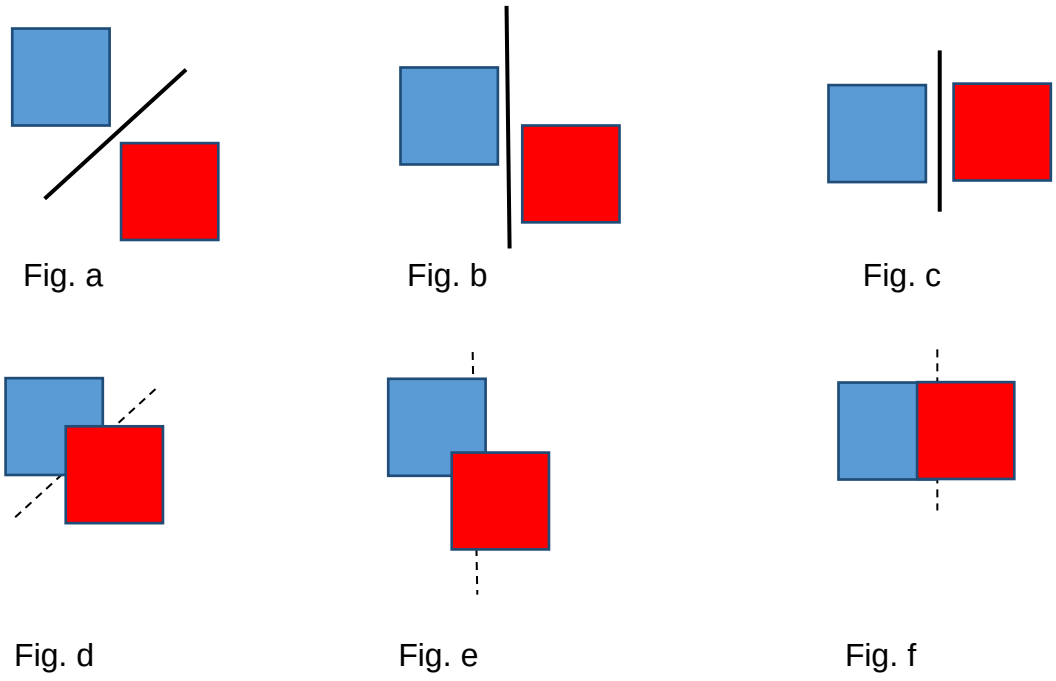


Figure 3.3: Separating axis theorem basic concept

In above Fig. 3.3. we can observe that in Fig. a., Fig. b. and Fig. c. we can draw a line between polygons, hence they are not colliding with each other. Where as in Fig. d., Fig. e., Fig. f., polygons are not colliding as we cannot draw a line between polygons.

Consider two squares box1 and box2 as shown in Fig. 16.

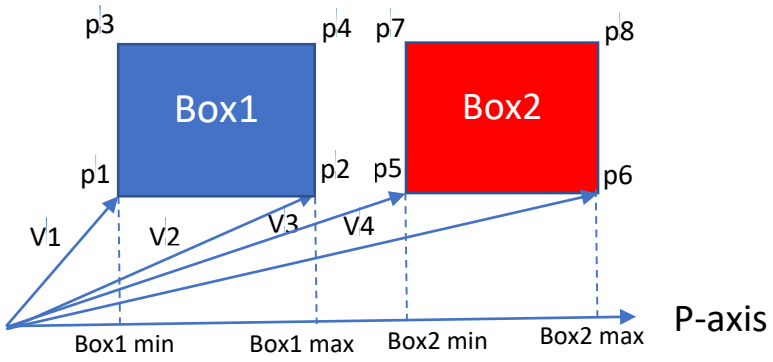


Figure 3.4: Check overlap along P-axis using vector projection method

To find collision between these boxes using separating axis theorem we find distance between two boxes, if distance is zero or less than zero boxes are overlapping/colliding. To find distance we apply concept of vector projection. Let corners of box1 p1, p2, p3 and p4 and corners of box2 p5, p6, p7 and p8 as shown above in Fig. 3.4. Let's take corners p1, p2, p5 and p6 as vectors v1, v2, v3 and v4. To check overlap/collision of boxes along P-axis we calculate projections of these vectors on vector P-axis to calculate 4 values Box1 minimum, Box1 maximum, Box2 minimum and Box2 maximum. If difference between Box2 min and Box1 max is greater than zero or Box2 min is greater than Box1 max, then Boxes are not overlapping with each other along P-axis. Similarly, if difference between Box2 min and Box1 max is equal to zero or less than zero or Box2 min is less than Box1 max, then Boxes are overlapping/colliding with each other along P-axis.

Similarly, we have to check overlap/collision along Q-axis as well.

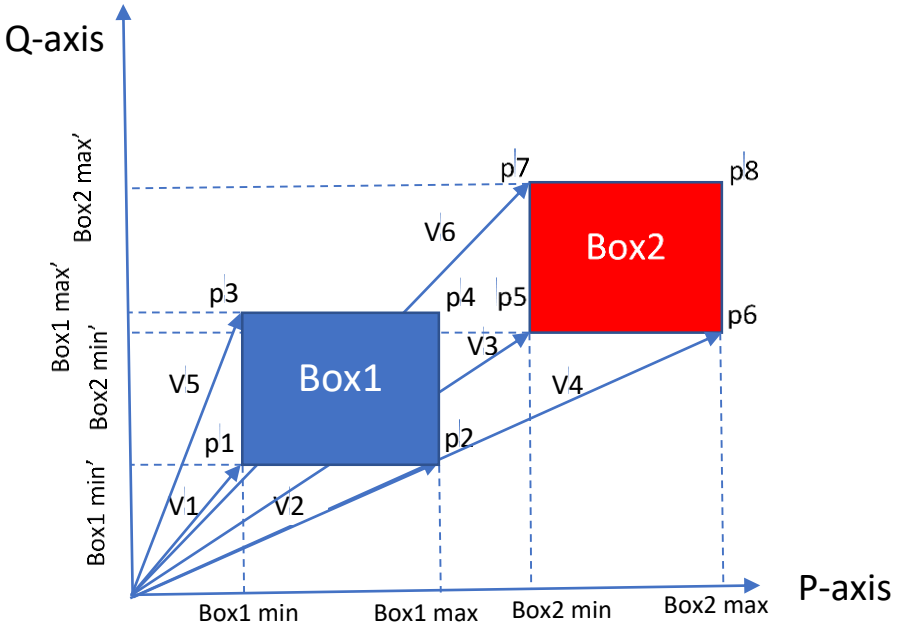


Figure 3.5: Check overlap along P-axis and Q-axis using vector projection method

To check overlap/collision of boxes along Q-axis we calculate projections of vectors along P-axis to calculate 4 values Box1 minimum', Box1 maximum', Box2 minimum' and Box2 maximum' as shown in Fig. 3.5. If difference between Box2 min' and Box1 max' is greater than zero or Box2 min' is greater than Box1 max', then Boxes are not overlapping with each other along Q-axis. Similarly, if difference between Box2 min' and Box1 max' is equal to zero or less than zero or Box2 min' is less than Box1 max', then Boxes are overlapping/colliding with each other along Q-axis.

Similarly, it is required to check overlap/collision along all possible axis. If Boxes are overlapping/colliding along every possible axis then boxes are overlapping/colliding.

3.4 Detect conflict among ACL policy using set theory, 3D representation and separating axis theorem:

To detect conflict between two ACL policies, let's say P1 and P2, we represent these polices in Ring as explained earlier. Then we represent these polices in 3D shapes, let's say S1 and S2, to check overlap/collision between two shapes by applying separating axis theorem as explained earlier. There is conflict in policies if shapes S1 and S2 are overlapping and both polices P1 and P2 contain same protocol but different access i.e. P1's access is permit whereas P2's access is deny.

3.4.1 Example:

Let's consider the following case where we have two policies p1 and p2 as written bellow.

P1 = access-list 151 permit tcp 171.64.24.128 0.0.0.127 any

P2 = access-list 151 deny tcp any 171.64.250.0 0.0.0.128

3.4.1.1 Step1 representation of policy in Ring:

Let's specify elements of Ring R for p1 and p2 are as follow.

$R =< R, \sum >$

$R = \{r_j, j = 1, |R|\}$

$\sum = \{U, \cap\}$

$r_1 = \{X_{a1}, X_{a2}, Y_{a1}, Y_{a2}, A\}$

$r_2 = \{X_{b1}, X_{b2}, Y_{b1}, Y_{b2}, A\}$

For p1,

X_{a1},X_{a2}= pool of source IP address in p1 i.e X_{a1}=171.64.24.128, X_{a2}=171.64.24.255

Y_{a1},Y_{a2}= pool of source IP address in p1 i.e Y_{a1}=0.0.0.0, Y_{a2}=255.255.255.255

A=1

$r_1 = \{171.64.24.128, 171.64.24.255, 0.0.0.0, 255.255.255.255, 1\}$

For p2,

X_{b1},X_{b2}= pool of source IP address in p1 i.e X_{b1}=0.0.0.0, X_{b2}=255.255.255.255

Y_{b1},Y_{b2}= pool of source IP address in p1 i.e Y_{b1}=171.64.250.0, Y_{b2}=171.64.250.128

$A = 1$

$r_2 = \{0.0.0.0, 255.255.255.255, 171.64.250.0, 171.64.250.128, 0\}$

3.4.1.2 Step 2 3D Representation:

We represent polices P1 and P2 in 2D shape because P1 and P2 contains source IP and destination IP only it does not contain ports as shown in Fig. 3.6.

