

11-2016

Early Packet Rejection Using Dynamic Binary Decision Diagram

Vasiqullah Molvizadah

Follow this and additional works at: https://scholarworks.uaeu.ac.ae/all_theses

Part of the [Information Security Commons](#)

Recommended Citation

Molvizadah, Vasiqullah, "Early Packet Rejection Using Dynamic Binary Decision Diagram" (2016). *Theses*. 450.
https://scholarworks.uaeu.ac.ae/all_theses/450

This Thesis is brought to you for free and open access by the Electronic Theses and Dissertations at Scholarworks@UAEU. It has been accepted for inclusion in Theses by an authorized administrator of Scholarworks@UAEU. For more information, please contact fadl.musa@uaeu.ac.ae.



United Arab Emirates University

College of Information Technology

Department of Information Systems and Security

EARLY PACKET REJECTION USING DYNAMIC BINARY DECISION DIAGRAM

Vasiqullah Molvizadah

This thesis is submitted in partial fulfilment of the requirements for the degree of
Master of Science in Information Security

Under the Supervision of Dr. Zouheir Trabelsi

November 2016

Copyright © 2016 Vasiqullah Molvizadah
All Rights Reserved

Advisory Committee

1) Advisor: Dr. Zouheir Trabelsi

Title: Associate Professor

Department of Information Systems & Security

College of Information Technology

2) Member: Dr. Ezedin S. Baraka

Title: Associate Professor

Department of Information Systems & Security

College of Information Technology

Approval of the Master Thesis

This Master Thesis is approved by the following Examining Committee Members:

- 1) Advisor (Committee Chair): Dr. Zouheir Trabelsi

Title: Associate Professor

Department of Information Systems & Security

College of Information Technology

Signature



Date

12/12/2016

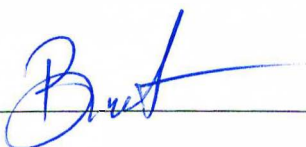
- 2) Member: Dr. Ezedin S. Baraka

Title: Associate Professor

Department of Information Systems & Security

College of Information Technology

Signature



Date

12/12/2016

- 3) Member (External Examiner): Dr. Fatma Outay

Title: Assistant Professor

Department of Innovative Technology

Institution: Zayed University

Signature




Date

12/12/2016

This Master Thesis is accepted by:

Dean of the College of Information Technology: Professor Omar El-Gayar

Signature 

Date Dec. 18, 2016

Dean of the College of the Graduate Studies: Professor Nagi T. Wakim

Signature 

Date 19/12/2016

Copy 7 of 7

Declaration of Original Work

I, Vasiqullah Molvizadah, the undersigned, a graduate student at the United Arab Emirates University (UAEU), and the author of this thesis entitled “*Early Packet Rejection using Dynamic Binary Decision Diagram*”, hereby, solemnly declare that this thesis is my own original research work that has been done and prepared by me under the supervision of Dr. Zouheir Trabelsi, in the College of Information Technology at UAEU. This work has not previously been presented or published, or formed the basis for the award of any academic degree, diploma or a similar title at this or any other university. Any materials borrowed from other sources (whether published or unpublished) and relied upon or included in my thesis have been properly cited and acknowledged in accordance with appropriate academic conventions. I further declare that there is no potential conflict of interest with respect to the research, data collection, authorship, presentation and/or publication of this thesis.

Student's Signature: _____ Date: _____

Abstract

A firewall is a hardware or software device that performs inspection on a given incoming/outgoing packets and decide whether to allow/deny the packet from entering/leaving the system. Firewall filters the packets by using a set of rules called firewall policies. The policies define what type of packets should be allowed or discarded. These policies describe the field values that the packet header must contain in order to match a policy in the firewall. The decision for any given packet is made by finding the first matching firewall policy, if any.

In a traditional firewall, the packet filter goes through each and every policy in the list until a matching rule is found, the same process is again repeated for every packet that enters the firewall. The sequential lookup that the firewall uses to find the matching rule is time consuming and the total time it takes to perform the lookup increases as the policy in the list increases. Nowadays, a typical enterprise based firewall will have 1000+ firewall policy in it, which is normal.

A major threat to network firewalls is specially crafted malicious packets that target the bottom rules of the firewall's entire set of filtering rules. This attack's main objective is to overload the firewall by processing a flood of network traffic that is matched against almost all the filtering rules before it gets rejected by a bottom rule. As a consequence of this malicious flooding network traffic, the firewall performance will decrease and the processing time of network traffic may increase significantly

The current research work is based on the observation that an alternative method for the firewall policies can provide a faster lookup and hence a better filtering performance. The method proposed in this research relies on a basic fact that the policy can be represented as a simple Boolean expression. Thus, Binary Decision Diagrams (BDDs), are used as a basis for the representation of access list in this study.

The contribution of this research work is a proposed method for representing firewall policies using BDDs to improve the performance of packet filtering. The proposed mechanism is called Static Shuffling Binary Decision Diagram (SS-BDD), and is based on restructuring of the Binary Decision Diagram (BDD) by using byte-wise data structure instead of using Field-wise data structure. Real world traffic is used during the simulation phase to prove the performance of packet filtering. The numerical results obtained by the simulation shows that the proposed technique improves the performance for packet filtering significantly on medium to long access lists. Furthermore, using BDDs for representing the firewall policies provides other useful characteristics that makes this a beneficial approach to in real world.

Keywords: Firewall, Packet Filter, Binary Decision Diagram, Early Rejection, Packet Matching.

Title and Abstract (in Arabic)

الرفض المبكر لطرود البيانات باستخدام رسم القرار الثنائي

الملخص

الجدار الناري هو جهاز أو برنامج يقوم بفحص طرود البيانات الواردة والخارجة، ثم يقرر إما السماح لها أو منعها من العبور. يرشح الجدار الناري طرود البيانات باستخدام قواعد تسمى سياسات الجدار الناري. تصف هذه السياسات قيم الحقول في بادئة طرد البيانات التي تماثل السياسة، ويتخذ الجدار الناري القرار بناءً على أول سياسة متطابقة.

في الجدر النارية التقليدية، يمر طرد البيانات على كل السياسات، واحدة تلو الأخرى حتى يصادف وجود سياسة مطابقة. تتكرر هذه العملية لكل الطرود بلا استثناء. ويعتبر البحث المتسلسل عن سياسة متطابقة مستهلكاً للوقت، كما أن الوقت اللازم للمطابقة يزداد طردياً بالنسبة لعدد السياسات المضبوطة. وتحتوي الجدر النارية على أكثر من ألف سياسة مضبوطة في هذه الأيام.

تشكل بعض الطرود خطراً على الجدر النارية، حيث تستهدف هذه الطرود الخبيثة آخر سياسة مضبوطة لكي ترهق الجدار الناري، في حين يعالج الجدار الناري هذه الطرود ويحاول مطابقتها مع كل السياسات المضبوطة حتى يصل إلى آخرها. وإن إرسال فيض من هذه الطرود الخبيثة يؤدي إلى نقص في أداء الجدار الناري وزيادة ملحوظة في وقت المعالجة.

إنّ البحث الحالي مبني على ملاحظة أن استخدام طرق أسرع للبحث سوف يؤدي إلى أداء أفضل، وهو مبني أيضاً على حقيقة أن السياسات يمكن تمثيلها كتعبيرات منطقية Boolean expressions. وببناءً على ذلك، تم استخدام رسم القرار الثنائي BDD في هذه الدراسة.

ويتجلى الإسهام في هذا العمل عبر طريقة مقترحة لتمثيل سياسات الجدار الناري باستخدام رسم القرار الثنائي BDD لتحسين أداء ترشيح طرود البيانات. وتسمى الطريقة المقدمة رسم القرار الثنائي المخلّط بسكون SS-BDD. وتستخدم هذه الطريقة إعادة ترتيب رسم القرار الثنائي باستخدام تكوين بيانات مبني على البايت byte بدلاً من الحقل. وقد تم استخدام طرود بيانات حقيقية في المحاكاة لإثبات فعالية الطريقة، كما أن القياسات المستخرجة تظهر أن هذه الطريقة تحسن الأداء بشكل ملحوظ في السياسات متوسطة الطول والطويلة.

مفاهيم البحث الرئيسية: الجدار الناري، ترشيح طرود البيانات، رسم القرار الثنائي، الرفض المبكر، مطابقة طرود البيانات.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor Dr. Zouheir Trabelsi for the continuous support of my study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my Master's study.

I would like to thank the chair and all members of the Department of Information Systems & Security at the United Arab Emirates University for assisting me all over my studies and research. My special thanks are extended to Dr. Mehedy Masud for providing me with the relevant reference material.

Special thanks go to my family: my parents and to my brothers for supporting me throughout writing this thesis and my life in general.

Dedication

To my beloved parents and family

Table of Contents

Title	i
Declaration of Original Work	ii
Copyright	iii
Advisory Committee	iv
Approval of the Master Thesis	v
Abstract	vii
Title and Abstract (in Arabic)	ix
Acknowledgements	xi
Dedication	xii
Table of Contents	xiii
List of Tables.....	xvi
List of Figures	xvii
Chapter 1: Introduction	1
1.2 Firewall Basic Approaches	2
1.2.1 Circuit level firewall	3
1.2.2 Application-level firewall	3
1.2.3 Packet filtering firewall.....	3
1.3 Firewall Policy	4
1.4 Problem Statement	5
Chapter 2: Related Work.....	7
Chapter 3: Using Binary Decision Diagram for Packet Filtering	8
3.1 Binary Decision Diagram.....	8
3.2 Ordering and Reducing	9
3.3 The Variable Ordering Effect.....	10
3.4 Binary Decision Diagram Packet Filter	11
3.4.1 Boolean Variables	12
3.5 Example of Firewall Policy list conversion	12
3.6 Performing a Lookup	15

3.7 Issues Field-wise lookup	15
Chapter 4: Static Shuffling Binary Decision Diagram (SSBDD)	18
4.1.1 Policy Representation Phase	20
4.1.2 Packet Filtering Phase	21
4.2 Field Ordering	22
4.2.1 Field Count Distribution	24
4.2.2 Rule Reordering vs Field Reordering	25
Chapter 5: Implementation.....	26
5.1 Policy Representation using SSBDD	26
5.1.1 Input the Firewall Rules	26
5.1.2 Rule to Binary Format.....	28
5.1.3 SSBDD Generation	30
5.2 SSBDD Packet Filtering	31
5.3 BDD Packet Filter vs SSBDD Packet Filter	33
5.3.1 BDD Packet Filter	34
5.3.2 SSBDD Packet Filter	35
Chapter 6: Analytical Discussion of the SSBDD.....	37
6.1 SSBDD Representation of Access List	37
6.1.1 The Effect of Rule Structure	39
6.1.2 The Effect of Variable Ordering	39
6.2 Performing Lookup on a SSBDD	40
6.2.1 Worst Case Analysis	40
6.2.2 Best Case Analysis.....	41
6.3 Memory Usage of SSBDD.....	42
Chapter 7: Architecture and Simulation.....	43
7.1 System Implementation.....	43
7.1.1 Packet Filter Specification	43
7.2 Coding and Modules	44
7.2.1 RunMe.py	45
7.2.2 Formula.py	46
7.2.3 MatchRule.py	47
7.3 Other coding modules	47

