

TOWARD HARDWARE-ORIENTED DEFENSIVE NETWORK INFRASTRUCTURE

BY

HAO CHEN

BS, Zhejiang University, 2003

DISSERTATION

Submitted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy in Electrical Engineering
in the Graduate School of
Binghamton University
State University of New York
2015

UMI Number: 3713553

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



UMI 3713553

Published by ProQuest LLC (2015). Copyright in the Dissertation held by the Author.

Microform Edition © ProQuest LLC.

All rights reserved. This work is protected against unauthorized copying under Title 17, United States Code



ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

© Copyright by HAO CHEN 2015

All Rights Reserved

Accepted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy in Electrical Engineering
in the Graduate School of
Binghamton University
State University of New York
2015

November 20, 2014

Dr. Yu Chen, Chair and Faculty Advisor
Department of Electrical and Computer Engineering, Binghamton University

Dr. Douglas H. Summerville, Member
Department of Electrical and Computer Engineering, Binghamton University

Dr. Aleksey S. Polunchenko, Member
Department of Mathematical Sciences, Binghamton University

Dr. Zhanpeng Jin, Member
Department of Electrical and Computer Engineering, Binghamton University

Dr. Michael Lewis, Outside Examiner
Department of Computer Science, Binghamton University

ABSTRACT

The prosperity of the Internet has made it attractive to hackers and malicious attackers. Distributed attacks, such as: DDoS attacks and Internet worms have become major threats towards the network infrastructure. Collaborating existent single-point-deployed security applications over multi-domains for distributed defense is promising. Taking advantage of the small-world network model, a three-layered network modeling platform was developed for exploring behaviors of collaborative defense under the scope of a complex system. Using this platform, a comparison study between two major collaborative defense schemes was conducted. Their performance and effectiveness against signature-embedded worm attacks were evaluated accordingly.

Given the rapid evolution of attack methods and toolkits, software-based solutions to secure the network infrastructure have become overburdened. The performance gap between the execution speed of security software and the amount of data to be processed is ever widening. A common solution to close this performance gap is through hardware implementation of security functions. After a comprehensive survey on major reconfigurable hardware-based approaches application on network infrastructure security area, an optimized design of FPGA-based Power Spectral Density (PSD) data converter for online Shrew DDoS attack detection was proposed and prototyped. Combining an innovative component-reusable Auto-Correlation (AC) algorithm and the adapted $2N$ -point real-valued Discrete Fourier Transform (DFT) algorithm, a maximum reduction of 61.8% processing time from 27471.4 *us* to 10504.8

us was achieved. These efficient hardware realization enabled the implementation of this design to a Xilinx Virtex2 Pro FPGA.

The scalability issue against continuously expanding signature databases is another major impediment affecting hardware application for network intrusion detection. With the observation that signature patterns are constructed from combinations of a limited number of primary patterns, a two-stage decomposition approach was developed to solve this issue. The evaluation results show that a reduction in size of over 77% can be achieved on top of signature patterns extracted from the Snort rule database after decomposition.

ACKNOWLEDGEMENTS

It is impossible to make this dissertation done without tremendous help and encouragement from many people. Bearing an attitude of gratitude, my appreciation goes to my advisor, committee members, friends and families.

First and foremost, I would like to sincerely thank my advisor, Professor Yu Chen for his excellent advising and continuous supporting through these years. His knowledge and vision help in guiding me on the right track of research for success. His affability and patience help in creating a flexible research atmosphere for achievement. Without his efforts, time and energy devoted to me, it is unlikely I would achieve today's success.

Many thanks to Professor Douglas H. Summerville for his supporting on my academic research. I will never forget his help in correcting grammar errors on my publications and providing great suggestions.

Many thanks to Professor Aleksey S. Polunchenko for his knowledge in mathematics as well as our great collaboration for change-point detection project.

I would like to thank Professor Zhanpeng Jin for his advice, support and friendship.

I would also like to thank Professor Michael Lewis who agreed to be the Outside Examiner for my defense without hesitation.

Great appreciation is to Mr. Thomas Gaska for his proofreading of this dissertation and his families' warmhearted treatment.

Your time and energies, your comments and feedback all contribute to guar-

antee the quality of this dissertation.

Special thanks to Dr. Qing Wu, Dr. Qinru Qiu who provided me great help from both study and life, when I came to Binghamton University.

My deep thanks to Mr. Xunwei Wu and Ms. Dai Zhou, for their enlightenment and encouragement in my life.

In addition, thank you all my friends. You make my life joyful.

Last but not least, I would like to thank all my families for their love and support, from the bottom of my heart. Without you, I could not make it!

Contents

List of Tables	xi
List of Figures	xii
1 Introduction	1
2 Background and Related Work	9
2.1 Review of Network Security Situation	9
2.2 Network Infrastructure Security	11
2.3 Requirements for a Successful Security Application	12
3 A Comparison Study of Modeling Collaborative Strategies for Distributed Defense	14
3.1 Related Work	17
3.2 Small-World Network Based Modeling Platform	19
3.2.1 Small World Network	19
3.2.2 Structure of the Three-layered Modeling Platform	21
3.3 Internet Worm Attack and Defense	23
3.3.1 Modeling a Worm Attack	24
3.3.2 Modeling Defense Schemes	25
3.4 Experiments and Performance Evaluation	27
3.4.1 Simulation Setup	27
3.4.2 Experimental Results and Discussions	30
3.4.2.1 Simple SI Model	30
3.4.2.2 The SIR Model	32
3.5 Summary	38
4 A Survey on Reconfigurable Hardware Based Technologies for Network Intrusion Detection	39
4.1 Packet Classification	42
4.1.1 Existing Applications	44
4.1.2 Technology Analysis	45
4.2 Pattern Matching Application	48
4.2.1 Finite Automata (FA) Technique	50
4.2.1.1 Existing Applications	50
4.2.1.2 Technology Analysis	51

4.2.2	Content Addressable Memory (CAM) Technique	55
4.2.2.1	Existing Applications	55
4.2.2.2	Technology Analysis	57
4.2.3	Optimization for Scalability	60
4.2.3.1	Existing Applications	61
4.2.3.2	Technology Analysis	62
4.3	TCP Stream Preprocessing Application	67
4.3.1	Existing Applications	68
4.3.2	Technology Analysis	70
4.4	Internet Worms and DDoS Attack Detection and Containment	73
4.4.1	Existing Applications	75
4.4.2	Technology Analysis	78
4.4.2.1	Anomaly Detection	79
4.4.2.2	System Update	83
4.5	Discussion	85
4.6	Summary	87
5	PSD Data Converter for Anomaly Detection	88
5.1	System Architecture and Algorithms	90
5.1.1	System Architecture	90
5.1.2	Component-Reusable AC Algorithm	92
5.1.3	Adapted $2N$ -point Real-valued DFT algorithm	97
5.2	Implementation	100
5.2.1	Design Overview	100
5.2.2	Design Implementation	102
5.3	Experimental Evaluation	107
5.3.1	Experimental Setup	107
5.3.2	Function Verification	108
5.3.3	Simulation Comparison	110
5.3.4	Synthesis Analysis	112
5.4	Discussion	113
5.5	Summary	114
6	Pattern Decomposition Method for Signature-based Detection	115
6.1	Related Work	118
6.2	Decomposition of SNORT Patterns	121
6.2.1	Redundancies in SNORT Patterns Sets	121
6.2.2	Pattern Decomposition	124
6.3	Further Analysis and Verification	128
6.3.1	Signature Pattern Reconstruction	129
6.3.2	Statistical Verification	130
6.4	Discussion	133
6.4.1	Scalable Pattern Update	133
6.4.2	Dynamic Signature Matching	134
6.5	Summary	135

7 Conclusion	137
7.1 Contribution	137
7.2 Future Work	139
A Data Parsing of Real Network Trace	141
B Prototyping on NetFPGA	148
B.1 Development of a NetFPGA-Based IP Packet Counter	150
B.2 Real Prototyping on TCP Packet Counting with Chipscope	154
C Publications	163
Bibliography	166

List of Tables

4.1	Applicability of basic security applications.	77
5.1	Results comparison of AC-DFT process between theoretical expectation and simulation output.	109
5.2	Device utilization summary.	113
6.1	Summary of parameters of signature pattern set and primary pattern set in different versions.	131
A.1	Categorized statistic information of attempted destination IPs Vs. corresponding source IPs.	144
A.2	Categorized statistic information of attempted destination IPs Vs. corresponding source IP flows.	145

List of Figures

1.1	The structure of dissertation.	7
3.1	Centralized and decentralized collaboration.	15
3.2	Illustrations of logic networks.	20
3.3	An example of small-world based network.	21
3.4	Structure of three-layered network model.	22
3.5	Infection/defense modes.	24
3.6	Pure worm infections without containment.	31
3.7	Alarming for worm attack.	32
3.8	Trends of infected nodes with containment.	33
3.9	Trends of susceptible nodes with containment.	34
3.10	Overall defense efficiency.	35
3.11	Correlated infection rate under different alarming rates.	36
3.12	Correlated infection rate under different collaborative scales.	37
4.1	NFA/DFA for regular expressions.	53
4.2	Binary CAM operation in match mode with packet length =1.	58
4.3	TCAM operation in match mode with packet length =1.	59
4.4	Basic bloom filter operations.	63
4.5	Basic unit for pattern matching application with bloom filters.	64
4.6	(a). Un-partitioned graph. (b).Partitioned graph.	66
4.7	Basic models for TCP stream processing.	71
4.8	Block diagram for TCP stream preprocessing.	72
4.9	Block diagram for payload anomaly detection.	80
4.10	Block diagram for PSD based anomaly detection.	82
4.11	Block diagram for updatable system architecture.	83
5.1	Data conversion steps.	90
5.2	Convolution with reusable parts.	92
5.3	Convolution without reusable parts.	94
5.4	The impact of sequence length L replacement rate to the computational complexity of multiplication.	95
5.5	The trend of increment rate for multiplication.	96
5.6	The reduction trend of FFT workload.	100
5.7	Refined architectures for PSD data conversion.	101
5.8	Block diagram for autocorrelation and FFT process.	102

5.9	Block diagram for parameter generating process.	105
5.10	Block diagram for final results generating process.	106
5.11	Timing analysis of simulation results between the improved approach and the conventional approach.	111
6.1	An example of a Snort rule.	122
6.2	A Sample of Snort signature pattern.	122
6.3	Repetition of patterns.	123
6.4	Composition of primary patterns.	123
6.5	Variation of pattern entities.	126
6.6	Variation of pattern sizes.	127
6.7	Signature Pattern Reconstruction.	130
6.8	The growing trends of pattern entities between different versions. . .	132
7.1	Connecting the IP packet counter to proposed detection approaches. .	139
A.1	Monitored traffic variation during the spread of the Witty Worm. . .	142
A.2	Relations between the number of attempted destination IPs and their corresponding source IPs.	143
A.3	Relations between the number of attempted destination IPs and their corresponding source IP flows.	144
A.4	Traffic variation of selected source IP flows.	145
A.5	A close look of the outbreak of Witty Worms.	146
A.6	Traffic variation of the individual source IP flow.	147
B.1	The first generation NetFPGA 1G (4x1G) platform.	149
B.2	A block diagram of NetFPGA system.	150
B.3	The user data path of NetFPGA-based designs.	151
B.4	The interface for inter-module communication between modules. . . .	152
B.5	The architecture of NetFPGA-based IP packet counter.	153
B.6	The overall experiment environment.	154
B.7	ChipScope involved FPGA debugging.	155
B.8	Manipulating IP packet counter with ChipScope.	157
B.9	Set up software registers through a Perl script.	158
B.10	Chipscope monitoring under real running.	160
B.11	The framework of our NetFPGA-based network testbed.	161
B.12	Lab configuration of NetFPGA-based tutorial routers.	162

Chapter 1

Introduction

The growth and success of the Internet has made it fertile ground for malicious attackers and abusers. The number of Internet users grew from 361 million in 2000 to 1.8 billion in 2009, and then 2.8 billion in 2013 Q4 [82]. The number of mobile-connected devices will even exceed the human population on earth (about 7.6 billion) by the end of 2014 [49]. According to the report from the Information Security Forum (ISF), entitled Threat Horizon 2014 [147], cyber criminality increases as Malspace matures further; the cyber arms race may lead to a cyber cold war; with most battles occurring come online. Thus, the matter of network security is not only related to people's privacy, corporates' profit, but to the vitality of a nation's future.

The current Internet looks much different from its early appearance. It has evolved from the widespread interconnection of separate physical networks. The original design focused on efficient data transmission in an environment where network resources were precious and user groups were limited to scientists and engineers with implicit trust in one another. Hence, security was not an important design requirement, especially to the protection of the network infrastructure. After decades of evolutionary development, the Internet has become a polluted place, infested with viruses and malware [78]. Today, billions of computers are connected in a complex global web supporting a wide-range of consumers. Corporate and government users have expectations of a reliable communication infrastructure. However, the main in-

frastructure of the Internet still remains the same end-to-end paradigm as when it was initiated in the 1970's [15].

Network infrastructure security is one of the major branches of network security. It mainly refers to the protection of hardware and software resources that enable network connectivity, communication, operations and management. Malicious activities, such as: Distributed Denial-of-Service (DDoS) attacks, turbo worms, e-mail spam, phishing, and viruses are considered as significant threats against the Quality-of-Service(QoS) that Internet Service Providers (ISPs) promise to their subscribers. Ideally, a comprehensive security solution is expected to cover the entire network infrastructure fabric. However, it is unfeasible due to scalability and complexity of the network in the real world. A more practical approach is deploying security applications onto a limited number of high profile network nodes, such as: ISPs, government agencies, financial institutions, and health care facilities, etc. The increase in both number and sophistication of attacks against the network infrastructure necessitates more robust security solutions.

To contribute to more robust security solution, this research first focused on collaborative defense for resilient protection. The prevalence of distributed attack over the Internet could make single-point defense insufficient for proper reaction. Collaboration of multiple "single-points" for co-defense is a promising way to solve this problem. To the best of our knowledge at the start of this research, most reported efforts were application dependent. There was no comprehensive study describing general characteristics of collaborative defense at the abstract level. Taking the advantage of small-world modeling, a three-layered network model for this study was developed. Centralized and decentralized collaborative schemes were evaluated on top of this model for fighting against Internet worms.

The study of collaborative defense reveals the importance of information sharing which boosts peers' performance for protection. However, it is at the premise that

either the coordinator or certain peer(s) manage to perform Triple-D operation (i.e.: Detect, Distill and Dispatch valuable information) successfully. As one of critical steps, detection serves as the gateway process. Effective and efficient detection is the foundation for the development of a defensive network infrastructure. Being enlightened from this perspective, this another's research moved one step further towards network intrusion detection —a sub-field of network infrastructure security.

Literally, network intrusion detection refers to monitoring network activities for malicious activities or policy violations and producing reports for further processing. Any security application that carries this function can be considered as a Network Intrusion Detection System(NIDS). A NIDS is usually deployed at a strategic point or gateway of a network to monitor traffic through all devices on the network. Unlike a network firewall which selectively blocks ingress/egress traffic for protection, a NIDS is able to evaluate traffic to discern suspected intrusions from normal traffic. There are several variations of NIDS, such as the Network Intrusion Prevention System (NIPS), the Host-based Intrusion Detection System (HIDS), and the Protocol-based Intrusion Detection System (PIDS) [142]. SNORT [140], one of the most widely used NIDS solutions, is an example of a software-based, light-weight NIDS.

Up to now, most NIDSs are software based. The flexibility of software implementation makes these NIDSs resilient against the evolvement of malicious activities. However, the performance gap between new NIDS requirements and software-only realizations is widening. The increasing number and sophistication of attacks, the performance limitations of sequential software execution and the increase in network throughput all contribute to the widening of this gap. In fact, many successful software-based approaches have been inhibited by the inability of implementing many security measures at the required performance levels [29]. Taking these facts into account, hardware-based acceleration is a natural way for performance improvement. Guaranteeing real-time processing of sophisticated security functions is a major con-

cern for this shift.

Meanwhile, it is still of equal importance to maintain system flexibility provided by general purpose computing power. Typical NIDS hardware-based implementations usually incorporated a few specific accelerations and in many cases still require the assistance of software for proper operation. Since security threats are constantly evolving, defense systems also require both static and dynamic update mechanisms. Thus, reconfiguration is as important to a hardware implementation as programmability is to software. Indeed, the design boundary between hardware and software is blurred at this point [78]. Field Programmable Gate Array (FPGA) devices have been widely adopted because they feature both the flexibility of software and the high performance of hardware [67]. The FPGA is a suitable and popular hardware platform for many network security applications, including protocol wrapper [20], packet classification [153], and intrusion detection [19, 103, 162]. The emergence of the NetFPGA platform [106] is a good example showing the demand of integrating FPGAs into network applications.

In addition to performance improvement, the inclusion of special purpose hardware also provides opportunity for architectural and algorithmic innovation. The fundamental performance difference between hardware and software solutions lies in their dissimilar execution paradigms. Hardware-based custom data path and control implementation allows executing many operations in parallel, while the execution of pure software implementation is serial or with limited parallelism [55]. As a result, high efficient execution is also an attractive research topic in hardware design area. Thus, unique features from the side of hardware involvement can bring detection process to a higher level. This was the major motivation behind this dissertation.

For a better understanding of reconfigurable hardware technologies application for network intrusion detection, a comprehensive survey was conducted. According to different phases of process for traffic analysis, the author's survey categorizes diverse

applications into the following groups: packet header classification, payload examination, TCP stream preprocess, general DDoS attack detection, and Internet Worm containment. Combining brief descriptions with intensive case-studies, a clear picture of mainstream technologies applied in this field is presented. Anomaly-based and signature-based detections complement each other for detecting known and unknown malicious activities, respectively. They lay down the foundation of network intrusion detection.

Anomaly-based detection focuses on tracking critical network characteristics in real time and generates an alarm if a suspicious event or trend is detected that could indicate the presence of a threat. Large-scale examples of such characteristics include traffic volume, bandwidth use and protocol use [123]. Thus, anomaly-based detection is particularly helpful in detecting new zero-day attacks.

Through investigation of diverse network attacks, this survey found that one particular advanced Distributed Denial-of-Service (DDoS) attack, called Shrew DDoS attack, can conceal itself into normal traffic. Its stealth characteristics makes conventional detection approaches inefficient. Extending detection cross spectral domains, a countermeasure of using Power Spectral Density (PSD) analysis was identified for more detailed investigation. By exploring energy distribution of network traffic in the frequency domain, this selected approach offers high fidelity for revealing malicious activities.

However, operation of online PSD analysis necessitates real-time PSD data conversion. The key is to perform computing the computation intensive Fourier Transform, which makes pure software-based applications unaffordable when operating under high speed network environments. A FPGA based accelerator has been developed for this conversion. It is based on an innovative component-reusable Auto-Correlation (AC) algorithm and the adapted $2N$ -point real-valued Discrete Fourier Transform (DFT) algorithm. Further optimization is achieved through the explo-

ration of algorithm characteristics and hardware parallelism. Evaluation results from both simulation and synthesis are provided. The overall design can be realized in a Xilinx Virtex2 Pro or later version's FPGA board. In order to connect to the real world for practice, a corresponding NetFPGA-based IP packet counter was also implemented to collect packet statistics.

In general, anomaly-based detection is sensitive to variation of network traffic. Its decision is usually made through the statistical analysis, which leaves relative large false positive. Rather than providing details, this type of detection is more likely to provide early warnings. Signature-based detection complements anomaly-based detection by checking against the monitored packets with known signatures or attributes of maliciousness for discovery. If a threat is detected, there is less false positive, but at the cost of a lag for signature matching. Therefore, keeping signature databases updated is critical.

The quick evolution in malicious attacks produces a continuously expanding signature database that becomes a major impediment for hardware-based implementations. Although hardware-based solutions possess high performance computing capability, their performance is very sensitive to the availability of hardware resources. High scalability is a major concern in development of a hardware-based implementation. Before jumping into a specific hardware implementation, a deeper investigation into these signature patterns enables identification of potentials for design innovation. One key observation for this investigation is that intrusion operation sequences should be accepted by computer systems, in order for manipulation. Possible acceptable sequences are limited in practice, which is determined by a finite set of allowable activities. Any signature pattern can correspond to one or a combination of multiple basic activities. To verify this thought as well as to prepare for further hardware design, a two-stage decomposition strategy was used. Intensive experiments on SNORT Rule database were also conducted.

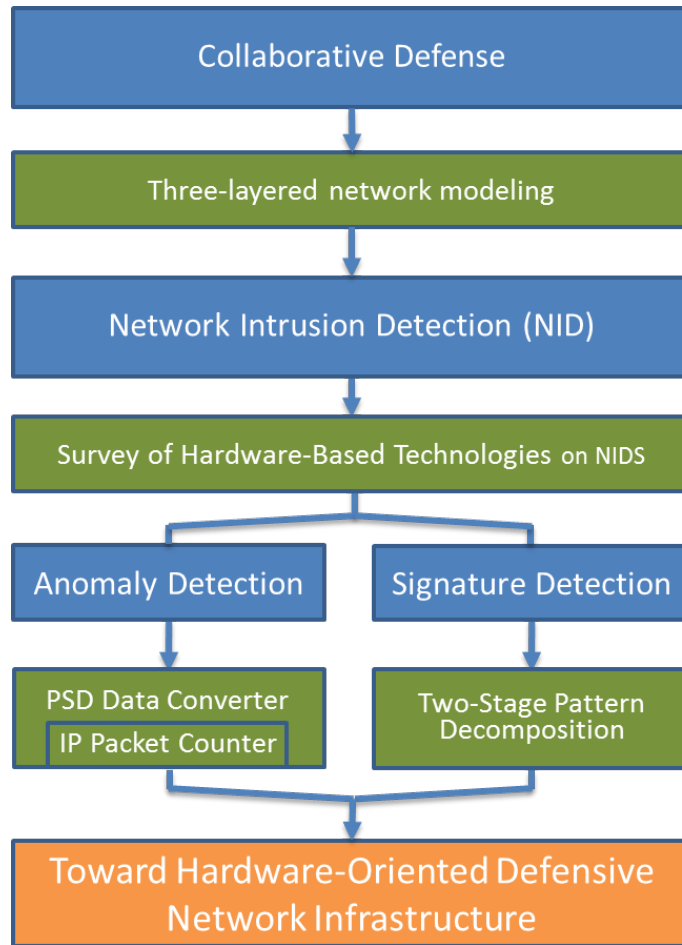


Figure 1.1. The structure of dissertation.

Starting from efforts in collaborative defense to network intrusion detection, Figure 1.1 outlines the structure of this dissertation. The blue blocks represent different sub-topics along the way, while the green blocks represent the corresponding achievements that have been obtained by this work. Each of the efforts have contributed to the goal of developing hardware-oriented defensive network infrastructure which is represented by the orange block at the end. The arrowed lines indicate the flow of the overall study.

The rest of this dissertation is organized as follows. From a brief review of network security situation, Chapter 2 gives the definition of network infrastructure security as well as requirements for achieving a successful security solution. Chapter

3 proposes this author's effort of exploring collaborative strategies for co-defense. A detailed survey of reconfigurable hardware-based technologies application on network infrastructure security is provided in Chapter 4, which is followed by a description of innovations in the two major types of network intrusion detection. Chapter 5 presents the development of an embedded PSD data converter for anomaly detection of a Shrew DDoS attack, and Chapter 6 describes a scalable, two-stage signature decomposition method that leverages a hardware implementation of pattern matching. Finally, Chapter 7 concludes this dissertation. Peripheral research on network trace parsing and prototyping on NetFPGA are discussed in Appendix A and B. Appendix C lists the author's publications related to this dissertation.

Chapter 2

Background and Related Work

2.1 Review of Network Security Situation

With its exponential growth, the Internet has successfully penetrated into almost all aspects of human society, changing people's lifestyles and social behaviors. Today, network security is considered as an essential element in protecting one of the most important national infrastructures. Based on the declining trend of Average Losses Per Respondent (ALPR), financial and intellectual investments in network security have begun to pay off [135]. However, the growth of the Internet as well as the evolution of threats keeps the security situation of the network at a severe threat level, which motivates continuing investment [81].

According to annual surveys of computer crime and security from Computer Security Institute (CSI) [135–138], the total losses due to cyber-crime increase every year; but the results of ALPR, the most useful way to evaluate these losses, has trended down. For 194 responses reported, the total losses were \$66,930,950 in 2007, up from \$52,494,290 (for 313 respondents) in 2006. After ALPR hit the highest score of \$3,149,000 in 2001, it gradually has fallen down. This set of data reflects both the severe security situation, as well as people's efforts in this battle.

Targeted attacks have become a trend in network security. A targeted attack

is a malware attack aimed exclusively at an organization or organizations within a sector or market [135]. Around 20% of respondents of the CSI surveys suffered this kind of security incident. Such narrowly targeted attacks are becoming more popular than ever [174].

For example, Denial-of-Service (DoS) attacks continuously contribute new threats to network security. Since 2000, DoS attacks have grown rapidly and have been one of the major threats to the availability and reliability of network-based services. This type of attack was listed as causing the second highest total cyber crime cost in 2004. Although the percentage of losses caused by DoS attacks has been decreasing in recent years, the total losses are still increasing. This is due to financial losses incurred for every minute that a site is down [135]. Distributed DoS (DDoS) attacks have evolved from DoS attacks and can cause even more significant damage in the DDoS attack, the attacker compromises multiple machines and recruits them into a zombie army. Exploring the asymmetry between powerful Botnets [53] and individual networks, DDoS attacks are launched against a specific victim from these zombies [131]. The DDoS attacks against Yahoo!, eBay, Amazon.com and other popular websites in February 2000 revealed the vulnerability of even very well equipped networks [32].

In addition, the trend of malicious threats has been moving towards massively distributed Internet worms and spyware. When combined with DoS attacks, the financial damage can be profoundly high. For example, CodeRed, a well known worm that included a built-in DoS attack payload, infected more than 250,000 systems in just 9 hours on July 19, 2001 [30]; and the numerous varieties of the MyDoom worm carried time-triggered DoS attack programs as their payload, causing devastating results in 2004 [69].

2.2 Network Infrastructure Security

Securing network infrastructure has become a high priority task due to its underlying effects on data protection, e-commerce and even national security [71]. A reliable network is expected to provide both information security and infrastructure security. Information security is based on the principle of information theory, and primarily focuses on the integrity and confidentiality of data [31]. Infrastructure security, on the other hand, focuses on the availability and reliability of network resources that support information transmission [1]. In the past decades, the same importance and urgency of the latter has been widely recognized. In fact, infrastructure and information security are complementary. A robust network should be secured in both dimensions for reliable operation.

This work focuses on securing pertinent components of network infrastructure, such as: DNS servers, routers, buffers and end-hosts. Since these components take responsibility for information gathering, exchange and distribution, they are high profile objects valuable for attack. Similar to the formidable challenge of securing all possible points of entry for attacks against a nation, it is neither practical nor necessary to respond everywhere in network infrastructure [87]. As long as these pertinent components are secured, the entire infrastructure survives. An effective network infrastructure security policy should well protect critical components inside the network, and prevent them from being attacked. Therefore, the availability, reliability and stability of network services can be achieved [37].

Few security issues were taken into account and standardized as protocols in the original design of the Internet. This intrinsic deficiency has given attackers many opportunities. Other sources of vulnerability in the network infrastructure include implementation deficiencies, misconfiguration, and selection of topologies and protocols. Although the situation has been improved due to the emergence of security

protocols such as IPSec (IP security) [89], it takes time to become widely adopted. In addition, the exponential growth of the Internet has resulted in a heterogeneous network infrastructure that often cause slow adoption of protocol solutions. As a result, security strategies are overdue to protect the network infrastructure. In order to provide Quality-of-Service(QoS) to its users, a robust network infrastructure is imperative.

Many infrastructure security solutions are based on network traffic analysis [28, 159]. Therefore, any useful application for traffic analysis can be considered as a security application in the context of network infrastructure security. TCP flow processing, packet classification, pattern matching, shrew attack defense and worm containment represent the major application categories focusing on specific parts of network infrastructure security [96, 100]. Network Intrusion Detection Systems (NIDS) and Network Intrusion Prevention Systems (NIPS), on the other hand, focus on the security of the entire infrastructure [102].

2.3 Requirements for a Successful Security Application

After surveying current security applications related to network infrastructure, one can observe that a successful solution usually carries the following features:

- *Real-Time processing.* It is essential for an effective protection mechanism to process data at line-speed with affordable cost. All traffic is subjected for inspection in a timely manner, and alerts are generated accurately when abnormal situations occur.
- *Updatable.* Constantly evolving malicious attacks require security solutions to be adaptive to retain effectiveness. The update could be knowledge-based infor-

mation, such as: statistic information, signature patterns, that security analysis depends on, or even the system itself. Updating an application is often more practical than replacing it.

- *Scalable*. Scalability is another critical concern for practical deployment. Many reported approaches work well on a small research network at laboratory level, but performance deteriorates rapidly when being deployed to a larger scale network, such as: campus level's network. The main reason for scalability limitations is often exponential increase of system complexity.

Chapter 3

A Comparison Study of Modeling Collaborative Strategies for Distributed Defense

In recent years, efficient defense against distributed attacks has been a hot topic in the network security community. Instead of establishing brand-new, dedicated systems, collaborating widely deployed, single-point network security applications for co-defense is a more feasible approach. Through collaboration, a security shield that covers infrastructure of multiple network domains can be built without significant modification. Besides keeping track of base functionality, collaboration offers individual security applications wider views of dynamic situations around which may otherwise not be observed. It improves the resilience and confidence of participating security applications to handle sophisticated cases with optimized strategies.

Existing collaborative schemes for distributed defense can be classified into either centralized or decentralized categories. The most significant characteristic of a centralized scheme lies in the use of a coordination daemon as shown in Figure 3.1(a). The coordination daemon can be resident on a powerful multi-role server, or a dedicated server. Most likely, this server is located at one of participating net-

work domains. Its major responsibilities include information collection, processing, analyzing, and distribution from/to individual nodes. This server may also conduct decision-making; it either makes decisions for all the participating nodes, or provides suggestions to those nodes for reference, depending on the specific mechanism for collaboration. The main advantages of centralized schemes are high accuracy and efficiency. Considering the overhead and scalability, centralized schemes have a distinct boundary for collaboration.

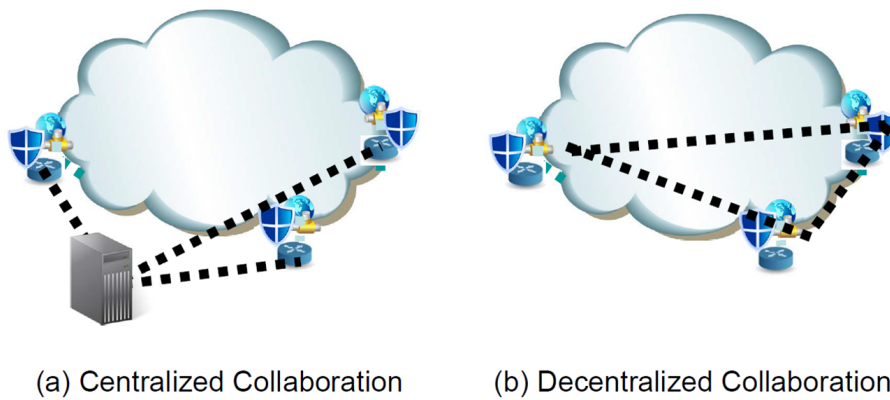


Figure 3.1. Centralized and decentralized collaboration.

In contrast, the decentralized scheme is very flexible. It behaves similar to the manner of Peer-to-Peer (P2P) networks. This is due to the fact that most of decentralized schemes are developed on top of P2P network protocols. The P2P collaborative architecture gives decentralized schemes good scalability. Theoretically, any network node featuring compatible collaboration protocols can participate, so that the boundary of the covered network can be flexible. Rather than having a dedicated server in a centralized scheme, each participating node takes responsibility for collaboration, as shown in Figure 3.1(b), which brings more flexibility for self-management. Obviously, the cost for application is relatively low, since it does not require any modification in network beyond the installation of the software.

It is important to have a comprehensive understanding on behaviors of variant collaborative schemes. A deeper insight is critical for designers of network security

system to adopt proper strategies that can match their requirements best for the application. However, there are only few reported efforts that studied collaborative behaviors of different schemes at the abstract level. Instead, most research has rather focused on application-specific solutions.

One main challenge in conducting such a behavioral study lies in the lack of methodology that is capable of presenting networks at the abstract level. In practice, many technical and/or non-technical issues make this task more complicated. Fortunately, this challenge can be handled through modeling technology. With the help of modeling, a virtual environment that mimics a simplified world can be set up [70]. After abstracting and scaling-down original problems, it is feasible to conduct further study on substantial behaviors.

In this chapter, a three-layered network modeling platform has been developed for the establishment of such an abstract environment. The Internet layer at the bottom and the overlay network layer in the middle take advantage of a small-world network model for setup, while the application layer on the top focuses on the description of defense schemes. Based on this platform, a preliminary behavior study comparing different defense schemes has been conducted. The single-point defense scheme and its corresponding centralized and decentralized collaborative schemes are modeled. Through the adjustment of modeling parameters, different scenarios have been created for the evaluation of their impacts on network infrastructure security when facing signature-embedded worm attacks at the abstract level.

The rest of this chapter is organized as follows: Section 3.1 provides a brief review on the reported efforts in distributed defense schemes for network infrastructure security. The developed three-layered modeling platform is introduced in Section 3.2. Then Section 3.3 focuses on the description of our modeled worm-based attack and defense. Section 3.4 expresses the operation details of how the simulation experiments have been conducted. In addition, this author's results for the overall performance

evaluation of applying multi-domain collaboration for distributed defense are presented. Section 3.5 summarizes this chapter.

3.1 Related Work

In network security systems, it is common that multiple end-hosts work collaboratively against attacks at the abstract level [6,86,119,155]. A blacklist is exchanged among the potential victims to mitigate the threat. Usually a two-stage operation is conducted for distributed defense, which includes local detection and global collaboration.

There are two popular collaboration schemes. Schnackenberg *et al.* proposed a centralized coordinative scheme called CITRA [143] for network intrusion detection in 2001. A central coordinator responds for coordinating countermeasures based on a complete view of the network. Janakiraman *et al.* [83] introduced a decentralized defense scheme for network intrusion prevention. Information is shared among trusted peers to guard the network against intrusion. The subscription-based group communication is conducted over a P2P architecture, which brings excellent scalability.

For collaborative detection at the victim end, some more advanced techniques have been developed. Beyond focusing on certain detectable facts at the same domain, the emergence of cross-class detection [139] and multi-domain alert correlation [168] are able to link these detectable facts to some deliberate essentials for further analysis. With the help of cross-class detection, hosts can monitor and share information of different attacks. Meanwhile, the multi-domain alert correlation can even aggregate alerts that possess common feature values. Instead of only focusing on traffic volume, researchers have extended the anomaly detection to frequency domain, in which the traffic distribution has been considered as random signals and its energy distribution in different frequency bands has been analyzed [33,40,99].

Beyond collaboration at the victim end, deploying network security systems into the network fabric increases the effectiveness of the defensive system. Gamer *et al.* [66] extended their research to achieve a coordinated collaboration among independent systems for anomaly-based attack detection. Their approach combines an in-network deployment of neighboring detection systems with information exchange. Working in a self-organized manner, however, each network node makes decisions independently.

Taking advantage of the P2P network, researchers have attempted to address the major challenges in large scale collaboration: the scalability and avoidance of central point of failure [183]. They have merged multi-dimensional correlation for collaborative intrusion detection [168], and developed a self-protecting and self-healing collaborative intrusion detection architecture for the trace-back of fast-flux phishing domains [184].

A Distributed Change-point Detection (DCD) scheme has been proposed to detect DDoS attacks over multiple network domains [42, 43]. Distributed information is collected through Change Aggregation Tree (CAT) for centralized analysis and decision-making [38]. Another collaborative approach was designed to detect and stop DDoS attacks at the intermediate network [182]. To achieve this purpose, detection nodes are deployed at both victim and source ends for collaborative detection [185]. In a more ambitious approach based on the DefCOM [117] scheme, the collaborative nodes are deployed all over the network. In addition to the victim end and the source end, the intermediate network is also included [126].

The Internet can be considered as a complex system. All network activities, including attack and defense can be treated as subsets of it. From this perspective, it is practical to study the behaviors of network security activities using complex system models. The earliest effort using complex system for the modeling of distributed network defense schemes was proposed in 2001 [65]. However, not much similar

research has been reported since then, to the best of our knowledge. Different from their proposal of only describing a preliminary agent-based model without concrete experiments, this chapter presents this author's efforts in a specific evaluation based on a more deliberate three-layered network model.

3.2 Small-World Network Based Modeling Platform

Many network security applications are based on traffic monitoring. The more traffic information is obtained, the more confident security applications can be in their event detection. In order to achieve the best performance, it is preferable that security applications are deployed at the gateway of the intended networks. In practice, it has been a trend to integrate traffic monitoring functions into routers for a simple solution. Today, many advanced commercial-available routers are security enhanced. Not only are software applications implemented, hardware based accelerations are also embedded for advanced security improvement. Essentially, a distributed collaborative security defense is set up to enable the cooperation of their corresponding network devices. An upper-level application chooses the countermeasure, while the lower-level agent supports its execution. Special channels are reserved for this collaboration, in order to avoid the interference with normal traffic. In this section, taking advantage of small-world network theory, an abstract three-layer platform is built for this modeling study.

