ADVANCING CYBER SECURITY WITH A SEMANTIC PATH MERGER PACKET CLASSIFICATION ALGORITHM

A Thesis
Presented to
The Academic Faculty

by

J. Lane Thames

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
December 2012

ADVANCING CYBER SECURITY WITH A SEMANTIC PATH MERGER PACKET CLASSIFICATION ALGORITHM

Date Approved: September 27, 2012

To Linda, my wife and best friend, forever plus infinity

ACKNOWLEDGEMENTS

First and foremost, I give special thanks to my advisor, Dr. Randal (Randy) Abler, who has been a great friend and mentor. I began working with Dr. Abler while I was an undergraduate student at Georgia Tech. He introduced me to the world of computer networking, information technology, and cyber security, and I am indefinitely grateful for the support, guidance, and knowledge he has given to me throughout the years.

I would like to thank Dr. George Riley, Dr. Henry Owen, Dr. Raghupathy Sivakumar, and Dr. Dirk Schaefer for taking time from their busy schedules to serve as my thesis committee. Their feedback, support, and guidance played an invaluable role in my dissertation research. I am truly honored to have studied under the guidance of my advisor and committee.

I am grateful to all of my friends, classmates, and colleagues from Georgia Tech for all of their support and encouragement. I am thankful for each and every one of the wonderful Georgia Tech faculty and staff who have helped me during my time as a student. I would like to extend a special thanks to Ms. Marilou Mycko, Mrs. Pat Potter, and Dr. Daniela Staiculescu for all of their administrative support during my time as a graduate student. I would like to express my sincere gratitude to Dr. David Frost for being such a great friend and mentor to me. Special thanks goes to Gail Wells. Gail allowed me to work with her as an undergraduate teaching assistant. It was during my time as a teaching assistant that I discovered a burning desire to teach, and it was this desire that influenced my decision to pursue a graduate degree. Several years ago, Dr. Schaefer and my colleagues in the Systems Realization Laboratory invited me to do collaborative research with them. Our research together over the last few years has been very rewarding, and, for this, I am deeply grateful.

Finally and with immeasurable gratitude, I want to thank my wife, Linda, whose love and encouragement made this achievement possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

ACL	Access Control List.
ADF	Autonomic Distributed Firewall.
ADM	Autonomous Detection Module.
AEI	Audit Export Interface.
ANN	Artificial Neural Network.
BC	Boundary Controller.
BEF	Blocking Enforcement Filter.
BLN	Bayesian Learning Network.
BMP	Best Matching Prefix.
BMU	Best Matching Unit.
CAM	Content Addressable Memory.
CBT	Complete Binary Tree.
CIDR	Classless Interdomain Routing.
CIS	Computational Intelligence System.
CMS	Central Management Server.
CP	Cross Producting.
CPB	Colored Path Bijection.
CPE	Controlled Prefix Expansion.
CPHG	Colored Path Hypergraph.
DAG	Directed Acyclycic Graph.
DARPA	Defense Advanced Research Projects Agency.
DDoS	Distributed Denial of Service.
DFAR	Distributed Firewall and Active Response Architecture.
DIMI	Distributed Infrastructure Management Interface.
DM	Detection Modules.
DT	Destination Trie.
EDP	Energy Delay Product.

EFW	Embedded Firewall.
EGoT	Extended Grid of Tries.
ENN	Evolutionary Neural Networks.
EPO	Energy Per Operation.
FCF	Firewall Collaboration Framework.
FDAG	Filter Directed Acyclic Graph.
FDR	False Discovery Rate.
FMM	Firewall Management Module.
FN	False Negative.
FNR	False Negative Rate.
FP	False Positive.
FPR	False Positive Rate.
FSI	Filter Specification Interface.
GA	Genetic Algorithm.
GoT	Grid of Tries.
GPP	Global Preemptive Protection.
HBSOM	Hybrid Bayesian and Self-organizing Map Algorithm.
HPB	Highly Predictive Blacklisting.
IBS	Interval Binary Search.
IDS	Intrusion Detection System.
IF	Inbound Filters.
IP	Internet Protocol.
IPv4	Internet Protocol version 4.
IPv6	Internet Protocol version 6.
IRS	Intrusion Response System.
ISP	Internet Service Provider.
ITR	Induced Transitive Relations.
ITU-T	International Telecommunication Union.
KDD	Knowledge Discovery in Data.

LAN	Local Area Network.
LC	Level Compression.
LPM	Longest Prefix Match.
LSA	Local Security Audits.
LSP	Local Security Policy.
MBT	Multibit Trie.
MDT	Multidimensional Trie.
MVQ	Multivalence Vertex Query.
NAE	National Academy of Engineering.
NBLN	Naive Bayesian Learning Network.
NIC	Network Interface Card.
NMOS	N-type Metal Oxide Semiconductor.
NN	Neural Network.
NNO	Neural Network Oracle Algorithm.
NPR	Negative Prediction Rates.
OC	Optical Carrier.
OF	Outbound Filter.
PC	Packet Classification.
PDS	Policy Description Specification.
PDSI	Policy Description Specification Interface.
PM	Path Matrix.
PMDAG	Path Merged Directed Acyclic Graph.
PMM	Policy Management Module.
PMP	Path Merger Process.
PPR	Positive Prediction Rate.
RBP	Resilient Back Propagation.
RFC	Recursive Flow Classification.
RPN	Resilient Propagation Neural Network.
SAN	Semantic Association Network.

SAS	Semantic Association System.
SCADA	Supervisory Control and Data Acquisition.
SDS	Search Data Structure.
SOM	Self-organizing Map.
SPM	Semantic Path Merger.
SRAM	Static Random Access Memory.
SSH	Secure Shell.
SVM	Support Vector Machine.
TCAM	Ternary Content Addressable Memory.
TCP	Transmission Control Protocol.
TDA	Trusted Domain of Administration.
TDAD	Trusted Domain of Administration Delegate.
THA	TDA Host Architecture.
TN	True Negative.
TP	True Positive.
TPO	Time Per Operation.
TRM	Trust Relationship Management.
TTH	Trie to Hardware.
UBT	Unibit Trie.
UDP	User Datagram Protocol.
VLP	Vertex Labeled Path.
VST	Variable Stride Trie.
WMV	Weighted Majority Vote.

SUMMARY

Internet security continues to be a complex and challenging problem. Security mechanisms such as authentication, data integrity, and data confidentiality along with intrusion detection, intrusion prevention, and firewall systems have traditionally provided respectable levels of protection. However, malicious actors and their associated attack technologies have advanced significantly. Moreover, Internet-enabled platforms such as cyber-physical infrastructures, advanced mobile voice communication systems, the mobile Internet, voice-over-Internet Protocol (VoIP), cloud computing, the Internet-of-Things, vehicular networks, aerospace networks, intelligent transportation systems, and smart-home environments will provide countless new attack vectors and opportunities for malicious actors. Consequently, advanced security mechanisms and defense methodologies are needed for the continual protection of traditional Internet systems along with these emerging Internet-enabled platforms.

However, existing security systems and architectures have a foundation based on the concepts of topological perimeters, non-cooperative and isolated operation, reactive reconfiguration, and human advisory. Perimeter-based, isolated, non-cooperative, and administratively reactive techniques are becoming ineffective, and this is especially true for emergent platforms such as the mobile Internet and cloud computing in which the constituent computational and networking resources reside outside of enterprise topologies and perimeters. Isolated and non-cooperative security mechanisms operate within a *confined awareness domain*, and knowledge related to new security occurrences are not shared outside of the awareness domain. Isolated is implied by confinement, and non-cooperative is implied by not sharing knowledge with other awareness domains. This imposes a significant impediment in terms of efficient, global-scale Internet security. Lastly, administratively reactive relates to techniques whereby a human security administrator reacts to newly discovered security events instead of automated, computer-driven reactions. Human reaction and remediation times are far too slow to counter the impact of many classes of high-speed,

large-scale cyberattacks.

According to the National Academy of Engineering (NAE), securing the Internet and its associated cybersystems is one of the most complex engineering challenges ever faced by the engineering community. The NAE claims that cybersecurity cannot be achieved with traditional perimeter defenses, that new methods of authentication for hardware, software, data, and users are needed, that new approaches to the design of secure software should be considered, and that new methods are required for monitoring, detecting, and responding to cyberattacks. Consequently, the NAE and its committee on Engineering Grand Challenges has identified cyberspace security as one of fourteen Grand Challenges faced by engineers in the 21st century.

This dissertation provides novel algorithms, theories, and supporting frameworks to significantly improve the growing problem of Internet security. A premise forming a basis for the objectives and contributions of this dissertation is that distributed, collaborative, and autonomic defense methodologies are needed in order to achieve reliable Internet security because the sheer volume of modern day cyberattacks, their permutation capabilities, their sophistication, and their speed cause isolated, non-collaborative, administratively-reactive response and remediation methods to be ineffective. Consequently, the objectives of this dissertation research were (1) to study the characteristics needed for reliable distributed Internet security architectures and infrastructures, (2) to design a modular, adaptive, and integrable framework for distributed Internet security infrastructures, (3) to develop robust solutions addressing the fundamental packet classification problem, and (4) to develop new classification algorithms for advanced cyberattack detection systems.

As a result of these objectives, the contributions provided by this dissertation are the following:

- A distributed firewall and active response architecture.
- A firewall-based blacklist classification and enforcement model.
- A theory of semantic association systems.

- A semantic path merger packet classification algorithm and its hardware implementation.
- Advanced cyberattack detection algorithms based on computational intelligence systems.

The distributed firewall and active response architecture is a modular, adaptive, and integrable distributed security framework that enables cyber devices within an organization's cyber infrastructure to participate in the detection of and response to cyberattacks. The architecture is a supporting contribution that establishes the foundation upon which the core contributions of this dissertation are framed. Particularly, the distributed firewall and active response architecture requires (1) efficient packet classification algorithms to enable its novel firewall-based blacklist classification and enforcement model and (2) effective classification algorithms for cyberattack detection. While studying packet classification and cyberattack detection algorithms, the theory of semantic association systems was devised and formulated. The theory of semantic association systems was inspired by the emerging field of semantic computing. The theory defines a compositional model and a family of graph theoretic constructs supporting the notion of merged conceptualization. From this theory and the notion of merged conceptualization, the semantic path merger packet classification algorithm was derived along with its hardware-based implementation. The theory of semantic association systems, the semantic path merger algorithm, and the hardware implementation of the semantic path merger algorithm as a packet classification system are the core contributions of this dissertation. Finally, two novel cyberattack detection algorithms have been developed. The first detection algorithm is a hybridization of self-organizing maps and naive Bayesian learning networks. The second detection algorithm is a neural network ensemble coupled with a parametrically optimized neural network oracle that combines ensemble classification results. The oracle's network configuration is parametrically optimized with genetic algorithms.

CHAPTER I

INTRODUCTION

Internet security continues to be a complex and challenging problem. Security mechanisms such as authentication, data integrity, and data confidentiality services along with intrusion detection, intrusion prevention, and firewall systems have traditionally provided respectable levels of protection. However, malicious actors and their associated attack technologies have advanced significantly. Moreover, Internet-enabled platforms such as cyber-physical infrastructures [30, 32, 40, 35, 168], advanced mobile voice communication systems [29], the mobile Internet [109, 122], voice-over-Internet Protocol (VoIP) [22], cloud computing [34, 46, 126], the Internet-of-Things [44], vehicular networks [65, 78, 165], aerospace networks [91], intelligent transportation systems [75], and smart-home environments [103, 169] will provide countless new attack vectors and opportunities for malicious actors. Consequently, advanced security mechanisms and defense methodologies are needed for the continual protection of traditional Internet systems along with these emerging Internet-enabled platforms [24, 130, 23, 125, 163, 117, 160, 43].

In a recent study, Kim et al. [81] claim that modern day cyberattacks are increasingly sophisticated, coordinated, and financially motivated. A report by Symantec [143] reveals that attacks are increasingly targeting enterprises, are increasingly Web-based, and are increasingly driven by financial gains available within the Internet underground economy. Their report also reveals an increasing usage of openly available attack tools. According to the Institute for Security Technologies Studies at Dartmouth college [66], over ten thousand attack tools were freely available on the Internet in 2004. In a recent report provided by the Georgia Tech Information Security Center [47], over 100,000 new malware samples are discovered each day.

Designers of Internet security architectures suggest that security cannot be provided by monolithic perimeter-based processes within a computational or networking system and advocate the incorporation of security mechanisms into each primary element contributing to a computing and communication framework [110]. Moreover, utilization of highly coordinated and cooperating security mechanisms are considered to be one of the most effective ways to prevent large-scale, persistent, targeted attacks [129]. Further, the exchange of information describing the attributes of attacks and malicious actors have been shown to significantly enhance security defense mechanisms [124, 161]. Consequently, security architectures and infrastructures that support methodologies for automatically detecting and responding to cyberattacks are needed to enable effective defense-in-depth strategies and robust cybersecurity systems [9, 63, 72, 123, 132]. However, existing security systems and architectures have a foundation based on the concepts of topological perimeters, non-cooperative and isolated operation, reactive reconfiguration, and human advisory. Perimeter-based, isolated, non-cooperative, and administratively reactive techniques are becoming ineffective, and this is especially true for emergent platforms such as the mobile Internet and cloud computing in which the constituent computational and networking resources reside outside of enterprise topologies and perimeters. Isolated and non-cooperative security mechanisms operate within a *confined awareness domain*, and knowledge related to new security occurrences are not shared outside of the awareness domain. Isolated is implied by confinement, and non-cooperative is implied by not sharing knowledge with other awareness domains. This imposes a significant impediment in terms of efficient, global-scale Internet security. Lastly, administratively reactive relates to techniques whereby a human security administrator reacts to newly discovered security events instead of automated, computer-driven reactions. Human reaction and remediation times are far too slow to counter the impact of many classes of high-speed, large-scale cyberattacks.

According to the National Academy of Engineering (NAE), securing the Internet and its associated cybersystems is one of the most complex engineering challenges ever faced by the engineering community [106]. The NAE claims that cybersecurity cannot be achieved with traditional perimeter defenses, that new methods of authentication for hardware, software, data, and users are needed, that new approaches to the design of secure software should be considered, and that new methods are required for monitoring, detecting, and responding to cyberattacks [106].

This dissertation provides novel algorithms, theories, and supporting frameworks to significantly improve the growing problem of Internet security. A premise forming a basis for the objectives and contributions of this dissertation is that distributed, collaborative, and autonomic defense methodologies are needed in order to achieve reliable Internet security because the sheer volume of modern day cyberattacks, their permutation capabilities, their sophistication, and their speed cause isolated, non-collaborative, administratively-reactive response and remediation methods to be ineffective. Consequently, the objectives of this dissertation research were (1) to study the characteristics needed for reliable distributed Internet security architectures and infrastructures, (2) to design a modular, adaptive, and integrable framework for distributed Internet security infrastructures, (3) to develop robust solutions addressing the fundamental packet classification problem, and (4) to develop new classification algorithms for advanced cyberattack detection systems.

As a result of these objectives, the contributions provided by this dissertation are the following:

- A distributed firewall and active response architecture.
- A firewall-based blacklist classification and enforcement model.
- A theory of semantic association systems.
- A semantic path merger packet classification algorithm and its hardware implementation.
- Advanced cyberattack detection algorithms based on computational intelligence systems.

The distributed firewall and active response architecture is a modular, adaptive, and integrable distributed security framework that enables cyber devices within an organization's cyber infrastructure to participate in the detection of and response to cyberattacks. The architecture is a supporting contribution that establishes the foundation upon which the core contributions of this dissertation are framed. Particularly, the distributed firewall and active response architecture requires (1) efficient packet classification algorithms to enable its novel firewall-based blacklist classification and enforcement model and (2) effective classification algorithms for cyberattack detection. While studying packet classification and cyberattack detection algorithms, the theory of semantic association systems was devised and formulated. The theory of semantic association systems was inspired by the emerging field of semantic computing. The theory defines a compositional model and a family of graph theoretic constructs supporting the notion of merged conceptualization. From this theory and the notion of merged conceptualization, the semantic path merger packet classification algorithm was derived along with its hardware-based implementation. The theory of semantic association systems, the semantic path merger algorithm, and the hardware implementation of the semantic path merger algorithm as a packet classification system are the core contributions of this dissertation. Finally, two novel cyberattack detection algorithms have been developed. The first detection algorithm, which is referred to as HBSOM, is a hybridization of self-organizing maps and naive Bayesian learning networks. The second detection algorithm, which is called NNO, consists of a neural network ensemble coupled with a parametrically optimized neural network oracle that combines ensemble classification results. The oracle's network configuration is parametrically optimized with genetic algorithms.

The remainder of this dissertation is organized as follows. Chapter 2 provides an overview of the basic technological concepts underlying the primary research thrusts presented in this dissertation. Chapter 3 contains background material and discussion of related work in distributed Internet security architectures, packet classification algorithms, and cyberattack detection with classification algorithms implemented by computation intelligence systems. In chapter 4, the distributed firewall and active response architecture is presented. Chapter 5 introduces the novel theory of semantic association systems. In chapter 6, a new packet classification algorithm based on semantic path merger is introduced. Further, chapter 6 provides the implementation details of the semantic path merger algorithm as a hardware-based packet classification system. In Chapter 7, the HBSOM and NNO algorithms are introduced as new classification algorithms for the cyberattack detection problem. Finally, the dissertation concludes with chapter 8.

CHAPTER II

CONCEPTS

In this chapter, the basic concepts underlying firewalls, intrusion detection, intrusion prevention, and packet classification are introduced.

2.1 Terminology

The International Telecommunication Union (ITU) and its ITU-T X.1205 Overview of Cybersecurity [67] provides the following definition of cybersecurity:

> *"Cybersecurity is the collection of tools, policies, security concepts, security safeguards, guidelines, risk management approaches, actions, training, best practices, assurance and technologies that can be used to protect the cyber environment and organization and user's assets. Organization and user's assets include connected computing devices, personnel, infrastructure, applications, services, telecommunications systems, and the totality of transmitted and/or stored information in the cyber environment. Cybersecurity strives to ensure the attainment and maintenance of the security properties of the organization and user's assets against relevant security risks in the cyber environment. The general security objectives comprise the following: Availability, Integrity (which may include Authenticity and Non-repudiation), and Confidentiality"*

Three key concepts form the core of cybersecurity problems: vulnerabilities, threats, and attacks. A vulnerability is any weakness contained within a computing or networking system that can be exploited. A threat is defined as any process that can potentially violate the security policies of a system. An attack is considered to be any active process that deliberately seeks to violate the security polices of a system.

Organizations seek to protect their computational and networking infrastructures based on defense-in-depth strategies. The objectives of this strategy are to reduce the impact of attacks that successfully exploit vulnerabilities within the system and to reduce the number of threats within the system. The design of systems using defense-in-depth strategies should be founded on the ideas of security architectures and security management infrastructures. These two concepts have been defined by Shirey [135] as follows. A security architecture is a plan and set of principles that describe (a) the security services that a system is required to provide to meet the needs of its users, (b) the system elements required to implement the services, and (c) the performance levels required in the elements to deal with the threat environment. A security management infrastructure includes system elements and activities that support security policy by monitoring and controlling security services and mechanisms, distributing security information, and reporting security events.

2.2 Firewalls

Firewalls are security mechanisms that control the flow of network traffic, i.e., communication packets, into or out of a communication system [26]. The flow of network traffic through a firewall is governed by *security policies*, which are defined by a collection of N *packet filters* (or filters for short) written in the native language of a particular firewall technology. Sometimes, packet filters are referred to as firewall rules.

Firewalls are dependent on network topology such that filters are assigned to the direction of traffic flow relative to the orientation of the firewall with respect to the virtual resources it protects. In particular, a firewall provides an interface between *secure resources* and *insecure resources*. Secure resources reside behind an *inside interface* of the firewall, whereas insecure resources reside beyond an *outside interface* of the firewall. Figure 1 illustrates the idea of a firewall and its inside-outside topology-dependent perspective.

Firewall designs are constructed by the *Principle of Least Privilege*, which is expressed as follows:

> *All communication traffic attempting to traverse a firewall must be explicitly permitted by a security policy. Otherwise, the traffic must be denied.*

In other words, if a packet is processed by a firewall and does not match an explicitly defined filter, then the firewall must deny the packet's access into or out of the communication

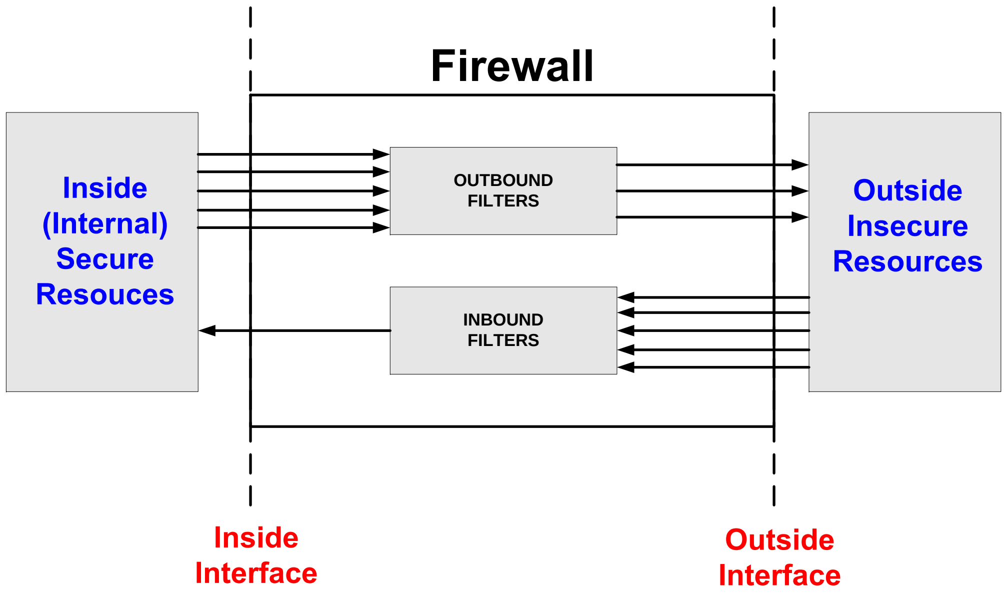


Figure 1: The inside-outside topology-dependent firewall perspective.

system. The filtering process provided by firewalls is demonstrated by the arrows of Figure 1, which represent packet flows entering and leaving the outbound and inbound filter collections. The arrows leaving out of the inbound/outbound filter collections represent packet flows that matched one or more packet filters. Packet denial is represented by more arrows entering a filter collection (inbound or outbound) than ones leaving. The suppressed arrows represent packet flows that are denied by the firewall.

Figure 2 illustrates a simple perimeter firewall system. As can be seen in Figure 2, the collection of packet filters assigned to the firewall state that packets from the 'outside' and destined to the inside host named *WWW* with destination port 80 as well as those destined to the inside host named *Email* with destination port 25 are *explicitly* permitted (allowed). Further, the filter written as *allow from inside to any port any* provides *unrestricted* access from inside users/resources to any external Internet resource. The last filter, *deny from any to any port any*, is the default filter blocking any packets that do not match the first three filters, which enforces the principle of least privilege.

Table 1 shows a snippet of packet filters from a real-world enterprize-class Cisco PIX

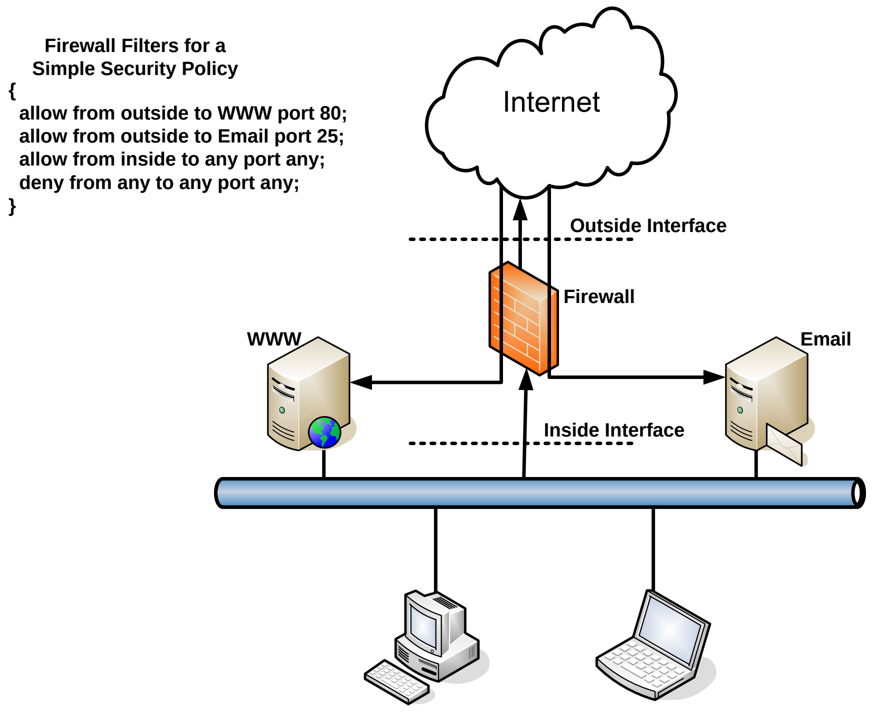

Firewall Filters for a
Simple Security Policy
{
allow from outside to WWW port 80;
allow from outside to Email port 25;
allow from inside to any port any;
deny from any to any port any;
}
Internet
Outside Interface
Firewall
WWW
Email
Inside Interface

Figure 2: A simple network system with a firewall and its filter-set.

firewall. The filters shown in Table 1 have a structure based on the so-called 5-tuple firewall filter, which includes *header field specifications* from the network and transport layers of the TCP/IP protocol stack. In particular, the 5-tuple includes the *(protocol type, source IP address, destination IP address)* from the network layer of the TCP/IP stack and the *(source port, destination port)* from the transport layer of the TCP/IP stack. In general, firewall packet filters can describe any value permissible by the specifications of the associated TCP/IP headers. For example, IPv4 addresses are contained in the source/destination IP headers in the network layer of the TCP/IP stack. Since IPv4 specifies 32 bits for IP addresses, then any one or more of the possible 2^{32} IP addresses can be used for the source or destination IP addresses of a packet filter. In most cases, firewall filters specify the tuples as a collection of *points, prefixes,* or *ranges*. A 'point' is a single numeric value, a prefix is an expression based on the first x-bits of a value followed by wildcards in the remaining $w - x$ bits where w is the number of bits allocated to the particular header field in the

Table 1: A real-world firewall filter-set.

Intf	Action	Proto	SrcIP	DestIP	SrcPort	DestPort
in	permit	gre	130.207.134.9	128.61.209.0/26	any	any
in	permit	tcp	130.207.134.9	128.61.209.0/26	any	eq pptp
in	permit	gre	130.207.134.9	128.61.209.128/26	any	any
in	permit	tcp	130.207.134.9	128.61.209.128/26	any	eq pptp
in	permit	tcp	128.61.252.110	143.215.254.64/29	any	eq https
in	permit	tcp	128.61.252.110	143.215.254.64/29	any	eq www
in	permit	tcp	128.61.252.110	143.215.254.64/29	any	range 22 23
in	permit	tcp	128.61.5.0/24	143.215.254.64/29	any	eq https
in	permit	tcp	128.61.5.0/24	143.215.254.64/29	any	eq www
in	permit	tcp	128.61.5.0/24	143.215.254.64/29	any	range 22 23
in	deny	ip	any	any	any	any
out	permit	ip	128.61.210.0/24	any	any	any
out	permit	ip	128.61.211.0/24	any	any	any
out	permit	ip	128.61.212.0/24	any	any	any
out	permit	ip	128.61.213.0/24	any	any	any
out	permit	ip	143.215.254.8/29	any	any	any
out	permit	ip	143.215.254.56/29	any	any	any
out	permit	ip	143.215.254.64/29	any	any	any
out	deny	ip	any	any	any	any

TCP/IP stack, and a range is of the form $[L, U]$ where L is the lower endpoint and U is the upper endpoint of a range. Table 2 gives an example of each. Prefixes are specified

Table 2: An example of points, prefixes, and ranges used by packet filters.

Point	Prefix	Range
128.61.208.3	128.61.55.0/24	$[20, 22]$
TCP	01000*	$[000, 110]$
http = 80	10.20.*.*	> 1023

by a certain number of *most significant bits* and are always inherently tied to the number of bits specified by a particular header field. For example, if the value 01000$*$ from the second row of Table 2 represents an IP address field, then an IP address where the first 5 bits equal 01000 will match the expression regardless of the remaining $32 - 5 = 27$ bits. In this case, the first 5 bits are the most significant bits of the filter specification 01000$*$ and the remaining 27 bits are the least significant *don't care* bits with respect to the filter expression. The range specification of the last row of Table 2 specifies any value greater than 1023. However, since the header fields of the TCP/IP stack are specified by a particular number of w bits, the range specification > 1023 implies that the range is interpreted as $[1024, 2^w - 1] = (1023, 2^w - 1]$. For instance, if the range is based on the 16 bits of the TCP port number specification, then the range above is $[1024, 65535]$.

2.3 Intrusion Detection

In general, cyberattack detection mechanisms are processes that collect data from within the cyber environment and utilize classification algorithms to determine the legitimacy of events associated with the data. A cyberattack detection system is a security mechanism that utilizes one or more cyberattack detection algorithms. A cyberattack prevention mechanism seeks to apply countermeasures that prevent adverse impacts caused by cyberattacks. A cyberattack prevention system employs one or more cyberattack detection mechanisms along with the associated techniques required to implement countermeasures.

An intrusion is defined as any unauthorized attempt to access cyber resources. Intrusion detection and intrusion detection systems are particular cases of the more general categories of cyberattack detection and cyberattack detection systems. The same is true for intrusion prevention and intrusion prevention systems. Consequently, intrusion detection/prevention and cyberattack detection/prevention will be used interchangeably throughout this dissertation.

Intrusion detection is concerned with the discovery of intruders attempting to gain unauthorized access to an organization's computing, networking, and information resources [8]. Intrusion detection is a process performed by an intrusion detection system (IDS) where events within a cyber environment are analyzed via audit data and sequences of events that indicate violations of an organization's security policies trigger alerts that notify security administrators of the abnormal system activities.

The fundamental premise underlying the design of an IDS is based on the idea of *behavior*. Understanding the behavior of computer users and processes is a central idea underlying the design of detection algorithms for intrusion detection systems. Particularly, the behavior of an intruder (human or agent) is assumed to be significantly different from the behaviors of legitimate users [142]. For example, in *most* cases, a legitimate user will not attempt to read to the contents of a system's password database. However, an intruder will *usually* attempt to retrieve the contents of the password database. The terms 'most' and 'usually' are highlighted to illustrate the *statistical* nature that actually underlies the *behavioral premise* of IDS design. But, the behaviors are not always different. In other words, a behavioral *overlap* exists whereby activities of legitimate users versus intruders (equivalently, legitimate cyber activities versus cyberattacks) cannot be perfectly differentiated at all points in time. The aforementioned behavioral overlap is illustrated by Figure 3. The preceding arguments imply that an intrusion detection system and its detection

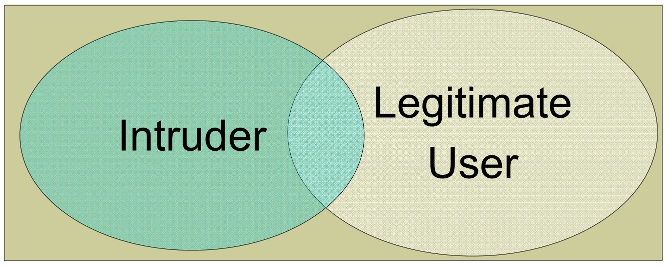


Figure 3: The behavioral overlap of legitimate users and intruders.

algorithm(s) must compensate for their inherent *false negative* and *false positive* detection rates [12].

An IDS is commonly categorized by two types of algorithmic approaches, which include *anomaly detection* and *signature detection*. Anomaly detection applies statistical analysis to real-time data based on historical archives of data representing legitimate (normal) user activity. Behavior deviating from the statistical profiles of normal activities is considered abnormal (anomalous), and these deviations are used as the basis for detecting intrusions. Signature detection utilizes a collection of rules derived from data and patterns related to known profiles of particular types of intrusive behavior. Activities that generate content with patterns matching one or more signatures within the rule-base are indicative of an intrusion. A special case of signature detection is known as *blacklisting*. A blacklist is a collection of distinct identifiers associated with well-known malicious sources. Blacklists can be composed of values such as IP addresses, application types, filenames, or transport layer port numbers. The primary distinction between these two intrusion detection approaches is the following. Anomaly detection is concerned with the behavioral (or activity) profiles of legitimate users, whereas signature detection is based on the behavioral or activity patterns (profiles) of intruders.

Intrusion detection systems require packet classification algorithms. For example, intrusion detection systems process data from TCP/IP header fields along with content belonging to application payloads during the detection process. Table 3 contains three signature detection rules similar to those found in the SNORT intrusion detection system [137]. The

Table 3: An example of SNORT detection rules (filters).

Action	TCP/IP Header Filter	Payload Filter
alert	tcp any any − > any 7070	(msg:"IDS411/dos-realaudio"; flags:AP; content:"|fff4 fffd 06|"; reference:arachnids,IDS411;)
alert	tcp any any − > 128.61.14.57 21	(msg:"IDS287/ftp-wuftp260-linux"; flags:AP; content:"|31c0db 31c9b0 31c0db|"; reference:cve,CAN-2000-1574;)
drop	tcp any any − > any 80	(msg:"Unauthorized Access"; flags:AP; content: "|21daff 03cf1a0|"; react: HTTP/1.1 403 Forbidden Connection: close Content-Type: text/html; charset=utf-8 <head> <title>Access Denied</title> </head> <body> <h1>Access Denied</h1> <p>Unauthorized Access</p> </body> </html>; sid:4;)

first two rules shown in Table 3 are *alert* rules. The first rule will be triggered for packets specified with TCP as the transport layer protocol and destined to any internal machine with a destination TCP port of 7070. Once the rule is triggered by a match on this particular TCP/IP header tuple, a second phase process will inspect the the TCP flag fields and the payload data. If the TCP flags are set to ACK (A) or PUSH (P) and if the content contains the hexadecimal string "fff4 fffd 06", then the packet matches the entire rule and an alert will be sent to a system administrator. The third rule is an *active response* rule. If a packet matches the TCP/IP header specifications along with the flags and content payload data as given in the third row of the table, then the IDS will actively *terminate* the communication flow by *dropping* (i.e., blocking) the packet.

2.4 Intrusion Prevention

Intrusion prevention can be categorized by two primary security mechanisms related to intrusion activities: (1) proactive mechanisms and (2) reactive mechanisms. *Proactive* intrusion prevention mechanisms enforce security policies with services such as authentication, authorization, and access control. For example, password systems and firewalls are proactive intrusion prevention systems. These systems implement security policies designed to prevent unauthorized access to cyber resources. *Reactive* intrusion prevention mechanisms function synergistically with intrusion detection systems and provide functionality that aims to stop intrusion occurrences in real-time and/or to recover from any adverse effects caused by a successful system penetration. Intrusion prevention systems capable of both detecting intrusions and reacting with countermeasures to intrusion occurrences are referred to as *active response systems.*

Collectively, intrusion detection and prevention systems are categorized as intrusion response systems (IRS). In [141], Stakhanova et al. propose an intrusion response system taxonomy, which is illustrated graphically by Figure 4. In their taxonomy, an IRS is classified by two top-level characteristics indicating the system's degree of automation and the system's response activity. IRS automation is categorized by notification systems, manual response systems, and automatic response systems, whereas response activity comprises passive and active response. Passive response systems are notification-based systems. Active response systems, as described above, provide real-time countermeasures to intrusions

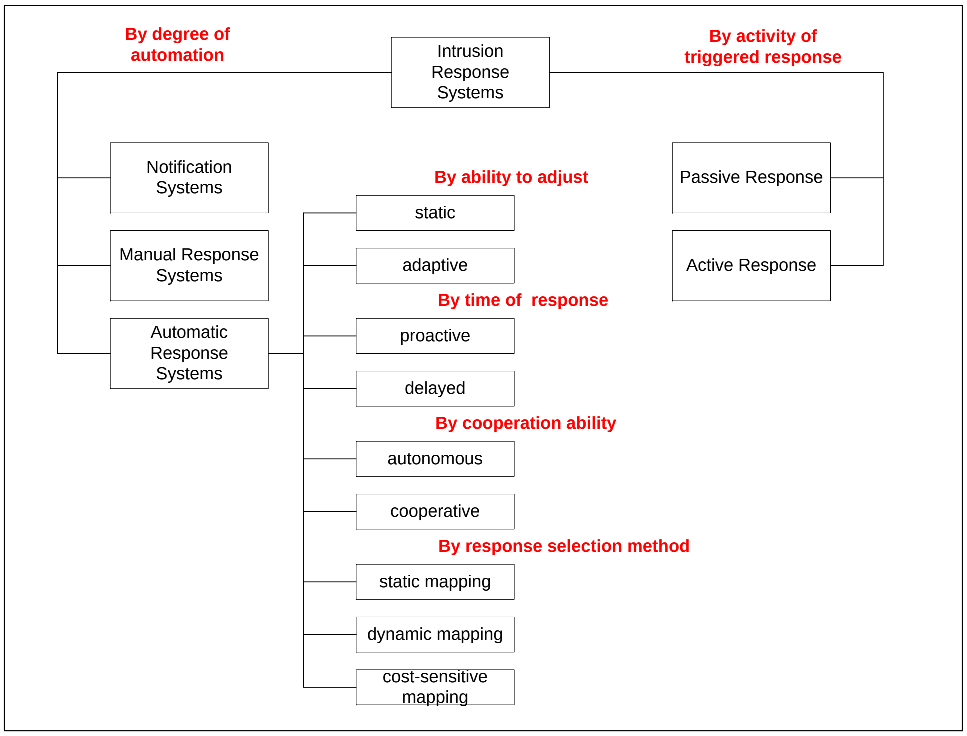


Figure 4: A taxonomy of intrusion response systems [141].

and attacks. Active response includes countermeasures such as disabling user accounts, terminating system processes, and traffic filtering. Automatic response systems are further partitioned by its adjustment ability, response time, cooperation ability, and response selection method.

2.5 Packet Classification

Packet classification (PC) provides the basis for network functionality such as Internet protocol (IP) routing, virtual private networks, firewalls, access control lists, quality of service, policy based routing, network traffic accounting, and network traffic billing. In essence, PC is the process of mapping packets into network traffic *flows*. From a theoretical perspective, PC is equivalent to the multidimensional point location problem from computational geometry whereby the fields of a packet represent a point in d-dimensional space, and the *packet filters* of a packet classifier represent d-dimensional hyperrectangles. Flows are defined by hyperrectangles, and a packet matches a flow if it is contained within the boundaries of a hyperrectangle. A *flow* is the set of all packets contained within a hyperrectangle.

Packet classification is formally described by the following. Let P be the set of all packets, R be a set of filters, and A be a collection of filter actions. A d-dimensional packet classification system implements a multivalence mapping $\mathcal{F}$ defined by the following relationship:

$$\mathcal{F} : R \times P \rightarrow A \tag{1}$$

A d-dimensional packet classifier $\mathcal{F}$ is specified by a set $R = \{R_1, R_2, \ldots, R_N\}$ of N packet filters where R_i is the i^{th} packet filter of $\mathcal{F}$. Each filter is specified over a set of d field descriptors where $R_i = \{r_{ij}\}$, $1 \leq i \leq N$, and $1 \leq j \leq d$. Each field descriptor, r_{ij}, is associated with a header field f_j from an underlying network protocol stack, where each f_j is allocated w_j bits based on its definition within the protocol stack. Further, each f_j is interpreted as a w_j bit, unsigned, binary number. Therefore, each f_j has a finite domain, D_j, where $D_j = [0, 2^{w_j} - 1]$. In other words, a field's domain is a finite range over the positive integers. In turn, each field descriptor r_{ij} can be specified in R_i as a range such that $r_{ij} \subseteq D_j$, where $r_{ij} = [l_{ij}, u_{ij}]$ and $0 \leq l_{ij} \leq u_{ij} \leq 2^{w_j} - 1$.

The semantics of packet classification can be described as follows. A packet classifier $\mathcal{F}$ receives a packet P from the network and extracts the associated d fields from the packet, and these d fields form the input field set $p = \{p_1, p_2, \ldots, p_d\} \subseteq P$. A packet *matches* the filter R_i if and only if $p_j \in r_{ij}$, $\forall j$, $1 \leq j \leq d$. In other words, a packet matches the i^{th} filter if and only if $l_{ij} \leq p_j \leq u_{ij}$, $\forall j$, $1 \leq j \leq d$. If a packet matches a filter R_i, then an associated action A_i shall be applied to the packet.

In general, a packet can match multiple filters specified by a packet classifier. In certain applications such as multi-match packet classification, the packet classifier will perform a set of actions on the packet for each of the matched filters. However, the semantics of most packet classifiers is such that a priority is assigned to the packet filters. In these cases, the packet classifier will perform the action of the highest priority matching filter. Priority, denoted by $Pri(x)$, is measured such that $Pri(R_i) = i$ and $i < j \Longrightarrow Pri(R_i) > Pri(R_j)$. Particularly, $Pri(R_1) > Pri(R_2) > \ldots > Pri(R_N)$.

Table 4 provides an example of a $d = 2$ dimensional packet classifier $\mathcal{F}$ with $N = 3$ filter-action tuples. The number of bits for header fields f_1 and f_2 are specified as $w_1 = 3$

and $w_2 = 2$. The last three rows of Table 4 are binary range representations of the first

Table 4: An example of a simple 2-dimensional packet classifier.

$\mathcal{F}$	$f_1 : w_1 = 3$	$f_2 : w_2 = 2$	
R_1	0*	01	A_1
R_2	1*	0*	A_2
R_3	11*	*	A_3
R_1	[000,011]	[01,01]	A_1
R_2	[100,111]	[00,01]	A_2
R_3	[110,111]	[00,11]	A_3

three rows, which are expressed with prefix notation.

In general, the field descriptors of a packet filter are ranges bound together over d dimensions. As a result, filters can be viewed geometrically as hyperrectangles. Figure 5 illustrates the geometry described by the filters of Table 4. The fields (f_1, f_2) generate a

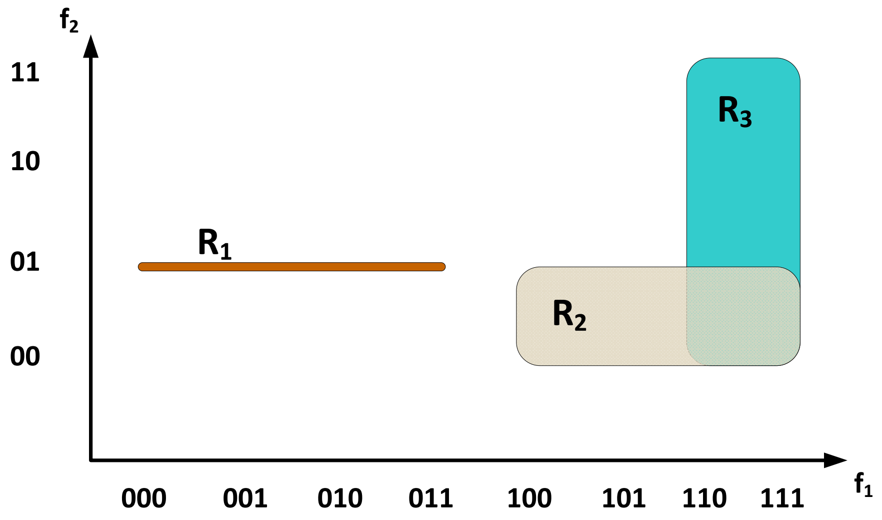


Figure 5: The geometric perspective of packet classification.

two-dimensional plane given by $[000, 111] \times [00, 11]$. Filter R_1 represents a line, whereas filters R_1 and R_2 are interpreted as rectangles. Observe that R_2 and R_3 overlap within the region $[110, 111] \times [00, 01]$. Regions with overlap encompass multi-match locations.

2.6 Summary

In this chapter, the basic concepts underlying the three primary research thrusts of this dissertation were discussed. In the next chapter, an overview of related works from within these research areas will be provided.

CHAPTER III

BACKGROUND

This chapter provides an overview of related works in distributed security architectures, packet classification algorithms, and cyberattack detection techniques with computational intelligence systems.

3.1 Distributed security architectures and defense systems

Bellovin [17, 64] introduced the concept of distributed firewalls. Perimeter firewalls enforce security policies at the ingress/egress locations of a network. With Bellovin's distributed firewall, security policies are centrally defined by administrators. However, policy enforcement is implemented by the local firewall of each host within an organization's network. Contrary to perimeter firewalls, distributed firewalls do not depend on network topology. As a result, distributed firewalls alleviate issues such as insider threats, provide security to mobile hosts that roam beyond organizational perimeters, and reduce configuration challenges associated with complex communication protocols.

Meredith et al. [97, 94] developed the autonomic distributed firewall (ADF). ADF is a distributed firewall architecture with independent (autonomic) firewalls residing at each host of the network and implementing a centrally defined security policy. The design objectives of ADF are similar to Bellovin's distributed firewall. However, ADF implements the firewall functionality within the host's network interface card (NIC). Meredith et al. claim that host-based firewalls controlled by operating systems software fail to meet the security requirements as defined by the *Trusted Computer System Evaluation Criteria* [37]. Particularly, they claim that software-based firewalls do not satisfy the *non-bypassable* and *tamper-resistant* properties of the trusted computer system criteria. To satisfy these criteria, ADF detaches the host-based firewall from the host's operating system and moves it to the host's NIC. The idea is to view the host-based firewall as an embedded firewall (EFW) within the network interface card. Figure 6 illustrates the main components of the ADF architecture. ADF has a central management server (CMS) responsible for audit collection

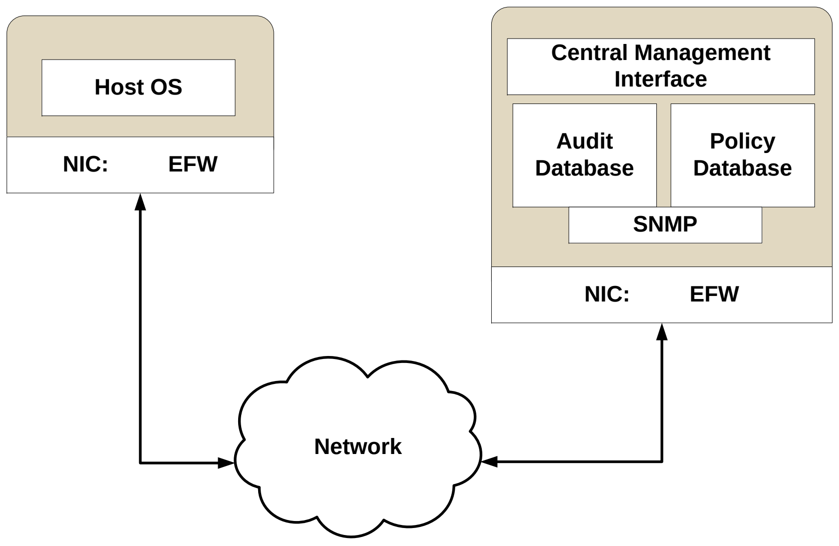


Figure 6: Primary functional components of the autonomous distributed firewall architecture.

and security policy management. Security policy is defined by security administrators, and the central management interface *pushes* the policy to the EFW. Each EFW has logging capabilities, whereby the firewall filters can can be configured to trigger alerts for various match conditions. For example, any packet attempting to traverse the firewall that gets denied can be logged, and the log data can be subsequently delivered to the CMS to be stored as security audit data.