7.4 Simulation Environment	48
7.5 Simulation Dataset	49
7.6 Simulation Framework.....	49
Chapter 8: Statistics	51
8.1 Simulation Results	51
8.1.1 Experiment A	52
8.1.2 Experiment B	52
8.1.3 Experiment C	53
8.1.4 Experiment D	53
8.1.5 Experiment E.....	54
Chapter 9: Conclusion and Future Work	55
9.1 Future Work	55
9.1.1 Variable Ordering and Reordering Prediction	55
9.1.2 Updating the SSBDD	56
9.1.3 Considering more Parameters for Performance	56
9.2 Conclusion	56
References	57

List of Tables

Table 1: The Boolean variable required for BDD representation	12
Table 2: Sample Firewall Policy List	13
Table 3: Rules	16
Table 4: A Sample Policy List	20
Table 5: Variable Ordering	22
Table 6: Binary equivalent of a rule	34
Table 7: Sample Access List	38
Table 8: Specification	49

List of Figures

Figure 1: Binary Decision Diagram	9
Figure 2: Two BDDs for the function $x_1y_1 \vee x_2y_2 \vee \dots \vee x_ny_n$ for $n=3$	11
Figure 3: BDD Lookup	15
Figure 4: BDD Tree of Table 3	16
Figure 5: SSBDD Architecture	19
Figure 6: BDD Representation.....	21
Figure 7: BDD Packet Filter	34
Figure 8: SSBDD Packet Filter.....	35
Figure 9: Modules of SSBDD implementation.....	45
Figure 10: Other Coding Modules.	48
Figure 11: Simulation Framework Overview	50
Figure 12: Experiment A Result	52
Figure 13: Experiment B Result.....	52
Figure 14: Experiment C Result.....	53
Figure 15: Experiment D Result	53
Figure 16: Experiment E Result.....	54

Chapter 1: Introduction

1.1 Overview

The Internet has come a long way since its inception, the accomplishment in terms of data accessibility and availability has been growing exponentially over the couple of decades (Cheswick, 2003). Today every other business is now reorganizing itself to utilize the power of the Internet to connect to its users. The type of services and application available on the Internet have become more powerful – starting from a simple static webpage in the 1990s to online banking, shopping. The fact of increasing number of users using the Internet implies an increasing number of malicious attacks, which means that systems and their networks require protection from unintentional incidents as well as malicious acts (Nikolaidis, 2000).

The increasing complexity of the Internet makes the solution of computer network security more complex, which is why organization does not use just one solution instead they apply layers of security to protect themselves. The best way of ensuring security is by using a network firewall. A firewall is a computer, router, or other communication device that filters access to the protected network (Nikolaidis, 2000). Cheswick and Bellovin (Cheswick, 2003) (Ballew, 1997) define a firewall as a collection of components or a system that is placed between two networks and possesses the following properties:

- All traffic from inside to outside, and vice-versa, must pass through it.
- Only authorized traffic, as defined by the local security policy, is allowed to pass through it.
- The firewall itself is immune to penetration.

Firewalling is the easiest method of all used by the network administrator to control the access between networks (Ballew, 1997). The idea to use firewall to protect the network, is that controlling access to the network and its resources by protecting each host is difficult and does not scale (Oppliger, 1998). Firewall solves this issue by creating a single connection point for multiple network and providing a single security checkpoint. This single checkpoint will have a security policy that defines what type of connection is allowed or rejected. It can be assumed that the firewall itself is immune to penetration.

One of the main criticism of firewalls is that they often create bottlenecks (Nikolaidis, 2000). The reason of this bottleneck is mainly how the firewall policies are constructed. If the policies are not constructed properly then it may cause loss of network performance. This motivates the need for faster firewall technologies, keeping in mind that there is a tradeoff between performance and security.

1.2 Firewall Basic Approaches

Firewall is usually installed at the edge of the network where the private or the Intranet connects to the public network, making it easier for the firewall to monitor all the traffic at once. Although firewall may also be placed between departmental networks within a company. The level of security and behavior exhibited by the firewall depends on the type of firewall used but for this research is focused on the Packet Filter Firewall. There are three basic approaches that a firewall uses to protect the network: packet filtering, circuit level firewall, and application level firewall (Cheswick, 2003)

1.2.1 Circuit level firewall

Circuit level firewall is a type of firewall that works at the session layer of the OSI model, between the application layer and the transport layer of the TCP/IP stack. They monitor the handshaking between two systems and decided whether a request session is legitimate or not. It filters the packet, by relying on the data contained in the packet headers for the TCP session-layer protocol. These type of firewalls usually operates two layers higher than a packet-filtering firewall does. It determines if the requested session is legitimate or not by checking the SYN flags, ACK flags, and the sequence numbers are involved in the TCP handshaking or not. The issue with circuit-level proxy is that it has no understanding of the application protocols they support. They cannot scan application data for dangerous commands or executable contents.

1.2.2 Application-level firewall

Application-level proxy operates at the application layer of the firewall. An application-level runs a proxy server for each application that it supports. The proxy request on behalf of the user to the destination host. Proxy server has some understanding of the application it is supporting and can be configured to reject malicious content packets. Application level firewall are not easy to scale.

1.2.3 Packet filtering firewall

Packet filtering firewall operates at the network layer and is the simplest type of firewall. Since, the firewall operates at the network layer so it has no idea of the content of the packets like the other type of firewall mentioned above. The packet filtering firewall works on the concept of policies. The policies use the information –

source port, destination port, source ip, destination ip, and protocol – to filter malicious traffic from the network.

Security is provided by comparing the packets against the list of the firewall rules and deciding whether to allow or deny the packets based on the action defined in the matched rule. Packet filtering firewall is widely used as a first line of defense in any enterprise. There are numerous reason for it (Cheswick, 2003) (Oppliger, 1998)

- Faster than other firewall technologies
- It is a low-cost technology. Many commercial routers have the packet filtering capabilities in it. There is various free open-source packet filtering firewall available.
- It is normally transparent to applications and users.

1.3 Firewall Policy

The packet filter firewall is usually specified by a set of rules. The rules are a simple if-then-else structure with each rule defining the action that should be taken, if any packet matches. A set of rules in the firewall is known as access control list, policy list or rules (Ballew, 1997). The firewall traverses the rules sequentially to find the matching rule for any given packet.

Defining the firewall policy is simple for any user but it has its own disadvantages if they are not defined properly. The order of how the rules are inserted and represented in the firewall is of high importance which can affect the overall performance of the firewall. For this reason, the packet filtering implementation represents the list of policies in the firewall in a linear fashion. The decision making process called lookup

goes through each rule one at a time in a linear fashion and decided whether the packet should be accepted or rejected until a matching rule is found. The time taken to perform the lookup is clearly proportional to the number of rules in the firewall.

1.4 Problem Statement

The main aim of this research is to propose a new representation technique called Static Shuffling Binary Decision Diagram or SSBDD for the access list of the firewall. The reason to propose a new representation technique is because the firewall rules are consulted more frequently and they are modified less frequently. There are many representation techniques for the firewall policies which are discussed briefly in the Related Work Section but for this research, Binary Decision Diagram or BDD is chosen as the base for the representation of the access list. SSBDD is a modified version of the regular BDD. Using BDD as the representation technique has its own advantages such as: -

- Each of the rule in the firewall is simply a logical expression that is based on the values in the rule. If any packet satisfies the condition in the rule, then the packet is either accepted or rejected based on the action in the rule.
- The entire firewall access list is represented as a single Boolean expression that describes (Gupta, 2001) (Trabelsi, 2014) what condition each packet must meet. BDD is a very well-known data structure for storing and manipulating Boolean expressions compactly and efficiently.

This research addresses the following question:

How to perform an early packet rejection using Binary Decision Diagram as its data structure?

Chapter 2: Related Work

Packet filtering consists into performing a sequential lookup for each network packet against the rule list until a matching rule is found. Due to the sequential lookup nature of the firewall, the performance of the firewall will degrade over the time if the size of the rule list of the firewall increases. Different approaches have been proposed to improve firewall performance, using mainly, specialized data structure (Srinivasan, 1999) or heuristics solutions (Gupta, 2001).

The idea of firewall optimization using data mining is discussed in (Trabelsi, 2014). The proposed technique uses classifier for packet filtering. At first, the technique tries to get the matching classifiers. If it is unable to get any classifier, then it will use the firewalls sequential lookup to find the matching rule.

Another approach is discussed in (Boutaba, 2009), which uses BDD to generate a relaxed version of the firewall rules that can be evaluated more quickly. After processing a packet, the proposed technique will conclude to one of the three following options: accept, reject, or more filtering is required. In case of more filtering, the original policy will be used to look for a matching rule in the list, if any. In (Zeidan, 2012), Splay Tree based technique (Statistical Splay Filtering with Binary Search on Prefix Length) is used to improve the firewall performance. The optimization technique allowed the firewall to perform an early packet rejection through multilevel filtering process including field and intersection filtering modules.

Chapter 3: Using Binary Decision Diagram for Packet Filtering

The original idea of BDD packet filter was developed by (Hazelhurst, 1998) (Bryant, 1992). BDD provides a powerful and flexible way to represent the policies of the firewall. Each policy of the firewall can be represented using BDD via a simple Boolean expression. Boolean expression is simply consisting of a number of predicates, where each predicate shows what path to follow in the BDD.

Since, in BDD each policy of the firewall is represented as a Boolean expression and as the number of policy increases the size of the BDD will increase as well. However, BDDs are well-known for its compact representation of Boolean expression. So using BDD as a packet filtering approach can provide an advantage in terms of performance, which is the most important factor for any firewall. This chapter is devoted to describing the BDD approach to packet filtering in detail.

3.1 Binary Decision Diagram

Binary Decision Diagram represent Boolean functions as rooted, directed acyclic graphs (Bryant, 1992). In a non-technical term, a BDD can look like a decision tree, as shown in Figure. 1. Each non-terminal node in a BDD represents a value to a particular variable, and each non-terminal node has two children representing the possible value for the non-terminal node (0 or 1). The dashed edge corresponds to the case where the variable is assigned 0 and the solid edge corresponds to the case where the variable is assigned 1. A BDD has two terminal nodes which are Boolean constants and has a value of 0 and 1.

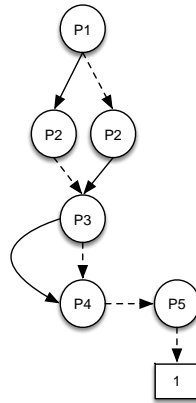


Figure 1: Binary Decision Diagram

3.2 Ordering and Reducing

The issue with BDD representation is, as the number of policy increases in the firewall the size of the BDD also increases as it will increase the Boolean expressions. To overcome this problem, (Bryant, 1992) introduced the concept of reduced, ordered binary decision diagrams (ROBDDs) that potentially provide a much more compact representation for many Boolean expressions. ROBDDs are basically a compact version of a BDDs due to the following restrictions on it (Bryant, 1992).

- The variables of a ROBDD must obey a total ordering, so that for any vertex labelled u and any of its children labelled v , u appears before v along any path from the root of the graph to a leaf.
- A ROBDD may not contain duplicate terminals. This leaves a ROBDD with a two terminal vertices (one labelled 0 and the other labelled 1).
- A ROBDD may not contain duplicate non-terminals. Duplicate non-terminals are those that represent the same variable where the corresponding branches lead to the same place.

- A ROBDD may not contain redundant tests. If, at a particular vertex in the graph, both possible values lead to the same place, then this test is unnecessary.