3.2.1 Small World Network

The real Internet is a scale-free network [3]. A scale-free network is a network whose degree distribution approximately follows the power law [13]. Most nodes in a scale-free network have only one or two links, while only a few nodes have a large

numbers of links. These small portion numbers of nodes act as hubs responding for the connection of the whole network.

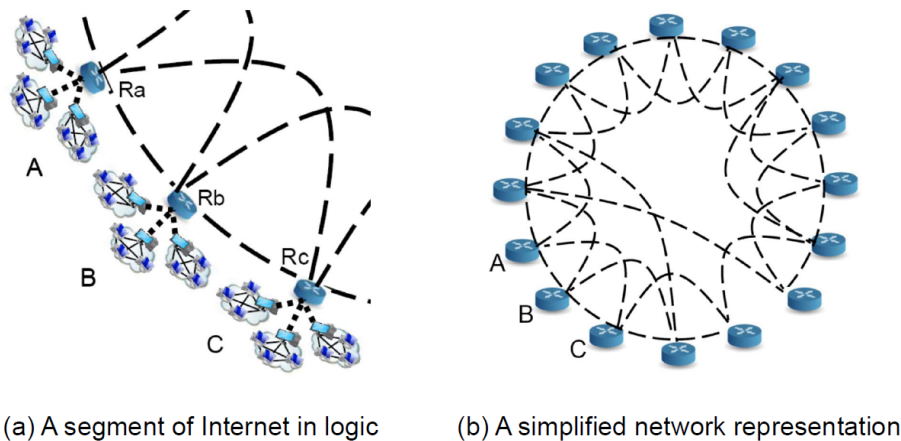


Figure 3.2. Illustrations of logic networks.

Figure 3.2(a) represents a segment of Internet in logic. With the connection to router R_a , R_b , and R_c respectively, end hosts belonging to different network domain A , B , and C are able to communicate with each other. In addition, multiple logic links among R_a , R_b , and R_c make communication more efficient by choosing optimal paths. For example, directly forwarding packets from domain A to C through the shortest links between R_a and R_c without passing R_b . Therefore, routers R_a , R_b , and R_c play critical roles as gateway hubs bringing local network A , B and C together to form a larger network.

The theory of small-world network describes a simplified scale-free network. A small-world network is a network being mostly local-connected but with a few global connections. In fact, many real-world networks can be described well using small-world network models, such as cells [84], social networks [171], World-Wide Web [77] and the Internet [2]. Watts and Strogatz model is the most famous small-world network model. It is a random graph generation model that produces graphs with short average path lengths and high clustering [172]. It is the foundation of our modeling platform.

Figure 3.2(b) illustrates a classic small-world network. This modeled network consists of a finite numbers of nodes. Each node represents a network domain. Let's assume that each network domain has only one outlet to the Internet. Actually, it is true in many cases. The dashed lines represent logic links among different network domains over the Internet at the abstract level. The graph in Figure 3.3 shows an example generated with the specific approach that we employed to construct such a small-world network. The whole network contains 50 nodes. Each node is connected to the 2 nearest neighbors and has the probability of 0.175 to add another edge. Small-world networks set up the basic topology for our intended network layers.

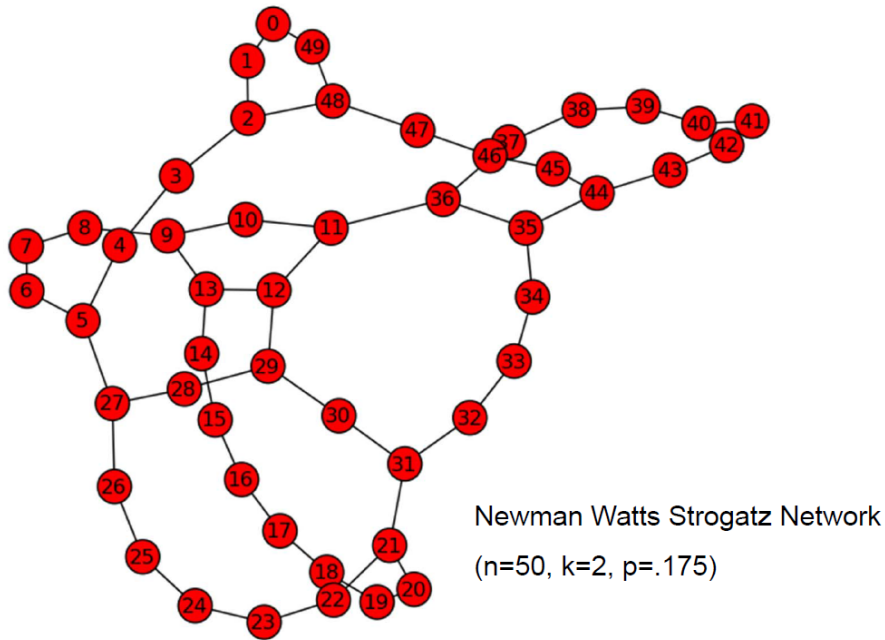


Figure 3.3. An example of small-world based network.

3.2.2 Structure of the Three-layered Modeling Platform

As demonstrated in Figure 3.4, the platform that was developed in this research for the modeling of collaborative schemes consists of three layers. From the bottom to the top, they are the Internet layer, overlay-network layer and application layer, respectively. The development of the bottom two layers is inspired by

the Watts and Strogatz small-world network model. The Internet layer models an abstract Internet environment in general, while the overlay network layer models a dedicating environment for the running of different collaborative schemes. Finally, the application layer focuses on the description of defense schemes.

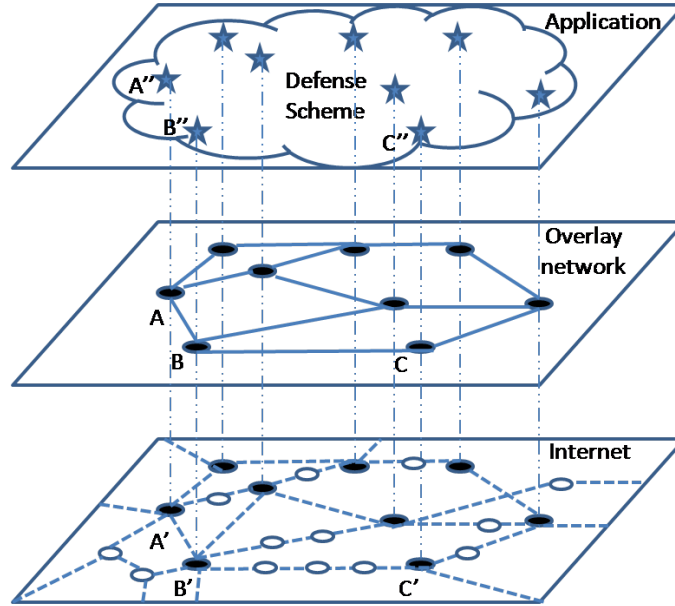


Figure 3.4. Structure of three-layered network model.

As shown in Figure 3.4, each solid node in the bottom Internet layer represents a network domain participating collaborative defense and the hollow nodes are network domains that do not participate in the collaboration. The dashed lines represent the physical network topology. The middle overlay network layer consists of those network domains that participate in collaborative defense. The solid lines in this layer represent the logic connections among these nodes.

In this model, link weight is adopted as the metric of distance between nodes. One hop is the minimal distance between any neighboring network domains. For example, in Figure 3.4, nodes A and B , nodes B and C are adjacent respectively, so that the weight of $Link_{ab}$ and $Link_{bc}$ are one. Meanwhile, the weight of $Link_{ac}$ is two.

However, their link weights may not remain the same when corresponding nodes are mapped to the bottom Internet layer. Though Node A' and B' still appears

to be adjacent, Node B' and C' are four hops away from each others, so the weight of $Link_{B'C'}$ is four. As illustrated by Figure 3.4, there are three other network domains on the path from B' to C' .

In the Internet layer, the six-degree-of-separation theory [116] points out that the average width of a large scale network is six. Specific to the Internet, earlier research [42] has statistically verified that more than 99% of network domains in the Internet can be reached within six hops. In this work, the weight assigned to the associated links follows normal distribution.

The top layer is the application layer, which is a conceptual layer where we define defense schemes. This layer focuses on the behavior description of participant network domains in an abstract manner. As shown in Figure 3.4, stars represent security applications deployed on top of corresponding network domains. The cloud generalizes the defense schemes organizing these applications for reaction. Three types of defense schemes are described in this layer: the single-point defense scheme, centralized and decentralized collaborative defense schemes. All three defense schemes are applied to the same constellation of security applications, but with different concentration on network coverage. The first single-point scheme concentrates on the protection of individual network domains. Each security applications work independently. The other two collaborative schemes, instead, concentrate on the protection of a wide range network area. Through multi-domain collaboration, valuable information is shared among individual security applications for the improvement of overall performance.

3.3 Internet Worm Attack and Defense

Based on above developed modeling platform, a preliminary study comparing three different defense schemes has been conducted to investigate their performance

against a typical Internet worm attack. This section describes the following two parts: modeling a worm attack, and modeling corresponding defense schemes.

3.3.1 Modeling a Worm Attack

Featuring self-duplication and automatic propagation, Internet worms are truly autonomous during attack. They are able to spread over the network, breaking into end hosts and replicating. It is extremely challenging to prevent Zero-day worms. In addition, worms themselves are good carriers for other malicious attacks. Some sophisticated worm attacks intend to propagate stealthily so as to survive for further actions, such as remote control of infected hosts for launching DDoS attacks.

Various strategies have been adopted to achieve fast propagation, such as exploring security holes, and increasing scanning rate with different scanning schemes [56, 170]. The propagation of most Internet worms shows certain similarities. After a short period of modest increase, the number of infected network domains presents an exponential growing. Once reaching the maximum infection, the corresponding curve trends to be flat, if there is no effective way for containment. At that time, a wave of worm attacks enters a saturate state.

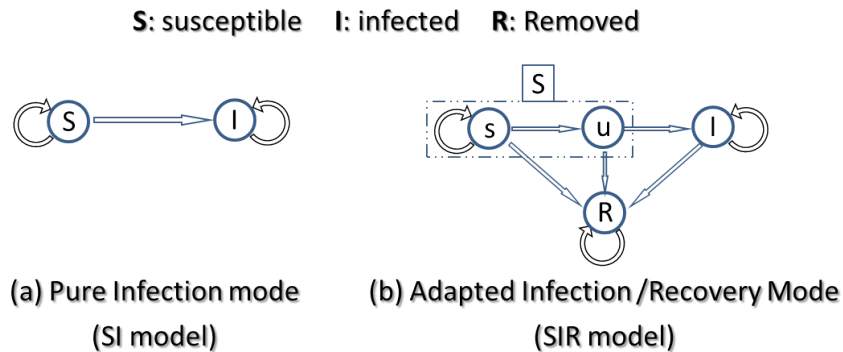


Figure 3.5. Infection/defense modes.

The worm attack was modeled as follows. Assume there is only one type of worm in the whole process. The infection of network domains follows the simple

classical epidemic model (SI model) as shown in Figure 3.5(a). Participant nodes in the interested space have two states: “Susceptible (S)” and “Infected (I)”. They are all initialized to be susceptible to the attack. One of the participant nodes is randomly selected as the first infected node. Most likely, worms propagate to all the neighbor nodes from the current infected node. This propagation follows the network topology at the Internet layer as shown in Figure 3.4. For a small chance, it may propagate to any nodes directly.

The SI model only considers the attack under pure infection mode. With the engagement of all kinds of defense efforts, worms may be detected and contained. Consequently, network domains may be immune from this attack, regardless of their current status either in a susceptible or infected state. The adapted SIR model depicts such an infection/recovery scenario as shown in Figure 3.5(b). A “Removed (R)” state is introduced. Once entering the “Removed” state, the current network domain is recovered and becomes invulnerable to this worm attack. The “Susceptible (S)” state in the original SI model was also adjusted to “susceptible (s)” and “under-attacking (u)” sub-states for the clarification of different situations when a susceptible network domain enters “Removed” state.

3.3.2 Modeling Defense Schemes

The adaptive SIR model also describes the basic behaviors of individual security applications. We assume the immunity of network domains to worm attacks results from the reaction of security applications. Each security application plays as an agent for reaction. Although individual defense behaviors may vary, the collective behaviors determine the overall effectiveness.

Single-point defense scheme is inefficient facing fast propagation, wide spread and stealth Internet worms. For example, considering the security applications modeled in the application layer of Figure 3.4, network domains A' , B' and C' adopts

different security applications A'' , B'' and C'' , respectively. While B'' has detected the signature of a worm on network B' , neither A'' nor C'' has detected that network A' has been infected. Since all the applications work individually, the successful detection at B'' does not imply that other network domains can get any benefit. Their collective behavior shows that the overall reaction efficiency of security applications running under the single-point defense scheme is low. It highly depends on the performance of each individual.

In contrast, collaborative defense can significantly improve the effectiveness. Benefiting the earlier alarm and assistance from B'' , C'' can start and optimize its defense in advance, and A'' will realize what happened and start to contain the maliciousness so as to minimize the negative impact.

The on-the-fly collaboration runs seamlessly without human intervention. As a result, corresponding network domains are all saved from the attack. The overall reaction effectiveness of participants running under the collaborative scheme is higher. Their performance is correlated and significantly impacted by the first agent reacting to the attack. Theoretically, the larger the numbers of network domains involved and the wider the area they span for co-defense, the higher the probability would be for prompt detection.

Due to the different collaborative strategies, centralized and decentralized schemes have been proposed as the improvement on top of the single-point defense scheme. The individual behaviors of the participant agents remain the same, but obviously their collective behaviors are changed. In this study, we will focus on different defense behaviors with or without collaboration.

The collaborative defense of participant agents running under the decentralized scheme behaves similar to social network activities. Besides defending individually, they interact with peers for information sharing and decision making. This type of collaboration is flat; no agent is dominative. We assume that all agents only collabo-

rate with their neighbors and all neighbors have the same significance to each other. On one hand, each agent still makes decision individually, but takes the reference from its neighbors under consideration, such as issuing an alarm for the worm attack. On the other hand, each agent acts as a relay that efficiently passes the proper information to others, such as spreading the issued worm alarm to its peers. Individual agents are highly flexible in collaboration.

In the centralized scheme, all the collaborative activities run under a root-leaves structure. A centralizer acts as the root that is in charge of the whole collaboration. It may be located in any of the participating network domains. This centralizer has reliable communication with all the participants. Security applications act as leaves, collecting and pre-screening useful information to the root. Through the analysis of gathered information, corresponding feedback is returned from the root to all the leaves. Obviously, the overall efficiency of agents running under this scheme is more consistent than what the decentralized scheme can achieve.

3.4 Experiments and Performance Evaluation

This section presents simulation results and performance evaluation. The simulation is described in detail, including basic assumptions, parameters, and attack-defense operation. Through the analysis of simulation results, performance of both centralized, decentralized collaboration schemes for distributed defense and the single-point scheme for individual defense have been evaluated.

3.4.1 Simulation Setup

In simulation, operation of attack and defense is relatively independent from each other. According to the platform shown in Figure 3.4, the simulation of a worm attack is conducted in the Internet layer. The propagation of worms spreads through

the paths defined in this layer. Their targets are those susceptible interested network domains (solid nodes). The propagation does not stop as long as any susceptible node has not been compromised in the interested network space.

The simulation of defense is carried out in the overlay network layer. The whole constellation of security applications applied to this layer act as agents. They take the defense schemes from the upper layer to protect the corresponding network domains mapped in the lower layer. The collaboration among agents follows the topology defined in this layer. The defense countermeasure will not stop until either all the solid nodes in the Internet layer are alarmed or all the nodes are immune from the worm, depending on the specific setup for simulation.

The appearance of the first infected node triggers the worm attack towards the interested network area. This node is randomly selected from the interested network domains in the Internet layer at the beginning of the simulation. After that, worms propagate following the described modeling topology.

Meanwhile, the location of the centralizer is also randomly assigned to one of the participant network domains. Through the association with its related defense agent, it records all the shortest paths from the related agent to all the other agents as defined in the overlay network layer. The structure for centralized collaboration is set up during the initialization of simulation.

The first alarm for worm detection from a defense agent triggers the whole defense reaction. This triggering event is associated with the progress of worm propagation. With the propagation of worms through the network space, the probability of being detected also grows. The worm packets are detectable once corresponding signatures have been identified [35].

Basically, the simulation process follows the adaptive SIR model as show in Figure 3.5(b). Those event-triggered activities can be manageable well under a Finite State Machine (FSM) mechanism. The simulation is executed with a discrete time

scale. The execution time of each activity is scaled to one or multiple unit time slots. The complete worm propagation procedure consists of three phases: online probe, data transmission, and local infection. For simplicity, assume that the time delay for one node infection is one unit time slot.

Consider that transferring 100k data in 100M/bps takes 1 ms, while infecting a node takes a few seconds. It is obvious that the infection time is dominative in a simple worm propagation scenario. From the perspective of defenders, the time for alarm spreading is expected to be short due to the utilization of reserved channel for collaboration. However, the overhead that resulted from the collaboration among different agents is non-trivial.

To describe the variable security vulnerabilities that network domains possess, we randomly assigned the resistant time of each network domain to worm attack from 1 to 3 unit time slots, with normal distribution. It means that the most vulnerable network domains would be infected in one unit time, and the least vulnerable network domains would also be infected in 3 time slots, if there is no available defense.

At the defender's side, assume the time delay of information exchange is one unit time for collaboration between adjacent agents in decentralized scheme, or between agents and centralizer in centralized scheme. The collaborative activities include alarm spreading, and advanced knowledge sharing which includes the updated signatures. It is also assumed that the execution time for all the local activities is one unit time, including the alarm issuing, the advanced knowledge issuing, relying determining and knowledge updating. Except the original agent issuing the alarm or other advanced knowledge, all the other agents have to receive the updated knowledge and finish the update processing in order to contain worms in their network domains.

Although it may be disputable that worm signatures are generated and ready for spread just one unit time later after the worm alarm is issued in our simulation, it does not affect the relative defending trends after the knowledge update processing is

finished in local agents. The only difference lies on the start point for reaction along the time axle during simulation.

3.4.2 Experimental Results and Discussions

According to the above description, extensive simulation experiments have been conducted on worm attack-and-defense atop the modeling platform. The sample of the first set simulation consists of 200 security applications in the overlay layer. Mapping to the Internet layer, they correspond to 200 intended network domains. Taking advantage of Watts-Strogatz network function, a small-world network environment has been generated for the Internet layer with $n = 200$, $k = 2$, $p = 0.4$. The average distance between any two adjacent nodes is 5.514, which is acceptable for the number representation of un-intended network domains in between them along the way. Since the nodes in overlay network are usually tightly connected in logic, their average distance between any pair of adjacent nodes should be shorter. Thus, the link probability was increased for setting up the topology of overlay network layer, as $n = 200$, $k = 2$, $p = 0.6$.

3.4.2.1 Simple SI Model

Figure 3.6 demonstrates the modeled worm propagation without any defense. The X -axis represents time. The Y -axis represents the number of network domains (nodes). The red line represents the increasing trend of worm infection over all of the network domains. The line that consists of small dots records the total number of infected nodes. The green line represents the decreasing of susceptible nodes due to the increase of infected nodes. The blue spots at the bottom indicate the number of newly infected nodes in each unit time.

In order to present a clear view of their variation, a log scale was used for the representation of values in the Y -axis. The number of newly infected nodes stays small

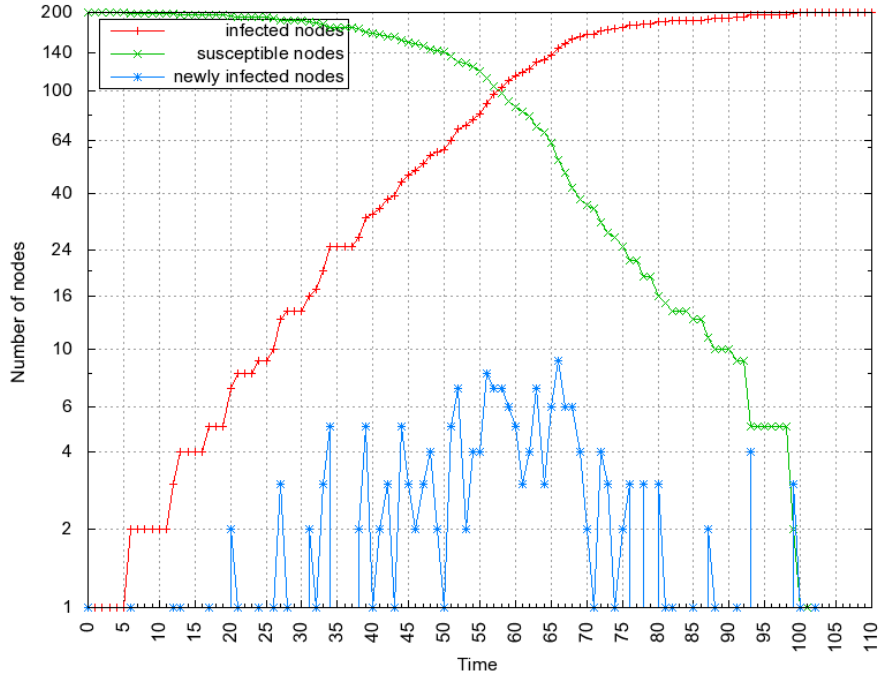


Figure 3.6. Pure worm infections without containment.

at both ends, but it is large in the middle. This is because the exponential increase of worm propagation usually happens in the middle with respect to the whole progress.

Figure 3.7 shows the alarm spreading after the worm attack is detected. The purple line across through the end-to-end from the left-bottom to the right-up represents the referred number of infected nodes. It is identical to the red infection line in Figure 3.6. The shape is different because the Y -axis is normal scaled, which represents the number of alarmed nodes. In this simulation, it is assumed that when 60% of the network domains have been infected, the worm is detected and the first alarm is generated. In Figure 3.7, the blue cubic box on the purple curve marks this point. Referring to X -axis, it is time of issuing the first alarm from that agent. The red line represents the agents working under single-point defense scheme, the blue and green lines represent agents working under centralized and decentralized defense schemes, respectively.

As expected, the red line is almost flat during the whole simulation period since

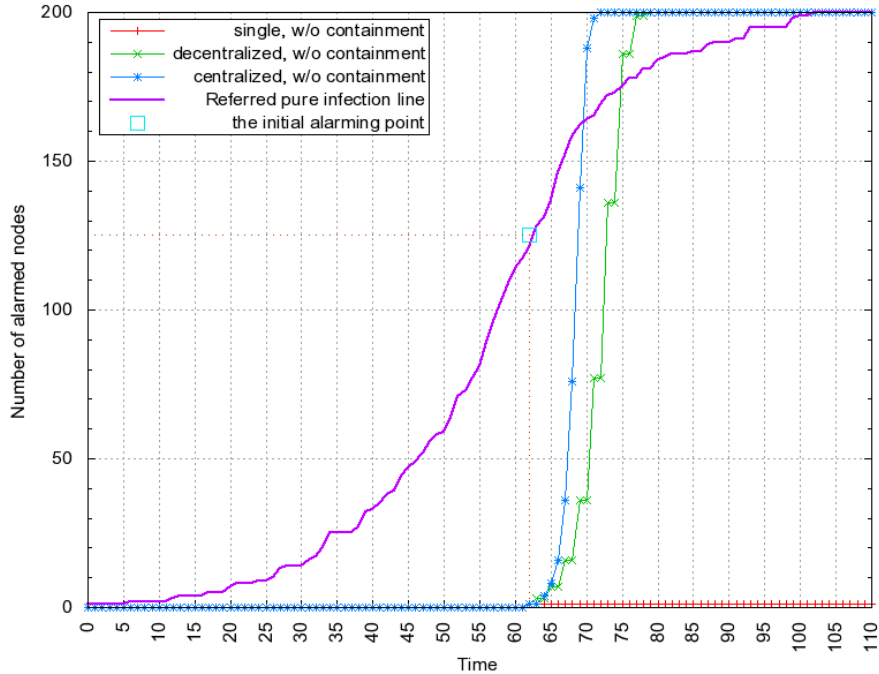


Figure 3.7. Alarming for worm attack.

none of the other peers are expected to be able to share the alarm. For centralized and decentralized schemes, the alarm quickly spreads to all the defense agents through the topology built in the overlay network layer. This topology models the paths for collaboration. It is obvious that centralized scheme is more efficient for alarm spreading with the same set of collaborative nodes and the same network topology.

3.4.2.2 The SIR Model

For further insight of the impact of different defense schemes, we simulated the defeat scenario. Once a worm attack has been detected and an alarm has been issued, the security agent continues to update its knowledge base and spread the newly generated signatures to peers for worm containment. Through sharing signature information, other agents can effectively prevent the attack.

Figure 3.8 presents the scenario of worm containment. The curves reflect the number of infected nodes along the time. The referred pure infection line and the

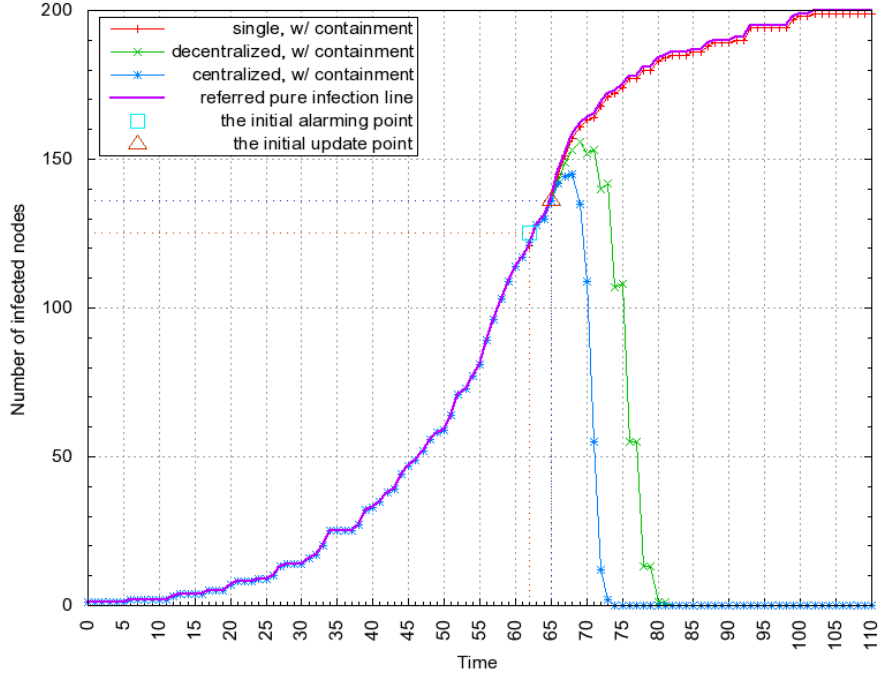


Figure 3.8. Trends of infected nodes with containment.

initial alarming point remain the same with previous examples for consistency. The difference is that the initial update point is introduced, which represents the time point when the first agent has finished knowledge updating and is ready for worm containment. The trend of red line that represents the defense running under single-point scheme almost stays the same with the purple referred pure infection line. Since only one node is alarmed and becomes immune from the attack, all other nodes are still vulnerable and get infected.

Two collaborative defense schemes show much better containing efficiency. As observed, the trends of blue and the green lines turn down sharply after a small delay. The blue line represents the number of infected nodes under the centralized scheme, while the green line represents that under the decentralized scheme, respectively. Finally, the former touches the ground at the 74th tick, and the latter touches down at the 82nd tick in this case.

The decreasing trend of susceptible nodes also supports observation that col-

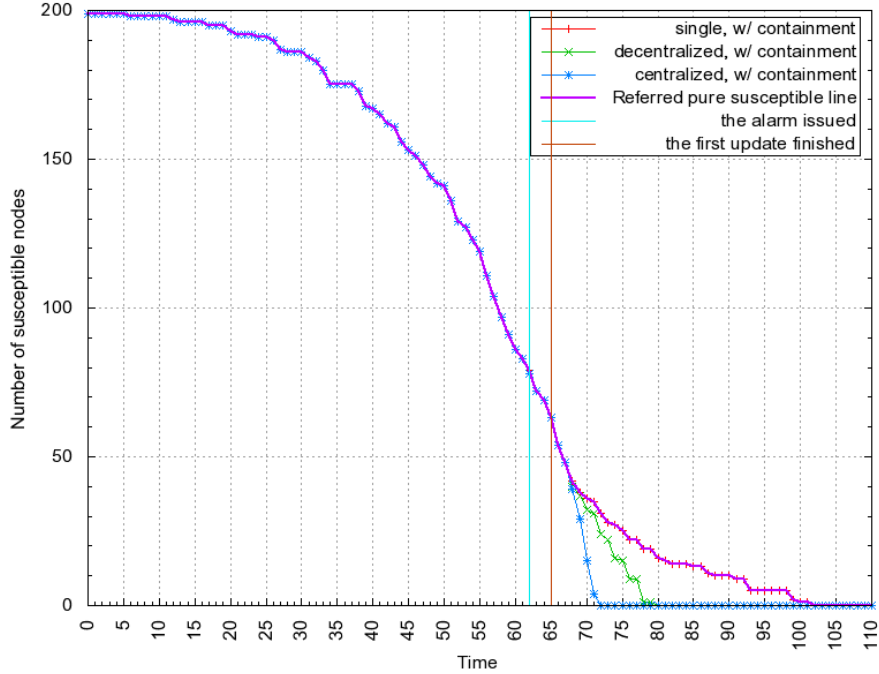


Figure 3.9. Trends of susceptible nodes with containment.

laborative schemes are more efficient in defense. As illustrated in Figure 3.9, the first and second vertical lines from the left to the right indicate the time that the initial alarm and update are set in the previous example, respectively. The trend lines regarding different defense schemes are overlapped most of time. Two lines with respect to the centralized and decentralized schemes divert shortly after they pass the initial update line, and quickly diminish to zero at the 74th and the 82nd tick. This quick diminishment is due to the efficient signature update of both collaborative schemes. Before being attacked by worms, they have already been immune.

To reveal the overall efficiency of different defense schemes against worm attack, we further explored the ratio between the immune and infected nodes in our simulation. Figure 3.10 gives an alternative view for this exploration. The immunity ratio is defined as Eq. (3.1):

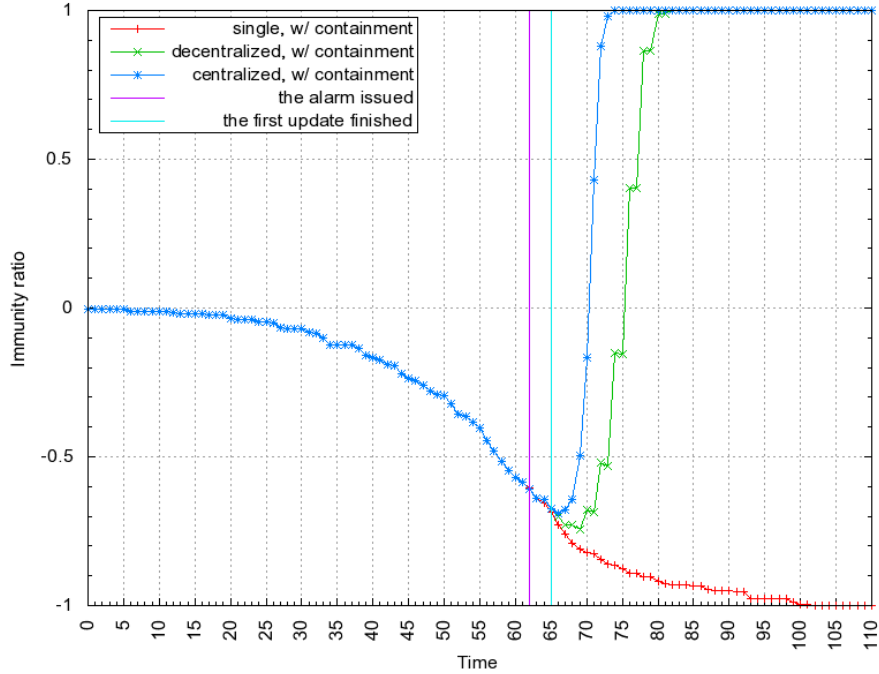


Figure 3.10. Overall defense efficiency.

$$R_{immunity} = \frac{\text{number of immuned nodes} - \text{number of infected nodes}}{\text{total number of nodes}} \quad (3.1)$$

At the beginning of the simulation, none of node are infected nor do they have immunity from the worm, so the immunity ratio is zero. Without effective containing measures, the propagating worms quickly pulls the immunity ratio down to negative and finally locks at the fully infected status, i.e.: -1 , or -0.995 in terms of single-point defense scheme. However, this ratio can also be pulled up with the efforts of proper countermeasures. As observed in Figure 3.10, the trend lines of two collaborative schemes quickly rise up and finally enter full immune status at 1. In fact, this trend really depends on two facts: the efficiency of collaboration and the delay of detection.

The second set of simulation exhibits the impact of variable alarming rates to the overall worm defense, as shown in Figure 3.11. The X-axis lists the range of alarming rates from 0 to 0.99, with an interval of 0.05. They represent ratios of

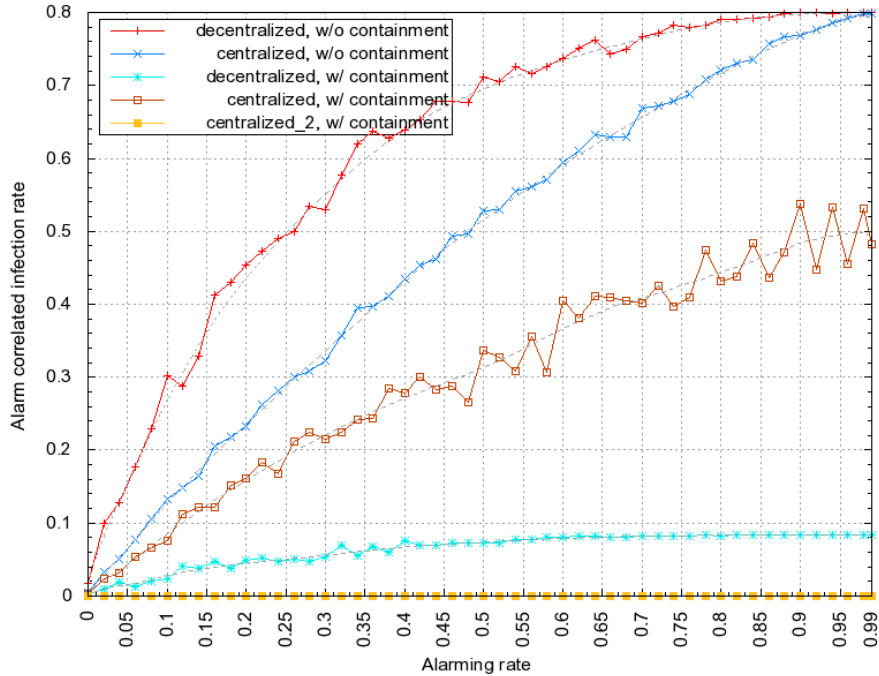


Figure 3.11. Correlated infection rate under different alarming rates.

infected network nodes out of the overall, when the first alarm is issued. The Y-axis represents the infection rate when the alarm reaches all the nodes. For consistency, all experimental configurations remain the same as previous examples. In order to achieve a fair output, every data point in Figure 3.11 is an average of 5 experiments. Intuitively, the earlier the alarm is issued, the lower the correlated infection rate would be. Two lines from the top down, the infection rate of decentralized scheme is higher than that of centralized scheme under the same alarming rate. It is obvious that both infection rates trend to increase when the alarming rate moves from 0 to 0.99.

As to evaluation with containment involved, both lines trend to be flatter. Through collaboration, nodes are able to resist attack or even recover from infection, so as to limit the increase of infection. The lower decentralized scheme line may cause illusion of its outperforming the higher centralized scheme line. On the contrary, the centralized scheme still performs better. The higher infection rate is because the speed

of containment or the fact that immunity does not catch up with the propagation of alarms. Instead, the propagation delay of decentralized scheme makes this difference not obvious, so as to make less infection rate as shown. From time perspective, the infection rate under centralized scheme has already been pulled down to zero at that moment, as shown by the bottom yellow line in this plot.

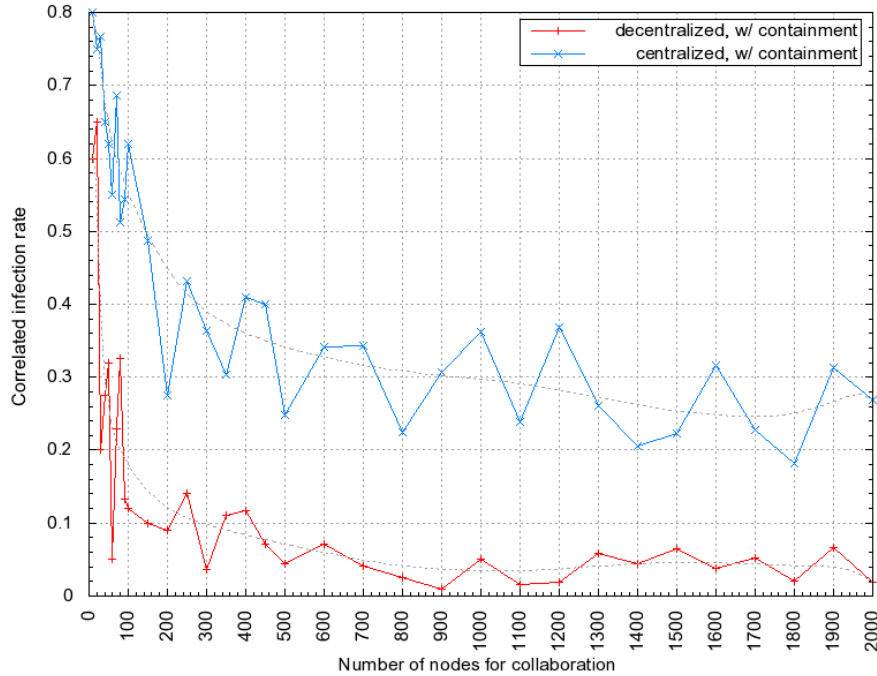


Figure 3.12. Correlated infection rate under different collaborative scales.