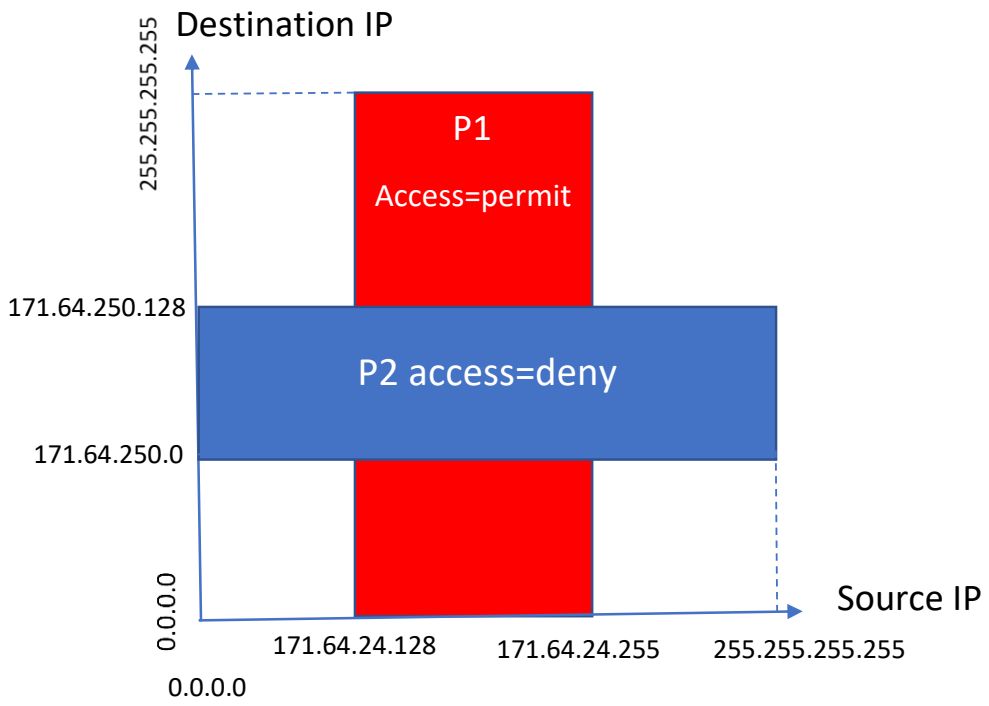


Figure 3.6: 2D representation of policies P1 and P2

.

3.4.1.3 Step 3 applying separating axis theorem to check overlap:

Let Box1 = P1 and Box2= P2

Using concept of vector projection as explained earlier following values are calculated

Box1 min= 171.64.24.128, Box1 max=171.64.24.255, Box2 min= 0.0.0.0, Box2 max= 255.255.255.255

Box1 min'= 0.0.0.0, Box1 max'=255.255.255.255, Box2 min'= 171.64.250.0, Box2 max'=255.255.255.255

3.4.1.3.1 Checking Overlap along x- axis:

$$Box2\ min - Box1\ max = 0.0.0.0 - 171.64.24.255 < 0 \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)$$

Boxes are overlapping along x-axis as distance between Box2 min, Box1 max

is less than zero.

3.4.1.3.2 Checking Overlap along y- axis:

$$Box2\ min' - Box1\ max' = 171.64.250.0 - 255.255.255.255 < 0 \ldots\ldots\ldots\ldots (2)$$

Boxes are overlapping along x-axis as distance between Box2 min', Box1 max' is less than zero.

Eq (1) and (2) show that P1 and P2 are overlapping along source IP and destination IP and P1 and P2 have different access i.e. P1's access is permit and P2's access is deny. If packet arrive at controller with source ip address 171.64.24.128 - 171.64.24.255 and destination IP address 171.64.250.0 – 171.64.250.128 then P1 is allowing this packet and P2 is denying which leads to conflict in ACL policy and it cause network wide invariant.

3.5 Merge policy using set theory and separating axis theorem:

To merge policies firstly we represent all policies in Ring using set theory as explained earlier. Then we represent all polices in 3D shape. Then we pick 1 shape at a time and compare it with rest of the shapes by applying separating axis theorem, if it is overlapping with other shapes and if criteria for merging, explained bellow, is fulfilled then it will be merged with other shapes by applying union operator on Ring. This whole process is continued until all the polices are cross checked with each other.

3.5.1 Conditions to check to Merge policies:

To merge two polices P1 and P2 we represent these polices in in Ring and then in 3D shape let's say S1 and S2. We check if two shapes are overlapping or not by applying separating axis theorem. If polices are

overlapping then we check after merging these shapes may lead to violation of policies as explained below with example.

Let's consider a case where we have two policies p1 and p2 as written bellow.

P1 access-list 108 permit ip 171.66.31.96 0.0.0.7 any

P2 access-list 108 permit ip 171.66.24.0 0.0.7.255 171.64.8.185 0.0.10.255

Let's specify elements of Ring R for p1 and p2 are as follow.

$R =< R, \sum >$

$R = \{r_j, j = 1, |R|\}$

$\sum = \{U, \cap\}$

$r_1 = \{X_{a1}, X_{a2}, Y_{a1}, Y_{a2}, A\}$

$r_2 = \{X_{b1}, X_{b2}, Y_{b1}, Y_{b2}, A\}$

For p1,

X_{a1}, X_{a2} = pool of source IP address in p1 i.e X_{a1} =171.66.31.96,

X_{a2} =171.66.31.103

Y_{b1}, Y_{b2}= pool of source IP address in p1 i.e Y_{b1}=0.0.0.0, Y_{b2}=255.255.255.255

A=1

$r_1 = \{171.66.31.96, 171.66.31.103, 0.0.0.0, 255.255.255.255, 1\}$

For p2,

X_{b1}, X_{b2}= pool of source IP address in p1 i.e X_{b1}=171.66.24.0,

X_{b2}=171.66.31.255

Y_{b1},Y_{b2}= pool of source IP address in p1 i.e Y_{b1}=171.64.8.185, Y_{b2}=171.64.18.255

$A = 1$

$r_2 = \{171.66.24.0, 171.66.31.255, 171.64.8.185, 171.64.18.255, 1\}$

3.5.2 3D Representation:

We represent ACL policy in 3D shape as explained earlier, here we have source IP address, destination IP address and access. Therefore, shape will be 2D as shown below in Fig. 3.7.

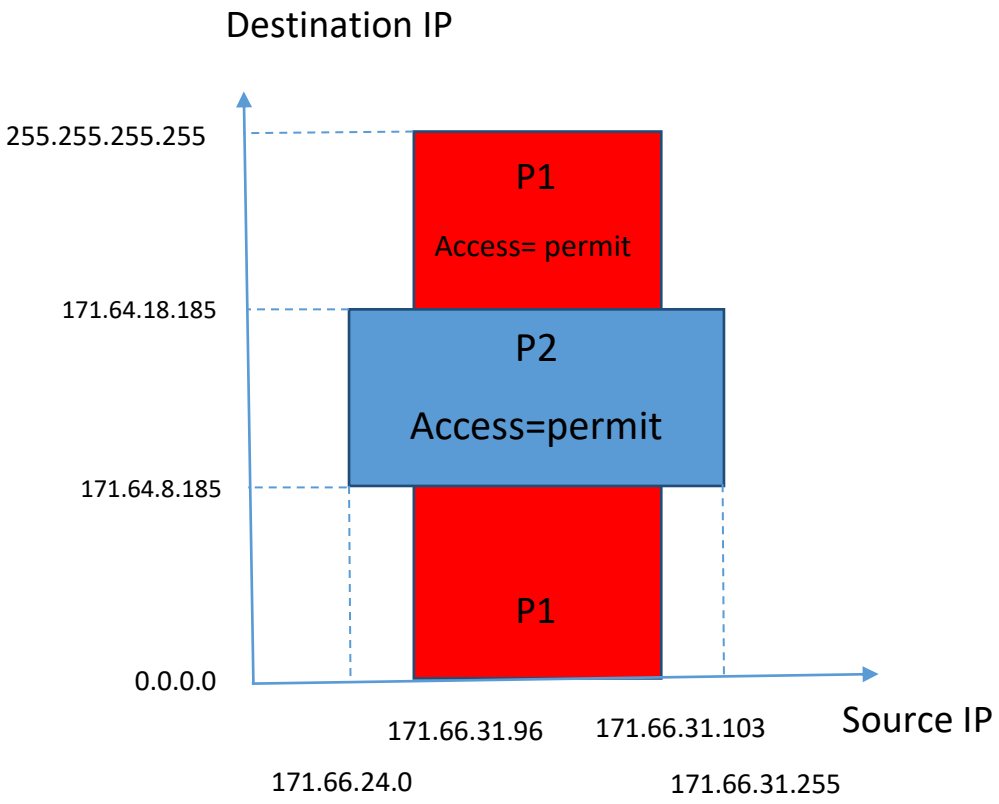


Figure 3.7: 2D representation of policy

3.5.3 Applying separating axis theorem to check overlap:

Let Box1 = P1 and Box2= P2

Using concept of vector projection as explained earlier following values are calculated.

Box1 min= 171.66.31.96, Box1 max=171.66.31.103, Box2 min= 171.66.24.0, Box2 max= 171.66.31.255

Box1 min'= 0.0.0.0, Box1 max'=255.255.255.255, Box2 min'= 171.64.8.185, Box2 max'=171.64.18.185

3.5.3.1 Checking Overlap along x- axis:

$$Box2\ min - Box1\ max = 171.66.24.0 - 171.66.31.103 < 0 \ldots\ldots\ldots\ldots\ldots\ldots (3)$$

Boxes are overlapping along x-axis as distance between Box2 min, Box1 max is less than zero.

3.5.3.2 Checking Overlap along y- axis:

$$Box2\ min' - Box1\ max' = 171.64.8.185 - 255.255.255.255 < 0 \ldots\ldots\ldots (4)$$

Boxes are overlapping along x-axis as distance between Box2 min, Box1 max is less than zero.

Eq (3) and Eq (4) show that P1 and P2 are overlapping along source IP and destination IP and both polices have same access i.e. P1 and P2 both have access permit. P1 and P2 can be merge using union operation on set of P1 and P2 as explained below.

$$r_1 = \{171.66.31.96, 171.66.31.103, 0.0.0.0, 255.255.255.255, 1\}$$

$r_2 = \{171.66.24.0, 171.66.31.255, 171.64.8.185, 171.64.18.255,1\}$

$r_3 = r_1 \; U \; r_2$

$r_3 = \{171.66.24.0, 171.66.31.255, 0.0.0.0, 255.255.255.255, 1\}$

3.5.4 3D representation of merged policy:

Polices P1 and P2 are merged as explained above and represented in 2D shape as show in Fig. 3.8.