These restrictions result in ROBDDs possessing some useful properties. Firstly, they provide compact representations of Boolean functions. Although in the worst case, their graph size can be exponential in the number of variables, many non-trivial Boolean functions have a polynomial size ROBDD (Bryant, 1992). Since ROBDDs offers so many advantages over unrestricted BDDs, most applications that use BDDs actually use ROBDDs, so it is very common to simply refer ROBDDs as BDDs (Andersen, 1997).

3.3 The Variable Ordering Effect

The variable ordering chosen for a BDD has a strong impact on its shape and size (Bryant, 1992). If the variable order is not chosen correctly, then it can make the BDD for the same Boolean function from a linearly sized to an exponentially sized BDD as shown in Figure 2.

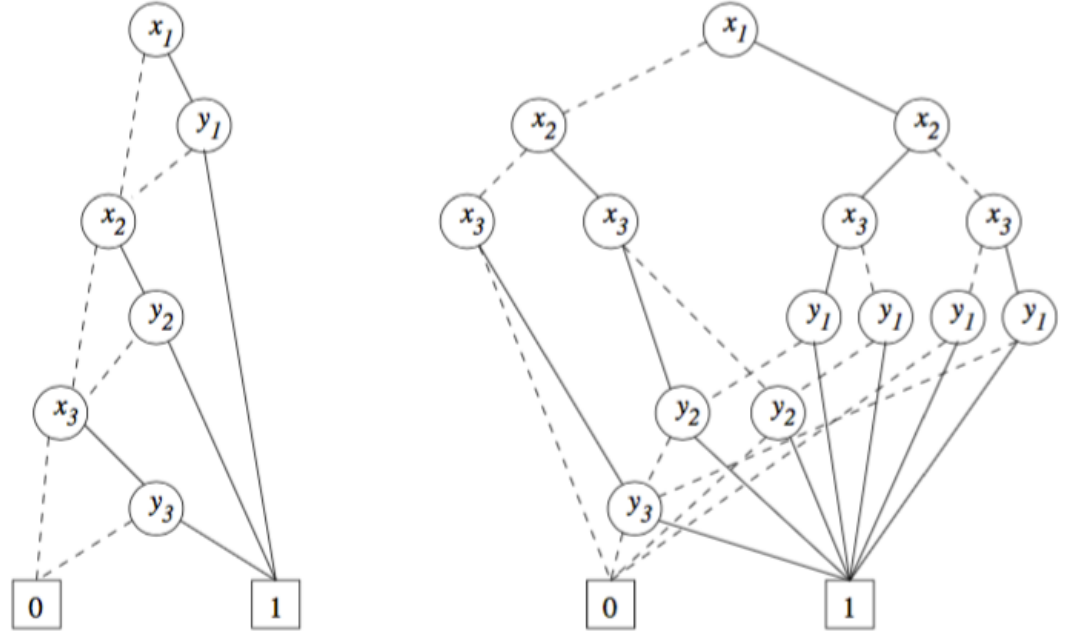


Figure 2: Two BDDs for the function $x_1y_1 \vee x_2y_2 \vee \dots \vee x_ny_1$ for $n=3$

For a function that cannot be represented in a compact format, it is best to choose an optimal variable ordering for it. However, finding an optimal variable ordering for a BDD is a NP-complete problem (Bollig, 1996). As a result, variable order are often chosen manually or using some heuristics.

3.4 Binary Decision Diagram Packet Filter

In a BDD as a packet filter, a BDD is used to represent the firewall's entire policy list, and the same BDD is then used to perform lookup on the incoming packets. The representation of the BDD is stored in the memory and whenever a new policy is added to the list the BDD is then regenerated again. The BDD is a representation of the Boolean expression that describes exactly what packets must be accepted or rejected. In simple words, all paths through the BDD that lead to the terminal labelled 1 represent the types of packets that are accepted, and the opposite is true for

all paths leading to the terminal labelled 0. Each node or variable in the BDD refers to a specific bit in the packet header.

3.4.1 Boolean Variables

To represent a firewall policy as a BDD it is important to know what variable in the BDD refers to which field in the packet header, as each variable in the BDD corresponds to a specific bit in the packet header itself. A BDD's Boolean expression will consist of multiple variable; in a normal case each policy is represented by at least 104 Boolean variables. Below is the table that describes the variable naming that will be used throughout this work.

Header Field	Boolean Variables	Total Number
Source IP address	s_ip1 ... s_ip32	32
Destination IP address	d_ip1 ... d_ip32	32
Protocol type	p1 ... p8	8
Source port	s_p1 ... s_p16	16
Destination Port	d_p1 ... d_p16	16
Total		104

Table 1: The Boolean variable required for BDD representation

3.5 Example of Firewall Policy list conversion

The example below demonstrates how a firewall policy list can be converted into a Boolean expression. The policy list to be used for the conversion process is shown in the Table 2. This BDD has its protocol variables ordered first, followed by the variables corresponding to destination information, followed by the variables corresponding to the source information. The default policy of the firewall is to deny all packets.

Rule	Proto	Source IP	Source Port	Destination IP	Destination Port	Action

1	TCP	172.21.1.89	9070	10.2.12.98	80	Permit
2	TCP	172.25.12.1	7788	81.23.1.87	443	Permit
Default Policy.						

Table 2: Sample Firewall Policy List

Step 1: Defining the Boolean Variables

The following 5 fields - source address, destination address, source port, destination port, and protocol – are used by the packet filter on the incoming packets. By summing up all the field sizes of the header gives a total of 104 bits, so a total of 104 variables are required to represent the BDD of this access list. The variable naming to be used for the BDD representation is shown in Table 1.

Step 2: Converting Individual Rules

This step involves in converting the give policy into a Boolean expression. This is accomplished by forming a conjunction of each predicate. Let R_i denote the Boolean representation of Rule I . To convert the first rule:

Let p = Protocol = TCP
 $= p_8' p_7' p_6' p_5' p_4' p_3 p_2 p_1'$

Let s_p = Source Port = 9070
 $= s_{p16}' s_{p15}' s_{p14} s_{p13}' s_{p12}' s_{p11}' s_{p10} s_{p9} s_{p8}' s_{p7} s_{p6}$
 $s_{p5}' s_{p4} s_{p3} s_{p2} s_{p1}'$

Let d_p = Destination Port = 80
 $= d_{p16}' d_{p15}' d_{p14}' d_{p13}' d_{p12}' d_{p11}' d_{p10}' d_{p9}' d_{p8}' d_{p7}$
 $d_{p6}' d_{p5} d_{p4}' d_{p3}' d_{p2}' d_{p1}'$

Let s_{ip} = Source IP = 172.21.1.89
 $= s_{ip32} s_{ip31}' s_{ip30} s_{ip29}' s_{ip28} s_{ip27} s_{ip26}' s_{ip25}' s_{ip24}'$
 $s_{ip23}' s_{ip22}' s_{ip21} s_{ip20}' s_{ip19} s_{ip18}' s_{ip17} s_{ip16}' s_{ip15}' s_{ip14}'$
 $s_{ip13}' s_{ip12}' s_{ip11}' s_{ip10}' s_{ip9} s_{ip8}' s_{ip7} s_{ip6}' s_{ip5} s_{ip4} s_{ip3}'$
 $s_{ip2}' s_{ip1}$

Let d_{ip} = Destination IP = 10.2.12.98
 $= d_{ip32}' d_{ip31}' d_{ip30}' d_{ip29}' d_{ip28} d_{ip27}' d_{ip26} d_{ip25}' d_{ip24}'$
 $d_{ip23}' d_{ip22}' d_{ip21}' d_{ip20}' d_{ip19}' d_{ip18} d_{ip17}' d_{ip16}' d_{ip15}'$
 $d_{ip14}' d_{ip13}' d_{ip12} d_{ip11} d_{ip10}' d_{ip9}' d_{ip8}' d_{ip7} d_{ip6} d_{ip5}'$
 $d_{ip4}' d_{ip3}' d_{ip2} d_{ip1}'$

Then $R1 = p \wedge s_p \wedge d_p \wedge s_{ip} \wedge d_{ip}$

The expression is constructed similarly for the other rules.

Step 3: Combining all the rules

Using the steps shown above the expression can be generated for all the rules in the same way and once it is done, the next step is generating a single expression for all the rules. The expression is generated as follows:

$$\text{Final expression} = (R1 \vee R2)$$

The final expression says that the incoming packets are either accepted by Rule R1 or Rule R2.

3.6 Performing a Lookup

In a linear or classic BDD, once generated, performing a lookup on the given incoming packet is simply a comparison. The comparison starts from the top node or root node of the BDD and continues till it reaches a terminal node either 0 or 1. In case if it reaches terminal node 0, then the given packet is rejected. However, if it reaches the terminal node 1, then the given packet is accepted.

Figure 3 shows how a regular BDD performs a lookup upon receiving an incoming packet. In this paper, this search will be referred as Field-wise search since it checks one entire field at a time.

Source IP	1	192.168.33.247 Rule #1 192.168.22.80 Rule #2
Source Port	2	1050 Rule #1 8900 Rule #2
Destination IP	3	192.168.32.125 Rule #1 10.11.43.21 Rule #2
Destination Port	4	3389 Rule #1 443 Rule #2
Protocol	5	TCP Rule #1 TCP Rule #2

Figure 3: BDD Lookup

3.7 Issues Field-wise lookup

Field-wise lookup gives acceptable performance when the ratio of the accepted traffic is higher than the ratio of the rejected traffic. But, if the ratio of accepted traffic is very less, compared to the rejected traffic's ratio, the field-wise search performance will degrade. This is because in attacks, like DoS, the traffic usually gets rejected at the bottom rules of the firewall.

To understand the issues of field-wise BDD lookup, let's take an example of two filtering rules with a protocol field, as shown in Table 3, whose BDD is shown in Figure 4.

Protocol	Decimal	Binary
TCP	6	0110 0000
ICMP	1	1000 0000

Table 3: Rules

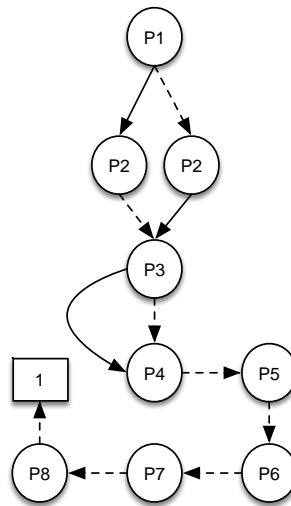


Figure 4: BDD Tree of Table 3

Considering DoS – it mainly targets the firewall by sending malicious traffic targeting the bottom rules of the firewall – the higher the rejection ratio the more likely the BDD performance is going to degrade. This is due to the fact, the entire BDD will be traversed to reach the final decision.

The issue of the Field-wise lookup can be solved by simply shuffling the field-order. For example, instead of checking the IP fields of the packet, it's much better to check the protocol field of the incoming packets. As the field-size is of 8-bit, and a high

amount of packet rejection occurs at this field. This solution won't help the firewall to maintain its performance for a long period. The traffic received by the firewall are random in nature, so relying on one field for early rejection won't work.

The solution is interesting but in-order to maintain the performance, the firewall needs more information, but not about the firewall rules, it needs information about the traffic. The traffic always gives you more information about why and how the firewall is not performing well. The type of characteristics that can help the BDD to perform well, are the high rejection nodes. Rejection nodes are the nodes in the firewall, that keeps track of all the nodes in the BDD tree that has the highest number of rejection. This rejection could be either due incoming packet not matching any specific field in the firewall or incoming packet not matching any rules at all in the firewall.