The scalability of collaborative defense schemes has also been evaluated. The configuration remains the same, except with different numbers of collaborative nodes for operation each time. Figure 3.12 presents the simulation results. The number of nodes was from 10 to 2000. According to the simulation result, although the vibration of infected rate is still obvious, they trend to flatten when the collaboration scale is greater than 200 nodes. The correlated infection rate of centralized scheme is higher than that of the decentralized scheme for the same reason as mentioned above.

3.5 Summary

This chapter described efforts in a comparison study of collaborative defense for worm containment. Most reported research is for application-dependent and problem-specific solutions. The work provides a new insight of collaborative strategies at a higher abstract level. On top of the developed three-layer network model, experimental results support that both centralized and decentralized collaborative schemes can effectively improve security performance, in comparison with single-point defense schemes. The key of this improvement is from the power of information sharing for Triple-D operation, that is Detect, Distill and Dispatch valuable information among peers for co-defense. As one of critical steps, detection serves as the gateway process. Effective and efficient detection is the foundation of all the following processes, so as for the development of a defensive network infrastructure.

Chapter 4

A Survey on Reconfigurable Hardware Based Technologies for Network Intrusion Detection

Hardware implementation of security solutions has become a trend as the gap between network data rates and off-the-shelf processor computing power continues to increase. With rapid development of technology, hardware devices play increasingly important roles in network security. Among the reported research, most hardware implementations have focused on construction of platforms that perform raw data collection or analysis. While these platforms may not provide complete security solutions, they exhibit great potential [78].

The migration towards hardware-based implementation is motivated by two major issues that prevent solely software-based security applications from moving forward. First, the performance of purely software-based applications is usually inadequate for practical deployment. For example, the saturation analysis throughput of Snort [140] was only 137 Mbps [52] using the standard configuration. Other systems, like Bro [127] and WebSTAT [167], were not able to handle data rates higher than 100Mbps [145]. Second, processors have become overburdened by the band-

width expansion of network connections. High-end processors such as Intel XEON Woodcrest (3.0 GHz) and Intel Itanium Montecito (1.6GHz) were unable to maintain high enough throughput while running multiple Regular Expression Engines based Snort IDS [118]. Thus, it is necessary to offload network applications to dedicated hardware [64] and free up the host processor [118].

In addition to powerful computing capability, hardware devices naturally support parallel execution of operations. Compared with mostly sequential execution of software-based implementations, this unique feature leads to more efficient data-parallelism and multi-stage processing, which is unbeatable by current software-based platforms.

In contrast to software implementations, application-oriented and highly parallel design paradigms make hardware implementations superior in terms of performance. For example, relative to TCP Stream Reassembly and State Tracking, an Application Specific Integrated Circuit (ASIC) developed at Georgia Tech., was able to analyze a single TCP flow at 3.2Gbps in 2002 [122]. A FPGA-based TCP-processor developed by Open Network Laboratory (ONL) at Washington University was capable of monitoring 8 million bidirectional TCP flows at OC-48 (2.5Gbps) data rate in 2004 [145].

ASIC-based devices not only possess the advantage of high performance, but have the potential for low unit price. However, their substantial non-recurring engineering investment can only be justified when ASIC devices achieve sufficiently high-volume production. Unfortunately, this may not be applicable to network security applications. Constant evolving standards and requirements make it unfeasible to fabricate ASIC-based network security applications at such a high volume [78]. Moreover, custom ASICs offer little or no reconfigurability, which is another reason that ASICs have not been widely applied in the network security area.

Reconfigurability is an essential requirement for the success of network security

applications. The emergence of reconfigurable hardware enables greater flexibility in the design of hardware-based security solutions [57, 181]. Briefly speaking, a reconfigurable device can be considered as a hybrid hardware/software platform, so as to enable design up to date [78]. FPGAs are the most representative reconfigurable hardware devices.

A Field-Programmable Gate Array (FPGA) is a type of general-purpose, multi-level programmable logic device that can be programmed by end users [165]. At the physical level, logic blocks and programmable interconnections compose the main structure of a FPGA. A logic block usually contains a 4-input look-up table (LUT) [22] and a flip flop for basic logic operations, while programmable interconnections between blocks allow users to implement multi-level logic. At the design level, a logic circuit diagram or a high level hardware description language (HDL) [115], such as VHDL or Verilog, is employed for development of specific functions. From the industrial side, it is always vital to keep short design cycle for a quality product. Because of their unique feature of flexibility in hardware development, FPGAs were quickly adopted for prototyping of new logic designs shortly after they were invented in the mid 1980s [22].

While the performance and size of FPGAs limited their application in the early days, advancements in density and speed have resulted in narrowing the performance gap between FPGAs and ASICs. This enables FPGAs not only to serve as fast prototyping tools but also to become primary components in embedded systems [175]. Current FPGAs share the performance advantage of ASICs, as they can implement parallel logic functions in hardware. They also share certain flexibility of embedded network processors in that they can be dynamically reconfigured [75].

Combining the performance advantages of ASICs with software-like re-programmability, current FPGA technologies enable new infrastructure security applications. With billions of computers now connected to the Internet, it is unfeasible to

rely on every user being diligent in keeping their security provisions up to date. A more practical way to maintain the efficiency and reliability of system security is to apply security applications at a much smaller number of aggregation points [78]. Featuring powerful functionality but compact size, FPGA-based security implementations are one of the most suitable candidates for this purpose. They can be deployed anywhere within the network infrastructure for considerable improvement in security.

In practice, most network infrastructure security applications follow the strategy of being deployed adjacent to routers and work as security reinforcement to them. In this way, the original configuration of network routers can be mostly preserved, reducing sharing of valuable resources and minimizing the negative impact to already heavily burdened routers. This strategy reduces the chance of malicious traffic touching routers. Also, the balance between intrusion analyzing and resolving can be maintained [32]. While it is not possible to detect and eradicate all malicious traffic at few aggregation points, the robustness of the system can usually be guaranteed at the cost of acceptable security risk.

According to the different phases for traffic analysis, a comprehensive survey is presented in this chapter. The phases include: packet header classification, payload examination, TCP stream pre-process, general DDoS attack detection, and Internet Worm containment.

4.1 Packet Classification

The network infrastructure faces serious security challenges due to the exponential growth of the Internet. In practice, it is unfeasible to have an entire network infrastructure protected due to economic, social, or political reasons. It is difficult to avoid the presence of malicious users inside the network, especially the Internet. Existing infrastructure security applications are usually focused on the protection

of a specific network area, such as a campus-network or an enterprise-network. As a result, most network infrastructure security applications adopt a passive defense strategy. The basic rationale is *I cannot stop you, but I can prevent you*.

Timely detection of malicious attacks is one of the essential functions for network security applications. Without effective detection, subsequent countermeasures are useless. To take full advantage of the high parallelism supported by hardware devices, signature-based detection is the most fundamental and spontaneous strategy adopted. The signature can include source/destination address, port numbers, protocols being applied, and patterns of content or any combination of these. By comparing specific strings contained in incoming packets to known signatures the detection system is able to identify previously known malicious attacks.

For network infrastructure security, packet inspection is the first approach for malicious activity detection. A packet consists of two kinds of data: control data in the packet header and user data in the payload [92]. Packet header fields are essentially constant in length and appear at fixed locations in the packet, while packet payloads can be variable in length with no fixed format. Conventionally, packet classification focuses on the processing of packet headers, while deep packet inspection focuses on processing of the payload. A combination of the two can be applied for a complete packet matching approach [153].

Packet classification is an important technology for network infrastructure security. From early firewalls to recent high-performance routers and sophisticated Intrusion Detection Systems (IDS), classification plays an indispensable role. Research in packet classification has achieved substantial progress in recent years. The reported research work in this area has been well summarized and categorized into four basic types by Taylor [163]. Therefore, instead of trying to repeat the big picture, this survey focuses more on recent emerging hardware based techniques.

4.1.1 Existing Applications

To the best of our knowledge, the first hardware implementation of packet classification was developed in 2002. Called a flow classifier [180], it is a module employing a hybrid hardware-software architecture performing straightforward operations. A 16-bit flow identity, which is generated through a hash calculation of a packet's five-tuple value, is assigned to each packet [161]. With these identities, individual packets are classified to different flows for further parallel flow monitoring. The design was implemented using a Xilinx VirtexII XC2V8000 FPGA board, and reported results indicate that it was sufficient for 10 Gbps traffic rates.

Song and Lockwood developed a hardware-based approach for network intrusion detection in 2005 [153]. Combining the advantages of Ternary Content Addressable Memory (TCAM) [160, 178] and the Bit Vector (BV) algorithm [9, 94], the BV-TCAM architecture was introduced. The approach is based on the observation that while TCAM structure is efficient for direct data-lookup with prefix or exact values, it is not suitable for those that fall within a range. The complementary Bit Vector algorithm, however, is well-suited for this task. In this manner, this architecture eliminates the requirement for prefix expansion or port range lookups. With a limited embedded TCAM, packet classification can be easily implemented in currently available FPGAs. This design was prototyped in a Xilinx XCV2000E based FPX platform [104]. The evaluation results show that this application is good enough to sustain at least OC48 traffic rate (2.5Gbps).

A more recent achievement in hardware-based packet classification was reported in 2007 [109]. A Gigabit packet filter was tailored for implementation on FPGAs. Instead of choosing other advanced search algorithms, the architecture adopts a linear-search based algorithm for packet classification. The powerful hardware allows the linear-search based algorithm to achieve high performance under multi-parallel

operation mode. Moreover, a linear search is the best choice when using the internal memory blocks of FPGA for storage [109]. This pipelined packet-filter architecture was implemented on a Virtex-4 FX-12 FPGA and was demonstrated to filter network traffic at layers 2 and 3 with a throughput of 1 Gbps. Evaluated with a system frequency at 125 MHz, it only took 2,300 ns for a filter to make a decision of accepting or dropping a packet.

4.1.2 Technology Analysis

In general, packet classification refers to the operation of categorizing different packets into equivalent classes based on their header information [9]. These equivalent classes, known as flows, are grouped by certain rules that include matching the source/destination addresses or ports of the packets, or the protocols being applied to them. A corresponding action is associated with each rule to indicate the subsequent processing, such as forwarding, copying, or dropping. The rules are stored in a database, one rule for each flow type. When a packet arrives, at least one matched rule should be found in the database, allowing further processing to be conducted. If more than one rule is matched by a packet, an arbiter makes a decision according to a predefined policy, usually the longest pattern match. Among these processes, the most important step is how to perform effective rule matching.

Intensive research on packet classification has been carried out and many algorithms and architectures have been developed for this purpose. Taylor's survey in 2004 [163] summarized the major packet classification techniques. He framed each technique as employing one or more of four high-level approaches: Exhaustive Search, Decision Tree, Decomposition, and Tuple Space. Our analysis follows the same categorization and is complementary to Taylor's work. While Taylor's survey focuses on the description of techniques, we focus on the analysis of hardware-based application of these techniques.

Exhaustive search is intuitively the most straightforward approach to find a matching rule by examining all the rules in database. The basic linear search looks for filter rules sequentially until the first rule matches. The matching priority of the search is usually implicit. It is easy to modify the linear search to run in parallel operation mode, and a fully parallelized search can be achieved by using a Content Addressable Memory (CAM) [68] or similar data-addressable memory-structure. Ternary Content Addressable Memory (TCAM) [178] is commonly used, as it can find a matching rule in a constant time interval.

Although CAM-based approaches are popular and practical methods for the implementation of packet classification, they have two major drawbacks. First, the hardware complexity results in storage inefficiency, high power consumption, and limited scalability for long input-key searching. The second issue is the inefficient representation of filters with port ranges. The emergence of Extend-TCAM (ET-CAM) [160] improves the performance, however, the cost of the E-TCAM approach doubles in terms of gate count [109].

Decision-tree based approaches require pre-construction of a decision tree based rule-databases, for which bit-keys obtained from packet header fields are used to traverse the tree from root to leaf. The searching time highly depends on the length of search key. The Bit Vector algorithm [94] is a classic tree-based search algorithm. More sophisticated tree-based algorithms introduce the concept of 'cut'. A 'cut' in multi-dimensional space is isomorphic to a branch in a decision tree [163]. These branch decisions in cutting algorithms are more complicated than single bit decision in a bit-vector, but the principles are the same. The BV-TCAM [146, 153] approach is a hybrid of the Bit-Vector algorithm and TCAM structures. It combines the advantages of both for performance improvement. However, the search of a decision tree inherits a serial nature precluding it from fully parallel implementation.

Decomposition achieves performance improvement through the reduction of

search complexity. Parallel operation can be performed by decomposing multiple-field searches into multiple instances of single-field searches. In this manner the total search time can be reduced. This approach leverages the common case in which a packet rarely matches multiples rules [73]. After searching, a mechanism is needed to collect the disjoint results for final matching evaluation. In general, decomposition based algorithms are known to have high memory requirements [109]. For instance, the Distributed Cross-producing of Field Labels (DCFL) approach provides higher throughput, but at the cost of exponential memory requirements. For a set of N filters containing d fields each, the size of the cross-product table could be $O(N^d)$ [164]. An improved bit-vector scheme called Aggregated Bit Vector (ABV) was developed to overcome this issue. By searching a constant number of memory words in each field instead of examining all the leaves, it reduces memory access from N to $\log_A N$ [9].

Unlike decision-tree based approaches that perform the rule-search by exploring relations among rules, tuple-space based approaches can quickly narrow down the scope of a multiple field search by partitioning the rule-set into similar rules.

A tuple defines a series of specified bit numbers in each rule field [163]. It can be considered as a hash-like value to check whether a packet matches specific rules [109]. The motivation of using tuples stems from the fact that the number of distinct tuples is much less than the number of rules in the rule set [161]. The more similar the rules are, the more they can be grouped into tuples. After scope-narrowing, the complexity of a search is reduced. Since the number of tuples depends on the number of characters, the search time is predictable and usually short.

While each technique has its own set of suitable applications, the brute-force linear search is the best for hardware implementation mainly because of its compatible features with hardware devices. Brute-force based linear search algorithms are usually considered inefficient and as a result are cautiously used in software-based applications. However, the simple search mechanism allows for simpler implementation

while high parallelism enables optimization in hardware-based applications. Modern high-performance hardware devices provide powerful functionality that can greatly accelerate processing speed. In addition, parallel processing can be realized without much overhead.

Linear search employs common RAM architectures for sequential rule database search. Since incorporation of RAM-based memory is relatively common within FPGAs, efficient utilization can be achieved. On the contrary, CAM-based approaches are much difficult for efficient hardware implementation on FPGAs due to complex structure of CAM, though theoretically they are able to achieve highly parallel search. The decision tree and decomposition techniques require memory structures to store and access the decision tree efficiently. So far, efficiently implementing decision tree in hardware is still an open problem [109].

Packet inspection consists of packet classification on header information and deep packet inspection on payload trunk. Indeed, many search or match techniques are orthogonal to each other.

4.2 Pattern Matching Application

With the emergence of application level network attacks, inspection of packet headers alone is no longer sufficient [46]. Deep packet inspection has been introduced to reinforce network security and is dedicated to the task of payload inspection. The inspection of packet headers is relatively easy due to the explicit format specified by protocols. However, the inspection of payload contents is more challenging since the payload contents can be any format that is determined by applications. Therefore, it is more complex and expensive to perform deep packet inspection at payload level.

Pattern matching is the most popular approach for payload inspection [101]. It is one of the fundamental functions for anomaly detection and has been widely

applied for security purposes in network infrastructure, including Firewalls and Intrusion Detection Systems (IDS). The major task of pattern matching is straightforward; incoming packets are compared with a large number of patterns (or signatures), with subsequent processing based on the matched pattern [51]. In order for effective comparison, it is essential to update pattern database frequently, due to constant evolution of threats.

The performance of pattern matching is usually evaluated by two metrics: throughput and scalability [85]. Indeed, the speed of pattern searching determines the acceptable processing throughput, since all meaningful operations should be performed in real time [10]. Scalability evaluates how well a design fits to a real implementation. In practice, available resources are always limited. A design with higher processing throughput, but at the cost of significantly increased resource utilization, is considered to possess poor scalability. In terms of hardware pattern matching applications, better scalability implies that more patterns could be accommodated for a given amount of memory while the Quality-of-Service (QoS) of pattern matching operations is maintained. The speed of pattern matching is improved by accommodating more patterns on chip, which results in improved overall performance.

The introduction of reconfigurable hardware, such as FPGA devices, enables great strides for pattern matching applications providing powerful processing capability that software-based solutions cannot match. Techniques specialized for hardware implementation of pattern matching have been developed [18, 48, 103, 124]. In the following sub-sections, four typical hardware-based pattern matching techniques will be discussed: Finite Automata (FA), Content Addressable Memory (CAM), bloom-filter, and min-cut partition.

4.2.1 Finite Automata (FA) Technique

Finite Automata (FA) describes a class of models of computation that are characterized as having a finite number of states [95]. This concept has been widely applied and is prevalent in the digital logic design area. As it applies to pattern matching, the result of a FA processed input string is either accepted or rejected. A successful matching occurs when the string of input characters match the labeled patterns, or, more precisely, the regular expressions, on any path of FA that leads from the initial state to the final state. Currently, there are two types of FA approaches reported for hardware implementation: Non-deterministic Finite Automata (NFA) and Deterministic Finite Automata (DFA).

4.2.1.1 Existing Applications

In recent years, several Non-deterministic Finite Automata (NFA) implementations have been reported. Sidhu and Prasanna [149] mapped the NFA logic for regular expression onto a Xilinx Virtex FPGA and the Self-Reconfigurable Gate Array (SRGA) to perform fast pattern matching in 2001. The proposed approach takes $O(n + m)$ time and $O(n^2)$ space to find matches to a regular expression of length n in m long text. Using this method, Hutchings and Franklin compiled patterns of the open-source NIDS system Snort using JHDL [16], a Java-based design tool, and then converted them to a FPGA bit-stream for implementation [80]. The result shows that the FPGA-based string matcher exceeds the performance of the software-based system by a factor of 600 for large patterns.

Clark and Schimmel further developed the NFA generator with a similar approach in 2003 [50], but focused more on complex regular expressions. They claimed that the entire Snort rule database consisting of over 1,500 rules and 17,000 characters can be fit on a single one-million-gate FPGA board while keeping pattern matching

at gigabit traffic rate, with their approach. In 2004, they proposed a scalable pattern matching using multi-character decoder NFA technology [51]. This approach offers flexible trade-offs between character capacity, throughput, and data bus width and rate. A wide range of pattern set can be covered by high-performance circuits. This approach enables contemporary FPGAs to match a large number of complex patterns at network data rates from 1Gbps to 100Gbps.

Compared to hardware application of the NFA approach, there only a few hardware Deterministic Finite Automata (DFA) implementations being reported. Moscola and Lockwood at Washington University translated regular expressions into DFA in 2003 [120]. The content scanner was implemented on their Field-programmable Port Extender (FPX) platform and tested using real Internet traffic on the Washington University Gigabit Switch (WUGS). The pattern matching can be guaranteed at speeds of 1.184 Gbps for twenty-one regular expressions each with 20 characters, and exceeding speeds of 2.5 Gigabits/second for smaller numbers of similar regular expressions.

In 2004, Bos and Huang [19] reported a DFA approach on the IXP1200 network processor with support for large rule sets. They employed the Aho-Corasick algorithm in a parallel fashion, where each micro-engine processes a subset of traffic for pattern matching. In order to allow for large patterns as well as a large number of rules, the patterns are stored in off-chip memory. The evaluation results show that the processing speed is only 200Mbps, though the system is capable of handling full content scan for realistic threats.

4.2.1.2 Technology Analysis

Although the detail implementations of Finite Automata (FA) are different, the main approaches are similar in that they build an efficient Finite State Machine (FSM) for pattern matching. Of course, NFA and DFA have their own features.

A regular expression is a pattern that is used to match a set of strings according to certain syntax rules [152]. An FA can be constructed for string matching of a regular expression. An FA can be expressed as a directed graph where nodes represent states and edges are labeled as characters. One state is designated as the initial state and the remaining states are intermediate or accepting (or final) states. When the last input symbol has been received it will report the matching status depending on whether the DFA/NFA is in an accepting state or not.

For each input symbol, the next state of the current DFA state is uniquely determined. No state in DFA has more than one outgoing edge with the same label [149]. Unlike DFA, the next state of NFA may not be uniquely determined. It could be any one of several possible states depending on its constituent sub-expressions. An extension of an NFA is an NFA-epsilon, which allows epsilon (ε) transitions that represent a transition to a new state without consuming any input symbols. Hence, the edge of an epsilon-NFA may be labeled with a single character or ε [149].

Figure 4.1, reproduced from [149], provides some logic structures for NFA/DFA models. Figure 4.1(a) illustrates the simplest structure for single character matching. It has an initial state q and a final state f , and is applicable to both NFA and DFA. Figure 4.1(b) and (c), however, are NFA exclusive structures for the computing of $r_1 | r_2$ and $r_1 \cdot r_2$, i.e.: to match either r_1 or r_2 , or to match both of them, respectively. N_1 and N_2 are NFAs for regular expressions of r_1 and r_2 . They each have an initial state q and final state f . If the next state cannot be determined immediately from the current state, processing will register the present status and move to all next possible states for joint prediction until reaching the final state, otherwise the matching fails. The feature of joint prediction gives NFA higher throughput for pattern matching than DFA.

Although the functionality of NFA and DFA are transferable, they feature different hardware behaviors [149]. For FPGA implementation, their construction

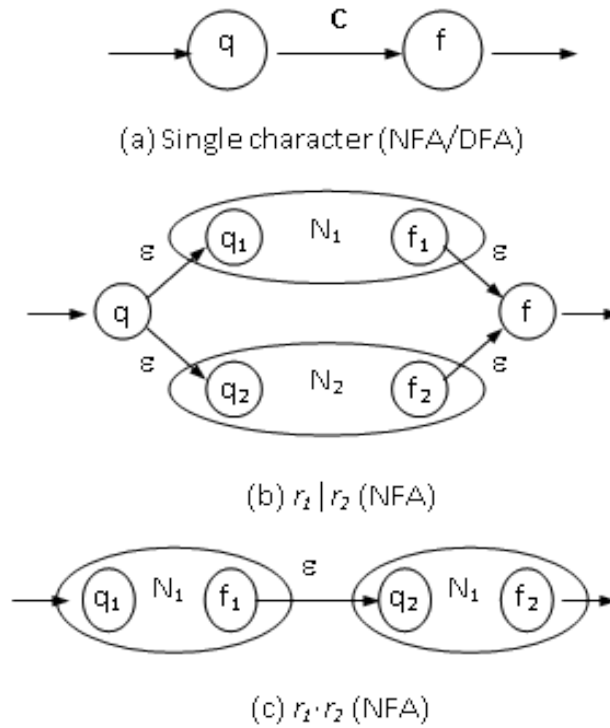


Figure 4.1. NFA/DFA for regular expressions.

time and relative memory occupation are much different. The construction technique shown in [149] can dynamically generate an NFA for a length n regular expression in $O(n)$ time with $O(n^2)$ memory, while constructing an equivalent DFA requires significantly more time and memory area, both at $O(2^n)$. After construction, both NFA and DFA are able to process one character per cycle. Assuming a pattern matching operation between a length n regular expression and length m input data, a serial machine requires $O(mn)$ time to reach the goal and requires $O(n)$ memory occupation. It takes total $O(2n + m)$ time and $O(2^n)$ memory occupation for the equivalent DFA and $O(n + m)$ time and $O(n^2)$ memory occupation for the equivalent NFA to process.

From the perspective of hardware implementation, construction time causes great impact to the overall performance of FA. Though the time complexity of NFA looks moderate, it actually requires dynamic updates, which may result in even heav-

ier construction overhead. In practice, optimized DFA construction techniques can be fast. However, for any DFA based matcher there will always be cases that cause exponential blowup in the time and memory required [149]. Since it is impossible to eliminate the configuration time of hardware, a feasible way is to minimize it. In this context, the preferred approach may be to implement the FA construction algorithm as a program that outputs the FA logic as a placed and routed netlist and subsequently use vendor tools for generating configuration bits [149].

In terms of logic level design, there are two standard methods for FA implementation on FPGAs [149]. One is through the use of binary encoding for state storage. However, this method is not efficient for NFA implementation, since the memory can look up only one next state at every clock cycle. For NFA construction, the one-hot encoding (OHE) scheme offers a better solution. It allows NFA to simultaneously look up any possible next state. The key is that each state of the NFA is associated with one flip-flop and that state is uniquely indicated by an output of one from that flip-flop; thus, multiple states can be encoded by having multiple ones with no conflict.

In comparison, NFA possesses shorter processing time and can be implemented in smaller overall area than DFA, but at the cost of dynamic NFA update. If the size of completed automaton is the most critical concern, the DFA approach contains up to $O(2^n)$ states, where n is the number of characters in the expression. However, the structure of DFA are simpler than NFA since all the states are explicit, but at the cost of higher memory consumption. This is more prominent in hardware applications when the size of patterns increases and the scalability issue dominates the overall performance of pattern matching applications. A good tradeoff between complexity and memory consumption is desired.

4.2.2 Content Addressable Memory (CAM) Technique

The most common hardware approach for pattern matching is regular expression based finite-automata (FA), either NFA or DFA, which results in designs with low cost but modest throughput [158]. Though the use of parallelism in FA implementations has been attempted, it is difficult in general. The implementations of FA are built with the implicit assumption that the input data are processed sequentially. Thus, the overall latency increases proportionally with the number of patterns. In addition, FA implementations are restricted in operating frequency by the complexity of combinational logic needed for state transitions, where complex expressions result in multilevel implementations [158]. In recent years, approaches based on Content Addressable Memory (CAM) have had renewed interest due to their unique ability to apply fast pattern comparison without memory addressing [72].

4.2.2.1 Existing Applications

To the best of our knowledge, Gokhale and Duois are the first who implemented Snort rules with a CAM in 2002 [68]. Their implementation achieved a throughput of 2.2 Gbps on a Xilinx Virtex XCV1000E device running at 68 MHz with 32-bit data each clock cycle.

A Ternary Content Addressable Memory (TCAM) based multiple-pattern matching approach was introduced by Yu and Katz in 2004 [178]. It features a don't care ('?') state in addition to the '0' and '1' states used in classical CAM approaches. The don't care state can be used as a mask, allowing one input to match multiple patterns simultaneously. Therefore, the TCAM approach is capable of handling complex patterns, such as arbitrarily long patterns, correlated patterns, and patterns with negation. For the ClamAv [91] virus database with 1768 patterns whose sizes vary from 6 bytes to 2189 bytes, the proposed approach can operate at 2 Gbps with a 240

kB TCAM.

Sourdis and Pnevmatikatos proposed a CAM implemented with discrete comparators in 2003 [157]. They adopted a scalable, low-latency architecture with extensive fine-grain pipelining to tackle the fan-out, match and encode bottlenecks. Their design achieved a processing bandwidth of 11 Gbps with operating frequencies at 340 MHz for fast Virtex devices. To increase throughput, they applied multiple comparators for parallel matching of multiple search strings. The evaluation of timing and area reports presented show that the match cost per search pattern character is between 4 and 5 logic cells. For a lower cost, they improved their implementation using shared comparators and developed the Decoded CAM (DCAM) in 2004 [158]. The results showed the area cost per search pattern character is less than 1.1 logic cells and an operating frequency of about 375 MHz (3 Gbps) on a Virtex2 device. When using quad parallelism to increase the matching throughput, the area cost was further decreased to less than one cell with a throughput of 10Gbps.

Hardware-based approaches are able to deal with pattern matching under tremendously high speeds but have high resource requirements. As the number of patterns increases, sufficient storage resources are essential. A Binary Decision Diagram (BDD) based CAM method was introduced by Yusuf and Luk in 2005 [179] to solve the problem of size limitation while retaining the speed advantage of hardware. Their pattern-matching engine is based on the tree-based CAM structure. Exploiting logic optimizations for multiple strings in the form of BDD, the approach involves hardware sharing at the bit level [179]. This method has been used to implement the entire SNORT rule set with approximately 12% of the area on a Xilinx CC2V8000 FPGA. The design can run at a rate of about 2.5 Gbps and is about 30% smaller than another related approach [12]. The main obstacle in optimizing for speed is large fan-out, which implies long latency. This is a common obstacle for tree-based structures. Adopting an optimized multi-stage pipeline may be a good solution for

mitigation.

4.2.2.2 Technology Analysis

The core part of pattern matching is performing fast comparison between input data and patterns. Since all the patterns are stored in memory devices, it is necessary to perform a memory search to obtain patterns for comparison. Traditional memory search techniques based on RAM technology require generating addresses to locate the patterns. As a result, the speed of addressing often becomes the limiting factor for memory search and further delays the comparison.

CAM is a class of parallel pattern matching circuits that is an outgrowth of RAM (Random Access Memory) technology [130]. In RAM circuits, an address is necessary to access the corresponding data. The number of address lines limits the depth of a memory, but the width of the memory can be extended as far as desired if possible. CAM circuits can also work in dual mode. In one mode, the CAM operates like a standard RAM circuit where data may be written to and read from the device [173]. In the other mode, the CAM circuits have a powerful parallel match mode: the entire memory array can be searched in parallel using hardware that compares each stored entry to an input value. If a matching value is found in any of the memory locations, a match signal is generated. Since no address line is required in match mode, the depth of a memory system using CAM can be extended as far as desired. As a further extension, a CAM circuit can provide the addresses of matched patterns in match mode when necessary, which is the opposite functionality of a standard RAM.

Content Addressable Memory (CAM) technology is optimal for memory search without addressing since it can directly compare the input data against entire list of patterns being stored in the memory simultaneously. This unique feature offers CAM based approaches superior speed for fast comparison. It results in an order-of-

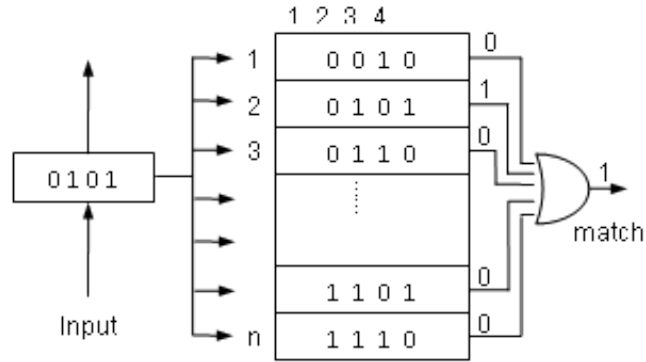


Figure 4.2. Binary CAM operation in match mode with packet length =1.

magnitude reduction in search time, in comparison with other memory search methods, such as: binary or tree-based searches or look-aside tag buffers [130].

Figure 4.2 illustrates the basic Binary CAM (BCAM) [68] operation for pattern matching based on FPGA implementation. As implied by its name, only '0' or '1' is stored in a BCAM. The width of CAM depends on how many LUTs are contained in one entry; it also determines the width of input data upon which lookup is based. A typical FPGA logic block contains a 4-input lookup table (LUT), which means an individual LUT can handle at most 4-bit input data. For simplicity, each entry in Figure 4.2 contains only one LUT, so the width of CAM is 4 bits. In Figure 4.2, 4 bits of input data are fetched for matching during each processing cycle. CAM circuits compare the input with all the pattern entries in parallel and 'OR' the results for final match indication.

In general, a CAM requires $n \times w$ bytes of memory space to store n entries, each w bytes wide. The simple scenario of an input packet of length $k \times w$ can always be processed in deterministic time $O(k)$ if the pattern contained in each processing entry is independent [178]. However, in many practical applications this is not the case since a pattern may cross processing entries. In addition, several patterns may be correlated to one another. If that is the case, more sophisticated mechanisms must be applied. As shown in Figure 4.2, shift registers may be applied for data holding to

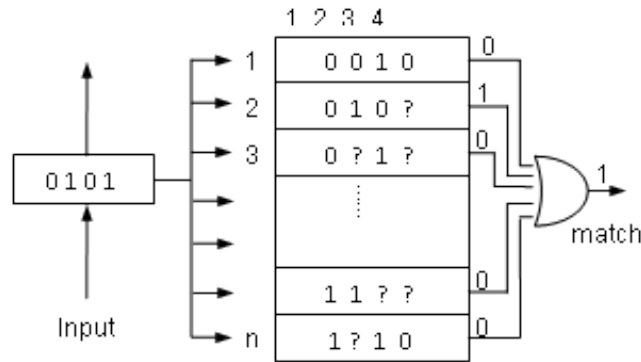


Figure 4.3. TCAM operation in match mode with packet length =1.

ensure that all possible patterns can be detected. Instead of simply passing bits four by four, shifting bits one by one will never miss a pattern spanning two processing entries.

Besides the basic BCAM approach, two other improved CAM approaches have been proposed: Ternary CAM (TCAM) [178] and Decoded CAM (DCAM) [157]. To perform exact matching with wildcards using a BCAM, the wildcards must be expanded to all possible matching patterns, and a BCAM entry is required for each. A more efficient method is to match only the interesting parts of the input data with patterns stored in each entry. The TCAM approach is dedicated for this purpose, as illustrated in Figure 4.3. In addition to the '0' and '1' states, it has a don't-care state, represented as '?' in the figure. With the application of wildcards, the TCAM screens out uninteresting bits at each entry for faster comparison.

Decoded CAM (DCAM) is a variety of basic CAM optimized for parallel matching. It was proposed in [157]. Unlike a basic BCAM, which employs a single large CAM to perform parallel comparison of all stored patterns, the DCAM is implemented with discrete 'comparators' for partitioned comparison. Patterns that would otherwise be stored in a large BCAM are first decomposed into elementary patterns according to certain rules at the character-level and are stored in smaller CAM circuits. Each comparator implements the functionality of such a small CAM. Com-

parisons are then performed between the input data and individual comparators in parallel. Finally, these intermediate results are processed by encoders for a matching decision. Taking advantage of pipelining and redundancy in patterns, this approach has potential to increase the efficiency of comparison operation dramatically. It was proposed in [157] and further developed in [158]; these enhancements are discussed in section 4.3.

In summary, CAM circuits feature regular structure and are suitable for pattern matching applications in hardware, especially for parallel implementation. The unique feature of searching entire memory contents simultaneously makes this approach faster than any RAM based searching approach. However, this speed comes at the cost of larger resource consumption than other approaches. With respect to FPGA implementation, the construction of CAM circuits relies on flip-flops for data storage, so the size of the implemented circuits is restricted by the availability of flip-flops [72]. Thus, as pattern matching circuits become more and more complicated, scalability issues emerge.

4.2.3 Optimization for Scalability

Usually, scalability is not an issue for small or simple systems, but can be the dominant factor in large complex systems. Hardware based approaches can significantly improve the performance of pattern matching, but at the cost of increased resource consumption. Without good system scalability, limited hardware resources can be quickly exhausted. Moreover, any modification in FPGA implementations, especially the constant update of pattern databases, requires synthesizing, mapping, and programming to the target device, which can be time consuming. As hardware based pattern matching applications become increasingly sophisticated, scalability becomes a large concern. In recent years, several approaches have been dedicated to maintaining scalability in pattern matching. Here, we introduce two popular tech-

niques: Bloom filter and min-cut partition.

4.2.3.1 Existing Applications

An application of using Bloom filter for packet payload inspection was reported by a research team from Washington University in 2004 [60]. Basically, its process consists of two stages: the first stage uses hardware Bloom filters to isolate all packets that potentially contain predefined signatures, and the second stage eliminates false positives produced from the previous step. A prototype based on the FPX platform [104] was built. Evaluation results show that this design can support up to 10,000 pattern matching operations at a line speed of OC-48 (2.4 Gbps). Extension of this effort was reported in [7] and [58] where detailed implementations were provided. They further improved their work to handle arbitrarily large pattern matching operations at the cost of slightly more on-chip memory in 2006 [61]. The idea of handling arbitrary length patterns is to split longer strings into multiple shorter fixed-length segments that can be handled by the basic Bloom filter.

Suresh and Guo proposed an automatic compilation framework in 2006 [162]. They used ROCC, a C to VHDL compiler, to generate a Bloom-filter based intrusion detection system on FPGAs. It is the first work that automatically generates VHDL for Bloom filter code written in C. Their synthesized hardware application can run at 73MHz and handle a throughput of 18.6 Gbps, while occupying approximately 8% of the area of a Xilinx XC2V8000 FPGA.

Researchers at the University of Southern California have dealt with the scalability issue using a partition based approach. They adopted a graph-based min-cut partition technique to decompose a large pattern database into certain basic pattern sets in 2004 [10]. After a reasonable partition is found, the redundancy contained in a pattern database can be reduced to reduce corresponding consumption of hardware resources. Since the partition problem is achieved at the Register Transfer Level

(RTL) rather than the gate level during the logic design phase, the system is affected less by synthesis tools, which facilitates pre-synthesis performance estimation [10].