In [102], Munz et al. present the DIADEM firewall architecture as a distributed approach to integrating intrusion detection systems and firewalls. The DIADEM firewall architecture, which is illustrated by Figure 7, is partitioned into three levels: the data level, the element level, and the administrative level. The data level is composed of routers, network monitors, and firewalls that are responsible for network monitoring and intrusion response. The element level provides an abstraction layer between the data and administrative levels and is composed of two components: monitoring elements and firewall elements. The administrative level components communicate with the data level devices via an application

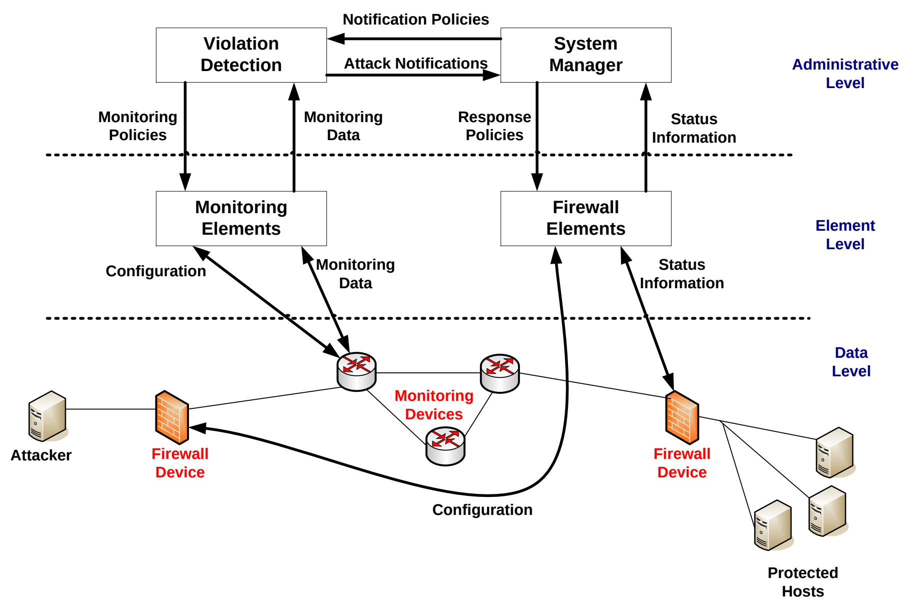


Figure 7: The DIADEM firewall architecture.

programming interface provided by the monitoring and firewall elements. The administrative level contains a violation detection unit and a system manager. The violation detection unit receives Internet Protocol Flow Information Export (IPFIX) [128] data from the network monitoring units, and it uses this data to perform intrusion detection analysis. Once an intrusion is observed, event information is delivered to the system manager. Once events are received by the system manager, it issues response policies to the network's perimeter firewalls. The response policies are firewall actions such as connection blocking, connection redirection, or connection rate-limiting. Once the violation detection system determines that an intrusion is no longer active, it sends new event information to the system manager. The system manager then removes the response policies from the firewall units.

Zou et al. proposed a firewall network system for worm defense [170]. Their architecture extends the classical perimeter firewall configuration whereby an enterprise network is divided into sub-networks that are isolated and protected by internal firewalls. Their system is designed for the purpose of defending against internal worm attacks and internal worm propagation. The architecture has the following components:

- Perimeter Firewalls

- Internal Network-Based Firewalls
- Internal Worm Detection System
- Vulnerability Assessment and Active Patching System
- Central Management Station

The authors realize that perimeter firewalls cannot protect internal hosts from an internal worm outbreak, and, therefore, recommend using internal firewalls to subdivide the internal network into isolated sub-networks. All of the internal firewalls use the same set of filtering rules, and these rules are defined, configured, and distributed with the central management station. Hosts within the enterprise network are defined by the administrator as clients or servers. The internal firewall rule base explicitly defines the clients, servers, and services offered within the enterprise's network. The authors posit that TCP and UDP requests sent from internal hosts to other internal hosts not defined as servers indicate worm propagation within the internal network. The central management station is responsible for detecting internal worm infestation. Once the internal firewall filters a connection that is targeted to a non-server host, it sends connection information to the internal worm detection system. After a certain threshold, the internal worm detection system applies quarantine methods and a vulnerability assessment and active patching system to fix the host responsible for sending the illegitimate connection requests. Figure 8 illustrates the firewall network system's architecture.

The Cooperative Intrusion Traceback and Response Architecture (CITRA) was designed to allow intrusion detection systems, firewalls, and network devices to cooperative with each other so attack traceback and dynamic attack blocking could be achieved [133]. CITRA is illustrated by Figure 9. CITRA environments are built from communities and neighborhoods. A community is composed of neighborhoods interconnected by boundary controllers (BC). Further, the community is a single administrative domain that is controlled by a discovery coordinator (DC), which is a centralized management unit. Neighborhoods are composed of CITRA devices within a perimeter formed by boundary controllers. Devices within a neighborhood include hosts, routers, intrusion detection systems, and firewalls.

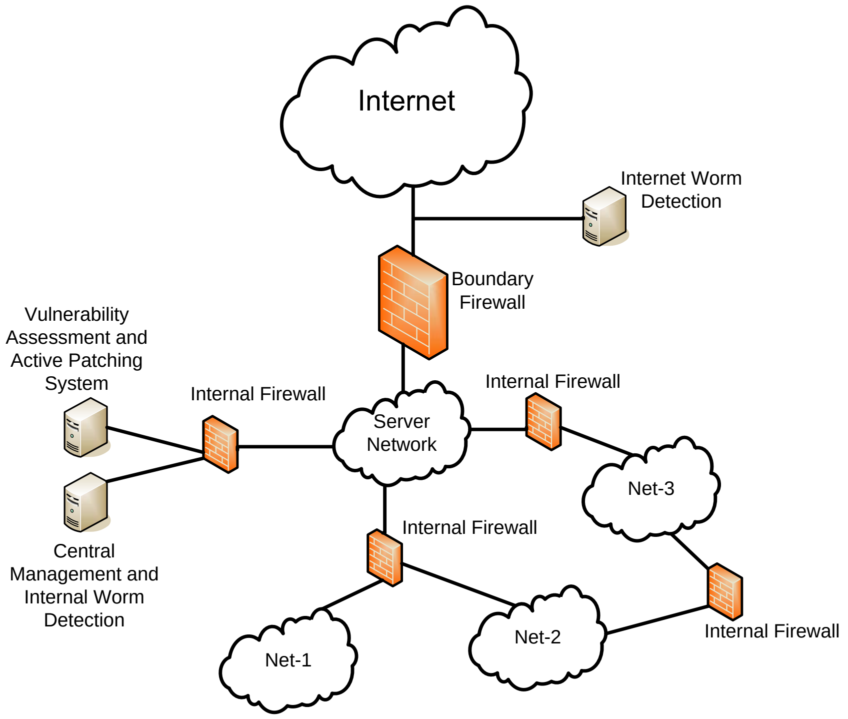


Figure 8: The firewall network system for worm defense.

The community structure allows for attack traceback. To enable traceback, CITRA devices that detect intrusions create audit trails describing the event and send a traceback request to its neighboring BC. If the BC determines that the intrusion has passed through its path, it sends a traceback request to its neighboring BC. This process is repeated until the source of the attack has been determined or until the perimeter of the CITRA community has been reached. This process allows the system to determine the attack source or the entry point of the attack within the community.

Event Monitoring Enabling Responses to Anomalous Live Disturbances (EMERALD) [115] is an intrusion detection and response architecture. The system uses distributed and independent monitors that perform intrusion observation, analysis, and response. EMERALD event analysis is performed with a layered approached. EMERALD service monitors are deployed throughout an administrative domain as service-layer monitors, domain-layer monitors, and enterprise-layer monitors. EMERALD's three-layer event analysis framework

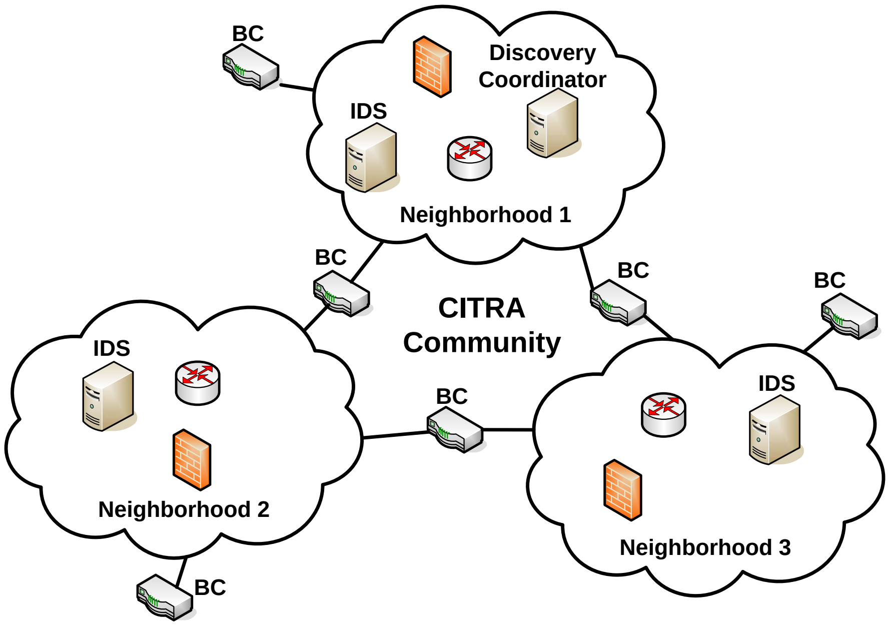


Figure 9: The cooperative intrusion traceback and response architecture.

is shown by Figure 10. The service-layer monitors are configured to use target-specific event streams when making localized intrusion decisions and responses. The collection of monitors within a single domain share information with a domain-layer monitor, and the domain monitor correlates and analyzes the domain specific event information. Then, enterprise-layer monitors correlate and analyze event information gathered by the domain monitors. Service monitors are responsible for detecting attacks targeting local services, whereas domain and enterprise monitors are configured to detect system wide attacks such as worms and denial of service. Service analysis covers the detection of network anomalies within a single domain. The service monitors are designed to be independently tunable for specific types of analysis. For example, the monitor might specialize in detecting anomalies that target an FTP server. Information is exchanged between monitors over a subscription-based, client-server messaging system.

The EMERALD monitor contains a target specific event stream, a profiler engine, a signature engine, a resolver, a resource object, and third party modules. Target specific event streams are collected and generated by target specific event collectors. The collectors

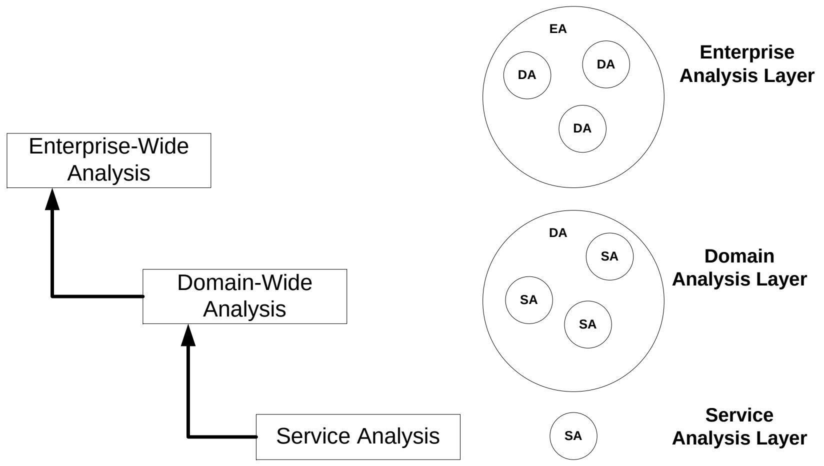


Figure 10: Three-layer analysis within EMERALD.

receive their data from audit trails, network packet capture, simple network management protocol messages, log files, or results produced by other intrusion detection systems. The event streams are sent to the profiler and signature engines for analysis. The profiler engine performs statistical, profile-based anomaly detection where statistical scores are assigned to observations of application activities. This score represents the degree to which observed application behavior relates to established patterns of application behavior. The signature engine performs rule-coding signature analysis where event streams are mapped to representations of event sequences that are known to indicate malicious activity. The resolver receives its input from the profiler engine, the signature engine, and third party analysis engines. The resolver coordinates analysis results and implements the intrusion response policy. The resolver can also send its results to third party security modules such as firewalls that provide the final response.

3.2 Software-Based Packet Classification Algorithms

Packet classification provides capabilities for tasks such as packet forwarding (routing), virtual private networking (VPN), implementing quality of service (QoS), traffic accounting and billing, network monitoring and measuring, traffic analysis, security applications such as intrusion detection/prevention systems, firewalls, and router access control lists. Packet classification and its associated notation was formally defined in chapter 2, section 2.5. In this section, an overview of related works addressing the design of packet classification algorithms is given.

3.2.1 One Dimensional Packet Classification Algorithms

Previous work on the one-dimensional packet classification problem consists largely of algorithms seeking to provide efficient packet forwarding techniques for routers and addresses the longest prefix match (LPM) route lookup problem arising from the introduction of CIDR IP addressing. Considerable attention has been given to the LPM route lookup problem with seminal results dating back to the early 1990's.

Packet classification algorithms must strive to achieve ***wire-speed*** classification of packets while supporting search data structures (SDS) that consume reasonable amounts of memory (space) and providing techniques for incremental (online) updates to the SDS when changes to the classifier's filters are needed. Achieving these goals simultaneously has proven to be very challenging. For example, LPM lookup algorithms need to achieve classification rates at 100's of millions of packets per second for routers with line rates of 10's to 100's of gigabits per second. Moreover, within the Internet core routing table updates can potentially occur 10's to 1000's of times per second.

Algorithms based on tree-type data structures have been proposed for the one-dimensional packet classification problem and for the LPM algorithm in particular. A trie is a tree-type structure that encodes sequences of character strings from a given symbol family. The paths through a trie represent strings of characters. *Multibit Tries* (MBT) are general purpose data structures and provide efficient encoding and retrieval methods for sequences of binary symbol patterns. Multibit tries are tree-type structures specified by a *stride*. The stride is a bit grouping or bit chunking factor defining the number of bits to inspect at each level of the trie. Given the parameter w specifying the number of bits in a symbol family, the stride c partitions a w-bit word into $\frac{w}{c}$ chunks, and each chunk generates up to 2^c possible bit patterns. A multibit trie with a single stride specification is also referred to as a *fixed stride trie* (FST). Variable stride tries extend the concept to include sequences of strides.

A unibit or binary trie is produced by assigning a stride of $c = 1$. Figure 11 illustrates a unibit trie encoding the prefixes from Table 5.

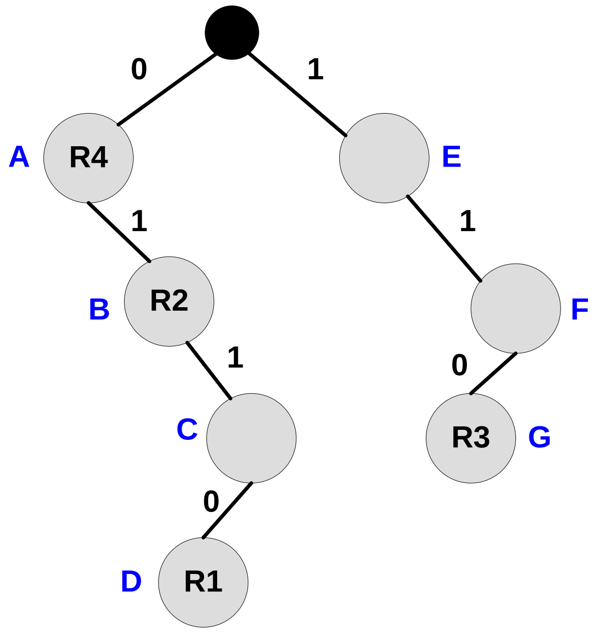


Figure 11: A unibit trie for the prefixes from Table 5.

Table 5: A one-dimensional routing table of prefixes.

$\mathcal{F}$	$f_1 : w_1 = 6$	Action
R_1	0110*	A_1
R_2	01*	A_2
R_3	110*	A_3
R_4	0*	A_4

From the figure, the top shaded node represents the root of the trie. Traversal of the trie proceeds with the first inspected bit. An inspection bit equal to 0 causes a left branch whereas a value of 1 causes a right branch. Nodes labeled with R_i indicate that the node is a *prefix node* and has an association with the i^{th} filter. The maximum height of a unibit trie is w, and the height of the root is defined to be 0.

Locating a matching prefix is straightforward. For example, let $p = 010001$. The search proceeds from the root. Since the first bit of p is 0, a left branch is taken to the node labeled A. The second bit is 1, so a right branch is taken to the node labeled by B. The third bit is 0. However, there is no left branch below node B. Hence, the best matching prefix is R_2 since node B is a prefix node and since it is the longest matching prefix in the trie for the sequence given.

Consider the case where $p = 011101$. Traversal of the trie yields the node sequence A, B, C. Since the 4^{th} bit of p is 1 and node C does not contain a branch for 1, the search can go no deeper into the trie. However, node C is an internal node that does not contain a matching prefix. Hence, the search must *backtrack* to the last prefix node that was seen along the path, which is node B. Backtracking in prefix tries is handled by recording the prefix nodes during the traversal. For the one-dimensional PC problem, backtracking is only a minor concern. However, backtracking prevents high-dimensional extensions, i.e., large values of d, to the basic one-dimensional prefix trie. The details of this problem will be discussed in later sections.

For the one-dimensional case, algorithms implementing the basic unibit prefix trie have an $O(Nw)$ memory complexity and an $O(w)$ search time complexity. As w grows large, the unibit prefix trie becomes less appealing, especially for applications requiring wire-speed packet classification. Particularly, route lookup algorithms using the unibit trie data structure require $O(32)$ memory accesses per lookup for IPv4 and $O(128)$ memory accesses per lookup for IPv6. Several techniques have been proposed seeking to *pipeline* unibit and multibit tries for route lookup applications. These hardware-based algorithms will be explored in later sections.

One method to reduce the search time complexity of the unibit trie is to utilize a multibit trie. The stride chunks the bits of a w-bit word into groups of c bits. A multibit trie provides the capability to perform multiple bit comparisons during each search operation, which makes it more favorable as compared to the unibit trie. The multibit comparisons of a multibit trie provide *multiway* branching decisions during a search. Specifically, a stride c has a multiway branching factor of 2^c whereby one of 2^c possible paths can be determined by inspection of c bits. A multibit trie with a fixed stride of c bits will contain a maximum number of levels equal to $\frac{w}{c}$. In other words, the maximum height of a multibit trie with stride c is $\frac{w}{c}$.

Figure 12 shows the prefixes from Table 5 encoded by a trie with a stride $c = 2$. A few

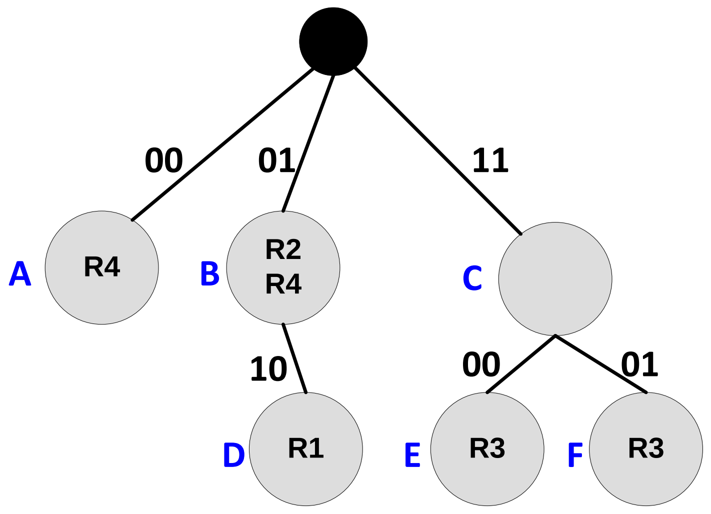


Figure 12: A multibit trie with stride of two for the prefixes from Table 5.

things should be observed from the trie in Figure 12. First, node B is *shared* by R_2 and R_4. Since R_4 has only the first bit specified with all others given as wildcards, the stride forces the trie to *replicate* R_4 across the two nodes labeled as A and B. Node B is associated with both filters. However, since R_2 has a longer prefix length given by the table, it is actually the filter to be assigned to node B. The node was labeled with both filters in this example in order to clarify this concept of *prefix expansion* that might be required by multibit prefix tries. In particular, if a prefix is of length L and not a multiple of the stride, then it will require an expansion. Observe that R_3, which is specified by the first 3 bits, was expanded into the second level of the trie since 3 is not a multiple of 2 (the stride). The search time complexity for a multibit trie is $O(\frac{w}{c})$ and its memory complexity is $O(N\frac{w}{c}2^c)$.

With a variable stride trie (VST), a sequence of strides $K_c = \{c_1, c_2, \ldots, c_m\}$ is specified for the w-bit binary symbol family. The VST has a height of m where $m = |K_c|$. The search process is similar to fixed stride tries. The w-bit string is processed by inspecting the first chunk of c_1 bits followed by the second chunk of c_2 bits and so on until the longest prefix match is found or until it is determined that no matching prefix exists.

When designing a VST, a new question arises: *"What is the best sequence of strides to* *select for K_c?"* Srinivasan and Varghese [139] sought to answer this question and proposed a dynamic programming approach coupled with their *controlled prefix expansion* algorithm. Particularly, they use dynamic programming [31] to determine the optimal set of strides that produce a guaranteed worse case number of lookups per classification (number of memory accesses per classification) while minimizing memory consumption.

The controlled prefix expansion (CPE) algorithm proposed by Srinivasan and Varghese [139] aims to transform prefixes with L distinct lengths to a new but equivalent set of prefixes having a predefined set of lengths. Any *particular* prefix of length ℓ_i can be expanded into an equivalent *collection* of prefixes with length $\ell_j > \ell_i$. Table 6 illustrates the basic concept of prefix expansion. The first three rows of Table 6 have three *distinct* prefix lengths of

Table 6: An example of prefix expansion.

$\mathcal{F}$	$f_1 : w_1 = 6$	Action
R_1	01*	A_1
R_2	110*	A_2
R_3	0*	A_3
R_{11}	010*	A_1
R_{12}	011*	A_1
R_2	110*	A_2
R_{31}	000*	A_3
R_{32}	001*	A_3
R_{33}	010*	A_3
R_{34}	011*	A_3

2,3, and 1 respectively. The lower set of rows represents the result of prefix expansion under the constraint that all prefixes must have length-3. Observe that R_1 expands into two *equivalent* prefixes and R_3 expands into four sets of length-3 prefixes. Observe that R_{11} and R_{33} contain the same prefix, as do R_{12} and R_{34}. In these cases, the prefixes are said to be *collisions*. With the CPE algorithm, a process termed *prefix capture* locates colliding prefixes and reassigns the filters accordingly. For example, prefix capture produces the filter-set shown in Table 7. The filters R_{33} and R_{34} are removed in order to retain the original filtering semantics defined by R_1, R_2, and R_3. In particular, R_1 from the original filter-set has a longer prefix specification (i.e., $\ell = 2$) than R_3, which has length $\ell = 1$.

Table 7: Resolving prefix collisions with prefix capture.

$\mathcal{F}$	$f_1 : w_1 = 6$	Action
R_1	01*	A_1
R_2	110*	A_2
R_3	0*	A_3
R_{11}	010*	A_1
R_{12}	011*	A_1
R_2	110*	A_2
R_{31}	000*	A_3
R_{32}	001*	A_3

Another technique considered for trie-based packet classification algorithms is known as *leaf-pushing.* The nodes of a trie are generally of the following types: *the trie root, prefix nodes, internal nodes,* or *leaf nodes.* Prefix nodes are nodes that contain information associated with a prescribed prefix and possibly pointers to child nodes at the next lower level of the trie. Internal nodes do not contain information but are required to build a path for one or more prefixes and their associated prefix nodes. Hence, internal nodes will only contain pointers to child nodes at the next lower level. Leaf nodes are the bottom-most nodes of any path in the trie. Leaf nodes will *always* be prefix nodes. However, the converse is not necessarily true–prefix nodes are not necessarily leaf nodes. Table 8 provides a generic model for the data structure of a trie node. The idea of leaf-pushing

Table 8: Generic model for the data structure of a trie node.

Information_Field	[null if internal node]
Child_Pointer_Array	$[cp_1, \ldots cp_j]$

is to push information fields from non-leaf prefix nodes to their associated leafs. Figure 13 illustrates the concept where the leafs are assumed to reside at level 2 of the trie. The assumption of leaf-pushing is that no internal nodes will be prefix nodes and all leaf nodes will be prefix nodes. As a result, the internal nodes will only require pointer fields and, consequently, total memory consumption can be reduced in many cases. The CPE algorithm of [139] uses leaf-pushing as another optimization technique seeking to further

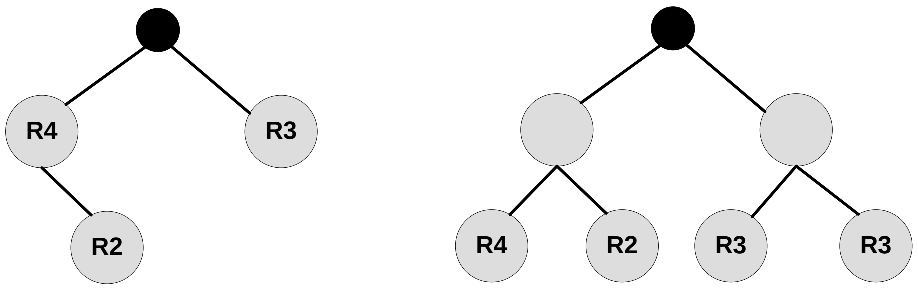


Figure 13: Pushing information fields to the leaf nodes in a trie.

reduce memory consumption. However, an algorithm based on a leaf-pushed trie requires precomputation overhead and *online* modifications to the associated prefix database can require changes to many of the nodes in the trie. Therefore, leaf-pushed trie schemes are unfavorable for packet classification applications where the prefix database (i.e., the filter-set) changes with high-frequency. As a result, algorithms utilizing a leaf-pushing scheme are generally categorized as non-incrementally updatable.

Degermark et al. [33] introduced a data structure and associated encoding methods designed for IPv4 route lookup algorithms. Their algorithm is commonly referred to as the *Lulea* algorithm, and their primary objective is to use special encoding techniques such that the entire data structure representing a routing table can fit into the high-speed cache memory of central processing units. The Lulea algorithm employs a variable stride trie with a stride sequence $K_c = \{16, 8, 8\}$. Further, the algorithm utilizes bitmaps to compress its data structures. Since $m = |K_c| = 3$, the Lulea trie has a height of 3. The first level of the trie covers prefixes of lengths 1-16, the second level covers lengths 17-24, and the third level covers lengths 25-32. Degermark et al. claim that the Lulea algorithm can compress an IPv4 routing table with 40,000 entries while only requiring 150-160 Kbytes. They claim, further, that the algorithm can perform a lookup amongst the 40,000 routing table entries with less than 100 instructions on an Alpha processor and only using 8 memory accesses. However, the Lulea algorithm suffers from poor incremental update times because of the amount of precomputation required to process and generate its compressed data structures.

Path compression is a technique seeking to eliminate *one-way* branches that may exist within prefix tries. A path compressed trie utilizes the notion of a *skip field* that circumvents the need to encode paths containing long sequences of one-way branches. In particular, if a prefix path within the trie contains an internal non-prefix node with only one *child* node, then the internal node is removed. Further, the removal process occurs consecutively along a prefix path containing a sequence of internal non-prefix one-child nodes. For the process to work, a skip field is added to the parent of the removed node(s), which indicates the number of inspection bits to skip during the traversal process. Figure 14 illustrates the path compression concept.

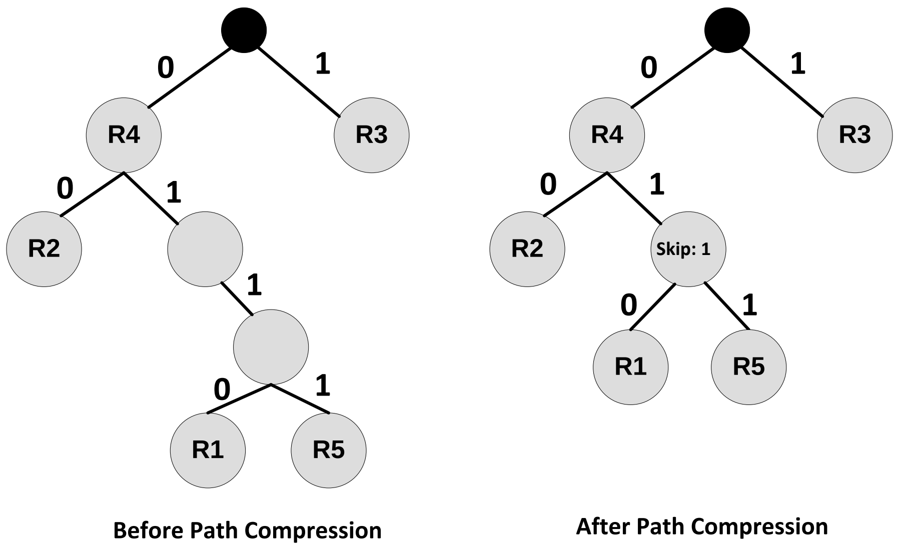
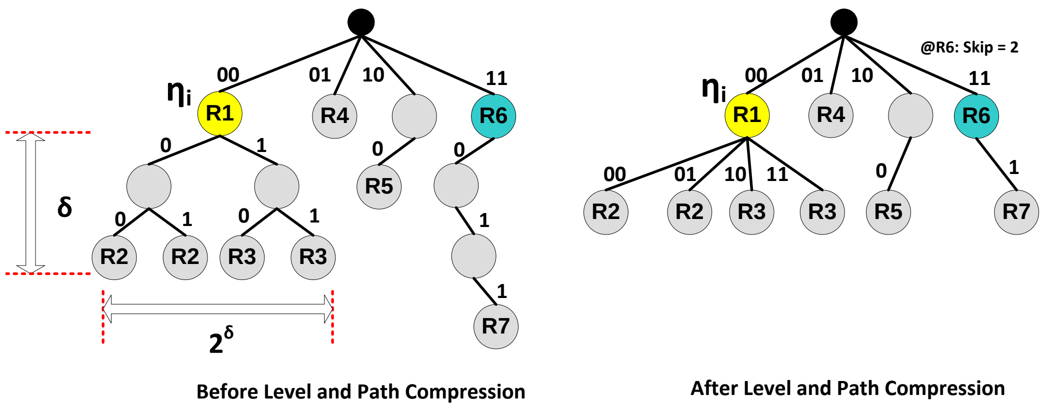


Figure 14: Illustration of the path compression concept.

Nilsson and Karlsson introduced the *LC-trie* algorithm for the LPM route lookup problem [107]. Their algorithm utilizes path compression along with a technique they refer to as *level compression*. The idea behind level compression is to decrease the number of memory accesses required for prefix lookups by means of reducing the length of prefixes stored within a trie. The LC-trie algorithm starts with a binary unibit trie (UBT) representation of routing table prefixes. Path compression is applied to the UBT followed by level compression. The level compression procedure transforms a *complete binary subtree* of depth δ into a subtree of depth 1. In particular, let a node η_i at level i of the trie be the *root* of a complete binary tree (CBT) with depth δ. Since the binary subtree rooted by η_i must be *complete*, the number of nodes at a depth of δ below the root η_i is 2^δ. The LC-trie algorithm compresses the number of levels of the CBT from δ to 1 by assigning each of the 2^δ descendants of the root node η_i as its direct-children. The process is called level compression because it reduces the depths of CBTs throughout the trie, which consequently reduces the number of levels of the trie for *dense* regions satisfying the complete binary tree constraint of the LC-trie algorithm. Figure 15 illustrates the level and path compression ideas of the LC-trie algorithm.



ηi
00
01
10
11
R1
R4
R6
0
1
R5
δ
R2
R2
R3
R3
R7
2δ
Before Level and Path Compression
@R6: Skip = 2
After Level and Path Compression

Figure 15: Level and path compression techniques of the LC-trie algorithm.

The algorithmic complexities of the LC-trie algorithm are similar to other multibit trie methods. The worst case time complexity of the LC-trie depends on its height, i.e., the longest prefix path encoded by the trie. However, the height will depend on the distribution of prefixes. Nilsson and Karlsson claim the average depth of an LC-trie to be $O(\log^* N)$ where $\log^* N$ is the iterated logarithm function.

A variant of the LC-trie algorithm was proposed by Cheung et al. [27]. Cheung et al. transform a prefix database into an LC-trie and then seek to partition the trie into a fixed set of equivalent table representations. The partitioning method is driven by an optimization process that models the access times and storage capacities of hierarchical memory architectures such as those in multi-level cache computer systems. The goal of their optimization strategy is to take as input a complete binary prefix tree and transform it into a level compressed trie such that when the elements of the LC-trie are stored across a 3-level memory hierarchy, the arrangement of the trie with respect to the hierarchy produces a structure that minimizes the average lookup time.

The prefixes of w-bit fields represent intervals within the domain $[0, 2^w - 1]$. One method for encoding the prefixes of a one dimensional packet classifier is the interval binary search (IBS) algorithm. Interval binary search is a general concept. For example, Berg et al. [18] describe techniques for searching interval data in computational geometry applications, and a data structure for an interval binary search tree (IBS-tree) for finding intervals overlapping query points was described by Hanson and Chaabouni [53].

The basic IBS algorithm encodes non-overlapping disjoint ranges. However, prefix ranges will in general have range overlap. Consequently, a preprocessing algorithm that disjoins the prefix ranges of a filter-set is required for packet classifiers that implement IBS lookup algorithms. Figure 16 will be used to illustrate the concept. The lines in the top portion of

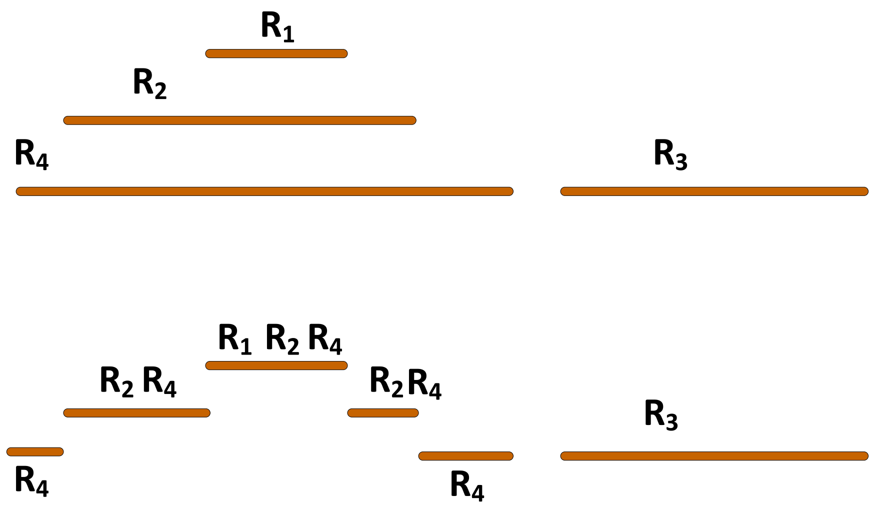


Figure 16: Creating disjoint ranges from the filter-set of Table 5.

Figure 16 are range representations of the prefixes from Table 5. As shown in the figure, the prefixes given by R_1, R_2, and R_4 are overlapping. The overlaps can be understood via inspection of the prefixes. The lower portion of the figure illustrates the collection of disjoint ranges. Each of the disjoint *subranges* are labeled by the filters sharing common overlaps. For example, filter R_1 overlaps both R_2 and R_4. Once the overlaps are removed, the original four ranges produce six disjoint ranges.

The disjoint subranges are called *basic intervals.* Once the basic intervals are precomputed from the filter-set, the filters associated with the overlapping regions (ranges) are pruned such that a *best matching prefix* (BMP) is assigned to each basic interval. The BMP is assigned according to the longest prefix length of the filters within an overlapping region. Figure 17 shows the corresponding BMPs for the basic intervals of Figure 16. In

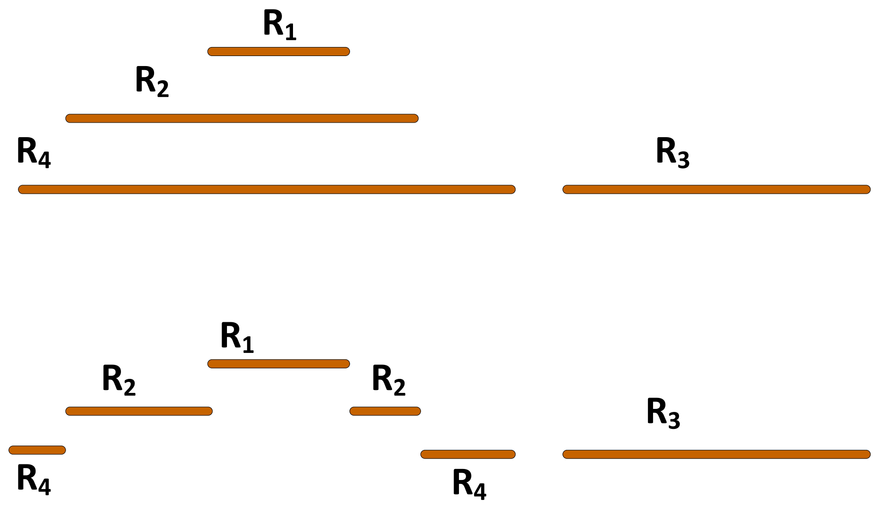


Figure 17: Assigning the best matching prefixes to the basic intervals from Table 5.

general, a collection of N overlapping ranges can be disjoined into a maximum of $2N-1$ disjoint ranges. As an example, the three ranges R_1, R_2, and R_4 with overlap produce $2*3-1=5$ disjoint ranges.

The IBS approach possesses the traditional $O(lgN)$ time complexity and $O(N)$ space complexity of binary search algorithms. However, the difficulty of the IBS approach for packet classification results from the precomputation time required to generate disjoint ranges. Moreover, binary search methods do not easily extend into higher dimensions.

3.2.2 Multidimensional Packet Classification Algorithms

Cache-based packet classification techniques aim to bypass the classification process by storing tuples of information that identify previously observed packets. A scheme similar to caching is referred to as *traffic aware* packet classification. The objective of traffic aware packet classification is to order packet filters such that the *heavy hitter* filters are processed during the early stages of the classification process. Traffic aware classifiers are enabled via statistical analysis of filter sets along with associated network traffic characteristics.

Acharya et al. have proposed several techniques based on the notion of traffic aware firewall optimization [5, 4, 3]. In [3], the OPTWALL firewall optimization framework is introduced. OPTWALL is based on a hierachical design and uses an online traffic adaptation scheme that monitors firewall log files in order to maintain optimal ordering of firewall filters. Hamed et al. [51, 50, 49] introduced algorithms for dynamic optimization and dynamic ordering of packet filters for high-speed firewall applications. Their algorithms utilize Internet traffic characteristics and statistical search trees to construct filter sequences that seek to minimize the average time required to perform packet classification in firewalls. Traffic aware packet classification techniques have the potential to increase packet classification performance for environments that utilize smaller numbers of filters and that possess slowly changing traffic characteristics. However, the performance of these systems suffer, similar to cache-based classifiers, when traffic characteristics are extremely diverse.

Traffic aware packet classification systems are typically designed for linear search algorithms, which is the underlying motivation for optimal placement of heavy hitter filters. Although traffic aware schemes seek to achieve higher average case performance, these methods have $O(N)$ worst case search time complexity. Moreover, Fulp has shown that finding the optimal order of packet filters using traffic aware schemes is NP-hard [42].

Srinivasan et al. [140] introduced the *Grid-of-Tries* algorithm as a tree-based search method for two dimensional packet classification with packet filters specified by source and destination IP address prefixes. Grid-of-Tries (GoT) is constructed by allocating a single destination trie for the destination prefixes and a grid of source tries representing the source prefixes. The basic idea is to have each prefix node in the destination trie representing a valid (specified) filter prefix contain a pointer interlinking the destination prefix node to a source trie containing its associated source prefixes. This is a generalized methodology for encoding multidimensional, i.e., d-dimensional, prefix strings using *multidimensional tries*. In short, a multidimensional trie (MDT) is constructed such that the root node of a sub-trie in the i^{th} dimension is interlinked to its conjunctively-associated prefix node(s) from the $(i-1)^{th}$ dimension and the prefix nodes of the sub-trie in the i^{th} dimension are interlinked to the root nodes of their corresponding sub-tries in the $(i+1)^{th}$ dimension. However, this method suffers from poor search time performance resulting from the well-known *backtracking* dilemma.

Figure 18 will be used to illustrate the backtracking problem within the context of the GoT algorithm. The figure assumes a two dimensional packet classifier where each field

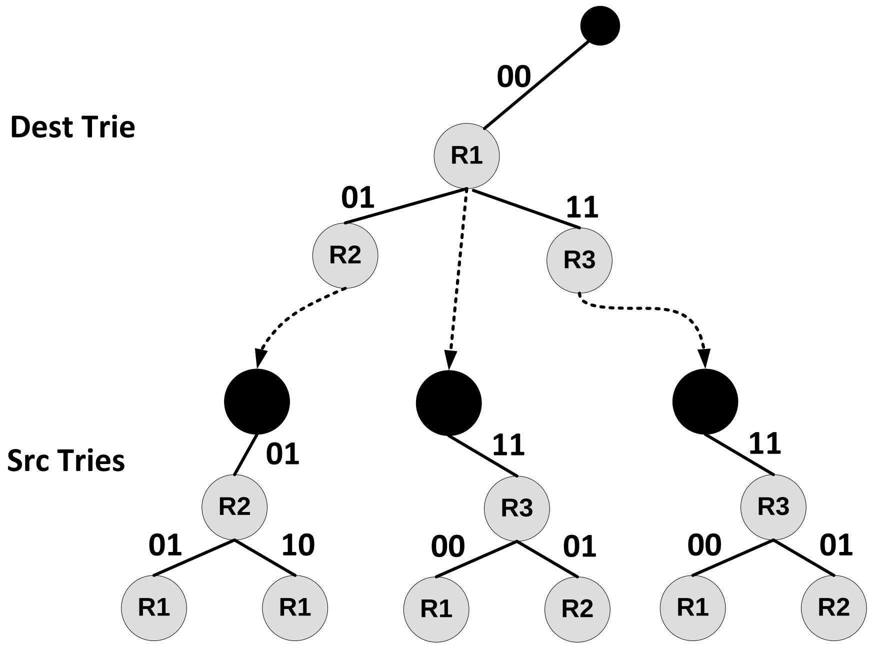


Figure 18: A two dimensional source/destination prefix trie.

is defined by $w = 4$ bits, and the tries have a stride of $c = 2$. From the figure, nodes with black shading represent the trie root nodes, and nodes labeled with R_i represent prefix nodes associated with the i^{th} packet filter. The upper Dest Trie (DT) is the trie for destination prefixes, and the Src Tries (ST) located in the lower region of the figure represent source prefixes. Consider a packet with source and destination fields given by $p = (src, dst) = (1110, 0011)$. The search starts at the DT root. The first two bits of the destination field equal 00, and the DT indicates that a partial match exists because of the edge labeled 00, which leads to a prefix node labeled by $R1$ indicating that the filter $R1$ is matched. However, the LPM algorithm dictates a continuation of the search since a valid branch exists based on the next two bits with value 11. At this stage, the search has converged onto a terminal node in the DT, which has only one branching option that leads to the rightmost source trie. The first two bits of the source field have a value of 11, which indicates that the node labeled $R3$ in the rightmost ST matches the packet. However, the final two bits 10 of the source field do not have a matching branch. At this point, the search has failed to find a longest prefix match along this particular traversal. However, other matches were observed along the way, and, hence, the search must backtrack to the most recent longest matching prefix. In this case, the node $R1$ in the DT was the last node matching the search. Node $R1$ is a level-one node and contains a pointer to the middle-most ST. Hence, the node represents a destination field specification given by $00*$. Traversal from $R1$ to the middle-most ST along with inspection of the first two bits of the source field indicates a positive match leading to node $R3$ in the middle-most ST. However, inspection of the last two bits of the source field with a value of 10 reveals that no match exists. At this stage, the search has completed and the algorithm returns a *no match* condition.

Packet classification algorithms based on the MDT are appealing because these algorithms *naturally* encode packet filter prefix specifications. However, the algorithmic complexities of MDT-based techniques cause challenges. In particular, MDT algorithms suffer from the backtracking problem. As a result, the time complexity for these algorithms is $O(w^d)$. The space complexity is $O(Ndw)$. Consequently, the baseline method behind the GoT packet classification algorithm for the $d = 2$ dimensional case of source and destination prefixes has a search time in $O(w^2)$. As seen by these algorithmic complexities, the backtracking problem can severely limit MDT-based algorithms when the magnitude of d and/or w increases.

To alleviate the issues associated with the baseline MDT methods than underly their GoT algorithm, Srinivasan et al. [140] consider set pruning trees to reduce GoT's overall memory consumption and utilize precomputation methods. The precomputation process assigns switch pointers to nodes within the source tries that enable jumps between the source trie nodes when traversals fail. Essentially, the switch pointers eliminate some backtracking operations but with a cost incurred by update time resulting from precomputation of the packet filters to generate the switch pointers. Srinivasan et al. [140] claim that improvements made by set pruning and switch pointer modifications result in a memory complexity that is $O(Nw)$ while reducing the search complexity from $O(w^2)$ to $O(2w)$.

Although the Grid-of-Tries algorithm extends packet classification with tries from one to two dimensions while maintaining reasonable search and memory performance, it does not easily extend into dimensions greater than $d = 2$. However, Baboescu et al. [14] introduced the d-dimensional Extended Grid-of-Tries (EGoT) packet classification algorithm in an effort to increase the dimensional capabilities of the GoT algorithm. EGoT utilizes GoT for its core functionality, but implements jump pointers between the nodes of longest matching prefix paths within the source tries to linear lists of filters over the remaining $d-2$ fields associated with the *jump node*. For example, consider the GoT from Figure 18 and let a $d = 4$ dimensional filter be specified by $R = (src, dst, protocol, port) = (1101, 0011, TCP, port = 25)$. Then, the rightmost Src Trie would have a jump pointer from its node labeled $R2$ to a list containing the filter residual $(TCP, port = 25)$. The various paths of the source and destination prefixes provide partitions of the overall list of N packet filters. However, the search efficiency of the algorithm depends on both the total number of filters and the amount of partitioning induced by the prefix paths.

Srinivasan et al. [140] proposed the Cross-Producting (CP) algorithm as a method for multidimensional packet classification. The CP algorithm is based on the notion of *field decomposition* whereby the d fields of a packet classifier are searched independently of one another to produce an intermediate classification result. The result is intermediate because the fields of a filter are conjunctively bound to one another. The general idea of field decomposition is to encode the filter specifications of each field separately and independently of the others using an efficient lookup method such as binary search. The result of this *phase-one* search is a collection of filters matching the independent field criteria. Then, each of the intermediate results are combined to find the matching filters common to all of the intermediate results.

Gupta [48] introduced the recursive flow classification (RFC) algorithm, which is considered one of the fastest software-based packet classification algorithms. RFC uses the idea of cross-products, similar to the scheme proposed by Srinivasan et al. [140]. However, RFC implements the cross-products throughout stages. Regardless of the number of filters, RFC can classify a packet with a small number of memory lookup and comparison operations, thereby providing a near-constant $O(k)$ classification time. However, the algorithm has poor memory and update time characteristics. The worst case memory requirements is $O(N^d)$. The data structures generally require a complete re-build during an update, and, consequently, RFC falls into the category of packet classification algorithms that are only applicable to problems where filter updates occur infrequently.

3.3 Hardware-based Packet Classification

This section provides an overview of hardware-based packet classifiers based on content addressable memories (CAM), ternary content addressable memories (TCAM), and SRAM-based trie-pipelining systems.

Content addressable memories and ternary content addressable memories are associative memory constructs that implement a fully parallelized linear search over a list of stored patterns. The CAM can be viewed as an $N \times w$ matrix of *cells* where each cell stores a single binary value from the set $B = \{0, 1\}$ and each row of the CAM stores a w-bit binary word. The primary functional components of a content addressable memory include the $N \times w$ matrix of binary cells consisting of N binary w-bit words $W_1, W_2, \ldots, W_N$, an array of w search line drivers, an array of N match line drivers and an N-to-$\lg N$ priority encoder. Figure 19 provides a block diagram illustrating the basic CAM components. Typically, a

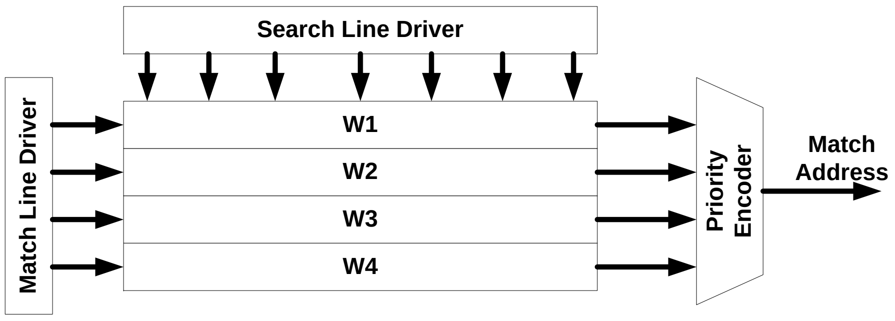


Figure 19: Block diagram of a content addressable memory and its main functional components.