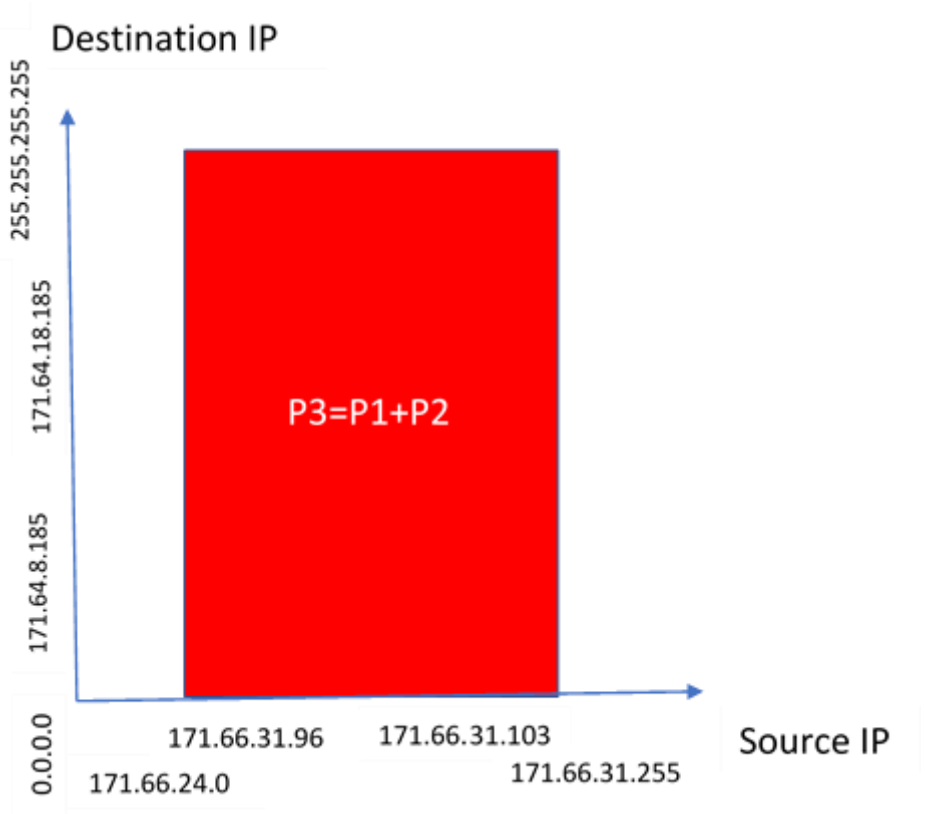


Figure 3.8: Merged policy

3.5.5 Problem after merging:

According to merged policy, if packet belongs to source IP address 171.66.24.0 to 171.66.31.96 and destination IP address 0.0.0.0 to 255.255.255.255, then this packet will be communicated as per merged policy. But according to policy P1, if packet belongs to source IP 171.66.24.0

to 171.66.31.96 and destination IP 171.64.8.185 to 171.64.18.185, it is allowed. But according to merged policy, packet with source IP 171.66.24.0 to 171.66.31.96 is allowed for destination outside range of 171.66.24.0 to 171.66.31.96 which is violation of policy P1. Keeping in mind this scenario we designed 6 different cases in which polices can be merged and there will be no policy violation. These cases are explained below with 3D representation.

3.5.5.1 Case 1:

If two shapes S1 and S2 are equal as shown in Fig. 3.9.

3.5.5.1.1 3D representation of Policy:

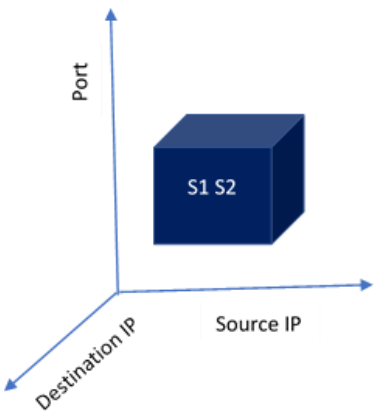


Figure 3.9: Case 1 before merge

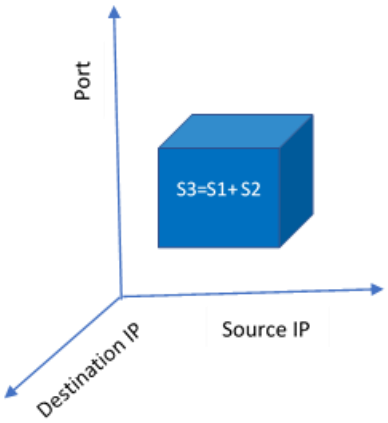


Figure 3.10: Case 1 After merge

3.5.5.1.2 Representation of policy in Ring:

As explained earlier ACL policy is represented in Ring R using set theory. Representation of above two shapes in ring is as follow

$$R =< R, \sum >$$

$$R = \{r_j, j = 1, |R|\}$$

$r_j = \{X_1, \ldots . X_n, Y_1, \ldots . Y_n, P_1, \ldots . . P_n, A_j\}$

$\sum = \{\Phi_1, \Phi_2\}$

$A_j = \{0,1\}$

Policy 1:

$r_1 = \{X_{a1}, X_{a2}, \ldots \ldots Y_{aN}, Y_{a1}, Y_{a2}, \ldots \ldots Y_{aN}, P_{a1}, P_{a2}, \ldots \ldots P_{aN}, A_j\}$

$\sum = \{U, \cap\}$

Policy 2:

$r_2 = \{X_{b1}, X_{b2}, \ldots \ldots Y_{bN}, Y_{b1}, Y_{b2}, \ldots \ldots Y_{bN}, P_{b1}, P_{b2}, \ldots \ldots P_{bN}, A_j\}$

$\sum = \{U, \cap\}$

Merge policies:

Above mentioned shapes represent the duplicate shapes, S2 is exact copy of S1. In this case we simply remove the redundant shape. Resultant shape will be S1 as shown in Fig. 3.10.

3.5.5.2 Case 2:

If S1 and S2 are overlapping along x-axis and S1 is subset of S2 as shown in Fig. 3.11.

3.5.5.2.1 3D Representation:

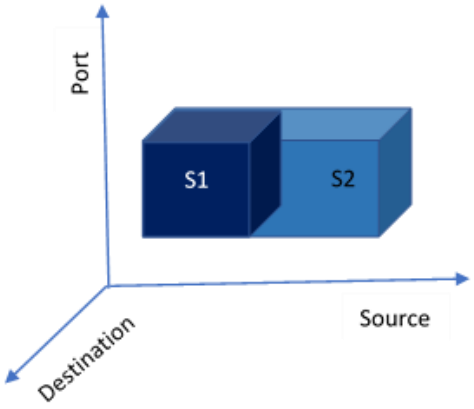


Figure 3.11: Case 2 before merge

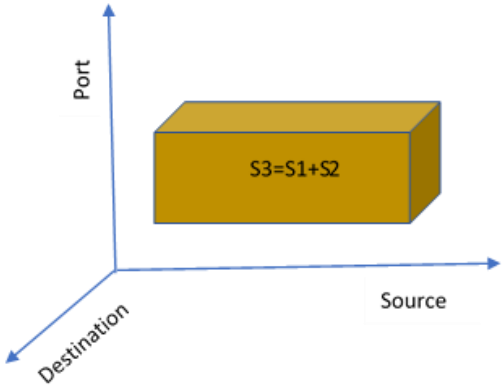


Figure 3.12: Case 2 After merge

3.5.5.2.1 Representation of policy in Ring:

As explained earlier ACL policy is represented in Ring R using set theory. Representation of above two shapes in ring is as follow

$R =< R, \sum >$

$R = \{r_j, j = 1, |R|\}$

$r_j = \{X_1, \ldots . X_n, Y_1, \ldots . Y_n, P_1, \ldots . . P_n, A_j\}$

$\sum = \{\Phi_1, \Phi_2\}$

$A_j = \{0,1\}$

Policy 1:

$r_1 = \{X_{a1}, X_{a2}, \ldots \ldots Y_{aN}, Y_{a1}, Y_{a2}, \ldots \ldots Y_{aN}, P_{a1}, P_{a2}, \ldots \ldots P_{aN}, A_j\}$

$\sum = \{U, \cap\}$

Policy 2:

$r_2 = \{X_{b1}, X_{b2}, \ldots \ldots Y_{bN}, Y_{b1}, Y_{b2}, \ldots \ldots Y_{bN}, P_{b1}, P_{b2}, \ldots \ldots P_{bN}, A_j\}$

$\sum = \{U, \cap\}$

Merge policies:

In this case S1 is subset of S2. These shapes can be merged if both shapes have same access, as mentioned by variable A_j in ring. To merge these shapes, we apply union operator as described below.

$r_1 = \{ X_{a1}, X_{a2}, \dots \dots Y_{aN}, Y_{a1}, Y_{a2}, \dots \dots Y_{aN}, P_{a1}, P_{a2}, \dots \dots P_{aN}, A_j\}$

$r_2 = \{X_{b1}, X_{b2}, \dots \dots Y_{bN}, Y_{b1}, Y_{b2}, \dots \dots Y_{bN}, P_{b1}, P_{b2}, \dots \dots P_{bN}, A_j\}$

$r_3 = r_1 \; U \; r_2$

$r_3 = \{ X_{a1}, X_{a2}, \dots \dots Y_{aN}, Y_{a1}, Y_{a2}, \dots \dots Y_{aN}, P_{a1}, P_{a2}, \dots \dots P_{aN}, A_j\}$ U

$\{X_{b1}, X_{b2}, \dots \dots Y_{bN}, Y_{b1}, Y_{b2}, \dots \dots Y_{bN}, P_{b1}, P_{b2}, \dots \dots P_{bN}, A_j\}$

$r_3 = \{X_{b1}, X_{b2}, \dots \dots Y_{bN}, Y_{b1}, Y_{b2}, \dots \dots Y_{bN}, P_{b1}, P_{b2}, \dots \dots P_{bN}, A_j\}$

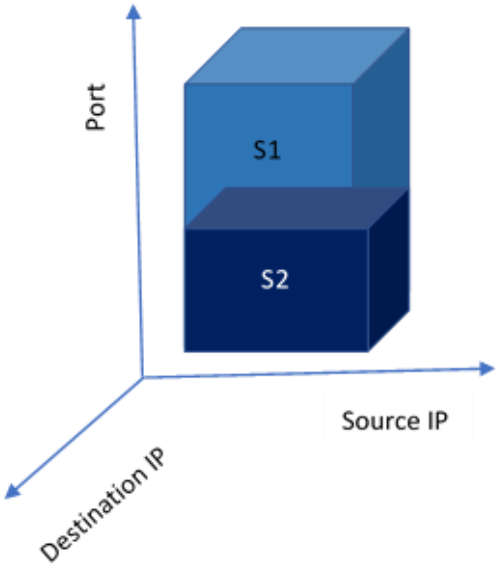
$r_3 = r_2$

In this case merged shape is equal to S2 as shown in Fig. 3.12.