In a linear-based BDD, the rejection nodes won't be of much use, as the BDD is generated only once, the regeneration only happens when a new rule is either added or deleted from the firewall. To overcome this problem, two new approach is proposed and explained in detail in the following section. The approach is basically based on using the traffic characteristics of the firewall and then generating a new BDD from time-to-time.

Chapter 4: Static Shuffling Binary Decision Diagram (SSBDD)

In this chapter a new method is proposed and it is called Static Shuffling Binary Decision Diagram or SSBDD. SSBDD is an improvement over a regular Binary Decision Diagram Packet Filter. The proposed method improves the performance of the firewall specially when the traffic it is receiving has a high rate of rejection packets. SSBDD uses the BDD as its base.

It also adds two more properties on top of regular BDD to improve the performance; the Field Ordering and the Split size as shown in Figure 5, both of them will be discussed extensively in the coming chapters. Together with these two properties an efficient and optimized BDD is generated. The name Static Shuffling comes from the idea of the way it generates and parse the packet headers; instead of following the traditional way of reading the entire field, it reads n bits from each field. One of the advantage of this method, it is not affected by the dependency of rules in the firewall because it relies on traffic log instead of rule analysis

There have been various studies conducted on the improvement of the firewall's packet filter. Most of the research focused on the rule analysis of the firewall or rejection traffic of the firewall. In our case, SSBDD is focused on rejection traffic because in any BDD, the acceptance traffic will always traverse the entire BDD, which is not the case in rejection traffic.

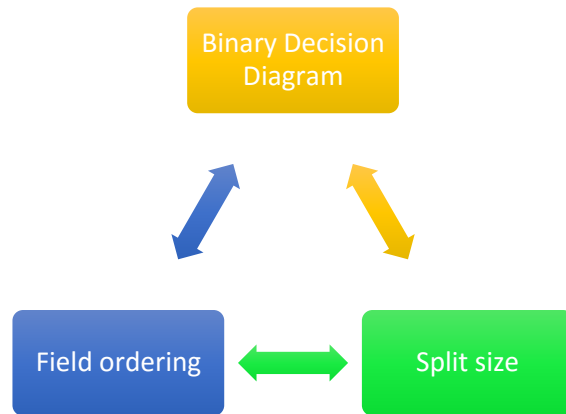


Figure 5: SSBDD Architecture

4.1 Split Size

Split size in the context of BDD defines how many bits the BDD should traverse in each field before moving on to the next field. In the previous chapter, the importance of the traffic characteristics was discussed. For e.g. the Protocol Distribution section showed that checking 4 bits of protocol field will give the result more quickly than checking the 8 bits of the field. Split size property is defined at the beginning of the BDD generation. It is not only used at the parsing phase but it also used during the policy representation phase of the BDD. The use of the split size in the two different context of BDD is explained in the coming sections. The only downside of the split size, is it cannot be an odd number and the value cannot be more than 8, this is not due to a performance issue but it is merely due to programming limitations.

The idea of split size is not the first time it is used in firewall packet filtering; the same technique is described (Boutaba, 2009). There's no need for the BDD packet filter to go through an entire field before moving on the next field. Separating them into a non-contiguous block can be useful too. For example, the protocol numbers might be best represented if it's bits be mapped from P0-P4 then P5-P8. So instead of

checking the entire 8 bits' fields from P0-P8, it is a good idea to check the first 4 bits and then the rest of the bits later.

4.1.1 Policy Representation Phase

Choosing the right split size for the BDD generation is as same as choosing a right variable ordering for a BDD, which is an NP-complete problem. But based on the analysis, it is possible to decide the split size to be used for the BDD. For e.g. based on the Protocol Distribution section, it is shown that high number of traffic were related to the TCP protocol. The split size can then be changed easily by analyzing the traffic characteristics but it can only be changed once during the BDD generation phase.

For example, assume having a simple packet header that consist of only source and destination fields each is just 4-bits long with all the rule's decision is Allow. Table 4 shows the demo packet header converted into Binary equivalent. Assuming the split size is set to 2. The Fig 6 shows how the Rule 1 will be represented.

Rule #	Source	Destination
R1	1011	1100
R2	1001	1110
R3	1010	1101

Table 4: A Sample Policy List

The left figure in Figure 6 shows the BDD representation of the firewall when the split size is considered and the figure on the right in Figure 6 shows the BDD representation of the firewall when the split size is not considered.

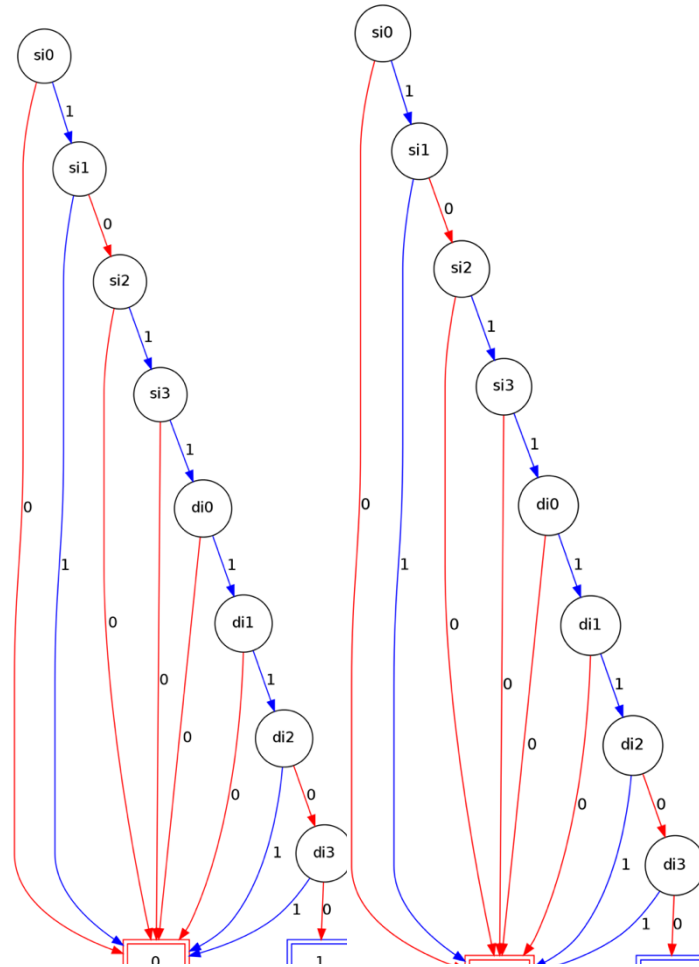


Figure 6: BDD Representation

4.1.2 Packet Filtering Phase

In the BDD packet filtering phase the split size defines the number of bits that must be checked by the packet filter before moving on to the next field in the firewall. So instead of considering the entire field of the firewall as one contiguous block, it is better to consider them as a non-contiguous block. For e.g. if a split size of 8 is chosen then the bits of the fields will be represented as shown in Table 5.

Field Field bits BDD variable	Proto (0,7) 0,7	Source IP (0,7) 8,15	Destination IP (0,7) 16, 23	Source Port (0,7) 24, 31
	Destination Port (0,7) 32,39	Source IP (8,16) 40, 47	Destination IP (8,16) 48, 55	Source Port (8,16) 56,63
	Destination Port (8,16) 64, 71	Source IP (16,24) 72, 79	Destination IP (16, 24) 80, 87	Source IP (24, 32) 88, 95
	Destination IP (24, 32) 96, 103			

Table 5: Variable Ordering

4.2 Field Ordering

Majority of the packet filtering devices like firewalls do not give specific consideration for optimizing packet rejection. If a packet does not match any of the rules in the policy, then it is discarded because the default rule (last rule) is assumed to be deny (Hamed, Discovery of policy anomalies in distributed firewalls, 2004). It is highly crucial for any firewall to implement a successful packet filtering. However, most of the packet filtering research done by authors focuses on exploiting the characteristics of filtering rules and ignores to consider the traffic behavior as another factor for optimization (Al-Shaer, 2006) schemes.

Optimization of the firewall packet filtering can be done at various stages during packet filtering process. Since, our work focuses only on BDD, so only the optimization technique related to BDD is discussed in this research. One of the most important optimization technique is field order, field order plays a very important role in the firewall. During network attacks such as DOS, the traffic is created in

such a way, that packets will always get rejected by the bottom rules or the last field of the bottom rules. If the fields are ordered in an optimal way, then the chance of packets being rejected at early stage increases which will then improve the performance of the firewall. But in a traditional firewall packet filter, the field order is fixed, which causes the performance to degrade during such network attacks.

As the networking speed increases, it is very important for the firewall to improve its packet filtering performance. Time is an important factor when considering the performance of the packet filter. To improve the performance, it is much more important to focus on the rejection packets of the firewall because if the rejection packets are rejected by the deny-all rule then it can cause more harm to the performance. Thus, it is more important to focus on early packet rejection.

There is an extensive amount of research work done on packet classification. The basic approach is to search the rules sequentially till a match is found. This approach is not time efficient because as the rule list increases the search time increases as well. So the performance of the basic approach is proportional to the length of the rule list in the firewall. Research on improving the search time for packet filtering uses one or more of the following approach: hardware-based solutions, specialized data structures, geometric algorithms, and heuristics (Al-Shaer, 2006).

Our study of the network traffic collected from (CADA, n.d.) shows that the major portion of the traffic flows gets rejected at a field in the firewall rules. It is also observed that this distribution is likely to stay for a time interval, if this distribution property is considered then it is highly likely that it can improve the performance of the packet filter. Therefore, a new method is proposed in this research, that uses the field distribution as one of the factor to improve the performance of the packet filter.

The proposed method called, Static Shuffling BDD [SSBDD] uses a typical Binary Decision Diagram as its base. The tree is mainly focused to reject the traffic as early as possible because a rejected packet might traverse long decision path of rule matching before getting rejected by the default-to-deny rule in the firewall. As the number of rejection packets increases the performance degrades as it causes significant overhead on the firewall. The implementation of SSBDD does not require any sort of special support from the firewall.

The SSBDD relies on field ordering for its optimization. Early rejection is possible in firewall if the field order is chosen efficiently. But choosing an optimal field order is an NP complete problem as the traffic is always random in nature. Predicting the type of traffic is not possible but a few assumptions can be made for any traffic based on the traffic characteristics.

4.2.1 Field Count Distribution

Another study conducted on the traffics to see the distribution of field counts. This study provides much more detailed overview into at which bit or node level of the field the packet got rejected. This data at first seems not useful but when collected from time to time can provide an overview into which fields is more important and can help to prioritize one field over other. It can also provide information to BDD, whether the BDD should start reading the packet headers from MSB to LSB or the other way around.

For e.g. given any network traffic that is received from the router to the firewall, it is safe to assume that all the traffic reaching the firewall will belong to the network. Since, the organization has LAN network, so they'll be having Private Address space. In that case, it is always best to start the search from MSB to LSB instead of

LSB to MSB. So it is suggested that depending on the traffic specifications the less important fields should be checked later in order to speed up the performance of the packet filter.

4.2.2 Rule Reordering vs Field Reordering

Rule reordering is the most focused area in the field of packet filtering. There has been various research work done on it. Based on the analysis shown above in Rule hit distribution, it is visible that reordering of the rule can have an impact on the packet filtering performance. But the success of rule reordering is mainly dependent on the how interconnected the rules are in the firewall rule list. If the majority of the rules are dependent, then rule reordering will not improve the firewall performance.