CAM cell is comprised of a single SRAM cell, NMOS access transistors used by read/write operations from/to the SRAM cell, and transistor circuitry that provides bit comparison logic. The primary function of content addressable memory is to store tables of binary words and to provide parallelized linear search over the stored data. In general, the CAM provides a one-to-one relationship between a stored binary word and its address. The one-to-one content-to-address relationship is actually an indirect association between the stored content and some other ancillary data. The stored words represent database content, and the address of the content relates to the associated result needing to be returned by the search. Consider the association (B,A). The content B is associated to the value A. The CAM will store B at some internal location L, which forms the content-to-address relationship (B,L). Then, the value A is stored within another memory M at address L where M(L) = A. During the search procedure, a search key is submitted to the CAM. If a match is found, the CAM returns the address of the matched word as the search response. When the search word B, is submitted to the CAM, it returns CAM(B) = L. L is then used to directly access the desired value associated with B as M(L) = A.

A CAM implements a fully-parallelized linear search over a collection of N binary w-bit words. The parallelized search is enabled by broadcasting the values of a w-bit search key throughout the entire matrix of $N \times w$ cells. The search line driver takes the search word and distributes its bits across the array of searchlines. Each cell in the matrix is connected to its appropriate searchline. When the cell receives its corresponding search bit, a comparison is made between the value stored in the cell and the value received on the searchline. Each stored word in the CAM has an associated *matchline* that provides indicator functionality. A *match* exists when each of the search bits received by the cells of a word equal the bits stored by the respective cells. A match is indicated by sending a logical-high (truth) value from the word's matchline to a priority encoder. A no-match condition is indicated by sending a logical-low (false) value from the associated matchline to the priority encoder. In most search applications that utilize CAMs, a prioritized list of words are assumed. Consequently, when a search key matches multiple content locations, the highest priority match is selected via the priority encoder. However, other multi-match decision resolver strategies can be employed.

The ternary content addressable memory provides similar search operations as the CAM. However, TCAMs are able to store elements in the cells that represent the ternary set $T = \{0, 1, *\}$ where $*$ is the canonical wildcard symbol. A word stored in a TCAM that contains the wildcard symbol for the i^{th} cell implies that the cell matches any search bit submitted to the cell. TCAM cells are similar to CAM cells, but TCAMs require two SRAM cells for its ternary logic as well as a slight variation in its transistor-based comparision logic. However, the overall operations provided by the searchlines and matchlines are the same–search keys that match all of the cells of a stored word result in a positive match indication and if one or more characters of the search key do not match the contents of its associated cell, a negative match indicator is generated.

The ternary encoding capabilities of TCAMs are directly suited to encode packet filter prefixes. The fully parallelized linear search enabled by TCAMs reduces the $O(N)$ search complexity to $O(1)$. However, TCAMs suffer from significant amounts of power consumption [108]. During a search, every circuit element of the TCAM is actively *processing* information. According to Taylor [147], a TCAM chip performs significantly worse than a comparative SRAM chip. Taylor provides the following comparisons between TCAMs and SRAMs (statistics based on data collected circa 2002):

- TCAM access times can be on the order of three times slower than SRAMS.
- TCAMs per bit power consumption is approximately 150 times the per bit power consumption of SRAMs

Research approaches investigating methods to reduce TCAM power consumption is generally categorized as architectural-level and circuit-level approaches. Circuit-level approaches seek to modify the transistor-based structures and the various interconnect mechanisms to reduce effective capacitance and resistance associated with the power consumption of the device. Circuit-level techniques that have been suggested for CAM/TCAM power reduction include: low-swing matchline voltage [74], selective precharging [171], sensing current races [10], and pipelined matchlines. Architectural-level techniques consider algorithms and precomputation methods seeking to reduce the width of encoded words and/or the total number of words.

The negative performance impacts resulting from the high power consumption of TCAMs has inspired the design of new hardware-based packet classification algorithms that utilize pipelined packet processing constructs. The majority of these SRAM-based pipelined packet processors are concerned with mapping the nodes of prefix tries to pipelined processing stages.

Binary words, which are sequences of bits over some specified width w defined by a given hardware architecture, can be stored within binary trie data structures. Binary tries are specified by a stride factor, c, constrained as $1 \leq c \leq w$. The stride groups the bits of a binary word into *chunks*, which determine the branching width of the trie's tree structure. Chunks are used at each level of the trie for determining the appropriate branch to take during the traversal of the tree during a search.

A unibit trie (UBT) is given by $c = 1$, and multibit tries (MBT) result from $1 < c < w$. For $c = w$, a single-level tree is formed and represents a direct table lookup implementation. However, direct lookup tables require 2^w unique addresses and memory consumption grows as $O(2^w)$. Consequently, direct table lookup is constrained by the memory capacity of its hardware architecture. Strides are not required to be fixed and can be specified by stride sequences $C = (c_1, c_2, \ldots, c_d)$ where $w = c_1 + c_2 + \ldots + c_d$. However, the assumption in this discussion is that multibit tries are used where $1 < c < w$ and $c|w$. Further, it is assumed that both c and w are powers of 2.

In general, tries are effective *data structure pipelining* constructs [16, 55]. A straightforward methodology to map MBTs to hardware is to allocate each level of the its tree structure into the memory subsystem of a pipelined hardware system. Each stage of the pipeline has local processing elements and memory storage for the binary words of its corresponding tree level. The hardware algorithm directly implements the trie traversal algorithm during a search. Figure 20 illustrates a generic trie-to-hardware (TTH) mapping scheme (Table 9 contains data represented by the trie of Figure 20.)

The left portion of Figure 20 contains a three-level trie representing binary words with $w = 4$ and a stride of $c = 2$. The right portion of the figure reveals a 3-stage hardware

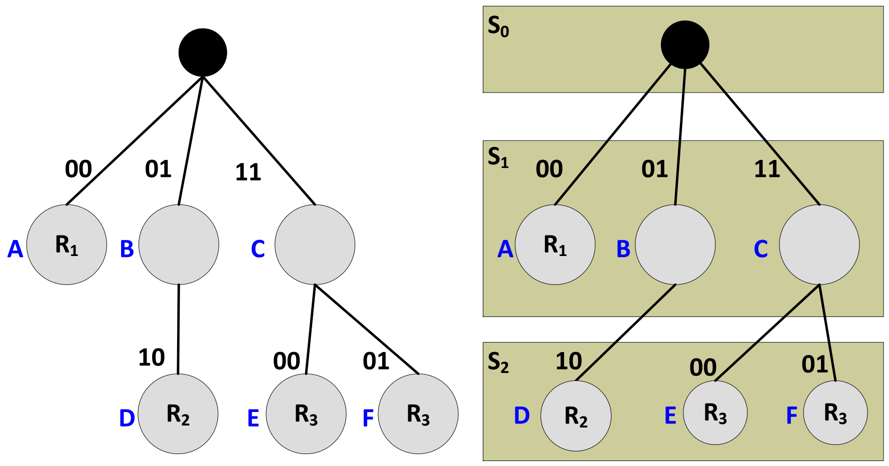


Figure 20: Generic trie-to-hardware mapping techniques.

Table 9: Data represented by the trie of Figure 20.

Encoded Words	**Binary Prefix**	**Reference Label**	**Node Sequence**
0000 0001 0010 0011	00*	R_1	(A)
0110	0110	R_2	(B, D)
1100 1101	110*	R_3	$\{(C, E), (C, F)\}$

pipeline. During each clock cycle, each stage S_i processes the i^{th} level of the trie. Each stage is processing in parallel, and a search result is returned in time proportional to the number of stages.

Pipelined trie data structures can significantly reduce the amount of time required to search memory for a stored binary word, i.e., the prefixes of a firewall or router. The pipeline architecture amortizes memory access times when performing extensive search operations (back to back search processes). The goal is to increase search throughput such that one search result per clock cycle is achieved. The latency of a particular search is related to the depth of the pipeline and the average memory access time of each pipeline stage. Although tries are effective pipeline data structures, a number of challenges must be addressed when employing these data structures and their associated search algorithms. Two particular challenges faced by these hardware algorithms include uniform memory distribution over the pipeline stages and power consumption [57, 70, 69]. The uniform memory distribution problem requires solutions that map the trie nodes to pipeline stages with an approximately even distribution of memory consumption. According to Basu and Narlikar [16], the uniform memory distribution problem is the dominating issue for trie-based pipelined architectures. Another issue faced by these algorithms is the backtracking problem associated with MBTs over multiple dimensions. TTH algorithms use the same search logic as their software-based MBT counterparts. Pipelining does not address the backtracking problem. Therefore, these algorithms must provide methods to mitigate the adverse performance effects caused by backtracking.

3.4 Cyberattack Detection Algorithms with Intelligent Systems

In this section, an overview of related work addressing the cyberattack detection problem with computational intelligence systems is provided.

Lichodzijewski et al. [89, 88] developed a host-based anomaly detection system using a hierarchical SOM. Their system uses a first-level SOM that provides feature detection based on six variables extracted from TCP connections. The second-level SOM acts as a feature integrator by combining the feature detections generated by the first-level SOM. The second-level SOM combines the detected features of the level-one SOM and the topological clustering aspects of the SOM algorithm provide visual indicators that can be utilized by a security administrator to make final detection decisions. Jirapummin et al. [71] introduced a network-based intrusion detection system utilizing a hybrid neural network. Their system employs a SOM along with a resilient propagation neural network (RPN). Their system uses the SOM to cluster and visualize audit data. Moreover, weight vectors of the SOM nodes are supplied as input to an RPN for detection purposes. The RPN uses the weight vector components from the SOM nodes as its input, and the output nodes of the RPN provide a multi-class classification decision. Rhodes et al. [119] proposed a multi-layer SOM for network-based intrusion detection. Their algorithm assigns individual SOMs to various protocols within the TCP/IP stack. Their goal was to categorize and classify attacks that target specific vulnerabilities within the TCP/IP protocol layers. Their results indicate that a layered SOM performs better than a single SOM classifier.

Bankovic et al. [15] developed an intrusion detection system using self-organizing maps and reputation systems for wireless sensor networks. They use the SOM to assign reputation scores to the sensor nodes of a wireless sensor network. Moya et al. [98] describe a technique similar to [15] for protecting supervisory control and data acquisition (SCADA) sensor systems for industrial control applications. Industrial control systems such as SCADA networks and infrastructure systems such as the *smart grid* have not received much attention in terms of cybersecurity as compared to traditional computer and network systems. Venayagamoorthy [156] argues that advanced computational intelligence systems will be required to enable intelligence functionalities and security of smart grids and cyberphysical infrastructures. Moreover, recent events such as the outbreak of the *stuxnet* worm, which specifically targeted industrial SCADA systems, have shown the importance for securing these critical systems [24]. Moya et al. [98] along with others such as [168, 35, 40, 32] have shown the importance of protecting existing and emergent cyberphysical systems and infrastructures with advanced detection and cybersecurity methodologies.

Amini et al. [6] compare self-organizing maps and adaptive resonance theory neural networks for network-based intrusion detection. Their results showed that neural networks based on learning methods from adaptive resonance theory can achieve better intrusion detection classification performance than networks based on single-layer self-organizing maps.

Artificial neural networks (ANNs) have configuration parameters for specifying network properties such as its topology, learning rates, learning algorithms, and activation functions. This selection process is usually manual and requires trial-and-error to find the best alternatives. In [101], Mukkamala and Sung use the CUP99 dataset to evaluate the classification performance of ANNs constructed with twelve types of learning methods. From their experiments, networks trained with the resilient back propagation algorithm provided the best classification accuracy. In another series of studies, Mukkamala et al. [99, 100] compare the classification performance of neural networks and support vector machines (SVM) for misuse detection. In their experiments, neural networks and support vector machines produced classification accuracies and detection rates that were not significantly different. However, they also compared the time required for each algorithm to be trained, and it was shown that the SVM learned within orders of seconds whereas the neural networks required learning time on the order of hours. Horng et al. [60] proposed an intrusion detection system based on SVM, hierarchical clustering, and feature selection. They use a hierarchical clustering approach to reduce the number of audit data features. Once the features have been selected with their clustering algorithm, a reduced dataset is constructed from the selected features. This reduced dataset is then used to train four separate SVMs, which function as the classification mechanism for intrusion detection.

Engen [39] proposed several machine learning methodologies to enable robust network-based intrusion detection. Engen explores the idea of evolutionary neural networks (ENN) in an effort to minimize classification errors of intrusion detection systems. Engen's study considers the effect of different fitness functions when evolving neural network weight vectors with genetic algorithms. His investigation shows that an ENN can achieve good classification performance, even when faced with highly imbalanced training data. Others have considered ENNs as classifiers for intrusion detection. Han and Cho [52] implement an anomaly detection system based on ENNs. They use a genetic algorithm to evolve artificial neural networks with audit data extracted from sequences of operating system calls. In [59], a network-based intrusion detection system is proposed by Hofmann et al. where genetic algorithms evolve radial basis function (RBF) neural networks.

Didaci et al. [36] consider neural network ensembles for intrusion detection. They design their system by categorizing the training data and assigning a neural network to each category. To evaluate their system, they use the CUP99 dataset and assign categories based on the basic TCP features, content features, and network-based traffic features of the dataset. Moreover, Didaci et al. only evaluate their classifier against the FTP service connections contained within the dataset. In their approach, majority voting, averaging, and belief functions were considered for generating the output decision of the neural network ensemble. Belief functions were claimed to provide the best results from their experimentation. Engen [39] also studies various ensemble methods for intrusion detection. His investigation considers the effects of multi-objective genetic algorithms and how they can be used to evolve a collection of ensembles. Engen's goal is to evolve ensembles where each has its own classification criteria and performance trade-offs.

In [114], Perdisci et al. introduce an anomaly detection system with SVM ensembles for payload-based attacks. An ensemble of *one-class* SVMs classify packet payloads and a majority voting rule combines the ensemble outputs. Perdisci et al. [113] extend the methods of [114] and focus their attention on attacks with packet payloads containing *shell code* as well as attacks that utilize polymorphism, blending, [28] and polymorphic-blending [41]. They use a method based on 2-grams to extract payload features for an SVM ensemble and experiment with different methods such as averages, products, maximums, minimums, and majority voting to combine the output results of their ensemble. Their algorithms had higher classification performance when ensemble outputs were combined with averages, products, and minimums.

Abraham and Thomas [1] study the efficacy of ensemble systems as applied to intrusion detection systems. Their algorithm uses a decision tree classifier to perform feature selection. After the feature selection process, multi-class audit data is partitioned into groups of single-class audit data, and each single class is assigned to a classifier that performs optimally for the given class. During runtime, data is fed to the ensemble and results are combined with methods such as majority vote.

Kumar and Selvakumar [84] propose a neural network ensemble to detect distributed denial of service attacks (DDoS). Their classification algorithm, referred to as RBPBoost, uses an ensemble of resilient back propagation (RBP) networks along with a hierarchical collection of methods to combine the ensemble outputs. They view the DDoS attack problem as a two-class classification problem consisting of attack classes and normal classes. They partition the training data into these two classes and then create subsets of data for the partitioned classes. RBP neural networks are assigned to each data subset during training. The outputs of the neural network ensembles are first combined via weighted majority vote, which generates a vector of decision results containing a majority vote for each neural network ensemble. The weighted majority votes for the attack class and the normal class are then combined with a weighted product rule. The output of the weighted product rule procedure is then processed by a Neyman-Pearson cost minimization method to determine the final classification result.

3.5 Summary

In this chapter, an overview of related works in distributed security architectures, packet classification algorithms, and computational intelligence systems as classifiers for cyberattack detection systems was presented. In the next chapter, a distributed firewall and active response architecture are introduced.

CHAPTER IV

CYBERATTACK DEFENSE WITH THE DISTRIBUTED FIREWALL AND ACTIVE RESPONSE ARCHITECTURE

Current cybersecurity architectures and infrastructures continue to be dominated by quasi-monolithic and perimeter-based security mechanisms that aim to provide security services for elements comprising an organization's overall cyber environment. For example, perimeter firewalls and intrusion detection systems provide network access control and detection services for resources contained within the network boundaries of an organization. Although individual hosts within the organization typically employ host-based firewalls and other services such as virus scanning software, these systems very seldom operate as collaborative and coordinated mechanisms. A modern framework is needed that guides the design of highly modular, adaptable, and integrable security architectures where ***any*** cyber device is capable of actively participating within the organization's overall security infrastructure. In this chapter, a distributed firewall and active response architecture is proposed that strives to alleviate various difficulties faced by existing cybersecurity systems.

4.1 Motivation

Although a number of security architectures have been proposed for various distributed security aspects, several problems remain and need to be addressed. An urgent need is the design of security architectures and corresponding security infrastructures that enable the participation of cyberattack detection and prevention for each cyber device within the cyber environment. Auxiliary needs to support such a distribute security system include the means for seamless exchange of cybersecurity information and modular, adaptive, and integrable frameworks that allow the overall system to be scalable to the growing needs of the cybersecurity problem and to provide the flexibility to incorporate the security mechanisms most appropriate for the particular cyber devices within the system.

The design of most distributed security systems proposed thus far are based on perimeter-type defense mechanisms. However, the Internet landscape is undergoing a transformation whereby a vast majority of Internet-connected devices, infrastructures, services, and data are characterized by dynamic properties. Dynamic properties such as mobility whereby cyber resources travel through varying and disparate networks render organizational perimeter defenses less effective. For example, entire information technology infrastructures can be deployed across any number of cloud computing data centers, and these infrastructures can be relocated seamlessly, elastically, on-demand, and non-disruptively. New solutions addressing the security challenges associated with these emergent Internet dynamics and dynamical systems are needed. In particular, the canonical methods associated with topology-dependent perimeter defenses will soon reach a point of diminishing returns as Internet-connected cyber resources become less stationary and more dynamic.

Traditional cybersecurity systems such as firewalls and attack detection systems are commonly implemented as perimeter mechanisms operating independently with no sharing and coordination of new information describing attacks and requiring configuration to counter new attacks by human administrators. Although these traditional technologies and associated security-best-practices are still necessary, they are no longer sufficient [129, 161].

Traditional security technologies are characterized by defense-in-depth, isolated, non-cooperative, and administratively-reactive strategies. Defense-in-depth provides a sound defense methodology where many forms of defense mechanisms are deployed concurrently by an organization in order to increase the security and reliability of the cyber environment. However, isolated, non-cooperative, and administratively-reactive techniques have become ineffective [68, 92, 2, 43]. Isolated and non-cooperative implies that security mechanisms operate within a confined *awareness domain*, and knowledge related to new security occurrences are not shared outside of the awareness domain. Isolated is implied by confinement, and non-cooperative is implied by not sharing knowledge with other awareness domains. This imposes a significant impediment in terms of efficient, global-scale Internet security [133, 7]. Lastly, administratively-reactive relates to techniques whereby human security administrators react to newly discovered security events instead of automated, computer-driven attack response systems. Human reaction and remediation times are far too slow to counter the impact of many classes of cyberattacks [141]. The sheer volume of modern day attacks, their permutation capabilities, their sophistication, and their speed cause isolated, non-collaborative, administratively-reactive response and remediation methods to be futile.

The distributed security mechanism proposed in this chapter is centered on the idea of providing globalized Internet security with *preemptive protection methodologies* using a modular, adaptive, and integrable distributed firewall and active response architecture along with an associated security management infrastructure. Preemptive protection is a process whereby an entity within a network is protected from future attacks based on observations made by other entities within the network. It is inherently a collaborative process of collecting and sharing information related to observations of attack events. Preemptive protection is achieved by way of observing attack events, dynamically creating a blocking enforcement filter, distributing blocking enforcement filters to entities within a collaborative awareness domain, and applying the blocking enforcement filter at each entity in the collaborative domain. This collaborative methodology is referred to as the global preemptive protection process, and it is described by the following set of generic steps: **Measure, Detect, Classify, Locally Isolate, Disseminate, and Globally Isolate**.

In the global preemptive protection (GPP) process, an entity within the collaboration domain observes attack events via measurements, detection, and classification. Once an event is classified as an attack, a blocking enforcement filter (BEF) is created. Local isolation of the attack at the observing entity is achieved via activation of the BEF within its blocking enforcement point (BEP). In the context of GPP, local isolation can be viewed as a form of self-preservation. The entity observing an attack must protect itself first. Then, it proceeds with helping to protect the remaining entities belonging to the collaboration domain. Once local isolation of the attack is complete, the entity disseminates the BEF to other entities within the collaborative domain, who in turn implement local isolation (self-preservation) via activation of the new BEF. Once all entities of the collaborative domain receive and activate the BEF, global isolation of the event has been achieved for the domain.

4.2 Functional Characteristics of Distributed Network Defense Mechanisms

Firewalls are effective mechanisms that secure computers and networks. A firewall is a device that processes packets flowing between communication interfaces and makes security decisions based on content extracted from packet header fields along with filters defined by a security policy. A firewall can be implemented as software running on a host computer or as a dedicated hardware device within a computer network.

Firewalls are common systems belonging to many modern security implementations. Indeed, firewalls have become *ubiquitous.* Firewall ubiquity is actually the main factor contributing to the selection of firewalls as the *blocking enforcement point* for the distributed security architecture proposed in this chapter. However, the firewall alone will not suffice. An active defense mechanism requires more than just blocking enforcement. It requires an entire collection of diversified technologies working in unison to achieve a common objective.

In this section, a firewall collaboration framework [149] is presented. The firewall collaboration framework is a generic framework that enumerates the basic functionality and desirable characteristics needed by distributed and collaborative defense mechanisms that utilize firewalls as blocking enforcement points.

4.2.1 The Firewall Collaboration Framework

The firewall collaboration framework (FCF) is a collection of basic building blocks seeking to categorize and describe the desirable and functional characteristics of distributed network defense mechanisms. The framework seeks to combine the common threads and ideas that have been published within the field of distributed network defense into a taxonomy of functional characteristics that should be considered during the design stages of distributed and collaborative firewall systems.

The framework is founded on the notion of independent firewall systems that join together and form a unified federation. The federation of firewalls collaborate with each other and share information related to newly observed attacks. Key to observing new attacks is the capability to measure, detect, and classify audit data. Moreover, the framework only considers cyberattacks that occur over a network, i.e., network-based cyberattacks. As a result, the framework assumes that firewalls within the federation have computational resources capable of basic functionality such as the following:

1. Reading (capturing) packets from the network.
2. Extracting audit event data from captured packets.
3. Processing audit event data with attack classification systems.
4. Associating IP addresses of attack sources with audit event data and classification results.
5. Inserting and/or removing firewall filters that deny all traffic to or from the IP address of a known attack source.

The global pool of information shared within the federation may contain data such as attack signatures, malicious activity profiles, firewall rules, access control lists, IP address blacklists, and other types of security information. Once new information regarding malicious activity is obtained either in real time with an attack detection system or by an administrator that learns of new attacks, the new information is distributed to firewall units belonging to the federation. The collaborating firewalls can then update their rule bases and policies to incorporate this new information.

Conceptually, firewall members of the federation can be viewed as distributed cyber-attack sensors that measure, detect, and classify traffic belonging to cyberattack sources. As more firewall nodes join the federation, the usefulness or utility of the system increases because a larger number of members distributed across the Internet increases the probability of observing attack activities, which increases the probability that a member can be protected from attacks via the sharing of event information.

Figure 21 illustrates the six functional components of the FCF, and these components are described as follows.

Federation Management: The purpose of federation management is to control mem-

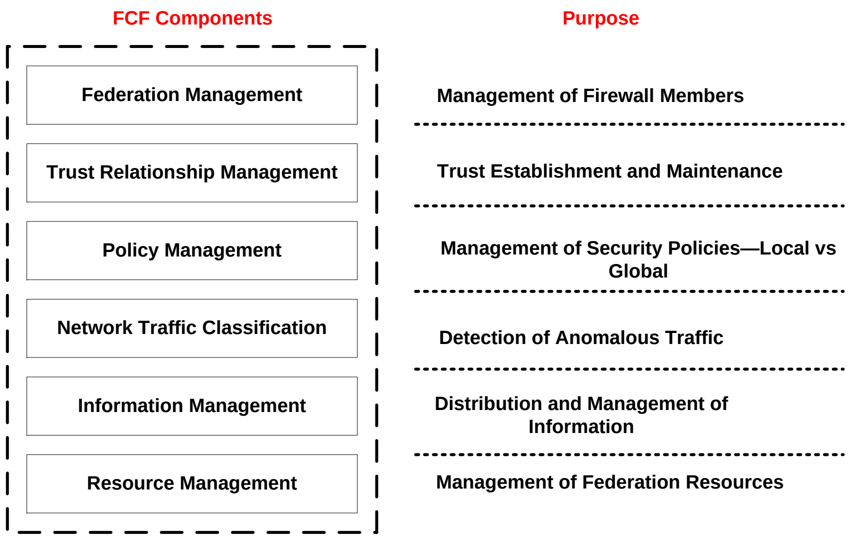

FCF Components
Purpose
Federation Management
Management of Firewall Members
Trust Relationship Management
Trust Establishment and Maintenance
Policy Management
Management of Security Policies—Local vs Global
Network Traffic Classification
Detection of Anomalous Traffic
Information Management
Distribution and Management of Information
Resource Management
Management of Federation Resources

Figure 21: The FCF functional components.

bership of new firewall elements to the federation. This component is responsible for establishing an initial trust between the firewall and the federation. If membership is not carefully controlled, denial of service attacks would be easy to orchestrate. For example, a rogue firewall could join the federation and inject falsified policies into the system. This could be used to deny service to members of the federation and their respective networks. Further, it could be used to allow unauthorized access to federation networks.

Trust Relationship Management: Once firewall elements have been allowed to join the federation by establishing the initial trust between firewall element and the federation, the Trust Relationship Management (TRM) component will be used to maintain the member relationships. TRM should address issues such as information authentication and credential management. Information authentication is needed to prevent denial of service. Credential management is needed because of the possibility that members could become rogue after being allowed to join the federation. TRM and federation management components are tightly coupled with each other.

Policy Management: In most organizations that follow security best-practices, security policies are designed based on business and end-user needs. These policies will differ from organization to organization. For this system to operate effectively there is a need to differentiate local policy from global policy. The local policy refers to rules and specifications that are specific to an organization. For example, allowing organizational users to browse the web. Global policies will be those that are distributed to the collaborating elements in the federation. Policy management is also concerned with issues such as policy decay (removing stale policies) and decision management (should a newly created policy be instantiated immediately or after some delay).

Network Traffic Classification: Network traffic classification is concerned with the when, where, how, and why of processes and systems implemented by federation members that measure, detect, and classify attacks. Systems that implement the measure, detect, and classify functionality in an automated and autonomic fashion are desired. As stated earlier, network administrators can update the rules and policies of a collaborating firewall. But, the time interval between the start of an attack and the human detection (using network analysis tools or security bulletins) of the attack is usually too large.

Information Management: The information management component should govern the way in which information is transported throughout the federation. This mechanism should address issues such as centralized versus peer-to-peer information distribution. The information management component should address issues such as data caching and staleness. This component should also address information confidentiality and integrity. This will be accomplished with encryption mechanisms, and this component will govern the types of encryption techniques that are used throughout the federation.

Resource Management: The resource management component should govern how cyber resources are deployed and utilized within a federation such that an effective and efficient outcome is achieved by the overall system.

4.3 The Distributed Firewall and Active Response Architecture

In this section, the distributed firewall and active response architecture (DFAR) [150, 151] is presented.

4.3.1 High-level Description

DFAR's design was guided by the components and characteristics outlined by the firewall collaboration framework. DFAR is founded on the idea of hosts within a trusted domain of administration (TDA) that detect cyberattacks and create firewall blocking filters against attacking sources. Blocking filters are shared with federation members belonging to the trusted domain of administration. The TDA is defined to be the set $\Sigma = \{S_1, S_2, \cdots, S_M\}$ of hosts under the control of a single administrative authority. Once a blocking filter (i.e., security policy) is created by a local TDA host or received by a local TDA host from a TDA neighbor, the policy is translated into a firewall filter that denies access to or from the anomalous host.

The fundamental design principle of the architecture is guided by the following edict:
Once a source IP address has been classified by a cyberattack detection mechanism as being untrustworthy, ***deny*** *all access* ***to*** *or* ***from*** *the attacking host for* ***all*** *members belonging to the trusted domain of administration.*
Some key points of this edict should be understood. First, the design principle states that DFAR is based on the notion of firewall blacklist enforcement. Untrustworthy IP addresses, which can be enumerated via cyberattack detection systems or retrieved from global blacklist repositories, are encoded as blocking enforcement filters within the local firewalls residing on TDA members. Second, blocking enforcement filters should be implemented at the inbound ***and*** outbound interfaces of the firewall. This ensures that packets sent by attack sources are blocked at the firewall's inbound interface. Moreover, it ensures that TDA hosts cannot initiate communication with the attack source as outbound traffic destined to the source will be blocked.

In essence, a new firewall filtering model is defined by the DFAR architecture. In particular, DFAR defines a firewall-based blacklist enforcement mechanism, which is illustrated by Figure 22. The blacklist classification module of the figure represents, conceptually, the

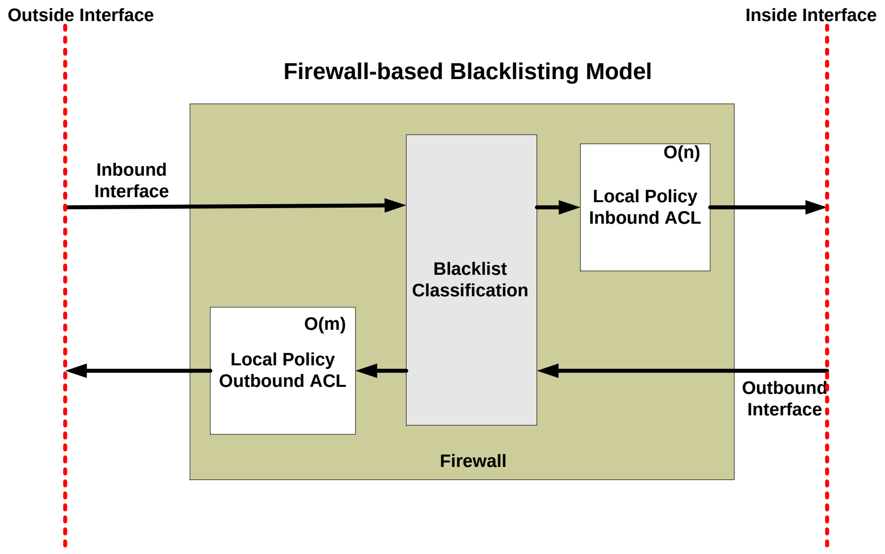


Figure 22: The DFAR firewall-based blacklist enforcement model.

location of blacklist blocking enforcement for TDA hosts. Inbound packets received by the outside interface are delivered to the blacklist classification module where the *source* IP address is compared with the encoded blacklist. If a match is found, then by definition the packet must be denied (blocked) by the firewall. Otherwise, the packet is delivered to the firewall's local policy inbound access control list (ACL) classification module. Outbound packet processing proceeds in a similar fashion. First, packets generated by the local host are delivered to the blacklist classification module. The *destination* IP address of an outbound packet is compared with the encoded blacklist. If a match is found, then the packet must be blocked by the firewall. Otherwise, the packet is delivered to the firewall's local policy outbound ACL classification module.

Figure 23 will be used to facilitate a discussion describing the behavior of the proposed distributed firewall and active response architecture. From Figure 23, the TDA is composed of $\Sigma = \{S_1, S_2, S_3, S_4, S_5\}$ where each S_i is a TDA host possibly offering network services to external Internet clients. Further, it is assumed that each host has a host-based firewall, attack detection software, and procedures that dynamically generate and install new firewall filters once attack detectors classify cyberattack sources.

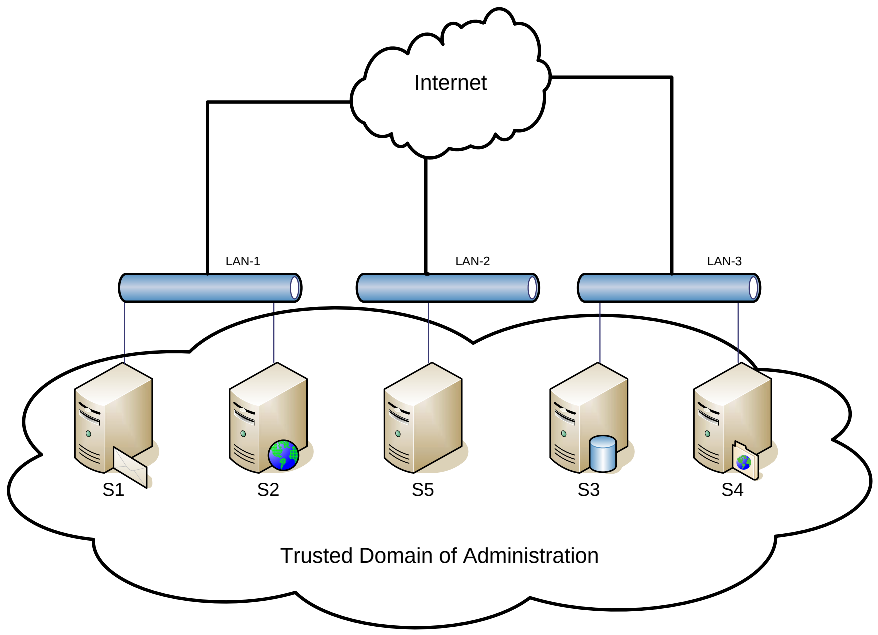


Figure 23: A trusted domain of administration.

As seen in Figure 23, the TDA can span across multiple networks (i.e., LANs). The only requirement is for TDA hosts to be administered by a single administrative authority. There is a purpose for this requirement and the definition of the TDA. Trust is a critical aspect of the architecture. If trust is neglected, there exists a non-zero probability that false policies can be injected into the system such that denial of service could be issued on random hosts, i.e., injecting blocking enforcement filters that deny access to/from a non-malicious source.

A simple description of the behavior of the distributed firewall and active response architecture follows. Each S_i has a detection process monitoring for network-based cyberattacks. Once an attack is detected, the offending IP source address is added to a security policy database and a filtering rule is created within its local host-based firewall that denies all access to or from the offending IP address. Then, the host sends this information to each of its neighbors within the TDA so that they can also add a deny rule for the offending address. This process is an act of distributed active response that implements preemptive protection. Consider the case where a malicious node performs a Secure Shell (SSH) dictionary attack that targets S_1. Once S_1 has detected the attack, a blocking rule will be created for the offending IP address, and after distribution the other neighbors will have a blocking rule too. At some time in the near future, the same malicious node initiates a sequence of attacks (not necessarily the SSH dictionary attack) against S_5. However, S_5 had previously added a blocking enforcement filter based on the distributed security policy received from S_1. Therefore, S_5 will not be impacted by the attacks. Hence, S_5 has been preemptively protected.

The behavior of DFAR is similar to a quarantine process. The deny to/from edict of DFAR's design principle induces quasi-quarantine whereby cyberattack sources are isolated from TDA members at the network interface for each member, i.e., isolation occurs at the host-based firewalls of TDA members. Moreover, isolation can occur for administratively disjoint TDAs that form collaboration federations via the establishment of inter-TDA relationships.

The behavior described thus far can be regarded as *intra*-TDA operations. However, similar processes can be established such that inter-TDA operations can be achieved. Figure 24 illustrates the inter-TDA concept. From this figure, it is assumed that a TDA controller

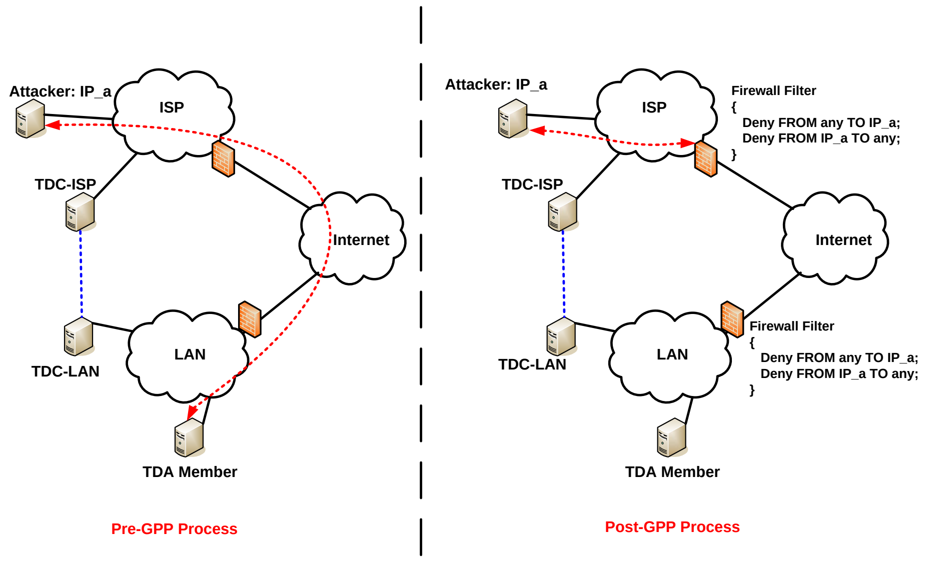


Figure 24: Quasi-quarantine process with DFAR.

(TDC) is responsible for inter-TDA transfer of information. In the example implied by the figure, the source address IP_a is actively attacking a TDA member controlled/administrated by the machine named TDC-LAN. The left side of the figure represents a point in time before the GPP process occurs. However, at some time after IP_a begins its attack, a detection mechanism observes the attack event and firewalls throughout the TDA controlled by TDC-LAN implement a blocking filter for IP_a. Moreover, the example assumes that a trust relation has been established between TDC-LAN and TDC-ISP, which happens to be the Internet service provider for the attack source. Information describing the attack is transferred to TDC-ISP via TDC-LAN, and, as a result, blocking enforcement filters are instantiated throughout the ISP's network. In this example, the ISP firewalls block all outbound traffic generated by IP_a and all inbound traffic destined to IP_a at the ISP's Internet point of presence, thereby completely isolating the entire Internet from the attack source.

4.3.2 Architectural Description

Figure 25 provides a block diagram of the proposed distributed firewall and active response architecture. DFAR is a distributed system of nodes that implement the TDA host architec-

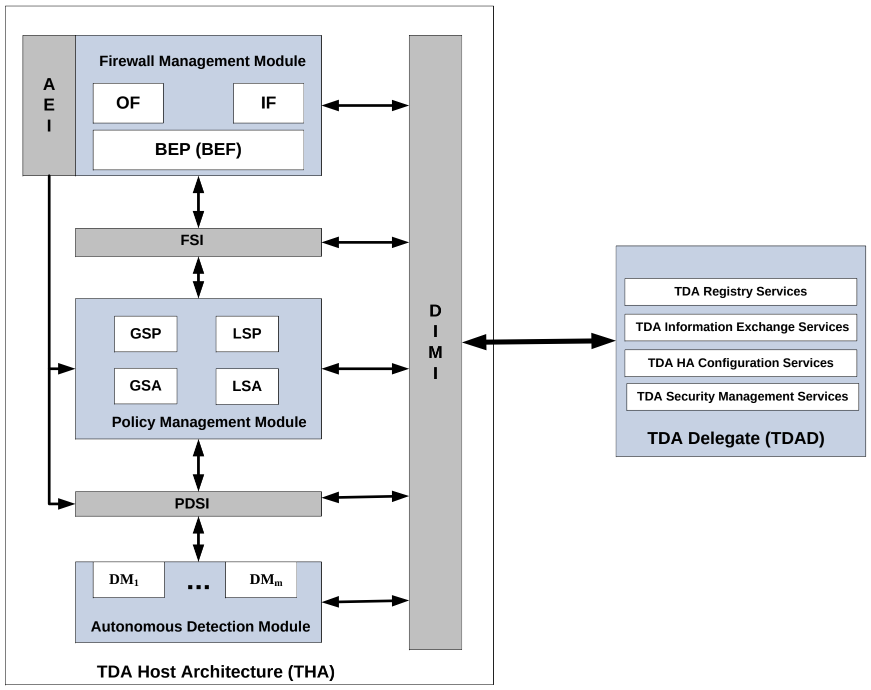

A E I
Firewall Management Module
OF
IF
BEP (BEF)
FSI
GSP
LSP
GSA
LSA
Policy Management Module
PDSI
DM1
...
DMm
Autonomous Detection Module
TDA Host Architecture (THA)
D I M I
TDA Registry Services
TDA Information Exchange Services
TDA HA Configuration Services
TDA Security Management Services
TDA Delegate (TDAD)

Figure 25: The distributed firewall and active response architecture.

ture (THA). The TDA is viewed as a distributed infrastructure with centralized interfacing and management. In other words, the TDA is a distributed system with centralized control. The TDA Delegate (TDAD) is responsible for centralized interfacing and security management of the overall system. DFAR centralized interfacing and security management seeks to solve various challenges that prevent large-scale distributed systems integration and information exchange for systems having large numbers dynamic cyber resources, i.e., mobile (roaming) hosts.

The TDA host architecture is designed to be modular with communication between THA modules as well as between the end host and the TDAD occurring via communication interfaces. The THA is comprised of three primary modules and four inter/intra process communication interfaces. The primary THA modules are the following:

- Autonomous Detection Module
- Policy Management Module
- Firewall Management Module

The THA communication interfaces include the following:

- Audit Export Interface
- Filter Specification Interface
- Policy Description Specification Interface
- Distributed Infrastructure Management Interface

Two terms are used extensively throughout the description of the distributed firewall and active response architecture and its associated behaviors. These two terms are *host* and *local*, and the precise semantics of these terms are defined as follows with respect to DFAR. A DFAR host is ***any*** cyber entity implementing an instance of the TDA host architecture (as illustrated by Figure 25 and described below). For example, a host can be a complete physical computing system such as a desktop computer, a server, or a laptop, or it can be a virtual system such as a software process running within a physical computing system.

The term local refers to cyber resources such as TDA hosts or cyber components protected by TDA hosts that are invariant to network topology.

4.3.2.1 Autonomous Detection Module

The autonomous detection module (ADM) is a collection of independent autonomous cyberattack detection modules (DM) that are active on the local system and the collection of rules that define how the ADM must interact with the policy management module. The autonomous detection module (ADM) contains zero or more independent cyberattack detection modules DM_i that are responsible for monitoring audit data collected by the local host and/or collected from collaborating audit data collection systems. These detection modules are responsible for real-time detection of cyberattack events.

ADM detection modules implement the measure, detect, and classify functionalities of the global preemptive protection process. The ADM can provide network-based audit event data to the various detection modules via (1) packet capture processes that directly interact with the network interface of the local host, (2) via audit event data collected by the firewall management module and delivered over the policy description specification interface, or (3) via audit event data provided by the TDAD over the distributed infrastructure management interface.

The DM can be as simple as a process monitoring the local host for failed login attempts to as complex as a full scale IDS deployment. Regardless, the ADM was designed to be modular and extensible. Cyberattacks are dynamic and continuously evolving. As a result, detection modules should be capable of adapting themselves in an autonomic fashion in order to adjust their detection behaviors. Computational intelligence systems are well suited for the emergent and adaptive changes that are required of detection applications. In a subsequent chapter, several computational intelligence systems are evaluated as detection modules that can be employed within the ADM.

Because cyberattacks are significantly diverse, it is not feasible to have a single, monolithic detection module capable of reliable classification for all possible attack scenarios. As a result, the ADM is designed to be modular such that many different detection modules operate, independently if needed, on a given host. The goal is to enable different types of independent, target-specific, autonomous attack detectors to reside on a single host, thereby providing a more robust defense in depth strategy for the local host and the overall TDA.

4.3.2.2 Policy Description Specification Interface

The policy description specification interface (PDSI) provides local communication interfacing between the policy management module, the attack detection module, and the audit export interface of the firewall management module. The PDSI delivers new blocking enforcement filters generated by the ADM to the policy management module. Moreover, it receives audit event data from the audit event interface and delivers audit event data to the attack detection module.

The PDSI defines a globally recognized policy description that all members of the TDA interpret with the same meaning, i.e. it is a formal syntax describing a policy. The following policy description specification (PDS) is defined by the PDSI:

PDS := (IP_x, P_x, S_i)

The syntax describes a 3-tuple containing the IP address of the attacking host (IP_x) discovered by the ADM, the policy action to take by the TDA against the attacking host (P_x), and the host identifier (S_i). The host identifier is some value that uniquely identifies the host that generates the PDS. P_x is a policy action construct defined as P_x := DENY_TO_-FROM. As an example, the PDS given as (10.1.2.3, DENY_TO_FROM, 128.61.209.23) can be interpreted as the following. The TDA member 128.61.209.23 observed network attacks generated by the source 10.1.2.3 and has initiated a blocking enforcement policy stating to deny all traffic to or from the attack source. The PDSI is responsible for delivering the PDS to the policy management module associated with the local host and also delivers the PDS to the TDAD via the DIMI.

4.3.2.3 Policy Management Module

The policy management module (PMM) is responsible for management of global and local security policies (GSP, LSP) for the local host. The PMM also manages the archival of global security audits (GSA) received from the TDA via the DIMI along with archival of local security audits (LSA) generated by the firewall management module and the autonomic detection module.

Global security policies and audits provide information representing views from distributed hosts comprising the TDA. This *global perspective* can be used by the host to make, in conjunction with outputs received by the ADM and the AEI, more informed decisions when new cyberattacks are discovered.

4.3.2.4 Filter Specification Interface

The filter specification interface (FSI) provides a communication mechanism that translates new PDSs received from the PMM into the filtering language of the blocking enforcement point (BEP) that implements the firewall functionality for the local TDA host. The FSI creates a blocking enforcement filter (BEF) for the outbound intefaces (outbound filters, OF) and the inbound interfaces (inbound filters, IF). This interface allows for heterogeneous firewall systems whereby (1) the local host may implement multiple firewall instances if multi-firewall functionality is needed and (2) the TDA is not required to implement any particular type of firewall.

4.3.2.5 Firewall Management Module

The firewall management module (FMM) is responsible for dynamically configuring firewall filters when rule structures are received from the FSI. In DFAR, firewalls residing on local hosts implement the local isolation functionality of the global preemptive protection process. The FMM inserts new inbound filters and outbound filters based on information provided by the FSI. The FMM also collects audit data based on traffic flowing through the outbound/inbound interfaces and delivers the audit data to the PMM and/or the PDSI via the audit export interface (AEI). This audit data is archived via the LSA functionality of the PMM and can be delivered to the ADM detection modules by the PDSI.

4.3.2.6 Distributed Infrastructure Management Interface

The distributed infrastructure management interface (DIMI) is the key communication interface that integrates the host members of the TDA into the overall system. It provides direct communication between THA elements and the TDAD, thereby providing collaboration capabilities for the system. The DIMI will be explained further in conjunction with a description of the TDAD.

4.3.2.7 TDA Delegate

The TDA delegate (TDAD) is responsible for centralized interfacing and security management of the overall system. It provides generic centralized security services such as management of encryption mechanisms, security compliance management, and configuration management for TDA hosts.

The TDAD is also responsible for information exchange services for the overall system. Recall that the goal is to distribute information based on newly discovered cyberattacks to each member of the TDA. However, this goal can be challenging for large systems containing many mobile hosts and other dynamic cyber resources. Reachability issues in terms of maintaining the IP addresses of dynamic nodes as well as dynamic nodes located behind perimeter firewalls belonging to foreign networks roamed by mobile nodes causes challenges with seamless information distribution.

In order to solve these problems, the TDAD provides a virtual hub-and-spoke communication mechanism based on connection-oriented command and control (C2) protocols. As such, the TDAD serves as a centralized interfacing system for each TDA member, whereby information exchange describing the sources of new cyberattacks are distributed to TDA members via the TDA delegate and its C2 protocol. When a TDA member detects a new cyberattack, the PDS is delivered to the TDAD via the DIMI. The TDAD, in turn, initiates a command via its C2 channels bound to the remaining hosts within the system. The TDAD reverse-connects to each member's PDSI with a connection oriented C2 protocol. Once each member receives the PDS from the TDAD via its DIMI, the PDS is processed by its PMM and then delivered to its firewall management module.