3.5.5.3 Case 3:

Shapes S1 and S2 are overlapping along y-axis as shown in Fig. 3.13.

3.5.5.3.1 3D Representation:



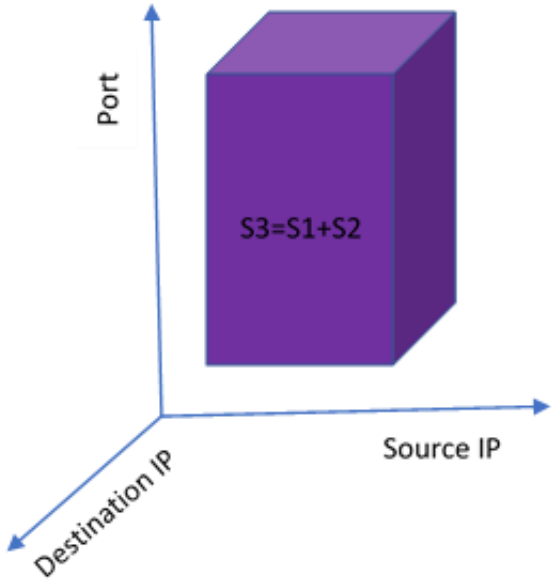


Figure 3.13: Case 3 before merge

Figure 3.14: Case 3 After merge

3.5.5.3.2 Representation of policy in Ring:

As explained earlier ACL policy is represented in Ring R using set theory. Representation of above two shapes in ring is as follow

$R =< R, \sum >$

$R = \{r_j, j = 1, |R|\}$

$r_j = \{X_1, \ldots . X_n, Y_1, \ldots . Y_n, P_1, \ldots . . P_n, A_j\}$

$\sum = \{\Phi_1, \Phi_2\}$

$A_j = \{0,1\}$

Policy 1:

$r_1 = \{X_{a1}, X_{a2}, \ldots \ldots Y_{aN}, Y_{a1}, Y_{a2}, \ldots \ldots Y_{aN}, P_{a1}, P_{a2}, \ldots \ldots P_{aN}, A_j\}$

$\sum = \{U, \cap\}$

Policy 2:

$r_2 = \{X_{b1}, X_{b2}, \ldots \ldots Y_{bN}, Y_{b1}, Y_{b2}, \ldots \ldots Y_{bN}, P_{b1}, P_{b2}, \ldots \ldots P_{bN}, A_j\}$

$\sum = \{U, \cap\}$

Merge policies:

In this case S1 is subset of S2. These shapes can be merged if both shapes have same access, as mentioned by variable A_j in ring. To merge these shapes, we apply union operator as described below.

$r_1 = \{X_{a1}, X_{a2}, \ldots \ldots Y_{aN}, Y_{a1}, Y_{a2}, \ldots \ldots Y_{aN}, P_{a1}, P_{a2}, \ldots \ldots P_{aN}, A_j\}$

$r_2 = \{X_{b1}, X_{b2}, \ldots \ldots Y_{bN}, Y_{b1}, Y_{b2}, \ldots \ldots Y_{bN}, P_{b1}, P_{b2}, \ldots \ldots P_{bN}, A_j\}$

$r_3 = r_1 \; U \; r_2$

$r_3 = \{X_{a1}, X_{a2}, \ldots \ldots Y_{aN}, Y_{a1}, Y_{a2}, \ldots \ldots Y_{aN}, P_{a1}, P_{a2}, \ldots \ldots P_{aN}, A_j\}$ U

$\{X_{b1}, X_{b2}, \ldots \ldots Y_{bN}, Y_{b1}, Y_{b2}, \ldots \ldots Y_{bN}, P_{b1}, P_{b2}, \ldots \ldots P_{bN}, A_j\}$

$r_3 = \{X_{a1}, X_{a2}, \ldots \ldots Y_{aN}, Y_{a1}, Y_{a2}, \ldots \ldots Y_{aN}, P_{a1}, P_{a2}, \ldots \ldots P_{aN}, A_j\}$

$r_3 = r_1$

In this case merged shape is equal to S1 as shown in Fig. 3.14.

3.5.5.4 Case 4:

Shapes S1 and S2 are overlapping along x-axis as shown in Fig. 3.15.

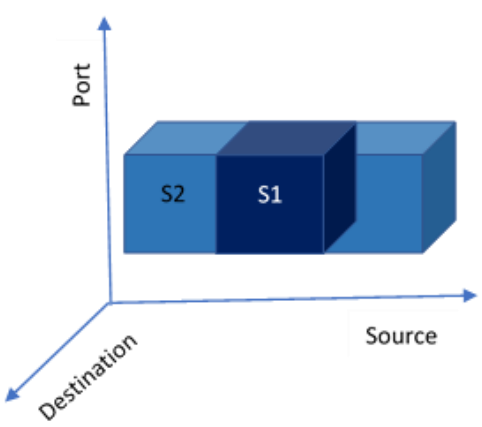


Figure 3.15: Case 4 before merge

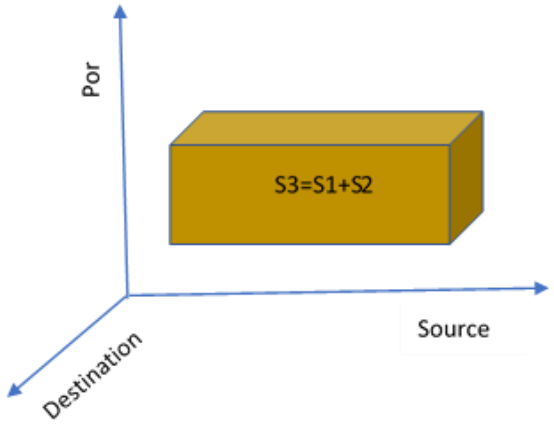


Figure 3.16: Case 4 After merge

3.5.5.4.1 Representation of policy in Ring:

As explained earlier ACL policy is represented in Ring R using set theory. Representation of above two shapes in ring is as follow

$R =< R, \sum >$

$R = \{r_j, j = 1, |R|\}$

$r_j = \{X_1, \ldots . X_n, Y_1, \ldots . Y_n, P_1, \ldots . . P_n, A_j\}$

$\sum = \{\Phi_1, \Phi_2\}$

$A_j = \{0,1\}$

Policy 1:

$r_1 = \{X_{a1}, X_{a2}, \ldots \ldots Y_{aN}, Y_{a1}, Y_{a2}, \ldots \ldots Y_{aN}, P_{a1}, P_{a2}, \ldots \ldots P_{aN}, A_j\}$

$\sum = \{U, \cap\}$

Policy 2:

$r_2 = \{X_{b1}, X_{b2}, \ldots \ldots Y_{bN}, Y_{b1}, Y_{b2}, \ldots \ldots Y_{bN}, P_{b1}, P_{b2}, \ldots \ldots P_{bN}, A_j\}$

$\sum = \{U, \cap\}$

Merge policies:

In this case S1 is subset of S2. These shapes can be merged if both shapes have same access, as mentioned by variable A_j in ring. To merge these shapes, we apply union operator as described below.

$r_1 = \{ X_{a1}, X_{a2}, \ldots \ldots Y_{aN}, Y_{a1}, Y_{a2}, \ldots \ldots Y_{aN}, P_{a1}, P_{a2}, \ldots \ldots P_{aN}, A_j\}$

$r_2 = \{X_{b1}, X_{b2}, \ldots \ldots Y_{bN}, Y_{b1}, Y_{b2}, \ldots \ldots Y_{bN}, P_{b1}, P_{b2}, \ldots \ldots P_{bN}, A_j\}$

$r_3 = r_1 \; U \; r_2$

$r_3 = \{ X_{a1}, X_{a2}, \ldots \ldots Y_{aN}, Y_{a1}, Y_{a2}, \ldots \ldots Y_{aN}, P_{a1}, P_{a2}, \ldots \ldots P_{aN}, A_j\}$ U

$\{X_{b1}, X_{b2}, \ldots \ldots Y_{bN}, Y_{b1}, Y_{b2}, \ldots \ldots Y_{bN}, P_{b1}, P_{b2}, \ldots \ldots P_{bN}, A_j\}$

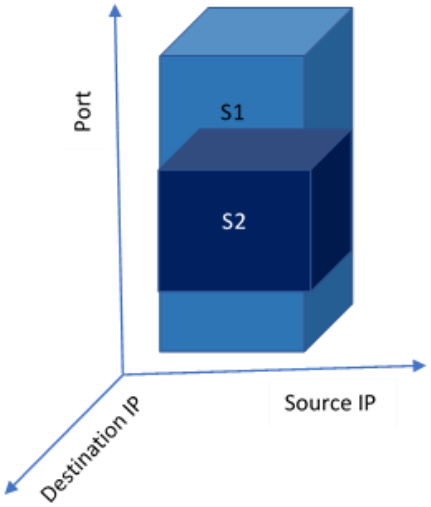
$r_3 = \{X_{b1}, X_{b2}, \ldots \ldots Y_{bN}, Y_{b1}, Y_{b2}, \ldots \ldots Y_{bN}, P_{b1}, P_{b2}, \ldots \ldots P_{bN}, A_j\}$

$r_3 = r_2$

In this case merged shape is equal to S2 as shown in Fig. 3.16.

3.5.5.5 Case 5:

Shapes S1 and S2 are overlapping along y-axis as shown in Fig. 3.17.