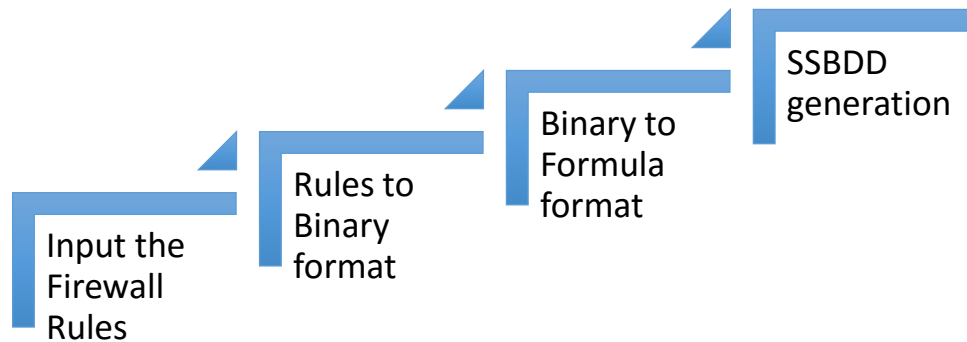
On the other hand, field reordering seems like a good alternate solution. The main advantage of field reordering, it is not affected by the rule dependency. Because the field ordering happens at the search time. If given enough information about the traffic log, then an optimal field reordering can be achieved which can improve the performance of the firewall overall.

Chapter 5: Implementation

In this chapter we will discuss about the implementation part of our proposed method. The implementation is applied once during the packet generation phase and then during the packet filtering phase, both of them are explained in detail. At last we check out the lookup comparison between the two of them.

5.1 Policy Representation using SSBDD

To generate an SSBDD for a given set of firewall rules, it has to go through 4 stages which starts from the taking the rule set as an input and onto the final stage that converts the rules into the final SSBDD. The chart below gives you the overview of what happens at each and every stage of the SSBDD generation.



The final SSBDD which is generated is in a graphical format i.e. DOT format, the DOT format is only the user but the generation of the DOT format file is disabled as it increases the CPU processing time. The another format which is generated is a tuple set for each node in the BDD, this format is used for the parsing.

5.1.1 Input the Firewall Rules

The input file used by the program which contains the rule set or the traffic is in CSV format. The packets were collected from CAIDA. The metadata contains more information than it is needed so only the necessary information is collected from the

file; source ip, source port, destination ip, destination port, and protocol. The format of the file for both the traffic and the firewall rule set is given below.

<ip.proto,ip.len,ip.src,ip.dst,tcp.srcport,tcp.dstport,tcp.flags,udp.srcport,udp.dstport,icmp.type,icmp.code>

```
with open(ruleFile, "rb") as csvfile:

    pkt_reader = csv.reader(csvfile)
    del dyn_order_var[:] # Resetting the dyn_order_var now ...
    read_config_file()

    for pkts in pkt_reader:

        newPkt = {}
        for order in ruleConf.order:
            if order == 'p':
                newPkt['p'] = pkts[field_pos['p']]
            elif order == 's_ip':
                newPkt['s_ip'] = pkts[field_pos['s_ip']]
            elif order == 's_p':
                newPkt['s_p'] = pkts[field_pos['s_p']]
            elif order == 'd_ip':
                newPkt['d_ip'] = pkts[field_pos['d_ip']]
            elif order == 'd_p':
                newPkt['d_p'] = pkts[field_pos['d_p']]

        ruleListBin.append(dyn_ruleFormula(ruleToBin(newPkt)))
```

The above is the algorithm for reading the rules from the CSV file is given below. The file is opened in a reading mode and the code goes through the file line-by-line as each line contains one firewall rule in it. For every firewall rule that is read from the file is converted into first binary equivalent and then it is converted into the formula format. Once the above code is executed, the code will have the entire rule set converted into the formula format which is then written to the CSV file.

5.1.2 Rule to Binary Format

In order for the BDD to be generated the values must be in binary format, so it is necessary to convert each and every rule or packet header into the equivalent binary format. The discussion on Binary conversion is also discussed in more detail in the Chapter BDD. This step is basically a conversion of the decimal values to the binary format. The algorithm for the binary conversion is shown below.

```
def ruleToBin(pktRule):
    """
        Takes the entire rule and then
        convert the rule into a binary
        format given below
        [src-ip] = 00011100
    """

    ruleBin = {}
    for key, value in pktRule.iteritems():

        if int(key == "s_ip") | int(key == "d_ip"):
            tmp = value.split('.')
            ruleBin[key] = ""
            for x in tmp:
                ruleBin[key] += padding(((bin(int(x))).replace('b', '')), 8)

        elif int(key == "s_p") | int(key == "d_p"):
            if len(value) > 0:
                ruleBin[key] = (padding(((bin(int(value))).replace('b', '')), 16))
            else:
                print "\tEither the Source or Destination Port is not given, skipping
this rule ..."
                return False

        elif key == "p":
            if (value.isdigit()):
                ruleBin[key] = padding(((bin(int(value))).replace('b', '')[::-1]), 8)
            else:
                ruleBin[key] = padding(((bin(protocol[value])).replace('b', '')[::-1]), 8)

    return ruleBin
```

A. Binary to Formula Format

This is an important stage in the SSBDD generation process, once the rules are converted into its binary equivalent the next stage is to convert the rule into a formula format. The formula format shows what does each and every node in the BDD contains. Below is the algorithm that shows how the binary equivalent rules are converted into the formula format.

```
def dyn_ruleFormula(ruleBin):

    # Convert the Binary rule format into a formula

    s_ip_len = 33
    d_ip_len = 33
    s_p_len = 17
    d_p_len = 17
    p_len = 9

    s_ip_tracker = 1
    d_ip_tracker = 1
    s_p_tracker = 1
    d_p_tracker = 1
    p_tracker = 1

    tmp_s_ip = {}
    tmp_d_ip = {}
    tmp_s_p = {}
    tmp_d_p = {}
    tmp_p = {}

    for j in ruleBin['s_ip']:
        if s_ip_len != s_ip_tracker:
            tmp_s_ip['s_ip'+str(s_ip_tracker)] = j
            s_ip_tracker += 1

    for k in ruleBin['d_ip']:
        if d_ip_len != d_ip_tracker:
            tmp_d_ip['d_ip'+str(d_ip_tracker)] = k
            d_ip_tracker += 1

    .....
```

```

for m in ruleBin['d_p']:
    if d_p_len != d_p_tracker:
        tmp_d_p['d_p'+str(d_p_tracker)] = m
        d_p_tracker += 1

for n in ruleBin['p']:
    if p_len != p_tracker:
        tmp_p['p'+str(p_tracker)] = n
        p_tracker += 1

s_ip_tracker = 1
d_ip_tracker = 1
s_p_tracker = 1
d_p_tracker = 1
p_tracker = 1

ruleFormula = collections.OrderedDict()

for tmp in dyn_order_var:
    tmp_field = tmp['og']
    if tmp['field_name'] == 's_ip':
        ruleFormula[tmp_field] = tmp_s_ip[tmp_field]
    elif tmp['field_name'] == 'd_ip':
        ruleFormula[tmp_field] = tmp_d_ip[tmp_field]
    elif tmp['field_name'] == 's_p':
        ruleFormula[tmp_field] = tmp_s_p[tmp_field]
    elif tmp['field_name'] == 'd_p':
        ruleFormula[tmp_field] = tmp_d_p[tmp_field]
    elif tmp['field_name'] == 'p':
        ruleFormula[tmp_field] = tmp_p[tmp_field]

return ruleFormula

```

5.1.3 SSBDD Generation

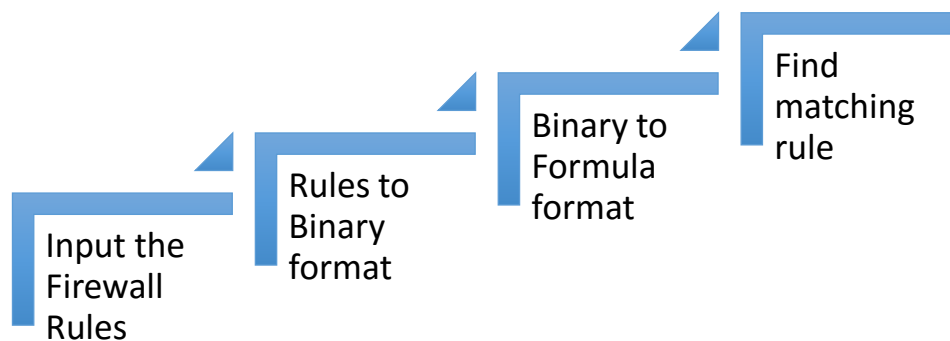
The final phase of the SSBDD generation is simple writing the formula generated above into a formula file. This formula file is then used to generate the graphical BDD, which is disabled in this stage as it increases the CPU processing time. Below is an example of what exactly does the formula file contains.

Let $p = \text{Protocol} = \text{TCP}$
 $= (p_1 \ \& \ p_2 \ \& \ p_3 \ \& \ p_4 \ \& \ p_5 \ \& \ \sim p_6 \ \& \ \sim p_7 \ \& \ p_8)$

The above example shows how the binary equivalent of the TCP protocol will be written in the formula file in the end.

5.2 SSBDD Packet Filtering

Packet filtering is the next stage of SSBDD, at this stage the packet is received by the packet filter which is then used to traverse the SSBDD to determine whether or not to accept or reject the packet. The step-by-step flow diagram is given below; it is same as the one described in the SSBDD generation phase. The only difference is in the last stage, instead of generating the BDD it will traverse the BDD. The final stage is merely a comparison stage where each node value is compared to see if it matches or not. The earlier stages of the SSBDD Packet Filter has been explained before.



Below is the implementation of SSBDD traversal that traverses the SSBDD tree once it receives a packet.

```

def pktMatcher(pktBin, ruleBDD):

    global nextNode
    global matchHit
    global tmpRuleBDD
    tmpRuleBDD = ruleBDD

    tmp_h_table = {}

    for data in ruleBDD['h_table'].items():
        tmp_h_table[data[1]] = data[0]

    for pktData in pktBin.items():
        if str(nextNode) == "1":
            log_data = "\tPacket has been accepted now ... and the hit count is "
            log_data += str(matchHit) + "\n"
            writeLog(log_data)
            matchHit = 0
            return

        elif str(nextNode) != "-1":
            if (db._get_var_name(ruleBDD, nextNode)) == pktData[0]:

                if str(pktData[1]) == "0":
                    indice = int(pktData[1])+1
                elif str(pktData[1]) == "1":
                    indice = 2

                if not is_matching(tmp_h_table[nextNode][indice]):
                    return
            else:
                nextNode = nextNode - 1
                while True:
                    if (db._get_var_name(ruleBDD, nextNode)) == pktData[0]:
                        break;
                    else:
                        nextNode = nextNode - 1

                if str(pktData[1]) == "0":
                    indice = int(pktData[1])+1
                elif str(pktData[1]) == "1":
                    indice = 2

```

```

if not is_matching(tmp_h_table[nextNode][indice]):
    return

elif str(nextNode) == "-1":
    nextNode = len(tmp_h_table)+1
    if (db._get_var_name(ruleBDD, nextNode)) == pktData[0]:

        if str(pktData[1]) == "0":
            indice = int(pktData[1])+1
        elif str(pktData[1]) == "1":
            indice = 2

        if not is_matching(tmp_h_table[nextNode][indice]):
            return
    else:
        nextNode = nextNode - 1
        while True:
            if (db._get_var_name(ruleBDD, nextNode)) == pktData[0]:
                break;
            else:
                nextNode = nextNode - 1
        if str(pktData[1]) == "0":
            indice = int(pktData[1])+1
        elif str(pktData[1]) == "1":
            indice = 2
        if not is_matching(tmp_h_table[nextNode][indice]):
            return

```

5.3 BDD Packet Filter vs SSBDD Packet Filter

The explanation of SSBDD generation and traversal is done in the previous section in detail. To see the practical approach of SSBDD, a single rule is provided to both the BDD's – BDD packet filter and SSBDD packet filter. This provides a clear overview of how the packet filter works at the filtering level. Since the BDD requires the rule to be converted into binary format so the binary equivalent of the rule is written instead of its decimal format. The Table 6 contains the Binary equivalent of the rule.