Central to DFAR is the dynamic creation of blacklists via the distributed detection modules residing on TDA hosts. Once a detection module operating on a TDA host detects a network-based attack, a blocking enforcement filter that denies traffic to or from the IP address of the attack source is instantiated and implemented within the firewall of the host. Moreover, the new blocking enforcement filter is distributed to each other entity of the TDA via the DIMI and the TDAD. Upon convergence, all hosts comprising the TDA are aware of the attack source and are isolated from the attack source via the implementation of its blocking enforcement filter. Consequently, the complete list of TDA blocking enforcement filters is a blacklist.

Although detection modules are core mechanisms for the creation of TDA blacklists, external blacklists that are maintained by Internet security organizations can be implemented as blocking enforcement filters within the TDA. Particularly, another feature enabled by the TDAD and DFAR is the idea of enforcing blacklists extracted from well-established global blacklists such as SPAM blacklist databases and other centralized blacklist repositories such as DSHIELD [162].

In DFAR, the TDAD is responsible for establishing relationships with external security organizations who provide global repositories of blacklisted IP addresses. The TDAD retrieves blacklist entries from global repositories, creates corresponding PDSs that are configured to filter traffic to/from the blacklist entries, and distributes these PDSs to TDA members. These ideas are illustrated abstractly by Figure 26.

4.3.3 Firewalls, Blacklisting, and the Packet Classification Problem

Several investigations have recently considered the use of global blacklisting techniques for the preemptive prevention of cyberattacks, with all of the studies indicating a positive security impact resulting from the use of such filtering techniques. For example, Chen et al. [25] studied the spatial-temporal characteristics of cyberattack sources using data collected by DSHIELD and found that one out of 27 hosts connected to the Internet are potentially malicious sources and that information obtained from global blacklist repositories can provide protection from many sources of cyberattacks.

Other works have shown that blacklisting techniques can significantly enhance the overall stature of an organization's cybersecurity architecture. Yegneswaran et al. [161] show

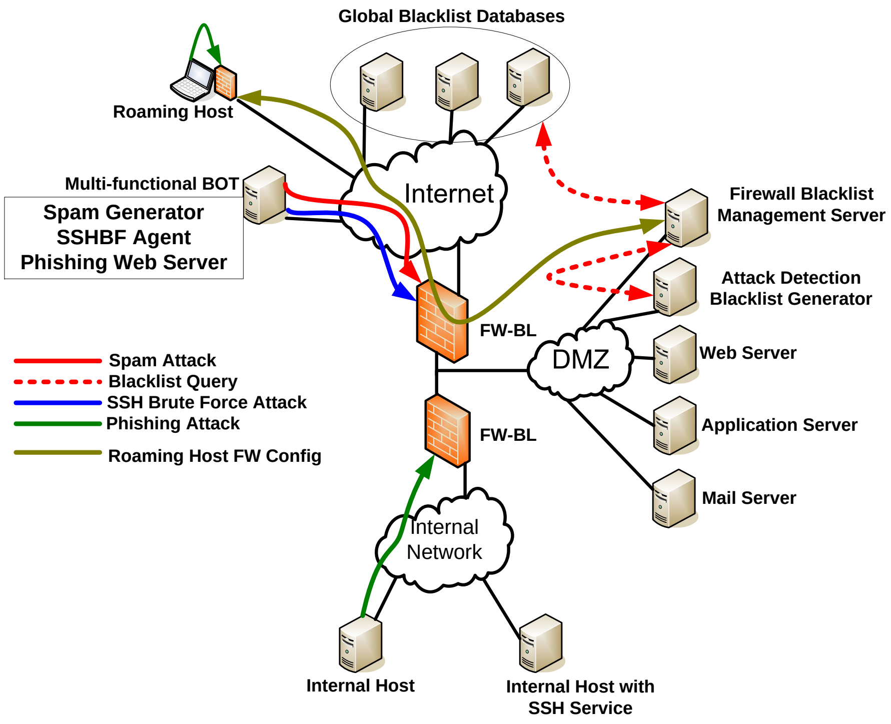


Figure 26: Distributed blacklisting systems with DFAR.

that IP address blacklisting significantly increases the performance of their distributed-collaborative intrusion detection system. Yegneswaran et al. claim (1) that IP address blacklists enable efficient classification of attack packets into well defined attack categories, (2) that on any given *day* (based on analysis of DSHIELD log files) a small number of attack sources are responsible for the largest fraction of attack volume, (3) that IP addresses of many attack sources are persistent (i.e., these sources are seen over and over for extended periods of time), (4) there is greater benefit to creating longer blacklists (i.e., limited blacklist pruning), and (5) that significant security benefits can be realized via blacklisting even when the blacklist becomes *old*.

Zhang et al. [166] introduced the idea of highly predictive blacklisting (HPB). The goal of HPB is to extract a blacklist subset from some globally accessible blacklist based on the idea of *relevancy*. In particular, a blacklist containing information with global-perspective will contain information more relevant to some organizations than others. The idea is to select the information from the blacklist whereby the likelihood of blacklisting a malicious source is maximized with respect to a given destination such as an organization's network. The HPB algorithm is similar to the relevance ranking system of the Google PageRank [20] algorithm. Soldo et al. [138] provide algorithmic techniques, with a goal similar to [166], to extract the most relevant blacklist members from a global blacklist. In [138], the authors realize that blacklist firewall filtering is limited because of the $O(N)$ search time complexity of many firewall implementations. Their paper introduces a family of algorithms that provide the most relevant blacklist members when there is a constraint on the total number of rules that can be implemented in a firewall. Kim et al. [77] provide the results of a study that measures the effectiveness of blacklisting to counter denial of service and scanning (probing) attacks. Their analysis is based on the idea of filtering *Martian* addresses along with unallocated IP addresses. Kim et al. use trace-based simulations to show that blacklisting Martian and unallocated IP addresses can significantly improve the security performance of systems when faced with denial of service and probing attacks.

Firewalls and attack detection systems require packet classification algorithms. Unfortunately, the packet classification problem is a theoretically challenging problem and fast packet classification algorithms are typically available only for high-end computational systems. Consequently, most host-based firewalls and detection systems use packet classification algorithms constructed by linear search engines to process the set of associated packet filters. However, TDA hosts will implement blacklist blocking enforcement within their firewalls and these blacklists will contain tens to hundreds of thousands of entries. As a result, efficient packet classification algorithms are required for the blacklist classification process used by DFAR's firewall-based blacklist enforcement model (i.e., Figure 22).

As an example, Figure 27 provides the results of an experiment performed on a production email server where the service time of its host-based firewall is measured versus the number of implemented packet filters. The filters range from $N = 1$ to $N = 7404$. As seen by the plot of Figure 27, the firewall service time for this server consistently increases with increasing numbers of filters (N). With $N = 1$, the firewall service time is less than 0.1ms, whereas with $N = 7404$ the service time is on the order of 1.2ms. Since the firewalls of TDA members will likely contain large numbers of filters, excessive communication delays

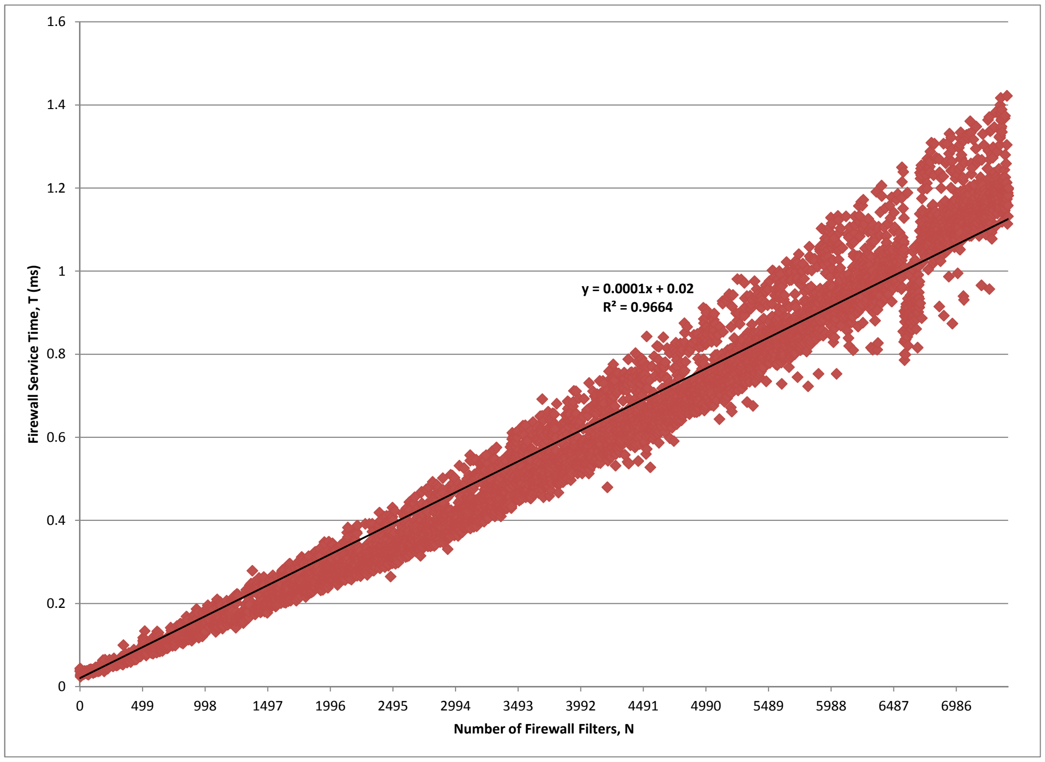


Figure 27: Experimental results of firewall service time versus N for an enterprise-class email server.

will occur as a result of increased firewall processing time. Therefore, an efficient packet classification algorithm is desired for firewall-based blacklist classification in TDA firewalls.

4.4 Summary

In this chapter, the distributed firewall and active response architecture has been proposed. The architecture provides a modular, adaptable, and integrable framework that explicitly supports the active participation of attack prevention and detection for each cyber device within an organization's environment. However, the packet classification problem faced by firewalls prevents the implementation of large numbers of blocking enforcement filters. As a result, the packet classification problem will be addressed by subsequent chapters.

CHAPTER V

A NOVEL THEORY OF SEMANTIC ASSOCIATION SYSTEMS

This chapter introduces a theory of semantic association systems inspired by the emerging paradigm of semantic computing. The constructs provided by this theory enabled the development of a new packet classification algorithm based on the notion of semantic path merger. After introducing the formal definitions and ideas of the semantic association system in this chapter, the semantic path merger packet classification algorithm along with the implementation details of its hardware realization will be given by the following chapter.

Semantic computing research seeks to establish a foundation for the derivation of semantics from content and to connect these semantics into knowledge [148]. Content, in this view of semantic computing, is any resource within the domain of Information Systems. This includes the World-Wide-Web , the Semantic Web (a vision of the future Web), and general information systems. Interest in semantic computing has emerged as a result of the research efforts of Semantic Web development [19, 164], but its foundations are rooted in the general field and study of semantics and the integration of the theory of semantics with computer science and computational semantics. Formally, semantics involves the study of *meaning*. In essence, it is the study of symbols, the relationships between symbols, and the representation thereof. The study of semantics in the non-computing sense is focused on understanding how humans interact with symbols such as the interpretation of words and sentences. Within the scope of semantic computing, the study of semantics includes fields such as program language specification, natural language processing [118], artificial intelligence [85], and intelligent information retrieval. The field has a strong foundation in terms of intelligent systems [54, 86]. However, the applications of semantic computing are growing with new research in diverse areas such as geodesics [121], team interactions [155], random walker algorithms [120], semantic filesystems [131, 38], and computer security [95, 154, 134, 127, 158].

5.1 A Compositional Computing Model of Semantic Association Systems

Perspective is a state of understanding, which is a time varying point-of-view attached to a collection of knowledge. Knowledge is a product of the comprehension, awareness, and understanding of information. Information is a key element of knowledge, but information in and of itself is not knowledge. Instead, information is a collection of data representing qualitative and/or quantitative characteristics of objects. When information is comprehended via synthesis and analysis of its data, an awareness of the inter-relationships and associations of diverse objects characterized by the data emerges. As a result, an understanding of the relational and associative aspects of diverse objects produced by awareness that emerges from the comprehension of information leads to either the creation of new knowledge and/or the modification of previous knowledge. Consequently, knowledge varies with both *time* and *context* (point-of-view). Hence, perspective is a time-dependent and context-dependent state of understanding affiliated with a collection of archived knowledge.

Concept, a construct representing an artifact of thought, forms the basic foundation of the compositional computing model of semantic association systems (SAS). In a semantic association system, a concept is defined to be the assignment of *meaning* to a *collection of components* under a given *perspective.* The meaning (semantics) of a concept, therefore, depends on a current state of understanding. A concept is a compositional bundle where component collections are bound under a perspective-constraint to a collection of one or more semantics (i.e., meanings). A *conceptualization* is a semantic family formed by a domain perspective and a knowledge system comprised of a collection of concepts.

Table 10 will be used to clarify the ideas presented thus far. Three values are provided as data representing temperature, location, and weather. When the data are combined into a compositional bundle, information is obtained whereby upon further comprehension and understanding, entities create knowledge. Perspective $P1$ assigns the knowledge 'A hot sunny day in Atlanta, GA' to the information bundle whereas perspective $P2$ prescribes 'Good swimming conditions' to the bundle. Either perspective may or may not be correct

Table 10: The ideas of data, information, semantics, knowledge, and perspectives.

Data
Temperature = $90°F$
Location = Atlanta, GA
Weather = Sunny

Information
($90°F$, 'Atlanta, GA', Sunny)

Knowledge
P1: 'A hot sunny day in Atlanta, GA'
P2: 'Good swimming conditions'

Semantics
$I(x, y, z) \Rightarrow$ '$WaterSportCriteria'$

(Knowledge | Information, Semantics)
($90°F$, 'Atlanta, GA', Sunny) $\Rightarrow$ P2: 'Good swimming conditions'

relative to a particular *ground truth.* However, when semantics are associated with information bundles, more precise knowledge with respect to the ground truth dictated by the semantics can be obtained. With the semantics $I(x, y, z) \Rightarrow$ '$WaterSportCriteria'$ along with the given information bundle, perspective $P2$ becomes the best alternative compared to $P1$.

In the semantic association system, concepts are represented as the binding of semantics to component bundles. The meta-language defining the semantic data structure of the SAS is shown in Table 11.

The meta-language is interpreted as follows. Concepts are bindings between component bundles and semantics, in which the semantics are prescribed meanings for the component bundles within some domain of understanding. Collections of concepts are assumed to belong to a particular domain perspective. Components and semantics are compositions of one or more atoms. An atom is a object composed of a single label, zero or more properties, and zero or more operators. A label is a typed element that can be any well-known or

Table 11: The meta-language of the semantic data structure.

$\langle \boldsymbol{Concept} \rangle$	$::=$	$\langle \boldsymbol{components} \rangle \longmapsto \langle \boldsymbol{semantics} \rangle$
$\langle components \rangle$	$::=$	$\{atom\}[1,*]$
$\langle semantics \rangle$	$::=$	$\{atom\}[1,*]$
$\langle atom \rangle$	$::=$	$(label\ \mathcal{L},\ property\ \mathcal{P}[0,*],\ operator\ \Theta[0,*])$
$\langle label \rangle$	$::=$	$typedVar\ t$
$\langle property \rangle$	$::=$	$(attribute = value)$
$\langle operator \rangle$	$::=$	$Y \leftarrow \Theta(X, \cdots)$
$\langle attribute \rangle$	$::=$	$identifier$
$\langle value \rangle$	$::=$	$identifier$
$\langle identifier \rangle$	$::=$	$Concept\ c \mid typedVar\ t$
$\langle typedVar \rangle$	$::=$	$(integer \vert float \vert char \vert string \vert boolean \vert \cdots)$

application specific *type* such as integers, floats, and strings. A property is an attribute-value pair where attributes and values are identifiers, and an identifier can be either a concept or a typed element. The structure allows concepts to specify properties of other concepts, which provides a high-degree of flexibility and expressibility. Finally, the operator is assumed to be a process or functionality that binds some set of inputs with some set of outputs relative to a particular concept.

5.2 Formal Model of the Semantic Association System

Let Σ be a universe of discourse. Given a set of labels $\mathcal{L} \subset \Sigma$, a set of properties $\mathcal{P} \subset \Sigma$, and a set of operators $\Theta \subset \Sigma$, let $\mathcal{V} = (\mathcal{L} \times \mathcal{P} \times \Theta)$ be an ordered collection of atoms.

Definition 1. *A* ***semantic family*** *is a perspective-dependent conceptualization represented by a directed acyclic graph $G = (V, E, R)$ where the vertex set*

$$V = \{V_1,\ \ldots,\ V_m, V_{m+1},\ \ldots,\ V_d\} \subseteq \mathcal{V}$$

is a semantically disjoint partitioned set of atoms comprising a group of components and a group of semantics. Without loss of generality, V can be rewritten as $V = (V_\rho, V_\sigma)$ where $V_\rho = \{V_1, V_2,\ \ldots,\ V_m\} \subset V$ is the collection of atoms categorized under perspective to be components and $V_\sigma = \{V_{m+1}, V_{m+2},\ \ldots,\ V_d\} \subset V$ is the collection of atoms categorized under perspective to be semantics. The vertex set of components and semantics are defined to be semantically disjoint partitions. Therefore, V must satisfy the following conditions.

1. $V_i = \{v_{ij}\} \in V, \qquad 1 \le i \le d, \quad 1 \le j \le |V_i|$

2. $\forall\,(i \neq j): \quad V_i \cap V_j = \emptyset$

3. $V = \bigcup_{i=1}^{d} V_i$

A collection $V_i \in V$ is a semantic-equivalence class within the semantic family G. For example, let V_i be the semantic-equivalence class representing *Integers*, then $v_{ij} = 9$ and $v_{ik} = 256$ are semantically-equivalent elements of the class *Integers*. Moreover, let $V = \{V_1, V_2, V_3\}$ where $V_1 = Integers$, $V_2 = Binary\ Operators$, $V_3 = Mathematical\ Entities$ and let $V_\rho = \{V_1, V_2\}$, $V_\sigma = \{V_3\}$. Then, the concept $(V_1, +) \longmapsto$ *Abelian Group* is the assignment of the semantic 'Abelian Group' to the component bundle (algebraic structure) $(V_1, +)$ where $V_1 = Z$ is the set of 'Integers' and $+ \in V_2$ is the 'addition operator' from the class of 'Binary Operators'. Similarly, the concept $(\{x, y\}, +) \longmapsto$ *Integer Addition* is the assignment of 'Integer Addition' to mean $x + y$.

Definition 2. *The* ***semantic associations*** *of a semantic family are defined to be the set of edges $E \in G$ where $E = \{e : \quad e = (u, v) \Rightarrow (u \in V_i) \wedge (v \in V_j) \wedge (i \neq j)\}$.*

Definition 3. *The* ***semantic relations*** *bound to a semantic family G are defined to be the collection of concepts $R = \{r_i\}$ where $r_i = (\rho, \sigma)_i$ such that:*

1. $u \in \rho \Rightarrow (u \in V_j) \wedge (V_j \in V_\rho)$

2. $v \in \sigma \Rightarrow (v \in V_k) \wedge (V_k \in V_\sigma)$

3. *$r = (\rho, \sigma)$ is an $s-t$ relational path in G such that $r = (v_s, \ldots, v_j, \ldots, v_t)$. v_s is the source node and by definition belongs to ρ, and v_t is the target node and by definition belongs to σ.*

By convention, the existence of an $s - t$ relational path r in G implies that:
$\forall\,(v \in r) : (v_m = r\,(k)\,,\ v_n = r\,(k+1)) \Rightarrow (v_m,\ v_n) \in E$ where $r(k)$ is the k^{th} atom of the $s - t$ relational path.

In terms of the semantic data structure defined by the SAS, observe that ρ is a component bundle, σ is the set of semantics *bound* to the component bundle, the semantic relation

$r = (\rho, \sigma)$ is a concept within the semantic family G, the $s-t$ path r in G represents the semantic binding of ρ with σ, and the set of semantic relations R represents a knowledge base of concepts within the semantic family G. In other words, R is a conceptualization defined by an underlying perspective, and G is a graph structure representing the conceptualization.

Definition 4. *A* ***semantic space*** *is defined to be a* ***merged conceptualization****. A semantic space is constructed by the structure* $\mathcal{S}_\sigma = [G_\sigma,\ \Gamma]$ *where* $G_\sigma = \{G_1,\ G_2,\ \ldots,\ G_N\}$ *is a collection of* N *unique semantic families, and* $\Gamma = \langle \Gamma_V, \Gamma_E, \Gamma_R \rangle$ *is a system of coloring operators.*

The structure $\mathcal{S}_\sigma$ is a *conceptual typing* construct defined by the production rules of Table 12. The production rules state that $[G_\sigma,\ \Gamma]$ assigns a *type* defined by color to each

Table 12: Conceptual typing production rules of the semantic space.

$[G_\sigma,\ \Gamma]$	$\vdash$	$G = (V, E, R) \leftarrow \Lambda$
Γ	$\vdash$	$\Lambda \leftarrow uniqueIdentifier$
Λ	$\vdash$	$\lambda_i \leftarrow indexPseudonym$
λ_i	$\vdash$	$i \leftarrow \{Z^+, String\}$

semantic family. The coloring operator Γ produces a set of unique colors $\Lambda = \{\lambda_i\}$. The elements of the color class, in this treatment, are specified to be either an element of the positive integers $Z^+ = \{0, 1, \ldots\}$ or an element of the class $String$. The system of coloring operators are defined by the following color maps:

$$\Gamma = \langle \Gamma_V, \Gamma_E, \Gamma_R \rangle :\ (V, E, R) \leftarrow \Lambda$$

$$\Gamma_V(V) = \lambda$$

$$\Gamma_E(E) = \lambda$$

$$\Gamma_R(R) = \lambda$$

The system of coloring operators assigns the same color to a semantic family and all of its constituents. The underlying idea is that applying a unique color to each semantic family within a semantic space induces a uniqueness property such that after the merged conceptualization process has occurred, one can still distinguish the features of the original semantic families.

Merged conceptualization is produced by superimposing the N color-mapped semantic families of G_σ onto a single graph structure. The resultant graph structure is referred to as the semantic association network.

Definition 5. *The* ***semantic association network****, denoted by G_ϕ, is a superimposed, semantic-preserving structure that encodes a semantic space. The semantic association network (SAN) is an edge-colored multi-digraph*

$$G_\phi = (V_\phi, E_\phi, R_\phi, \Lambda_\phi)$$

where R_ϕ is the set of colored semantic relations, Λ_ϕ is the set of colors produced by Γ over the semantic space G_σ, $V_\phi = \{V_1,\ V_2,\ \ldots,\ V_\delta\}$ is the set of colored vertices where $\delta \geq d$, and E_ϕ is a set of colored edges.

The semantic association network is a structure representing the merged conceptualizations of a semantic space, and its construction is specified by the graph superimposition $G_\phi = \bigcup_{i=1}^{N} G_i$ produced by the following:

1. $\forall\ V_i\ \in G_\sigma :\ V_\phi = \bigcup V_i$

2. $\forall\ E_i\ \in G_\sigma :\ E_\phi = \bigcup E_i$

3. $\forall\ R_i\ \in G_\sigma :\ R_\phi = \bigcup R_i$

4. $\forall\ \lambda_i\ \in [G_\sigma, \Gamma] :\ \Lambda_\phi = \bigcup \lambda_i$

In general, a component $v \in V_\phi$ will belong to one or more colored semantic relations or, equivalently, it will belong to one or more semantic families. The coloring operator associates a unique color to each element of a semantic family. Hence, the construction of a semantic association network implies that vertices and edges contained in G_ϕ will have multiple colors.

Definition 6. *The* ***prism*** *is defined to be a mapping $\psi : V_\phi \to \Lambda_\phi$ such that $\psi(v) = \Lambda_v$ where $\Lambda_v = \{\lambda_i\} \subseteq \Lambda_\phi$ is the* ***spectrum*** *of vertex v.*

Definition 7. *The* ***spectral intensity*** *of an atom v is defined to be the mapping $\Psi : V_\phi \to (0, 1]$ where $\Psi(v) = \frac{|\Lambda_v|}{|\Lambda_\phi|} = \frac{|\psi(v)|}{N}$.*

In essence, the spectrum is a membership function indicationg the set of semantic relations to which a vertex belongs within the semantic space, and spectral intensity provides a measure of the vertex's semantic affiliation within the semantic association network.

The superimposition of a semantic space onto a semantic association network implies that an edge $e = (u, v)$ will belong to one or more edge-sets from $E_i \in G_i \in G_\sigma$. As stated above, the coloring operator assigns a color to each component of a semantic family. The edge-color association can be written as (e, λ_i) where $e = (u, v) \in E_i$. In terms of applying the color to an edge, the following convention will be used. The edges in the semantic association network are written as (u, λ, v) such that $E_\phi = \{e = (u, \lambda, v) : \ \forall \lambda_k \in (\Lambda_u \cap \Lambda_v)\,,\ \exists e = (u, \lambda_k, v)\}$. Figure 28 illustrates the edge-colored multi-digraph of a simple semantic association network.

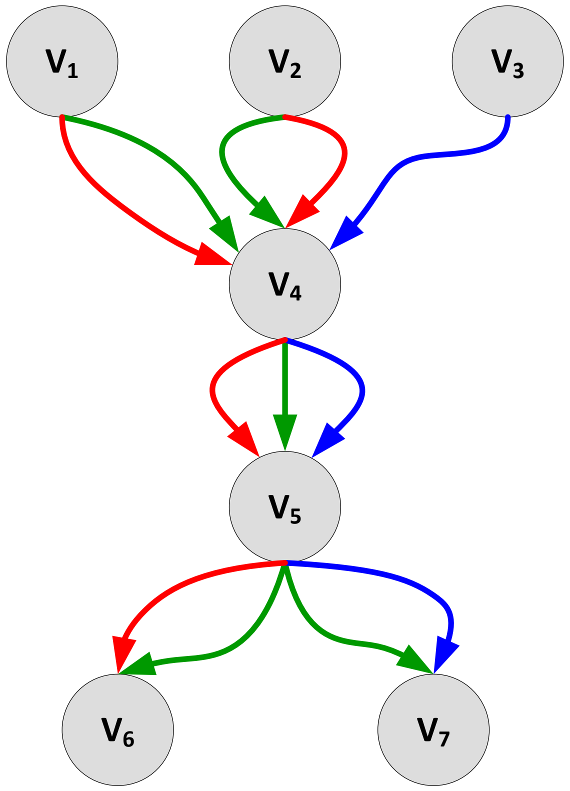


Figure 28: A simple edge-colored multi-digraph.

5.3 Hypergraph Representation of the Semantic Association Network

A hypergraph is defined by the graph $G = (V, E)$ where V is a set of vertices and E is a set of hyperedges. A hyperedge can connect more than two vertices. Particularly, a hyperedge is an element of the power set $\mathbb{P}(V)$ of the vertex set V. From a graphical point-of-view, a hyperedge supports containment of more than two vertices. Figure 29 illustrates a hyperedge enclosing multiple vertices.

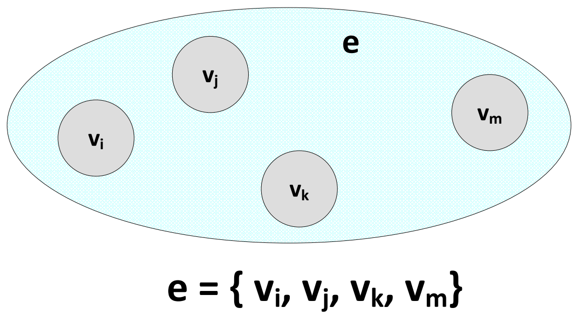


Figure 29: A hyperedge enclosing three vertices.

Definition 8. *The* ***colored path hypergraph*** *(CPHG) of the semantic association network is defined to be the graph* $G_\Phi = (V_\phi, E_\lambda)$ *where* $V_\phi \in G_\phi$ *and:*

$$E_\lambda = \{e_m = (v_i, v_j,\ \ldots,\ v_k) :\ \lambda_m \in \left(\Lambda_{v_i} \bigcap \Lambda_{v_j} \bigcap \cdots \bigcap \Lambda_{v_k}\right)\}.$$

The CPHG is constructed by grouping the colored edges of G_ϕ. The set of hyperedges, therefore, represent equivalence classes with respect to the vertices and colors of the SAN such that $e_m := [v]_{\lambda_m}$. Figure 30 provides a visualization of the CPHG representing the SAN of Figure 28. From Figure 30, the following constituents of the CPHG can be observed with respect to the definitions of the SAS.

$V_\phi = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$

$E_\lambda = \{e_1, e_2, e_3\}$

$\Lambda_\phi = \{\lambda_1, \lambda_2, \lambda_3\} \equiv \{red, green, blue\} \equiv \{r, g, b\}$

$\lambda_1\ :\ e_1 = \{v_1, v_2, v_4, v_5, v_6\}$

$\lambda_2\ :\ e_2 = \{v_1, v_2, v_4, v_5, v_6, v_7\}$

$\lambda_3\ :\ e_3 = \{v_3, v_4, v_5, v_7\}$

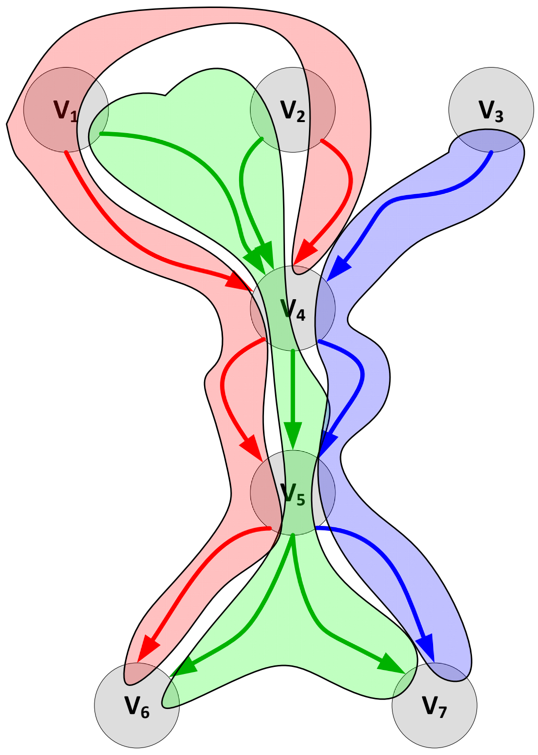


Figure 30: A colored path hypergraph.

A hypergraph can be represented by an *incidence matrix* that relates the edges of a graph to the vertices of a graph. The incidence matrix of a graph is the zero-one matrix $M = [m_{ij}]$ such that $m_{ij} = 1$ if and only if edge e_i is incident to vertex v_j. The incidence matrix of a hypergraph is defined similarly.

Definition 9. *The incidence matrix of the colored path hypergraph G_Φ of a semantic association network G_ϕ is defined to be the zero-one matrix denoted by Φ such that:*

$$\begin{aligned} \Phi &= [\phi_{ij}] \text{ and} \\ \phi_{ij} &= 1 \text{ if and only if } e_i \in E_\lambda,\ v_j \in V_\phi,\ v_j \in e_i, \text{ and } \lambda_i \in \Lambda_{v_j}. \end{aligned}$$

*The incidence matrix Φ is called the **Path Matrix** of the semantic association network.*

The incidence matrix Φ is referred to as the path matrix (PM) because it provides a unique encoding of the semantic relations R_ϕ of the semantic association network. Recall that the semantic relations are $s-t$ paths representing concepts, which are by definition the bindings of conceptual *components* to conceptual *semantics*. These $s-t$ paths are encoded by the incidence matrix of the CPHG. Hence, this incidence matrix is denoted as the path matrix. Table 13 contains the PM for the CPHG of Figure 30. Given the path matrix Φ and

Table 13: The path matrix for the CPHG from Figure 30.

	$\boldsymbol{v}_1$	$\boldsymbol{v}_2$	$\boldsymbol{v}_3$	$\boldsymbol{v}_4$	$\boldsymbol{v}_5$	$\boldsymbol{v}_6$	$\boldsymbol{v}_7$
$\boldsymbol{e}_1$	1	1	0	1	1	1	0
$\boldsymbol{e}_2$	1	1	0	1	1	1	1
$\boldsymbol{e}_3$	0	0	1	1	1	0	1

its associated CPHG G_Φ and since the hyperedges $e_i \in E_\lambda$ represent color-wise groupings of vertices in V_ϕ, a correspondence between Φ and Λ_ϕ can be observed, which is illustrated by Figure 31. The correspondence relationship shown in Figure 31 is represented by the

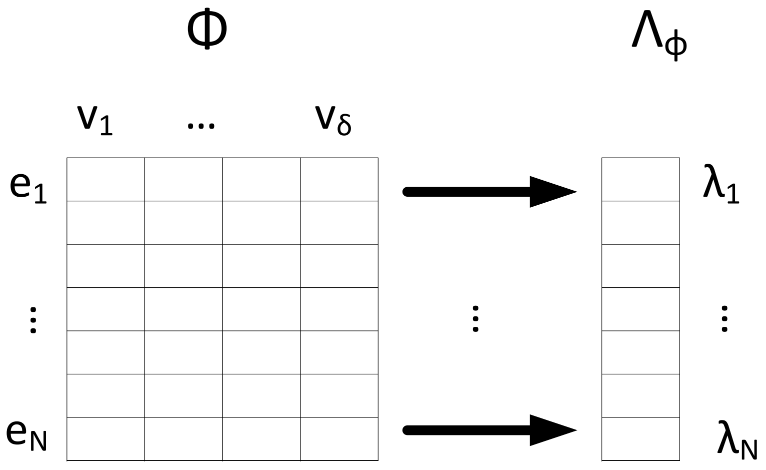


Figure 31: A correspondence relationship between the vertices and edges of the CPHG and the semantic colors.

following:

$\Phi : \ E_\lambda \ \times \ V_\phi \ \leftrightarrow \ \Lambda_\phi$

such that given $\Phi \ = \ [\phi_{ij}]$, then $\forall \phi_{ij} = 1, \ \exists (e_i \in E_\lambda, \ v_j \in V_\phi, \ \lambda_i \in \Lambda_\phi)$ where $\Phi(e_i, v_j) \leftrightarrow \lambda_i$. The subscript indices for the hyperedges and the colors are not to be neglected. In other words, $(e_i \rightarrow \lambda_i) \wedge (e_j \rightarrow \lambda_i) \ \Rightarrow \ (i = j)$. Based on the formulations introduced thus far, the following general relationships are observed. First, there is a one-to-one relationship between E_λ and Λ_ϕ. Second, there is a many-to-one relationship between V_ϕ and Λ_ϕ for a particular $e_i \in E_\lambda$. Third, there is a many-to-many (multivalence) relationship between V_ϕ

and Λ_ϕ when considering all of E_λ. These relationships are illustrated by Figure 32 and enumerated below.

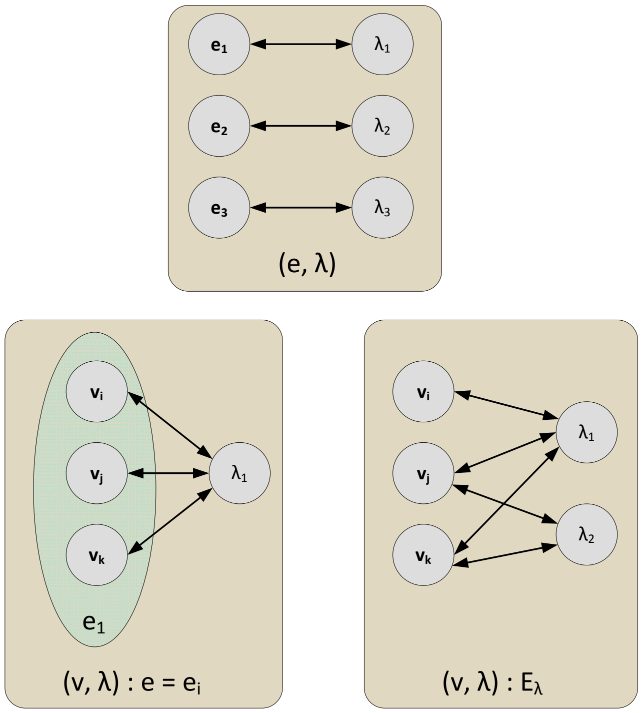


Figure 32: General m-to-n relationships between the vertices, edges, and colors of the CPHG.

1. For the system (e, λ) :

   (e_1, λ_1)

   (e_2, λ_2)

   (e_3, λ_3)

2. For the system $(v, \lambda) : e = e_1$:

   (v_i, λ_1)

   (v_j, λ_1)

   (v_k, λ_1)

   $e_1 = \{v_i, v_j, v_k\} \ \leftrightarrow \ \lambda_1$

3. For the system $(v, \lambda) : E_\lambda$:

    (v_i, λ_1)

    $(v_j, \{\lambda_1, \lambda_2\})$

    $(v_k, \{\lambda_1, \lambda_2\})$

    $e_1 = \{v_i, v_j, v_k\} \ \leftrightarrow \ \lambda_1$

    $e_2 = \{v_j, v_k\} \ \leftrightarrow \ \lambda_2$

5.4 Characteristics and other Transformations of the Semantic Association System

As described in the previous section, vertices of an SAS can belong, in general, to more than one semantic family. Hence, a particular vertex will belong to one or more CPHG hyperedges. The multivalence associative property of the vertex set V_ϕ is explored further in this section.

Definition 10. *A multivalence vertex query (MVQ) is defined to be the graph query:*

$Q : \ V_\phi \ \rightarrow \ (E_\lambda \times \Lambda_\phi)$

such that

$Q(v_j) \ = \ \langle(e, \lambda)\rangle \ = \ \langle(e_{i_1}, \lambda_{i_1}), (e_{i_2}, \lambda_{i_2}), \ldots, (e_{i_m}, \lambda_{i_m})\rangle$

where $\langle(e, \lambda)\rangle$ *is a 2-tuple hypervector of length* m *and* $\forall e \in E_\lambda, v_j \in e \Rightarrow (e, \lambda) \in Q(v_j)$.

The prism was defined by $\psi(v_j) = \Lambda_{v_j}$ where $\Lambda_{v_j} \subseteq \Lambda_\phi$ is the spectrum of a vertex (atom) v_j. The spectrum enumerates a vertex's semantic colors, and, by extension, to which semantic families it belongs. Consequently, the prism and the multivalence vertex query are closely related. The prism produces the set of colors associated with a vertex, whereas the MVQ produces the set of associated hyperedges *and* the set of associated colors. An MVQ produces a 2-tuple hypervector of length m. Since $Q(v_j)$ enumerates the m unique (e, λ) tuples associated with v_j, then the prism must produce a spectrum Λ_{v_j} where $m = |\Lambda_{v_j}|$. Consequently, $\forall \lambda_i \in Q(v_j), \ \exists \lambda_i \in \Lambda_{v_j}$. A straightforward implementation of these maps can be achieved via the path matrix of the CPHG.

Definition 11. *Given a path matrix $\Phi = [\phi_{ij}]$ where $\phi_{ij} = 1$ if and only if $v_j \in e_i$, the bit vectors $\phi = (\phi_{i*}, \phi_{*j})$ are defined by the following basis:*

$\phi_{i*} \;:=\; \{0,1\}^{\omega} \;:=\; (b_1, b_2, \ldots, b_{\omega})$

$\phi_{*j} \;:=\; \{0,1\}^{N} \;:=\; (b_1, b_2, \ldots, b_{N})$

where $b_k \in \{0,1\}$, ϕ_{i*} is a binary bit vector of length ω, and ϕ_{*j} is a binary bit vector of length N. Further, ω and N are given by:

$N = |E_\lambda| = |\Lambda_\phi|$

$\omega = \sum\limits_{i=1}^{\delta} |V_i|, \quad V_i \in V_\phi$

The formal definitions of the bit vectors are given by the following:

$\phi_{i*}: \; E_\lambda \to \{0,1\}^{\omega}$ *such that* $\phi_{i*}(e_i) \;=\; \Phi[i,*]$

$\phi_{*j}: \; V_\phi \to \{0,1\}^{N}$ *such that* $\phi_{*j}(v_j) \;=\; \Phi[*,j]$

The notation $\Phi[i,*]$ represents the extraction of all the columns of Φ over the row indexed by i, and $\Phi[*,j]$ represents the extraction of all the rows of Φ over the column indexed by j. The symbol $*$ has similar meaning to the *wildcard* symbol of most regular expression languages whereby an expression containing $*$ at some position x within a lexicographically ordered word represents all possible lexicons for the symbol substituted by $*$. For example, $\Phi[i,*] \;\equiv\; (\Phi[i,1], \Phi[i,2], \ldots, \Phi[i,\omega])$ and $\Phi[*,j] \;\equiv\; (\Phi[1,j], \Phi[2,j], \ldots, \Phi[N,j])$.

The interpretation of ϕ_{i*} and ϕ_{*j} can be viewed from the *characteristic* perspective. In particular, ϕ_{i*} is the zero-one characteristic vector of e_i with respect to V_ϕ, and ϕ_{*j} is the zero-one characteristic vector of v_j with respect to E_λ. The relationship between E_λ and Λ_ϕ is a bijection (one-to-one correspondence). The bijection can be observed by the illustration shown previously in Figure 31.

Definition 12. *Given a semantic association system, there exists a* ***colored path bijection*** *from the set of CPHG hyperedges to the set of semantic colors. The colored path bijection (CPB) is defined by the following map:*

$H: \; E_\lambda \to \Lambda_\phi$ *where*

$H(e_i) \;=\; \lambda_i$

Definition 12 is a direct consequence of the construction of the colored path hypergraph. In particular, the hyperedges of the CPHG are color-wise groupings of vertices from the semantic association network such that all vertices in a particular hyperedge share a common color. In essence, the CPHG is a **λ-composition** or, similarly, a **vertex-color composition** of the semantic association network.

The characteristic interpretation of ϕ_{i*} and ϕ_{*j} can now be extended. ϕ_{i*} is the i^{th} row of Φ, which is the characteristic vector of e_i with respect to the complete vertex set V_ϕ. Moreover, the colored path bijection formally describes the one-to-one correspondence between the hyperedges of G_Φ and the set of semantic colors Λ_ϕ. Consequently, the binary column ϕ_{*j} representing v_j can be viewed as a *spectrum vector* since it is equivalent via the CPB to the binary encoding of semantic colors to which v_j belongs. In other words, ϕ_{*j} has an equivalent representation as being the characteristic vector of v_j with respect to the complete set of semantic colors.

Because the CPB is a bijection, the mapping $H(e)$ implies that H has a bipartite graph. Further, $\forall(e_i, \lambda_i),\ H(e_i) = \lambda_i$ and $H^{-1}(\lambda_i) = e_i$. For clarity, let H_2 be the bipartite graph of H. By definition, a graph $G = (V, E)$ is bipartite if V can be partitioned by two disjoint sets $V = \{V_1, V_2\}$ where $V_1 \cap V_2 = \emptyset$ and $\forall e \in E,\ e = (u, v) \Rightarrow (u \in V_1 \wedge v \in V_2)\ \vee\ (u \in V_2 \wedge v \in V_1)$. A bijection guarantees that its graph is bipartite. However, a bipartite graph does not guarantee the existence of bijection. In short, the bipartite-ness of a graph does not require injectivity. Hence, a many-to-one non-injective mapping can be represented by a bipartite graph if the mapping satisfies the bi-partitioning property and edge constraints defined by graph bipartite-ness. A single hyperedge of the CPHG is, by definition, a collection of two or more vertices in V_ϕ such that the vertices are associated by semantic color. Consequently, a single hyperedge represents a many-to-one relationship from V_ϕ to Λ_ϕ, which was explained previously and illustrated by the system $(v, \lambda) : e = e_1$ shown in Figure 32. With the bijection H and its bipartite graph H_2, the bipartite graph of the tuple (V_ϕ, Λ_ϕ) can be constructed. The CPHG G_Φ or, similarly, its incidence matrix Φ represent a collection of N many-to-one associations from V_ϕ to E_λ. Further, the colored path bijection H and its bipartite graph H_2 represent one-to-one correspondences from E_λ

to Λ_ϕ. Therefore, a bipartite graph representing the N different many-to-one relationships from V_ϕ to Λ_ϕ can be constructed.

Definition 13. *Let the colored path bijection H be given along with its bipartite graph $H_2 = (H_v, H_e)$ such that the vertex set $H_v = \{E_\lambda \bigcup \Lambda_\phi\}$ where E_λ is the domain of H, Λ_ϕ is the codomain of H, and the edge set $H_e = \{h : \forall H(e_i) = \lambda_i, \exists h = (e_i, \lambda_i)\}$. The semantic-preserving* ***vertex-color decomposition*** *of the colored path bijection is the bipartite graph H_Δ defined by:*

$H_\Delta \ = \ (V_\Delta, E_\Delta)$

$V_\Delta \ = \ \{ \ \lfloor\{v \in E_\lambda\}\rfloor \ \bigcup \ \Lambda_\phi \ \}$

$E_\Delta \ = \ \{e_\alpha : \forall h = (e_m, \lambda_m) = (\{v_i, v_j, \ldots, v_k\}, \lambda_m) \in H_e, \exists e_{\alpha_1} = (v_i, \lambda_m), e_{\alpha_2} = (v_j, \lambda_m), \ldots, e_{\alpha_n} = (v_k, \lambda_m)\}$

Since E_λ may have many hyperedges that contain a particular vertex v, the notation $\lfloor\{v \in E_\lambda\}\rfloor$ is introduced and interpreted as the set of all distinct vertices extracted from E_λ. The definition of E_Δ can be interpreted as a color-preserving hyperedge introspection whereby a hyperedge is decomposed into its constituent vertices and a λ-matching from each of the decomposed vertices in e_i to its associated λ_i is used to construct H_Δ. Thus, the transformation from H_2 to H_Δ is referred to as a vertex-color decomposition. The mappings and transformations presented in this section are now clarified. Recall the following systems shown in Figure 32. The system $(v, \lambda) : e = e_i$ represents a single hyperedge in G_Φ and reveals the one-to-one mapping from e_1 to λ_1 along with the many-to-one mapping from $\{v_i, v_j, v_k\} \in e_1$ to λ_1. The system given by (e, λ) represents the bipartite graph H_2 of the colored path bijection $H(e) = \lambda$. Lastly, the system $(v, \lambda) : E_\lambda$ is the many-to-many bipartite graph H_Δ. Figure 33 illustrates the entire family of graph transformations of the semantic association system.