In practice, pattern decomposition can be achieved via parallel hardware decoders. This operation is called "pre-decode" at the system level because pattern decomposition is done prior to pattern matching. The idea is to compare each input character to a set of known values of interest. The comparator outputs are fed through a network of shift-register delay elements to maintain the temporal relationship among characters; the outputs of these shift registers become individual match lines that are combined to perform pattern matching. This concept is an extension of the approach [158] mentioned in the previous section. Reported results indicate at least 8 times more area-efficiency than other shift-and-compare architectures, and 2 times more efficiency than other pre-decoded architectures [10].

A subsequent work, reported in 2005 [11], further optimized the approach by combining graph-based partitioning and tree-based matching of large pattern databases. Moreover, an optimized incremental design strategy was introduced for place-and-route based on the min-cut partition approach. Usually, full place-and-route is required when any modification is made in FPGA design, no matter how small the change. The proposed strategy needs only to perform partial place-and-route, since other partition patterns are not affected.

4.2.3.2 Technology Analysis

Bloom Filter based Technique A Bloom filter is a data structure allowing representation of a set of elements with probabilistic membership queries [60]. Patterns are stored in Bloom filters compactly via the computation of multiple hash functions. An important property of Bloom filters is that the computation time for pattern searching is independent of the number of patterns contained in the database. In addition, the required memory space for hashed pattern storage is independent of the size of

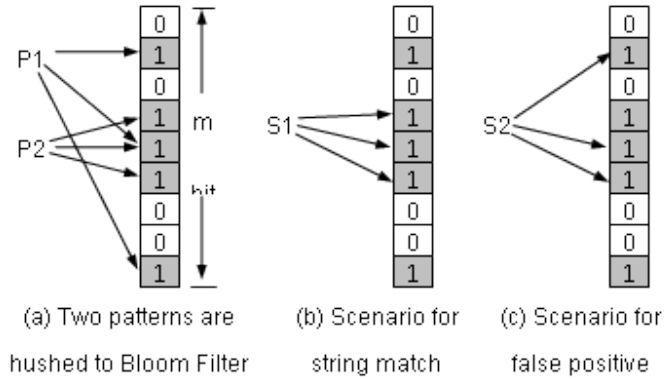


Figure 4.4. Basic bloom filter operations.

real patterns, though capacious memory space is preferred. Both membership queries and memory space depend on the number of hash functions. Hence, the Bloom filter technique naturally possesses good scalability.

Assume that a Bloom filter contains k independent hash functions and a memory with m bits. As shown in Figure 4.4, for example, $k = 3$ and $m = 9$. Each hash function maps to a single location for a given key (pattern). During initialization, the Bloom filter sets k bits out of m memory bits according to the hash functions for each stored pattern, as shown in Figure 4.4(a). While testing for membership, if any of the k memory bits is zero, the pattern is not stored; however, ambiguities exist when all bits are one. Figure 4.4(b) and (c) illustrate such scenarios. S_1 and S_2 represent the corresponding hash values of incoming data. Though S_1 and S_2 both hit bits that have already been set, only S_1 is a true match since there is no such pattern mapping to the exact location where S_2 points. The erroneous matching result of S_2 is called a false positive [5]. Thus, each query result may be a match or a false positive but never a false negative. Bloom filters may over-report matches, but never miss any.

Due to the existence of false positives, Bloom filter based approaches usually contain two basic stages: one for Bloom filter querying and another for false positive elimination. A model adapted from [58] presents a good illustration, as shown in Figure 4.5.

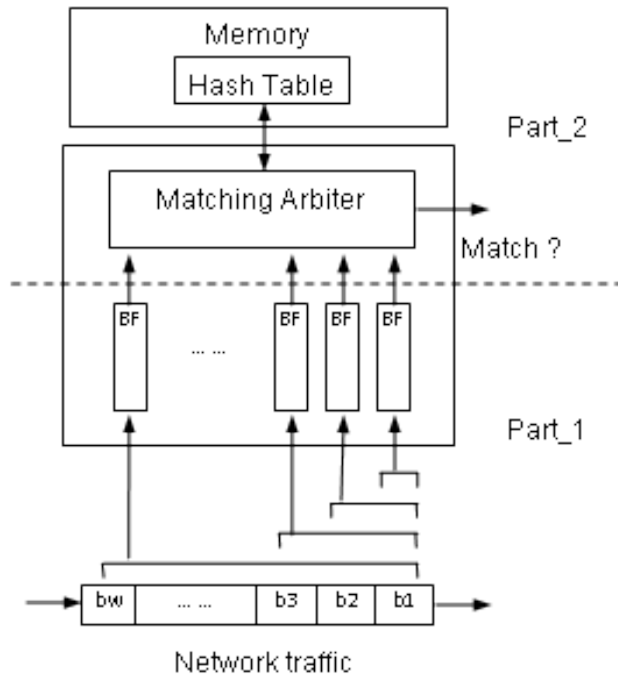


Figure 4.5. Basic unit for pattern matching application with bloom filters.

The lower portion of the figure, labeled Part_1, is for Bloom filter querying. This model includes w Bloom filters each having a different number of hash functions. Hashed patterns with the same length are stored in the same Bloom filter unit. In the figure, the rightmost Bloom filter unit contains all hashed patterns with the shortest length.

For simplicity both the shortest length of hashed patterns, and thus the number of corresponding hash functions, are one in this case. From the right to the left, the number increases by one. This indicates that the model has the ability to process different length patterns simultaneously. With data traffic connected, the size of the operation window is w bytes. At each clock cycle, one byte of data is shifted through the window and all subsets of this w -byte sequence of data are subjected to be simultaneously queried. In the case of multiple hits a potential match is chosen by a pre-defined policy; usually, either the longest match or the shortest will be considered as the interesting one.

As shown in Figure 4.5, the functions of Part_2 include false positive elimination and final commitment of a match. It consists of two main components: a matching arbiter and a hash table. The hash table may be stored in the on-board RAM. The matching arbiter buffers the hash value of potential matched data. Then, it recalls the hash values of interesting patterns from the hash table and compares them. If they are the same, a match is committed and a match signal is output. The whole procedure requires temporary freezing of current Bloom filter querying for processing, so as to cause certain delay to the incoming traffic. Since the potential matching rate is expected to be low in real traffic, this kind of delay may have a little effect to system performance. A proper applied data buffer for traffic holding could further mitigate this risk.

Hardware based Bloom filter implementations are quite attractive for quick pattern matching. In addition to possessing good scalability, other possible security enforcements may be added to the process of hash value generation. It is practical for special purpose hardware to handle those tasks. However, some researchers argue that the main weakness of the Bloom filter approach lies in its fixed length pattern processing. Without optimization, accommodating n different length patterns requires n Bloom filter units.

Min-cut Partition Based Technique The min-cut partition technique stems from graph theory. With respect to pattern matching applications, the partition focuses on the pattern database, with the goal of exploiting redundancy for hardware optimization. Patterns can be considered as combinations of different characters and some elementary characters tend to repeat across patterns. According to the study in [10], Baker pointed out that only about 100 different characters are ever matched against within the whole set of the Snort database, if more than 2000 patterns could be completely decomposed into single characters.

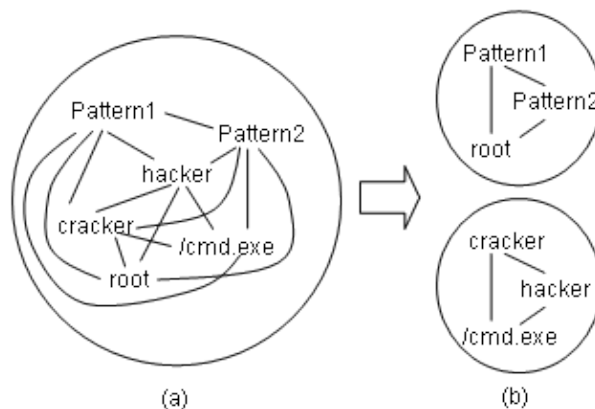


Figure 4.6. (a). Un-partitioned graph. (b).Partitioned graph.

The rationale for partition is that the number of repeated characters within a group be maximized, while the number of characters repeated between groups is minimized. With min-cut, the entire pattern database is partitioned into several small groups. Figure 4.6, reproduced from [10], illustrates this concept. Figure 4.6(a) represents the original pattern database, which is a densely connected graph. Each node is a pattern and the edges between them represent shared characters. Figure 4.6(b) represents the result after partition.

In terms of hardware implementation, this result implies that the number of pipeline registers can be decreased, since small groups naturally contain fewer characters, which reduces the length of pipeline stage. Meanwhile, the average utilization of each character is increased, because similar characters more frequently appear within a few groups. In addition, more parallel operations can be performed since more pattern units are created due to the min-cut partition. Considering these advantages, redundant hardware resources for pipelined operations can be released or be used to support more parallel operations if possible.

Through reasonable min-cut partitioning, the pattern matching operation can be achieved efficiently and compactly. However, rules for partitioning are usually heuristic; thus, it is difficult to find a single algorithm to maintain efficiency across applications. Fortunately, the emergence of design automation tools enables efficient

application of trial-and-error.

4.3 TCP Stream Preprocessing Application

TCP is a connection-oriented protocol which provides guaranteed transport for network applications over unreliable network environments. More than 85% of traffic uses the TCP protocol over the Internet [89,148]. With TCP, packets can be dropped, duplicated, or re-ordered during transmission. Hence, data streams observed at network connection points may deviate from their original patterns. Since this scenario can be exploited for concealing maliciousness, it is a necessity to have a seamless TCP traffic monitoring [122].

A Network Intrusion Detection System (NIDS) is a type of network security application that provides more sophisticated protection than firewalls. TCP session monitoring is one of the basic functions of a NIDS. To achieve TCP session monitoring in a NIDS it is necessary to perform packet reordering, stream reassembly, and state tracking operations of TCP traffic [134]. In this paper, we consider these operations as preprocessing of TCP traffic since most security applications require well organized TCP streams for analysis.

Modern security applications that focus on the network infrastructure will be inefficient or less effective without the capability of high-speed TCP traffic handling. The data rate of communication links on which current Internet backbones operate is in the range from OC-3 (155 Mbps) to OC-768 (40 Gbps) [145]. Many advanced infrastructure security services require high-speed TCP traffic manipulation, including packet inspection, content-based routing, Internet worm detection, DDoS attack resistance, and spam removal. Hardware-based implementations are essential to most security applications for adequate performance.

However, hardware resources are generally limited in practice and efficient re-

source utilization is highly desired. Modular design of hardware offers an opportunity to separate TCP stream preprocessing from security functions. Since many security functions require TCP stream preprocessing, separating this function allows subsequent applications to be relieved of this task, which can result in more resources being available for security functions. These recovered resources can be utilized to support other tasks, enabling high quality network services at the Gigabit rate.

Thus far, the research in hardware based TCP stream preprocessing has not received sufficient recognition, as only a few papers related to this topic have been published. Its reasons could be twofold. On the one hand, the processing of TCP streams itself may seem to be fairly straightforward, so it may not be considered worth that effort. On the other hand, the benefit of modular design for hardware TCP stream preprocessing may not be clearly identified. In fact, implementing this function as an individual module is meaningful to practical applications. In our opinion, a modular TCP stream reassembly block can not only save valuable resources for subsequent functions, but also optimize the system architecture in a concise way. This property is highly desirable to implement complicated security applications at the system level.

4.3.1 Existing Applications

A TCP stream reassembly and state tracking module was reported in 2002 [122]. They proposed to implement portions of the NIDS functions to a high performance reconfigurable network card. Its feasibility has been demonstrated by a reassembly module designed in VHDL and implemented on the Xilinx XCF1000 FGPA platform. It operates at a 100 MHz system frequency with 32-bit data width, and the maximum throughput achieved is 3.2 Gbps for raw TCP/IP traffic.

After successfully adapting the reassembly function from software to hardware with a single thread, feasibility with multi-threaded execution was subsequently ex-

plored. This led to the question of how to manage multi-threaded processing. Though they did not present any detailed implementation, it is conceivable that the proposed design works for multi-threaded TCP reassembly processing under modest network circumstances.

In 2003, Li and Torresen [98] implemented a reconfigurable hardware architecture to replace a software based STREAM4 [62] preprocessor in Snort, which performs TCP stateful inspection and reassembly functions. Through logging and analyzing the TCP stream packet by packet, it is possible to predict what will happen next based on the status of present and past packets. This function is very helpful for the detection of certain abnormal activities.

The major innovation is the implementation of a Server TCP stream reassembly block and a Client reassembly block. Unlike the implementation in [122] where reassembled flows for opposite directions are simply stored in two different RAMs at the end of the processing, their approach segregates unprocessed server-oriented streams and client-oriented streams at the very beginning. The design was implemented on a Xilinx Virtex XCV1000-6 FPGA. The core part of the TCP stream reassembly unit consists of two 32 bit comparators and one 32 bit adder. Experimental results showed that this design achieved a throughput of 3.06 Gbps.

Another similar implementation with a slight modification was reported in [97], which was evaluated on top of the FPX platform from Washington University at St. Louis (WUSTL) [105]. The whole system has been placed and routed into a XCV2000E FPGA with throughput at 2.75 Gbps. Only a total of 2.5 percent of the SLICES (496 of 19600 SLICES) of an XCV2000E-6 were required to implement this system [97].

Schuehler and Lockwood reported their TCP-splitter in 2002 [144]. TCP-splitter was designed as a lightweight, high performance circuit module that provides a consistent TCP data stream to client application systems. It consists of two logical

sections: TCP-input, which handles ingress IP frames and most of the job of a TCP-splitter; and TCP-output, which is responsible for packet routing and frame delivery to either the IP stack or its client applications for further security processing. Their synthesized design on a Xilinx Virtex 1000E-7 FPGA demonstrated a throughput of 3.2 Gbps at 100 MHz clock frequency.

A TCP processing engine has been developed based on the splitter [146]. The most significant improvement in the design is the appearance of off-chip memory. Off-chip memory dramatically expands the available resources for TCP stream processing. In addition, with the assistance of off-chip memory, cooperation between different functional modules becomes feasible. Information from higher-level applications can be used to assist service of future TCP stream processing. In this way, processing of a TCP stream can be evaluated at a higher level, incorporating this intelligence with self-updating.

A structure for parallel TCP stream scanning was also discussed [146]. The designed circuit was targeted for the Xilinx XCV2000E FPGA on the FPX platform. This architecture is able to monitor eight million simultaneous TCP flows at OC48 rates (2.5 Gbps). In addition, the cooperation issue was discussed in a subsequent publication in 2004 [145], in which they added a set of encoding and decoding circuits for the cooperation among multiple FPGA devices.

4.3.2 Technology Analysis

Among the published TCP stream processing applications, most adhere to one of two models. One model acts as a connection endpoint of network traffic that accepts and processes all the incoming network traffic; the other acts as a monitor that inspects the TCP streams passing through. Each model has distinct characters that can complement the other. Selection of a suitable model for implementation depends on specific design requirements. Resource consumption is always a big concern that

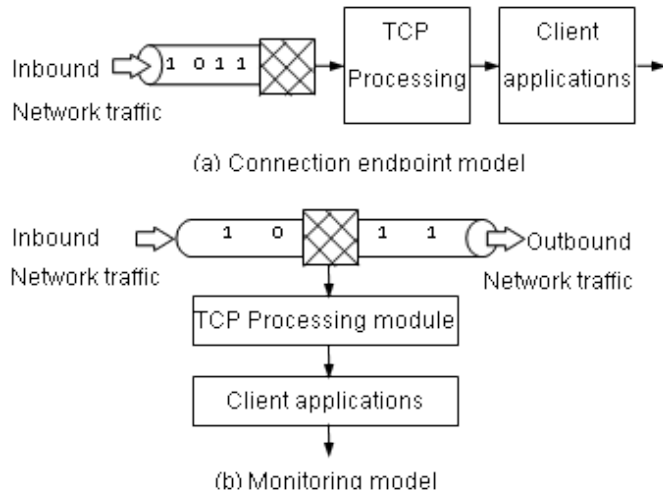


Figure 4.7. Basic models for TCP stream processing.

affects the practical implementation.

Usually, the endpoint model is suitable for fine TCP stream processing at end-system networks with moderate TCP connections, such as deep state inspection. Necker and Contis’s design [122] follows this model. By instantiating multiple processing circuits, up to 30 TCP/IP connections can be processed simultaneously on a single FPGA board. On the other hand, the monitoring model is suitable for coarse TCP stream processing in high throughput networks with massive TCP connections, such as reassembly, reordering, or counting. Following the monitoring model, it is able to monitor all TCP flows passing through the TCP-splitter [144]. Figure 4.7(a) represents the connection endpoint model, and Figure 4.7(b) represents the monitoring model.

Applications for TCP stream preprocessing require placement at locations where the interesting traffic passes. Indeed, all packets associated with a TCP/IP connection must be available for inspection. Only in this manner can consistent security protection be guaranteed to all packets within a TCP connection. In general, the best deployment locations for TCP stream applications are at places where edge routers are located [144]. It is extremely difficult to process traffic in the middle of

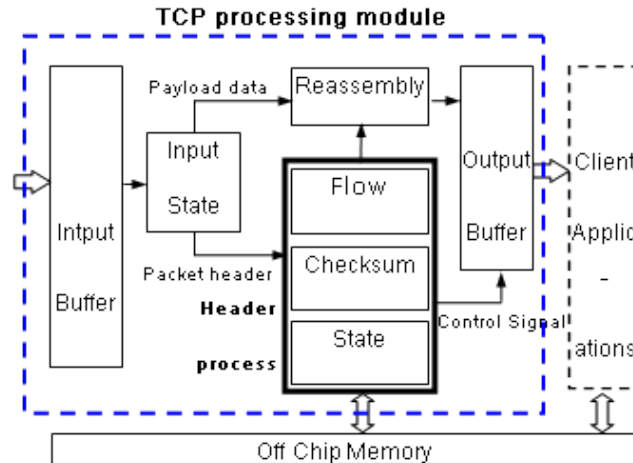


Figure 4.8. Block diagram for TCP stream preprocessing.

the network in practice, since packets may subsequently be dropped, duplicated, or re-ordered along the way. However, it is not difficult to process them at the location of connected points. In addition, the traffic passing through the network is not widely distributed. It is usually concentrated to flows over a limited number of routers [63].

To correct the TCP stream defects produced during transmission, TCP stream applications mainly focus on the inspection of the packet header. Although packet classification also focuses on packet header, it concentrates on malicious inspection other than the typical TCP defects inspection. Figure 4.8 demonstrates the basic procedure of TCP steam pre-processing.

When an incoming data stream enters the processing module via a network adapter, an input buffer holds its packets for synchronization. Buffered packets are then decomposed into packet headers and payloads by an input state machine. Corresponding tags are assigned to both parts for later association. Subsequently, payloads are dispatched to another buffer to wait for reassembly, while packet headers are dispatched to the header processing engine for advanced processing. During inspection, all information contained in a packet header is subject to check, including IP address, port number, sequence number, acknowledgment number, and flags, etc.

A state manager works as a coordinator handling all components of the header processing engine, including the checksum engine and flow classifier. In addition, it takes the responsibility of off-chip memory access for updating. Since the off-chip memory may contain useful information from other applications, continuous updating not only improves the resilience of the header processing engine, but also makes it more intelligent. The header processing engine may feature the capability of segregation or dropping verified malicious flows at this point. Therefore, the boundary between typical TCP preprocessing and packet classification is blurred from this perspective. Of course, this function is optional; its usage depends on specific application requirements.

After being inspected, eligible headers are rearranged into flows according to certain rules. Individual packets are reassembled using the association numbers previously assigned. Meanwhile, certain useful control signals may be assigned to these flows for future use. Finally, reorganized TCP flows are stored in an output buffer for further processing by subsequent client applications.

4.4 Internet Worms and DDoS Attack Detection and Containment

Internet worms and Distributed Denial-of-Service (DDoS) attacks have been identified as the two major security threats towards the network infrastructure [24]. Since both aim at vulnerabilities of the network infrastructure, our daily activities are more sensitive to these attacks than other higher level attacks. MyDoom, an Internet mailer worm first detected on 26 January 2004, is considered one of the fastest-spreading and possibly the most dangerous worm in history [132]. In the first 36 hours after release, it reached more 160 countries and infected over 100 million emails [169]. Such widespread infection can be even more severe if malicious DDoS

content, such as a "Trojan horse", is piggybacked onto the payloads of worm packets for a joint attack. Unfortunately, this became true when approximately 500,000 computers worldwide were infected by MyDoom.a, which then launched a Denial-of-Service (DoS) attack on SCO Systems services at www.sco.com on 1st February 2004, according to a report from CRN.COM [88].

In practice, security applications can only protect a limited network area and have little information about the network as a whole. Even during a single attack, security strategies may need to vary depending on the specific attack stage in progress. Though different attacks exhibit diverse behaviors, security strategies against them usually can be categorized as pre-attack prevention, under-attack containment, and post-attack update.

Currently, most security applications against worm attacks are applied during the under-attack stage. Good disguise and quick outbreak make worms difficult to prevent at the pre-attack stage. Due to the property of self-propagation, worms can spread quickly and have the potential to evolve into massive attacks over the whole network within a short time. Though many worms make no attempt to modify the systems that they pass through, they may quickly exhaust resources of and finally crash these systems. It is essential to contain the propagation of worms as early as possible during their outbreak.

Compared with worm containment, security strategies against DDoS attacks are more complicated due to the two-phase discipline of DDoS attacks [131]. The first phase of a typical DDoS attack is to compromise a large number of computers and recruit them into a zombie army; the second phase is to indirectly launch DDoS attacks towards a specific target through these zombies [131]. The joint attack of DDoS and worms can be even more destructive. If attackers take the advantage of worm spreading to comprise thousands or even millions of computers and subsequently launch a DDoS attack, none of today's network infrastructure will be able to survive under

such an aggressive attack without proper protection.

Exploring this two-phase attack discipline, security applications can limit or reduce the number of zombies by filtering the malicious content intended for this purpose at the pre-attack stage. However, under-attack containment is the core part of the defense against a DDoS attack once it begins. Finally, post-attack update should not be ignored, since it is the most convenient way to maintain the resilience of security applications.

4.4.1 Existing Applications

Although the importance of defending against Internet worms and DDoS attacks has been widely recognized and many efforts have been reported, a few of these have addressed defense approaches based on hardware. Excluding the fact that certain applications may not be released due to national security or commercial concerns, current academic research in this area is still at the initial stage.

Lockwood and Moscola reported their first hardware-based implementation of a worm containment scheme in 2003 [107]. This design uses FPGA devices for worm scanning at high speed Internet traffic rates. The system consists of three interconnected components. The major component is the Data Enabling Device (DED) which is deployed at key traffic aggregation points for packet inspection. Content Matching Service (CMS) updates the signature database and automatically generates the corresponding binary code for the reconfiguration of DED circuits. Regional Transaction Processor (RTP) works as a coordinator that manages the system to maintain proper operation.

The Field Programmable Port Extender (FPX) card [104] functions as the interface between network traffic and the hardware device. It is the heart of DED. Through the utilization of layered protocol wrappers [20], application-level flows can be decomposed for further analysis. Implementing four signature detection modules

in parallel, the FPX is able to process data at the rate of 2.4 Gbps with a single Xilinx Virtex 2000E FPGA. At this rate, the DED module can achieve full throughput for IP packet lengths ranging from 40 bytes to 1500 bytes. A more detailed design can be found in another paper published in 2003 [108].

The above design is a typical application of signature detection. All decisions are made depending on prior attack knowledge. However, this technology is not sufficient for unknown attack detection. In 2004, an anomaly detection based approach for worm containment was proposed [112]. It was inspired by the idea that a worm detection system should keep looking for frequently occurring contents [150]. The system can process traffic at full line rate and was implemented on the FPX platform. The circuit runs at a frequency of 91.5 MHz and network traffic is fetched into the device with 32 bit-width. The circuit fits well with Xilinx Virtex 2000 FPGA device and allows the processing at OC-48 line speed on the FPX platform. With a pipeline delay of 7 clock cycles, the circuit introduces a delay of only 70 ns. This implies that the system is feasible for real-time worm containment.

Due to the fact that even legitimate data packets, such as SYN-ACK, RST, or ICMP packets in TCP flows, can be used for DDoS attacks [32], it is often difficult to discern malicious activities via signature-based detection. However, the flooding property of DDoS attacks makes them relatively easy to be detected via anomaly detection. A recently published DDoS detection application by Oh and Park in 2007 is based on the inspection of network traffic rates [125]. A double token-bucket mechanism was applied for bandwidth control.

This double token-bucket mechanism sets two essential thresholds, one for the first bucket size and the other for the sum of both bucket sizes. During the continuous flow of incoming traffic, the current token level of traffic bandwidth varies. If the token level exceeds the threshold of the first bucket size but not the second one, a warning will be issued for notification. If the token level exceeds the second threshold, packets

will be dropped due to the limitation of throttling bandwidth. At the cost of dropping packets, valuable bandwidth can be saved. This design has been implemented on a security board that integrates Xilinx Vertex II Pro FPGAs with a set of two gigabit Ethernet interfaces for evaluation. It can serve up to 2 Gbps Ethernet on bidirectional traffic and a full 1 Gbps bidirectional traffic without loss [125].

The theory of using spectral analysis for DDoS attack defense has been proposed for years [14,44]. However, few corresponding hardware-based approaches have been developed due to the infeasibility of using hardware for implementation and the cost of hardware devices. A Power Spectral Density (PSD) analysis based hardware application was recently developed [33]. The design is specifically targeted for online detection of shrew attacks [93,111], a type of stealthy DDoS attacks. The idea stems from the observation that malicious shrew attack flows are more distinguishable in the frequency domain. With a proper DFT conversion, it is more convenient and accurate to perform the anomaly detection in the frequency domain [39]. The key procedure to achieve this goal is the consecutive calculation of autocorrelation and Power Spectrum Density (PSD) of sampled data. This application was designed on the Xilinx ISE platform with the Virtex-4 XC4VFX12 as the targeted device. Under extreme experimental conditions, with a system frequency of only 5 MHz and a long TCP flow window of 4096 packets (normal TCP-flow length is less than 1024 packets), it takes only 6.88 ms to achieve the major processing required.

4.4.2 Technology Analysis

Defending against Internet worms or DDoS attacks is a systematic process that integrates various technologies and strategies. The choice of applying different security technologies and adopting corresponding strategies depends on the specific application. Observing existing security solutions against Internet worm and DDoS attacks, we find that many basic technologies are shared among applications. Though

Table 4.1. Applicability of basic security applications.

Technology Category	Sub-category	Best Time to Apply		
		Pre-attack	Under-attack	Post-attack
Signature detection	Packet header classification	✓	✓	
	Deep payload inspection	✓	✓	
Anomaly detection	Direct content counting	✓	✓	
	Spectral analysis	✓	✓	
Knowledge update		✓	✓	✓

we will only focus on the defense of the two types of attacks here, the above observation is generally applicable to most security solutions. For convenience, Table 4.1 associates the major technologies with their best application time. Following this categorization, this section summarizes the technologies adopted for reconfigurable hardware based implementations.

From the perspective of defenders, threats to the network infrastructure can be classified into known and unknown types. The known threats imply that these threats have been captured and their properties have been studied. Their signatures are usually collected and stored in a knowledge database for future reference. On the contrary, unknown threats include all that have not been identified.

For known threats, signature detection is the most intuitive and straightforward technique for quick and accurate identification. It can be applied at both pre-attack and under-attack stages for prevention and containment. Any known type of worm and malware containing a malicious DDoS attack payload can be easily detected through packet header classification and deep payload inspection. The intensive computing required of signature comparison can be accomplished on high performance hardware. An analysis of hardware-based applications of these has been presented in previous sections. Anomaly detection is another option for known threat detection, but is more popular for unknown threat detection.

In addition to techniques developed directly for maliciousness detection, update techniques for resiliency of security applications is equally important. Knowledge update mainly focuses on update of all kinds of databases, especially the signature database. Due to the complex procedure of hardware reconfiguration, a dedicated mechanism is desired.

4.4.2.1 Anomaly Detection

Generally speaking, anomaly detection can run faster than signature detection. Instead of performing exact matching of a large number of signatures, as in signature detection, modeling normal behavior and evaluating deviation from thresholds is typically less computing intensive. However, it is at the cost of generally higher false positive rates. Anomaly detection is widely applied for detection of unknown malicious activities at pre-attack and under-attack stages. Due to constantly changing malicious threats, it is impossible to have all knowledge available before new threats emerge. Through continuous observation and modeling of normal behavior, anomaly detection offers a way to find possible threats via deviation from the normal model without knowing signatures. The key is to find reasonable thresholds, so a good performance balance between detection and false positive rate can be achieved.

Most Internet worms and DDoS attacks exhibit a common characteristic of flooding during attack, though Internet worms spread randomly while the DDoS attacks target specific victims. Nearly all prevalent Internet worms and DDoS attacks had not exposed themselves before their breaking out. Taking advantage of anomaly detection, these kinds of threats can be effectively detected, so as to stop them from further propagation to curtail wider impact on the network infrastructure. In the following subsections, two application modules are presented. In order to explain their operation, both are assumed to be run under a single thread. However, higher throughput can be achieved using multi-threaded operation via multiple-module in-

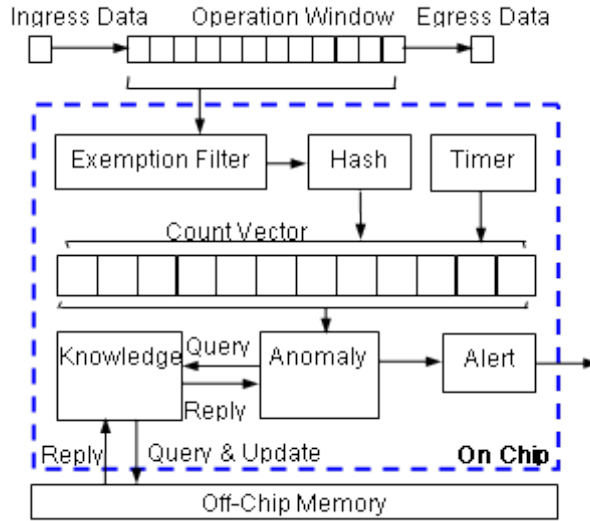


Figure 4.9. Block diagram for payload anomaly detection.

stantiation.

Direct Content-Counting Technique Direct content-counting is the most intuitive and widely used technique for anomaly detection. Any observable elements are valid for counting, including packet payloads, addresses, port numbers, etc. Regardless of the specific content counted, an anomaly can be detected as long as the counts of inspected objects deviate far enough from the normal range. Figure 4.9 shows a block diagram of a typical logic module for anomaly detection [112]. It focuses on detecting anomalous packet payloads.

With a suitable data rate, all packets are expected to be inspected. A fixed-length operating window limits the number of packets being processed simultaneously. Specifically, a window of only one data byte is assumed in this example. An exemption filter works as a coarse filter excluding many typical patterns with high frequency upfront. Through the hash functions, data patterns are hashed into specific locations of a counter vector, which adds one to the corresponding counters each time. If the counter of a pattern goes beyond the suspicious thresholds stored in the anomaly arbiter, an anomaly is detected. A timer is employed to refresh the count vector

periodically. Proper timer length is important to efficiently capture as many anomaly patterns as possible. An important issue related to start-up is how to set the initial counter values. A typical way to handle this problem is to subtract an average value from all the counters, instead of resetting all the counters to zeros [112].

In practice, it is not wise to make decisions only on the result of anomaly arbitration due to its high false positive rate. It is both desirable and necessary to reduce this uncertainty. We call the block performing this function a knowledge analyzer, as shown in Figure 4.9. The goal is to reduce the false positive rate of suspected anomalies. The knowledge analyzer not only contains signatures or statistic estimations for reference, but also provides other available knowledge for support. Instead of occupying limited on-chip memory, off-chip memory provides a flexible space for knowledge storage. In addition, associating with off-chip memory is convenient due to its potential capability of sharing with other functions and even other systems.

Power Spectral Density based technique Other than direct content-counting based anomaly detection, spectral analysis based anomaly detection has begun to emerge in recent years. The rationale is that certain complicated processes in the time domain may have more concise behavior in the frequency domain. A key step for spectral analysis is converting time domain data into the frequency domain. This operation is very expensive from the perspective of computing power, especially for real time applications. Though many hardware devices can provide the necessary computing power, the high cost has prevented them from gaining widespread use. The rise of spectral analysis based approaches benefit from advancing hardware technologies, especially the emergence of high-performance cost-efficient FPGA devices.

Up to now, there are more and more hardware devices with powerful cores and larger on-chip memories than ever.

The traditional processing bottleneck of the Fourier transform and its rela-

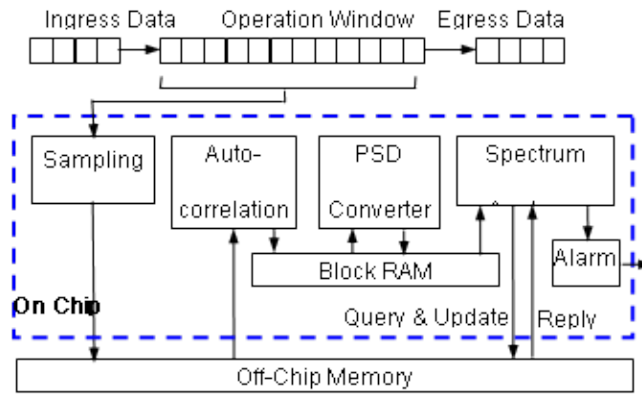


Figure 4.10. Block diagram for PSD based anomaly detection.

tively high memory access requirements is being eliminated. In addition, hardware vendors are willing to release diverse pre-designed intellectual property (IP) cores that are optimized for their products. These IP cores offer great convenience for design. From the perspective of hardware applications, the Power Spectral Density (PSD) based technique is quite mature and very suitable for hardware implementation.

Power Spectral Density (PSD) based anomaly detection exploits the property that normal TCP flows possess a periodic Round-Trip Time (RTT) property during transmission, while anomalous TCP flows do not [45]. It is efficient to reveal this phenomenon by checking the PSD of corresponding packet flows in the frequency domain. However, network traffic is naturally sampled in the time domain and it is necessary to convert the sampled sequences into the frequency domain in order for this examination. In practice, the whole procedure is treated as discrete signal processing; N samples are taken within a fixed-length time interval for processing. A detailed explanation can be found in [45] and [33].

An abstract diagram of a PSD based anomaly detection module is given in Figure 4.10. A fixed-length operating window is set for data acceptance. At each time interval, the same quantity of data will shift in and out of the window. Due to the high operating frequency of hardware devices and low sampling rates of real data, it

is typically better to buffer the raw data in off-chip memory so that valuable on-chip resources can be reserved for more important tasks. After one unit of N sampled data points is gathered they are fetched to an autocorrelation generator. From there, the data is processed in a high-speed DFT hardware device. After the DFT operation, the values of power spectral density of fixed length packets are generated and ready for analysis.

The spectrum analyzer performs anomaly analysis in the frequency domain. It usually contains some normal frequency-patterns of regular packet flows or specific thresholds for decision. Additional knowledge residing in off-chip memory is available to provide extensive support. Since this mechanism exploits the periodicity of TCP flows, it is sensitive to abnormal deviation. Low false positive rates make it possible to eliminate the corresponding mitigation required in other approaches.

4.4.2.2 System Update

Techniques for knowledge-update mainly focus on the update of different kinds of databases, especially for those storing signatures. Through continuous update, the resilience of security applications can be maintained at a high level. Usually, the majority of update is scheduled routinely. However, dynamic updating during attack requires time-efficient update to mitigate losses. In addition, corresponding hardware circuits may require certain modifications to accommodate the updated knowledge. With regards to FPGA devices, this process is referred to as reprogramming or reconfiguration. It is a multi-step process, including synthesis, mapping, place-and-route, and finally downloading the binary code to the target device.

An abstract block-diagram in Figure 4.11 illustrates such an updatable architecture at the system level [107]. This system consists of three major modules: inspector, updater, and manager. Each module performs specific functions. In order to emphasize the process of update, all the functions corresponding to packet inspec-

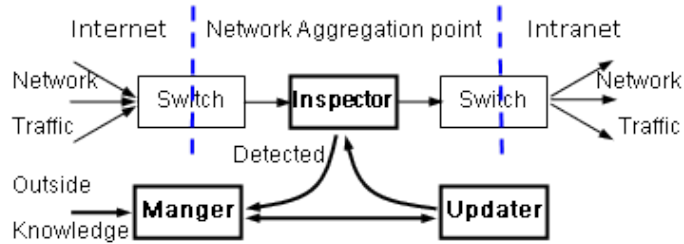


Figure 4.11. Block diagram for updatable system architecture.

tion are abstracted in the inspector module. The manager module takes charge of the process of update and the updater module performs the hardware reconfiguration.

The manager module has two sources of knowledge update: one is from the outside of the system, such as existing knowledge databases; the other is from the system itself, utilizing the detection results of the system. At least two update modes are required: by-request update and routine update. Using these modes, a system administrator can easily manage the update. However, both modes are relatively static. A dynamic update mode that can update the system online, especially the signature database and related knowledge, is highly desired. That way, any knowledge of newly detected maliciousness can be quickly incorporated for the inspection of future packets.

However, dynamical update of hardware-based applications within a restricted time period is nontrivial, due to the complex procedure of hardware reconfiguration. Certain tradeoff can be made to handle this problem. An intuitive method is to use half of a FPGA to actively run operations leaving the inactive half ready for reprogramming. When an update has been made in the inactive half, it becomes active and the previously active half is released for the next reprogramming. This approach, however, may be naive for practical implementation. In general, dynamic hardware update is still an open question.