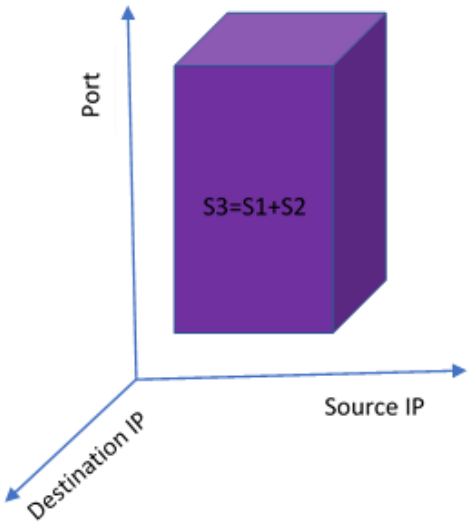


Figure 3.17 : Case 5 before merge

Figure 3.18: Case 5 After merge

3.5.5.5.1 Representation of policy in Ring:

As explained earlier ACL policy is represented in Ring R using set theory. Representation of above two shapes in ring is as follow

$R =< R, \sum >$

$R = \{r_j, j = 1, |R|\}$

$r_j = \{X_1, \ldots . X_n, Y_1, \ldots . Y_n, P_1, \ldots . . P_n, A_j\}$

$\sum = \{\Phi_1, \Phi_2\}$

$A_j = \{0,1\}$

Policy 1:

$r_1 = \{X_{a1}, X_{a2}, \ldots \ldots Y_{aN}, Y_{a1}, Y_{a2}, \ldots \ldots Y_{aN}, P_{a1}, P_{a2}, \ldots \ldots P_{aN}, A_j\}$

$\sum = \{U, \cap\}$

Policy 2:

$r_2 = \{X_{b1}, X_{b2}, \ldots\ldots Y_{bN}, Y_{b1}, Y_{b2}, \ldots\ldots Y_{bN}, P_{b1}, P_{b2}, \ldots\ldots P_{bN}, A_j\}$

$\sum = \{U, \cap\}$

Merge policies:

In this case S1 is subset of S2. These shapes can be merged if both shapes have same access, as mentioned by variable A_j in ring. To merge these shapes, we apply union operator as described below.

$r_1 = \{X_{a1}, X_{a2}, \ldots\ldots Y_{aN}, Y_{a1}, Y_{a2}, \ldots\ldots Y_{aN}, P_{a1}, P_{a2}, \ldots\ldots P_{aN}, A_j\}$

$r_2 = \{X_{b1}, X_{b2}, \ldots\ldots Y_{bN}, Y_{b1}, Y_{b2}, \ldots\ldots Y_{bN}, P_{b1}, P_{b2}, \ldots\ldots P_{bN}, A_j\}$

$r_3 = r_1 \; U \; r_2$

$r_3 = \{X_{a1}, X_{a2}, \ldots\ldots Y_{aN}, Y_{a1}, Y_{a2}, \ldots\ldots Y_{aN}, P_{a1}, P_{a2}, \ldots\ldots P_{aN}, A_j\}$ U

$\{X_{b1}, X_{b2}, \ldots\ldots Y_{bN}, Y_{b1}, Y_{b2}, \ldots\ldots Y_{bN}, P_{b1}, P_{b2}, \ldots\ldots P_{bN}, A_j\}$

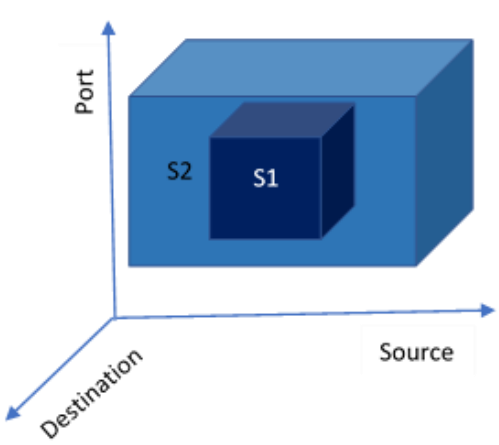
$r_3 = \{X_{a1}, X_{a2}, \ldots\ldots Y_{aN}, Y_{a1}, Y_{a2}, \ldots\ldots Y_{aN}, P_{a1}, P_{a2}, \ldots\ldots P_{aN}, A_j\}$

$r_3 = r_1$

In this case merged shape is equal to S1 as shown in Fig. 3.18.

3.5.5.6 Case 6:

Shapes S1 is subset of S2 as show in Fig. 3.19.



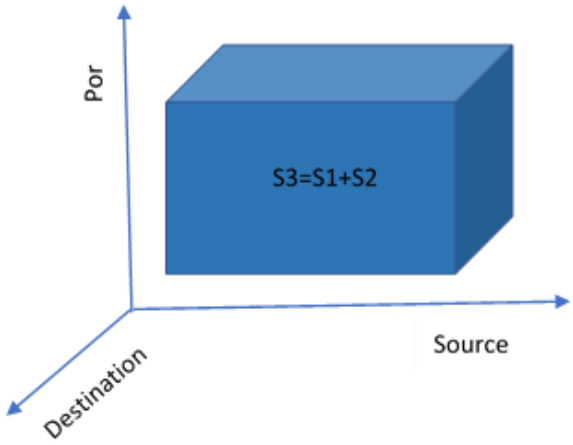


Figure 3.19: Case 6 before merge

Figure 3.20: Case 6 After merge

3.5.5.6.1 Representation of policy in Ring:

As explained earlier ACL policy is represented in Ring R using set theory. Representation of above two shapes in ring is as follow

$R =< R, \sum >$

$R = \{r_j, j = 1, |R|\}$

$r_j = \{X_1, \ldots . X_n, Y_1, \ldots . Y_n, P_1, \ldots . . P_n, A_j\}$

$\sum = \{\Phi_1, \Phi_2\}$

$A_j = \{0,1\}$

Policy 1:

$r_1 = \{X_{a1}, X_{a2}, \ldots \ldots Y_{aN}, Y_{a1}, Y_{a2}, \ldots \ldots Y_{aN}, P_{a1}, P_{a2}, \ldots \ldots P_{aN}, A_j\}$

$\sum = \{U, \cap\}$

Policy 2:

$r_2 = \{X_{b1}, X_{b2}, \ldots \ldots Y_{bN}, Y_{b1}, Y_{b2}, \ldots \ldots Y_{bN}, P_{b1}, P_{b2}, \ldots \ldots P_{bN}, A_j\}$

$\sum = \{U, \cap\}$

Merge policies:

In this case S1 is subset of S2. These shapes can be merged if both shapes have same access, as mentioned by variable A_j in ring. To merge these shapes, we apply union operator as described below.

$r_1 = \{X_{a1}, X_{a2}, \ldots \ldots Y_{aN}, Y_{a1}, Y_{a2}, \ldots \ldots Y_{aN}, P_{a1}, P_{a2}, \ldots \ldots P_{aN}, A_j\}$

$r_2 = \{X_{b1}, X_{b2}, \ldots \ldots Y_{bN}, Y_{b1}, Y_{b2}, \ldots \ldots Y_{bN}, P_{b1}, P_{b2}, \ldots \ldots P_{bN}, A_j\}$

$r_3 = r_1 \; U \; r_2$

$r_3 = \{X_{a1}, X_{a2}, \ldots \ldots Y_{aN}, Y_{a1}, Y_{a2}, \ldots \ldots Y_{aN}, P_{a1}, P_{a2}, \ldots \ldots P_{aN}, A_j\}$ U

$\{X_{b1}, X_{b2}, \ldots \ldots Y_{bN}, Y_{b1}, Y_{b2}, \ldots \ldots Y_{bN}, P_{b1}, P_{b2}, \ldots \ldots P_{bN}, A_j\}$

$r_3 = \{X_{b1}, X_{b2}, \ldots \ldots Y_{bN}, Y_{b1}, Y_{b2}, \ldots \ldots Y_{bN}, P_{b1}, P_{b2}, \ldots \ldots P_{bN}, A_j\}$

$r_3 = r_2$

In this case merged shape is equal to S2 as shown in Fig. 3.20.

3.5.5.7 Merge policies Example:

Let’s consider another case where we have two policies p1 and p2 as written below.

P1 access-list 151 permit tcp any 171.64.250.28 0.0.0.1

P2 access-list 151 permit tcp any 171.64.250.0 0.0.0.31

Let's specify elements of Ring R for p1 and p2 are as follow.

$R =< R, \sum >$

$R = \{r_j, j = 1, |R|\}$

$\sum = \{U, \cap\}$

$r_1 = \{X_{a1}, X_{a2}, Y_{a1}, Y_{a2}, A\}$

$r_2 = \{X_{b1}, X_{b2}, Y_{b1}, Y_{b2}, A\}$

For p1,

X_{a1},X_{a2}= pool of source IP address in p1 i.e X_{a1}=0.0.0.0,

X_{a2} =255.255.255.255

Y_{a1},Y_{a2}= pool of source IP address in p1 i.e Y_{a1}=171.64.250.0, Y_{a2}=171.64.250.31

A=1

$r_1 = \{0.0.0.0, 255.255.255.255, 171.64.250.0, 171.64.250.31,1\}$

For p2,

X_{b1},X_{b2}= pool of source IP address in p1 i.e. X_{b1}=0.0.0.0, X_{b2}=255.255.255.255

Y_{b1},Y_{b2}= pool of source IP address in p1 i.e. Y_{b1}=171.64.250.28, Y_{b2}=171.64.250.29

A=1

$r_2 = \{0.0.0.0, 255.255.255.255, 171.64.250.28, 171.64.250.29,1\}$

3.5.5.7.1 3D Representation:

Polices P1 and P2 are represented in 2D shape to apply separating axis theorem to check overlap as shown in Fig. 3.21.