5.5 Summary

The novel theory of semantic association systems was presented in this chapter. The theory is founded on a compositional computing model based on concepts, components, and semantics. A formal description of a semantic data structure provides a basis to which the

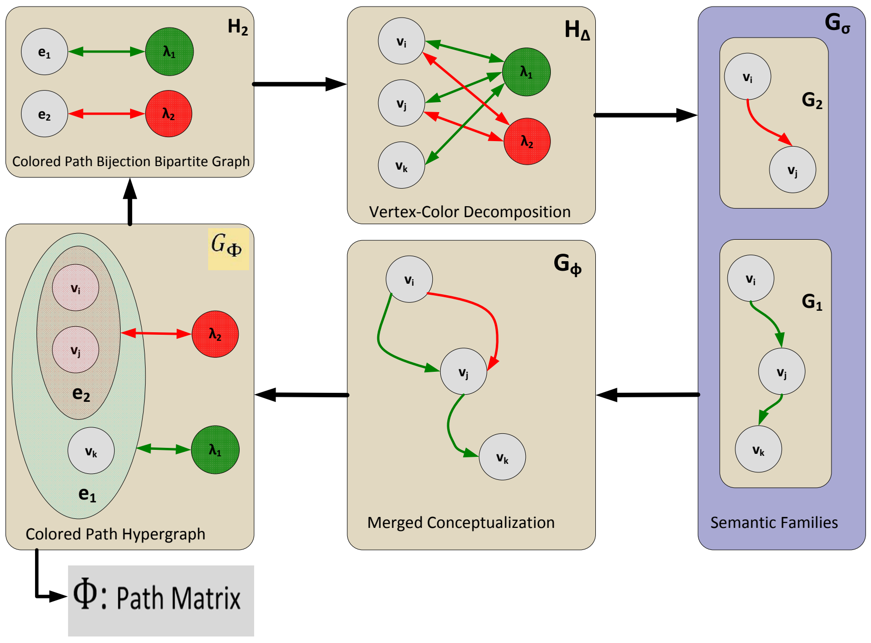


Figure 33: The family of graph transformations of the semantic association system.

elements of an SAS are constructed. Particularly, a concept is defined to be the result of binding (semantic binding) component bundles with semantics. A collection of semantically bound concepts forms a semantic family, which is also referred to as a conceptualization. Merged conceptualization is a process that superimposes a collection of semantic families onto a common graph structure that yields a semantic association network G_ϕ. In order to preserve the underlying meanings of concepts across different conceptualization domains, a semantic-preserving operation is provided by the conceptual typing structure $S_\sigma = [G_\sigma, \Gamma]$ where G_σ is the collection of N semantic families and Γ is the system of semantic coloring operators. During superimposition, each constituent of a semantic family is given a common color, and the coloring operator abides by the color uniqueness property such that each semantic family and its associated constituents are assigned a unique color. The uniqueness property assures that if a particular constituent exists in multiple conceptualizations, it can be decomposed with its spectrum such that an inverse mapping to its associated semantic families can be achieved. The colored path hypergraph G_Φ is a color-wise vertex-grouping graph transformation of the semantic association network. The incidence matrix of G_Φ

encodes the λ-paths between the semantically disjoint vertex classes V_ϕ of the semantic association network. As a result, this incidence matrix is referred to as the path matrix Φ. The colored path bijection H and its bipartite graph H_2 are simple representations of the one-to-one correspondence between the hyperedges of G_Φ and the set of semantic colors Λ_ϕ. A vertex-color decomposition produced by hyperedge introspection and λ-matching produces the bipartite graph H_Δ, which is a many-to-many vertex-to-color graph that represents the matching between vertices of V_ϕ and Λ_ϕ.

CHAPTER VI

HARDWARE-BASED MULTIDIMENSIONAL PACKET CLASSIFICATION WITH THE SEMANTIC PATH MERGER ALGORITHM

Research by the packet classification community over recent years suggests that hardware-based packet classifiers are much desired because software-based packet classification solutions cannot satisfy the stringent constraints of modern-day IP networks, especially within the core of the Internet's routing infrastructure [69, 70, 16, 57, 79]. Others suggest that future packet classification systems will likely be hybrids comprised of software components coupled with hardware acceleration components [147, 56].

Three key issues drive the need for hardware-based packet classification:

- Algorithmic complexity
- Scale
- Speed

The algorithmic complexity of packet classification can be seen by the theoretical bounds of its space-time tradeoffs. These tradeoffs are shown in Table 14. Recall that N is the number of filters and d is the number of dimensions (the number of header fields per filter). The tradeoffs show that a packet classification algorithm can achieve logarithmic

Table 14: Theoretical space-time bounds for packet classification algorithms.

Space	**Time**
$O(N^d)$	$O(lgN)$
$O(N)$	$O((lgN)^{d-1})$

time if memory can grow exponentially with dimension. On the other hand, reducing memory consumption to linear complexity requires a time penalty that does not favor high

Table 15: Line rates, packets per second, and classification rates.

OC-n	λ (bps)	λ_{pc} (pps)	τ_{pc} (spp)
OC-1	51.8M	162K	6.2u
OC-3	155.5M	486K	2.1u
OC-12	622.1M	1.94M	514.4n
OC-24	1.2G	3.89M	257.2n
OC-48	2.5G	7.78M	128.6n
OC-192	10G	31.25M	32n
OC-256	13.3G	41.47M	24.1n
OC-768	40G	125M	8n

dimensionality. Obviously, these tradeoffs affect scale and speed. However, packet classifiers must support both.

In terms of scale, packet classifiers need to support large numbers of filters in high dimension. For example, firewalls require $d = 5$ dimensions just for the basic filter specifications defined by the tuple (protocol, source IP, destination IP, source port, destination port). Detection systems require these five tuples along with others such as payload content fields. Consequently, an algorithm that supports these applications can achieve logarithmic time complexity only if the system supports memory consumption in $O(N^5)$.

In terms of speed, even the best time complexity of $O(lgN)$ is not sufficient when the wire-speed packet classification constraint must be met. Wire-speed packet classification states that a packet classification algorithm must classify packets at speeds such that buffering is not required at the input/output interface. These speeds can be calculated if the line rate of the interface is known along with the expected size of a packet. When calculating these speeds, network engineers assume worst case packet streams consisting of 40B TCP acknowledgment packets. Table 15 lists a sequence of OC-n line rates along with the associated classification rates. The line rate (λ) has units of bits per second (bps), λ_{pc} is the associated packets per second (pps) classification rate, and τ_{pc} is the associated amount of time required in units of seconds per packet (spp).

Consider a line rate of 10Gbps corresponding to an OC-192 network. The packet classifier must process 31.25 million packets per second. To achieve this rate, the classifier must classify a packet in time less than or equal to 32ns. Now, consider a packet classifier with $N = 2^{10}$ filters and assume the algorithm has an $O(lgN)$ worst case time complexity. Then, the classifier has a worst case number of memory accesses that is $O(10)$. Assume further that the algorithm has a memory subsystem such as SRAM with a 10ns access time. In this example, the classifier must process each packet within 32ns. With a 10ns memory access time, a maximum of 3 memory accesses must be met in order to meet the time requirement. However, the worst case for this system requires 10 memory accesses. Hence, the system cannot guarantee wire-speed classification, even for an algorithm with logarithmic time complexity.

In summary, memory access times coupled with high-speed line rates and wire-speed classification constraints limit the applicability of software-based packet classification algorithms. In order to address these issues, hardware-based packet classification algorithms that support various levels of parallelism and/or pipelining are needed. Currently, two primary types of hardware-based systems exist for this problem: content addressable memory (CAM/TCAM) and SRAM-based trie pipelines. Content addressable memories scale with both N and d but suffer significantly from extreme amounts of power consumption resulting from their massive parallelism. On the other hand, SRAM-based trie pipelines consume less power because they use embedded SRAM memory and amortize memory accesses over pipeline stages. However, these systems do not scale with dimension because they implement the same trie traversal algorithm that software-based algorithms use and, therefore, suffer from the backtracking problem.

In an effort to advance the state-of-art in high speed packet classification, this chapter introduces a novel hardware-based packet classification algorithm called *semantic path merger*. In a previous chapter, the formal mathematics, definitions, and construction methods of the semantic association system (SAS) were introduced. In this chapter, the semantic path merger algorithm, which employs the constructs provided by the SAS, is introduced along with its realization as a hardware-based packet classification system.

The colored path hypergraph (CPHG) of the semantic association network is of particular interest and importance. The SAS can be viewed as an associative system, and the incidence matrix of the SAS's colored path hypergraph (CPHG) provides a straight-forward hardware realization of the associative system. In particular, the hardware realization of the CPHG is an ***associative memory***. The goal is to formulate a methodology such that the header field specifications of a packet filter can be encoded by a directed acyclic graph, which will then be viewed as a semantic family. Each filter will be defined as a unique semantic family, and, consequently, the merged conceptualization process will be applied to the collection of packet filters, which are equivalent to the system of semantic families when viewed in the context of the semantic association system. In short, the packet classification system proposed in this dissertation implements the set of packet filters as the hyperedges of a graph, the individual points (header fields extracted from a packet based on the field specifications of a packet filter) are the vertices of a hyperedge, and the actions associated with each filter are represented by semantic colors.

The objective of this chapter is to introduce the algorithmic methods and the generic hardware design of the proposed SPM packet classification system. The chapter will also provide details of the simulation methodology that was used to compare SPM performance metrics versus content addressable memory.

6.1 The SPM multidimensional packet classification algorithm

The fundamental idea underlying SPM is notion of *merged prefix paths*. When using a multibit trie (MBT) to encode the prefixes of a packet filter, the stride c determines the number of bit inspections required to traverse the levels of the MBT. Given a stride c, there are 2^c possible bit combinations at each level, and these combinations lie within the integer range $[0, 2^c - 1]$. These bit combinations are fixed at each level, but the tree structure of the MBT provides a unique path through the trie. Tree structures in general possess the property of providing unique paths, and any encoding within a tree structure can be uniquely decoded. As the length of word encodings increases, both the height of the tree and the width of the tree's base increases. Previous efforts seeking to map trie data structures to pipelined hardware have been mostly concerned with balancing the number of trie nodes that are mapped to each pipeline stage, and the need for node balancing is caused by growth patterns of the trie's base width. A tree's base width increases as a consequence of path uniqueness whereby multiple tree nodes that have the equivalent semantics (meanings) are replicated along the unique paths of the tree.

One method to control the width of a tree base is to fix the labels with a structured graph and to superimpose the paths onto this fixed graph structure. However, this technique collapses the path uniqueness property of trees. In particular, path superimposition can induce *phantom paths* that do not exist within the tree. However, this problem can be solved with the semantic association system. A simple example will be given to illustrate the underlying ideas.

Consider the two dimensional packet filters shown in Table 16. These packet filters can be encoded with a two dimensional MBT. Figure 34 illustrates one particular MBT

Table 16: Two dimensional packet filters.

$\mathcal{F}$	$f_1 : w_1 = 4$	$f_2 : w_2 = 4$	
R_1	001*	110*	A_1
R_2	[0001, 0010]	[0101, 0110]	A_2

encoding of these filters where a stride of $c = 2$ is used for the tries of each field. The upper trie in the figure encodes the prefixes for field f_1, and the lower tries represent the prefixes for the second field of the packet filter specifications. A dotted line represents the interconnection between the f_1 and f_2 tries. The leaf nodes of the f_2 tries point to their associated action information containers.

In the following, two scenarios consisting of a non-matching and matching classification process will be considered with respect to Table 16 and Figure 34. For the first scenario, let the (f_1, f_2) fields of a packet P be given by $p = (p_1, p_2) = (0010, 0100)$. The packet classification process starts at the root of the f_1 trie. The binary sequence for p_1 yields a traversal that leads to a leaf node with f_2-pointers to both of the lower f_2 tries. Inspection of the first two bits of the binary sequence for $p_2 = (0100)$ indicates that the traversal should continue through the left-most f_2 trie. However, the final two bits of p_2 are not encoded within this trie. Hence, a packet with fields $p = (0010, 0100)$ do not match the packet filter specifications from Table 16, and, by definition, the default filter action will be applied to the packet.

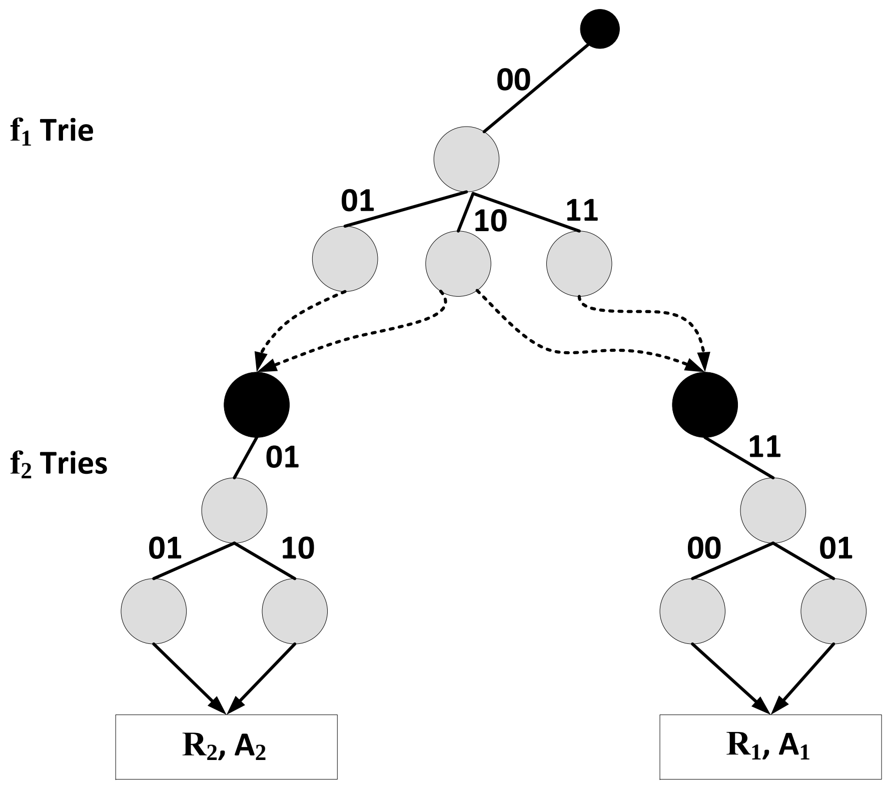


Figure 34: Two dimensional MBT representation of the packet filters from Table 16.

For the second scenario, let the packet header fields be given by $p = (p_1, p_2)$ where $p = (0011, 1100)$. The process begins by inspecting the first two bits of p_1 followed by the last two bits of p_1, which leads to the right-most leaf node of the f_1 trie. This leaf node has a single pointer to its associated f_2 trie, and traversal of this trie in accordance with the bit values of p_2 reveals that the packet matches the field specifications of the first packet filter (R_1). Therefore, the actions specified by A_1 will be applied to the packet.

The SPM algorithm begins by viewing the set of d-dimensional packet filters $R = \{R_1, R_2, \ldots, R_N\}$ as a collection of N unique multibit tries. Let these unique MBTs be referred to as *filter tries*. Each filter R_i is represented by a filter trie, and each filter trie is a collection of d interconnected MBTs that encode the prefixes for each of the header field specifications. Figure 35 illustrates the idea of independent MBT filter encoding. The tries for filter R_1 are shown on the left side of the figure, whereas the tries for R_2 are shown on the right side. Further, the tries for the f_1 fields are located at the upper half of the figure, and the f_2 tries are located at the bottom half of the figure. The dotted lines represent the

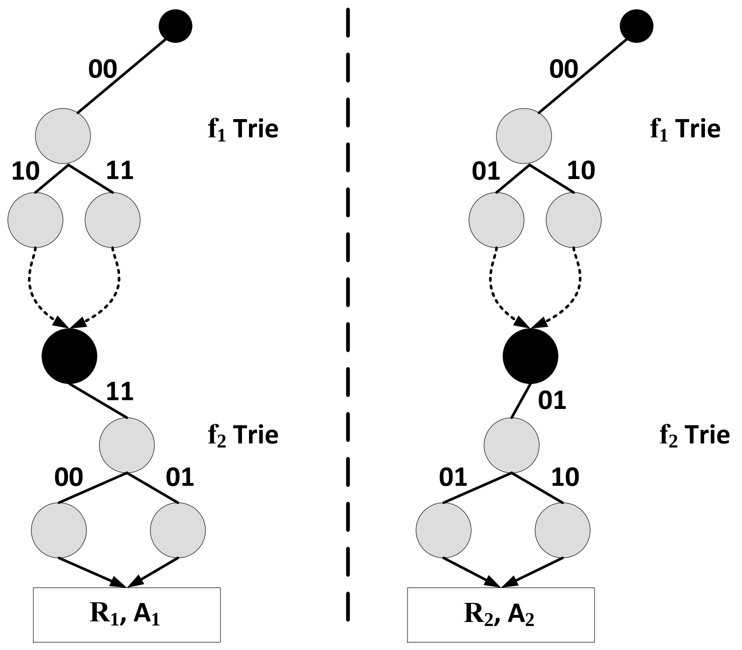


Figure 35: Independent MBT encodings of the packet filters from Table 16 with each filter having its own collection of prefix tries.

interconnections between the f_1 and f_2 tries.

Next, each unique filter trie is transformed to a directed acyclic graph (DAG), which will be referred to as a filter DAG (FDAG). During the transformation, the binary sequences defining the multiway branching (traversal) criteria of the trie are concatenated with unique symbols corresponding to the levels of the trie. The concatenated sequences produce unique labels for the vertices of the FDAGs. The following symbol-to-level correspondence will be used for this example:

$A \iff Level\ 1$

$B \iff Level\ 2$

$C \iff Level\ 3$

$D \iff Level\ 4$

Figure 36 shows the graphs produced when applying this transformation to the MBTs of Figure 35. The graphs G_1 and G_2 represent the filter tries for R_1 and R_2, respectively. For example, the node labeled $A00$ represents the branching criteria for the first two bits of the f_1 trie and the symbol A indicates that the branch leads to level 1 of the trie. Essentially, the

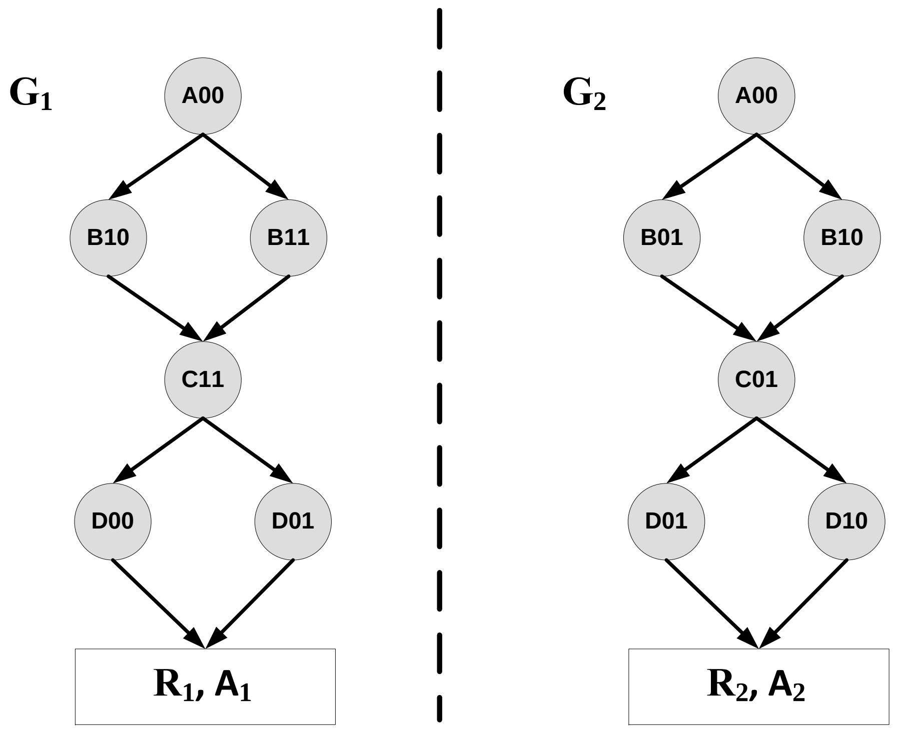


Figure 36: Labeled directed acyclic graph transformation of the MBTs from Figure 35.

transformation converts an edge-labeled trie traversal to a vertex-labeled DAG traversal. To illustrate the DAG traversal and packet matching process, consider the packet fields $p = (p_1, p_2) = (0011, 1100)$. The stride was chosen to be $c = 2$ for each field trie. As such, the bit fields of p are tokenized as follows: $(A00, B11, C11, D00)$. Inspection of G_1 in Figure 36 reveals the existence of the vertex path $(A00, B11, C11, D00)$. Hence, these packet fields match filter R_1.

After transformation of the filter tries, the resultant collection of FDAGs are superimposed onto a common graph structure. In other words, a new graph G^* is constructed by merging the collection of filter DAGs:

$$G^* = \bigcup_{i=1}^{N} G_i$$

G^* is denoted as the path-merged DAG (PMDAG). The PMDAG shown in Figure 37 represents the merger of G_1 and G_2 from Figure 36.

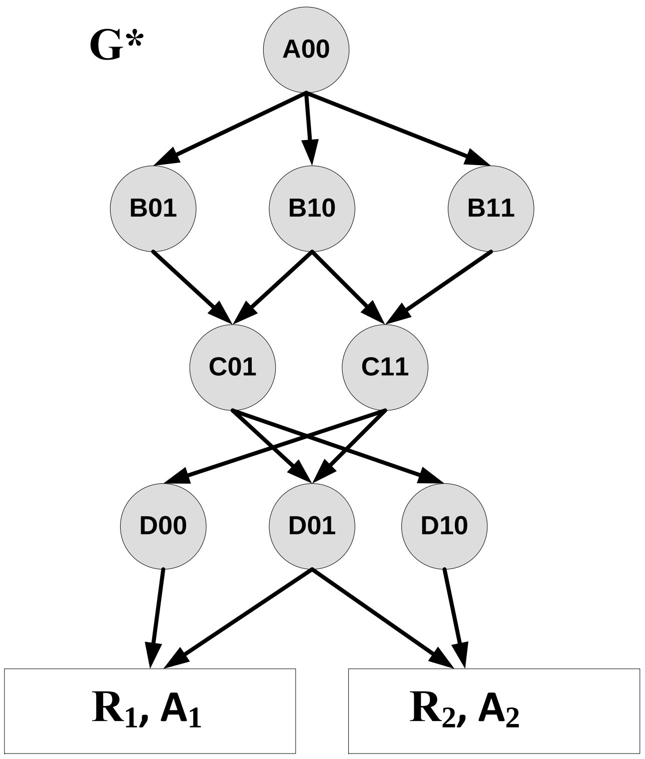


Figure 37: The directed acyclic graph produced by merging the filter graphs of Figure 36.

An individual FDAG represents (encodes) a particular packet filter. Specifically, the i^{th} FDAG, G_i, is a vertex-labeled graph and encodes the i^{th} packet filter R_i. The vertex-labeled paths (VLPs) in G_i correspond to the edge-labeled prefix paths of the i^{th} filter trie. Consequently, the path merged graph G^* encodes the set of packet filters $R = \{R_1, R_2, \ldots, R_N\}$. However, the path merging process as described thus far has an issue that must be addressed: ***Induced Transitive Relations***

Definition 14. *Given two vertex-labeled $s-t$ paths $V_{s-t} = (v_s, v_j, \ldots, v_k, v_t)$ and $U_{s-t} = (u_s, u_j, \ldots, u_k, u_t)$, V_{s-t} and U_{s-t} are $s-t$ equivalent, denoted by $V_{s-t} \equiv_{s-t} U_{s-t}$, if and only if $\forall v \in V_{s-t}, u \in U_{s-t} :\ (v_i = V_{s-t}(i) \ \wedge\ u_i = U_{s-t}(i) \ \wedge\ v_i = u_i)$.*

Thus far, usage of the term 'vertex-label' was to emphasize the vertex labeling process of the trie-to-DAG transformation. Henceforth, vertices and vertex-labels will be used synonymously. Merged paths that share one or more vertices are said to be *correlated vertex-label paths.*

Definition 15. *Given two vertex-labeled $s-t$ paths $V_{s-t} = (v_s, v_j, \ldots, v_k, v_t)$ and $U_{s-t} = (u_s, u_j, \ldots, u_k, u_t)$, V_{s-t} and U_{s-t} are correlated vertex-label paths, denoted by $V_{s-t} \sim_{s-t} U_{s-t}$, if $\exists v \in V_{s-t}, u \in U_{s-t}$ such that $v = u$. A vertex satisfying this equality criteria is called a locus of correlation. If $V_{s-t} \equiv_{s-t} U_{s-t}$, then V_{s-t} and U_{s-t} are fully correlated vertex-labeled paths. Otherwise,the paths are said to be partially correlated.*

Induced transitive relationships (ITR) result from the merger of two or more vertex-labeled paths that are not $s-t$ equivalent but share one or more vertex-labels. In other words, a partially correlated vertex path produced by the path merger process induces transitivity.

An example will be given to explain ITR. Let the graphs G_1 and G_2 be constructed by the following:

$$G_1 = (V_1, E_1): \ V_1 = \{A, C, D\}, \ E_1 = \{(A, C), (C, D)\}$$
$$G_2 = (V_2, E_2): \ V_2 = \{B, C, E\}, \ E_2 = \{(B, C), (C, E)\}$$

G_1 contains the $s-t$ path (A, C, D), and G_2 contains the path (B, C, E). Figure 38 illustrates the construction of G^* from these two graphs, which yields the following constituents.

$$G^* = (V^*, E^*) = G_1 \bigcup G_2$$
$$V^* = \{A, B, C, D, E\}$$
$$E^* = \{(A, C), (B, C), (C, D), (C, E)\}$$
$$R^* = \{R_1, R_2, R_3, R_4\}$$
$$R_1 = (A, C, D)$$
$$R_2 = (B, C, E)$$
$$R_3 = (A, C, E)$$
$$R_4 = (B, C, D)$$

The path merger process induces two new paths in G^* that do not exist in G_1 or G_2.

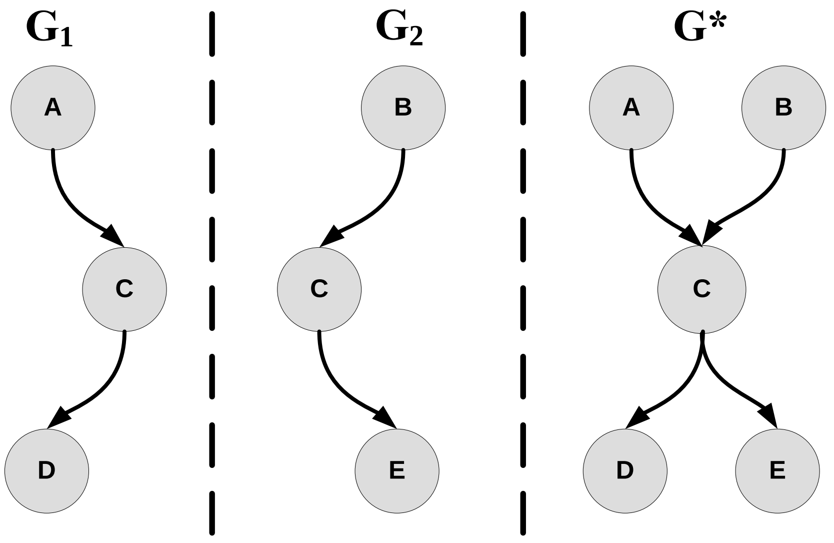


Figure 38: A path-merged DAG containing induced transitivity.

Namely, the paths R_3 and R_4 are *phantom paths* with respect to G_1 and G_2, and these phantom paths are established via induced transitive relationships, which result from merging the partially correlated paths R_1 and R_2. In this case, vertex C is the locus of correlation.

The methodologies provided by the theory of semantic association systems can eliminate the problems caused by induced transitive relationships. The solution can be realized by reformulating the FDAG path merger processes (PMP) within the context of a semantic association system. The connection can be observed with the correspondences listed in Table 17. A filter DAG is equivalent to a semantic family, a vertex-labeled path is viewed

Table 17: Correspondences between the path merger processes and the semantic association network.

FDAG	$\Longleftrightarrow$	Semantic Family
VLP	$\Longleftrightarrow$	Semantic Relations
PMP	$\Longleftrightarrow$	Semantic Space (merged conceptualization)

as a semantic relationship, the set of VLPs encoded by a particular FDAG form a collection of semantic relationships, and the path merger process is considered to be a merged conceptualization that produces a semantic space.

The construction of a path-merged DAG, as described above, can potentially generate a set of one or more phantom paths. Phantom paths are formed when a collection of two or more VLPs, which belong to different FDAGs and possess a locus of correlation, are merged onto a common PMDAG. The correlation loci created by the path merger process destroys the *path uniqueness* property possessed by the individual FDAGs to diminish. Consequently, the question becomes: "*How can path uniqueness be maintained?*" The answer is semantic coloring, and the result is semantic path merger. In other words, the semantics of a path are conserved by coloring them during the merger process.

The SPM algorithm starts by constructing a common hypergrid upon which the FDAGs will be superimposed. Given a collection of d-dimensional packet filters R, select the strides c_i for the MBTs of each dimension $1 \leq i \leq d$. In general, the stride of an MBT can be variable. For the remainder of this discussion, however, and with respect to the SPM algorithm, fixed stride tries will be assumed. Further, it is assumed that $c_i \mid w_i$ and by definition, $1 \leq c_i \leq w_i$.

Each w_i for the i^{th} field is associated with a stride c_i. Let this association be denoted by $W_c = (w_i, c_i)$. Let $D = [0, 2^c - 1] \subset \mathbb{Z}^+$ denote the subinterval domain of positive integers generated by stride c. The number of subinterval domains generated by W_c can be calculated as $\delta_i = \frac{w_i}{c_i}$. Equivalently, δ_i corresponds to the number of levels of an MBT with w_i bits and stride c_i. The total number of levels (subinterval domains) is denoted as δ^* where $\delta^* = \sum\limits_{i=1}^{d} \delta_i$

For each c_i, construct the collection of subinterval domains $\{D_{ij} = [0, 2^{c_i} - 1]\}$ for $1 \leq j \leq \delta_i$. The cross-product of the subinterval domains generated by W_c represents the complete domain spanned by w_i. This equivalence is shown by Equation 2 and states that the δ_i-dimensional hyperbox generated by the cross-product of the collection $\{D_{ij}\}$ is equivalent to the 1-dimensional line generated by $\hat{D}$.

$$(D_{i1} \times D_{i2} \times \ldots \times D_{i\delta_i}) \;\equiv\; (\hat{D} = [0, 2^{w_i} - 1]) \tag{2}$$

Equation 2 suggests the notion of dimensional folding and is illustrated by Figure 39 for the case of $w = 4$ and $c = 2$. Observe that $\hat{D} = [0, 2^4 - 1] = [0, 15]$, $D_1 = [0, 3]$, $D_2 = [0, 3]$, and $D_1 \times D_2 = [0, 3] \times [0, 3]$.

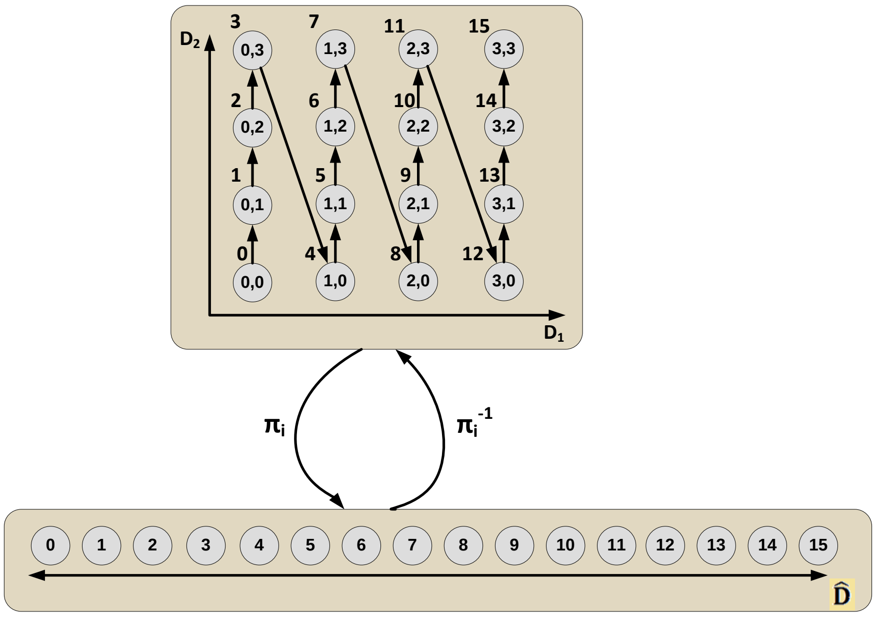


Figure 39: Dimensional folding as the cross-product of subinterval domains generated by $W_c = (w, c)$.

Given the subinterval domains, the hypergrid vertex sets are created as follows. Let $V^* = \{V_{ij}\}$ be the collection of hypergrid vertex sets where $1 \le i \le d$ and $1 \le j \le \delta_i$. For each V_{ij} and D_{ij}, let $k \in D_{ij}$ and for $0 \le k \le 2^{c_i} - 1$, define $V_{ij}(k) = k$. In other words, labels are assigned to the vertex sets such that a one-to-one correspondence is established between vertex labels and the integers belonging to the subinterval domains.

To illustrate these concepts, let the field parameters for a $d = 3$ dimensional packet classifier be given as in Table 18. The parameters from Table 18 yield the following con-

Table 18: Parameters for a 3-dimensional packet classifier illustrating the construction of the SPM hypergrid.

Field	w_i	c_i	δ_i
f_1	4	2	2
f_2	2	1	2
f_3	4	2	2

stituents.

$D_{11} = [0, 3]$

$D_{12} = [0, 3]$

$D_{21} = [0, 1]$

$D_{22} = [0, 1]$

$D_{31} = [0, 3]$

$D_{32} = [0, 3]$

$V_{11} = \{0, 1, 2, 3\} = \{00, 01, 10, 11\}$

$V_{12} = \{0, 1, 2, 3\} = \{00, 01, 10, 11\}$

$V_{21} = \{0, 1\} = \{0, 1\}$

$V_{22} = \{0, 1\} = \{0, 1\}$

$V_{31} = \{0, 1, 2, 3\} = \{00, 01, 10, 11\}$

$V_{32} = \{0, 1, 2, 3\} = \{00, 01, 10, 11\}$

$V^* = \{V_{11}, V_{12}, V_{21}, V_{22}, V_{31}, V_{32}\}$

The elements of each V_{ij} are enumerated in both decimal and binary format.The resultant hypergrid with $\delta^* = 6$ levels is shown in Figure 40. The level indicators, i.e., $A, B, \ldots, F$, are not applied to the vertices and are implied based on the levels of the hypergrid. Further, a dashed line has been applied to the grid to illustrate the grid regions corresponding to each field.

Once the hypergrid has been constructed, the independent filter tries for each of the R_i packet filters are generated, a unique semantic color is assigned to each filter trie, and the resultant colored filter tries are superimposed onto the hypergrid. In terms of the semantic association system, the final graph produced by this procedure is the semantic association network G_ϕ. Moreover, the set of vertices of the hypergrid and G_ϕ are the same: $V^* = V_\phi$.

Let a set of packet filters corresponding to the parameters of Table 18 be specified by Table 19. The semantic association network generated by these filters in conjunction with the stride parameters of Table 18 is shown in Figure 41.

An interesting interpretation of the semantic association network from Figure 41 can be made. The colored paths represent the complete set of cross-products defined by the

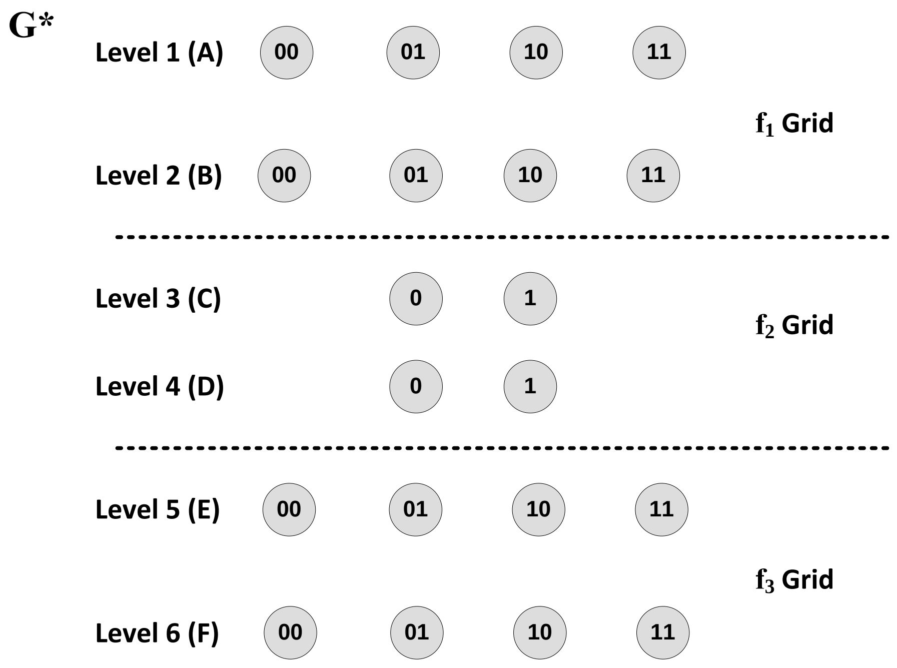


Figure 40: The hypergrid generated from the parameters in Table 18.

Table 19: A set of packet filters with parameters given by Table 18.

Filter	f_1	f_2	f_3	Action	Semantic Color
R_1	0000	$0*$	$000*$	A_1	$\lambda_1 =$ red
R_2	0101	11	0101	A_2	$\lambda_2 =$ blue
R_3	$111*$	$1*$	$111*$	A_3	$\lambda_3 =$ green

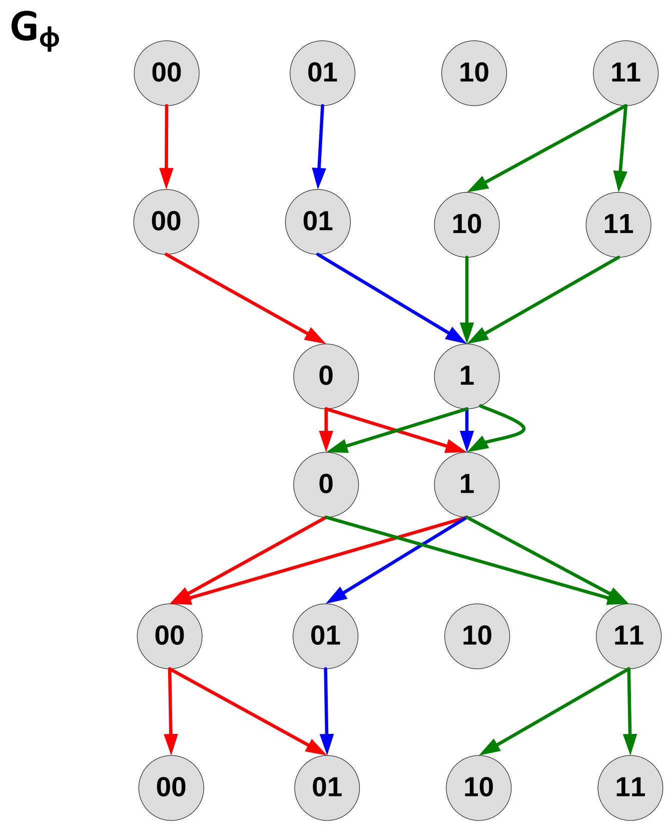


Figure 41: The semantic association network encoding the filters from Table 19.

subinterval domains over the fields of the packet filters. Let π_i denote the cross-products of the subinterval domains for the i^{th} packet filter R_i. The cross-products defined by the filters of Table 19 and encoded by the SAN of Figure 41 are listed below:

$\pi_1 = \{00\} \times \{00\} \times \{0\} \times \{0, 1\} \times \{00\} \times \{00, 01\}$

$\pi_2 = \{01\} \times \{01\} \times \{1\} \times \{1\} \times \{01\} \times \{01\}$

$\pi_3 = \{11\} \times \{10, 11\} \times \{1\} \times \{0, 1\} \times \{11\} \times \{10, 11\}$

Let π_i^* denote the expansion of cross-product π_i. The expansion of π_1 yields the following collection:

$\pi_1^* = \{\{00, 00, 0, 0, 00, 00\}, \{00, 00, 0, 0, 00, 01\}, \{00, 00, 0, 1, 00, 00\}, \{00, 00, 0, 1, 00, 01\}\}$

Given a collection of sets $S = \{S_i\}$ where each $S_i = \{s_{ij}\}$, a transversal of S is the set T_S that contains exactly one element from each of the $S_i \in S$. Consequently, the colored paths in G_ϕ correspond to the space of allowable transversals defined by R_i and represented by π_i^*. Given a packet with header fields $p = (p_1, p_2, \ldots, p_d)$, each p_i with w_i bits is partitioned into δ_i groups of c_i bits, which produces the chunked header fields $p = (p_{ij})$. Observe that if p is a transversal of the cross-product expansion π_k^*, then p matches packet filter R_k.

At this point, all that remains is the construction of the colored-path hypergraph G_Φ and its path matrix (incidence matrix). The CPHG groups the vertices of the SAN by color where each hyperedge of G_Φ is the collection of vertices belonging to a common semantic color. The path matrix is the zero-one incidence matrix of the CPHG, whereby $\Phi[e_i, v_j] = 1$ if $v_j \in e_i$. Otherwise, $\Phi[e_i, v_j] = 0$. Given that the vertices have been partitioned as $V_\phi = \{V_{ij}\}$, let the path matrix be partitioned in a similar fashion such that $\Phi = \{\Phi_{ij}\}$ where for each $v \in V_{ij}$ a corresponding column exists in Φ_{ij}. The partitioned path matrix corresponding to the semantic association network from Figure 41 is given by Table 20.

Table 20: Partitioned path matrix corresponding to the colored paths from the semantic association network of Figure 41.

Φ_{11}	$v = 00$	$v = 01$	$v = 10$	$v = 11$
e_1	1	0	0	0
e_2	0	1	0	0
e_3	0	0	0	1
Φ_{12}	$v = 00$	$v = 01$	$v = 10$	$v = 11$
e_1	1	0	0	0
e_2	0	1	0	0
e_3	0	0	1	1
Φ_{21}	$v = 0$	$v = 1$		
e_1	1	0		
e_2	0	1		
e_3	0	1		
Φ_{22}	$v = 0$	$v = 1$		
e_1	1	1		
e_2	0	1		
e_3	1	1		
Φ_{31}	$v = 00$	$v = 01$	$v = 10$	$v = 11$
e_1	1	0	0	0
e_2	0	1	0	0
e_3	0	0	0	1
Φ_{32}	$v = 00$	$v = 01$	$v = 10$	$v = 11$
e_1	1	1	0	0
e_2	0	1	0	0
e_3	0	0	1	1

Once the packet filters have been encoded by the path matrix, packet classification can be performed with simple bit vector operations. Given a packet with chunked header fields $p = (p_{ij})$, the objective is to find the set of valid transversals defined by π_i^* that are matched with p. From the set of matched transversals, the algorithm returns the semantic color associated with the highest priority matching transversal. Let $\Phi_{ij}(p_{ij})$ be the column-wise bit vector of Φ_{ij} indexed by the header chunk value p_{ij} and let ϕ_p be the transversal resolution vector defined by:

$\phi_p = \bigodot_{\forall p_{ij} \in p} \Phi_{ij}(p_{ij})$

The transversal resolution vector results from the bit-wise conjunction of the path matrix columns, which are indexed by the corresponding partitioned values of the chunked packet header fields. The resolution vector is a characteristic vector representing the set of filters matched by p. In other words, if $\phi_p(k) = 1$, then p matches the k^{th} packet filter R_k. If ϕ_p is the zero-vector, then the packet has no matching packet filter.

As an example, let the packet header fields be given by $p = (p_1, p_2, p_3) = (1110, 10, 1110)$. Based on the parameters given by Table 18, the chunked header fields are given by $p = (p_{11}, p_{12}, p_{21}, p_{22}, p_{31}, p_{32}) = (11, 10, 1, 0, 11, 10)$. Let $< x >^T$ denote the transpose of the vector $< x >$. The bit vectors extracted by this particular set of chunked header fields from the path matrix given in Table 20 are the following:

$\Phi_{11}(p_{11} = 11) =< 0, 0, 1 >^T$

$\Phi_{12}(p_{12} = 10) =< 0, 0, 1 >^T$

$\Phi_{21}(p_{21} = 1) \ =< 0, 1, 1 >^T$

$\Phi_{22}(p_{22} = 0) \ =< 1, 0, 1 >^T$

$\Phi_{31}(p_{31} = 11) =< 0, 1, 1 >^T$

$\Phi_{32}(p_{32} = 10) =< 0, 1, 1 >^T$

Bitwise conjunction of these vectors produces a transversal resolution vector given by $\phi_p =< 0, 0, 1 >^T$, which implies that a packet with header fields $p = (1110, 10, 1110)$ corresponds to semantic color $\lambda_3 = green$ and, therefore, the packet matches filter R_3 from Table 19.

Figure 42 provides a block diagram illustrating a hardware-based implementation of the SPM algorithm. In essence, the hardware design of the SPM algorithm can be viewed as

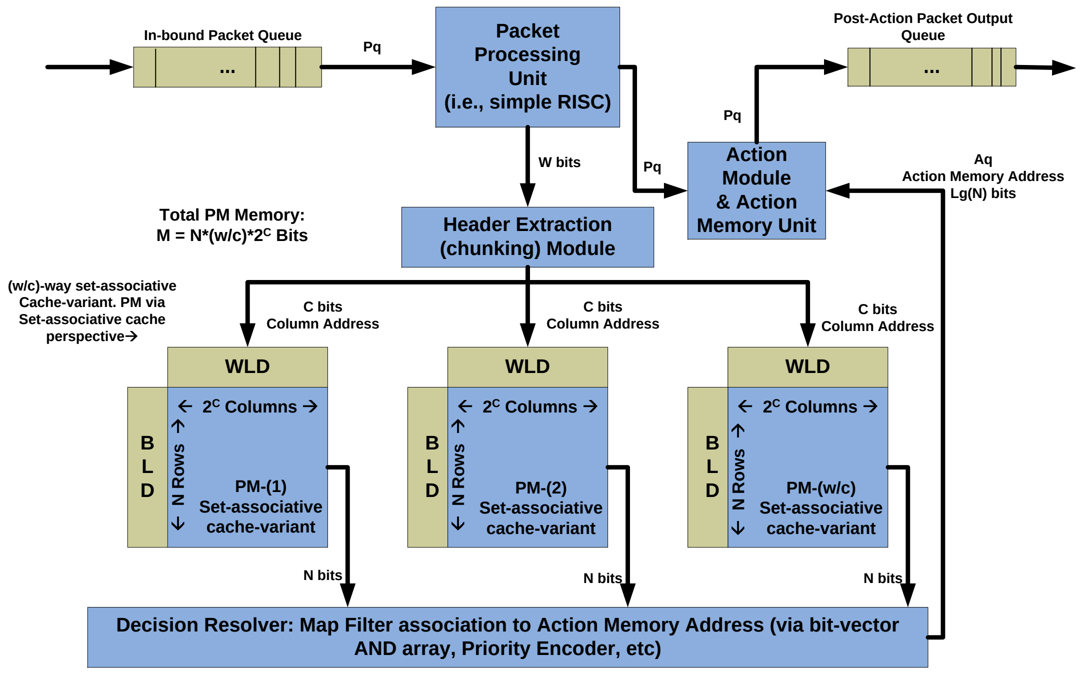

In-bound Packet Queue
...
Pq
Packet Processing Unit (i.e., simple RISC)
Post-Action Packet Output Queue
...
Pq
W bits
Pq
Action Module & Action Memory Unit
Aq
Action Memory Address
Lg(N) bits
Total PM Memory:
M = N*(w/c)*2^C Bits
Header Extraction (chunking) Module
(w/c)-way set-associative Cache-variant. PM via Set-associative cache perspective→
C bits Column Address
C bits Column Address
C bits Column Address
WLD
WLD
WLD
BLD
BLD
BLD
← 2^C Columns →
← 2^C Columns →
← 2^C Columns →
← N Rows →
← N Rows →
← N Rows →
PM-(1) Set-associative cache-variant
PM-(2) Set-associative cache-variant
PM-(w/c) Set-associative cache-variant
N bits
N bits
N bits
Decision Resolver: Map Filter association to Action Memory Address (via bit-vector AND array, Priority Encoder, etc)

Figure 42: Block diagram illustrating a hardware-based implementation of the SPM algorithm.

a variant of the traditional m-way set-associative cache architecture. The δ_i path matrix partitions correspond to the *ways* of a cache. The *sets* of the architecture correspond to the 2^{c_i} columns for each partitioned path matrix. The hardware design consists of a simple top-level packet processing unit for extraction of packet header fields. These fields are then delivered to a header chunking module that extracts bit-chunks based on the stride values c_i. The values of each chunk are then used as direct indexes into their corresponding memory partitions. The indexes activate the wordline drivers (WLD) for each corresponding column and the N-bit words addressed by the chunks are accessed via traditional bitline driver (BLD) read operations. The N bits from each partition are accessed in parallel and delivered to the decision resolver module. In general, resolution will depend on the application. For priority-based packet classification, the decision resolver will perform bitwise AND operations across the δ_i bit vectors. The output of this conjunction will be delivered to a priority encoder that translates the result and delivers a lgN bit result the to action module. This value is used by the action module to process the packet according to the action associated with the highest priority filter that matched.