Finally, it is the responsibility of the update module to set up the hardware-based inspector through a reconfiguration procedure. Since the detailed reconfigura-

tion procedure depends on specific hardware devices and software kits, it is beyond the scope of this paper. In a security oriented system, however, any potential vulnerability should be considered. The last major step for reconfiguration of a FPGA is downloading the configured bit-stream to the target device. If a remote operation is performed through the network, some new security issues regarding the protection of bit-streams during transmission need to be considered.

4.5 Discussion

The application of hardware for network infrastructure security has become an active area of research. With support of powerful hardware devices, many previously unfeasible computing intensive security tasks in software-based applications now become feasible. Meanwhile, hardware-based applications still face severe performance challenges due to the fiercely rising demands to handle higher traffic rates, more intensive analysis, real-time operation, and even the collapse of Moore's law for sequential processing [128]. Without a deliberate design, valuable hardware resources and powerful computing capabilities could be quickly exhausted.

Researchers have pointed out that now is the time to rethink the use of hardware for security applications [128]. They presented their vision to pursue the high parallelism, stating: "the key is devising an abstraction of parallel process that will allow us to expose the parallelism latent in semantically rich, stateful analysis algorithms." [128] Their proposal includes three major steps: 1) using high-level language to express rich forms of network analysis, 2) executing compiled elements of the language with a powerful parallel abstraction, and 3) compiling the results from the abstraction to concrete hardware implementations. Through decomposition and dispatching, the implementation of a sophisticated parallel process can be efficiently assigned to suitable devices. In addition to potentially high system performance,

the utilization of available hardware resources can be maximized. Furthermore, an architecture exploiting multi-core processors to parallelized Network Intrusion Prevention (NIP) was proposed recently [129]. These proposals have introduced certain new design concepts that are currently not available; thus they are still open issues.

In addition to issues related to hardware implementation of local security applications, the challenge of system-wide collaboration lies in the global cooperation among these applications. This goes well beyond the scope of local hardware implementation and must be considered high-level issues of systems engineering. The idea of system collaboration was proposed in [24, 25].

System-wide collaboration offers individual applications a global view of dynamic security situations. Since individual single-point-deployed security applications can only cover limited areas of the network infrastructure, they generally do not have global knowledge. With multi-point-collaboration, a security shield which covers a wider area of the network infrastructure can be established without major modifications to the current single-point-deployed architecture. In this manner, applications at individual nodes have more ability to handle sophisticated security problems.

From our viewpoint, achieving system-wide collaboration requires addressing collaboration at two levels. The first level is a collaboration between different single-point-deployed security applications within the same system. The second is a collaboration between different single-point-deployed security applications in different systems. Then, the corresponding hardware implementation changes should follow. Until now, collaboration technologies have been based on the use of an overlay network within the same security system [24] Through effective information sharing and updating, system performance can be improved substantially.

Collaboration at the system level can not only further improve the performance of current security applications but is essential to build intelligent security systems, such as neural-network-like security systems. The prospective benefits stim-

ulate research interests to develop distributed systems [166]. However, this research is still at the initial stages. The collaboration among different security systems is non-trivial, and up to now there has been no reported mechanism to effectively coordinate different security systems. In fact, many questions regarding collaboration are open. However, the potential benefits justify more effort.

4.6 Summary

This chapter surveyed existing reconfigurable hardware-based applications for network infrastructure security. Since traffic analysis at packet level serves as the fundamental for all the relative processes, and the volume of TCP-based streams dominates the overall network traffic on the Internet, applications related to TCP traffic analysis are my focus on this survey. The organization of this chapter also follows the order of TCP packet processing for traffic analysis. According to different phases of process, diverse applications are categorized to four groups: packet header classification, payload examination, TCP stream preprocessing, general DDoS attack detection, and Internet Worm containment.

Combining brief descriptions with intensive case-studies, this survey presented a clear picture of mainstream technologies applied in this field from basic principle to systematic architecture. From the system perspective, signature-based detection and anomaly-based detection complement each other for covering detection of known and unknown malicious activities, respectively. They compose the foundation of network intrusion detection, and are my focus on the following study.

Chapter 5

PSD Data Converter for Anomaly Detection

The low-rate TCP-targeted Distributed Denial-of-Service (DDoS) attack, also known as Shrew DDoS attack [93] or pulsing attack [111], takes advantage of the time-out mechanism of the TCP protocol to create illusory congestions. By sending bursts at a high-pulse data rate while keeping a low average data rate, these attacks can throttle the throughput of legitimate TCP flows to as low as 10% of normal bandwidth usage, and last indefinitely until detected [39, 41].

Analyzing the power spectral density (PSD) of monitored traffic can be an efficient way to detect DDoS attacks. PSD describes the power distribution of a signal in frequency domain. By counting arriving network packets during fixed intervals, a sequence of data points is collected over a specific time interval. Treating the data sequence as a random process, the PSD of traffic flow can be obtained through the calculation and subsequent conversion of its autocorrelation results to frequency domain. The PSD difference between real-time-monitored traffic flows and their normal statistical energy distribution exposes concealed malicious activities without the need of deep packet payload inspection [33].

Several research efforts have been reported that use PSD analysis for the de-

tection and containment of DDoS attacks. Cheng *et al* [45] first reported the concept of utilizing PSD of network flow to detect general TCP SYN flood. Chen *et al* [37,41] proposed using PSD of network flow to detect shrew attacks. Hashim *et al* [74] further extended PSD analysis-based detection to the next generation mobile network (NGMN). They claimed that their proposed algorithm can accurately classify traffic as normal, DoS and DDoS. He *et al* [76] proposed a method for remote detection of bottleneck links, which is applicable to the detection of any network congestion, including DDoS attacks and Internet worms in theory.

However, a major challenge that hampers the application of these approaches is at the lack of an appropriate data converter that can meet intensive computing requirements for real-time PSD data processing. Software solutions are incapable of coping with high data rates, which naturally leads to pursuing hardware solutions.

Possessing the flexibility of software and high parallelism of hardware, reconfigurable hardware devices, such as Field Programmable Gate Arrays (FPGAs), have been widely applied for many network security applications, including: protocol wrapping, packet classification, and intrusion detection [34]. The growing number of FPGA Intellectual Property (IP) modules not only makes it easy for the implementation of diverse DSP functions to circuits, but provides excellent design balance between high performance and lower power consumption. Given these considerations, the FPGA is the obvious choice for implementation of this PSD converter.

In this chapter, an optimized FPGA based real-time PSD converter is proposed, which converts data of interest into frequency domain for analysis. An innovative embedded converter is able to close the gap between supply and demand of real-time PSD data. Taking advantage of processing continuous data sequences with partial overlap, the component-reusable Auto-Correlation (AC) algorithm is capable of significantly reducing computing workload in comparison with that of the conventional AC algorithm. In conjunction with the employment of an FFT IP core, the

adapted $2N$ -point real-valued Discrete Fourier Transform (DFT) algorithm not only reduces the length of the operational data sequence to half, but fully utilizes the dual input channels of the FFT core to achieve an optimized design. For performance comparison, a design based on the conventional approach was also implemented.

The remainder of this chapter is structured as follows. In Section 5.1, the overall system architecture and algorithms are described. Section 5.2 depicts further implementation details. Evaluation results are then explained in Section 5.3. After discussing some development issues in Section 5.4, Section 5.5 summarizes this chapter.

5.1 System Architecture and Algorithms

5.1.1 System Architecture

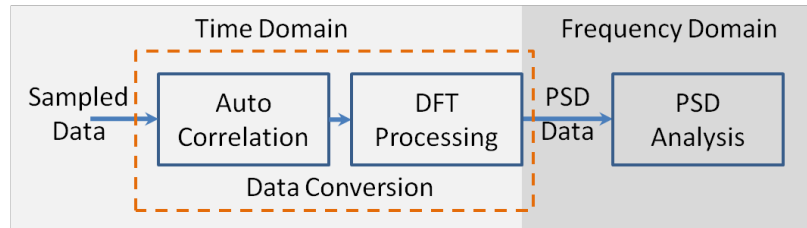


Figure 5.1. Data conversion steps.

Figure 5.1 shows the overall architecture of the embedded data conversion accelerator, as outlined by the dashed box. It consists of two major modules: AC (Autocorrelation) and DFT (Discrete Fourier Transform). The AC module takes a batch of sampled data and calculates its auto-correlation, i.e.: the similarity to its previous batch. The DFT module converts this AC result from time domain to frequency domain for PSD analysis. One can treat the number of arriving packets within a fixed time slot as a random process:

$$\{x(t)|t = n \cdot \Delta, n \in N\} \quad (5.1)$$

where Δ is a constant time-interval, N is the set of natural numbers and n is a set of positive integers. $x(t)$ is a random variable that refers to the number of packets sampled within a unit time interval at time t , i.e.: $(t-\Delta, t]$. If we consider Δ is a unit time slot, Eq. (5.1) could be reduced to:

$$\{x(t)|t = n, n \in N\} \quad (5.2)$$

Also, $x(n)$ represents the total number of packets sampled at the n^{th} sampling period. The sampling process is assumed as a Wide-Sense Stationary (WSS) random process. The AC function is defined as:

$$A(m) = \frac{1}{L-m} \sum_{n=0}^{L-m-1} [x(n) \cdot x(n+m)] \quad (5.3)$$

where $m = 0, 1, 2, \dots, L-1$.

L refers to the length of the data sequence, i.e.: the total number of sample points participating in a batch of the AC process. $A(m)$ is the AC with a shift of m points from the n^{th} sampling point, where $\{0 \leq m < L\}$. After AC processing, the DFT is applied to convert the intermediate data to the frequency domain for PSD analysis by:

$$Y(k) = \sum_{m=0}^{L-1} A(m) \cdot W_L^{km} \quad (5.4)$$

where $W_L^{km} = e^{-j\frac{2\pi km}{L}}$, $k = 0, 1, 2, \dots, L-1$.

Eq. (5.4) gives a standard DFT formula and more detail can be found in [133]. As shown by Eq. (5.3) and Eq. (5.4), the data conversion process is computationally intensive. To reduce the delay, a hardware-based approach is introduced, which is

based our innovative component-reusable AC algorithm and the Adapted 2N-point Real-valued DFT algorithm.

5.1.2 Component-Reusable AC Algorithm

According to Eq. (5.3), the essential part of the AC process is the convolution between $x(n)$ and $x(n+m)$. With a specified m , it takes $(L-m)$ multiplications per round to calculate the corresponding $A(m)$, as shown in Figure 5.2. To achieve a full batch of $A(m)$ with $\{0 \leq m < L\}$, a fixed number of multiplications are required, as:

$$\sum [L + (L - 1) + \dots + 2 + 1] = \frac{L \cdot (L + 1)}{2} \quad (5.5)$$

For instance, in Figure 5.2, it is assumed that the length of the operational data sequence (L) is 10 and the current shift (m) is 2. The convolution result of the i^{th} batch is:

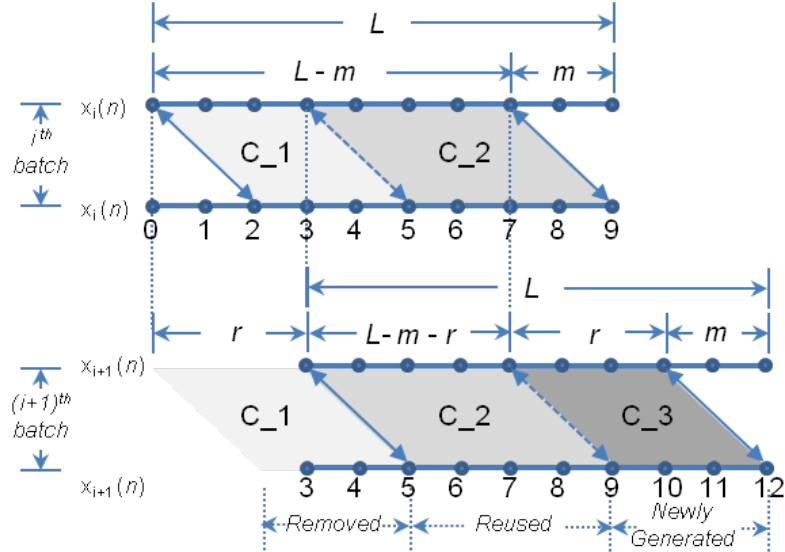


Figure 5.2. Convolution with reusable parts.

$$C_{x(n) \cdot (n+m)} = x(0) \cdot x(2) + x(1) \cdot x(3) + \dots + x(7) \cdot x(9) \quad (5.6)$$

which includes 8 multiplications per round as represented by the area combining C_1 and C_2 in the figure.

In practice, the workload of AC process on a continuous data series can be reduced if convolution results are reusable. As long as partial existing data points remain available for the following batched AC process, their convolution results can be reused, so as to reduce the overall computational complexity. To describe the replacement of data points for the continuous AC process, a new parameter r is introduced.

As illustrated in Figure 5.2, the replacement (r) is 3 for the $(i + 1)^{th}$ batch's convolution. The three oldest data points are removed and the same number of new data points is appended to keep the same data length (L) for operation. Moving from the i^{th} batch, the operational data points are now from 3 to 12. The current convolution result is represented by the combined area of C_2 and C_3. C_1 is excluded, since the corresponding old points have been removed. Obviously, C_2, which was generated in the i^{th} batch, can be reused as partial current result to avoid recalculation. Only C_3 requires calculation, which is 3 multiplications in this case.

A further study reveals that, when $L - m \geq r$, results of $(L - m - r)$ rounds of multiplication can be reused for the calculation of the next batch's $A(m)$. In terms of the number of multiplications, that is:

$$P_{reusable} = \sum_{m=0}^{L-1} (L - r - m) = \frac{(L - r) \cdot (L - r + 1)}{2} \quad (5.7)$$

Only r new multiplications need to be calculated, as illustrated in Figure 5.3. However, when $L - m < r$, $(L - m)$ multiplications need to be calculated, as illustrated in Figure 3. Therefore, the total number of multiplications required for calculation of a full batch AC process is:

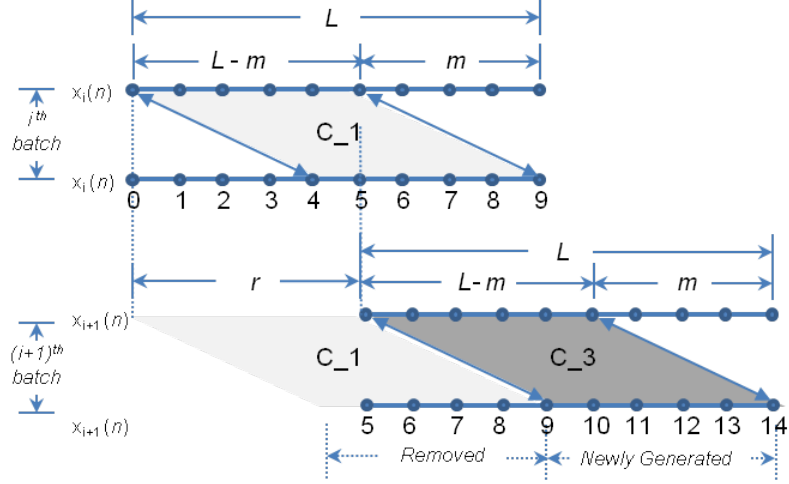


Figure 5.3. Convolution without reusable parts.

$$\begin{aligned}
 P_{total} &= \sum_{m=0}^{L-r} (r) + \sum_{m=L-r+1}^{L-1} (L-m) \\
 &= r \cdot (L-r+1) + \frac{(r-1) \cdot (r)}{2} \\
 &= \left(L + \frac{1}{2}\right) \cdot r - \frac{r^2}{2}
 \end{aligned} \tag{5.8}$$

When $r \geq L$, since all the old data points are replaced, P_{total} reaches its maximum, as:

$$P_{total_max} = \left(L + \frac{1}{2}\right) \cdot L - \frac{L^2}{2} = \frac{L \cdot (L+1)}{2} \tag{5.9}$$

This equation also applies to the initial batch's data sequence for AC processing, since all the data points are new. In fact, $P_{reusable}$ is the difference between P_{total_max} and P_{total} given $0 \leq r \leq L$.

According to Eq. (5.8) and Eq. (5.9), P_{total} calculated on top of a continuous data series is determined by L and r ; and its upper bound is only determined by L . Figure 5.4 gives an intuitive demonstration of their relationship. The replacement rate R ($R = r/L$) is introduced for the evaluation of the impact of this component-

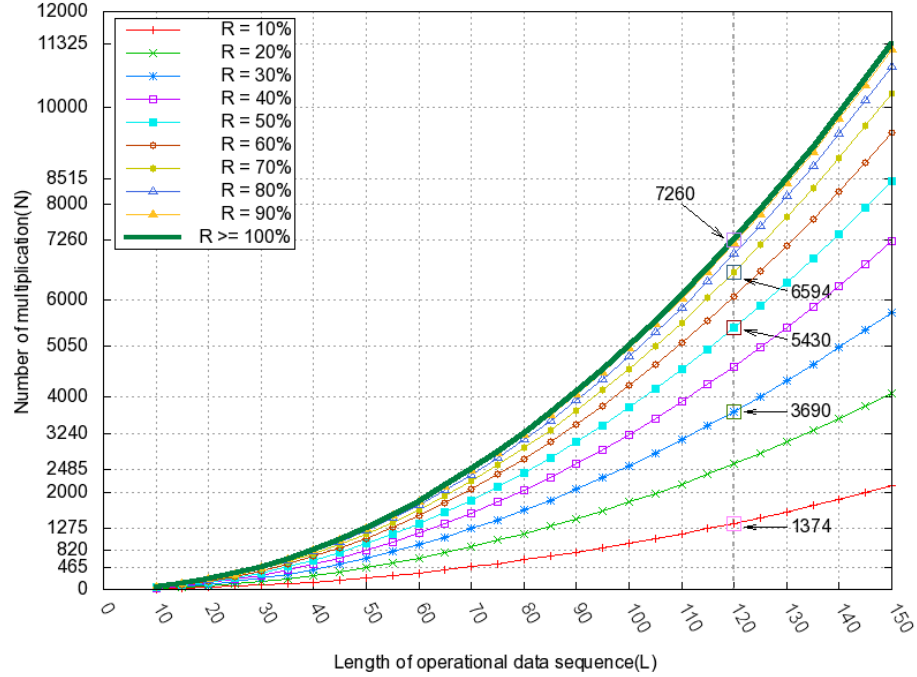


Figure 5.4. The impact of sequence length L replacement rate to the computational complexity of multiplication.

reusable AC algorithm on the overall computational complexity. In Figure 5.4, the X -axis represents the length of data sequence (L), and Y -axis represents the number of required multiplication operations (N). Ten curves are presented that illustrate the relationship between L and P_{total} under different R values, bottom up from 10% to 100%, respectively. For L ranging from 10 to 150, the impact of different R_s to the number of required multiplications N is presented. Without any reusable part, i.e.: $R \geq 100$, multiplication is applied to all the available data points, which is given by the upper curve. This curve overlaps the results of $P_{total.max}$ from Eq. (5.9), which also validates our algorithm.

With reusable part, N depends on the percentage of reuse. The smaller R is, the less the N is, since more elements are reusable for AC computation. Consider a series of N values when $L = 120$, for example. As labeled on the plot, N is 1374 when $R = 10\%$, and it goes to 3690 when $R = 30\%$. In comparison with $N = 7260$ when $R = 100$, approximately 81% and 50% multiplications are saved, respectively,

when R is within 10-30%. However, N grows much faster with the increase of R . It rises to 5430 when R goes to 50%, and becomes much closer to the maximum N when R crosses 70%. Considering the corresponding overhead, it is more inappropriate to adopt this algorithm with high R for AC processing.

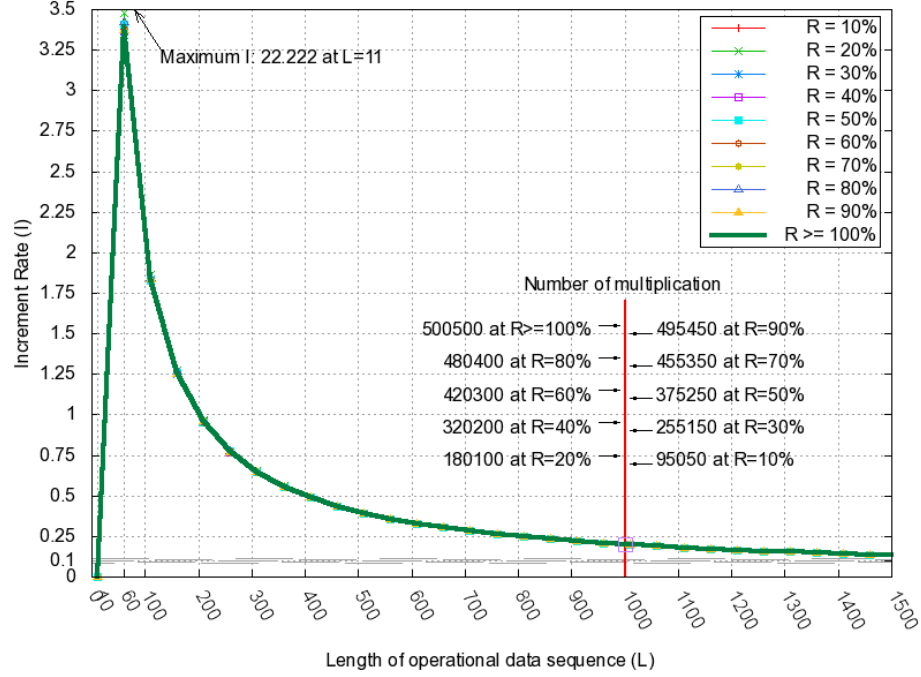


Figure 5.5. The trend of increment rate for multiplication.

Figure 5.5 further evaluates our component-reusable AC algorithm in terms of the increment rate (I) of N with respect to L . The X -axis represents the length of data sequence (L), with a scale from 0 to 1500. The Y -axis represents the increment rate (I) which is defined as:

$$I(l+1) = \frac{\Delta N}{N(l)} = \frac{N(l+1) - N(l)}{N(l)} \quad (5.10)$$

where l and $l+1$ refer to L on two adjacent stages, respectively. $\Delta N = N(l+1) - N(l)$, which refers to the increment of N between two adjacent stages.

Corresponding to different R values, ten curves are plotted illustrating the trends of I with the increase of L . Starting from $L = 10$, all curves achieve their

peaks at $L = 11$ with I at approximately 20. The maximum vertex is 22.222 for $R = 30\%$. In fact, $I(11)$ is the first available I according to Eq. (5.10). Then, the curves decrease quickly and finally trend to 0.1 based on the evaluation of L up to 1500 points. Despite wide variation of N , curves under different R values show high consistency in I as illustrated in Figure 5.4.

From the above analysis, although our component-reusable AC algorithm does not optimize I in terms of R , it decreases the value of N significantly with the increase of L . For $L = 1000$, a saving of 81% N could be achieved in comparison with different R values, at $R = 10\%$ versus $R = 100\%$. However, determining the right R for a perfect fit is application-dependent. On the other hand, it affects the efficiency of using this algorithm.

5.1.3 Adapted $2N$ -point Real-valued DFT algorithm

After AC processing, the operational data sequence $\{A(m)|0 \leq m \leq (L - 1)\}$ from Eq. (5.3) needs to be converted from the time to frequency domain. The DFT is employed for this conversion. However, the implementation of DFT conversion in hardware is complicated and time-consuming. Employing existing Intelligent Property (IP) cores for this purpose is more practical than developing own modules in FPGA design. It can not only facilitate prototyping, but also achieve optimized performance and resource utilization on the targeted platform.

Applying a Fast Fourier Transform (FFT) IP core is a common approach for implementing a DFT function in an FPGA. Since the processing of general purpose DFT is based on complex-valued data, the Xilinx FFT IP core features dual-channel input for taking both parts of a complex number simultaneously, i.e.: one channel dedicated for real part, and the other for imaginary part. However, the data sequence output from $A(m)$ is real-valued only. Fetching them directly to the FFT IP core for processing leaves the input channel for imaginary-valued data unused. By fetching

the real-valued data sequence into both input channels, not only can the throughput of DFT processing be doubled, but the operation timing of the FFT IP core can be cut due to the length reduction of the operational sequence. Based on this idea and inspired from [23, 114, 133], an adapted $2N$ -point real-valued DFT algorithm is proposed.

This algorithm only applies to the data sequence with even length, i.e.: $L = 2N$. The basic idea is to split the $2N$ -point real data sequence to form an N -point complex data sequence for regular FFT processing. Then, the results of the other half are reconstructed by application of the symmetry conjugate property of complex numbers, rather than calculation. The following description outlines this procedure:

1. Decimate the real data sequence $A(m) | 0 \leq m \leq 2N - 1$ into two parts for composing a complex data sequence with length N .

$$x(n) = A(2 \cdot n) + JA(2 \cdot n + 1) \quad (5.11)$$

where $n = 0, 1, 2, \dots, N - 1$.

2. Perform an FFT operation on the N -point sequence. Since the FPGA IP core is employed, this step is considered as a black-box process.

$$X(k) = FFT\{x(n)\} \quad (5.12)$$

where $n, k = 0, 1, 2, \dots, N - 1$.

3. Calculate the first N -point results of $Y(k)$ by:

$$Y(k) = M_A(k) \cdot X(k) + M_B(k) \cdot X^*(N - k) \quad (5.13)$$

where M_A and M_B are pre-calculated values for lookup.

$$\begin{cases} M_A(k) = \frac{1}{2}(1 - j \cdot W_{2N}^k) \\ M_B(k) = \frac{1}{2}(1 + j \cdot W_{2N}^k) \end{cases} \quad (5.14)$$

where $k = 0, 1, 2, \dots, N - 1$.

4. Obtain the remaining N -point result of $Y(k)$ by using symmetry conjugate property of complex numbers:

$$Y(2N - k) = Y^*(k) \quad , \text{ where } k = 0, 1, 2, \dots, N - 1.$$

and

$$\begin{cases} Y_r(k) = X_r(0) - X_i(0) \\ Y_i(k) = 0 \end{cases} \quad , \text{ where } k = N. \quad (5.15)$$

5. Complete a batch of DFT conversion for $A(m)$ with $L = 2N$. The results should be the same as that obtained through direct processing via $2N$ -point DFT:

$$Y(k) = DFT\{A(m)\} \quad (5.16)$$

where $k, m = 0, 1, 2, \dots, 2N - 1$.

The benefit of this approach is two-fold. Not only is the dual-channel input of the IP core fully utilized, but the workload of essential FFT processing is also cut by at least half. Considering the complexity of the FFT operation for a length L data sequence is $O(L \cdot \log(L))$, a sequence with length $N = L/2$ is $O((L/2) \cdot \log(L/2))$. The reduction rate (R) is:

$$R = 1 - \frac{\frac{L}{2} \cdot \log \frac{L}{2}}{L \cdot \log L} \cdot 100\% = \frac{1}{2} \cdot (1 + \log_L^2) \cdot 100\% \quad (5.17)$$

which represents how much the original work load is reduced.

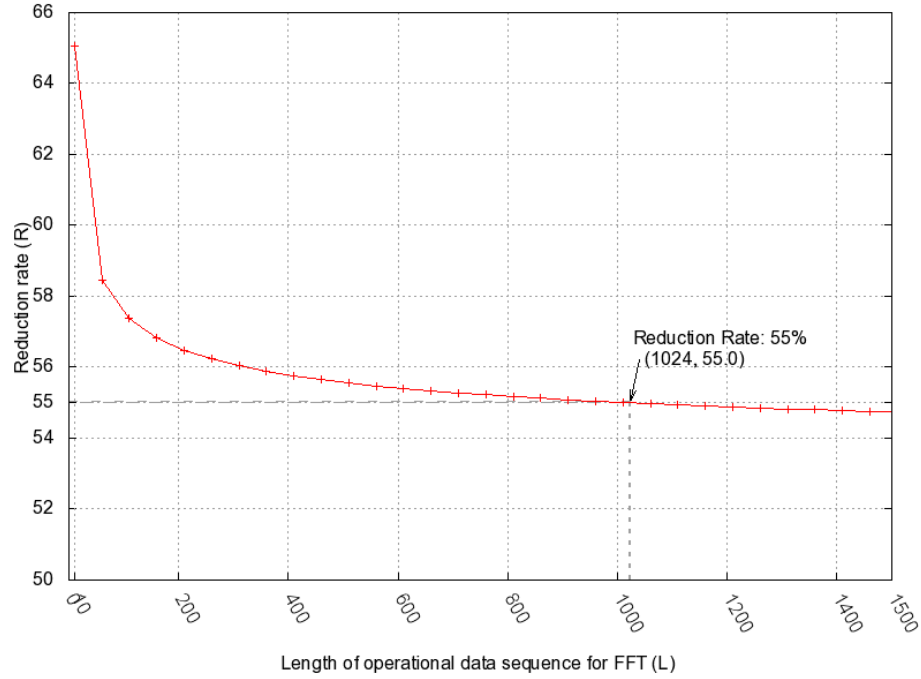


Figure 5.6. The reduction trend of FFT workload.

Figure 5.6 gives an intuitive demonstration. The X -axis shows the length (L) of data sequence for FFT operation ranging from 10 to 1500, and the Y -axis is the reduction rate (R). For a sequence with $L = 1024$, a reduction of 55% the original FFT workload can be achieved. The reduction approaches nearly half as L increases further.

5.2 Implementation

5.2.1 Design Overview

The block diagrams in Figure 5.7 further refine the data conversion process presented in Figure 5.1. Figure 5.7(a) illustrates a conventional architecture. Besides the AC and FFT processing blocks, memory blocks are introduced for data storage so as to achieve parallel execution. Memory M1 holds the input data sequence,

memory M2 the intermediate results from the AC process, and memory M3 and M4 the real and imaginary parts of processed DFT data, respectively. The depth of these memories is set to $2N$, the length of the operational data sequence.

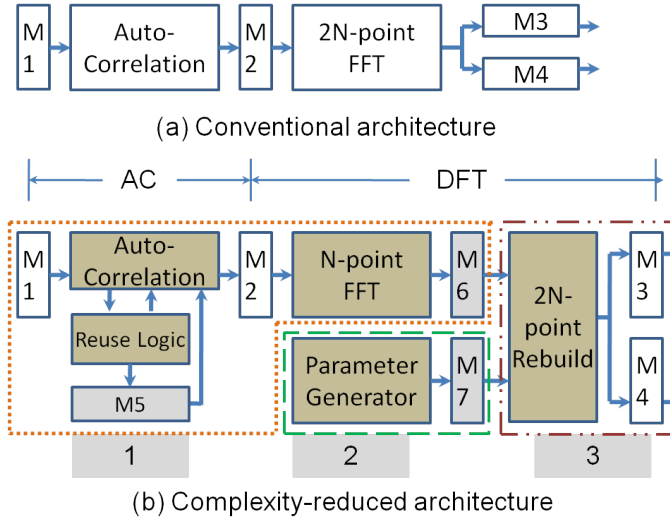


Figure 5.7. Refined architectures for PSD data conversion.

Figure 5.7(b) illustrates our innovative data conversion architecture. In comparison with the previous architecture, both the AC and DFT blocks are modified substantially. In the AC part, the reuse logic and its corresponding M5 for holding reusable data are added. In the DFT part, the direct $2N$ -point FFT is replaced by the process of the N -point FFT plus the $2N$ -point rebuild. The intermediate data from the FFT are stored in M6, while the static parameters are stored in M7. The process of $2N$ -point rebuilding starts when both data sets are available.

Based on the data traveling through the conversion process, the overall design is organized into three parts, as outlined in Figure 5.7(b). The first part includes the AC process and the following N -point FFT process. It starts with reading the data from M1 and ends with writing the data to M6. The second part is the parameter generating process, which consists of the parameter generator and M7. The generator only runs during initialization: after all the necessary parameters are produced and stored in M7 for subsequent access, it sleeps until reset. The third part consists of

the remaining components that take data from both M6 and M7 for $2N$ -point rebuild and write to M3 and M4. Therefore, Part 1 and Part 3 set up the main data path.

5.2.2 Design Implementation

The essential goal of this design is to reduce the volume of necessary computation workload. Besides implementing Eq. (5.3) for AC calculation, Part 1 emphasizes this goal with the involvement of additional memory. Figure 5.8 presents the implementation of Part 1. This diagram consists of memory blocks, IP cores and Finite State Machine (FSM) logic. For a clear view of the data path, all the FSM logic is concentrated to a solid bar standing in the middle of the diagram. Memory blocks are initialized by M. The two Xilinx IP cores are for division and FFT operations, respectively.

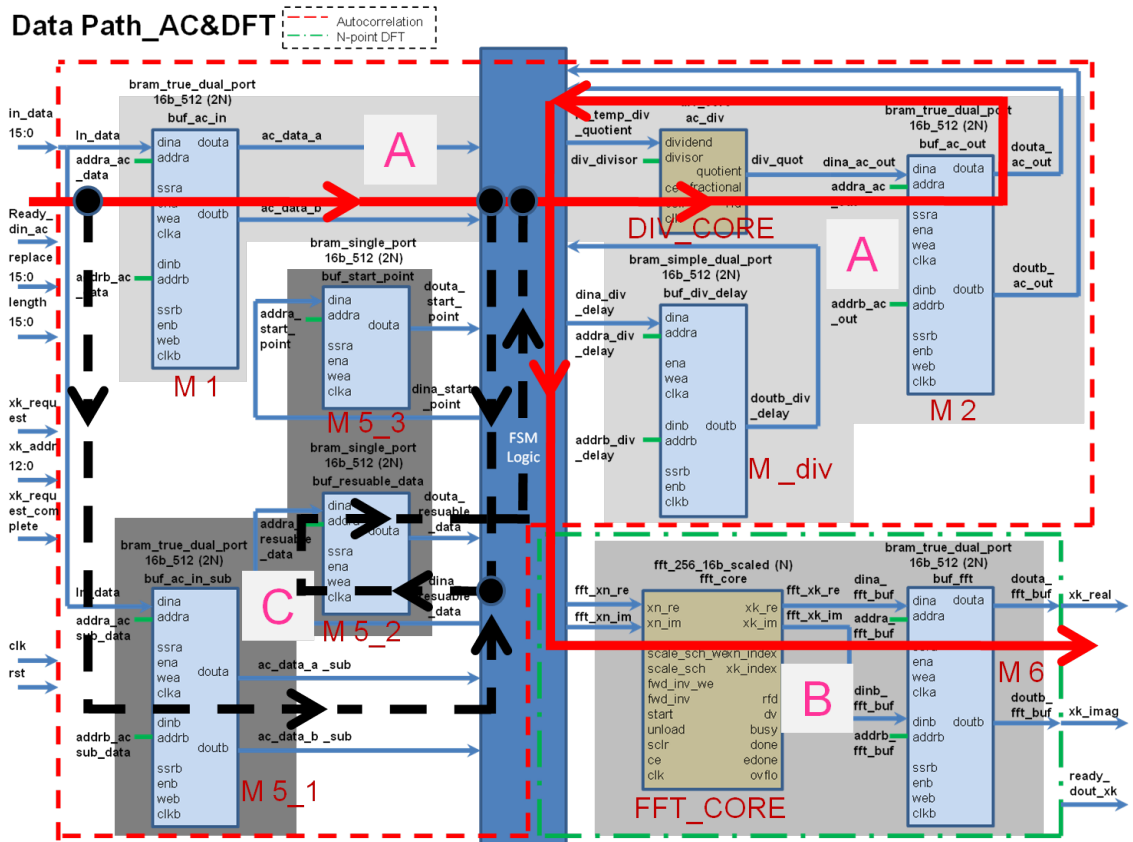


Figure 5.8. Block diagram for autocorrelation and FFT process.

In terms of functionality, the block diagram of Figure 5.8 can be divided into three areas, as indicated with different grey scale backgrounds. Area A performs the AC function. Since the division IP core takes delay for quotient output and the arrival of operands may vary from time to time, M_div is employed to solve the timing issue of quotient receiving by storing correct time intervals between arrival operands. Area B conducts the FFT function. The length of input operational sequence is pre-configured in the FFT IP core and the number of output is doubled in terms of input. These two areas are the foundation of data conversion process. The arrow that travels through the red bold line identifies the main data path.

Area C dedicates the workload reduction of AC process. It consists of three memory blocks. M5_2 is the key component of holding reusable data. M5_3 stores the insertion points of reusable data for the calculation of new AC results. M5_1 stores a duplication of M1 for expediting the generation of reusable data. The arrow that travels through the black dashed line identifies the data path of reusable data.