Protocol	0000 0110
Source IP	11000000 10101000 00100001 11110111
Destination IP	11000000 10101000 00100000 01111101
Source Port	00000100 00011010
Destination Port	00001101 00111101

Table 6: Binary equivalent of a rule

5.3.1 BDD Packet Filter

In a regular BDD packet filter, the traversal is done by going from one field to the another field like a regular firewall. Below is a single packet converted into its binary equivalent. Once converted it starts from the first field and takes the entire field and starts the traversal. In a regular BDD packet filter, the field order is fixed so they cannot be changed. This is the limitation of this BDD because several traffic can be rejected earlier or may be some bits are not needed to be checked in order to get the decision.

Protocol	0000 0110
Source IP	11000000 10101000 00100001 11110111
Destination IP	11000000 10101000 00100000 01111101
Source Port	00000100 00011010
Destination Port	00001101 00111101

Figure 7: BDD Packet Filter

5.3.2 SSBDD Packet Filter

In SSBDD filter since the field order can be changed based on the traffic analysis which shows what fields are more important than the other. Another important factor which was discussed in detail in Chapter Split Size, deciding how many bits to check from each field before moving on to the another field.

Since at the beginning there's no traffic log to analyze so SSBDD will go with a default option of setting the property to the following values

<field order = protocol, source ip, destination ip, source port, destination port>
<split size = 8>

Protocol	0000 0110	2	3	4
Source IP	11000000	10101000	00100001	11110111
Destination IP	11000000	10101000	00100000	01111101
Source Port	00000100	00011010		
Destination Port	00001101	00111101		

Figure 8: SSBDD Packet Filter

In Figure 8 it is visible that instead of checking the entire field at a time, SSBDD takes 8 bits from each field and then goes to another field. The advantage here over the BDD packet filter is the necessary bits are not checked at the first stage instead they are delayed for the next stage. This gives a performance improvement to SSBDD which can be very useful when there's a high amount of rejection traffic.

An interesting question about the BDD approach is with regard to how robust the BDDs are to the access list rule structure. Techniques such as RFC, cross-production

and the tuple space search rely heavily on structured access lists for good performance – some even become unusable in extreme conditions. This question is tackled in Chapter 6 once the experimental evidence has been presented.

Chapter 6: Analytical Discussion of the SSBDD

The previous chapter provides a framework for the construction of SSBDD, representations of access lists and using them to perform lookup in packet filters. This chapter presents an analytical discussion into SSBDD approach to packet filtering. The main reason behind this chapter is to discuss the computing aspects of the SSBDD packet filter and get an overview of the lookup time.

The discussion begins into what SSBDD representations of access lists look like and what determines the structure of these SSBDDs, since the shape and size of a SSBDD directly affects its time and space performance. Then the discussion goes into the bounds for, as well as factors affecting, the lookup time of the BDD packet filter, while the next section discusses factors affecting memory usage.

6.1 SSBDD Representation of Access List

The SSBDD representation of an access list is a regular BDD and it describes at a lowest level of what packets are accepted or rejected by the packet filter. SSBDD contains two nodes 1 or 0 which are the terminal nodes, where 1 means the packet is Accepted and 0 means the packet is Rejected. Every node in the SSBDD refers to a particular bit in the packet header. Reaching to the terminal node 0 – rejection node – has more than one path and the same applies to the acceptance traffic as well, so these complete set of paths for rejection and acceptance is the entire search space of the SSBDD.

To understand the concept of the access list in SSBDD let's consider a demo access list shown below in Table 7. This list accepts only TCP and UDP type traffic and from the certain IP address as mentioned in the access list itself. Although this is a somewhat an unlikely access list in a real world scenario, it does share some properties with real access lists which is important to understand the sections to be

discussed ahead. This access list is going to be used in the rest of this chapter to explain the sections ahead.

Rule #	Source Addr	Destination Addr	Source Port	Destination Port	Protocol
1	192.168.1.10	10.23.12.43.2	9000	80	TCP
2	172.16.2.12	80.75.45.3.12	10234	80	TCP
3	192.168.3.10	98.3.12.5	8769	878	UDP

Table 7: Sample Access List

There are many factors that can affect the structure of the SSBDD, not all the factors can be controlled but there are few factors which can be controlled. Those factors were discussed in detailed in the Chapter Packet filter optimization and Chapter Split size. Field ordering is one factor that can improve the performance of the firewall's packet filtering. If the field order is chosen correctly, then it can improve the performance of the packet filter. In a regular BDD based packet filter the field order is fixed and cannot be changed so the factor of field ordering is not considered. The shape and size of a BDD representing an access list are affected by various factors such as the number of nodes, number of variables and BDD depth, which in turn affects the performance of the SSBDD packet filter.

Of the factors affecting the structure of the BDD, some can be controlled and others cannot. The two main factors involved are the access list itself and the variable ordering of the BDD. The variable ordering of the BDD can be manipulated to improve the performance of the packet filter, whereas the access list is generally fixed. Variable orderings can also often be chosen to take advantage of the rule structure in the access list. The next two sections discuss how these factors affect the structure of the BDD.

6.1.1 The Effect of Rule Structure

SSBDD consist of number of variables and these variables are reference to the packet header bits of the access list. This illustrates two points. Firstly, during the SSBDD construction the variables that are not needed are automatically excluded in the construction process. This optimization is very useful as it can affect the performance of the SSBDD overall. Second point, the number of variable in SSBDD is completely dependent on the complexity of the access list. The more specific the rule is the more variables an SSBDD will need.

Common values in the field is also another factor that can affect the SSBDD of the access list. Rule that share common values with another rule can often help the SSBDD to generate less variable. For example, Rule # 1 and Rule # 2 shares the same destination port i.e. 80, so during the SSBDD generation time both the rule will share the variables but will have different exit points. This aggregation of similar values greatly affects the size of the BDD, since the degree to which expression in the BDD can be shared determines the number of nodes required for the BDD. In a simple statement, the more data the rules share the less variables the SSBDD will need.

6.1.2 The Effect of Variable Ordering

The variable ordering chosen for the example has the source address first followed by the source port then destination address then destination port and then protocol at the end. During the lookup the source address will always be checked first but what if the packet that is coming is not a TCP or a UDP protocol. Even though if the addresses the packet will still have to go through the first field and then get rejected at the end. If a different variable ordering is used – for example having the protocol

field appearing as the first testing for every packet, then the packet filter may be able to reject packets early. Since the protocol has the smallest field size so less number of bits will be checked.

Changing the field ordering does not change the semantics of the SSBDD for any given access list as they are semantically equivalent. The only change in the SSBDD will be its structure. The number of nodes, the path lengths, and the order in which the variables appear from root node will be different.

Different field ordering will have a different affect on the overall SSBDD structure. It is very common to see that on ordering can reduce the path lengths or vice versa. The memory usage of SSBDD is completely dependent on how long is the SSBDD. The bigger the SSBDD the more memory it will take which will in turn take more time to traverse the SSBDD. So it is safe to make this assumption that different variable ordering creates different space-time tradeoffs.

6.2 Performing Lookup on a SSBDD

The SSBDD algorithm starts from the first node – root node, checks the value and follows the appropriate edge of the node. This process will continue until the the terminal nodes is reached. If the the terminal node is 0 then the packet is rejected or accepted if the terminal node is 1. So, the time taken by the SSBDD to reach a specific decision is equivalent to the length of the path traversal the SSBDD follows.

6.2.1 Worst Case Analysis

The worst case in SSBDD occurs only when the SSBDD traverses the longest path to reach the terminal node. In other words, it can be assumed that, the worst case occurs when the SSBDD traverses most of the nodes in the SSBDD.

- The upper bound of the SSBDD structure defines how many nodes the SSBDD will traverse in its worst case scenario. Since the variable needed to create the SSBDD is based on the rule structure so the worst case scenario of the SSBDD will never exceed the maximum number of nodes used in the SSBDD.
- In the worst case scenario, the SSBDD is meant to traverse majority of the nodes. But to represent any given rule in SSBDD it needs n variable. So the SSBDD will never exceed more than n variable to reach a decision. The lower bound of the SSBDD will always be either exactly the n variable needed to represent the SSBDD or less than the n variable.

6.2.2 Best Case Analysis

The best case of the lookup algorithm depends on the variable ordering of the SSBDD, so different SSBDD will lead into a different result. Best case in SSBDD occurs only when the SSBDD choses the shortest path from the tree to reach a terminal node. In majority of the SSBDD the best case can occur during the matching of the protocol field since it requires maximum of 8 nodes.

It also worth noticing that the best case of SSBDD can also be reduced from 8 nodes to 4 nodes comparison. For example, if taking the protocol field, it requires 8 bits to represent in the SSBDD but the maximum number of protocol used in Internet traffic is either TCP, UDP, or ICMP, which can be represented easily on SSBDD from the first 4 bits. So the best case of the SSBDD can also be 4 nodes if the SSBDD considers checking the first 4 bits first instead of checking all the 8 bits.

The best case scenario has little value because in real world it is unlikely to occur in practice. If the firewall is receiving more accepted traffic, then the rejection traffic so

the best case scenario will not occur as the best case scenario usually means the packet is getting rejected.

6.3 Memory Usage of SSBDD

The memory usage of SSBDD is based on the number of nodes that are needed to represent the access list. As shown in the Section 1 (a) and 1 (b) discusses the factors that can affect the overall structure of the SSBDD. The same factors are discussed in detail as a separate chapter – Split Size and Field Ordering.

In real world, the access list usually has lots of similarity between rules, this is due to the fact that the firewall is filtering between multiple networks internally. So with this assumption, it can be said, that the possibility of rules sharing nodes in SSBDD is very high which in the end can reduce the overall count of the nodes in the SSBDD. For example, if the source address in the Rule #1 and Rule #3 shares the same first two bytes which means the nodes will be shared for the first byte in the SSBDD. These little factors can greatly contribute to the structure of the access list in the SSBDD.

Predicting the number of nodes that the SSBDD will take is not possible as it is shown earlier that it can vary based on the field ordering. But having the knowledge of the access list can help to understand the right field order for the SSBDD in order to create a better and compact structure.

Chapter 7: Architecture and Simulation

Experimental evaluation is the predominant technique used in this research for evaluating the SSBDD packet filter and fulfilling the research objectives. Evaluation is achieved by comparing the SSBDD packet filter to a packet filter that evaluates its rules sequentially. Two sequential packet filters are used for comparison and details regarding these are given later.