The complexity analysis of the SPM algorithm is the following. The SPM algorithm can be implemented in either hardware or software. Let M be the length of a machine word. For each of d dimensions with $1 \leq i \leq d$, the path matrix has $\frac{w_i}{c_i}$ partitions with 2^{c_i} columns having a length of N bits. Therefore, the space complexity of SPM is $O(dN\frac{w}{c}2^c)$. For the general case where M bits are read per operation, the algorithm requires $\frac{N}{M}$ memory accesses for each of the $\frac{w}{c}$ partitions over each of the d dimensions. Therefore, the time complexity of the SPM algorithm is $O(d\frac{N}{M}\frac{w}{c})$.

For the hardware design presented above, it is assumed that the N bits of each column are read in parallel and that each partition is operating concurrently. In this case, $M = N$ and concurrent operation of the $d\frac{w}{c}$ partitions gives the hardware implementation of the algorithm an $O(1)$ time complexity.

6.2 Simulation Methodology

Power consumption is a key design parameter of modern computing architectures and digital systems [80]. Consequently, the designs underlying hardware-based packet classification algorithms should consider the various trade-offs in terms of energy, delay, and power. In this section, simulation results and comparative analysis of the performance metrics for the hardware-based SPM algorithm versus content addressable memory (CAM) are given.

6.2.1 The Cacti Integrated Memory Simulator

CACTI is an integrated memory simulation tool used by computer architects [58] to evaluate diverse memory architectures and configurations along with the associated performance trade-offs such as area, delay, and power [159, 116, 136, 144, 153, 104]. CACTI has been incorporated into several simulation frameworks such as the Structural Simulation Toolkit (SST) [62], Orion [157, 73], McPat [87], Waatch [21], and eCACTI [93]. According to Muralimanohar et al. [105], CACTI has been cited by over 1000 published research papers. These observations show how important CACTI has become throughout the years for modeling and analysis of computer architectures.

CACTI version 6.5 (CACTI6.5) was used to evaluate the energy, time, and power components of the proposed hardware-based implementation of the SPM algorithm. The source code for the numerous versions of CACTI along with the associated technical reports can be retrieved from [61]. Appendix A provides a sample configuration listing for CACTI6.5 along with comments describing the options that are available.

McPat [87] is an architectural-level simulator for multicore and manycore computer architectures. It incorporates a modified version of CACTI6.5. Specifically, McPat extends the functionality of CACTI6.5 and provides designers with the capability to model standalone CAMs. Appendix C provides a sample configuration listing for the modified version of CACTI6.5 included with McPat along with comments describing the options that are available. For the remaining discussion, usage of the term CACTI will be in reference to version 6.5.

6.2.2 Simulation performance metrics and parameters

Computer architects have a plethora of design alternatives that must be considered during the initial stages of memory system design. Some of these alternatives include transistor type, wire type, signaling methods, banking strategies, and access methods, just to name a few. Particular combinations of these alternatives lead to different performance characteristics. These performance characteristics must be coupled with other design constraints dictated by a given problem such as total memory capacity, timing requirements, and power budgets. Consequently, a great deal of effort is needed in order to find an appropriate combination of alternatives that satisfy the design constraints. CACTI was created as a simulation framework seeking to provide computer architects and memory designers with an integrated tool that takes high-level design configurations as input and performs an exhaustive design space exploration in order to find an optimal configuration in terms of power, area, and time constraints.

Table 21 lists the main performance metrics used for the analysis of the hardware-based SPM algorithm and comparisons with content addressable memories. For the SPM and CAM implementations, energy per operation (EPO) is the amount of energy consumed to

Table 21: Metrics used to evaluate the performance of the SPM hardware algorithm.

Symbol	Description	Units
EPO	Energy per Operation	nJ (nano-Joule)
TPO	Time per Operation	ns (nano-seconds)
EDP	Energy Delay Product	nJ*ns
Power	Power Consumption	W (Watts)

classify a single packet. Time per operation (TPO) is the amount of time required by the hardware to classify a single packet. The total amount of power consumption consists of the device's dynamic power used during an operation and static power results from transistor leakage current.

6.3 Simulation Results

The experimental results produced by CACTI simulations are given in this section. The design constraints for the SPM algorithm and the CAM are a function of the number of filters N and the header field width w. Further, the stride c is also used as a design parameter for the SPM algorithm. The stride determines the total number of path matrix partitions as well as the number of columns contained in each partition. Particularly, given a stride of c, the total number of partitions is given by $\delta = \frac{w}{c}$ and the number of columns in each partition is given by 2^c. Therefore, the total number of bits N_b required for each partition is given as:

$N_b = N2^c$

and the total number of bits used by the partitioned path matrices is given as:

$N_\Phi = N\frac{w}{c}2^c$.

The selection of stride c along with design constraints N and w determine the overall performance of the SPM algorithm.

The raw data presented by the plots of this chapter are located in Appendix E. In the following figures, data are presented as panel plots. The x-axis varies N from 1024 to 262,144 in powers of 2. The x-axis is partitioned into panels for each value of the variable w. The values selected for w were 16, 32, 64, and 128. The SPM algorithm was configured over three different stride values, including $c = 2$, $c = 4$, and $c = 8$. The curves for the SPM algorithm were labeled by SPM-c2 (blue curves), SPM-c4 (red curves), and SPM-c8 (green curves) for the strides 2, 4, and 8, respectively.

Figures 43, 44, and 45 plot simulation results comparing EPO, TPO, and EDP, respectively. As seen in Figure 43, the EPO for each algorithm increases with both N and w. For small values of N and w, SPM and CAM perform similarly. However, as N or w increases, the CAM's EPO becomes significantly larger than SPM. SPM with stride values 2 and 4 give similar EPO. However, a stride of 8 reveals larger increases in EPO versus N and w.

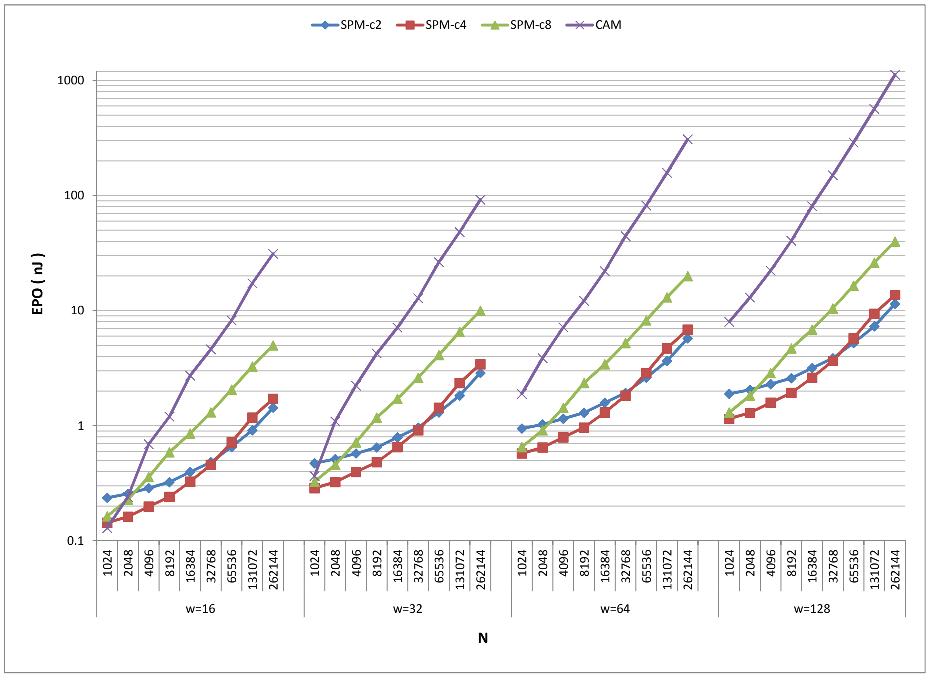


Figure 43: Comparing EPO versus (N, w, c) for the SPM algorithm and the content addressable memory.

Figure 44 plots simulation results for TPO. The TPO results are interesting and require several explanations. Similar to EPO, SPM and CAM show increases in TPO as N increases. TPO is a measure of memory access delay. It is the amount of time required to access the contents of memory. Access time and energy consumption depend on hardware configurations such as memory layout and aspect ratios. CACTI performs layout optimization, which can vary for different devices and different memory sizes. These optimizations can be seen in Figure 44 where the CAM has some cases such that a larger value of N

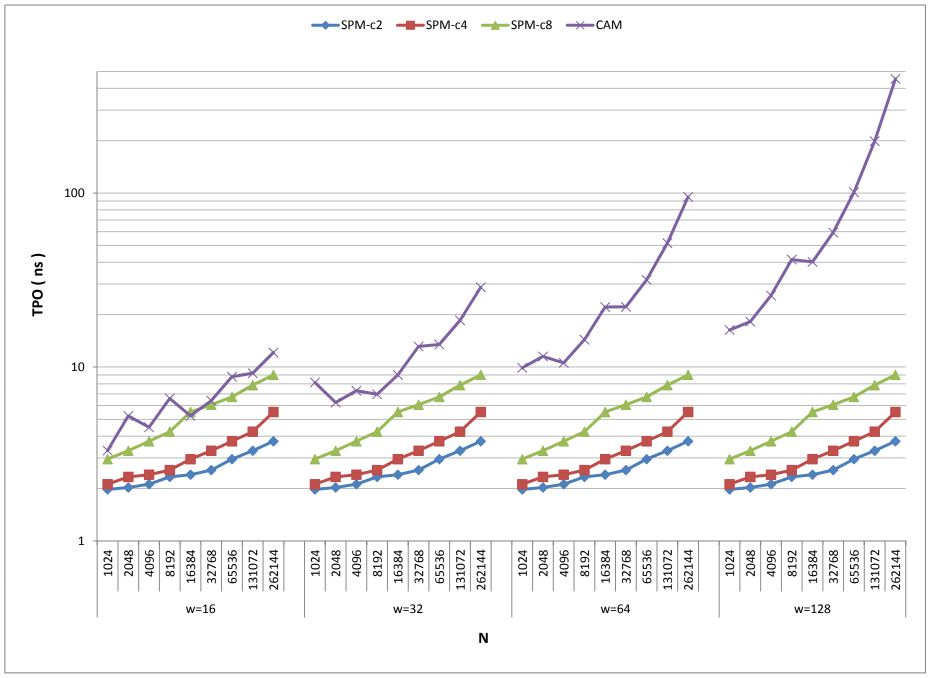


Figure 44: Comparing TPO versus (N, w, c) for the SPM algorithm and the content addressable memory.

performs better than a smaller value of N.

Another observation from the TPO plots is how the parameter w affects TPO. Both SPM and CAM have EPO that increases with N and w. However, TPO does not change with w for SPM. The reason is that w is not a direct factor for a single SPM path matrix partition. For a given value of w and c, SPM requires $\frac{w}{c}$ partitions. Each partition has 2^c columns containing N bits per column. The memory consumption is exactly $N2^c$ bits for a *single* partition. In other words, the TPO for a single partition is impacted by N and c, and the complete SPM system operates $\frac{w}{c}$ partitions in parallel. The dependence on c for SPM's TPO is clearly obvious from the TPO plots. For larger c, SPM requires more time. Moreover, SPM's TPO increases faster for larger values of c.

Figure 46 reveals the dynamic power consumed by each design. The dynamic power of the systems *appear* to be comparable with each other. However, dynamic power results alone do not give a complete picture of the overall performance of the algorithms. For example, the dynamic power consumption of the four algorithms for $w = 128$ indicate very little preference over each other. However, when these metrics are coupled with their

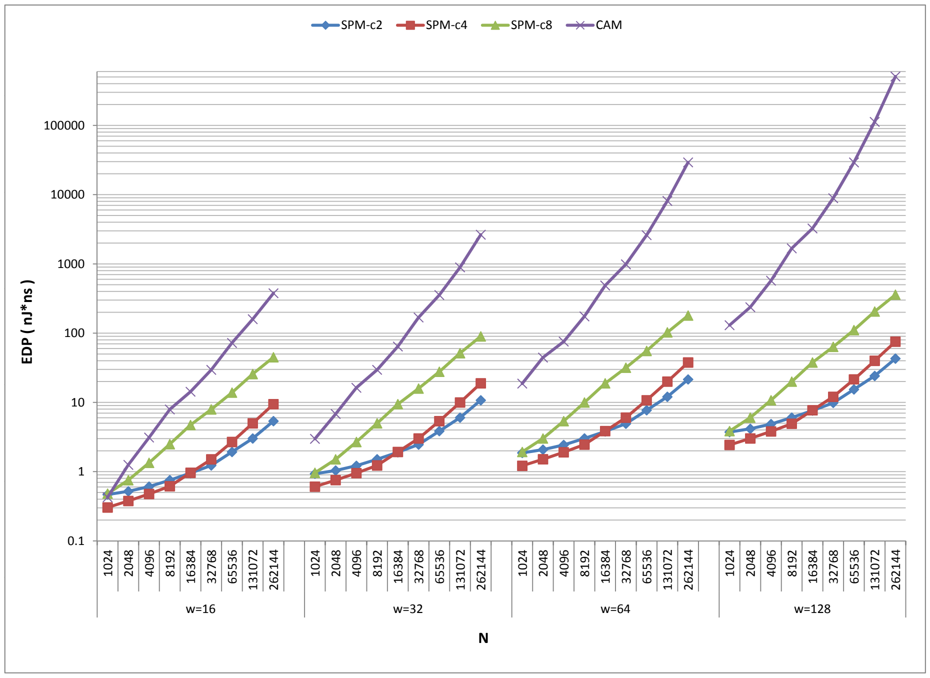


Figure 45: Comparing EDP versus (N, w, c) for the SPM algorithm and the content addressable memory.

associated energy-delay-products, as given by Figure 45, appropriate preference becomes obvious. Particularly, EDP for the CAM can be orders of magnitudes larger than SPM for any value of c. The result when coupled with dynamic power is that for a similar dynamic power budget, the SPM algorithm can process a much larger number of packets per second. Conversely, the SPM algorithm can process packets at speeds comparable to the CAM but with significantly less power.

The total static power of each algorithm is provided in Figure 47. Static power is a result of transistor leakage current. For $w = 16$ and $w = 32$, the static power of the CAM coincides closely with the SPM algorithm with stride $c = 8$. The total number of bits used by the SPM algorithm grows exponentially with c. Hence, for larger values of c, the SPM algorithm will suffer from greater amounts of static power consumption.

Figure 48 provides the total power consumption (static + dynamic) of the four algorithms. For $c = 8$ and $w = 16$, very little difference exists between SPM and CAM. Similar to the other performance metrics, SPM with $c = 2$ and $c = 4$ perform comparable to each other. Further, as w increases, the total power of the CAM increases faster that SPM.

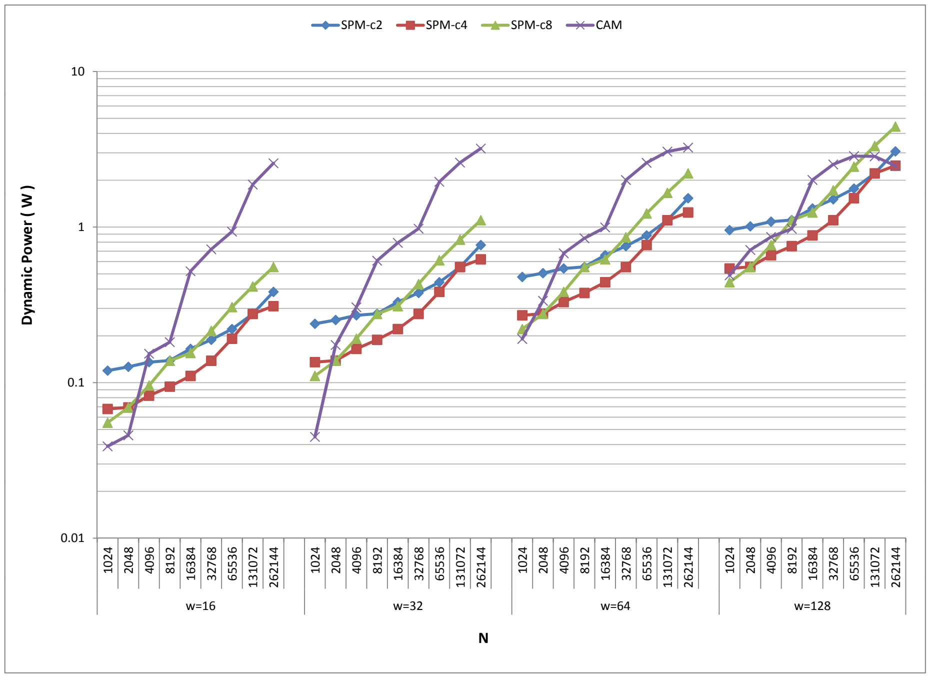


Figure 46: Comparing dynamic power versus (N, w, c) for the SPM algorithm and the content addressable memory.

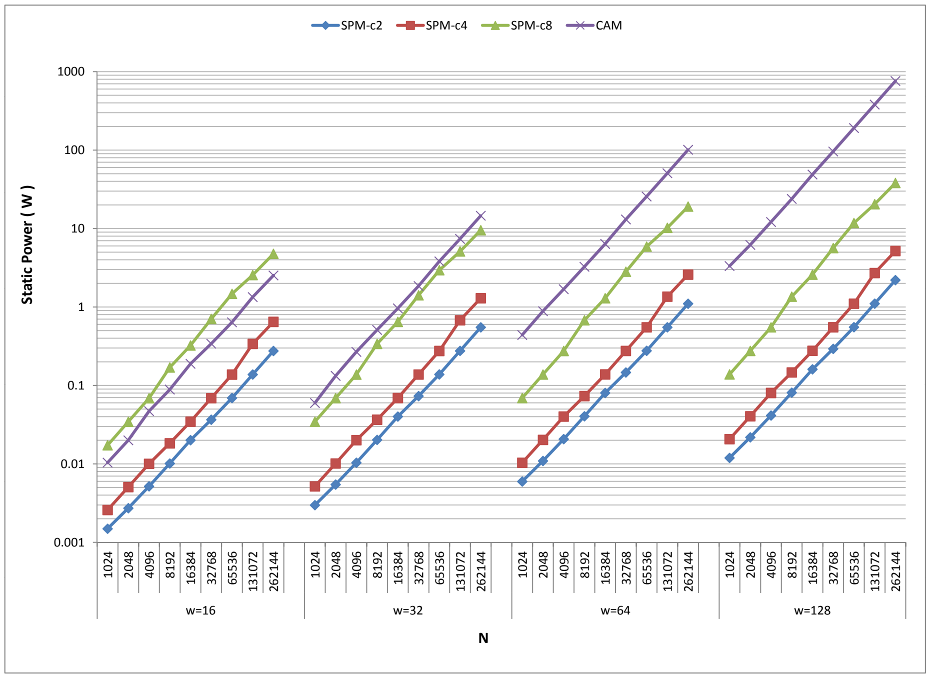


Figure 47: Comparing static power versus (N, w, c) for the SPM algorithm and the content addressable memory.

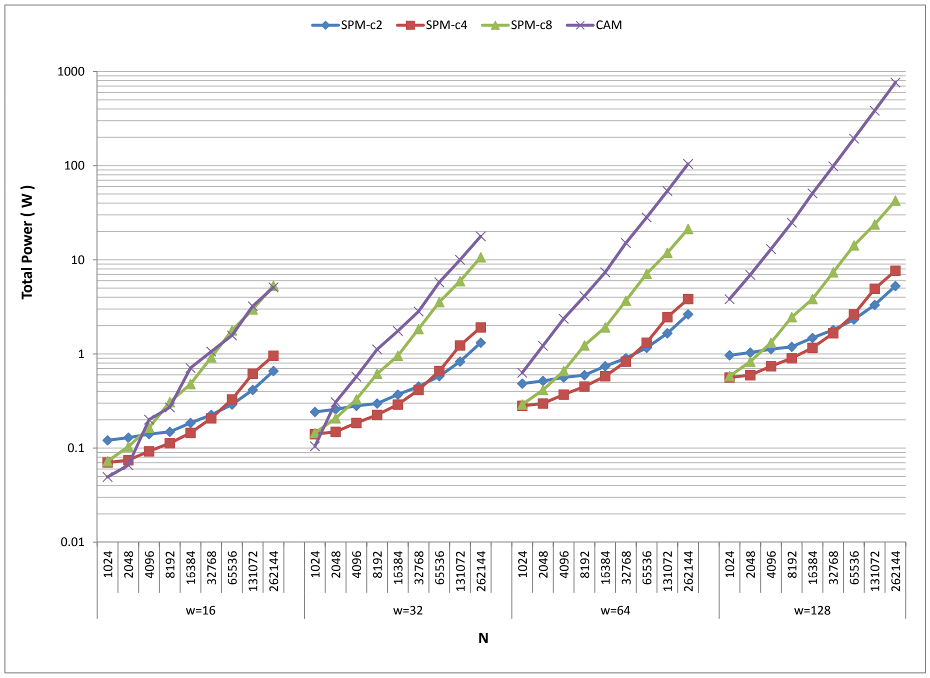


Figure 48: Comparing total power versus (N, w, c) for the SPM algorithm and the content addressable memory.

Based on the simulations results presented in this section, it can be seen that, excluding a few configurations with small N and w, the SPM algorithm significantly outperforms the CAM. Moreover, the performance difference between the two systems grows faster as w increases. Based on these results, the SPM algorithm appears to perform more optimally with stride values of $c = 2$ and $c = 4$.

6.4 Summary

In this chapter, the SPM hardware-based packet classification algorithm was introduced. SPM is based on the idea of merging the prefixes defined by filter specifications onto a common graph structure. The constructs of the semantic association system are applied to the prefix merger process, and the path matrix generated by the process is used as a straightforward mechanism that can be constructed with slight modifications to the traditional set-associative cache architecture. Simulation results produced by the well-known CACTI memory simulation framework have revealed that the SPM algorithm can significantly outperform content addressable memories in terms of energy per operation, time per operation, energy delay product, dynamic power, static power, and total power.

CHAPTER VII

ADVANCED CYBERATTACK DETECTION WITH COMPUTATIONAL INTELLIGENCE SYSTEMS

Traditional cybersecurity architectures incorporate security mechanisms that provide services such as confidentiality, authenticity, integrity, access control, and non-repudiation. These services are used extensively to prevent computer and network intrusions and attacks. For instance, access control services prevent unauthorized access to cyberresources such as computers, networks, and data. However, the modern Internet security landscape is characterized by attacks that are voluminous, constantly evolving, extremely fast, persistent, and highly sophisticated [132, 9]. These characteristics impose significant challenges on preventive security services. Consequently, methodologies that enable autonomic detection and response to cyberattacks should be employed synergistically with prevention techniques in order to achieve effective defense-in-depth strategies and robust cybersecurity systems [63, 72, 123].

Cyberattack detection systems require algorithms that collect and analyze data generated by various events occurring within a cyber environment. The objective of a detection algorithm is to accurately discover suspicious activities based on the analysis of event data. This objective is fundamentally important as it forms the core of any attack detection system. However, the objective is hard to achieve, especially in terms of ***accuracy***. A detection algorithm that generates inaccurate results can negatively impact the performance of the entire system. Axelsson [12] claims that the performance of an intrusion detection system, in terms of *effectiveness*, is limited by its false alarm rate. This performance limit is a consequence of the *base-rate fallacy*. For example, inaccurate detection algorithms generate large volumes of false alarms, which can lead to issues such as collateral damage, unnoticed detection of live attacks or intrusions, and unmanageable numbers of alarm notifications that overwhelm security administrators. Consequently, research has explored new algorithms and methodologies aiming to increase the performance and accuracy of detection systems [13, 45, 8, 167, 76].

The study of computational intelligence systems (CIS) is concerned with the theory and design of evolutionary and adaptive systems that possess emergent behavior and intelligent decision making capabilities and that operate within complex and dynamic environments [156]. These systems are generally designed to cope with high dimensional and noisy data during their decision making processes. Since cyberattack detection systems are faced with large volumes of high dimensional data along with continuously evolving attack characteristics, computational intelligence systems have become logical choices to consider when designing new classification algorithms for detection systems.

Computational intelligence algorithms based on new hybridization and ensemble methodologies are presented in this chapter. The algorithms are constructed as generalized systems with no underlying domain-specific assumptions influencing the design. However, the systems are evaluated as classification frameworks for cyberattack and intrusion detection.

7.1 Datasets and Performance Metrics for Evaluating Cyberattack Detection Systems

Appropriate datasets for training and testing classification algorithms along with reliable metrics to evaluate classification performance are needed for the design of effective cyberattack detection systems. This section discusses the datasets and performance metrics used to evaluate the classification algorithms proposed in this chapter.

7.1.1 Datasets

Datasets containing relevant features that characterize cyberattacks are needed for the design, evaluation, and comparative analysis of new classification algorithms for attack detection systems. However, datasets and testing environments for evaluating attack detection systems are rare [11]. Of the few datasets publicly available, the ones most frequently used by researchers were produced by the DARPA intrusion detection evaluation program [146]. The objective of the DARPA intrusion detection evaluation program was to produce a collection of standardized datasets that could be used to formally evaluate and objectively compare the performance of intrusion detection systems [90]. The datasets have played a critical role in the advancement of intrusion detection systems along with the development of new attack detection and classification algorithms. The DARPA datasets were collected at MIT Lincoln Laboratories during the years 1998, 1999, and 2000. These datasets contain various types of *audit* data with features representing normal and attack traffic.

The KDD CUP99 dataset, which was used as a benchmark for the Third International Knowledge Discovery and Data (KDD) Mining Tools Competition, is frequently used to evaluate intrusion and attack detection algorithms. The CUP99 dataset is a derivative of the 1998 DARPA dataset (DARPA98). DARPA98 contains audit data generated by simulated background traffic representing *normal* packet flows between a military network and the Internet along with traffic representing *attack* packet flows.

The CUP99 data are viewed as sequences of connection records representing unique packet flows. A flow can be defined, similar to the packet filters of a firewall, by specifying a matching criteria over some set of header fields. The canonical flow is specified by a 5-tuple containing source IP address, destination IP address, source port, destination port, and protocol type. The records are classified by two types of flows: attack flows or normal flows. The attack flows are further categorized by 24 unique attack types.

Each record of the CUP99 dataset contains 42 fields. One field provides a label specifying the record's flow type, i.e., normal or attack type. The remaining 41 fields are comprised of features representing data extracted from the flow. The features are categorized as basic TCP features, content features, network-based traffic features, and host-based traffic features. The features are encoded numerically or symbolically. The goal of the KDD competition was to use the CUP99 dataset as a benchmark for evaluating attack classification algorithms produced by the various competitors of the KDD mining tools competition.

The DARPA and CUP99 datasets are out of date. Further, these datasets have been criticized by several whose investigations have discovered various limitations and deficiencies of the data [96, 145]. However, the datasets remain widely used for testing and evaluation of attack detection and classification algorithms because there are no suitable alternatives currently available [113, 39].

In this chapter, classification algorithms for the cyberattack detection problem are introduced. Each algorithm was evaluated with the CUP99 dataset. Particularly, the well-known 10% CUP99 dataset was employed.

7.1.2 Performance Metrics

The ***accuracy*** of a classification algorithm is a key performance indicator that determines the algorithm's suitability for solving a particular problem. However, other performance indicators are commonly measured in conjunction with accuracy. For example, true positive rates and true negative rates measure a classifier's capability to correctly distinguish *positive* cases from *negative* cases. Classification problems based on two-class decision spaces can use positive and negative rates as performance measures. However, the positive and negative classes must be defined when designing the classifier. Many researchers who design algorithms for intrusion and attack detection problems define attack instances as the positive class and normal instances as the negative class. This is especially true for classifiers implementing anomaly detection. Anomaly detection is based on audit data that represents normal instances and instances deviating from the characteristics underlying the audit data are assumed to be anomalous and, by definition, indicative of an attack. Deviations indicate 'positively' that the instance is an anomaly. Hence, classification of an anomaly is defined to be positive whereas classification of a normal instance is defined to be negative, i.e., not anomaly.

The new classification algorithms proposed in this chapter for the attack detection problem were trained and tested with the CUP99 dataset. CUP99 contains records representing audit data that characterize both normal instances (normal traffic flows) and attack instances (attack traffic flows). Moreover, the dataset is comprised of a multiplicity of attack types. Consequently, CUP99 can be used to evaluate detection systems based on two-class or multi-class classification algorithms. The algorithms proposed in this chapter are designed as two-class classification systems.

Since CUP99 contains audit data characterizing normal and attack instances, several design scenarios can be considered. The data can be partitioned into normal classes for the design of anomaly detection, into attack classes for misuse detection, or the data can remain un-partitioned for mixed detection. The classification algorithms presented in this chapter were trained and tested as two-class mixed detection (non-partitioned) methodologies.

The mixed detection approach with two-class (binary) classification enables two perspectives for defining the positive and negative classes. These perspectives are illustrated by Table 22. The perspective defining the normal class to be positive and the attack class to be negative was chosen for the work described in this chapter.

Table 22: Perspectives of the positive and negative classes for mixed detection binary classification.

Positive	Negative
Normal	Attack (NOT Normal)
Attack	Normal (NOT Attack)

Several metrics are commonly used when evaluating the performance of classification algorithms. Classification accuracy is a key performance indicator of classification systems. However, accuracy measurements alone do not provide complete information for comparative analysis and optimization purposes. Other key performance indicators include metrics such as error rates, true/false positive/negative rates, and predictive rates.

Evaluations of the new algorithms introduced in this chapter where made with the basis parameters shown in Table 23 and the performance metrics shown in Table 24.

Table 23: Basis parameters of the performance metrics defined in Table 24.

Symbol	Description
TP	Total number of true positives
TN	Total number of true negatives
FP	Total number of false positives
FN	Total number of false negatives

7.2 *Cyberattack Detection with Hybrid Intelligent Systems*

Classification systems, in general, attempt to map the features of an input space to classes of an output space. For this to be accomplished *reliably*, the classification algorithm needs a

Table 24: Definitions and nomenclature of the performance metrics used to evaluate the proposed classification algorithms.

Name	Symbol	Calculation
Accuracy	ACC	$ACC = \frac{TP+TN}{TP+TN+FP+FN}$
Positive Prediction Rate	PPR	$PPR = \frac{TP}{TP+FP}$
Negative Prediction Rate	NPR	$NPR = \frac{TN}{TN+FN}$
False Discovery Rate	FDR	$FDR = \frac{FP}{TP+FP}$
False Positive Rate	FPR	$FPR = \frac{FP}{TN+FP}$
False Negative Rate	FNR	$FNR = \frac{FN}{TP+FN}$

collection of data that appropriately characterizes the input-output mappings and a learning algorithm that teaches the classifier how to assign the mappings.

In this section, a hybrid intelligent system based on self-organizing maps and Bayesian learning networks is proposed [152]. The details of the algorithm are provided along with its classification performance derived from simulation results obtained with the CUP99 dataset.

7.2.1 The HBSOM Classification Algorithm

The Bayesian Learning Network (BLN) is a graphical computation approach to learning and reasoning with domain specific data [111, 112]. The general construct of a BLN includes the representation of features from a given knowledge domain as the nodes of a directed acyclic graph (DAG). Directed edges between the nodes of a BLN represent feature space dependencies. Let X_i and X_j be two features (i.e., random variables) from a knowledge domain having an $n+1$ dimensional feature space described by a collection of domain specific data $D = \{(X_1, \ldots, X_n, Y)\}$. A directed edge $e = (X_i, X_j)$ in the BLN indicates a dependency between the features X_i and X_j, and the significance of a dependency is measured as a conditional probability. Particulary, the data D contains *prior* knowledge related to a specific domain. The BLN encodes this prior knowledge as a directed acyclic graph. The graph represents joint distributions of the complete feature space with a collection of conditional probabilities implied by edges of the DAG.

When viewed in the context of classification, a BLN seeks to provide answers to queries regarding existential questions of particular instances described by the feature space. The general objective of classification is to answer queries of the form given by Equation 3 where the collection $X_1, \ldots, X_n$ are features and Y is a class variable (or target class).

$$Q : X_1 \times X_2 \times \ \ldots \ \times X_n \longrightarrow Y \tag{3}$$

Bayes theorem provides the quantitative foundation for answering queries such as Equation 3 over the feature space represented by a BLN. Given two events A and B, Bayes theorem states the following:

$P(A|B) = \frac{P(B|A)}{P(B)} P(A)$

An answer to the query $y = Q(x_1, \ldots, x_n)$ submitted to a BLN is given by $P(y|x_1, \ldots, x_n)$. More specifically, the most appropriate answer to the query $Q(\cdot)$ given by the BLN should be the most probable value of $y \in Y$ given that $((x_1 \in X_1) \wedge (x_2 \in X_2) \wedge \ldots \wedge (x_n \in X_n))$. Let this *most probable* value be denoted by y^*. The representative data D contains a collection of known instances $(x_1 \in X_1, \ldots, x_n \in X_n, y \in Y)$ from which the joint probability distributions can be calculated, and Bayes theorem provides the quantitative machinery that enables the determination of y^*. Specifically, y^* is the value of $y \in Y$ that maximizes $P(Y|x_1, \ldots, x_n)$ as illustrated by Equation 4.

$$y^* = \arg\max_{y \in Y} P(y|x_1, \ldots, x_n) = \arg\max_{y \in Y} \frac{P(x_1, \ldots, x_n|y)}{P(x_1, \ldots, x_n)} P(y) \tag{4}$$

The term $P(x_1, \ldots, x_n)$ is independent of Y and, therefore, the arg max functionality from Equation 5 is independent of $P(x_1, \ldots, x_n)$, which simplifies to Equation 5.

$$y^* = \arg\max_{y \in Y} P(x_1, \ldots, x_n|y) P(y) \tag{5}$$

A special case of Bayesian learning networks occurs if the features are conditionally independent. When the features are conditionally independent, the BLN is a bipartite graph with edges directed from the features X_i to the target class Y and no edges between the collection of features. This special case is known as the naive Bayesian learning network

(NBLN). Conditional independence of the feature space allows Equation 4 to be rewritten as shown by Equation 6.

$$y^* = \arg\max_{y \in Y} \frac{P(x_1, \ldots, x_n|y)}{P(x_1, \ldots, x_n)} P(y) = \arg\max_{y \in Y} P(y) \prod P(x_i|y) \tag{6}$$

Self-Organizing Maps (SOM) belong to the class of unsupervised artificial neural networks (ANNs). The SOM transforms a high dimensional input domain to elements of a low dimensional array of neurons [82, 83]. The SOM is a grid of nodes indexed by grid coordinates $g = (i, j)$, and each node η_{ij} represents a neuron having a weight vector ω_{ij} of length n, where ω_{ij} is an element of the n-dimensional space of real vectors: $\omega_{ij} \in \mathbb{R}^n$. The SOM is a clustering algorithm. Given a set of training data $X = \{x_k\}$ where $x_k \in \mathbb{R}^n$, the goal of the SOM algorithm is to modify the weight vectors of the underlying neural grid in such a way to represent the set of training data X. To achieve this objective, the SOM employs a clustering algorithm based on the concept of elastic interconnections between the nodes of the neural grid.

The training algorithm of the SOM is described by the following. Let $d(x_k, \omega_{ij})$ be a function describing the distance between an input vector x_k and a weight vector ω_{ij}. Let the best matching unit (BMU) of the grid be described by the index $g^* = (i^*, j^*)$ that minimizes the distance function:

$g^* = \arg\min(d(x_k, \omega_{ij}))$

The SOM employs a weight update rule that modifies each weight vector based on the input vector and the distance of each node in the grid from the BMU. Equation 7 describes the SOM weight update rule.

$$\omega_{ij}(t) = \omega_{ij}(t-1) + \alpha(t)[x_k(t) - \omega_{ij}(t)]H(g^*) \tag{7}$$

From Equation 7, $\alpha(t)$ is the learning rate of the training algorithm, normally chosen to be a monotonically decreasing function of the training epoch t, and $H(g^*)$ is referred to as the neighborhood function, which decreases with distance from the BMU and models elastic interconnections between the neural nodes of the grid. Equation 8 illustrates a particular neighborhood function.

$$H(g^*) = H(i^*, j^*) = e^{[(i^*-i)^2+(j^*-j)^2]^{1/2}} \tag{8}$$

According to the theory of self-organizing maps given by Kohonen [82], the weight update rule given by Equation 7 produces a collection of weights representing the training data such that upon convergence $\omega_{ij}(t) \approx \omega_{ij}(t-1)$.

Figure 49 will be used to illustrate this weight training process. A single neuron is

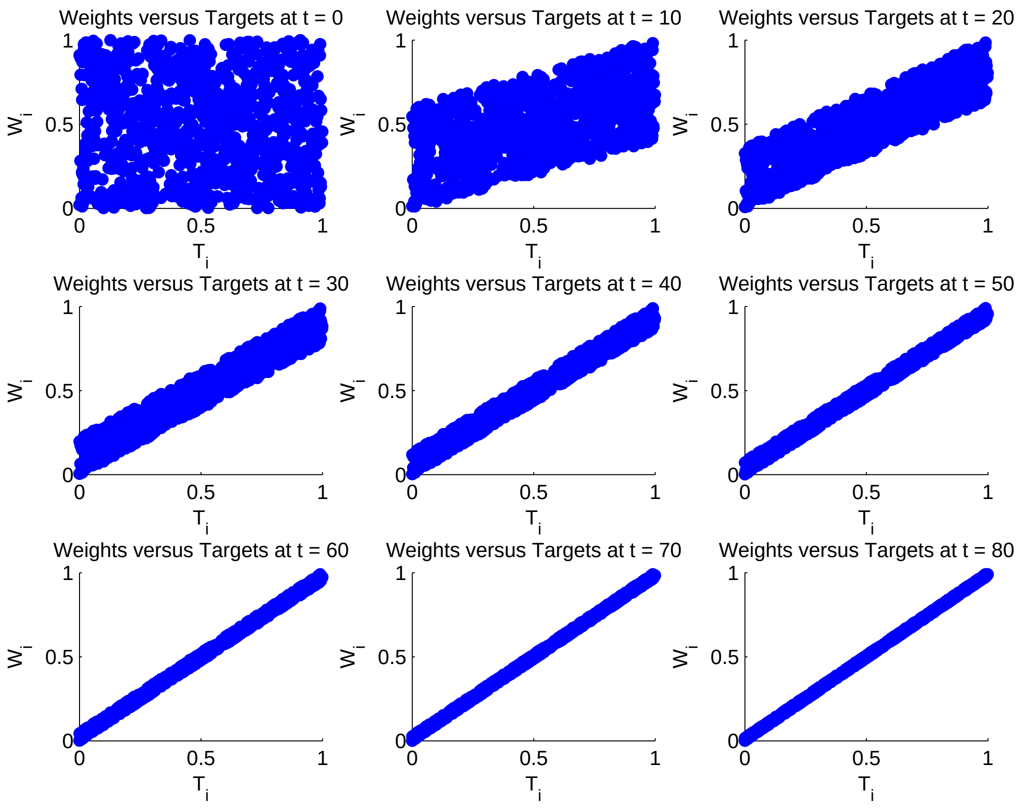


Figure 49: Learning weights.

simulated that is initiated with a weight vector consisting of a randomly generated set of 1000 values. A single target vector with 1000 elements is used as the training data. The algorithm employs a constant learning rate of $\alpha = 0.5$. Figure 49 plots the target vector versus the weight vector as the training epoch is increased from $t = 0$ to $t = 80$. The emergence of a straight line indicates how the weight vector successfully converges towards the target vector with increasing training epochs.

The HBSOM algorithm is a hybrid intelligent system that hybridizes naive Bayesian learning networks with self-organizing maps. HBSOM augments the theory of self-organizing maps. In particular, an attribute variable is assigned to each node of the SOM grid. During training, the class label of the training data is assigned to the attribute of the BMU. After training, the augmented SOM classifies input data by way of locating the BMU and responding with its attribute value. This augmented SOM implements the following classification query:

$$y_{som} = Q(x_1, x_2, \ldots, x_n) = SOM.BMU(x_1, x_2, \ldots, x_n).attribute$$

The NBLN of the HBSOM algorithm classifies data by augmenting the input data with the output of the augmented SOM. In particular, the HBSOM implements an NBLN that responds to queries as follows:

$$y_{nbln} = Q(x_1, x_2, \ldots, x_n, y_{som})$$

In short, the HBSOM algorithm uses an augmented SOM that classifies input data, and the SOM output is provided along with the input data to the NBLN that performs a final classification decision.

The HBSOM algorithm is trained as follows. First, the SOM learns from the associated training data representing the particular classification problem. Once the SOM has been trained, a modified collection of training data is created where the input space is augmented with the output of the trained SOM. The NBLN, in turn, learns from this modified collection of training data. Once the NBLN has been trained, classification of data proceeds as follows. The input data vector is submitted to the SOM. The SOM's output is then appended to the input vector and submitted to the NBLN. The NBLN receives the appended input vector, classifies it, and produces the final output for the system.

7.2.2 Simulation Results

The CUP99 dataset was used to evaluate the performance of the HBSOM algorithm. The simulations compared the performance of a non-hybrid NBLN to the performance of HBSOM. The classifiers were trained and tested with host-based and network-based features from the CUP99 dataset. A training data subset was selected randomly from the overall dataset. Likewise, a testing data subset was selected randomly for testing the associated algorithms. The classification results produced by the simulations are given by Table 25.

The columns of the table are labeled by $NBLN-HN, NBLN-N, HBSOM-HN$, and $HBSOM-N$, which represent the results of the NBLN using host and network based features (HN) along with just network based features (N). Similarly, the column labeled HBSOM-HN represent the results of the HBSOM algorithm using host and network based features while the column labeled HBSOM-N represents the results of the HBSOM algorithm using just network based features. The results shown in Table 25 reveal that both

Table 25: Classification results produced by simulations of the HBSOM and NBLN algorithms.

	NBLN-HN	NBLN-N	HBSOM-HN	HBSOM-N
Total Test Cases	65505	62047	65505	62407
Cases Correctly Classified	65019	59734	65238	61631
Cases Incorrectly Classified	486	2313	267	416
Percent Accuracy	99.26%	96.27%	99.59%	99.33%

approaches to classifying the types of traffic flows contained within the CUP99 dataset produce respectable accuracies. However, HBSOM had a noticeable gain in classification accuracy compared to the non-hybrid NBLN when only network-based features were used for training and testing.

7.3 Cyberattack Detection with Ensembles of Computational Intelligence Systems

The performance of an artificial neural network is sensitive to the selection of parameters that define its overall configuration. Some of these parameters include the transfer function used within each layer, the total number of layers, the total number of units per layer, the learning rate, the training algorithm, and the number of training epochs to use. Furthermore, these parameters are not generalized to any given network and depend on characteristics of the classification data. Appropriate selection of neural network parameters is typically a trial-and-error process whereby the designer seeks the set of parameters that minimize classification error produced by the network. Optimization algorithms that autonomically tune the parameters of artificial neural networks can alleviate the trial-and-error parameter selection process and can lead to neural networks with better classification accuracy.

In this section, a classification algorithm for attack detection based on ensembles of neural networks is proposed. The novelty of the algorithm stems from the methodology employed for combining outputs of neural network ensembles. Particularly, a neural network oracle is utilized to combine the ensemble outputs. The neural network oracle is constructed with a genetic algorithm that finds a set of configuration parameters that produce high classification accuracy and low classification error.

7.3.1 The NNO Classification Algorithm

The proposed algorithm referred to as NNO employs a genetic algorithm to find an optimal selection of configuration parameters for a neural network oracle, which is responsible for combining the outputs of an ensemble of neural networks that classify features belonging to audit data for the cyberattack detection problem. The overall idea is described as follows. A set of artificial neural networks $\eta = \{\eta_i\}$ is assigned to features of the collection of labeled audit data that describe a classification domain. The collection of ANNs are trained with standard procedures. Once the collection has been trained, the training data is used to generate a secondary set of training data. The secondary set of training data contains the output of each ANN along with the actual output defined by the baseline training data. The secondary training data is then used to train the neural network oracle. However, the oracle uses a genetic algorithm to find the set of configuration parameters that minimizes its error.

The algorithm consists of two primary phases. During phase 1, a GA is constructed that contains a population of chromosomes that are numerical representations of ANN configuration parameters. At each evolution time epoch, t, the chromosome for each population member is submitted to the ANN. The ANN maps the chromosome's numerical values to their respective parameter types, implements a self-configuration based on these values, and then learns from a training set. Once the ANN has been trained, a set of labeled validation data from the input-output space is used to evaluate the ANN's post-training error response. This error response is then used to evaluate the fitness of the population member whose chromosome was submitted to the ANN for configuration and training. Since the goal is to find an ANN with minimal error, the error response, which is given by Equation 9, is used as input to the fitness function of the genetic algorithm.

$$E_i = \sqrt{\sum_{j=1}^{N} (f(x_j) - f_a(x_j))^2} \tag{9}$$

In Equation 9, E_i is the error of the ANN configured and trained based on the chromosome h_i of the i^{th} population member. $f(x)$ is the value of the target function for input x, whereas $f_a(x)$ is the approximation of $f(x)$ produced by the ANN. The error is calculated over a total of N evaluations from a validation dataset. The fitness function for the system is given by Equation 10.

$$F(h_i; E_i) = \frac{1}{E_i} \tag{10}$$

The fitness function is inversely proportional to the error of the ANN configured by parameters represented by the i^{th} chromosome (i.e., the i^{th} hypothesis h_i) of the GA's population. With the fitness function of Equation 10, a decrease in error produced by an ANN configured via h_i produces an increase in fitness, which is the underlying objective of the algorithm.

The steps described above are performed for each member of the GA's population. Once each member in the population has been evaluated for fitness, the GA performs selection, crossover, and mutation operations and then proceeds to the next evolution epoch, t+1. This entire process proceeds until the evolution process terminates.

During phase 2, which proceeds after the simulated evolution process terminates, the GA submits the chromosome from the terminal population's best fit individual to the ANN. The ANN uses this chromosome to configure its parameters and then trains from a set of phase-2 training data. Once this training is complete, the system is ready to be deployed for its target application.

7.3.2 Simulation Results

The NNO ensemble methodology investigated for the cyberattack detection problem used a neural network oracle parametrically optimized with genetic algorithms as described above. Although several parameters can be considered for the optimal response of the NNO, the evaluations described in this section were based on an NNO configured with two hidden layers of nodes. The optimal parameters that were determined with the genetic algorithm included the number of neural units (nodes) to use for the first and second hidden layers, the type of transfer functions to use in the hidden layers, and the type of transfer function to use for the output layer.

The CUP99 dataset was used to evaluate the performance of the proposed system. An ensemble of 41 neural networks was created. Each feature of the CUP99 dataset was assigned to a single member of the ensemble. A subset of the CUP99 dataset was extracted for training. Each neural network was trained with respect to its corresponding feature. After the ensemble components were trained, a secondary training set was produced by collecting the output of each neural network in the ensemble over the phase-1 testing data and augmenting these outputs with the target class associated with each corresponding training vector. This secondary set of training data was then used to train the NNO. The NNO was parametrically optimized with a genetic algorithm as described above. Once the genetic algorithm converged to a best fit candidate representing the configuration parameters that minimized the NNO's error response with respect to the training data, the NNO was configured and trained via these parameters.

The simulation methodology is described as follows. The CUP99 dataset was partitioned into two disjoint sets comprising training data and testing data. The training procedure was described above. For evaluation, testing proceeded as follows. A total of 500 trials was performed. For each trial, a random selection of records were selected from the test data, and performance metrics of the algorithm were computed and stored as average values. Moreover, the performance of the proposed algorithm was compared to that of the weighted majority vote (WMV) algorithm.