The design of the parameter generation block is straightforward, accomplished by implementing Eq. (5.14) and storing the results to M7 as indicated in Figure 5.7(b). A key step is applying sine-cosine values for the production of M_A and M_B . With careful exploration of equation characteristics, a simplified design is achieved. With the replacement of twiddle factor W_{2N}^k by sinusoids through Eulers formula, Eq. (5.14) can be rewritten as:

$$\left\{ \begin{array}{l} M_A(k) = \frac{1}{2}[1 - j \cdot (\cos(\frac{\pi}{N}k) - j \cdot \sin(\frac{\pi}{N}k))] \\ \quad = \frac{1}{2}[1 - \sin(\frac{\pi}{N}k) - j \cdot \cos(\frac{\pi}{N}k)] \\ M_B(k) = \frac{1}{2}[1 + j \cdot (\cos(\frac{\pi}{N}k) - j \cdot \sin(\frac{\pi}{N}k))] \\ \quad = \frac{1}{2}[1 + \sin(\frac{\pi}{N}k) + j \cdot \cos(\frac{\pi}{N}k)] \end{array} \right. \quad (5.18)$$

where $k = 0, 1, 2, \dots, N - 1$

Since hardware cannot directly support complex algorithms, it is helpful to

express them separately as real and imaginary parts for implementation.

$$\begin{cases} M_{A,r}(k) = \frac{1}{2}(1 - \sin(\frac{\pi}{N}k)) \\ M_{A,i}(k) = -\frac{1}{2}\cos(\frac{\pi}{N}k) \\ M_{B,r}(k) = \frac{1}{2}(1 + \sin(\frac{\pi}{N}k)) \\ M_{B,i}(k) = \frac{1}{2}\cos(\frac{\pi}{N}k) \end{cases}, \text{ where } k = 0, 1, 2, \dots, N-1 \quad (5.19)$$

The range of k in the above equations with respect to $\sin(\frac{\pi}{N}k)$, which is actually $\sin(\frac{2\pi}{2N}k)$, is $k = 0, 1, 2, \dots, N-1$. This implies that only half number of sinusoid values are required, that is N out of $2N$ points. In addition, taking the advantage of triangular conversion formula,

$$\begin{cases} \sin(x + \frac{\pi}{2}) = \cos x \\ \cos(x + \frac{\pi}{2}) = -\sin x \end{cases}, \text{ where } 0 \leq x \leq \frac{\pi}{2} \quad (5.20)$$

the following equations are obtained:

$$\begin{cases} M_{A,r}(k + \frac{\pi}{2}) = \frac{1}{2}(1 - \cos(\frac{\pi}{N}k)) \\ M_{A,i}(k + \frac{\pi}{2}) = \frac{1}{2}\sin(\frac{\pi}{N}k) \\ M_{B,r}(k + \frac{\pi}{2}) = \frac{1}{2}(1 + \cos(\frac{\pi}{N}k)) \\ M_{B,i}(k + \frac{\pi}{2}) = -\frac{1}{2}\sin(\frac{\pi}{N}k) \end{cases}, \text{ where } k = 0, 1, 2, \dots, N/2 - 1 \quad (5.21)$$

Therefore, the required number of sinusoid values can be further cut by half. With only having the first quarter's sinusoid values, i.e. for $k = 0, 1, 2, \dots, N/2 - 1$, two N -point parameter vectors M_A and M_B are able to be calculated and stored in M7, respectively. Figure 5.9 shows the data path of this procedure. A Sine/Cosine IP core is employed to generate the number of $N/2$ sinusoid values as above. Through

the FSM logic which executes function of Eq. (5.19) and Eq. (5.21), the number of N -point real and imaginary parts of M_A are generated and stored in M7_1, while both parts of M_B are stored in M7_2.

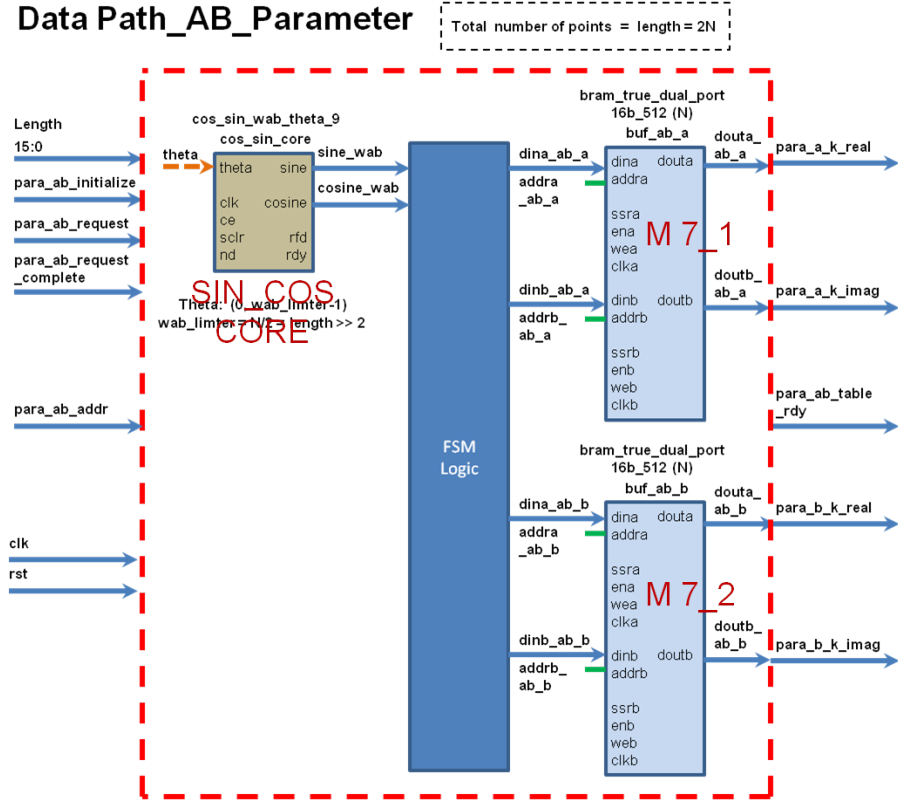


Figure 5.9. Block diagram for parameter generating process.

Once one batch of data from the N -point FFT process is ready in M6, the $2N$ -point DFT results are rebuilt in conjunction with parameters stored in M7. This process fulfills the operation of Eq (5.13) and Eq (5.15). Similar to what was done above, Eq (5.13) is decomposed to real and imaginary part for processing, as:

$$\left\{ \begin{array}{l} Y_r(k) = X_r(k) \cdot M_{A_r}(k) - X_i(k) \cdot M_{A_i}(k) \\ \quad + X_r(N - k) \cdot M_{B_r}(k) + X_i(N - k) \cdot M_{B_i}(k) \\ Y_i(k) = X_i(k) \cdot M_{A_r}(k) + X_r(k) \cdot M_{A_i}(k) \\ \quad + X_i(N - k) \cdot M_{B_i}(k) - X_i(N - k) \cdot M_{B_r}(k) \end{array} \right. \quad (5.22)$$

where $k = 0, 1, 2, \dots, N - 1$.

And, further decomposition of Eq (5.15) to:

$$\begin{cases} Y_r(2N - k) = Y_r(k) \\ Y_i(2N - k) = -Y_i(k) \end{cases}, \text{ where } k = 0, 1, 2, \dots, N - 1. \quad (5.23)$$

$$\begin{cases} Y_r(k) = X_r(0) - X_i(0) \\ Y_i(k) = 0 \end{cases}, \text{ where } k = N.$$

With Eq (5.22) and Eq (5.23), $2N$ -point DFT values are calculated and finally stored in M3 and M4 as real and imaginary parts, respectively. It completes the function of Part 3 as depicted in Figure 5.7(b).

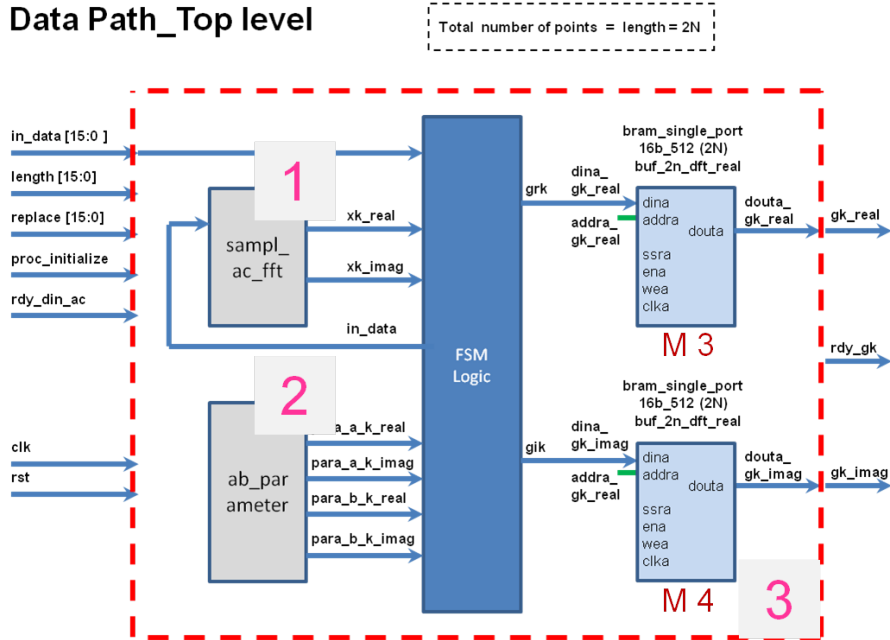


Figure 5.10. Block diagram for final results generating process.

Figure 5.10 depicts the data path of this process, which also abstracts the top level of our design. Three major parts are included corresponding to Figure 5.7, where block_1 is detailed in Figure 5.8 and block_2 is detailed in Figure 5.9. The red dotted line outlines the block_3 which outputs the final results.

5.3 Experimental Evaluation

5.3.1 Experimental Setup

The implementation of this data converter targets on a Xilinx Virtex-2 Pro XC2VP50 FPGA, with Xilinx ISE Design Suite 10.1 on Redhat Enterprise Linux 5.8 OS as its development environment. As part of this effort on defensive infrastructure for network security [45], this data converter is designed to have the capability of processing data at Gigabit network rate. According to Sinha *et al.*'s study in [151], the size of internet packets varies from 40 to 1500 *Bytes* and its distribution is strongly bimodal at both ends with 40% and 20%, respectively. Theoretically, the expected maximum throughput of Gigabit network is 217 *byte/ms*, which corresponds to 88 to 3277 *packets/ms* according to the packet sizes indicated above. The 16-bit input data bus is capable of accepting data at rates up to 64 *k*, which is enough to accommodate the number of incoming packets collected in a time interval of 20 *ms*.

The network traffic embeded with Shrew DDoS attacks concentrates its energy on the low frequency area less than 50 *Hz*, while healthy traffic shows a much wider energy distribution [39, 41]. For a clear distinction of DDoS attacks in frequency domain, it is necessary to set the sampling rate of incoming network traffic greater than 50 *Hz*. Setting it to 512 *Hz* extends the observable spectral range 9 *times* wider than that of 50 *Hz*, which enables an accurate PSD analysis. Converting this frequency rate back to time domain yields 1.95 *ms* per sample. For a data sequence with $L = 512$, it takes one second to refresh all its data points, where each data point represents the number of incoming packets counted during a time interval of 1.95 *ms*.

On top of the above understanding, two sets of evaluation were conducted. Both of them followed the same procedure but with different parameter settings. The first set ran on short length's data sequences which makes it easy for function

verification. The second set ran on full length's sequences which makes the timing assessment more meaningful. From the valuation point of view, they complement each other.

5.3.2 Function Verification

The first set of evaluation runs on data sequences with length $L = 16$, and the replacement $r = 3$, i.e. $R = 0.19$. In conjunction with this case, the simulation procedure is described as follows:

1. After initialization, Ma and Mb parameters are first generated and stored in M7_1 and M7_2 as depicted in Figure 9.
2. Assuming an external sampler takes the responsibility of data collection and rearrangement. When the specified numbers of data points are all ready, a handshake signal is sent out to our converter.
3. When the handshake signal is on for the first time, it indicates that the initial data sequence is ready for in-put. In this case, it is a fresh 16-point long data sequence.
4. The converter accepts the data to M1 and stores the processed results to M3 and M4. It is assumed that the data conversion process always completes in time before the following data sequence is ready.
5. When the handshake signal is active for the rest time, this indicates that the next data sequence is ready. The length of data sequence still remains the same as the initial 16, but only 4 data points are replaced in this case. From timing point of view, this implies that the maximum available time for processing a 16-point long data sequence is the time interval of sampling 4 new data points.
6. Repeat Step (5) for 3 times and terminate the simulation.

Table 5.1. Results comparison of AC-DFT process between theoretical expectation and simulation output.

No.		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
INPUT	Pre	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
	Cur	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
OUT.r	G_r	4708	7	32	39	40	40	41	41	40	41	41	40	40	39	32	7
	G_{rr}	4708	8	33	38	40	39	41	40	40	40	41	40	40	39	32	7
	G_{rk}	4708	6	32	39	40	38	40	39	40	39	40	38	40	39	32	6
OUT.i	G_i	0	-214	-103	-61	-42	-27	-17	-8	0	8	17	27	42	61	103	214
	G_{ii}	0	-212	-102	-61	-42	-26	-18	-8	0	8	18	26	42	61	102	212
	G_{ik}	0	-215	-105	-62	-42	-28	-19	-8	0	8	19	28	42	62	105	215

For result verification, Matlab code for standard AC-DFT process according to Eq (5.3) and Eq (5.4), as well as for our improved algorithms are also developed for assistance. Table 5.1 gives an example showing the results coming out of the same input but through different verification methods. The index of a data sequence is given in the first row, which implicates $L = 16$. In Input section, the upper and the lower lines show the previous and the current data sequences, respectively. The current sequence is from 10 to 25, a shift of 3 points from its predecessor.

In the Out section, results from three verification sources are listed, which consist of real (r) and imaginary (i) parts. G_r and G_i are results directly from the equation approach mentioned above, while G_{rr} and G_{ii} are results from our improved approach. Both of them are obtained from the execution of corresponding Matlab code. G_{rk} and G_{ik} are results from ModelSim simulation where the functionality of our hardware design for this improved approach is verified. The comparison between two Matlab verification methods turns out much similar results, which proves the correctness of our improved algorithm theoretically. The further comparison between theoretical and simulation results verify the functionality of our design. The small difference between them lies in the data trimming during hardware operation, which is acceptable.

5.3.3 Simulation Comparison

Simulation provides an intuitive timing assessment of processing full length data sequences with $L = 512$. The replacement r was set to 100, roughly $R = 0.2$. With clock frequency at 5 MHz for simulation, our improved approach takes 102.4 μs to store the initial 512-points data sequence to M1, according to Figure 5.7(b). The AC process then takes 26,679.0 μs for operation until storing the intermediate data to M2. The 256-point FFT process takes 329.0 μs from retrieving the data in M2 to storing the processed data to M6. The final 512-point DFT rebuilding process takes 307.6 μs to complete the entire batch of data conversion and store the output to M3 and M4. The time it takes for the production and storage of Ma and Mb parameters according to Eq (5.21) to M7 is 51.8 μs . Since this step runs independently at the initial stage, it has no impact with the delay on the main data path. The overall processing delay for the initial sequence is 27,418.0 μs .

For a fair comparison, a hardware design of the conventional approach was also developed. As shown in Figure 5.7, all necessary modules required by the conventional approach in Figure 5.7(a) are all included in our improved approach in Figure 5.7(b). The only difference is the length of operational sequences between $2N$ -point FFT and N -point FFT modules. This could be easily covered through parameter alteration without additional logic modification. Taking the advantage of modularized design, and adapting our improved approach to the conventional approach is relatively straightforward. Meanwhile, the shared base of both designs makes the evaluation results more convincing.

The graphs in Figure 5.11 illustrate the measured results from both our improved approach as well as the conventional approach depicted in Figure 5.7(a) and Figure 5.7(b), respectively. The left bar in each set represents the processing time under the conventional approach, while the remaining two bars represent that under our

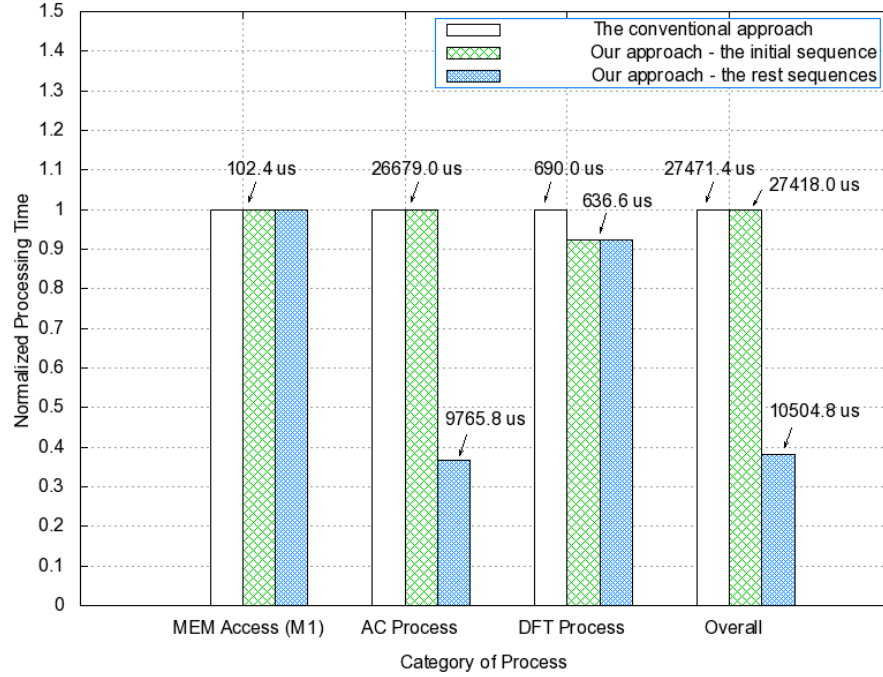


Figure 5.11. Timing analysis of simulation results between the improved approach and the conventional approach.

improved approach. The middle bar indicates the processing time for the initial data sequence, and the right bar represents the processing time for the following sequences. To be readable, the processing time is normalized, with the original measured values indicated on the top of each bar for reference.

The first (leftmost) set depicts the processing time of memory access to M1 for writing the input data sequence. It takes 102.4 *us* and there is no difference between both approaches. The second set shows the delay of the AC process. Taking the conventional approach according to Eq (5.3), the same amount time, 26679.0 *us*, always applies to the processing of each batch's data sequence. With the component-reusable AC algorithm, this amount of time only applied to the processing of the initial batch's data sequence. When reusable data is available for the processing of the following sequences, only 9,765.8 *us* is required to process. A significant delay reduction of 63.4% is achieved.

The third set represents the delay for DFT processing. The processing time

of direct 512-point DFT conversion according to Eq (5.1) is 690.0 *us*, while this approach combining 256-point FFT plus 512-point DFT rebuild takes 636.6 *us* for completion. A saving of 7.7% is obtained. Since the DFT processing time is only equivalent to 2.6% or 6.5% of the AC processing time with respect to the conventional or our approach, respectively. This reduction has trivial contribution to the overall processing time which is concluded in the fourth histogram. Overall, a maximum reduction of 61.8% processing time from 27471.4 *us* to 10504.8 *us* is achieved in this instance.

According to the above timing analysis, it is the AC process that dominates the delay of data conversion. More than 97.3% of the overall processing time is taken for this purpose, in terms of our improved approach. On the other hand, the adapted $2N$ -point real-valued DFT conversion does not show much time saving on the DFT process. However, the avoidance of another 256 points participating in direct FFT conversion does relieve the workload as analyzed in Figure 5.6, which cannot be simply evaluated through timing analysis.

5.3.4 Synthesis Analysis

After simulation, the design for the improved approach was synthesized for the assessment of resource utilization as well as more accurate timing report. Table 5.2 lists the summary of device utilization. Though the available resources that the Xilinx Virtex-2 Pro FPGA can provide are conservative in comparison with the latest Virtex family devices, it is still spacious enough to accommodate our design according to this summary. The timing report indicates that the maximum frequency of this board at Speed Grade: Virtex-7 is 91.893 *MHz*. This means that all evaluation results from the above simulation with 5 *MHz* clock frequency could be sped up roughly 17 *times*. Considering the clock frequency at 90 *MHz*, the overall processing delay for the initial sequence is 1.523 *ms*, and the delay from the subsequent sequences is 583.6

Table 5.2. Device utilization summary.

Device Name	Used Amount	Total Amount	Utilization Rate
Slices	2711	23616	11%
Slice Flip Flops	2901	47232	6%
LUTs-Overall	4162	47232	8%
LUTs-Logic	4081	4162	98%
LUTs-ShiftReg.	81	4162	2%
IOs	52		
Bonded IOBs	52	812	6%
BRAMs	14	232	6%
Mult18X18s	23	232	5%
GCLKs	1	16	6%

us. Therefore, the entire conversion process can be done during sampling of one data point at 1.95 *ms* in this case.

5.4 Discussion

Although this data converter is developed for the Shrew DDoS attack detection system, it could be applied to any application requiring real-time PSD data. Modularized design and IP core utilization enable easy adaptation. During development, it was noticed that even the definition of auto-correlation varies slightly depending on application background. Thus, logic modification on one functional block without impacting others is highly preferred. Segmenting individual modules is thus a good solution to both support design reuse and facilitate modification. The utilization of IP cores aids in managing data storage requirements as well as enabling FFT processing for new applications. With simple parameter adjustment, such as depth/length of B-RAM and FFT cores, a desired memory and FFT size can be achieved. In conjunction with a modularized interface, the customization of data converter can be completed efficiently.

An important reason of choosing Xilinx Virtex-2 Pro FPGA as the target

device is at the intention of using available NetFPGA-1G [106] development board. With four Gigabit Ethernet Network Interface Controllers (NICs) and FPGA chips integrated together, the NetFPGA board seamlessly bridges the network traffic to digital signal processing. Meanwhile, many relative open source packages are available supporting for diverse applications. Configuring a NetFPGA to be an open core gigabit router is one of the most popular applications. Implementing a data converter on top of this platform further expands its adaptability for network related applications.

To apply this converter for real practice, a NetFPGA-based IP packet counter is also developed. This counter is capable of counting any packet header information at the TCP/IP layer, which serves as the sampler collecting data from the real world for following conversion. For more design and implementation details of this counter, please refer to Appendix B. Though the available resources of a single Xilinx Virtex-2 Pro FPGA limit the implementation of both counter and converter together, an alternative approach of cascading multiple NetFPGA devices may work it out. The development of this IP counter bridges the gap of my study from theory to reality.

5.5 Summary

In this chapter, a FPGA based real-time Power Spectral Density (PSD) converter for Shrew DDoS attack detection has been proposed. The innovative component-reusable AC algorithm and the adapted $2N$ -point real-valued DFT algorithm are the foundation of this work. With the inclusion of the component-reusable AC algorithm, the computing burden of AC processing on top of partially overlapped data sequences is significantly reduced. Both theoretical analysis and experimental study demonstrate the benefit of our approach in comparison with the conventional approach.

Chapter 6

Pattern Decomposition Method for Signature-based Detection

In past decades, the performance gap between processing requirements of Network Intrusion Detection Systems (NIDS-s) and their software-based implementations has been widened, due to the escalation of sophisticated attack tools and performance limitations of sequential execution. A common solution to close this performance gap is through hardware implementation of security functions. The major motivation is to pursue more powerful computing capability for execution of more sophisticated security functions in real time. In addition, through exploration of unique features of hardware execution style, such as multi-threaded parallelism and multi-stage pipelines, further improvements could be achieved.

Packet inspection is one of the most important tasks performed by NIDS-s. It consists of two parts: packet classification and deep packet inspection. The former focuses on the packet header and later examines the payload. Due to diverse contents of packet payloads, deep packet inspection is more complex, so as to attract more research efforts in this field.

Pattern matching or signature detection is a fundamental technique for deep packet inspection. By comparing input data with predefined signature patterns, ma-

licious content can be accurately identified. Actually, more than 80% of the Snort rules contain signature patterns and more than 80% of CPU time is taken by pattern match operations [113].

Snort [141] is a well-known software based security application originally designed for lightweight network intrusion detection. Unlike commercial NIDS-s, in which core parts are hidden for intellectual property protection, Snort is a free open-source NIDS. Its rule database is used to generate regular expressions for intrusion detection. Users are allowed to strengthen the rule database as well as other parts of Snort, and verified rules are collected for updating its core rule database. Over more than a decade's evaluation, the Snort rule database has earned a good reputation for its accuracy, comprehensiveness and efficiency.

With the trend of using hardware-based applications for security protection in high speed network environments, continuously growing rule databases have become a significant impediment to hardware implementation. Nowadays, hardware solutions are likely to be integrated on a single chip with tight resource constraints. Along with many advantages, the consequent side-effect is that hardware implementations are more resource-sensitive. Without deliberate design methodologies, hardware resources can be quickly exhausted.

At the same time, it is highly desirable that a NIDS can update its rule database in real time, so that newly emerging attacks can be handled promptly. However, this issue is still an open problem, since hardware design is not as flexible as software programming. Modification in reconfigurable hardware implies replacement and rerouting of circuits. In addition, many realistic conditions that are seldom considered in software implementations must be taken into account in a hardware implementation, such as signal delay, fan-in/fan-out and power consumption.

Therefore, it would be ideal if a NIDS could be implemented in a lightweight manner, so that the signature database neither occupies huge memory space nor

increases infinitely. Furthermore, the update operation should be done in real time and online, which implies that adding new signatures should not lead to significant replacement and rerouting efforts.

Signature detection can vary from being as simple as checking the value of a single header field to as complex as calculating the statistical characteristics of a connection or conducting sophisticated protocol analysis. Essential signatures reveal activities that an intruder has to perform to gain access into a computer system. These activities include, but are not limited to, launching programs, running scripts, querying databases and following the steps of a protocol. They can be broken down into a set of operations, and signatures describe sequences of these operations.

Since intruders must interface through computers, their operation sequences should be input sequences that computers can accept. In practice, basic input sequences are limited, defining a finite set of allowable activities. Any sophisticated input sequence could be disassembled to basic sequences. If an input sequence could be mapped to a signature, a basic sequence could be mapped to a primary pattern. Hence, no matter how large the signature database grows, the number of primary patterns which represent fundamental operations is limited.

This observation implies that the total number of primary patterns is limited if each primary pattern corresponds to an operation. If we can represent signatures as permutations of these primary patterns, the memory or hardware resources required to characterize primary patterns appearing in the signature database can be significantly reduced.

By analyzing signature pattern sets extracted from the Snort database, we verified this rationale as we observed that many primary patterns appear repeatedly within signatures. Even when new signatures are added, many of them can be decomposed into the existing set of primary patterns, with infrequent additions. Practically, this implies that implementation of these primary patterns can be fulfilled

with a smaller amount of memory or other hardware resources.

Since all patterns are distilled from network traffic carrying data for common protocols and services, it is reasonable to believe that this observation is not only applicable to the pattern set abstracted from the Snort rule database that we studied, but also applicable to pattern sets from other pattern-match based NIDS-s as well. Though the appearance of signature pattern sets may be different because of diverse environments, the primary pattern sets should be similar.

The remainder of this chapter is organized as follows: Section 6.1 briefly introduces reported work close to our effort. Section 6.2 illustrates our observations and presents a two-step approach to decompose the Snort signature/rule database. Section 6.3 verifies our observations through detailed analysis. Section 6.4 discusses some basic principles of our current ongoing work based on the decomposed Snort signatures. Finally, Section 6.5 summarizes this chapter.

6.1 Related Work

To the best of our knowledge, it was Hutchings *et al.* who first attempted implementing patterns from the Snort rule database on top of reconfigurable hardware [79]. Since their primary focus was on hardware adaptation, many other important issues were not deliberately considered. For example, they just mentioned the scalability issue of patterns due to the growing size of Snort rule database, but did not give details. From the perspective of implementation, they only considered that hardware circuits can be pre-compiled and the compiling time is not a big issue, since patterns are predefined, but they did not consider on-line update.

Facing today's rapid evolution of network security requirements, on-line updates of signature databases are crucial to maintain reliance of any signature-detection based NIDS. As to reconfigurable hardware applications, recompilation is one of the

integrated procedures for update. High-efficiency recompilation makes it possible to achieve on-line updates. up to now, design innovations based on a variety of techniques have been applied to hardware-based pattern matching approaches. However, they have generally suffered from limited hardware resources, especially the tight budget of memory resources to store patterns or perform pattern comparison.

Cho *et al.* [47] proposed to relieve the scalability issue of pattern comparison and to improve the performance by using 8-to-1 decoders for each byte comparator and retaining one decoder output for all the duplicated decoders to reduce redundancy. The 8-to-1 character decoder is set to the first stage of the pipeline. Before conducting a comparison, input data at the character level is decoded to a single bit level; all comparisons are then performed based on these single bits. By this means, they reduced the overall size of the comparators to one-eighth. In circuit design, reducing the number of fan-in also reduces the side effect to gate. Secondly, since all comparators use the same data input and many decoders are exactly the same, instantiating one output for sharing saves hardware resources. To achieve more efficient operation, patterns are divided into prefix and suffix parts, and the prefix part was used to index a major portion of patterns contained in suffix part. While the prefix part was kept on-chip, the suffix part was kept on an off-chip ROM. It was obvious that the size of on-chip RAM and the bandwidth in-between memories has great impact on the system performance.

There are reported efforts that focus on the implementation of regular expression pattern matching engine [17, 156]. By using Non-deterministic Finite Automata (NFA), instead of centralizing on character decoder, they employed the SRL 16 module [51], as well as explored certain common prefix sharing techniques. Good experimental results were achieved, containing 500 IDS regular expressions from Snort in 25K logic cells. However, the scalability issue would be prominent with larger numbers of expressions being implemented.

Aldwairi *et al.* [4] developed a configurable string matching accelerator to speed up the deep packet inspection. They executed software on a general purpose processor for Finite State Machine (FSM) operation with the support of standard RAM. The software generates a FSM from patterns extracted from the Snort rule database, and the FSM is in charge of pattern matching operation. However, both generation and operation of the FSM for pattern matching would become more complicated as the number of patterns increases. In addition, although they tried to increase throughput by increasing on-chip RAM bandwidth, there is a limit to how much bandwidth can be increased.

Instead of focusing on pattern comparison for performance improvement, researchers also worked on other approaches to improve performance by taking advantage of some properties in network traffic [8, 154]. They indicated that malicious packets make up only a small share of total traffic. Consequently, they adopted hybrid architectures in which hardware devices handle pre-filtering and PC-based software implements Snort for final identification. It was reported that more than 90% of the workload of traffic processing could be relieved from Snort software through this approach. In addition, this design could sustain more than 10Gbps throughput compared to 1Gbps Snort throughput. Compared with the other approaches, this approach is more flexible and has good scalability. However, the problem of maintaining 1Gbps throughput of software-based Snort is not trivial.

Another approach [10] that is close to our effort attracts our attention because of its good performance on both feasibility and scalability achieved by partitioning Snort patterns. While many hardware-based pattern matching applications focus more on hardware innovation, their investigation focused more deeply into the original pattern sets. The graph-based “min-cut” technique [90] was applied to help achieve optimal design. The basic idea of min-cut partition is that the number of edges between nodes within the group is maximized, and the number of edges between

different groups is minimized. After partitioning, optimized pattern structures are implemented in hardware for pattern matching operation. In fact, this is the only work we know of that focuses on the analysis of relations concealed in Snort patterns to improve the performance of corresponding hardware processing and to solve scalability issues.

6.2 Decomposition of SNORT Patterns

Most efforts to improve the performance of hardware-based pattern matching implementations focus on the innovation of hardware architectures or matching algorithms. However, these complexity and scalability issues root in the inflation of signature pattern sets. Therefore, gaining a deeper insight into these signature pattern sets before further hardware exploration may bring an alternative view angle for innovation.

This study reveals that the most significant factor causing the size inflation of Snort signature pattern sets is the existence of internal redundancies. Successful removing of these redundancies can significantly reduce the size of signature pattern sets, so as to relieve the complexity and scalability issues of subsequent implementation. A two-step decomposition method proposed in this section is dedicated to this purpose.

6.2.1 Redundancies in SNORT Patterns Sets

The signature patterns used in this study are extracted from the Snort rule database. In addition to these patterns, other packet information is also included in the database for detection and administrative purposes, such as: source/destination addresses, source/destination ports, protocols, flags and tags. In general, a complete Snort rule consists of two logic sections, the rule header and the rule options [141].

Figure 6.1 shows a Snort signature. The text up to the first parenthesis is the rule header portion, which includes basic packet header information with specific IP addresses replaced by wildcards. The second portion within the parentheses is the rule options, which includes alert messages, pre-identified signature patterns, and other support information for Snort NIDS.

```
alert tcp $EXTERNAL_NET any -> $SQL_SERVERS 1433
(msg:"MS-SQL shellcode attempt"; flow:to_server,established;
content:"9 |D0 00 92 01 C2 00|R|00|U|00|9 |EC 00|";
classtype:shellcode-detect; sid:691; rev:5;)
```

Figure 6.1. An example of a Snort rule.

Within this rule, the data string being quoted after the “content” is the signature pattern that is sought. “Content” works as a keyword in the Snort rules database, introducing the specific signature patterns for deep packet inspection [141]. An example of such a signature pattern is shown in Figure 6.2. In this case, the pattern is composed of both text characters and binary data. The binary data is represented in hex format and enclosed within pipe characters (|).

```
9 |D0 00 92 01 C2 00|R|00|U|00|9 |EC 00|
```

Figure 6.2. A Sample of Snort signature pattern.

For convenience, one can take a group of signature patterns extracted from the Snort rule database as an example to study its properties. This work calls such a group of signature patterns a signature pattern set. This set shows two important properties: repetition and composition.

1. **Repetition:** As shown in Figure 6.3, signature patterns contained in 3 rules out of 6 are the same. This redundancy implies that increasing the number of signatures may not necessarily lead to a commensurate increase in the number of primary patterns.

2. **Composition:** As shown in Figure 6.4, each signature can be considered as a combination of smaller pattern fragments, or primary patterns. Thus, a signature pattern can be decomposed into one or more primary patterns.

```
|5C|PIPE|5C 00 05 00 0B|  
|98 D0 FF|k|12 A1 10|6|98|3F|C3 F8|~4Z  
|05 00 0B|  
|98 D0 FF|k|12 A1 10|6|98|3F|C3 F8|~4Z  
|04 00|  
|98 D0 FF|k|12 A1 10|6|98|3F|C3 F8|~4Z
```

Figure 6.3. Repetition of patterns.

```
dbms_repcat.alter_priority_raw  
dbms_repcat.alter_priority  
dbms_repcat.alter_priority_varchar2  
dbms_repcat.alter_site_priority_site  
dbms_repcat.alter_site_priority
```

Figure 6.4. Composition of primary patterns.

The presence of repetition and composition implies the existence of redundancy in this pattern set. Instead of passing these redundancies to corresponding hardware operations, or applying more sophisticated mechanisms for redundancy mitigation at the hardware design level, it is meaningful to remove these redundancies up front at the time when the pattern set is generated.

Once we carefully peel off these redundancies, not only the complexity of hardware design can be reduced, but also the scalability of the system can be improved. Then, we can maintain the same functionality as the original design with

fewer patterns implemented in real circuits. Obviously, this approach relieves the conflict between the growing size of signature databases and the limited hardware storage resources.

6.2.2 Pattern Decomposition

Based on the above observation, we decompose the abstracted pattern set further to remove redundancy. The decomposition consists of two steps which correspond to the properties of repetition and composition, respectively. The first step, dealing with repetition in the pattern set, is straightforward. It checks all individual signature patterns inside the pattern set in turn and simply removes repeated patterns. The second step, handling composition, involves further decomposing signature patterns into primary patterns. A primary pattern is a substring of a signature pattern. This step is more difficult since there are many ways one can parse and decompose individual signature patterns into primary patterns.

The most challenging issue in the second step of deep decomposition is balancing the tradeoff between system performance and redundancy. Signature patterns can be decomposed into primary patterns with various lengths, and ranging from single characters up to entire signature patterns. It is naive and inefficient to decompose the signature patterns down to the character level. Although the total number of characters used, and hence the number of primary patterns, will never exceed the number of ASCII codes (256, with even fewer used in practice [10]), such fine-grained decomposition leads to tremendous overhead in system performance while performing dynamic pattern matching since it is extremely inefficient to reconstruct signature patterns from individual characters.

Pattern decomposition on text-based patterns at the “word” level takes advantage of text-based redundancies exhibited in signature sets, such as those shown in Figure 6.4. Although pattern decomposition could also be performed on binary data

patterns, such as those shown in Figure 6.3, methods of delineating primary patterns are not as immediately obvious. Since our primary goal is to verify the correctness and feasibility of the proposed approach, this preliminary analysis of decomposition biased on text dominant patterns is enough.

For decomposition, choose the following heuristic. A string of adjacent characters not containing any “hyphen symbol” between any two characters is considered as a primary pattern, as are the hyphen symbols by themselves. Taking signature patterns in Figure 6.4 for example, “*dbms*”, “*repcat*”, “*alter*”, and “*priority*” are all considered as primary patterns and “_” and “.” are considered as “hyphen symbols”. In general, any symbol in between two character strings could be considered as “hyphen symbol”; we have chosen underscore (‘_’), hyphen (‘-’), period (‘.’) and space (‘ ’) and forward slash (‘/’) characters to be hyphen symbols. Specifically, in the case when patterns form combinations of both text characters and binary data, any binary data string in between pairs of “|” is considered as an individual primary pattern. Considering the patterns in Figure 6.3 for example, “|5C |”, “PIPE ”, “|5C 00 05 00 0B |” are all considered as individual primary patterns. It worth noting that “|” itself is not a part of patterns in practice. Instead, it is used only as a mark to distinguish between text characters and binary data.

Figure 6.5 shows the change in the number of pattern entities resulting from the two-step decomposition method for various categories of Snort rules, while Figure 6.6 shows the change in total storage size. The signature patterns are taken from Snort V.2.3.3 and divided into 48 categories. To maintain a clear view, pattern categories that contain less than 50 entities are excluded from Figure 6.5 and those that contain less than 500 characters total are excluded from Figure 6.6. The complete figures are included in our technical report [36].