In order to achieve meaningful and generalizable results, it is important for the experimental methodology to provide an experimental environment that is as realistic as possible. This chapter is devoted to discussing how this is achieved, starting with the overall experimental setup and then investigating its components separately.

First the implementation part of the system is discussed in detail with all of its module involved in it. This is followed by an explanation of the simulation environment, which discusses the technical specs of the system used to evaluate the performance of both the BDDs. The next section discusses the Simulation Data that is used by the system to evaluate the performance of the BDDs. At end the Timing section will discuss how the performance is evaluated for the BDDs.

7.1 System Implementation

This section discusses the specifications of the packet filters in this research, as well as some important factors affecting their design and implementation. It then presents the implementations of the SSBDD and list-based or field-based packet filters.

7.1.1 Packet Filter Specification

The key requirement of the proposed packet filters used in this research is that they be *stateless*, meaning that the decision of whether to accept or reject a packet is based on each packet individually, independently from what happened in the past.

This forces each filter to invoke its lookup algorithm for every packet. The proposed packet filters are not required to handle fragmented packets since fragmentation can be resolved with caching, which is not the part of the research focus.

Furthermore, each packet filter must allow filtering of IP packets on the following fields:

- Source and destination address(es): Single IP addresses are accepted the current research work does not support the use of masking in the IP address.
- Source and destination port numbers: Single numbers only are accepted.
- Protocol type: This refers to the transport protocol type. Accepted values are TCP, and UDP

If any field is omitted from a rule, it is skipped from and the packet filter moves to the next packet. For example, if the protocol field is ICMP then the packet is not considered since there won't be any source and destination port number given for it. Finally, each packet filter must support the two actions PERMIT and DENY.

7.2 Coding and Modules

The programming language used to implement the SSBDD packet filter is Python. The code to implement the regular Binary Decision Diagram is already implemented in [1]. The SSBDD's coding work is basically an extension to the Tyler's [1] work. The regular BDD implementation shown in [1] is based on the research work conducted in [2].

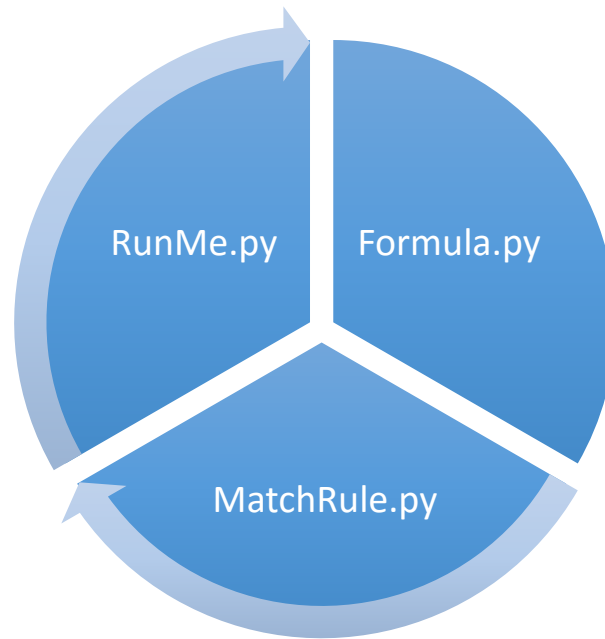


Figure 9: Modules of SSBDD implementation

Figure 9 shows the name of the 3 important modules for the SSBDD implementation. The above are not the only modules of the system, there are many more modules used behind the system but explaining that is not necessary. The main working of these implementation are as follows:

7.2.1 RunMe.py

This is the main module of SSBDD. The simulation starts by executing this file which then calls the rest of the other files. In order, to execute the module there are certain prerequisite that should be met.

The following are the pre-requisite that is needed to execute the module RunMe.py

- The config.py file must be present, this file defines the two parameters that were discussed in the earlier chapters – field ordering and split size.

- CSV file that contains all the rules needed for the given simulation. The rules must be in CSV format and the structure of the rules should follow the structure give below:

<ip.proto,ip.len,ip.src,ip.dst,tcp.srcport,tcp.dstport,tcp.flags,udp.srcport,udp.dstport,icmp.type,icmp.code>

- CSV file that contains all the packets needed for the simulation. The packets must be in CSV format again and also the structure must follow the same structure as mentioned for the rule file.

This module at the end will provide the following outcome.

- Calling the other necessary modules to generate the BDD and SSBDD.
- Performing the packet filter.
- Displaying the CPU Performance Time for both the BDDs.
- Displaying the count of Accepted, Rejected, and Total Traffic received by the packet filter.

7.2.2 Formula.py

In order to generate a BDD, the rules first must be converted into a Binary equivalent and then the binary formatted rule is then converted into a Boolean expression or a formula. All that conversion process is taken care by this module.

This module is not called independently, instead it is called by the RunMe.py module, and it only asks for a file name that has the rules in it. The following are the task that this module will perform in order to generate a single binary expression or a formula which is then written into a file.

- Reading the CSV rule file line-by-line.
- Converting the rule into a binary format.

- Converting the binary formatted rule into a formula or boolean expression.
- Writing the generated formula into a file.

Once this module is executed, the program will generate a single binary expression for the entire rule set given to it. And also, if the user asks, a graphical BDD and SSBDD. The option of generating the graphical BDD and SSBDD is usually disabled as it consumes lot of CPU Processing Time which is not necessary.

7.2.3 MatchRule.py

This is the final module and it is called when the BDD and SSBDD have been generated for the provided rule set. It is not executed independently; it is called by the RunMe.py module. The following are the pre-requisite of this module: -

- Needs the reference to the generated BDD and SSBDD for the given rule set.
- CSV file that contains all the packets needed for the simulation. The packets must be in CSV format again and also the structure must follow the same structure as mentioned for the rule file.

The outcome of this module is to go through each packet one at a time and then perform a matching by traversing the BDD. At the end, the result of accepted and rejected traffic is given to the main module.

7.3 Other coding modules

The above were the only coding modules that were implemented from the scratch.

There are other modules too that are called in the background in to generate the BDD and SSBDD. The packet filtering or traversing BDD and SSBDD module is a part of MatchRule.py which is explained in the previous section.

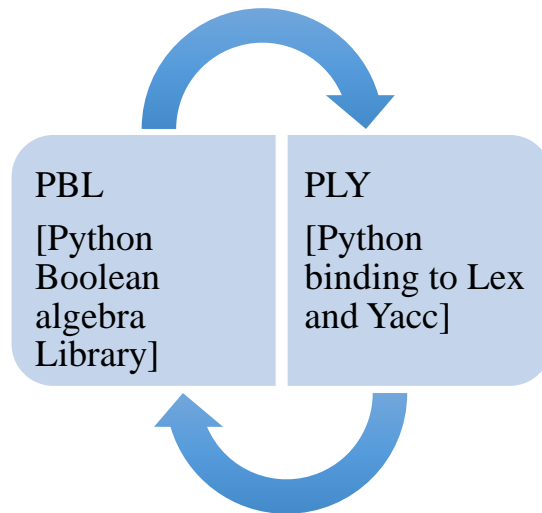


Figure 10: Other Coding Modules.

The above are the rest of the two modules which are implemented by Tyler's, the code is open-source and it can be downloaded from (Tyler). The two modules are dependent on each other and with these two modules none of the above modules would run.

7.4 Simulation Environment

In order to simulate real packet filtering scenarios, an Amazon Web Server was used to run the system. Running the system on a regular machine takes days to execute even with a lowest traffic volume. The simulation environment uses just one server and the same server traverses the rules file first and converts it into a BDD and SSBDD. Then the packet file is read and it is passed against the rule.

To evaluate the performance of the system, CPU Processing Time is considered and it is started right when the rule is read by the system and stops when all the packets have been parsed by it. The specification of the server on the AWS are as follows: -

Processor Type	Intel Xeon E5-2666 v3 (Haswell processor)
Processor Speed	2.9 GHz
CPU Count	2
RAM	3.75 GiB
Hard Disk Drive	40 GB SSD

Table 8: Specification

7.5 Simulation Dataset

In this section we describe the data used in the experimental study. The data set used in the experimental study is obtained from a CAIDA. The data provided by them consist of around 17 million packet header information.

The demo of the packet header is shown below. Not all the data mentioned in the dataset is used. For e.g. for the simulation purpose of our system only source ip, destination ip, source port, destination port, and protocol field is used rest all the other fields are skipped.

<ip.proto,ip.len,ip.src,ip.dst,tcp.srcport,tcp.dstport,tcp.flags,udp.srcport,udp.dstport,icmp.type,icmp.code>

7.6 Simulation Framework

The framework used for the simulation purpose act like any regular firewall and requires a set of rules and packets to process. The dataset that is been used for the evaluation purpose does not contain any firewall rules. So randomly packets were chosen from dataset and were used as a firewall rule. By default, all the firewall rule's action property was set to Allow. Since the packets whose rule exist in the firewall and whose Actions is either Allow or Deny will traverse the entire BDD. But the main focus of this research work is for early packet rejections.

The simulation performed on the system were different each time with a different set of firewall rules, different amount of traffic, and mostly different set of acceptance vs rejection ratio. Since, the research work is mostly focused on early rejection, so the amount of traffic that were passed against the firewall rules had a high rejection ratio.

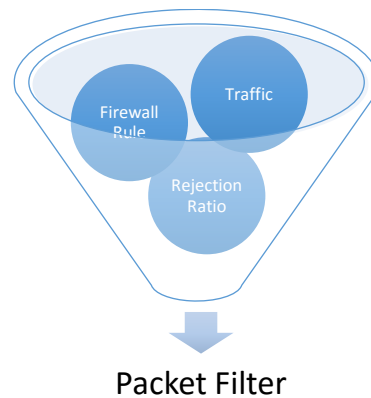


Figure 11: Simulation Framework Overview

Figure 11 gives an overview of what does the simulation framework contains. For every simulation the three parameters as shown in the Figure 3 were changed. The simulation was performed on both the BDD's regular BDD and SSBDD.

Chapter 8: Statistics

This chapter covers the result of the experimental evaluation of the SSBDD packet filter. In the experimental evaluation, the efficacy of the SSBDD approach to packet filtering is evaluated in terms of lookup. For the experiments performed, two sets of data were collected – Linear BDD and Static Shuffling BDD. The CPU Timing functionality was implemented by inserting the code to keep track of the overall execution time.

In each and every experiment different set of Acceptance and Rejection Ratio were used, to see the performance of the SSBDD during different rejection ratio. This provides a better randomness in terms of simulation and at the same time will provide a better result to verify the performance of BDD vs SSBDD. Each experiment has 4 sets of simulation in them, where each simulation is run twice – once on BDD and another one on SSBDD. The statistics that are shown in the graphs below are based on the CPU timing.

8.1 Simulation Results

This section shows the performance of the various simulations that was performed on the BDD based packet filter and SSBDD based packet filter. Each simulation has been performed 4 times for BDD and SSBDD based packet filter, where each simulation has varying sets of traffic passed to it with a varying set of firewall rules for it.