Figure 50 plots the average accuracy over 500 trials for each of the 41 neural networks of the ensemble along with the accuracy of the NNO. The accuracy of the NNO is highlighted by the thick black line towards the top of the figure. Two things should be noted from Figure 50. First, the range of accuracies reveals that the system has diversity, which is a fundamental requirement for the design of ensemble systems. The accuracies indicate that

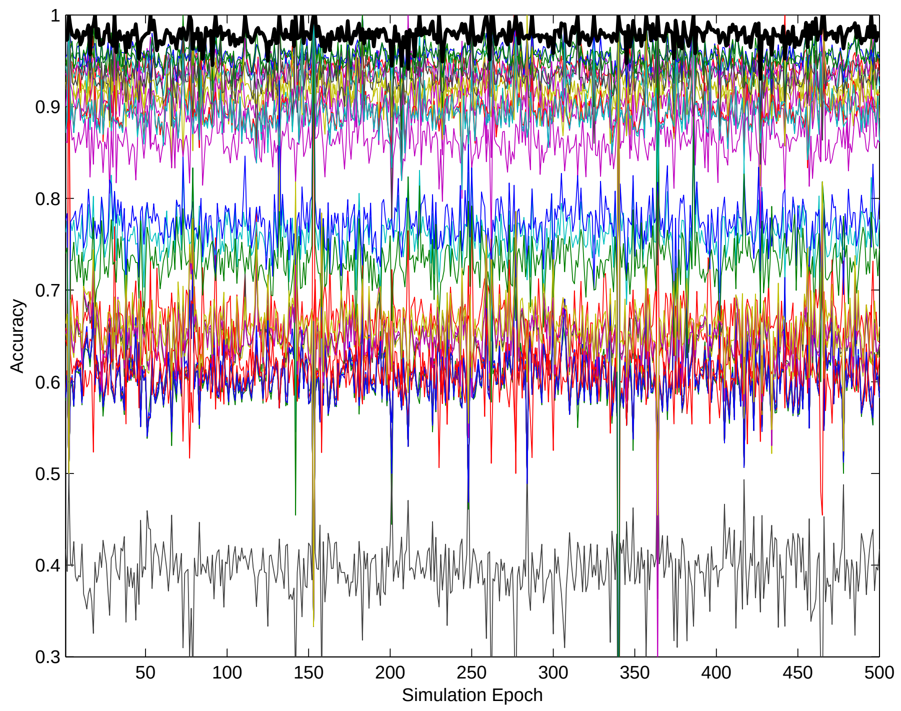


Figure 50: Accuracies of the individual ensemble members and the neural network oracle over the 500 trials.

some of the networks are poor classifiers, some are decent classifiers, and some are good classifiers. Second, the NNO performs consistently better than any given member of the ensemble.

Figure 51 provides a bar chart showing the average of the accuracies produced over the 500 trials for each neural network of the ensemble along with the NNO and WMV. As seen from the figure, the NNO outperforms each of the ensemble members as well as the WMV algorithm.

The frequency distributions of the various metrics calculated over the 500 trials for the ensemble components, the NNO method, and the WMV method were generated in order to provide a finer-grained perspective of the performance of each system. A bin width of 0.01 was used to calculate the frequency distributions. Each of the following figures provide metrics for ensemble members, NNO, and WMV. The ensemble results are provide for completeness and for illustrating diversity of the system. However, comparing the performance of NNO versus WMV is the main objective.

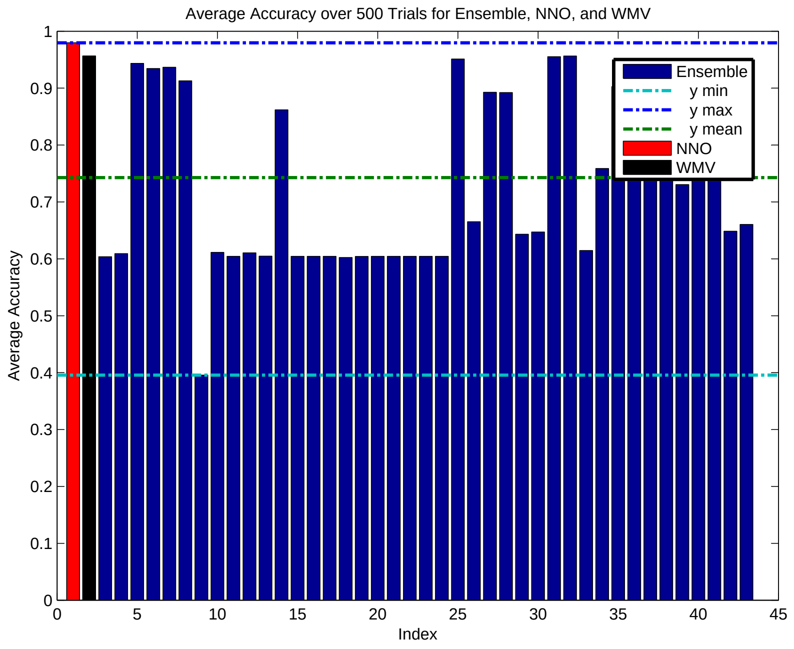


Figure 51: Bar chart revealing the average accuracies produced over the 500 trials.

Figure 52 plots the frequency distributions for classification accuracies. As seen by the lower right plot, NNO has better accuracy than WMV. Figure 53 plots the frequency distributions for positive prediction rates (PPR) for each classification algorithm. PPR measures how well a classification algorithm predicts the positive class correctly. As seen in the figure, NNO has a better PPR than WMV.

Figure 54 plots the negative prediction rate (NPR) of the algorithms. NPR measures how well a classification algorithm correctly classifies negative instances. For NPR, NNO and WMV both perform well and similarly. Figure 55 provides the false discovery rates (FDR) produced by the simulations for the classification algorithms. FDR should be small for good classification algorithms. As seen in the figure, NNO has better FDR performance than WMV.

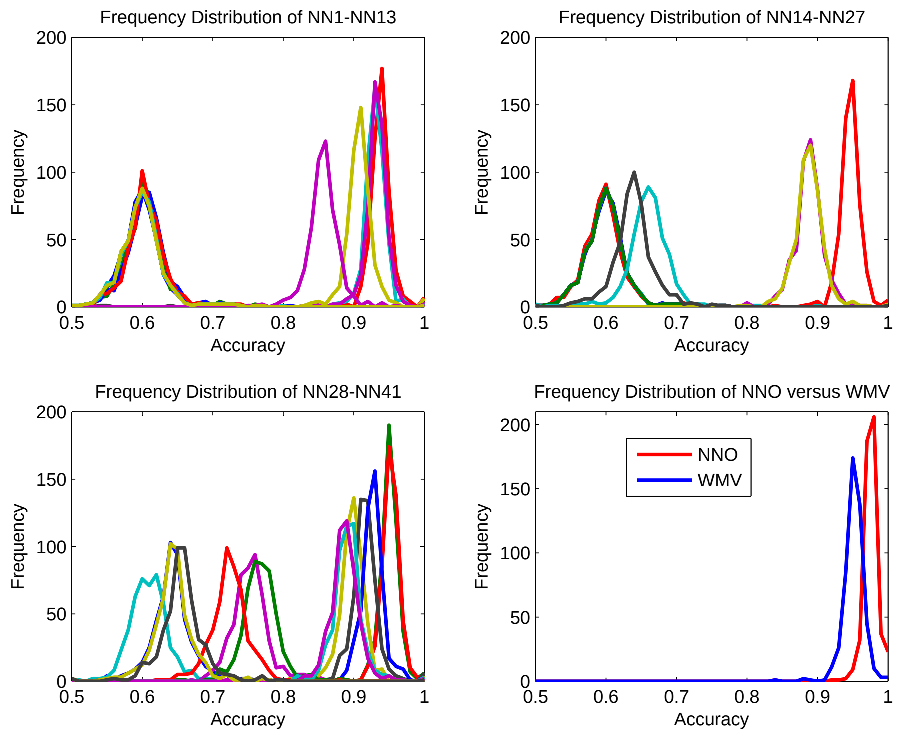


Figure 52: Frequency distribution of the accuracies for each system over 500 simulation trials.

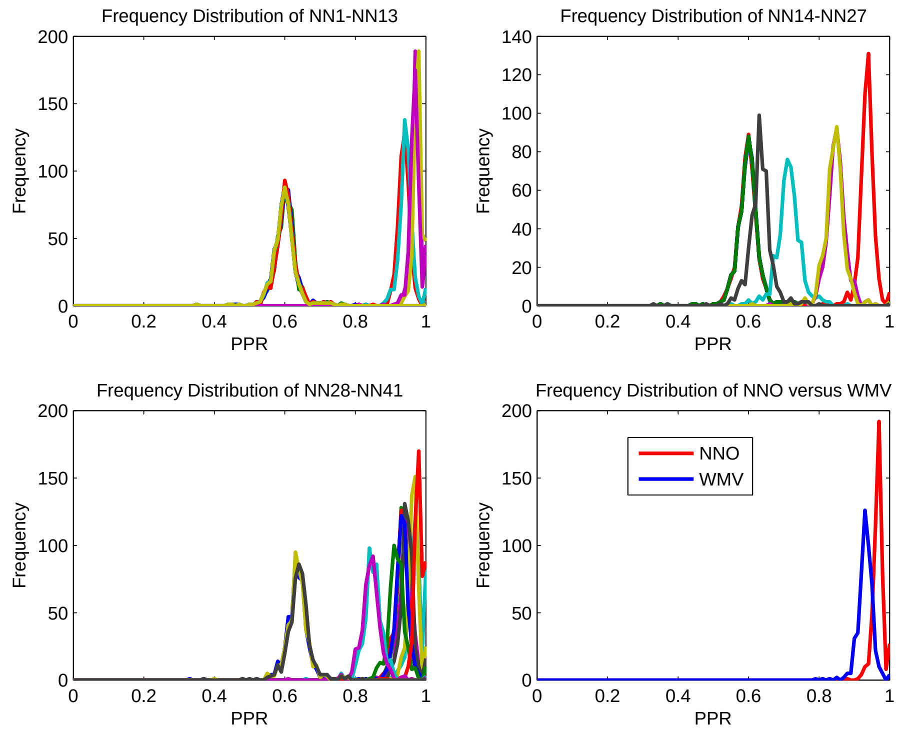


Figure 53: Frequency distribution of the PPR for each system over 500 simulation trials.

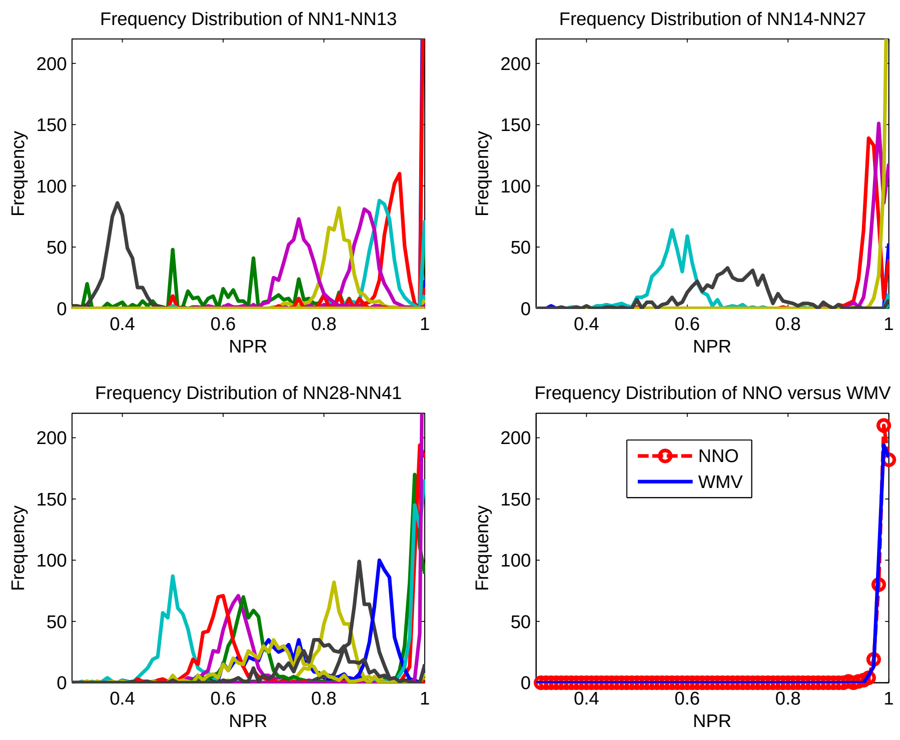


Figure 54: Frequency distribution of the NPR for each system over 500 simulation trials.

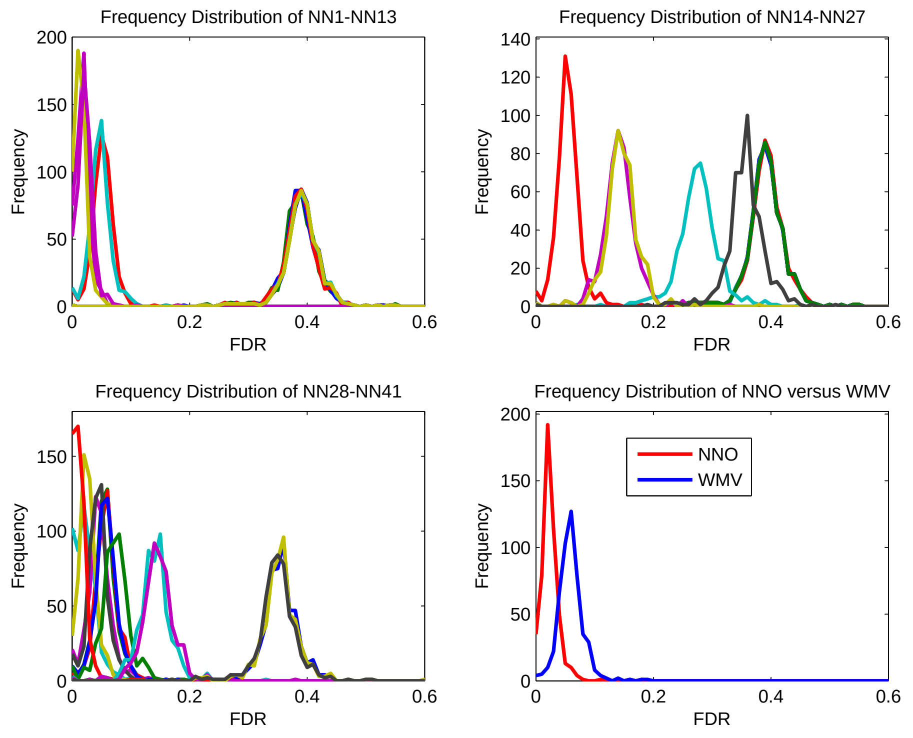


Figure 55: Frequency distribution of the FDR for each system over 500 simulation trials.

Figures 56 and 57 plot the frequency distributions for the false positive rates (FPR) and false negative rates (FNR), respectively. Similar to FDR, a good classification algorithm should have small FPR and FNR. As seen in Figure 56, NNO has better FPR performance than WMV. However, NNO and WMV perform comparably for FNR.

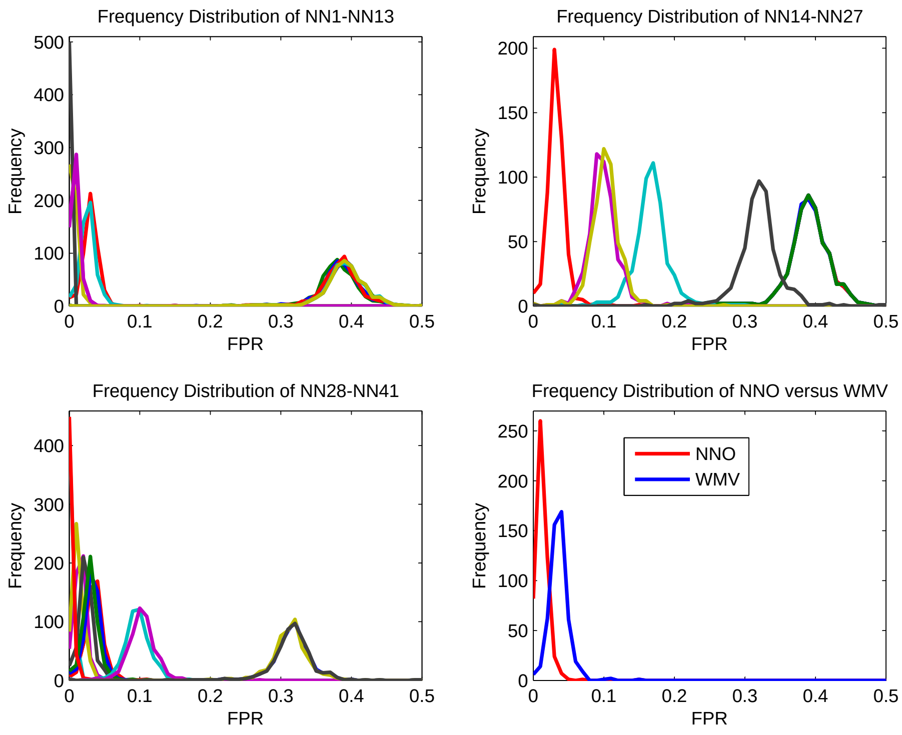


Figure 56: Frequency distribution of the FPR for each system over 500 simulation trials.

Based on the results obtained from simulations along with analysis of its performance metrics, the proposed ensemble methodology using a parametrically optimized neural network oracle provides good performance as a classification system for the cyberattack detection problem.

7.4 Summary

In this chapter, two classification algorithms for the cyberattack detection problem were introduced. The HBSOM algorithm is based on the notion of hybrid intelligent systems. HBSOM utilizes a modified version of self-organizing maps in conjunction with naive Bayesian learning algorithms. The second algorithm employs an ensemble of neural networks along

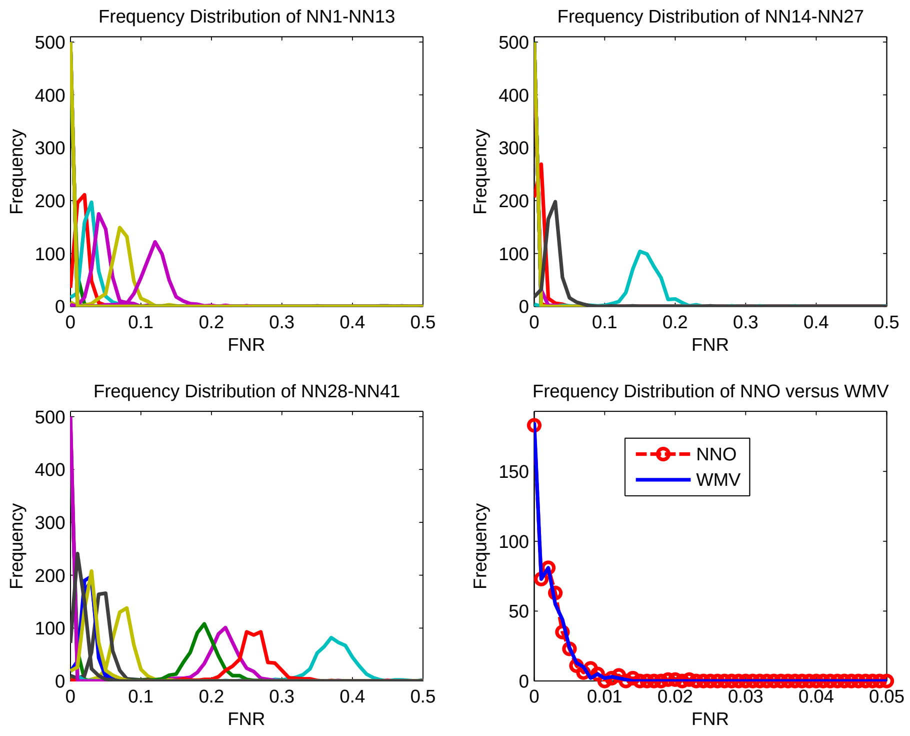


Figure 57: Frequency distribution of the FNR for each system over 500 simulation trials.

with a neural network oracle with its configuration parameters optimized by genetic algorithms using a fitness function evaluated with neural network error responses. The performance evaluation of the proposed algorithms were based on the CUP99 intrusion detection dataset. According to the simulation results obtained, both algorithms were shown to provide good classification performance when trained and tested with the CUP99 intrusion detection dataset.

CHAPTER VIII

CONCLUSION

Current cybersecurity architectures and infrastructures continue to be dominated by quasi-monolithic and perimeter-based security mechanisms that aim to provide security services for elements comprising an organization's overall cyber environment. For example, perimeter firewalls and intrusion detection systems provide network access control and detection services for resources contained within the network boundaries of an organization. However, perimeter-based, isolated, non-cooperative, and administratively reactive techniques are becoming ineffective, and this is especially true for emergent platforms such as the mobile Internet and cloud computing in which the constituent computational and networking resources reside outside of enterprise topologies and perimeters. Although individual hosts within the organization typically employ host-based firewalls and other services such as virus scanning software, these systems very seldom operate as collaborative and coordinated mechanisms. A modern framework is needed that guides the design of highly modular, adaptable, and integrable security architectures where ***any*** cyber device within a security infrastructure is capable of detecting and responding to attacks. To address this need, the distributed firewall and active response architecture was introduced in this dissertation. The architecture is based on trusted domains of administration whereby cyber entities implement the trusted domain of administration host architecture. The dissertation also introduced a new firewall-based blacklist classification and enforcement model the describes how blacklists should be implemented by firewall systems.

Research by the packet classification community over recent years suggests that hardware-based packet classifiers are much desired because software-based packet classification solutions cannot satisfy the stringent constraints of modern-day IP networks, especially within the core of the Internet's routing infrastructure. Others suggest that future packet classification systems will likely be hybrids comprised of software components coupled with hardware acceleration components. In an effort to address the fundamental packet classification problem, the theory of semantic association systems was developed from which the semantic path merger (SPM) algorithm was derived. The semantic path merger algorithm and its hardware implementation are core contributions of this dissertation that significantly advance the state-of-art in packet classification algorithms. Simulation results of the hardware implementation revealed significant performance improvements over the *de facto* content addressable memory.

The modern Internet security landscape is characterized by attacks that are voluminous, constantly evolving, extremely fast, persistent, and highly sophisticated. Consequently, security researchers must continuously strive to find new classification algorithms that are capable of detecting cyberattacks. In an effort to address this challenge, the HBSOM and NNO algorithms were introduced as new classification algorithms for cyberattack detection systems. HBSOM hybridizes Bayesian learning networks with self-organizing maps. NNO uses a genetic algorithm to parametrically optimize a neural network oracle that combines the output results generated by an ensemble of neural networks. Simulation results showed that these algorithms perform robustly when classifying cyberattacks.

In conclusion, this dissertation provided novel algorithms, theories, and supporting frameworks to significantly improve the growing problem of Internet security. The objectives of this dissertation research were (1) to study the characteristics needed for reliable distributed Internet security architectures and infrastructures, (2) to design a modular, adaptive, and integrable framework for distributed Internet security infrastructures, (3) to develop robust solutions addressing the fundamental packet classification problem, and (4) to develop new classification algorithms for advanced cyberattack detection systems.

As a result of these objectives, the contributions provided by this dissertation are the following:

- The distributed firewall and active response architecture.
- A firewall-based blacklist classification and enforcement model.
- The theory of semantic association systems.

- The semantic path merger packet classification algorithm and its hardware implementation.
- Advanced cyberattack detection algorithms based on computational intelligence systems.

The distributed firewall and active response architecture is a modular, adaptive, and integrable distributed security framework that enables cyber devices within an organization's cyber infrastructure to participate in the detection of and response to cyberattacks. The architecture is a supporting contribution that establishes the foundation upon which the core contributions of this dissertation are framed. Particularly, the distributed firewall and active response architecture requires (1) efficient packet classification algorithms to enable its novel firewall-based blacklist classification and enforcement model and (2) effective classification algorithms for cyberattack detection. While studying packet classification and cyberattack detection algorithms, the theory of semantic association systems was devised and formulated. The theory of semantic association systems was inspired by the emerging field of semantic computing. The theory defines a compositional model and a family of graph theoretic constructs supporting the notion of merged conceptualization. From this theory and the notion of merged conceptualization, the semantic path merger packet classification algorithm was derived along with its hardware-based implementation. The theory of semantic association systems, the semantic path merger algorithm, and the hardware implementation of the semantic path merger algorithm as a packet classification system are the core contributions of this dissertation. Finally, two novel cyberattack detection algorithms have been developed. The first detection algorithm is a hybridization of self-organizing maps and naive Bayesian learning networks. The second detection algorithm is a neural network ensemble coupled with a parametrically optimized neural network oracle that combines ensemble classification results. The oracle's network configuration is parametrically optimized with genetic algorithms.

APPENDIX A

CACTI 6.5 CONFIGURATION EXAMPLE

The following provides an example of a configuration file for the Cacti 6.5 cache/memory simulator.

```
###########  CACTI 6.5 Configuration File ##########################
# A basic Cacti 6.5 Configuration File Example.
#
# NOTES:
# 1. Lines beginning with # or // are comments.
#
# 2. Parameters and their arguments are used based on the
# type of memory being simulated. Various memory structures
# will not utilize all of these parameters.
#
# 3. Reference technical reports for Cacti versions 1.0, 2.0, 3.2,
# 4.1, 5.3, and 6.5 for specific details of the simulator as
# well as detailed explanations of the configuration parameters
# below. Technical reports available from HP Research Labs located
# at http://www.hpl.hp.com/research/cacti/.
####################################################################

# Cache size
//-size (bytes) 4096
-size (bytes) 524288
//-size (bytes) 1048576
//-size (bytes) 2097152
//-size (bytes) 4194304
//-size (bytes) 8388608
//-size (bytes) 16777216
//-size (bytes) 33554432
//-size (bytes) 134217728
//-size (bytes) 67108864
//-size (bytes) 1073741824

# Line size
//-block size (bytes) 8
//-block size (bytes) 128
-block size (bytes) 32768

# To model Fully Associative cache, set associativity to zero
#
# Fully Associative
//-associativity 0
# Direct Mapped
-associativity 1
# m-way set associative: m := 2|4|8|16
//-associativity 2
//-associativity 4
//-associativity 8
//-associativity 16

-read-write port 1
-exclusive read port 0
-exclusive write port 0
-single ended read ports 0

# Multiple banks connected using a bus
-UCA bank count 1
```

```
# Technology Node
//-technology (u) 0.032
//-technology (u) 0.045
//-technology (u) 0.068
-technology (u) 0.090

# Main memory parameters
-page size (bits) 8192
-burst length 8
-internal prefetch width 8

# The following parameter can have one of five values:
# (itrs-hp, itrs-lstp, itrs-lop, lp-dram, comm-dram)
//-Data array cell type - "comm-dram"
//-Data array cell type - "itrs-hp"
//-Data array cell type - "itrs-lstp"
-Data array cell type - "itrs-lop"

# The following parameter can have one of three values:
# (itrs-hp, itrs-lstp, itrs-lop)
//-Data array peripheral type - "itrs-hp"
//-Data array peripheral type - "itrs-lstp"
-Data array peripheral type - "itrs-lop"

# The following parameter can have one of five values:
# (itrs-hp, itrs-lstp, itrs-lop, lp-dram, comm-dram)
-Tag array cell type - "itrs-lop"
//-Tag array cell type - "itrs-lstp"

# The following parameter can have one of three values:
# (itrs-hp, itrs-lstp, itrs-lop)
-Tag array peripheral type - "itrs-lop"
//-Tag array peripheral type - "itrs-lstp"

# Bus width include data bits and address bits required by the decoder
-output/input bus width 256
//-output/input bus width 256

// Temperature range 300-400 in steps of 10
-operating temperature (K) 350

# Type of memory - cache (with a tag array),
# or ram (scratch ram similar to a register file),
# or cam,
# or main memory (no tag array and every access will happen
# at a page granularity Ref: CACTI 5.3 report)
#
//-cache type "cache"
-cache type "ram"
//-cache type "main memory"

# To model special structure like branch target buffers, directory, etc.
# change the tag size parameter
# if you want cacti to calculate the tagbits, set the tag size to "default"
-tag size (b) "default"
//-tag size (b) 45

# fast - data and tag access happen in parallel
# sequential - data array is accessed after accessing the tag array
# normal - data array lookup and tag access happen in parallel
#          final data block is broadcasted in data array h-tree
#          after getting the signal from the tag array
//-access mode (normal, sequential, fast) - "fast"
-access mode (normal, sequential, fast) - "normal"
//-access mode (normal, sequential, fast) - "sequential"

# DESIGN OBJECTIVE for UCA (Uniform Cache Access)
# or banks in NUCA (Non Uniform Cache Access
-design objective (weight delay, dynamic power, leakage power, cycle time, area) 0:100:0:0:0

# Percentage deviation from the minimum value
# Ex: A deviation value of 10:1000:1000:1000:1000 will try to find an organization
# that compromises at most 10% delay.
# NOTE: Try reasonable values for % deviation. Inconsistent deviation
# percentage values will not produce any valid organizations. For example,
# 0:0:100:100:100 will try to identify an organization that has both
# least delay and dynamic power. Since such an organization is not possible, CACTI will
# throw an error. Refer CACTI-6 Technical report for more details
-deviate (delay, dynamic power, leakage power, cycle time, area) 60:100000:100000:100000:1000000

# Objective for NUCA
-NUCAdesign objective (weight delay, dynamic power, leakage power, cycle time, area) 100:100:0:0:100
-NUCAdeviate (delay, dynamic power, leakage power, cycle time, area) 10:10000:10000:10000:10000

# Set optimize tag to ED or ED^2 to obtain a cache configuration optimized for
# energy-delay or energy-delay sq. product
# Note: Optimize tag will disable weight or deviate values mentioned above
# Set it to NONE to let weight and deviate values determine the
# appropriate cache configuration
//-Optimize ED or ED^2 (ED, ED^2, NONE): "ED"
//-Optimize ED or ED^2 (ED, ED^2, NONE): "ED^2"
-Optimize ED or ED^2 (ED, ED^2, NONE): "ED"

-Cache model (NUCA, UCA)  - "UCA"
//-Cache model (NUCA, UCA)  - "NUCA"

# In order for CACTI to find the optimal NUCA bank value the following
# variable should be assigned 0.
-NUCA bank count 0

# NOTE: for nuca network frequency is set to a default value of
# 5GHz in time.c. CACTI automatically
# calculates the maximum possible frequency and downgrades this value if necessary

# By default CACTI considers both full-swing and low-swing
# wires to find an optimal configuration. However, it is possible to
# restrict the search space by changing the signalling from "default" to
# "fullswing" or "lowswing" type.
//-Wire signalling (fullswing, lowswing, default) - "Global_10"
-Wire signalling (fullswing, lowswing, default) - "default"
//-Wire signalling (fullswing, lowswing, default) - "lowswing"

-Wire inside mat - "global"
//-Wire inside mat - "semi-global"
-Wire outside mat - "global"

-Interconnect projection - "conservative"
//-Interconnect projection - "aggressive"

# Contention in network (which is a function of core count and cache level) is one of
# the critical factor used for deciding the optimal bank count value
# core count can be 4, 8, or 16
//-Core count 4
-Core count 8
//-Core count 16
-Cache level (L2/L3) - "L3"

-Add ECC - "false"

//-Print level (DETAILED, CONCISE) - "CONCISE"
-Print level (DETAILED, CONCISE) - "DETAILED"

# for debugging
-Print input parameters - "true"
//-Print input parameters - "false"
```

```
# force CACTI to model the cache with the
# following Ndbl, Ndwl, Nspd, Ndsam,
# and Ndcm values
//-Force cache config - "true"
-Force cache config - "false"
-Ndwl 64
-Ndbl 64
-Nspd 64
-Ndcm 1
-Ndsam1 4
-Ndsam2 1
```

APPENDIX B

CACTI 6.5 SIMULATION OUTPUT

The following data results from running Cacti 6.5 using the configuration file presented in Appendix A.

```
####
# Cacti 6.5 Output for RAM of size 512KB with a Block size (length of the bitline)
# of 32768 Bytes = 256K bits.

Cache size                    : 524288
Block size                    : 32768
Associativity                 : 1
Read only ports               : 0
Write only ports              : 0
Read write ports              : 1
Single ended read ports       : 0
Cache banks (UCA)             : 1
Technology                    : 0.09
Temperature                   : 350
Tag size                      : 42
cache type                    : Scratch RAM
Model as memory               : 0
Access mode                   : 0
Data array cell type          : 2
Data array peripheral type    : 2
Tag array cell type           : 2
Tag array peripheral type     : 2
Design objective (UCA wt)     : 0 100 0 0 0
Design objective (UCA dev)    : 60 100000 100000 100000 1000000
Design objective (NUCA wt)    : 100 100 0 0 100
Design objective (NUCA dev)   : 10 10000 10000 10000 10000
Cache model                   : 0
Nuca bank                     : 0
Wire inside mat               : 2
Wire outside mat              : 2
Interconnect projection       : 1
Wire signalling               : 0
Cores                         : 8
Print level                   : 1
ECC overhead                  : 0
Page size                     : 8192
Burst length                  : 8
Internal prefetch width       : 8
Force cache config            : 0

---------- CACTI version 6.5, Uniform Cache Access SRAM Model ----------

Cache Parameters:
    Total cache size (bytes): 524288
    Number of banks: 1
    Associativity: direct mapped
    Block size (bytes): 32768
    Read/write Ports: 1
    Read ports: 0
    Write ports: 0
    Technology size (nm): 90

    Access time (ns): 5.51156
    Cycle time (ns):  9.97772
    Total dynamic read energy per access (nJ): 0.427615
    Total leakage power of a bank (mW): 161.701
    Cache height x width (mm): 1.82573 x 4.28617

    Best Ndwl : 2
    Best Ndbl : 4
    Best Nspd : 0.015625
    Best Ndcm : 32
    Best Ndsam L1 : 1
    Best Ndsam L2 : 1

    Data array, H-tree wire type: Delay optimized global wires

Time Components:

  Data side (with Output driver) (ns): 5.51156
        H-tree input delay (ns): 0.143935
        Decoder + wordline delay (ns): 2.51244
        Bitline delay (ns): 1.55724
        Sense Amplifier delay (ns): 0.0149121
        H-tree output delay (ns): 1.28302


Power Components:

  Data array: Total dynamic read energy/access  (nJ): 0.427615
        Total leakage read/write power of a bank (mW): 161.701
        Total energy in H-tree (that includes both address and data transfer) (nJ): 0.147739
        Output Htree Energy (nJ): 0.13871
        Decoder (nJ): 0.000607173
        Wordline (nJ): 0.00400229
        Bitline mux & associated drivers (nJ): 3.78242e-05
        Sense amp mux & associated drivers (nJ): 0
        Bitlines (nJ): 0.157832
        Sense amplifier energy (nJ): 0.000997449
        Sub-array output driver (nJ): 0.0756909


Area Components:

  Data array: Area (mm2): 7.8254
        Height (mm): 1.82573
        Width (mm): 4.28617
        Area efficiency (Memory cell area/Total area) - 63.3857 %
                MAT Height (mm): 0.907826
                MAT Length (mm): 3.90673
                Subarray Height (mm): 0.336384
                Subarray Length (mm): 1.9296

Wire Properties:

  Delay Optimal
        Repeater size - 363.803
        Repeater spacing - 2.10262 (mm)
        Delay - 0.0562903 (ns/mm)
        PowerD - 0.000625057 (nJ/mm)
        PowerL - 0.00307724 (mW/mm)
        Wire width - 0.36 microns
        Wire spacing - 0.36 microns

  5% Overhead
        Repeater size - 220.803
        Repeater spacing - 2.70262 (mm)
        Delay - 0.0590965 (ns/mm)
        PowerD - 0.000431745 (nJ/mm)
        PowerL - 0.00145303 (mW/mm)
        Wire width - 0.36 microns
        Wire spacing - 0.36 microns
```

```
10% Overhead
      Repeater size - 195.803
      Repeater spacing - 3.10262 (mm)
      Delay - 0.0619085 (ns/mm)
      PowerD - 0.000399725 (nJ/mm)
      PowerL - 0.0011224 (mW/mm)
      Wire width - 0.36 microns
      Wire spacing - 0.36 microns

20% Overhead
      Repeater size - 154.803
      Repeater spacing - 3.40262 (mm)
      Delay - 0.0675259 (ns/mm)
      PowerD - 0.000359491 (nJ/mm)
      PowerL - 0.000809136 (mW/mm)
      Wire width - 0.36 microns
      Wire spacing - 0.36 microns

30% Overhead
      Repeater size - 133.803
      Repeater spacing - 3.80262 (mm)
      Delay - 0.0730296 (ns/mm)
      PowerD - 0.000339637 (nJ/mm)
      PowerL - 0.000625804 (mW/mm)
      Wire width - 0.36 microns
      Wire spacing - 0.36 microns

Low-swing wire (1 mm) - Note: Unlike repeated wires,
      delay and power values of low-swing wires do not
      have a linear relationship with length.
      delay - 1.9649 (ns)
      powerD - 8.30606e-05 (nJ)
      PowerL - 1.39416e-06 (mW)
      Wire width - 7.2e-07 microns
      Wire spacing - 7.2e-07 microns
```

APPENDIX C

CACTI 6.5 CONFIGURATION EXAMPLE WITH MCPAT

The following provides an example of a configuration file for the Cacti 6.5 cache/memory simulator used with McPat.

```
###########  CACTI 6.5 Configuration File #########################
# A basic Cacti 6.5 Configuration File Example used with McPat
#
# NOTES:
# 1. Lines beginning with # or // are comments.
#
# 2. Parameters and their arguments are used based on the
# type of memory being simulated. Various memory structures
# will not utilize all of these parameters.
#
# 3. Reference technical reports for Cacti versions 1.0, 2.0, 3.2,
# 4.1, 5.3, and 6.5 for specific details of the simulator as
# well as detailed explanations of the configuration parameters
# below. Technical reports available from HP Research Labs located
# at http://www.hpl.hp.com/research/cacti/.
####################################################################

# Cache size
//-size (bytes) 528
//-size (bytes) 4096
//-size (bytes) 262144
//-size (bytes) 1048576
//-size (bytes) 2097152
-size (bytes) 4194304
//-size (bytes) 8388608
//-size (bytes) 16777216
//-size (bytes) 33554432
//-size (bytes) 134217728
//-size (bytes) 67108864
//-size (bytes) 1073741824

# Line size
//-block size (bytes) 8
//-block size (bytes) 32
-block size (bytes) 32768

############# McPat Specific  ###########################
#
# McPat adds functionality to the Cacti 6.5 Codebase that
# allows one to model a pure binary CAM. The following
# parameter specifies 'w' in terms of packet classification,
# but in general it is the width of the word pattern stored
# in the CAM.
#
#########################################################
//cam search width
//-search port 32
//-search port 64
-search port 128

# To model Fully Associative cache, set associativity to zero
#
```

```
# Fully Associative
//-associativity 0
# Direct Mapped
-associativity 1
# m-way set associative: m := 2|4|8|16
//-associativity 2
//-associativity 4
//-associativity 8
//-associativity 16

-read-write port 1
-exclusive read port 0
-exclusive write port 0
-single ended read ports 0

# Multiple banks connected using a bus
-UCA bank count 1

# Technology Node
//-technology (u) 0.032
//-technology (u) 0.045
//-technology (u) 0.068
-technology (u) 0.090

# Main memory parameters
-page size (bits) 8192
-burst length 8
-internal prefetch width 8

# The following parameter can have one of five values:
# (itrs-hp, itrs-lstp, itrs-lop, lp-dram, comm-dram)
//-Data array cell type - "comm-dram"
//-Data array cell type - "itrs-hp"
//-Data array cell type - "itrs-lstp"
-Data array cell type - "itrs-lop"

# The following parameter can have one of three values:
# (itrs-hp, itrs-lstp, itrs-lop)
//-Data array peripheral type - "itrs-hp"
//-Data array peripheral type - "itrs-lstp"
-Data array peripheral type - "itrs-lop"

# The following parameter can have one of five values:
# (itrs-hp, itrs-lstp, itrs-lop, lp-dram, comm-dram)
-Tag array cell type - "itrs-lop"
//-Tag array cell type - "itrs-lstp"

# The following parameter can have one of three values:
# (itrs-hp, itrs-lstp, itrs-lop)
-Tag array peripheral type - "itrs-lop"
//-Tag array peripheral type - "itrs-lstp"

# Bus width include data bits and address bits required by the decoder
-output/input bus width 256
//-output/input bus width 256

// Temperature range 300-400 in steps of 10
-operating temperature (K) 350

############# McPat Specific  ############################
#
# McPat adds functionality to the Cacti 6.5 Codebase that
# allows one to model a pure binary CAM. The following
# parameter specifies the cache type to be cam. Note that
# some of the parameter terminology used in the Cacti
# configuration is based on 'legacy' ideas, i.e., Cacti
# originally started as a basic cache model but has since
# evolved to model different types of SRAM/DRAM memory
# structures. For example, the cam specification below
# does not imply the CAM within a fully associative
# cache. Instead, it means that the structure to be simulated
# will be a pure cam.
#
##########################################################
#
# Type of memory - cache (with a tag array),
# or ram (scratch ram similar to a register file),
# or cam,
# or main memory (no tag array and every access will happen
# at a page granularity Ref: CACTI 5.3 report)
#
//-cache type "cache"
//-cache type "ram"
//-cache type "main memory"
-cache type "cam"

# To model special structure like branch target buffers, directory, etc.
# change the tag size parameter
# if you want cacti to calculate the tagbits, set the tag size to "default"
-tag size (b) "default"
//-tag size (b) 45

# fast - data and tag access happen in parallel
# sequential - data array is accessed after accessing the tag array
# normal - data array lookup and tag access happen in parallel
#          final data block is broadcasted in data array h-tree
#          after getting the signal from the tag array
//-access mode (normal, sequential, fast) - "fast"
-access mode (normal, sequential, fast) - "normal"
//-access mode (normal, sequential, fast) - "sequential"

# DESIGN OBJECTIVE for UCA (Uniform Cache Access)
# or banks in NUCA (Non Uniform Cache Access
-design objective (weight delay, dynamic power, leakage power, cycle time, area) 0:100:0:0:0

# Percentage deviation from the minimum value
# Ex: A deviation value of 10:1000:1000:1000:1000 will try to find an organization
# that compromises at most 10% delay.
# NOTE: Try reasonable values for % deviation. Inconsistent deviation
# percentage values will not produce any valid organizations. For example,
# 0:0:100:100:100 will try to identify an organization that has both
# least delay and dynamic power. Since such an organization is not possible, CACTI will
# throw an error. Refer CACTI-6 Technical report for more details
-deviate (delay, dynamic power, leakage power, cycle time, area) 60:100000:100000:100000:1000000

# Objective for NUCA
-NUCAdesign objective (weight delay, dynamic power, leakage power, cycle time, area) 100:100:0:0:100
-NUCAdeviate (delay, dynamic power, leakage power, cycle time, area) 10:10000:10000:10000:10000

# Set optimize tag to ED or ED^2 to obtain a cache configuration optimized for
# energy-delay or energy-delay sq. product
# Note: Optimize tag will disable weight or deviate values mentioned above
# Set it to NONE to let weight and deviate values determine the
# appropriate cache configuration
//-Optimize ED or ED^2 (ED, ED^2, NONE): "ED"
//-Optimize ED or ED^2 (ED, ED^2, NONE): "ED^2"
-Optimize ED or ED^2 (ED, ED^2, NONE): "ED"

-Cache model (NUCA, UCA)  - "UCA"
//-Cache model (NUCA, UCA)  - "NUCA"

# In order for CACTI to find the optimal NUCA bank value the following
# variable should be assigned 0.
-NUCA bank count 0

# NOTE: for nuca network frequency is set to a default value of
# 5GHz in time.c. CACTI automatically
# calculates the maximum possible frequency and downgrades this value if necessary

# By default CACTI considers both full-swing and low-swing
# wires to find an optimal configuration. However, it is possible to
# restrict the search space by changing the signalling from "default" to
# "fullswing" or "lowswing" type.
//-Wire signalling (fullswing, lowswing, default) - "Global_10"
-Wire signalling (fullswing, lowswing, default) - "default"
//-Wire signalling (fullswing, lowswing, default) - "lowswing"

-Wire inside mat - "global"
//-Wire inside mat - "semi-global"
-Wire outside mat - "global"

-Interconnect projection - "conservative"
//-Interconnect projection - "aggressive"

# Contention in network (which is a function of core count and cache level) is one of
# the critical factor used for deciding the optimal bank count value
# core count can be 4, 8, or 16
//-Core count 4
-Core count 8
//-Core count 16
-Cache level (L2/L3) - "L3"

-Add ECC - "false"

//-Print level (DETAILED, CONCISE) - "CONCISE"
-Print level (DETAILED, CONCISE) - "DETAILED"

# for debugging
-Print input parameters - "true"
//-Print input parameters - "false"
# force CACTI to model the cache with the
# following Ndbl, Ndwl, Nspd, Ndsam,
# and Ndcm values
//-Force cache config - "true"
-Force cache config - "false"
-Ndwl 64
-Ndbl 64
-Nspd 64
-Ndcm 1
-Ndsam1 4
-Ndsam2 1
```

APPENDIX D

CACTI 6.5 SIMULATION OUTPUT WITH MCPAT

The following data results from running Cacti 6.5 using the configuration file presented in Appendix C.

```
###############################################################
Cacti 6.5 Output for CAM Simulation: Parameters (N=256K, w=128)

Cache size                    : 4194304
Block size                    : 32768
Associativity                 : 0
Read only ports               : 0
Write only ports              : 0
Read write ports              : 1
Single ended read ports       : 0
Search ports                  : 128
Cache banks (UCA)             : 1
Technology                    : 0.09
Temperature                   : 350
Tag size                      : 42
array type                    : CAM
Model as memory               : 0
Access mode                   : 0
Data array cell type          : 2
Data array peripheral type    : 2
Tag array cell type           : 2
Tag array peripheral type     : 2
Design objective (UCA wt)     : 0 0 0 100 0
Design objective (UCA dev)    : 20 100000 100000 100000 1000000
Cache model                   : 0
Nuca bank                     : 0
Wire inside mat               : 2
Wire outside mat              : 2
Interconnect projection       : 1
Wire signalling               : 0
Print level                   : 1
ECC overhead                  : 0
Page size                     : 8192
Burst length                  : 8
Internal prefetch width       : 8
Force cache config            : 0

---------- CACTI version 6.5, Uniform Cache Access SRAM Model ----------

Cache Parameters:
    Total cache size (bytes): 4194304
    Number of banks: 1
    Associativity: fully associative
    Block size (bytes): 32768
    Read/write Ports: 1
    Read ports: 0
    Write ports: 0
    search ports: 128
    Technology size (nm): 90

    Access time (ns): 453.324
    Cycle time (ns):  225.248
    Total dynamic associative search energy per access (nJ): 1121.21
    Total dynamic read energy per access (nJ): 46.2443
    Total dynamic write energy per access (nJ): 47.9503
    Total leakage power of a bank (mW): 760105
    Total gate leakage power of a bank (mW): 396225
    Cache height x width (mm): 2146.2 x 125.571

    Best Ndwl : 1
    Best Ndbl : 512
    Best Nspd : 1
    Best Ndcm : 1
    Best Ndsam L1 : 1
    Best Ndsam L2 : 1

    Data array, H-tree wire type: Delay optimized global wires

Time Components:

  Data side (with Output driver) (ns): 453.324
H-tree input delay (ns): 58.407
CAM search delay (ns): 956.629
Bitline delay (ns): 220.103
Sense Amplifier delay (ns): 0.0149121
H-tree output delay (ns): 94.3026
H-tree output delay (ns): 94.3026


Power Components:

  CAM array:
  Total dynamic associative search energy/access  (nJ): 1121.21
Total energy in H-tree (that includes both match key and data transfer) (nJ): 127.082
Keyword input and result output Htrees inside bank Energy (nJ): 127.082
Searchlines (nJ): 458.16
Matchlines  (nJ): 534.875
Sub-array output driver (nJ): 1.09099

  Total dynamic read energy/access  (nJ): 46.2443
Total energy in H-tree (that includes both address and data transfer) (nJ): 43.4164
Output Htree inside bank Energy (nJ): 31.5676
Decoder (nJ): 0.0420707
Wordline (nJ): 0.000758249
Bitline mux & associated drivers (nJ): 0
Sense amp mux & associated drivers (nJ): 0
Bitlines (nJ): 0.166502
Sense amplifier energy (nJ): 0.000187022
Sub-array output driver (nJ): 2.61838

  Total leakage read/write power of a bank (mW): 760105


Area Components:

  CAM array: Area (mm2): 269501
Height (mm): 2146.2
Width (mm): 125.571
Area efficiency (Memory cell area/Total area) - 43.2577 %
MAT Height (mm): 189.382
MAT Length (mm): 5.68774
Subarray Height (mm): 81.7165
Subarray Length (mm): 2.82082

Wire Properties:

  Delay Optimal
Repeater size - 372.495
Repeater spacing - 2.15134 (mm)
Delay - 0.0550154 (ns/mm)
```

```
PowerD - 0.000625495 (nJ/mm)
PowerL - 0.0030794 (mW/mm)
PowerLgate - 0.00160486 (mW/mm)
Wire width - 0.36 microns
Wire spacing - 0.36 microns

  5% Overhead
Repeater size - 225.495
Repeater spacing - 2.75134 (mm)
Delay - 0.057753 (ns/mm)
PowerD - 0.000431832 (nJ/mm)
PowerL - 0.00145763 (mW/mm)
PowerLgate - 0.000759659 (mW/mm)
Wire width - 0.36 microns
Wire spacing - 0.36 microns

  10% Overhead
Repeater size - 199.495
Repeater spacing - 3.15134 (mm)
Delay - 0.0605024 (ns/mm)
PowerD - 0.000399415 (nJ/mm)
PowerL - 0.00112588 (mW/mm)
PowerLgate - 0.000586763 (mW/mm)
Wire width - 0.36 microns
Wire spacing - 0.36 microns

  20% Overhead
Repeater size - 160.495
Repeater spacing - 3.55134 (mm)
Delay - 0.0659862 (ns/mm)
PowerD - 0.000361002 (nJ/mm)
PowerL - 0.000803755 (mW/mm)
PowerLgate - 0.000418885 (mW/mm)
Wire width - 0.36 microns
Wire spacing - 0.36 microns

  30% Overhead
Repeater size - 138.495
Repeater spacing - 3.95134 (mm)
Delay - 0.0713336 (ns/mm)
PowerD - 0.000340961 (nJ/mm)
PowerL - 0.000623367 (mW/mm)
PowerLgate - 0.000324874 (mW/mm)
Wire width - 0.36 microns
Wire spacing - 0.36 microns

  Low-swing wire (1 mm) - Note: Unlike repeated wires,
delay and power values of low-swing wires do not
have a linear relationship with length.
delay - 1.02882 (ns)
powerD - 2.14911e-05 (nJ)
PowerL - 1.75338e-07 (mW)
PowerLgate - 1.69667e-07 (mW)
Wire width - 7.2e-07 microns
Wire spacing - 7.2e-07 microns
```

APPENDIX E

RAW CACTI SIMULATION DATA

This appendix contains the raw data that was generated by the Cacti 6.5 memory simulator for configurations defined by the SPM algorithm and content addressable memory.