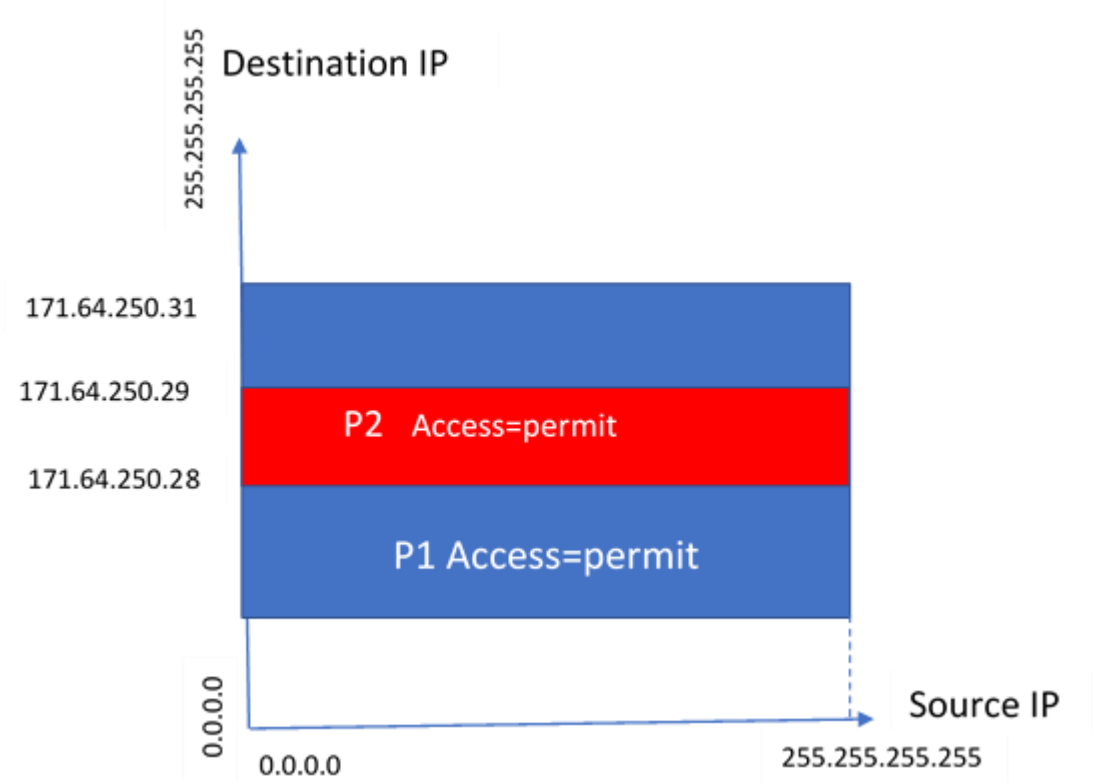


Figure 3.21: 3D representation of polices to apply separating axis theorem

3.5.5.7.2 Check overlap using separating axis theorem:

Let Box1 = P1 and Box2 = P2

Applying vector projection method to calculate following values to apply separating axis theorem

Box1 min= 0.0.0.0, Box1 max=255.255.255.255, Box2 min= 0.0.0.0, Box2 max= 255.255.255.255

Box1 min'= 171.64.250.0, Box1 max'=171.64.250.31,

Box2 min'= 171.64.250.28,

Box2 max'=171.64.250.29

Checking Overlap along x- axis:

$$Box2\ min - Box1\ max = 0.0.0.0 - 255.255.255.255 < 0 \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (5)$$

Since the difference is less than zero it shows shapes are overlapping along x-axis;

Checking Overlap along y- axis:

$$Box2\ min' - Box1\ max' = 171.64.250.28 - 171.64.250.31 < 0 \ldots\ldots\ldots\ldots\ldots . (6)$$

Since the difference is less than zero it shows shapes are overlapping along y-axis;

Eq (5) and (6) show P1 and P2 are overlapping along both i.e. axis source IP and destination IP and it matches with condition of case 5. Polices can be merged using union operator as explained below.

$$r_1 = \{0.0.0.0, 255.255.255.255, 171.64.250.0, 171.64.250.31,1\}$$

$$r_2 = \{0.0.0.0, 255.255.255.255, 171.64.250.28, 171.64.250.29,1\}$$

$$r_3 = r_1\ U\ r_2 = \{0.0.0.0, 255.255.255.255, 171.64.250.0, 171.64.250.31,1\}\ U$$

$$\{0.0.0.0, 255.255.255.255, 171.64.250.28, 171.64.250.29,1\}$$

$$r_3 = \{0.0.0.0, 255.255.255.255, 171.64.250.0, 171.64.250.31,1\}$$

Merged policy is shown in Fig. 3.22.

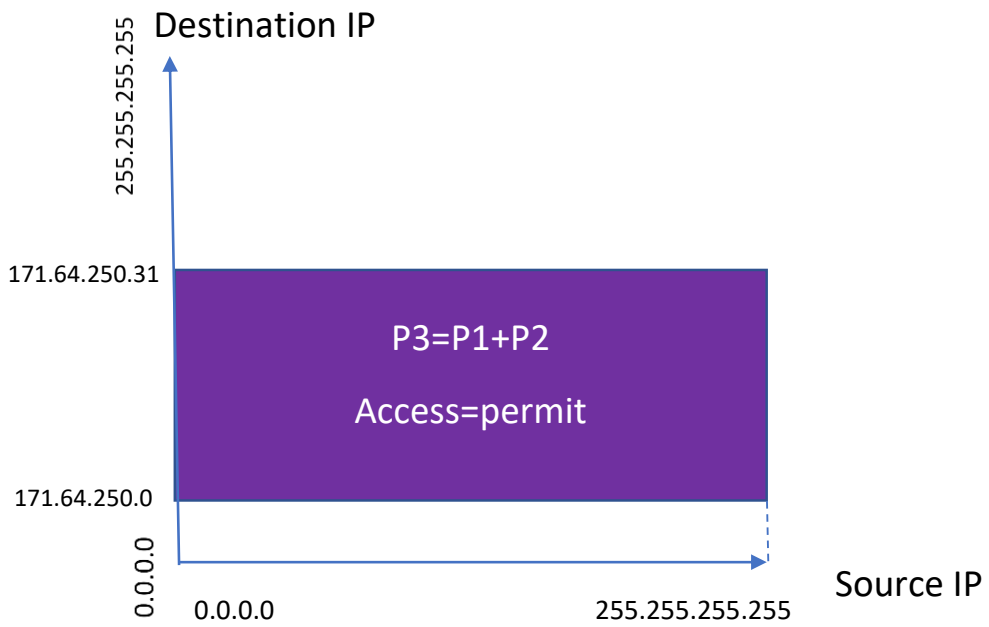


Figure 3.22: Merge polices using separating axis theorem vector projection method

Chapter 4

DISCUSSION AND RESULTS

In this chapter results have been discussed of proposed method. Network performance is discussed with and without proposed method. It has been deduced from our result that our method improved network performance.

4.1 Merge policies using set theory and separating axis theorem:

As explained earlier we used Stanford's backbone ACL polices to test our method. We used 3 different set of ACL polices let's name it as Stanford1, Stanford2 and Stanford3. Stanford1 contains 203 ACL polices, Stanford2 contains 43 ACL polices and Stanford3 contains 53 ACL polices as shown in Fig. 4.1.

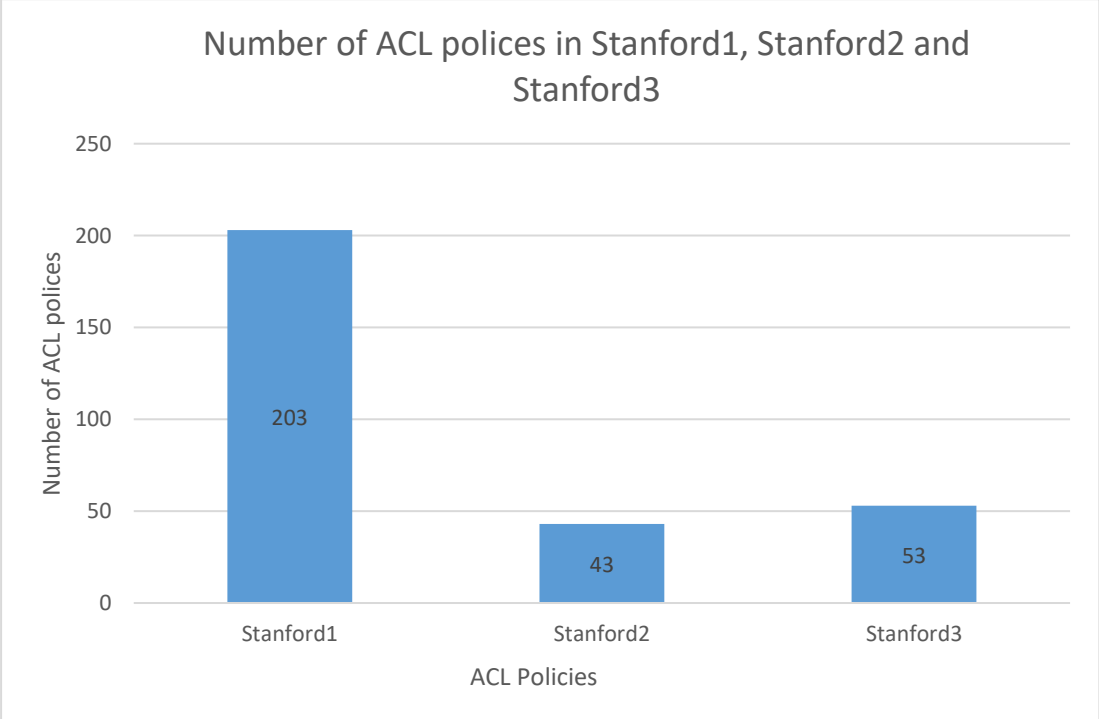


Figure 4.1: Number of ACL polices in Stanford1, Stanford2 and Stanford3

Fig. 4.2. shows reduced number of ACL polices as discussed earlier we apply set theory and separating axis theorem to merge polices. Stanford1 is reduced to 144, Stanford2 is reduced to 24 and Stanford3 is reduced to 18 ACL polices.