Within each category, the graphs show the original number of signature entities values (before two-step processing), the number after redundancy removal and the

number of primary patterns after subsequent deep decomposition. To highlight the change within each category, rather than variation cross categories, all values are normalized to the original count (Figure 6.5) or size (Figure 6.6) of the signature patterns in that category. Though there is a wide variation in the number of pattern entities and the total size of different categories, results may not be that evident after normalization. The rightmost category in each figure provides an overview combining results from all categories in this data set, respectively.

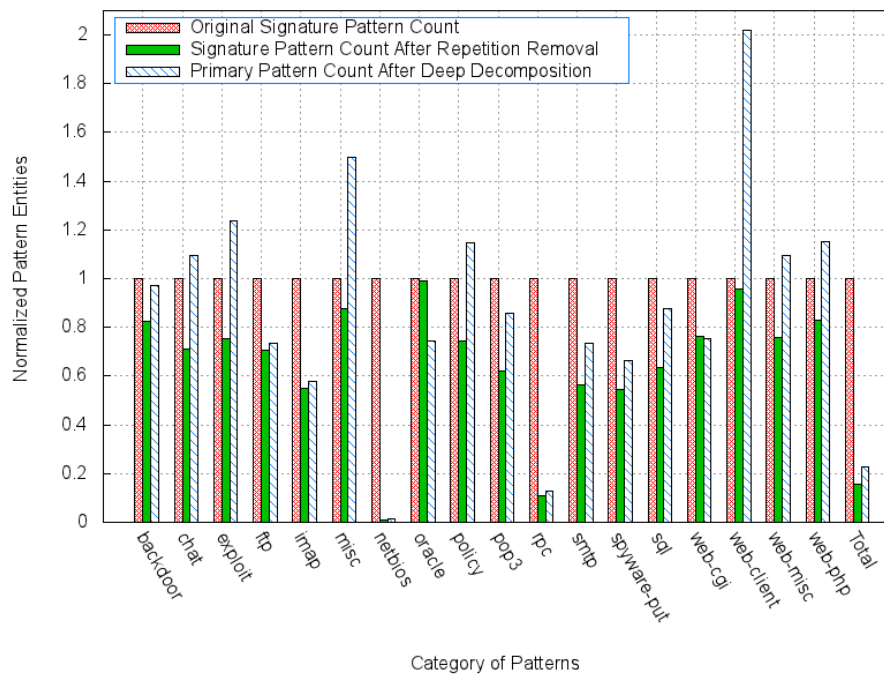


Figure 6.5. Variation of pattern entities.

For nearly every signature pattern category in Figure 6.5, there is a significant reduction in the number of signature patterns after redundancy removal, which suggest a high rate of reuse of entire signature pattern entities. One of the extreme cases in Figure 6.5 is the *netbios* rule category, for which the original number of signature entities (20,087) is reduced to 148 after repetition removal. The *Oracle* and *web-client* rule sets have the lowest redundancy of the pattern categories. The cumulative results for the entire data set show that there are a total of 25,802 signature pattern

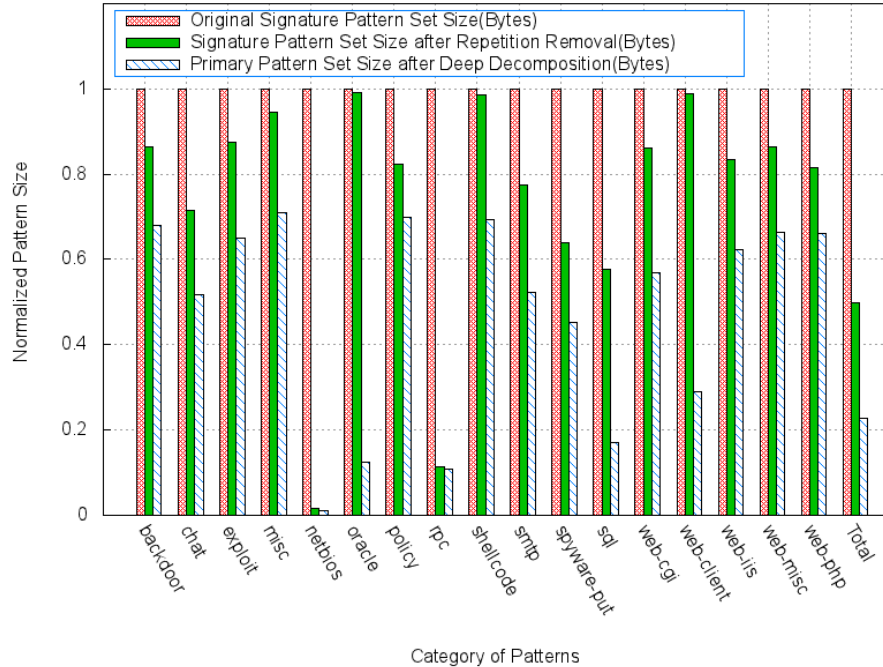


Figure 6.6. Variation of pattern sizes.

entities in the database, only 4076 of which are unique. However, the number decreases to 164 entities and 1355 bytes after deep pattern decomposition, respectively. In Figure 6.6, one observes a decrease in size due to repetition removal that is based on the amount of reuse and size of the patterns. The aggregate reduction in size for all pattern categories is from 145,348 bytes down to 72,315 bytes, a reduction of just over 50%.

After subsequent deep decomposition, the pattern entity count representing the number of unique primary patterns presents. Thus, one would expect the count to increase. Since each pattern is broken down into substrings, and each pattern itself is unique, both may contribute something new to the total. In the case of the *oracle* and *web-cgi* categories, there is a decrease in the number of pattern entities. This suggests the signature patterns are formed from permutations of a smaller set of primary patterns. Even more important than entity count is the size of the resulting primary pattern set (32,898 bytes), which represents a further decrease of 55% after

repetition removal, resulting in a total decrease of over 77% for the two step processing method.

6.3 Further Analysis and Verification

The fundamental motivation of deep decomposition is that signature patterns are correlated to activities that an intruder must perform in order to gain access to a computer system. An activity may consist of multiple atomic operations that are mapped to certain primary patterns contained in a signature pattern. Depending on the service being attacked, the activities of an intruder may exhibit a wide range of diverse data formats and/or functionalities; yet for each service, the atomic operations should be limited in practice to the set of operations (inputs) that are recognized by the computer software implementing the service. This set is further limited by the activities that will benefit the attacker. Since the number of services running on a practical machine is expected to be finite, the total number of primary patterns should not exceed a finite upper bound. After reaching some practical size, the number of primary patterns is unlikely to increase significantly with incremental increases in the number of signature patterns in the database.

Assuming that it is feasible to reconstruct signature patterns from a limited number of primary patterns, the end result should be smaller, more efficient hardware implementations. We consider issues related to recomposing signature patterns in Section 6.3.1. To verify the premise that the number of primary patterns is bounded in practice and that this bound is not based on signature database size, we analyze the incremental growth in primary patterns in Section 6.3.2.

6.3.1 Signature Pattern Reconstruction

To benefit from the reduced size of the primary pattern set, it is critical to find an efficient approach to express and reconstruct the original signature patterns from primary patterns. Such an approach can overcome the conflict between the growing size of signature databases and limited hardware resources, since newly added signature patterns can be decomposed to and reconstructed from existing primary patterns. In another words, the implementation achieves scalability with respect to the number of signature patterns.

Compared to approaches that require more memory space to store new patterns [4, 17, 79, 156], our novel approach has the potential to achieve much better scalability. Compared to approaches that require complex hash functions for pattern condensing [59, 162], our approach is much simpler and more feasible for on-board application.

The problem of efficient reconstructing signature patterns based on primary patterns resulting from our two-step decomposition procedure is an area of further research. Considering each primary pattern as a single graph node, reconstructing a signature pattern is simply a process of forming a directed graph for each signature. As illustrated in Figure 6.7, “*dbms*”, “*repcat*”, “*alter*”, “*site*” and “*priority*” are five primary patterns obtained from examples of Figure 6.4.

Figure 6.7 also illustrates the pattern reconstruction phase, in which these primary patterns could be used to recover original signature patterns. Figure 6.7(a-c) shows how original signature patterns are reconstructed, while Figure 6.7(d) shows how a new signature pattern can be reconstructed from the same set of primary patterns. This is the essential feature that we are pursuing: increasing the variation of edges between nodes rather than increasing the number of nodes, and allowing different signature patterns to be easily reconstructed. Efficient methods for implementing

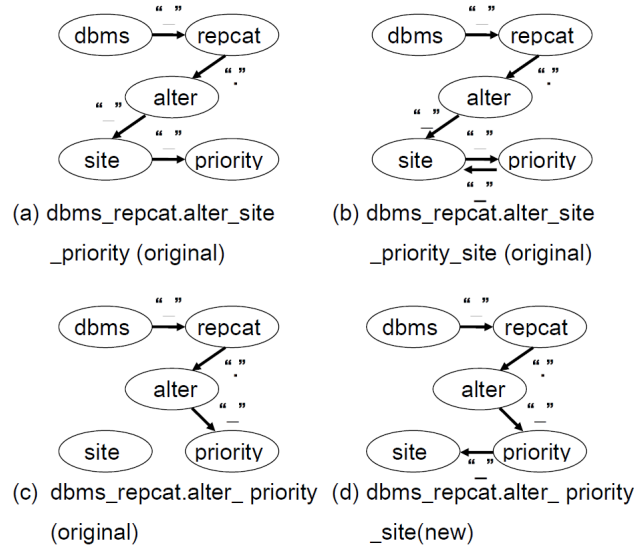


Figure 6.7. Signature Pattern Reconstruction.

this reconstruction in hardware are an area of further study.

6.3.2 Statistical Verification

In order to verify our premise that the number of primary patterns increases at a much slower rate compared to the rapid growth of signature pattern databases, we extend our analysis of pattern decomposition to determine variation across multiple versions of the Snort rule database. Since these versions represent the evolution of the Snort database over a period of several years, this experiment presents us a clear view on the relationship of between growth in the number of signature patterns and growth in primary patterns.

We analyze a series of existent signature pattern sets, representing seven major public versions of Snort rule databases. They are versions from V2.1 to V2.4 and V2.6 to V2.8, which were released over a period from April 2005 to October 2008. Although new types of rules were added to rule databases to accommodate the continuous evolution of network intrusion patterns, major parts of Snort rule databases still follow an accumulating update policy. From V2.1 to V2.2, V2.3 to V2.4 and V2.6 to

V2.8, the corresponding types of Snort rules were increased from 48 to 50, and later to 52. However, the sizes of rule database inflated from 935Kb to 8.86 Mb.

The number of pattern entities and the size of pattern sets are two key factors to be used for the analysis of the growth relation between signature patterns and primary patterns. By extracting the “content” signature pattern rules from the databases, we obtain the original signature pattern sets, summarized in Table 6.1. The size of the signature patterns grows from 31.1 KB in version 2.1 to 164.2 kB in version 2.8. We then further decompose each set of original signature patterns down to primary pattern sets following the two-step decomposition procedure presented in Section 6.2.

Table 6.1. Summary of parameters of signature pattern set and primary pattern set in different versions.

Version	Signature Pattern Set		Primary Pattern Set	
	Entities	Size	Entities	Size
2.1	2,739	31.1 KB	2,432	11.7 KB
2.2	3,442	33.0 KB	2,501	11.7 KB
2.3	25,802	145.3 KB	5,856	32.9 KB
2.4	26,936	147.9 KB	5,867	32.9 KB
2.6	31,103	164.1 KB	6,482	37.0 KB
2.7	31,097	163.9 KB	6,525	36.9 KB
2.8	31,156	164.2 KB	6,523	37.0 KB

The plot in Figure 8 illustrates the incremental trend in the size of the decomposed database as the number of signature pattern entities increases. From the most recent set of rules tested (version 2.8), the *netbios* content signature rule set was selected, since it is the largest set of rules.

Because the rules do not contain any timestamp indicating when they were added to the database, the rules were randomized to minimize the effect of grouping within the file. To generate the plot, one additional rule was added to the database

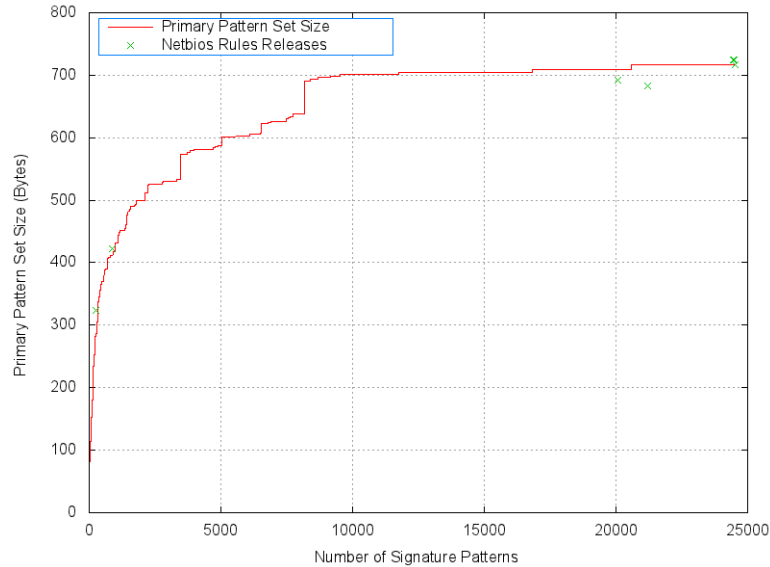


Figure 6.8. The growing trends of pattern entities between different versions.

at each step, and the size of the database after two-step decomposition was recorded. Thus, the graph shows the incremental growth in the primary pattern set size as the signature set size increases.

Clearly, as the number of signatures increases the growth of the primary pattern set size appears to be decreasing rapidly. The individual points on the plot show where the *netbios* (signature pattern set size, primary pattern set size) pairs fall on this plot.

Note that except for version 2.8, the points do not exactly fall on this curve since the rules were incrementally added in random order and some rules may differ between versions. However, the points do show good agreement, justifying the sampling procedure used to generate the plot.

While Figure 6.8 supports our premise that after reaching some practical size, the number of primary patterns saturates and is unlikely to increase significantly with incremental increases in the number of signature patterns in the database, we cannot say for certainty that this is indeed the case. Many of the signature pattern sets in the snort database are too small to have reached the point of saturation and the database

as a whole, being composed of many types of traffic, also have not yet reached the point of saturation.

Despite this, the signature pattern sets all exhibited sub-linear incremental growth when analyzed. Furthermore, even if the database never reaches the saturation point given changes in service types, the overall reduction in size achieved by our two-step decomposition procedure still justifies the use of static signature pattern decomposition and dynamic reconstruction.

6.4 Discussion

The main advantage of adopting the proposed decomposition and reconstruction based pattern processing technology is to achieve a good scalability. As we have successfully demonstrated a scalable way to reduce redundancy in signature pattern sets, the natural extension is to investigate scalable approaches for real-time dynamic updating and high-speed online signature matching. However, as this is ongoing work, only some basic principles are discussed in this section.

6.4.1 Scalable Pattern Update

Static update refers to the update mode that is pre-scheduled. Due to its easy operation, static update is the most conventional and popular mode used for the update of signature patterns. However, static update is not conducive to dynamic update requirements. Although manual operations could provide certain flexibility, their impact is limited. Hence, the efficiency of static update is questionable due to the increasing possibility of inaccurate results. Increasing the frequency of update is not an ideal solution. If the update is conducted too often, it causes unnecessary overhead to the system since there is not new pattern to be added. In contrast, it may be still not fast enough to handle a burst of signature update. A more flexible

and intelligent update scheme is expected to overcome these drawbacks. Therefore, events driving dynamic updates is desired.

As illustrated in previous sections, this approach has relieved the pressure on storage space requirements by decomposing signatures into primary patterns. When a new signature has been caught, the possible update process could be described in two sub-tasks as follows. First, adding new entry into the primary pattern set if new primary pattern appears. Second, storing relations among primary patterns into the relational database. Since new signature patterns do not necessarily lead to the increase of primary patterns, particularly when the set of primary patterns become “large”, a new signature more likely implies a new relation among existent primary patterns.

Unfortunately, it is non-trivial to describe the complex association relationships concisely and completely. One approach is simplifying the associated relations through graph clustering algorithms, which cluster nodes according to specific sharing features [110]. Hierarchical Dirichlet Process with Hidden Markov State [177] is a possible solution for further processing of these relations.

It is also a critical mission to develop a non-disruptive updating strategy, since the normal operation of an intrusion detection system should not be interrupted for signature updating. A delayed write strategy could be an option. While a new primary pattern is added, the system only updates the record of the bloom filter [21]. The pattern will be stored temporarily in a buffer, and written into the database later when the system is not so busy.

6.4.2 Dynamic Signature Matching

One advantage of the dynamic matching scheme is that we do not need to conduct the “matching” operation bit-by-bit as the current technologies. Once the incoming pattern has been decomposed, the existence of the primary patterns is

detected through the Bloom Filters in parallel. Indexing by the Bloom Filters, a set of “links” will be identified, which is corresponding to association relationships among those primary patterns.

Treating the Snort signature database as a complex graph, we can re-model the problem as a path-finding problem. A signature matching is finding an existent directional path going through numbers of nodes in the graph.

Based on the above principle, we are designing a two-phase fast parallel dynamic signature matching scheme that takes advantage of the decomposition operation. Similar to the dynamic scalable pattern update operation, first phase will decompose the investigating patterns into primary patterns. Then the Bloom Filters [21] checks whether these patterns match the signature primaries in the Snort signature sets. If the result indicates there are multiple attack/intrusion primary patterns, we will try to walk through those nodes in the graph to detect the existence of a path.

6.5 Summary

In this chapter, we proposed a novel two-step pattern decomposition scheme to remove hidden redundancy in the Snort signature database. Signature patterns are decomposed into primary patterns that can be stored along with their association relationships. Our approach has been validated through detailed analysis of multiple versions of the Snort signature database. In particular, our results suggest that increases in the number of signature patterns do not necessarily lead to commensurate increases in the number of primary patterns. This technique relieves the resource demands on hardware implementations. In addition, it is promising towards a self-adaptive network infrastructure.

As an extension from this work, we plan to work on a scalable dynamic pattern

update and pattern matching for real time distributed intrusion detection application. Section 6.4 has presented the rationale of our ongoing efforts. One of the major challenges is how to handle the complex association relationship in a concise but accurate manner. Successful solution of this problem would not only benefit signature pattern based network intrusion detection, but also benefit general complex pattern matching/updating applications.

Chapter 7

Conclusion

This dissertation focuses on securing the availability and reliability of network infrastructure resources. A systematic study towards developing a defensive network infrastructure with hardware involvement has been achieved. Though there still much work left for further development. Rather than consider the end of this dissertation as a completion, the author would prefer more to call it as a stage of success for a new commencement.

7.1 Contribution

According to Figure 1.1 in Chapter 1, the major contributions of this dissertation can be summarized as follows:

- Developed a three-layered network modeling platform for studying collaborative defense at the abstractive high level.
- Conducted a comprehensive survey on hardware-based technologies for network intrusion detection.
- Designed a FPGA-based Power Spectral Density (PSD) data converter for acceleration detection process in frequency domain.

- Explored the potential of decomposing signature patterns of malicious activities for a scalable storage in hardware implementation.

Originally developed for my coursework, a small-world based three-layered network modeling platform was built for studying two major collaborative defense schemes. Through modeling, I was able to concentrate on the trend of reaction for evaluation, without falling into details. Though no direct study was further carried out in terms of collaborative defense, a subset observation from this study directs me to explore more on network intrusion detection.

Combining brief descriptions with intensive case-studies, my survey provides a comprehensive view on mainstream technologies application on network intrusion detection, from basic principles to systematic architectures. This survey not only enhances my fundamental knowledge in the field of network intrusion detection, but offers a good reference to those who are interested in this subject.

As an anomaly detection approach working in the frequency domain, the Power Spectral Density (PSD) based analysis features a broad application range for detection, but at the cost of intensive computing burden for data conversion crossing domains. My FPGA-based PSD data converter dedicates for acceleration of this conversion. With the innovative component-reusable Auto-Correlation (AC) algorithm and the adapted $2N$ -point real valued Discrete Fourier Transform (DFT) algorithm, a maximum reduction of 61.8% processing time is reached. Meanwhile, a corresponding NetFPGA-based IP packet counter was also developed. It not only resolves the issue of connecting real data traffic into the developed converter for further evaluation, but also bridges the gap of my study from theory to reality.

The development of a two-stage decomposition approach aims at a scalable storage of signature patterns for hardware-based detection application. A experimental result shows that a reduction in size of over 77% can be achieved, when comparing the decomposed primary patterns with the original signature patterns abstracted

from the Snort rule database. In fact, a desired future goal is restoring signature patterns for fast detection. The idea of neuromorphic computing is a promising way for fulfillment.

Hard work yields fruitful output. Along the way of this study, I have also published more than ten relative academic papers, as listed in Appendix C. They represent the research community’s recognition to my work.

7.2 Future Work

Based on the current achievements of this dissertation, the consequent work could be connecting the developed IP packet counter to my PSD data converter for deep evaluation and further development of a PSD analyzer for real practice. Figure 7.1 depicts this future work. However, the issue of limited available resources is still a big concern. Though both the IP packet counter and the PSD data converter have been proven to be implemented on a single NetFPGA development board respectively, it may not be the case while integrating both together, according to the synthesis information.

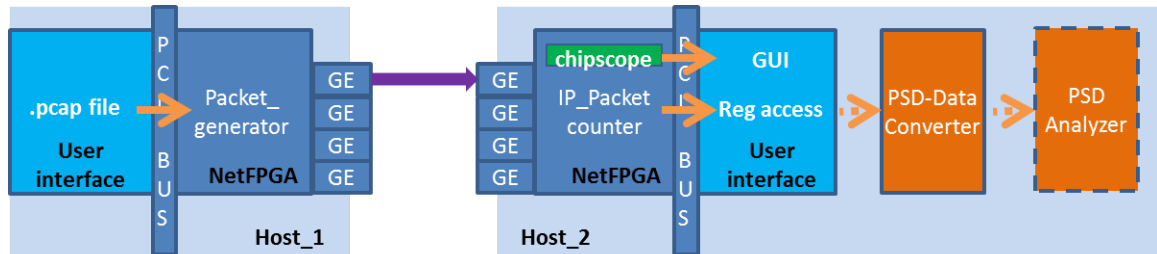


Figure 7.1. Connecting the IP packet counter to proposed detection approaches.

There are two ways to work it out. One is adopting the new generation’s NetFPGAs. It is the easiest choice if budget is allowed. The other is utilizing the cascading interface on each NetFPGA development board for co-work. Though this part looks a little bit far from the main topic, successful management of this issue

can significantly increase the flexibility of resource utilization, so as to expand the capability of adding more processing modules.

On top of the above step, a more ambitious goal is to extend the operation of detection from two network nodes to multiple nodes. With the developed network testbed as depicted in Figure B.11, we have the potential to achieve it. Adapting the small-world model developed for collaborative defense, it is practicable to build a small-scaled close-loop network environment for mimic evaluation of network intrusion detection, so as contribute to the development of a defensive network infrastructure. This could be a long-term goal.

Overall, serving as partial fulfillment of the requirements for the degree of Doctor of Philosophy, this dissertation systematically elaborates the author's research efforts *Toward Hardware-Oriented Defensive Network Infrastructure* in a comprehensive way. Developing effective and efficient approaches for network intrusion detection is the kernel of my study. With the involvement of reconfigurable hardware, it targeted on performance improvement and design innovation for bridging the gap of foreseeable limitation caused by pure software execution. Years of study on this topic have produced fruitful achievements, and will continuously drive towards additional contributions in the future.

Appendix A

Data Parsing of Real Network Trace

All plots and statistic results illustrated in this appendix are produced from the data set of "Witty Worm 2004" [26] which was originally collected on the UCSD Network Telescope [27] between March 19 (20:01:40 PST) ~ March 25, 2004 (25 00:01:40 PST). The Witty Worm is a type of malware that self-replicates and distributes copies of itself to its network. By randomly producing IP addresses to be targets of infection, the Witty Worm attempts to send maliciousness off to the target hosts, in the hope of using their vulnerabilities.

Figure A.1 gives a full-range demonstration of monitored traffic variation during the spread of Witty Worms. The X -axis marks the running time in seconds. It is for more than five days' monitoring. The Y -axis marks the number of destination IP addresses that packets inside each specified source IP flow try to reach per second. Here, a source IP flow is defined as a group of IP packets bearing the same first three octets of their source IP addresses. Legends at the upper right corner list the selected source IP flows. In order to have a general representation, source IP flows with different spreading scales are selected. "Flow: 12.102.95.x Total: 100" represents packets from this source IP flow attempt to reach 100 different destination IPs

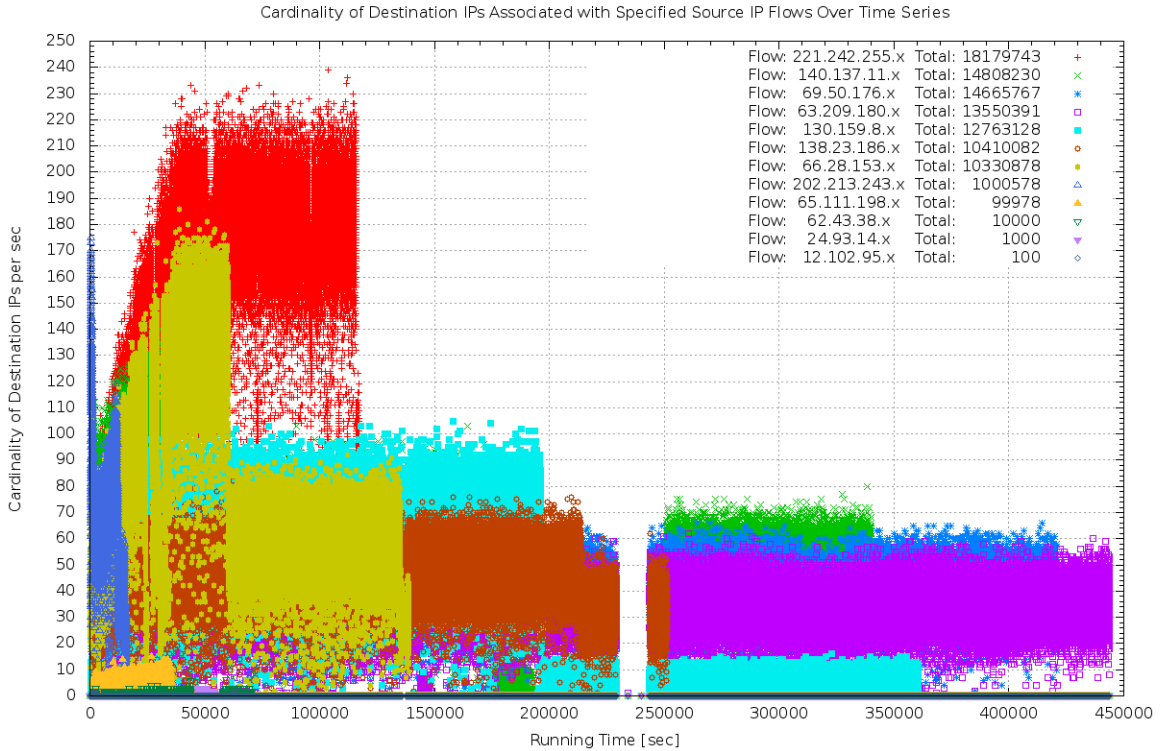


Figure A.1. Monitored traffic variation during the spread of the Witty Worm.

over five days’ monitoring; while ”Flow: 221.242.225.x Total: 18179743” represents attempts of 18179743 different IPs.

Figure A.2 plots the evaluation result between the number of attempted destination IPs and their corresponding source IPs on top of the overall monitoring. The X-axis marks the number of unique source IPs of monitored packets. The order of these source IPs is determined by the ascending order of numbers of destination IPs they associate with. The Y-axis marks the number of attempted destination IPs corresponding to a unique source IP. From this plot, it is obvious that packets from most source IP addresses show low probing attempts. In other words, they follow their regular routes for transmission. Only packets from a few source IPs show prohibitively high probing attempts, which are highly suspicious.

The list in Table A.1 provides more detailed information. Statistically, 60,739 unique source IPs are found in the date set of ”Witty Worm 2004”, which correspond

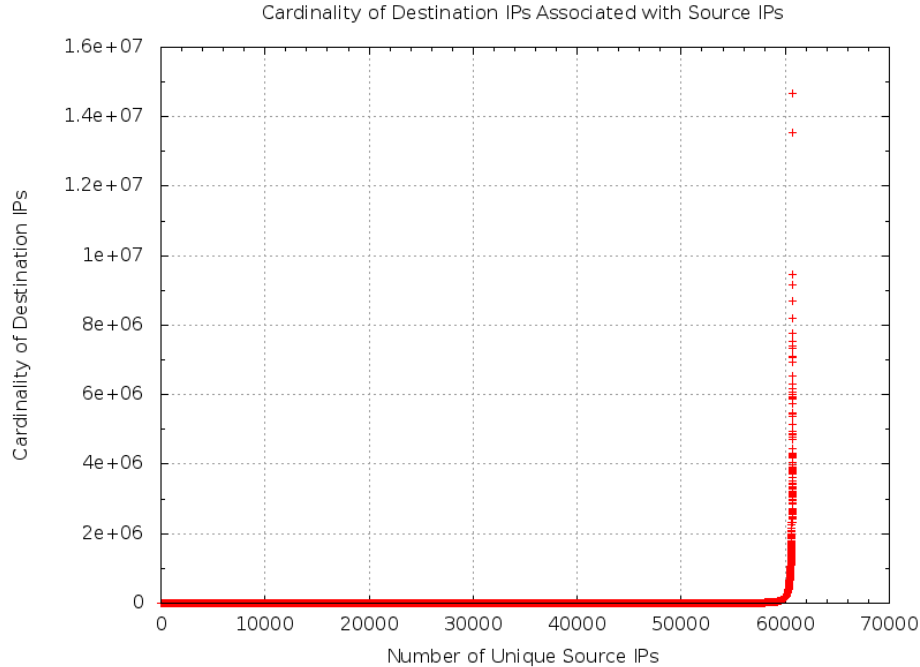


Figure A.2. Relations between the number of attempted destination IPs and their corresponding source IPs.

to 999,241,439 destination IPs in total. More than 98% source IPs are associated with around 11% destination IPs; while only less than 2% source IPs are responsible for probing the rest 89% destination IPs. In particular, two source IPs which only occupy 0.003% of total source IPs contribute 2.8% association with destination IPs.

The plot and list in Figure A.3 and Table A.2 work the same manner as both in Figure A.2 and Table A.1 respectively, instead of using source IP flows as the replacement on the X -axis. Though the result comparison between Table A.2 and Table A.1 dose not show significant difference, statistic results on top of source IP flows feature a wider angle of view. Besides, the log scale is applied to values of the Y -axis in Figure A.3. By this means, the trend of increase at the lower part of source/destination IPs association can be observed more clearly.

Figure A.4, A.5 and A.6 are plotted with the same six selected IP flows, but different formats. Basically, they all follow the rules of plotting Figure A.1. Since major variations of these IP flows end 70,000 seconds (about 20 hours) after

Table A.1. Categorized statistic information of attempted destination IPs Vs. corresponding source IPs.

Number of dst_ips Per src_ip	number of src_ip	Percentage of src_ip (%)	Count of dst_ip	Percentage of dst_ip (%)
Overall	60,739	100	997,241,439	100
1_to_1000	49,205	81.0106	5,483,635	0.54988
1001_to_10000	7,867	12.9521	26,345,702	2.64186
10001_to_100,000	2,636	4.33988	80,768,428	8.09918
100,001_to_1,000,000	807	1.32864	255,234,496	25.5941
1,000,001_to_10,000,000	222	0.365498	601,193,020	60.2856
above_10,000,000	2	0.00329278	28,216,158	2.82942

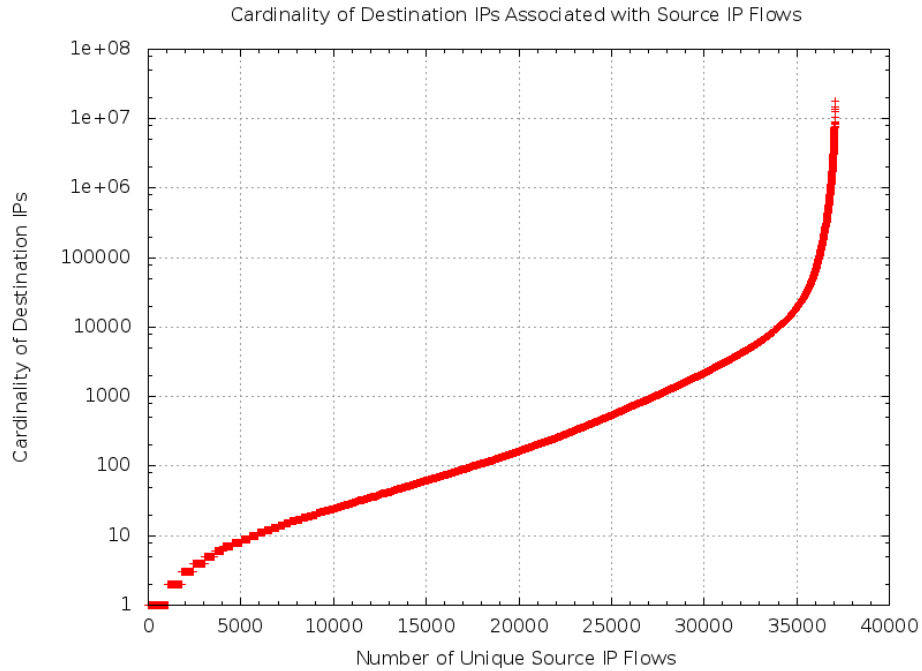


Figure A.3. Relations between the number of attempted destination IPs and their corresponding source IP flows.

monitoring, that is the time range setting for this plot. Figure A.5 presents a close look of variations just after the outbreak of Witty Worms. Thus, this plot zooms into the range of the first 10 minutes for details. Figure A.6 further provides a clear view of variations of each selected IP flow, according to Figure A.1.

Table A.2. Categorized statistic information of attempted destination IPs Vs. corresponding source IP flows.

Number of dst_ips Per src_ip_flow	number of src_ip_flow	Percentage of src_ip (%)	Count of dst_ip	Percentage of dst_ip (%)
Overall	37,060	100	997,241,439	100
1_to_1,000	27,287	73.6292	4,069,329	0.408059
1,001_to_10,000	6,704	18.0896	23,097,310	2.31612
10,001_to_100,000	2,209	5.9606	63,681,495	6.38577
100,001_to_1,000,000	643	1.73502	215,176,545	21.5772
1,000,001_to_10,000,000	210	0.566649	596,508,541	59.8159
above_10,000,000	7	0.0188883	94,708,219	9.49702

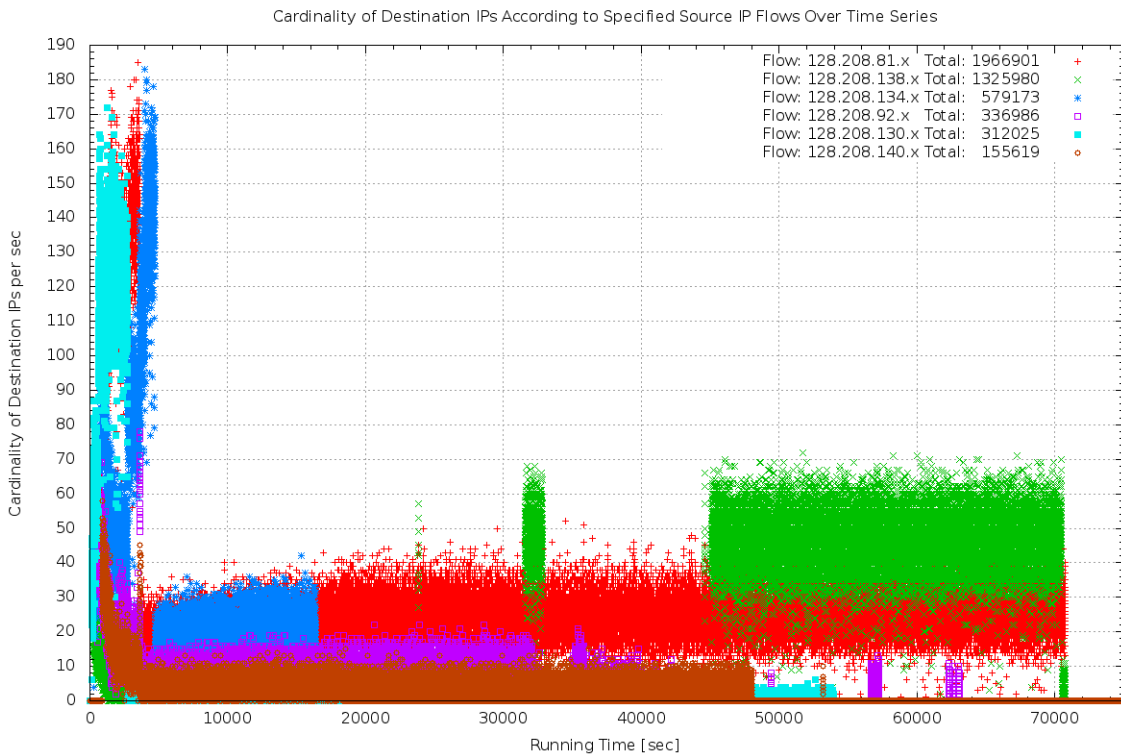


Figure A.4. Traffic variation of selected source IP flows.