Table 26: EPO, TPO, and EDP data for $w = (16, 32)$.

		SPM	(c=2)		SPM	(c=4)		SPM	(c=8)		CAM		
		EPO	TPO	EDP	EPO	TPO	EDP	EPO	TPO	EDP	EPO	TPO	EDP
	N	(nJ)	(ns)	$(nJ * ns)$	(nJ)	(ns)	$(nJ * ns)$	(nJ)	(ns)	$(nJ * ns)$	(nJ)	(ns)	$(nJ * ns)$
w=16	1024	0.236	1.976	0.467	0.143	2.118	0.304	0.163	2.952	0.481	0.129	3.307	0.426
	2048	0.256	2.028	0.520	0.162	2.333	0.377	0.228	3.300	0.753	0.240	5.230	1.254
	4096	0.287	2.118	0.608	0.198	2.403	0.476	0.358	3.742	1.341	0.691	4.501	3.110
	8192	0.323	2.333	0.755	0.241	2.554	0.615	0.587	4.247	2.494	1.200	6.593	7.913
	16384	0.396	2.403	0.952	0.326	2.952	0.963	0.855	5.512	4.714	2.724	5.236	14.264
	32768	0.482	2.554	1.230	0.457	3.300	1.506	1.306	6.069	7.926	4.609	6.403	29.512
	65536	0.652	2.952	1.926	0.717	3.742	2.683	2.058	6.728	13.847	8.217	8.791	72.228
	131072	0.913	3.300	3.013	1.175	4.247	4.989	3.264	7.865	25.670	17.278	9.208	159.093
	262144	1.434	3.742	5.365	1.710	5.512	9.427	4.985	9.006	44.892	31.151	12.110	377.240
w=32	1024	0.472	1.976	0.934	0.287	2.118	0.608	0.326	2.952	0.963	0.365	8.150	2.977
	2048	0.513	2.028	1.039	0.323	2.333	0.755	0.457	3.300	1.506	1.090	6.245	6.810
	4096	0.574	2.118	1.215	0.396	2.403	0.952	0.717	3.742	2.683	2.228	7.303	16.271
	8192	0.647	2.333	1.509	0.482	2.554	1.230	1.175	4.247	4.989	4.240	6.972	29.561
	16384	0.792	2.403	1.904	0.652	2.952	1.926	1.710	5.512	9.427	7.127	9.002	64.159
	32768	0.963	2.554	2.460	0.913	3.300	3.013	2.612	6.069	15.851	12.821	13.127	168.296
	65536	1.305	2.952	3.851	1.434	3.742	5.365	4.116	6.728	27.695	26.355	13.482	355.313
	131072	1.826	3.300	6.026	2.350	4.247	9.978	6.528	7.865	51.340	48.116	18.535	891.828
	262144	2.868	3.742	10.730	3.421	5.512	18.855	9.969	9.006	89.784	91.902	28.735	2640.789

Table 27: EPO, TPO, and EDP data for $w = (64, 128)$.

		SPM	(c=2)		SPM	(c=4)		SPM	(c=8)		CAM		
		EPO	TPO	EDP	EPO	TPO	EDP	EPO	TPO	EDP	EPO	TPO	EDP
	N	(nJ)	(ns)	$(nJ * ns)$	(nJ)	(ns)	$(nJ * ns)$	(nJ)	(ns)	$(nJ * ns)$	(nJ)	(ns)	$(nJ * ns)$
w=64	1024	0.945	1.976	1.867	0.574	2.118	1.215	0.652	2.952	1.926	1.890	9.895	18.697
	2048	1.025	2.028	2.078	0.647	2.333	1.509	0.913	3.300	3.013	3.863	11.499	44.419
	4096	1.148	2.118	2.431	0.792	2.403	1.904	1.434	3.742	5.365	7.159	10.554	75.559
	8192	1.294	2.333	3.019	0.963	2.554	2.460	2.350	4.247	9.978	12.164	14.355	174.613
	16384	1.585	2.403	3.808	1.305	2.952	3.851	3.421	5.512	18.855	22.029	22.132	487.541
	32768	1.927	2.554	4.921	1.826	3.300	6.026	5.223	6.069	31.702	44.508	22.171	986.785
	65536	2.609	2.952	7.703	2.868	3.742	10.730	8.232	6.728	55.389	82.046	31.685	2599.642
	131072	3.652	3.300	12.052	4.699	4.247	19.955	13.056	7.865	102.680	157.582	51.546	8122.659
	262144	5.736	3.742	21.461	6.842	5.512	37.709	19.938	9.006	179.568	308.149	94.986	29269.810
w=128	1024	1.890	1.976	3.734	1.148	2.118	2.431	1.305	2.952	3.851	7.989	16.312	130.314
	2048	2.050	2.028	4.157	1.294	2.333	3.019	1.826	3.300	6.026	12.997	18.237	237.019
	4096	2.295	2.118	4.862	1.585	2.403	3.808	2.868	3.742	10.730	22.236	25.740	572.350
	8192	2.588	2.333	6.038	1.927	2.554	4.921	4.699	4.247	19.955	40.445	41.455	1676.643
	16384	3.170	2.403	7.616	2.609	2.952	7.703	6.842	5.512	37.709	80.813	40.227	3250.836
	32768	3.854	2.554	9.842	3.652	3.300	12.052	10.447	6.069	63.405	149.905	59.299	8889.142
	65536	5.219	2.952	15.405	5.736	3.742	21.461	16.464	6.728	110.779	288.940	101.040	29194.498
	131072	7.305	3.300	24.104	9.398	4.247	39.911	26.112	7.865	205.360	566.073	198.772	112519.462
	262144	11.472	3.742	42.921	13.684	5.512	75.418	39.877	9.006	359.135	1121.210	453.324	508271.402

Table 28: Dynamic, static, and total power consumption data in units of Watts.

		SPM	(c=2)		SPM	(c=4)		SPM	(c=8)		CAM		
	N	Dynamic	Static	Total	Dynamic	Static	Total	Dynamic	Static	Total	Dynamic	Static	Total
w=16	1024	0.120	0.001	0.121	0.068	0.003	0.070	0.055	0.017	0.073	0.039	0.010	0.049
	2048	0.126	0.003	0.129	0.069	0.005	0.074	0.069	0.034	0.104	0.046	0.020	0.066
	4096	0.135	0.005	0.141	0.082	0.010	0.092	0.096	0.069	0.165	0.153	0.047	0.201
	8192	0.139	0.010	0.149	0.094	0.018	0.113	0.138	0.169	0.308	0.182	0.089	0.271
	16384	0.165	0.020	0.185	0.110	0.035	0.145	0.155	0.323	0.479	0.520	0.189	0.709
	32768	0.189	0.037	0.225	0.138	0.069	0.207	0.215	0.705	0.920	0.720	0.340	1.060
	65536	0.221	0.069	0.290	0.192	0.138	0.329	0.306	1.471	1.777	0.935	0.639	1.574
	131072	0.277	0.138	0.415	0.277	0.339	0.616	0.415	2.553	2.968	1.876	1.337	3.213
	262144	0.383	0.275	0.659	0.310	0.647	0.957	0.553	4.763	5.317	2.572	2.524	5.096
w=32	1024	0.239	0.003	0.242	0.135	0.005	0.141	0.110	0.035	0.145	0.045	0.060	0.105
	2048	0.253	0.005	0.258	0.139	0.010	0.149	0.138	0.069	0.207	0.175	0.133	0.307
	4096	0.271	0.010	0.281	0.165	0.020	0.185	0.192	0.138	0.329	0.305	0.268	0.573
	8192	0.277	0.020	0.298	0.189	0.037	0.225	0.277	0.339	0.616	0.608	0.511	1.119
	16384	0.330	0.040	0.370	0.221	0.069	0.290	0.310	0.647	0.957	0.792	0.962	1.753
	32768	0.377	0.073	0.451	0.277	0.138	0.415	0.430	1.410	1.840	0.977	1.852	2.828
	65536	0.442	0.138	0.580	0.383	0.275	0.659	0.612	2.943	3.554	1.955	3.821	5.776
	131072	0.553	0.276	0.829	0.553	0.678	1.231	0.830	5.107	5.937	2.596	7.391	9.987
	262144	0.767	0.551	1.317	0.621	1.294	1.914	1.107	9.526	10.633	3.198	14.570	17.768
w=64	1024	0.478	0.006	0.484	0.271	0.010	0.281	0.221	0.069	0.290	0.191	0.440	0.631
	2048	0.506	0.011	0.516	0.277	0.020	0.298	0.277	0.138	0.415	0.336	0.885	1.221
	4096	0.542	0.021	0.563	0.330	0.040	0.370	0.383	0.275	0.659	0.678	1.687	2.365
	8192	0.555	0.041	0.595	0.377	0.073	0.451	0.553	0.678	1.231	0.847	3.262	4.110
	16384	0.660	0.080	0.740	0.442	0.138	0.580	0.621	1.294	1.914	0.995	6.376	7.371
	32768	0.755	0.146	0.901	0.553	0.276	0.829	0.861	2.820	3.680	2.007	13.073	15.080
	65536	0.884	0.277	1.161	0.767	0.551	1.317	1.223	5.885	7.109	2.589	25.594	28.184
	131072	1.107	0.552	1.659	1.107	1.356	2.462	1.660	10.214	11.874	3.057	50.783	53.840
	262144	1.533	1.101	2.634	1.241	2.587	3.829	2.214	19.053	21.266	3.244	101.010	104.254
w=128	1024	0.956	0.012	0.968	0.542	0.021	0.563	0.442	0.138	0.580	0.490	3.327	3.817
	2048	1.011	0.022	1.033	0.555	0.041	0.595	0.553	0.276	0.829	0.713	6.197	6.909
	4096	1.084	0.041	1.125	0.660	0.080	0.740	0.767	0.551	1.317	0.864	12.131	12.995
	8192	1.109	0.081	1.190	0.755	0.146	0.901	1.107	1.356	2.462	0.976	23.860	24.836
	16384	1.319	0.161	1.480	0.884	0.277	1.161	1.241	2.587	3.829	2.009	48.785	50.794
	32768	1.509	0.293	1.802	1.107	0.552	1.659	1.721	5.639	7.361	2.528	96.023	98.551
	65536	1.768	0.553	2.321	1.533	1.101	2.634	2.447	11.770	14.217	2.860	191.052	193.912
	131072	2.214	1.103	3.317	2.213	2.712	4.925	3.320	20.427	23.747	2.848	380.551	383.399
	262144	3.066	2.202	5.269	2.483	5.174	7.657	4.428	38.105	42.533	2.473	760.105	762.578

REFERENCES

[1] Abraham, A. and Thomas, J., "Distributed intrusion detection systems: A computational intelligence approach," in *Applications of Information Systems to Homeland Security and Defense* (Abbass, H. and Essam, D., eds.), pp. 105–135, Idea Group, 2005.

[2] Acharya, S., "Situational awareness in computer network defense," ch. GCD: A global collaborative defense approach to thwart Internet attacks, IGI Global, 2012.

[3] Acharya, S., Abliz, M., Mills, B., Znati, T. F., Wang, J., Ge, Z., and Greenberg, A., "Optwall: A hierarchical traffic-aware firewall," in *Proceedings of the 14th Annual Network and Distributed System Security Symposium (NDSS'07)*, (San Diego, CA, USA), 2007.

[4] Acharya, S., Wang, J., Ge, Z., Znati, T., and Greenberg, A., "Simulation study of firewalls to aid improved performance," in *Proceedings of the 39th Annual Symposium on Simulation (ANSS '06)*, (Washington, DC, USA), pp. 18–26, IEEE Computer Society, 2006.

[5] Acharya, S., Wang, J., Ge, Z., Znati, T. F., and Greenberg, A., "Traffic-aware firewall optimization strategies," in *Proceedings of the IEEE International Conference on Communications (ICC'06)*, vol. 5, (Istanbul, Turkey), pp. 2225–2230, IEEE, June 2006.

[6] Amini, M., Jalili, R., and Shahriari, H., "RT-UNNID: A practical solution to real-time network-based intrusion detection using unsupervised neural networks," *Computers & Security*, vol. 25, no. 6, pp. 459–468, 2006.

[7] Anagnostakis, K. G. and Greenwald, M. B., "A cooperative immunization system for an untrusting Internet," in *In Proceedings of the 11th IEEE International Conference on Networks (ICON)*, pp. 403–408, 2003.

[8] Anderson, J., *Computer Security Threat Monitoring and Surveillance.* Fort Washington, PA: James P. Anderson Co., 1980.

[9] Anuar, N., Papadaki, M., Furnell, S., and Clarke, N., "An investigation and survey of response options for intrusion response systems (irss)," in *Information Security for South Africa (ISSA)*, pp. 1–8, IEEE, 2010.

[10] Arsovski, I., Chandler, T., and Sheikholeslami, A., "A ternary content addressable memory based on 4T static storage and including a current-race sensing scheme," *Journal of Solid-state Circuits*, vol. 38, no. 1, pp. 155–158, 2003.

[11] Athanasiades, N., Abler, R., Levine, J., Owen, H., and Riley, G., "Intrusion detection testing and benchmarking methodologies," in *Proceedings of the First IEEE International Workshop on Information Assurance (IWIA'03)*, IEEE, 2003.

[12] Axelsson, S., "The base-rate fallacy and the difficulty of intrusion detection," *ACM Transactions on Information and System Security*, vol. 3, no. 3, pp. 186–205, 2000.

[13] Axelsson, S., "Intrusion detection systems: A survey and taxonomy," in *Technical Report, Dept. of Computer Engineering, Chalmers University of Technology*, (Sweden), pp. 99–115, 2000.

[14] Baboescu, F., Singh, S., and Varghese, G., "Packet classification for core routers: Is there an alternative to CAMS?," in *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2003)*, pp. 53–63, IEEE, 2003.

[15] Bankovic, Z., Moya, J. M., Araujo, A., Fraga, D., Vallejo, J. C., and Goyeneche, J.-M., "Distributed intrusion detection system for wireless sensor networks based on a reputation system coupled with kernel self-organizing maps," *Integrated Computer-Aided Engineering*, vol. 17, pp. 87–102, April 2010.

[16] Basu, A. and Narlikar, G., "Fast incremental updates for pipelined forwarding engines," *IEEE/ACM Transactions on Networking*, vol. 13, no. 3, pp. 690–703, 2005.

[17] Bellovin, S. M., "Distributed firewalls," *;login:*, vol. 24, no. 5, pp. 37–47, 1999.

[18] Berg, M., Kreveld, M., Overmars, M., and Schwarzkopf, O., *Computational Geometry*. Springer-Verlag, 2000.

[19] Berners-Lee, T., Hendler, J., and Lassila, O., "The semantic web," *Scientific American*, pp. 35–43, May 2001.

[20] Brin, S. and Page, L., "The anatomy of a large-scale hypertextual web search engine," in *Proceedings of the Seventh International World Wide Web Conference*, pp. 107–117, 1998.

[21] Brooks, D., Tiwari, V., and Martonosi, M., "Waatch: A framework for architectural-level power analysis and optimizations," in *Proceedings of the 27th International Symposium on Computer Architecture (ISCA)*, pp. 83–94, 2000.

[22] Butcher, D., Li, X., and Guo, J., "Security challenge and defense in VoIP infrastructures," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 37, pp. 1152 –1162, November 2007.

[23] Caragata, D., Assad, S., Tutanescu, I., Shoniregun, C., and Akmayeva, G., "Security of mobile Internet access with UMTS/HSDPA/LTE," in *World Congress on Internet Security (WorldCIS)*, pp. 272–276, February 2011.

[24] Chen, T. and Abu-Nimeh, S., "Lessons from stuxnet," *IEEE Computer*, vol. 44, pp. 91–93, April 2011.

[25] Chen, Z., Ji, C., and Barford, P., "Spatial-temporal characteristics of internet malicious sources," in *IEEE 27th Conference on Computer Communications (INFOCOM 2008)*, pp. 2306–2314, IEEE, 2008.

[26] Cheswick, W., Bellovin, S., and Rubin, A., *Firewalls and Internet Security*. Boston, Massachusetts: Addison-Wesley, second ed., 2003.

[27] Cheung, G. and McCanne, S., "Optimal routing table design for ip address lookups under memory constraints," in *Proceedings of IEEE International Conference on Computer Communications (INFOCOM'99)*, pp. 1437–1444, IEEE, 1999.

[28] Chien, E. and Szor, P., "Blended attacks, exploits, vulnerabilities and buffer-overflow techniques in computer viruses," in *Proceedings of the Virus Bulletin Conference (VB2002)*, 2002.

[29] Colas, C., "Developing trusted services for mobile using global platform standards," in *7th European Telecommunications Standards Institute (ETSI) Security Workshop*, January 2012.

[30] Coppolino, L., "Enhancing SIEM technology for protecting critical infrastructures," in *7th European Telecommunications Standards Institute (ETSI) Security Workshop*, January 2012.

[31] Cormen, T., Leiserson, C., and Rivest, R., *Introduction to Algorithms.* Cambridge, MA: MIT Press, 1990.

[32] D'Antonio, S., "Protecting power grids from cyber-attacks: the INSPIRE approach," in *6th European Telecommunications Standards Institute (ETSI) Security Workshop*, January 2011.

[33] Degermark, M., Brodnik, A., Carlsson, S., and Pink, S., "Small forwarding tables for fast routing lookups," in *Proceedings of the ACM Special Interest Group on Communications (SIGCOMM'97)*, pp. 3–14, 1997.

[34] Demchenko, Y., de Laat, C., Lopez, D., and Garcia-Espin, J., "Security services lifecycle management in on-demand infrastructure services provisioning," in *IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 644 –650, December 2010.

[35] Depoy, J., Phelan, J., Sholander, P., Smith, B., Varnado, G., and Wyss, G., "Risk assessment for physical and cyber attacks on critical infrastructures," in *IEEE Military Communications Conference (MILCOM 2005)*, pp. 1961–1969, 2005.

[36] Didaci, L., Giacinto, G., and Roli, F., "Ensemble learning for intrusion detection in computer networks," in *Proceedings of the AI*IA Workshop on Apprendimento automatico: metodi e applicazioni.*, 2002.

[37] DoD, "DoD 5200.28-STD: Department of defense trusted computer system evaluation criteria." United States Department of Defense, 1985.

[38] Eck, O. and Schaefer, D., "A semantic file system for integrated product data management," *Advanced Engineering Informatics*, vol. 25, no. 2, pp. 177–184, 2011.

[39] ENGEN, V., *Machine learning for network based intrusion detection.* PhD thesis, Bournemouth University, 2010.

[40] Fletcher, K. and Liu, X., "Security requirements analysis, specification, prioritization and policy development in cyber-physical systems," in *5th International Conference on Secure Software Integration Reliability Improvement Companion (SSIRI-C)*, pp. 106–113, June 2011.

[41] Fogla, P., Sharif, M., Perdisci, R., Kolesnikov, O. M., and Lee, W., "Polymorphic blending attacks," in *Proceedings of the 2006 USENIX Security Symposium*, 2006.

[42] Fulp, E. W., "Optimization of network firewall policies using ordered sets and directed acyclical graphs," tech. rep., Wake Forest University, Department of Computer Science, 2004.

[43] Ganame, A., Bourgeois, J., Bidou, R., and Spies, F., "A global security architecture for intrusion detection on computer networks," *Journal of Computers and Security*, vol. 27, no. 1-2, pp. 30–47, 2008.

[44] Gang, G., Zeyong, L., and Jun, J., "Internet of things security analysis," in *International Conference on Internet Technology and Applications (iTAP)*, pp. 1–4, August 2011.

[45] Ghorbani, A., Lu, W., and Tavallaee, M., "Detection approaches," *Network Intrusion Detection and Prevention*, pp. 27–53, 2010.

[46] Govind, N., "Developments in universal core-to-cloud layered security interface model," in *7th European Telecommunications Standards Institute (ETSI) Security Workshop*, January 2012.

[47] GTISC, "Emerging cyber threats report 2011," tech. rep., Georgia Institute of Technology: Georgia Tech Information Security Center, 2010.

[48] Gupta, P., *Algorithms for routing lookups and packet classification.* Phd dissertation, Stanford University, Department of Computer Science, 2000.

[49] Hamed, H. and Al-Shaer, E., "Dynamic rule-ordering optimization for high-speed firewall filtering," in *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security (ASIACCS'06)*, (New York, NY, USA), pp. 332–342, ACM, 2006.

[50] Hamed, H., El-Atawy, A., and Al-Shaer, E., "Adaptive statistical optimization techniques for firewall packet filtering," in *Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM'06)*, pp. 1–12, IEEE, 2006.

[51] Hamed, H., El-Atawy, A., and Al-Shaer, E., "On dynamic optimization of packet matching in high-speed firewalls," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 10, pp. 1817–1830, 2006.

[52] Han, S. and Cho, S., "Evolutionary neural networks for anomaly detection based on the behavior of a program," *IEEE Transactions on Systems, Man, and Cybernetics: Part B*, vol. 36, no. 3, pp. 559–570, 2006.

[53] Hanson, E. N. and Chaabouni, M., "The ibs-tree: A data structure for finding all the intervals that overlap a point," Tech. Rep. TR94-040, Wright State University, School of Computer Science, 1994.

[54] Hartley, R. and Barnden, J., "Semantic networks: Visualizations of knowledge," *Trends in Cognitive Science*, vol. 1, pp. 169–175, 1997.

[55] Hasan, J. and Vijaykumar, T., "Dynamic pipelining: Making IP lookup truly scalable," in *Proceedings of the ACM SIGCOMM*, pp. 205–216, 2005.

[56] He, X., Peddersen, J., and Parameswaran, S., "LOP: A novel SRAM-based architecture for low power and high throughput packet classification," in *Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, CODES+ISSS'09, (New York, NY, USA), pp. 137–146, ACM, 2009.

[57] He, X., Peddersen, J., and Parameswaran, S., "LOP_RE: Range encoding for low power packet classification," in *IEEE 34th Conference on Local Computer Networks (LCN'09)*, pp. 137–144, 2009.

[58] Hennessy, J. and Patterson, D., *Computer Architecture: A Quantitative Approach, Third Edition.* San Francisco, CA, USA: Morgan Kaufmann, 2003.

[59] Hofmann, A., Schmitz, C., and Sick, B., "Rule extraction from neural networks for intrusion detection in computer networks," in *IEEE International Conference on Systems, Man, and Cybernetics*, pp. 1259–1265, IEEE, 2003.

[60] Horng, S., Su, M., Chen, Y., Kao, T., Chen, R., Lai, J., and Perkasa, C., "A novel intrusion detection system based on hierarchical clustering and support vector machines," *Expert Systems with Applications*, vol. 38, pp. 306–313, 2011.

[61] HP-Labs, "The CACTI integrated memory simulator (website)." http://www.hpl.hp.com/research/cacti/.

[62] Hsieh, M.-y., Rodrigues, A., Riesen, R., Thompson, K., and Song, W., "A framework for architecture-level power, area, and thermal simulation and its application to network-on-chip design exploration," *SIGMETRICS Performance Evaluation Review*, vol. 38, no. 4, pp. 63–68, 2011.

[63] Iheagwara, C., Awan, F., Acar, Y., and Miller, C., "Maximizing the benefits of intrusion prevention systems: Effective deployment strategies," in *Proceedings of the 18th Annual Forum of Incident Response and Security Teams (FIRST) Conference*, 2006.

[64] Ioannidis, S., Keromytis, A., Bellovin, S., and Smith, J., "Implementing a distributed firewall," in *Proceedings of the Computer and Communications Security (CCS) Conference*, pp. 190–199, 2000.

[65] Isaac, J., Zeadally, S., and Camara, J., "Security attacks and solutions for vehicular ad hoc networks," *IET Communications*, vol. 4, no. 7, pp. 894–903, 2010.

[66] ISTS, "Law enforcement tools and technologies for investigating cyber attacks: Gap analysis report," tech. rep., Dartmouth College: Institute for Security, Technology, and Society, February 2004.

[67] ITU-T, "X Series: Data networks, open system communications and security." International Telecommunication Union (ITU), Telecommunication Standardization Sector (ITU-T).

[68] Janakiraman, R., Waldvogel, M., and Zhang, Q., "Indra: a peer-to-peer approach to network intrusion detection and prevention," in *Proceedings of the 12th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET-ICE'03)*, pp. 226–231, IEEE, 2003.

[69] Jiang, W. and Prasanna, V. K., "A memory-balanced linear pipeline architecture for trie-based IP lookup," in *Symposium on High-Performance Interconnects*, (Los Alamitos, CA, USA), pp. 83–90, IEEE Computer Society, 2007.

[70] Jiang, W. and Prasanna, V. K., "Sequence-preserving parallel IP lookup using multiple SRAM-based pipelines," *Journal of Parallel and Distributed Computing*, vol. 69, no. 9, pp. 778–789, 2009.

[71] Jirapummin, C., Wattanapongsakorn, N., and P., K., "Hybrid neural networks for intrusion detection system," in *Proceedings of the International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC 2002)*, 2002.

[72] Kabiri, P. and Ghorbani, A., "Research on intrusion detection and response: A survey," *International Journal of Network Security*, vol. 1, no. 2, pp. 84–102, 2005.

[73] Kahng, A. B., Li, B., Peh, L.-S., and Samadi, K., "ORION 2.0: A fast and accurate NoC power and area model for early-stage design space exploration," in *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '09, (3001 Leuven, Belgium, Belgium), pp. 423–428, European Design and Automation Association, 2009.

[74] Kasai, G., Takarabe, Y., Furumi, K., and Yoneda, M., "200MHz/200MSPS 3.2W at 1.5V Vdd, 9.4Mbits ternary CAM with new charge injection match detect circuits and bank selection scheme," in *Proceedings of the Custom Integrated Circuits Conference*, pp. 387–390, IEEE, 2003.

[75] Khanafer, M., Guennoun, M., and Mouftah, H., "Intrusion detection system for WSN-based intelligent transportation systems," in *IEEE Global Telecommunications Conference (GLOBECOM 2010)*, December 2010.

[76] Khor, K., Ting, C., and Amnuaisuk, S., "From feature selection to building of bayesian classifiers: A network intrusion detection perspective," *American Journal of Applied Sciences*, vol. 6, no. 11, pp. 1949–1960, 2009.

[77] Kim, H. and Kang, I., "On the effectiveness of martian address filtering and its extensions," in *Global Telecommunications Conference*, pp. 1348–1353, IEEE, 2003.

[78] Kim, J.-Y., Choi, H.-K., and Copeland, J., "An efficient authentication scheme for security and privacy preservation in V2I communications," in *IEEE 72nd Vehicular Technology Conference*, September 2010.

[79] Kim, K. S. and Sahni, S., "Efficient construction of pipelined multibit-trie router-tables," *IEEE Transactions on Computers*, vol. 56, pp. 32–43, 2007.

[80] Kim, N., Austin, T., Blaauw, D., Mudge, T., Flautner, K., Hu, J., Irwin, M., Kandemir, M., and Vijaykrishnan, N., "Leakage current: Moore's law meets static power.," *Computer*, vol. 36, no. 12, pp. 65–77, 2003.

[81] Kim, S. H., Wang, Q.-H., and Ullrich, J. B., "A comparative study of cyberattacks," *Communications of the ACM*, vol. 55, no. 3, pp. 66–73, 2012.

[82] Kohonen, T., "The self-organizing map," in *Proceedings of the IEEE*, vol. 78, pp. 1464–1480, 1990.

[83] Kohonen, T., Simula, O., and Oja, E., "Engineering applications of the self-organizing map," *Proceedings of the IEEE*, vol. 84, no. 10, pp. 1358 – 1384, 1996.

[84] Kumar, P. and Selvakumar, S., "Distributed denial of service attack detection using an ensemble of neural classifier," *Computer Communications*, vol. 34, pp. 1328–1341, 2011.

[85] Lehmann, F., "Semantic networks," *Computers and Mathematics with Applications*, vol. 23, no. 2-5, pp. 1–50, 1992.

[86] Lehmann, F. and Rodin, E., *Semantic networks in artificial intelligence.* International series in modern applied mathematics and computer science, Pergamon Press, 1992.

[87] Li, S., Ahn, J., Strong, R., Brockman, J., Tullsen, D., and Jouppi, N., "McPat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, ACM, 2009.

[88] Lichodzijewski, P., Zincir-Heywood, A., and Heywood, M., "Dynamic intrusion detection using self-organizing maps," in *Proceedings of the 14th Annual Canadian Information Technology Security Symposium*, 2002.

[89] Lichodzijewski, P., Zincir-Heywood, A., and Heywood, M., "Host based intrusion detection using self-organizing maps," in *Proceedings of the 2002 IEEE World Congress on Computation Intelligence*, 2002.

[90] Lippmann, R., Haines, J. W., Fried, D. J., Korba, J., and Das, K., "The 1999 DARPA off-line intrusion detection evaluation," *Computer Networks*, vol. 34, no. 4, pp. 579–595, 2000.

[91] Liu, J., Demirkiran, I., Yang, T., and Helfrick, A., "Feasibility study of IEEE 802.15.4 for aerospace wireless sensor networks," in *IEEE/AIAA 28th Digital Avionics Systems Conference (DASC'09)*, October 2009.

[92] Locasto, M., Parekh, J., Keromytis, A., and Stolfo, S., "Towards collaborative security and P2P intrusion detection," in *Proceedings of the 2005 IEEE workshop on Information assurance and security*, pp. 30–36, 2005.

[93] Mamidipaka, M. and Dutt, N., "eCACTI: An enhanced power estimation model for on-chip caches," Tech. Rep. CECS Technical Report 04-28, University of California, Irvine, September 2004.

[94] Markham, T., Meredith, L., and Payne, C., "Distributed embedded firewalls with virtual private groups," in *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX'03)*, IEEE, 2003.

[95] Masud, M., Al-khateeb, T., Khan, L., Thuraisingham, B., and Hamlen, K., "Flow-based identification of botnet traffic by mining multiple log files," *Proceedings of the First International Conference on Distributed Frameworks and Applications*, 2008.

[96] McHugh, J., "Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory," *ACM Transactions on Information and System Security*, vol. 3, no. 4, pp. 262–294, 2000.

[97] Meredith, L., "A summary of the autonomic distributed firewall (ADF) project," in *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX'03)*, IEEE, 2003.

[98] Moya, J., Araujo, A., Z., B., Goyeneche, J., Vallejo, J., Malagon, P., Villanueva, D., Fraga, D., Romero, E., and Blesa, J., "Improving security for scada sensor networks with reputation systems and self-organizing maps," *Sensors*, vol. 9, no. 11, pp. 9380–9397, 2009.

[99] Mukkamala, S., Janoski, G., and Sung, A., "Intrustion detection using neural networks and support vector machines," in *Proceedings of the 2002 International Joint Conference on Neural Networks*, pp. 1702–1707, 2002.

[100] Mukkamala, S. and Sung, A., "Artificial intelligence techniques for intrusion detection," in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, 2003.

[101] Mukkamala, S. and Sung, A. H., "A comparative study of techniques for intrusion detection," in *Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2003)*, 2003.

[102] Munz, G., Fessi, A., Carle, G., Carlinet, Y., Paul, O., Yusuf, S., Sloman, M., Thing, V., Gabrijeleie, D., Lunteren, J., Sagmeister, P., and Dittmann, G., "Diadem firewall: Web server overload attack detection and response," in *Proceedings of Broadband Europe (BBEurope)*, (Bordeaux, France), 2005.

[103] Muraleedharan, R. and Osadciw, L., "Secure self-adaptive mission-critical communication for distributed smart home sensor network," in *6th Annual International on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous'09)*, July 2009.

[104] Muralimanohar, N., Balasubramonian, R., and Jouppi, N., "CACTI 6.0: A tool to model large caches," Tech. Rep. HPL-2009-85, Hewlett-Packard Laboratories, April 2009.

[105] Muralimanohar, N., Balasubramonian, R., and Jouppi, N., "Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0," in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 40, (Washington, DC, USA), pp. 3–14, IEEE Computer Society, 2007.

[106] NAE, "The National Academy of Engineering: Grand challenges for engineering." http://www.engineeringchallenges.org, 2010.

[107] Nilsson, S. and Karlsson, G., "Ip-address lookup using lc-tries," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 6, pp. 1083–1092, 1999.

[108] Pagiamtzis, K. and Sheikholeslami, A., "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, 2006.

[109] Pathan, A.-M., Mottalib, M., and Zibran, M., "An Internet framework to bring coherence between WAP and HTTP ensuring better mobile internet security," in *The 8th International Conference of Advanced Communication Technology (ICACT)*, February 2006.

[110] Paul, S., Pan, J., and Jain, R., "Architectures for the future networks and the next generation Internet: A survey," *Journal of Computer Communications*, vol. 34, pp. 2–42, 2011.

[111] Pearl, J., *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Los Altos, California: Morgan Kaufmann, 1988.

[112] Pearl, J. and Russell, S., "Bayesian networks," in *Handbook of Brain Theory and Neural Networks*, (Cambridge, MA, USA), MIT Press, 2002.

[113] Perdisci, R., Ariu, D., Fogla, P., Giacinto, G., and Lee, W., "McPAD: A multiple classifier system for accurate payload-based anomaly detection," *Computer Networks: The International Journal of Computer and Telecommunications Networking*, vol. 53, no. 6, pp. 864–881, 2009.

[114] Perdisci, R., Gu, G., and Lee, W., "Using an ensemble of one-class SVM classifiers to harden payload-based anomaly detection systems.," in *Proceedings of the Sixth International Conference on Data Mining (ICDM'06)*, pp. 488–498, 2006.

[115] Porras, P. and Neumann, P., "Emerald: Event monitoring enabling responses to anomalous live disturbances," in *Proceedings of the 20th National Information Systems Security Conference (NISSC)*, pp. 353–365, 1997.

[116] Reinman, G. and Jouppi, N., "CACTI 2.0: An integrated cache timing and power model," Tech. Rep. WRL 2000/7, Compaq Western Research Laboratory, February 2000.

[117] Ren, K., Lou, W., Kim, K., and Deng, R., "A novel privacy preserving authentication and access control scheme for pervasive computing environments," *IEEE Transactions on Vehicluar Technology*, vol. 55, no. 4, pp. 1373–1384, 2006.

[118] Resnik, P., "Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language," *Journal Of Artificial Intelligence Research*, vol. 11, pp. 95–130, 1999.

[119] Rhodes, B. and adn J. Cannady, J. M., "Multiple self-organizing maps for intrusion detection," in *Proceedings of the 23rd National Information Systems Security Conference*, pp. 16–19, 2000.

[120] Rodriguez, M., "Grammar-based random walkers in semantic networks," *Knowledge-Based Systems*, vol. 21, no. 1, pp. 727–739, 2008.

[121] Rodriguez, M. A. and Watkins, J. H., "Grammar-based geodesics in semantic networks," *Knowledge-Based Systems*, vol. 23, no. 8, 2010.

[122] Rohwer, K. and Krout, T., "Multiple levels of security in support of highly mobile tactical internets-ELB ACTD," in *IEEE Military Communications Conference (MILCOM)*, pp. 81–86, 2001.

[123] Ruighaver, A., "Organisational security requirements: An agile approach to ubiquitous information security," in *Proceedings of the Sixth Australian Information Security Management Conference*, 2008.

[124] Rutkowski, A., Kadobayashi, Y., Furey, I., Rajnovid, D., Martin, R., and Takahashi, T., "Cybex the cybersecurity information exchange framework (X.1500)," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 5, pp. 59–64, 2010.

[125] Rutkowski, T., "Enhancing security for next generation networks and cloud computing," in *6th European Telecommunications Standards Institute (ETSI) Security Workshop*, January 2011.

[126] Rutkowski, T., "The emerging cloud ecosystem: cyber security plus LI/RD," in *7th European Telecommunications Standards Institute (ETSI) Security Workshop*, January 2012.

[127] Sabelfeld, A. and Myers, A., "Language-based information-flow security," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 1, pp. 5–19, 2003.

[128] Sadasivan, G. and Brownlee, N., "Rfc 5470: Architecture model for ip flow information export." IETF Network Working Group, March 2009.

[129] Sadok, D., Souto, E., Feitosa, E., Kelner, J., and Westberg, L., "RIP - a robust IP access architecture," *IEEE Transactions on Vehicluar Technology*, vol. 57, no. 6, pp. 3357–3368, 2008.

[130] Sahlin, B., "3GPP security," in *7th European Telecommunications Standards Institute (ETSI) Security Workshop*, January 2012.

[131] Salmon, B., Schlosser, S., Cranor, L., and Ganger, G., "Perspective: Semantic data management for the home," in *Proceedings of the 7th Usenix Conference on File and Storage Technologies (FAST'09)*, (San Francisco, CA, USA), Usenix, 2009.

[132] Schnackenberg, D., Djahandari, K., and Sterne, D., "Infrastructure for intrusion detection and response," in *Proceedings of the 2000 DARPA InformationSurvivability Conference and Exposition*, (Hilton Head, SC), 2000.

[133] Schnackenberg, D., Holliday, H., Smith, R., Djahandari, K., and Sterne, D., "Cooperative intrusion traceback and response architecture (citra)," in *Proceedings of the 2001 DARPA Information Survivability Conference and Exposition*, pp. 56–68, 2001.

[134] Schneider, F., Morrisett, G., and Harper, R., "A language-based approach to security," in *Informatics - 10 Years Back. 10 Years Ahead*, (London, UK), pp. 86–101, Springer-Verlag, 2001.

[135] Shirey, R., "Rfc 2828: Internet security glossary." IETF Network Working Group, May 2000.

[136] Shivakumar, P. and Jouppi, N., "CACTI 3.0: An integrated cache timing, power, and area model," Tech. Rep. WRL 2001/2, Compaq Western Research Laboratory, August 2001.

[137] SNORT-IDPS, "The snort intrusion detection and prevention system." http://www.snort.org.

[138] Soldo, F., Defrawy, K., and Markopoulou, A., "Filtering sources of unwanted traffic," in *Information Theory and Applications Workshop*, pp. 199–208, IEEE, 2008.

[139] Srinivasan, V. and Varghese, G., "Fast address lookups using controlled prefix expansion," *ACM Transactions on Computer Systems*, vol. 17, no. 1, pp. 1–40, 1999.

[140] Srinivasan, V., Varghese, G., Suri, S., and Waldvogel, M., "Fast and scalable layer four switching," *SIGCOMM Computer Communications Review*, vol. 28, no. 4, pp. 191–202, 1998.

[141] Stakhanova, N., Basu, S., and Wong, J., "A taxonomy of intrusion response systems," *International Journal of Information and Computer Security*, vol. 1, no. 1/2, pp. 169–184, 2007.

[142] Stallings, W., *Network Security Essentials: Applications and Standards.* Upper Saddle River, New Jersey: Prentice Hall, second ed., 2003.

[143] Symantec, "Symantec global Internet security thread report." http://www.symantec.com/business/threatreport/archive.jsp, 2010.

[144] Tarjan, D., Thoziyoor, S., and Jouppi, N., "CACTI 4.0," Tech. Rep. HPL-2006-86, Hewlett-Packard Laboratories, June 2006.

[145] Tavallaee, M., Bagheri, E., Lu, W., and Ghorbani, A. A., "A detailed analysis of the KDD CUP 99 data set," in *Proceedings of the Second IEEE international conference on Computational intelligence for security and defense applications*, CISDA'09, (Piscataway, NJ, USA), pp. 53–58, IEEE Press, 2009.

[146] Tavallaee, M., Stakhanova, N., and Ghorbani, A. A., "Toward credible evaluation of anomaly-based intrusion-detection methods," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications*, vol. 40, no. 5, pp. 516–524, 2010.

[147] Taylor, D. E., "Survey and taxonomy of packet classification techniques," *ACM Computing Surveys*, vol. 37, no. 3, pp. 238–275, 2005.

[148] TCSEM, "Ieee technical committee on semantic computing." http://www.computer.org/portal/web/tandc/tcsem.

[149] Thames, L. and Abler, R., "Implementing distributed internet security using a firewall collaboration framework," in *Proceedings of the 2007 IEEE Southeast Conference*, (Richmond, Virginia).

[150] Thames, L., Abler, R., and Keeling, D., "A distributed active response architecture for preventing ssh dictionary attacks," in *Proceedings of the 2008 IEEE Southeast Conference*, (Huntsville, Alabama).

[151] Thames, L., Abler, R., and Keeling, D., "A distributed firewall and active response architecture providing preemptive protection," in *Proceedings of the 2008 ACM Southeast Conference (ACMSE08*, (Auburn, Alabama).

[152] Thames, L., Abler, R., and Saad, A., "Hybrid intelligent systems for network security," in *Proceedings of the 2006 ACM Southeast Conference (ACMSE06)*, (Melbourne, Florida).

[153] Thoziyoor, S., Muralimanohar, N., Ahn, J., and Jouppi, N., "CACTI 5.1," Tech. Rep. HPL-2008-20, Hewlett-Packard Laboratories, April 2008.

[154] Thuraisingham, B., Khan, L., Masud, M., and Hamlen, K., "Data mining for security applications," in *Proceedings of the IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, pp. 585–589, IEEE, 2008.

[155] Uflacker, M. and Zeier, A., "A semantic network approach to analyzing virtual team interactions in the early stages of conceptual design," *Future Generation Computer Systems*, vol. 27, no. 1, pp. 88–99, 2011.

[156] Venayagamoorthy, G. K., "Dynamic, stochastic, computational, and scalable technologies for smart grids," *IEEE Computational Intelligence Magazine*, vol. 6, no. 3, pp. 22–35, 2011.

[157] Wang, H.-S., Zhu, X., Peh, L.-S., and Malik, S., "Orion: A power-performance simulator for interconnection networks," in *Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*, MICRO 35, (Los Alamitos, CA, USA), pp. 294–305, IEEE Computer Society Press, 2002.

[158] Wenyin, L., Fang, N., Quan, X., Qiu, B., and Liu, G., "Discovering phishing target based on semantic link network," *Future Generation Computer Systems*, vol. 26, pp. 381–388, 2010.

[159] Wilton, S. and Jouppi, N., "An enhanced access and cycle time model for on-chip caches," Tech. Rep. WRL 93/5, Compaq Western Research Laboratory, 1994.

[160] Yan, Y., Qian, Y., and Sharif, H., "A secure data aggregation and dispatch scheme for home area networks in smart grid," in *IEEE Global Telecommunications Conference (GLOBECOM 2011)*, December 2011.

[161] Yegneswaran, V., Barford, P., and Jha, S., "Global intrusion detection in the domino overlay system," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2004.

[162] Yegneswaran, V., Barford, P., and Ullrich, J., "Internet intrusions: global characteristics and prevalence," *SIGMETRICS Performance Evaluation Review*, vol. 31, no. 1, pp. 138–147, 2003.

[163] You-guo, L. and Ming-fu, J., "The reinforcement of communication security of the internet of things in the field of intelligent home through the use of middleware," in *Fourth International Symposium on Knowledge Acquisition and Modeling (KAM)*, pp. 254 –257, October 2011.

[164] Yue, K., Liu, W., Wang, X., Zhou, A., and Li, J., "Discovering semantic associations among web services based on the qualitative probabilistic network," *Expert Systems with Applications*, vol. 36, pp. 9082–9094, 2009.

[165] Zhang, C., Lin, X., Lu, R., Ho, P., and Shen, X., "An efficient message authentication scheme for vehicular communications," *IEEE Transactions on Vehicluar Technology*, vol. 57, no. 6, pp. 3357–3368, 2008.

[166] Zhang, J., Porras, P., and Ullrich, J., "Highly predictive blacklisting," in *Proceedings of the 17th USENIX Security Symposium*, pp. 107–122, July 2008.

[167] Zhang, J., Porras, P. A., and Ullrich, J., "Gaussian process learning for cyber-attack early warning," in *Proceedings of the SIAM International Conference on Data Mining*, pp. 255–264, 2008.

[168] Zhu, B., Joseph, A., and Sastry, S., "A taxonomy of cyber attacks on SCADA systems," in *2011 International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing*, pp. 380–388, 2011.

[169] Ziegler, M., Mueller, W., Schaefer, R., and Loeser, C., "Secure profile management in smart home networks," in *Sixteenth International Workshop on Database and Expert Systems Applications*, pp. 209–213, August 2005.

[170] Zou, C., Towsley, D., and Weibo, G., "A firewall network system for worm defense in enterprise networks," Tech. Rep. TR-04-CSE-01, University of Massachusetts, 2004.

[171] Zukowski, C. and Wang, S., "Use of selective precharge for low power content addressable memories," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 1788–1791, IEEE, 1997.

VITA

Lane Thames was born on April 19, 1972 in McComb, Mississippi. He attended elementary and high school at Parklane Academy, graduating in 1990. Lane began his career working within the transportation industry. His interests in electrical, computer, and mechanical systems inspired him to pursue an engineering degree. He began his collegiate career as an undergraduate student at Armstrong Atlantic State University. In his second year as an undergraduate, Lane transferred to Georgia Tech where he received his B.S. degree in Computer Engineering with Highest Honors in December of 2003.

While pursuing undergraduate studies, Lane worked for Georgia Tech as a lead teaching assistant. During this time, he realized that he had a burning desire to teach and conduct cutting-edge research. As such, he made the decision to pursue a course of graduate study that would eventually lead him to receive a Ph.D. in engineering. During the Spring of 2006, Lane received his M.S. degree in Electrical and Computer Engineering from Georgia Tech, and he will receive his Ph.D. in Electrical and Computer Engineering from Georgia Tech in December 2012.

Lane's PhD dissertation includes an investigation based on three primary thrusts: 1) distributed active-response firewall systems and architectures enabling globalized Internet security, 2) detection of computer and network attacks using computational intelligence and hybrid intelligence systems, and 3) high-speed, energy-efficient hardware-based algorithms for the multidimensional packet classification problem. His research and professional interests are diverse and revolve around the intersection of several topics including high-performance Internet Protocol networking, Internet security, cloud-based design and manufacturing systems, engineering education, computational intelligence systems, knowledge discovery in data, and large-scale distributed systems integration.

During his graduate studies, Lane spent 18 months working for VeriSign's communication services division. In December 2006, Lane became a full-time staff member at Georgia

Tech working for the information technology department on the Georgia Tech Savannah campus.

On July 23, 1994 Lane married his wife Linda A. Fruda, who holds a doctorate in nursing practice from the University of Alabama and currently practices in the field of neurology.