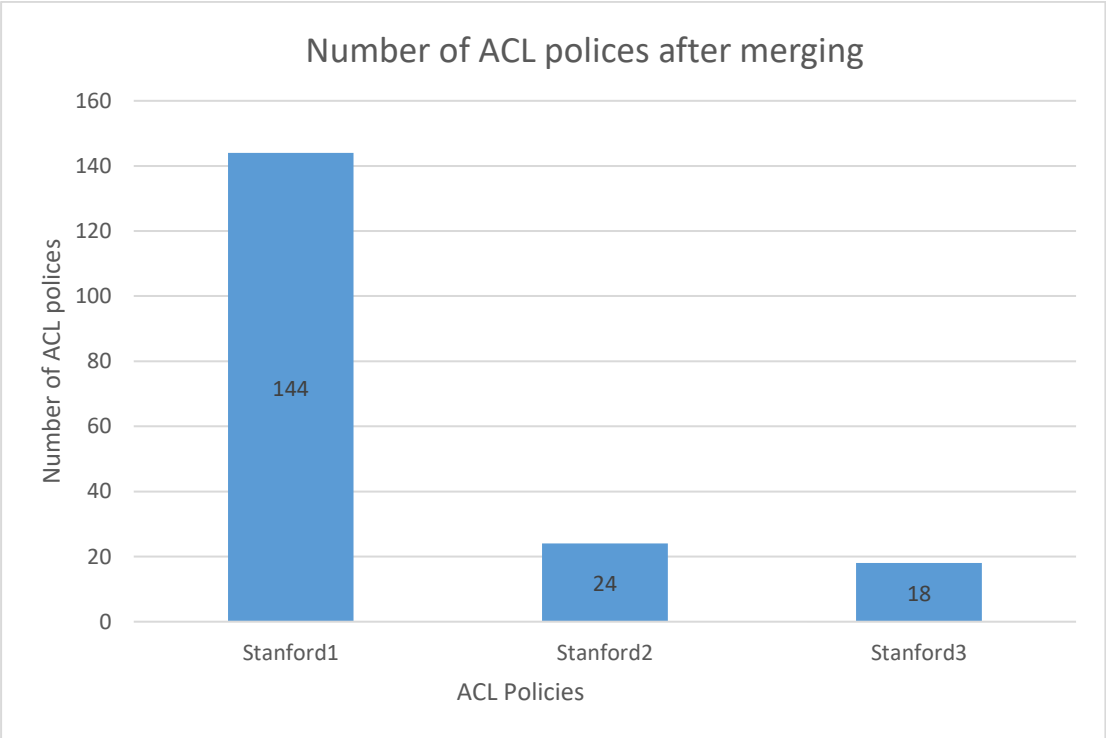


Figure 4.2: Merged ACL polices

Fig. 4.3 shows time taken in milliseconds by our method to merge ACL polices of Stanford1, Stanford2 and Stanford3. It takes 210 ms to merge 59 ACL polices of Stanford1, 56 ms to merge 29 ACL polices of Stanford2 and 25 ms to merge 25 ACL polices. Number of ACL polices merged of Stanford1, Stanford2 and Stanford3 are shown in Fig. 4.4.

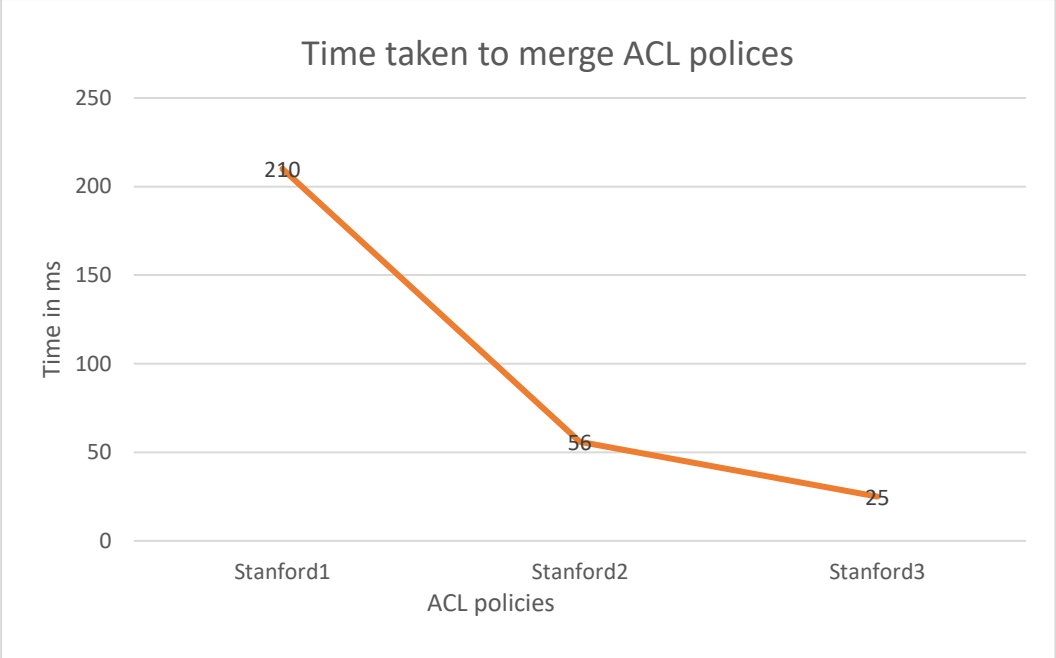


Figure 4.3: Time complexity to merge ACL polices of Standord1, Stanford2 and Stanford3

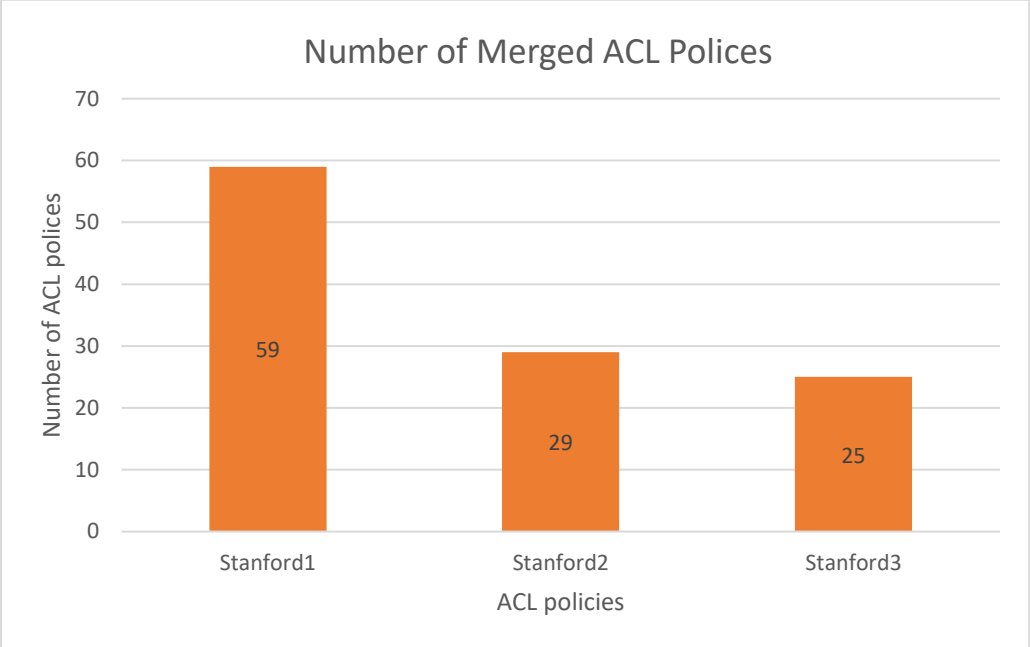


Figure 4.4: Number of ACL polices merged in Stanford1, Stanford2 and Stanford3

4.1.2 Performance Evaluation:

Fig. 4.5. shows comparison between traversal time of ACL polices before merging and after merging. It shows traversal time is improved after merging the ACL polices. Before merging it takes 900 ms to traverse Stanford1, 445 ms to traverse Stanford2 and 450 ms to traverse Stanford3. As polices are merged, number of ACL polices are reduced which takes less time to traverse as shown in Fig. 4.5. After merging it takes 150 ms to traverse Stanford1 and 95 ms to traverse Stanford2 and 110 ms to traverse Stanford3.

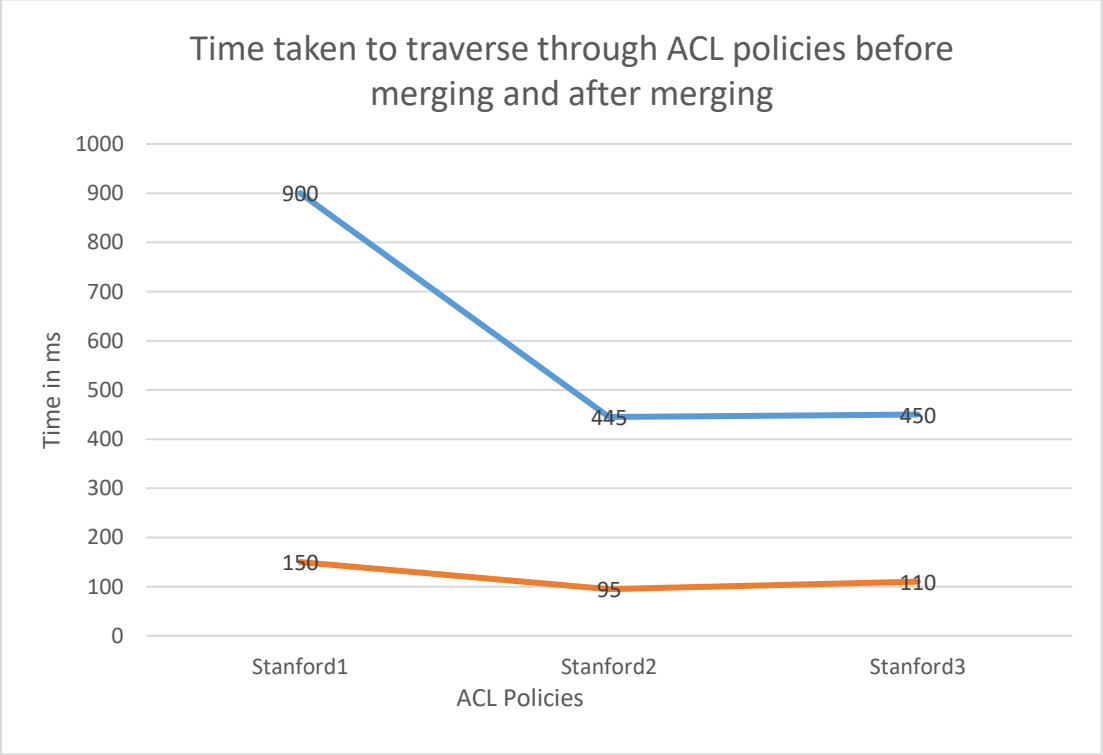


Figure 4.5: Time taken to traverse through ACL polices before and after merging

4.2 Detection of Conflicts

Detection of conflicts in ACL polices is explained earlier in chapter 3. In this section time taken by our method to detect conflicts is discussed and graph of total conflicts found in Stanford's backbone ACL polices. Fig. 4.6. presents 2

conflicts are found in Stanford1, whereas 0 conflicts found in Stanford2 and Stanford3. Fig. 4.7. presents time taken to find conflicts in Stanford1, Stanford2 and Stanford3.