8.1.1 Experiment A

Acceptance Ratio = 10% & Rejection Ratio = 90%

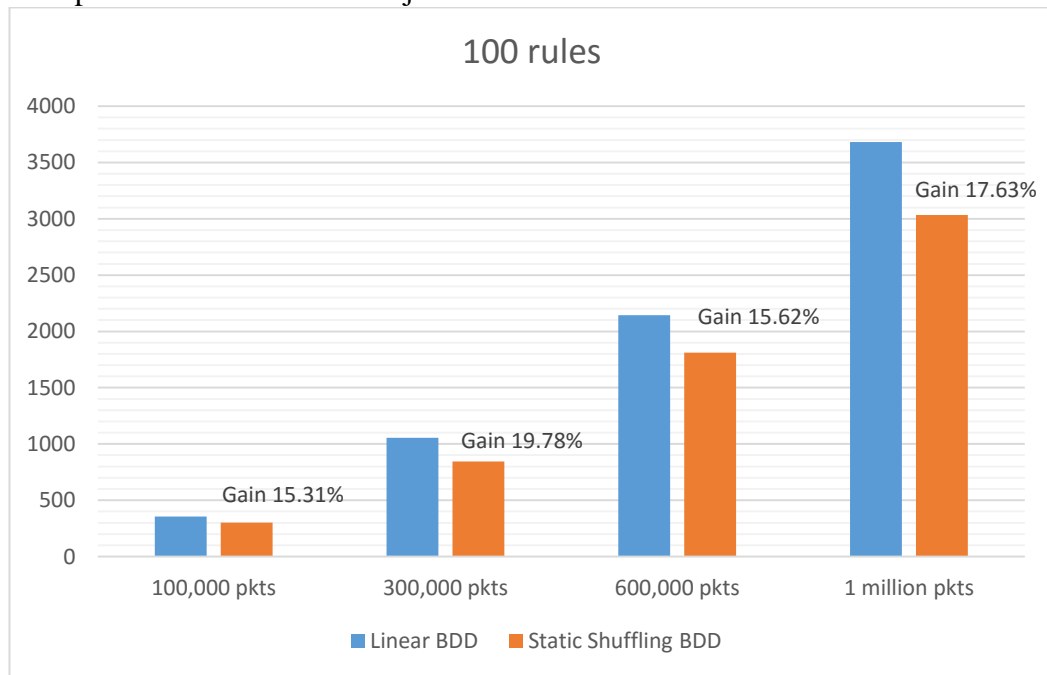


Figure 12: Experiment A Result

8.1.2 Experiment B

Acceptance Ratio = 5% & Rejection Ratio = 95%



Figure 13: Experiment B Result

8.1.3 Experiment C

Acceptance Ratio = 95% & Rejection Ratio = 5%



Figure 14: Experiment C Result

8.1.4 Experiment D

Acceptance Ratio = 50% & Rejection Ratio = 50%



Figure 15: Experiment D Result

8.1.5 Experiment E

Acceptance Ratio = 70% & Rejection Ratio = 30%

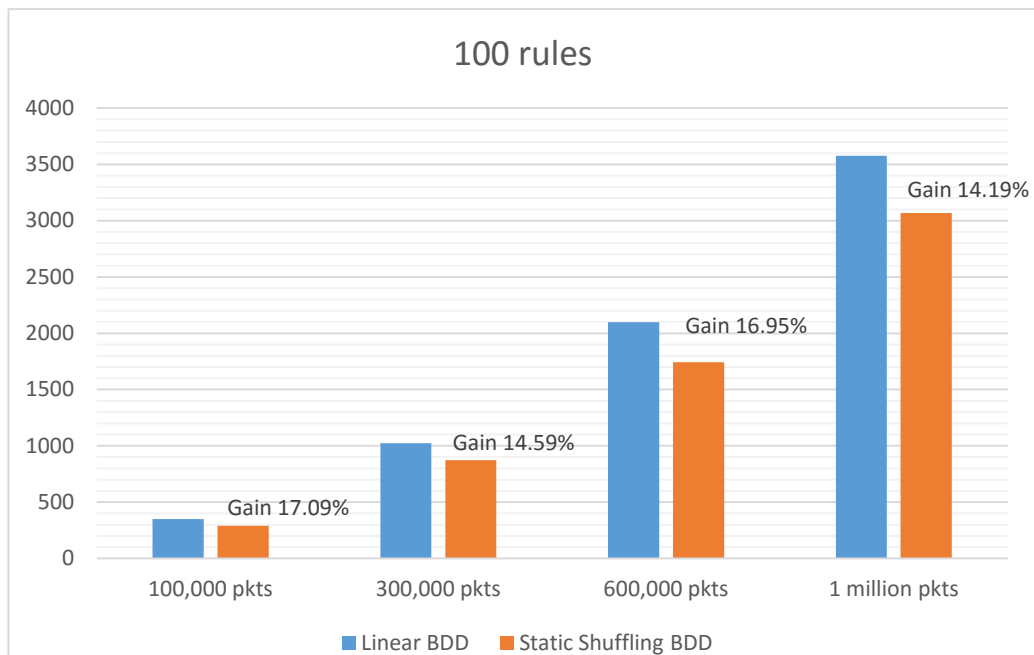


Figure 16: Experiment E Result

Chapter 9: Conclusion and Future Work

The contribution of this research was to find an efficient way to represent the access filters of the packet filtering firewall. Packet filtering is the mechanism that is being implemented in every hardware or software firewall. The issue with a traditional packet filtering firewall, they perform rule matching sequentially. So, the latency issued by this lookup process is equal to the size of the list.

The representation technique used in this research work is based on BDD. This follows from the fact that BDDs is capable of providing a compact representation for complex Boolean functions. The aim of this research was for two fold, in the first fold, a new method was proposed to provide an efficient way to represent the access list. The second fold was to evaluate the performance of the BDD in terms of their lookup and also memory requirements.

9.1 Future Work

This section discusses the improvement areas of the SSBDD that can improve the performance of it. These ideas, as well as others, are discussed in this section in detail. Some ideas require more extensive research, while others are simple enough to be implemented easily.

9.1.1 Variable Ordering and Reordering Prediction

Variable ordering is the major factor for performance improvement. Currently in this research, SSBDD analyzes the access list to come up with a better variable ordering at first, since there's no traffic to analyze. Later on after several million packets the SSBDD then uses the traffic characteristics to choose an optimal variable ordering. Choosing good variable orderings for lookup is most effective when traffic is taken

into consideration, so an algorithm that can continuously monitor the traffic and update the variable ordering instead of checking after every several million packets would maximize the potential of SSBDD packet filter.

9.1.2 Updating the SSBDD

Any change in the split size or variable ordering or in the access list requires the SSBDD to be regenerated again. SSBDD regeneration is a time consuming process, but if the SSBDD is able to update itself incrementally then it can save a lot of CPU processing time and improve the performance overall.

9.1.3 Considering more Parameters for Performance

As the scope of the thesis, CPU time was considered as a performance factor. But more factors can be considered, such as – memory utilization, CPU utilization etc. CPU utilization can give a better overview of the proposed method, in terms of how much load it is putting on the CPU.

9.2 Conclusion

The aim of this research was to propose a new method for the purpose of representing the access list of the firewall. This aim was achieved in two ways. Firstly, the proposed method SSBDD was discussed in detail. Secondly, the simulation was performed to prove that SSBDD performs efficiently. The advantages of this approach extend beyond performance as it helps to understand other problems with traditional packet filter.

References

- Cheswick, W. R. (2003). *Firewalls and Internet Security: Repelling the Wily Hacker* (2 ed.). Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.
- Nikolaidis, I. (2000). Firewalls: a complete guide. *IEEE Network* , 14 (4), 6-6.
- Ballew, S. M. (1997). *Managing IP Networks With Cisco Routers* (1st ed.). Sebastopol, CA, USA: O'Reilly & Associates, Inc.
- Oppliger, R. (1998). *Internet and Intranet Security*. Norwood, MA, USA: Artech House, Inc.
- Srinivasan, V. a. (1999). Packet Classification Using Tuple Space Search. *ACM* , 135-146.
- Gupta, P. a. (2001). Algorithms for Packet Classification. *Netwrk. Mag. of Global Internetwkg.* , 24-32.
- Trabelsi, M. M. (2014, March). A data driven firewall for faster packet filtering. *Communications and Networking (ComNet), 2014 International Conference o* , pp. 1-5.
- Boutaba, A. E.-A.-S. (2009, April). Adaptive Early Packet Filtering for Defending Firewalls Against DoS Attacks. pp. 2437-2445.
- Zeidan, Z. T. (2012, June). Multilevel early packet filtering technique based on traffic statistics and splay trees for firewall performance improvement. pp. 1074-1078.
- Hazelhurst, S. a. (1998). Binary decision diagram representations of firewall and router access lists. *Department of Computer Science, University of the Witwatersrand, Tech. Rep* .
- Bryant, R. E. (1992). Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys (CSUR)* , 293--318.
- diagrams, A. i. (1997). Andersen, Henrik Reif. *Lecture notes, available online, IT University of Copenhagen* .
- Andersen, H. R. (1997). An introduction to binary decision diagrams. *Lecture notes, available online, IT University of Copenhagen* .
- Bollig, B. a. (1996). Improving the variable ordering of OBDDs is NP-complete. *Computers, IEEE Transactions* , 993-1002.
- Hamed, E. S.-S. (2004). Discovery of policy anomalies in distributed firewalls. pp. 2605-2616.
- Al-Shaer, H. H.-A. (2006). Adaptive Statistical Optimization Techniques for Firewall Packet Filtering. pp. 1-12.

CADA. (n.d.). Retrieved March 25, 2016, from www.caida.org

Acharya, S. a. (2006). Simulation Study of Firewalls to Aid Improved Performance. (pp. 18--26). IEEE Computer Society.

Greenberg, S. A. (2006). Traffic-Aware Firewall Optimization Strategies. *Communications, 2006. ICC '06. IEEE International Conference on*, (pp. 2225-2230).

Hamed, E. S.-S. (2004). Modeling and Management of Firewall Policies. *IEEE Transactions on Network and Service Management* , 2-10.

Tyler. (n.d.). Retrieved March 27, 2016, from <https://github.com/tyler-utah>

Ben-Neji, Nizar, and Adel Bouhoula. "Dynamic scheme for packet classification using splay trees." *Proceedings of the International Workshop on Computational Intelligence in Security for Information Systems CISIS'08*. Springer Berlin Heidelberg, 2009.

Cherian, Mimi Mariam, and Madhumita Chatterjee. "Firewall Optimization with Traffic Awareness Using Binary Decision Diagram."

Cherian, Mimi, and Madhumita Chatterjee. "Optimized Firewall with Traffic Awareness."

Choudhari, Pragati M. "Efficient Packet Matching for Packet Filtering Firewall."

Khummanee, Suchart, and Kitt Tientanopajai. "The Policy Mapping Algorithm for High-speed Firewall Policy Verifying." *International Journal of Network Security* 18.3 (2016): 433-444.

Winter, Christian. "Firewall Best Practices." *Future Internet (FI) and Innovative Internet Technologies and Mobile Communications (IITM)* 1 (2016).

Hager, Sven, et al. "Minflate: Combining Rule Set Minimization with Jump-based Expansion for Fast Packet Classification." *Proceedings of the 2016 Symposium on Architectures for Networking and Communications Systems*. ACM, 2016.

Tongaonkar, Alok, and R. Sekar. "Condition Factorization: A Technique for Building Fast and Compact Packet Matching Automata." *IEEE Transactions on Information Forensics and Security* 11.3 (2016): 468-483.

Shaikh, Riaz Ahmed, Kamel Adi, and Luigi Logrippo. "A Data Classification Method for Inconsistency and Incompleteness Detection in Access Control Policy Sets." *International Journal of Information Security* (2016): 1-23.