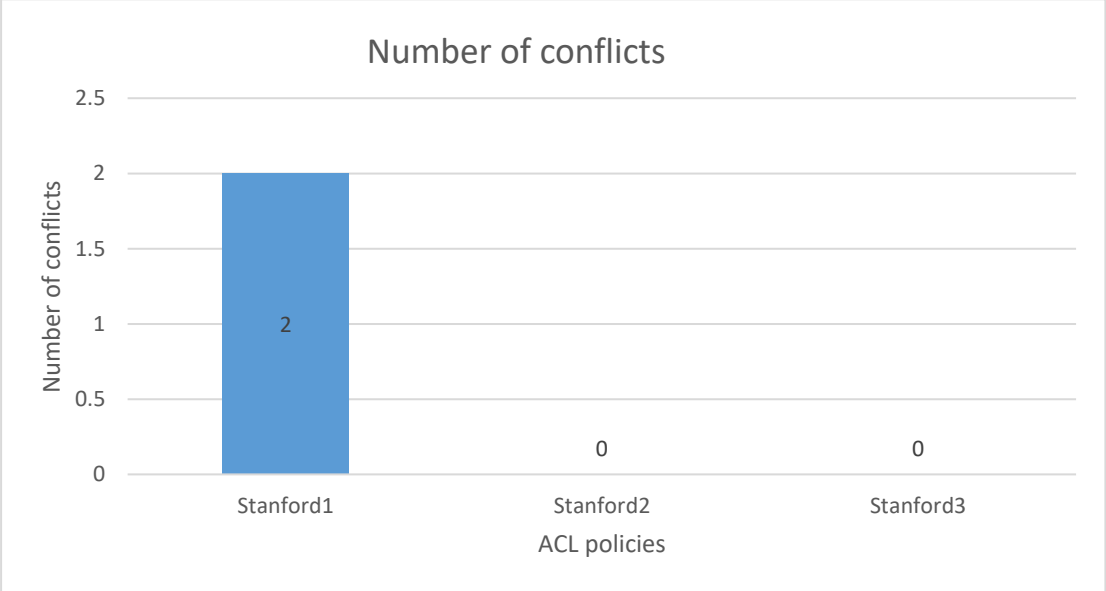


Figure 4.6: Number of conflicts in Stanford1, Stanford2 and Stanford3

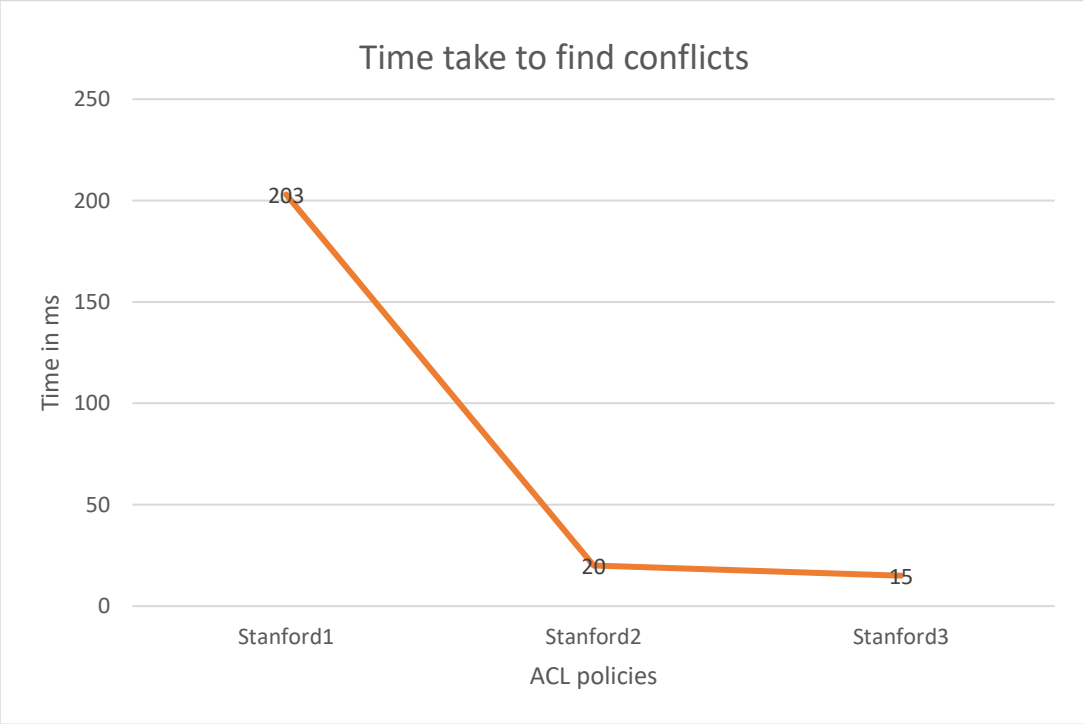


Figure 4.7: Time taken to find conflicts

Chapter 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

In chapter 4 we discussed comparison of traversal through ACL polices with and without our proposed method. It has been discussed that our method detects conflicts among ACL polices and find redundant ACL polices which can be merged with other polices. These polices have been merged to reduced number of ACL polices. As number of ACL polices have been reduced, our results show us traversal time is improved. Moreover, our proposed method has detected conflicts which cause network wide invariants.

5.2 Future Work

In this research work we provides solution to detect network wide invariants specifically unexpected packet lost due to conflicts in ACL polices. Future work includes detection of other network wide invariants like loops, black holes etc.

BIBLIOGRAPHY

[1]. A. Greenberg, G. Hjalmtysson, D. Maltz, D.A. Myers, J. Rexford, G. Xie, H. Zhang, "A clean slate 4D approach to network control and management". *ACM SIGCOMM Computer Communication*, vol. 35, no. 5, pp. 41-54, 2005.

[2] K.T. Foerster, S. Schmid, S. Vissicchio, "Survey of Consistent Software-Defined Network Updates". *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1435 – 1461, 2018.

[3] T. Benson, A. Akella, and D. Maltz, ''Unraveling the complexity of network management,'' *6th USENIX Symp. Networked Syst. Design Implement*, pp. 335–348, 2009.

[4] H. Kim and N. Feamster, ''Improving network management with software defined networking,'' *IEEE Communication letter. Mag*.,vol. 51, no. 2, pp. 114–119, 2013.

[5] N. McKeown et al., ''OpenFlow: Enabling innovation in campus networks,'' *SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[6] *Open Networking Foundation (ONF), [Online]. Available:* https://www.opennetworking.org/

[7] P. Newman, G. Minshall, and T. L. Lyon, ''IP switchingVATM under IP,'' *IEEE Transaction on Network,* vol. 6, no. 2, pp. 117–129, 1998.

[8] N. Gude et al., ''NOX: Towards an operating system for networks,'' *SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[9] D. Kreutz, F.M.V. Ramos, P.E., Veríssimo et al., "Software-defined networking: a comprehensive survey", *Proc. IEEE,* vol. 103, no. 1, pp. 14-76, 2015.

[10] J. Lu, Z. Zhang, T. Hu, P. Yi, J. Lan, "A Survey of Controller Placement Problem in Software-Defined Networking". *IEEE Access*, vol. 7, pp. 24290 – 24307, 2019.

[11] P.W. Tsai, C.W. Tsai, C.W. Hsu, and C.S. Yang, "Network Monitoring in Software-Defined Networking: A Review". *IEEE Systems Journal*, vol. 12, no. 4, 2018.

[12] D.S. Rana, S.A Dhondiyal, S. K. Chamoli, "Software Defined Networking (SDN) Challenges, issues and Solution", *JCSE*, vol. 7, no. 1, 2019.

[13] A. Nayyer, A.K. Sharma L.K. Awasthi, “Issues in Software-Defined Networking”. *Proceedings of 2nd International Conference on Communication, Springer*, vol. 46, pp. 989-997. 2018.

[14]. A. Khurshid, X. Zou,W. Zhou, M. Caesar, and P. B. Godfrey, “Veriflow: Verifying network-wide invariants in real time,” *in Proc. Present. 10th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, pp. 15-27. 2013

[15] G. VARGHESE, “Network Algorithmics: An interdisciplinary approach to designing fast networked devices”, Dec 2004.

[16] N. Handigol, B. Heller, V. Jeyakumar, D. Maziéres, & N. McKeown, “Where is the debugger for my software-defined network?”. *In Proceedings of the first workshop on hot topics in software defined networks* pp. 55–60. 2012.

[17] GDB: “The GNU project Debugger available online, https://www.gnu.org/software/gdb/”

[18] M. Hussain, N. Shah, “Automatic rule installation in case of policy change in software defined networks”, *N. Telecommunication System*, vol. 68, no. 3, pp. 461–477, 2018.

[19] P. Kazemian, G. Varghese, & N. McKeown, “Header space analysis: Static checking for networks”. *NSDI 12*, pp. 113–126. 2012.

[20] C. Prakash et al., “PGA: Using graphs to express and automatically reconcile network policies,” *ACM SIGCOMM Computer Communication Review*, pp. 29-42, 2015.

[21] M. Canini, D. Venzano, P. Peresini, D. Kostic, & J. Rexford. (2012). “A NICE way to test OpenFlow applications”. *NSDI, 12*, pp. 127–140, 2012.

[22] V. Zaborovsky, V. Mulukha, A. Silinenko, S. Kupreenko, “Dynamic Firewall Configuration: Security System Architecture and Algebra of the Filtering Rules”, *The Third International Conference on Evolving Internet. INTERNET*, 2011.

[23] A. Titov and V. Zaborovsky. “Firewall Configuration Based on Specifications of Access Policy and Network Environment”. *Proceedings of the 2010 International Conference on Security & Management.* 12-15, 2010.

[24] D.F. Ferraiolo and D.R. Kuhn. “Role-Based Access Control”. *15th National Computer Security Conference*. 1992.

[25] Organisation-based access control (OrBAC.org). Available: “*http://orbac.org/index.php?page=orbac&lang=en*”

[26] A. Silinenko. “Access control in IP networks based on virtual connection state models”, PhD. Thesis 05.13.19: / SPbSTU, Russia, 2010.

[27] R. Amin, N. Shah, B. Shah and O. Alfandi, “Auto-configuration of ACL policy in case of topology change in hybrid SDN,” *in IEEE Access*, pp. 9437 – 9450, 2016.

[28] M. Markovitch and S. Schmid, “SHEAR: A highly available and flexible network architecture marrying distributed and logically centralized control planes,” *in Proc. 23rd IEEE ICNP*, pp. 78-89, 2015.

[29] A. Silinenko. “Access control in IP networks based on virtual connection state models”. *PhD. Thesis 05.13.19: /SPbSTU,* Russia, 2010.