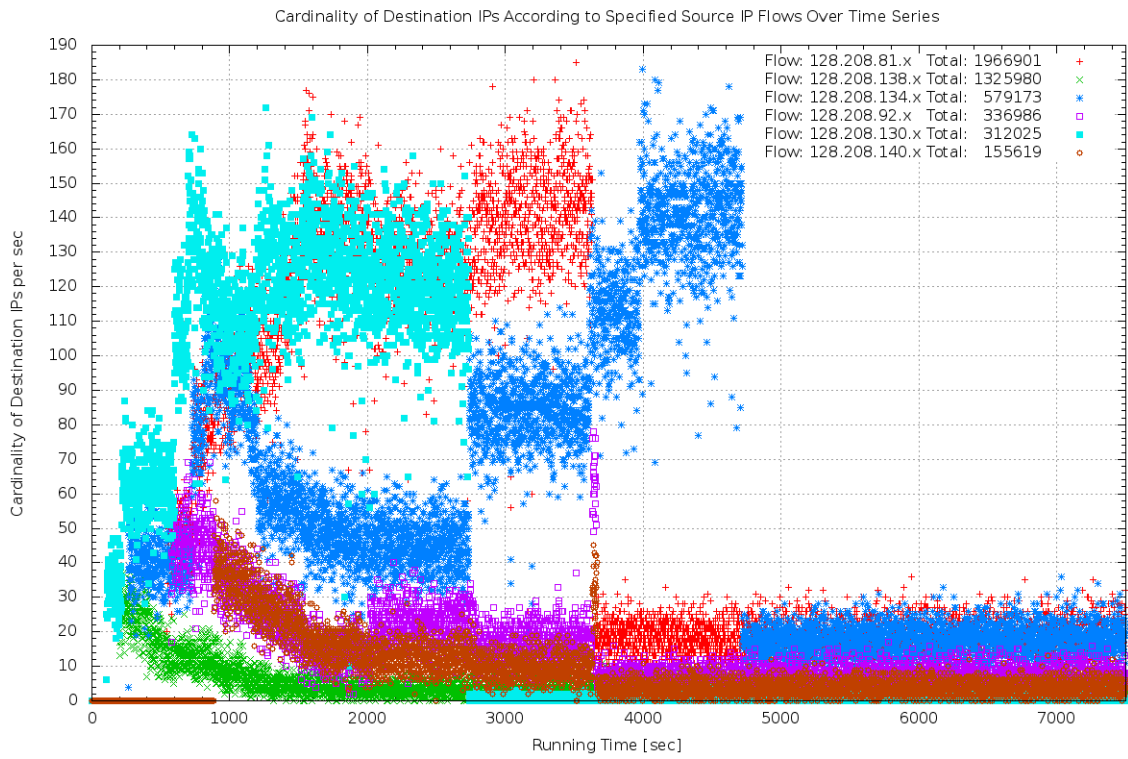


Figure A.5. A close look of the outbreak of Witty Worms.

Cardinality of Destination IPs According to Specified Source IP Flows Over Time Series

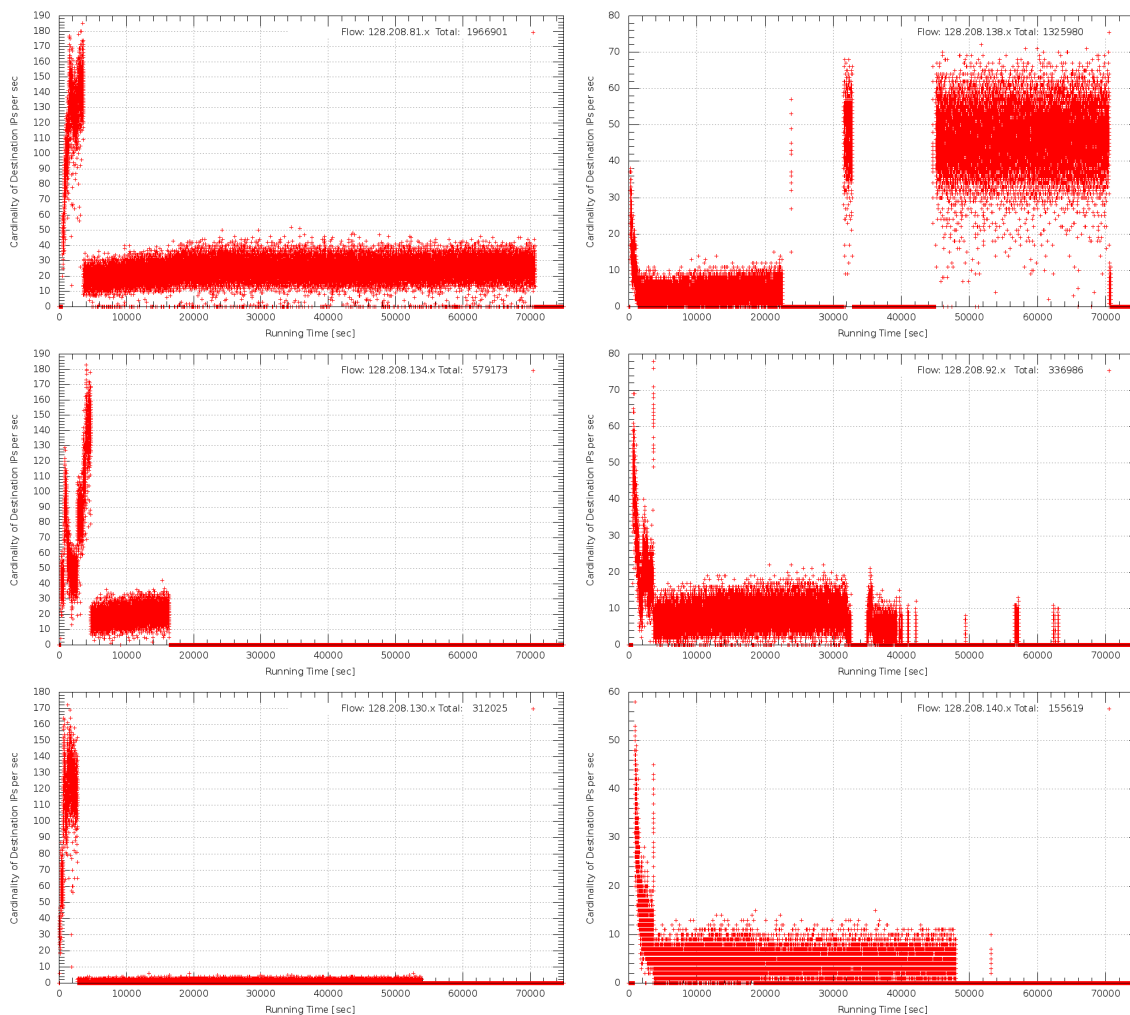


Figure A.6. Traffic variation of the individual source IP flow.

Appendix B

Prototyping on NetFPGA

In network intrusion detection, monitoring packet rate at the network/Internet layer is a common approach for surveillance. The packet rate refers to the amount of packet contents collected per unit time. The interested contents could be any type of information contained in the packet header and/or payload. This process is fulfilled through a small application called network packet counter, which is more commonly known as IP packet counter.

Reviewing my previous work, an important reason preventing further evaluation in the real world is a lack of such an application that can forward real network trace into hardware device for processing. Application of such a IP packet counter can not only bridge the gap of my study from theory to real, but contribute very helpful information for further process.

However, extracting information of an IP packet from network traffic is not an easy task to accomplish. As the famous Internet protocol stacks application on any single packet traveling through the Internet, the unwanted protocol headers wrapped outside the packet should be first removed before counting. This "onion-peeling" process involves much low level co-work between hardware and software, which sets a high barrier to researchers who want to realize their goal with less involvement of low level protocols and implementations.

The emergence of NetFPGA and the relative open sources projects bring a



Figure B.1. The first generation NetFPGA 1G (4x1G) platform.

breeze to the community. On top of a reference router design, our NetFPGA-based IP packet counter was developed. This counter is capable of extracting any information from the header of a packet for sampling, such as: counting the number of TCP packets per unit time. For easy debugging and operation, tools of logic analyzer and virtual I/Os from Xilinx ChipScope Pro are also employed.

Along the way of this development, tremendous time and energy were devoted. Lacking support documents leads prototyping on this immature research device extremely difficult. Years of effort were put into debugging and configuration of these NetFPGA devices, from settling down a single piece, to build a system, to further expansion to a NetFPGA-based testbed. Figure B.11 illustrates the framework of the NetFPGA-based testbed, and Figure B.12 presents the configuration of setting up a lab for running NetFPGA-based tutorial routers.

The remainder of this Appendix is organized as follows. Appendix B.1 describes the development of our NetFPGA-based IP packet counter. Appendix B.2 elaborates the debugging process of this counter with Xilinx ChipScope.

B.1 Development of a NetFPGA-Based IP Packet Counter

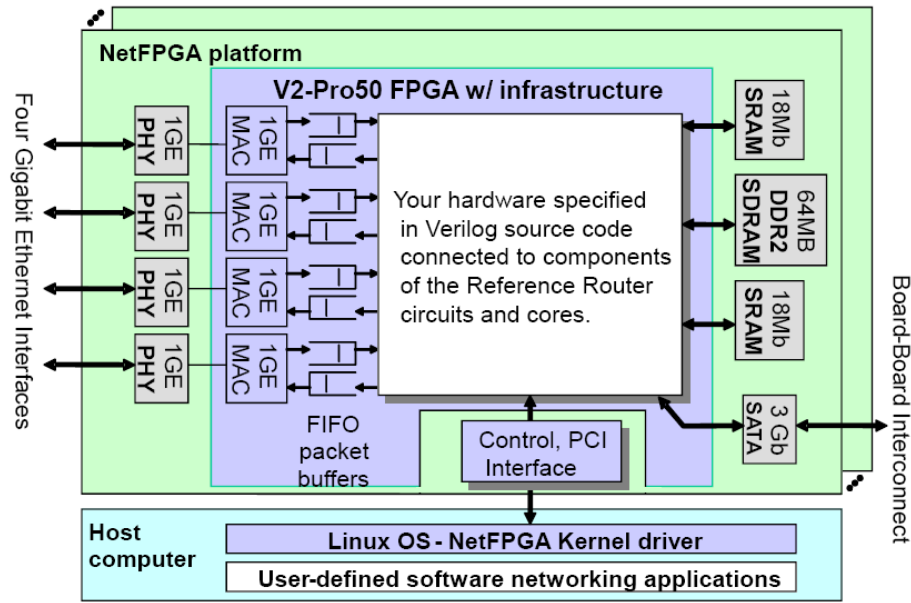


Figure B.2. A block diagram of NetFPGA system.

The NetFPGA [106] is a low-cost reconfigurable hardware platform optimized for high-speed networking. Figure B.1 shows a platform of NetFPGA-1G. It is primarily designed as a tool for study networking hardware and router design. As its name, this device bridges "Network" to "FPGA", with four 1G Ethernet NICs integrated on a Xilinx Virtex-II Pro 50 FPGA development board. In cooperation with other onboard resources, such as logic components and memories, this platform is capable of building a complete switch, router, and/or possible network security device.

Figure B.2 provides a concise view of a NetFPGA system with major components outlined. As illustrated, the NetFPGA has a standard PCI interface allowing it to be connected to a host computer. It offloads processes from a host processor. The host's CPU has access to main memory and can DMA to read and write registers and memories on the NetFPGA. Therefore, any software part can run on the host side,

while a hardware accelerator built with FPGA drives Gigabit network links. Since the NetFPGA provides a full hardware-accelerated datapath, the system can support back-to-back packets at Gigabit line rates and has a processing latency measured in only a few clock cycles [121].

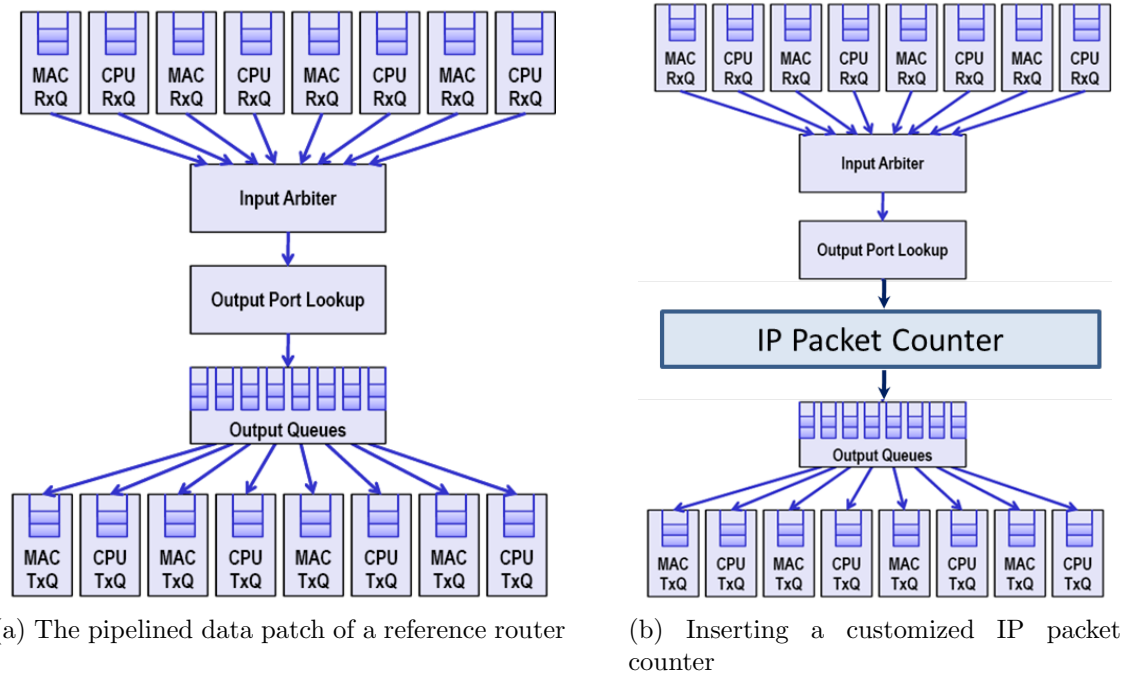


Figure B.3. The user data path of NetFPGA-based designs.

There are multiple ways to implement the user logic on a NetFPGA. A common approach is extending the reference design with a custom user module. The development of our IP packet counter follows this way. The data path presented in Figure B.3a gives insight on how a NetFPGA is configured to work as a router. It features five-stage styled pipeline, with the packet-based module interface and plug-gable design. Though the detail explanation of this pipeline is skipped here, you may find more online at netfpga.org.

The interface for inter-communication between any two modules is depicted in Figure B.4. In terms of functionality, the user data path traveling through modules can be further categorized to three groups. The main user data path is for passing packet information. It is designed as 64-bit wide for running at 125 MHz frequency,

which gives the data path a peak bandwidth of 8 Gbps. An individual control data path takes the responsibility of passing control signals with 8-bit wide. Finally, two shake-hand signals insure a seamless communication in between.

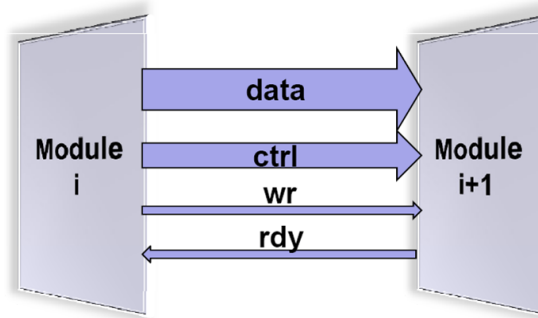


Figure B.4. The interface for inter-module communication between modules.

This modularized interface makes the insertion of our IP packet counter relatively easy. As shown in Figure B.3b, the module of IP packet counter is inserted in-between "Output Port Lookup" and "Output Queues" modules. To be simple, that is the location where the basic routing function has been fulfilled except for further transmission, so as to be the best place for our insertion.

Beyond following the explicit user data path for design adaptation, setting dedicated I/Os for this counter is also a necessity. Without an effective control, the counter would be less meaningful in real practice. A dedicated reset is at least required to set the start point of sampling without affecting the original function of network routing. In addition, an adjustable timer is also necessary to achieve a desirable sampling period. As to output, there should be a place that can hold results from the counter for access.

The fulfillment of this interaction is through the access of generic registers. Two types of generic registers serve for this purpose. One is software (SW) registers which accept instructions from the host for counting manipulation, the other is hardware (HW) registers which accept results from the counter for access. The block diagram in Figure B.5 presents the architecture of our design. It composes with three

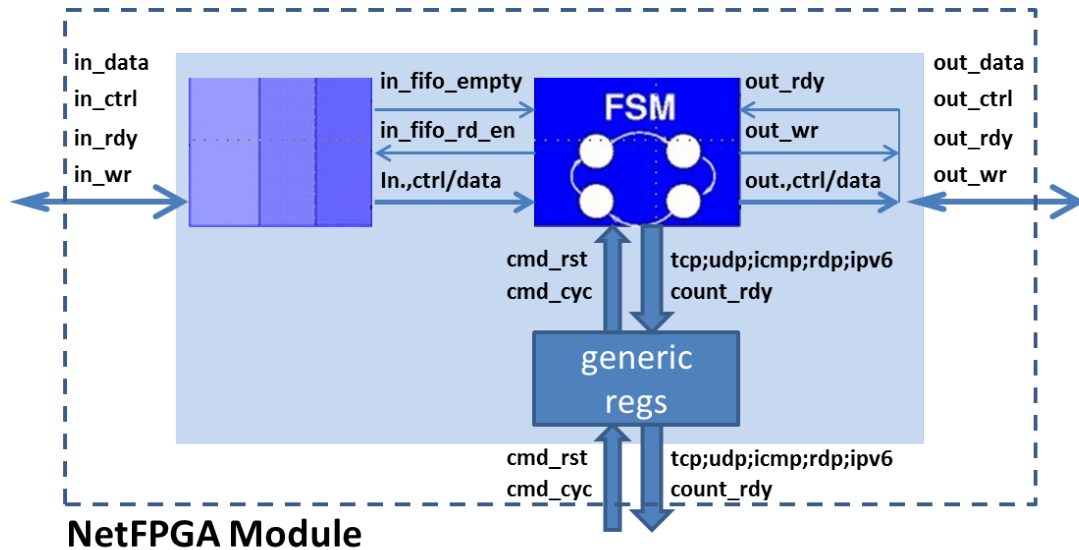


Figure B.5. The architecture of NetFPGA-based IP packet counter.

major blocks. The kernel is a Finite State Machine (FSM) for all logic. An instantiated FIFO memory is used for holding input data. Two SW and five HW generic registers serve for interaction.

The designed IP packet counter is dedicated to identifying and counting five major types of network packets. They are: TCP, UDP, ICMP, RDP, and IPV6 packets, respectively. As long as the header of any packet is ready, the search logic will immediately compare the packet type information with the above five, and handle the counter properly. At the end of each sampling period, all five counters are reset to zero. Meanwhile, counting values stored in the corresponding HW generic registers are updated. These values will be held till the end of the next sampling period. Besides, two SW generic registers are applied for management of this counter dynamically. One is for resetting all counters, and the other is for setting the length of a sampling period.

B.2 Real Prototyping on TCP Packet Counting with Chipscope

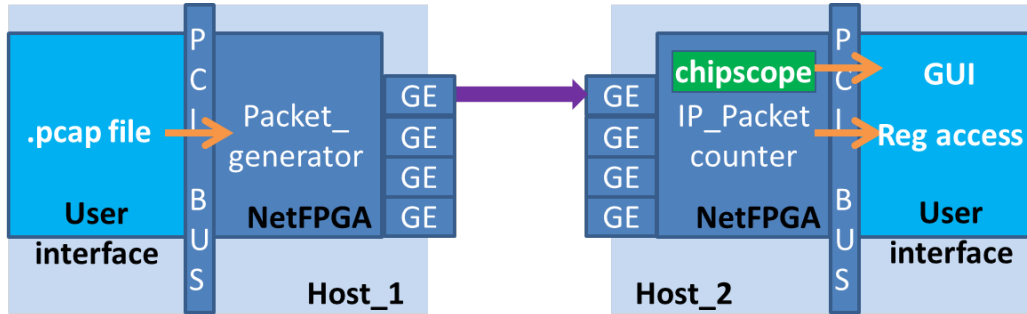


Figure B.6. The overall experiment environment.

In general, the logic design part of this IP packet counter is not that difficult. After a series of successful simulation, synthesis, placing and routing procedures, an executable bit file of this IP packet counter was downloaded to a NetFPGA board.

To make the prototype controllable and verifiable, a dedicated experiment environment was set up. This environment utilizes our developed NetFPGA testbed, but with only two NetFPGA systems involved. They are simply connected through an Ethernet cable from one end of each Gigabit NIC ports. Figure B.6 gives the detail of this setup. Host_2 serves as the receiver with the developed IP packet counter on board, while Host_1 servers as the transmitter for sending packet.

As observed in Figure B.6, the NetFPGA in Host_1 is configured as a packet generator. This packet generator [54] is a user-contributed, open-source application from the community. In fact, it is another successful example extended from the design of reference router. This application can reliably replay time sensitive traffic while giving users the ability to modify replay behaviors. Under a set of given instructions, the rate of the queues, the delay between packets, and the number of iterations that a PCAP file are cycled through are all controllable. On the other hand, it also

possesses the capability of packet capturing. Both processes can be operated under a full Gigabit line rate.

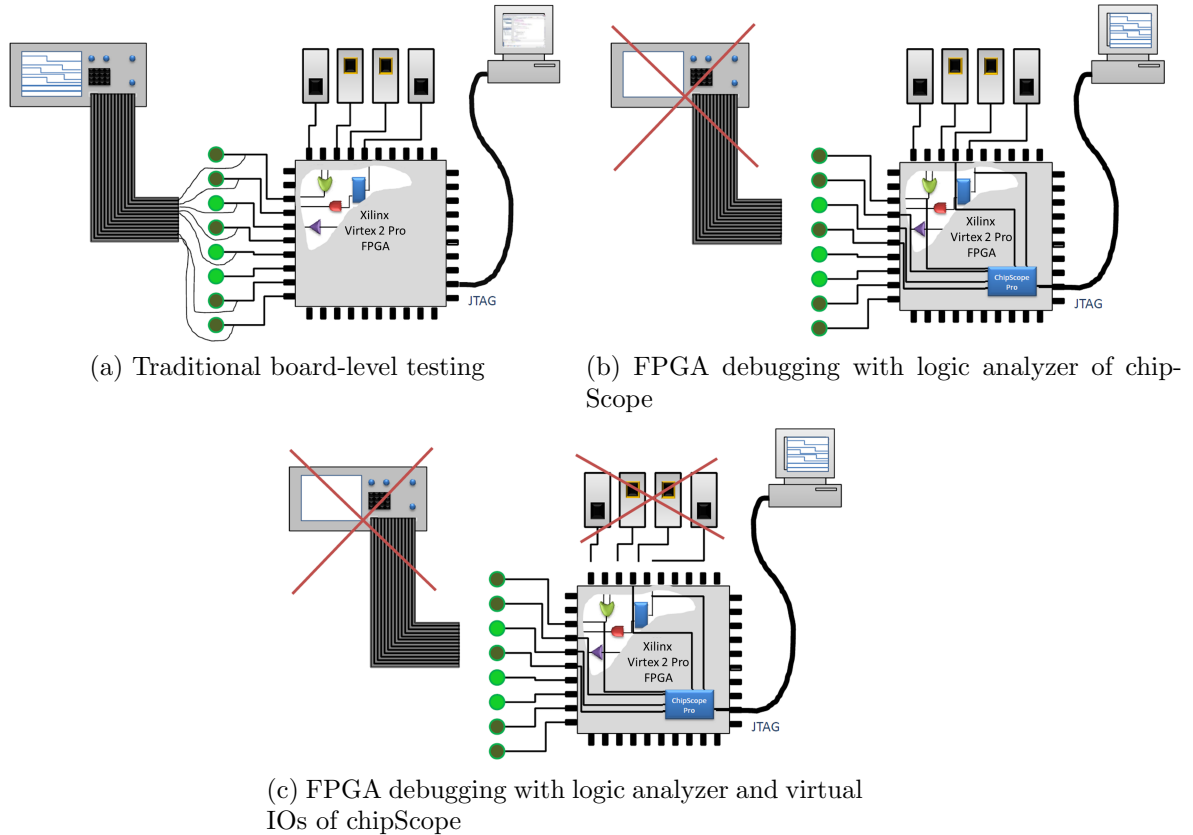


Figure B.7. ChipScope involved FPGA debugging.

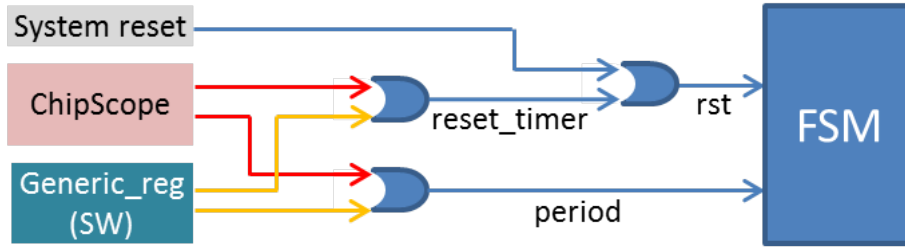
In comparison with settling the transmitter at Host_1, debugging this IP packet counter at Host_2 is much more complicated. A conventional debugging approach is wiring one end of I/O pins to a group of switches for input, while connecting the other end to LEDs, seven-segment displays, or even oscilloscope for observation. Figure B.7a depicts such a board-level testing. However, this set of approaches lacks the capability of debugging internal signals. Though application of a professional external logic analyzer may resolve this concern, it is at the cost of over \$10,000. This leaves room for ChipScope - a lightweight, embedded software logic analyzer to work it out.

ChipScope [176] consists of a set of tools made by Xilinx. Through a link of

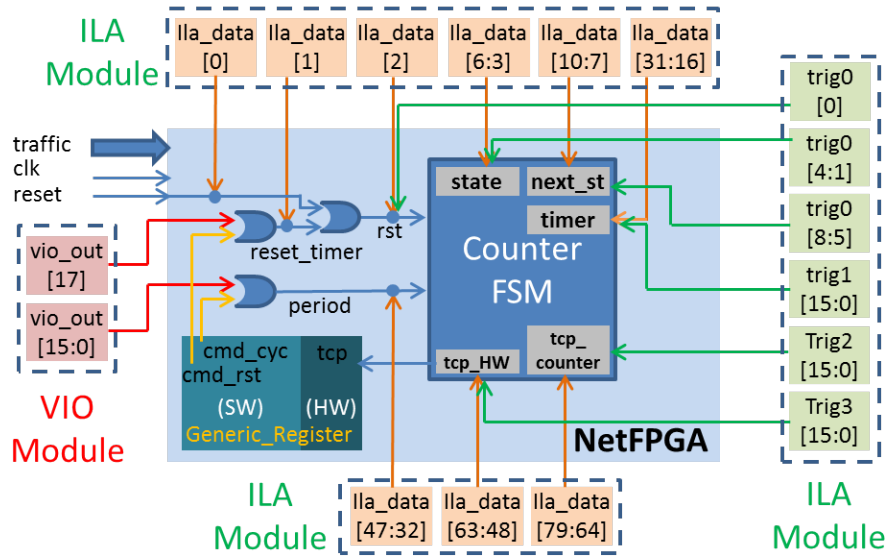
JTAG cable, it allows easy probing and manipulating signals on hardware from the host during runtime. The "Integrated Controller Core" (ICON) provides an interface between FPGA device and other ChipScope cores. By inserting an ICON and an "Integrated Logic Analyzer" (ILA) into a design and connecting them properly, one can monitor any internal signals. Beyond that, the core of "Virtual Input/Output" (VIO) makes it efficient for signal manipulation through the GUI interface of a management console. Since these cores are instantiated with real Verilog modules and netlists, they get incorporated, synthesized, and implemented into a design just like any other Verilog modules. From this perspective, application of ChipScope is at the cost of on-board resources. Figure B.7b and B.7c depict the progress of applying ICON, ILA cores, and further the VIO core of ChipScope for FPGA debugging, respectively.

Debugging of this IP packet counter follows the approach of Figure B.7c, with both ILA and VIO cores integrated. Figure B.8a gives a schematic view of control logic for counter manipulation. To be concise, the signal of system clock is hidden. In this figure, the FSM block at the right side represents the internal logic of this counter, while three blocks at the left represent different control resources. The links in-between reveal the control logic with OR gates. One may observe that signals from both ChipScope and SW generic registers have the same functionality. This arrangement is at the convenience OF debugging, since interaction with virtual I/Os is more efficient than execution of perl scripts during runtime. Meanwhile, the OR gates still leave room for testing both independently.

Figure B.8b provides detail wiring of ChipScope cores for application. The light blue area in the middle represents the implemented logic. Blocks of the Counter FSM and Generic Registers inside are highlighted with monitored signals. Two groups of ILA module interface above and below the logic block point out connections of all the monitored signals. Besides, the ILA module interface at the right side further indicates the trigger-able signals. Trigger-able signals are special monitored signals,



(a) A schematic view of driving control for debugging

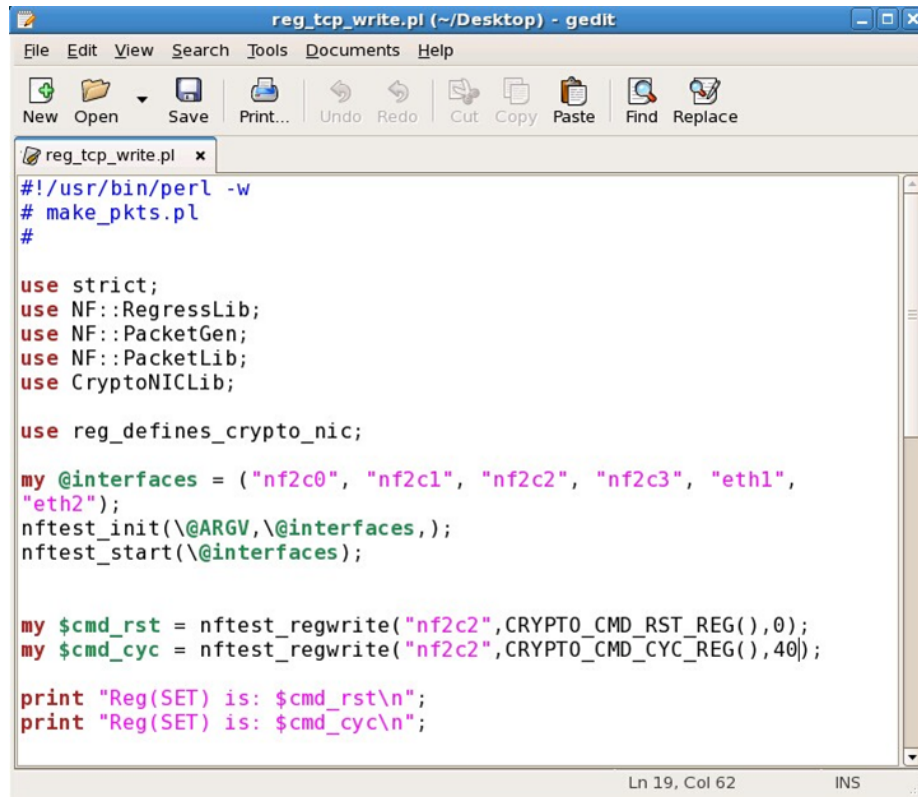


(b) Intergration of ChipScope cores into IP packet counter

Figure B.8. Manipulating IP packet counter with ChipScope.

which are used to trigger debugging mode when preset condition(s) are met. Finally, the VIO module interface at the left side connects two virtual input signals for counting manipulation. For clear demonstration of debugging results, only counting value of TCP packets are monitored in this practice. The monitored signal of *tcp_HW* stores the corresponding counting results and is wired to a HW generic register for access. Meanwhile, this signal is also connected to *Trig3* for event triggering as illustrated.

To have a meaningful running, control signals should be first set. Through observation, we find that the default value of "System reset" is zero which enables the rest logic. Thus, "reset" signals from both ChipScope and generic register should also keep "zero" to maintain the same status. Meanwhile, the signal of "period"



```
#!/usr/bin/perl -w
# make_pkts.pl
#

use strict;
use NF::RegressLib;
use NF::PacketGen;
use NF::PacketLib;
use CryptoNICLib;

use reg_defines_crypto_nic;

my @interfaces = ("nf2c0", "nf2c1", "nf2c2", "nf2c3", "eth1",
"eth2");
nftest_init(\@ARGV,\@interfaces,);
nftest_start(\@interfaces);

my $cmd_rst = nftest_regwrite("nf2c2",CRYPTO_CMD_RST_REG(),0);
my $cmd_cyc = nftest_regwrite("nf2c2",CRYPTO_CMD_CYC_REG(),40);

print "Reg(SET) is: $cmd_rst\n";
print "Reg(SET) is: $cmd_cyc\n";
```

Figure B.9. Set up software registers through a Perl script.

should be assigned to determine the sampling period. A snap shot of a Perl script for setting two SW generic registers is given in Figure B.9. Signals of *cmd_rst* and *cmd_cyc* are set accordingly. Here, the value of *cmd_cyc* refers to the number of clock cycles of a sampling period. Setting such a small value is for the verification whether this SW generic register can be written correctly. The dominant part of sampling period is set through Virtual I/Os of ChipScope.

Taking advantage of a GUI for management, all ChipScope related operations can be well coordinated. Figure B.10 gives a real example. The layout of this GUI consists of three parts. The left panel is for navigation of projects and signals; the right panel is for runtime manipulation; and the bottom panel is for status information. Our focus is on the right panel which also composes with three sections. The bottom section of VIO Console is where we enter values for virtual I/Os. Here, period of 40000 clock cycles is set and reset remains to 0.

The top and middle sections belong to ILA. The top section of Trigger Setup looks a little bit complicated. Basically, it is for setting trigger event(s) of debugging. Only *TriggerPort3*, i.e: *Trig3* in Figure B.8b is activated in our case. *TriggerPort3* ≥ 26 implies that only when the number of captured TCP packets reaches 26 per sampling period, will the monitored signals be displayed in the beneath Waveform section. Strictly speaking, only a trunk of 4096 cycles' data after the triggering point will be reserved and displayed, according to the current setting. The maximum range of this reservation is actually determined by the available resource allocated to ChipScope.

Results displayed in the Waveform section have more details after triggering. Keeping zero of *rst* makes sure the availability for running. The ordered values of *state* and *next_state* mean the core FSM works well, so as to enable proper counting operation accordingly. The moment of triggering debugging mode, a detection of the first qualified counting result at *tcp* = 31, not only triggers ChipScope into a debugging mode, but starts sampling process for a new iteration. The increase of *TCP_counter* tells the story. The expected sampling period of 40040 verifies the functionality of both virtual I/Os as well as SW generic registers with an OR gate. However, only roughly one tenth of the whole sampling period can be displayed, because of the mentioned length limitation above. Therefore, the functionality of this IP packet counter can be verified, through the process of hardware debugging on top of prototyping.

Chipscope Pro Analyzer [crypto_ila_vio_2]

Project: crypto_ila_vio_2

Signals: DEV:1 UNIT: 0

- next_state
- period
- state
- tcp
- TCP_counter
- timer
- CH: 0 reset
- CH: 1 rst_timer
- CH: 2 rst
- CH: 3 DataPort[3]
- CH: 4 DataPort[4]
- CH: 5 DataPort[5]
- CH: 6 DataPort[6]
- CH: 7 DataPort[7]
- CH: 8 DataPort[8]
- CH: 9 DataPort[9]
- CH: 10 DataPort[10]
- CH: 11 DataPort[11]
- CH: 12 DataPort[12]
- CH: 13 DataPort[13]
- CH: 14 DataPort[14]
- CH: 15 DataPort[15]
- CH: 16 DataPort[16]
- CH: 17 DataPort[17]
- CH: 18 DataPort[18]
- CH: 19 DataPort[19]
- CH: 20 DataPort[20]
- CH: 21 DataPort[21]
- CH: 22 DataPort[22]
- CH: 23 DataPort[23]
- CH: 24 DataPort[24]

Trigger Setup - DEV:1 MyDevice1 (XC2VP50) UNIT:0 MyILA0 (ILA)

Match	Function	Value	Radix	Counter
M0: TriggerPort0	==	XXXX	Hex	disabled
M1: TriggerPort1	==	XXXX	Hex	disabled
M2: TriggerPort2	==	XXXX	Hex	disabled
M3: TriggerPort3	>=	26	Unsigned	disabled

Trigger Condition Name: TriggerCondition0
Trigger Condition Equation: M3

Active: Active

Type: Window | Depth: 4096 | Position: 0

Storage Qualification: All Data

Waveform - DEV:1 MyDevice1 (XC2VP50) UNIT:0 MyILA0 (ILA)

Bus/Signal	X	0	160	320	480	800	960	1120	1280	1440	1600	1760	1920	2080	2240	2400	2560	2720	2880	3040	3200	3360	3520	3680	3840	4000
period	0	4004																								
timer	0																									
state	0000							0110		0000																
next_state	0000							0110		0000																
TCP_counter	0							1		2																
tcp	0							3		4																
fst	0																									

VIO Console - DEV:1 MyDevice1 (XC2VP50) UNIT:1 MyVIO1 (VIO)

```

AsyncOut[17]_rst_vio
period

```

Value: 0

Value: 40000

COMMAND: run 1 0

INFO - Device 1 Unit 0: Waiting for core to be armed

Upload

DONE

Figure B.10. Chipscope monitoring under real running.

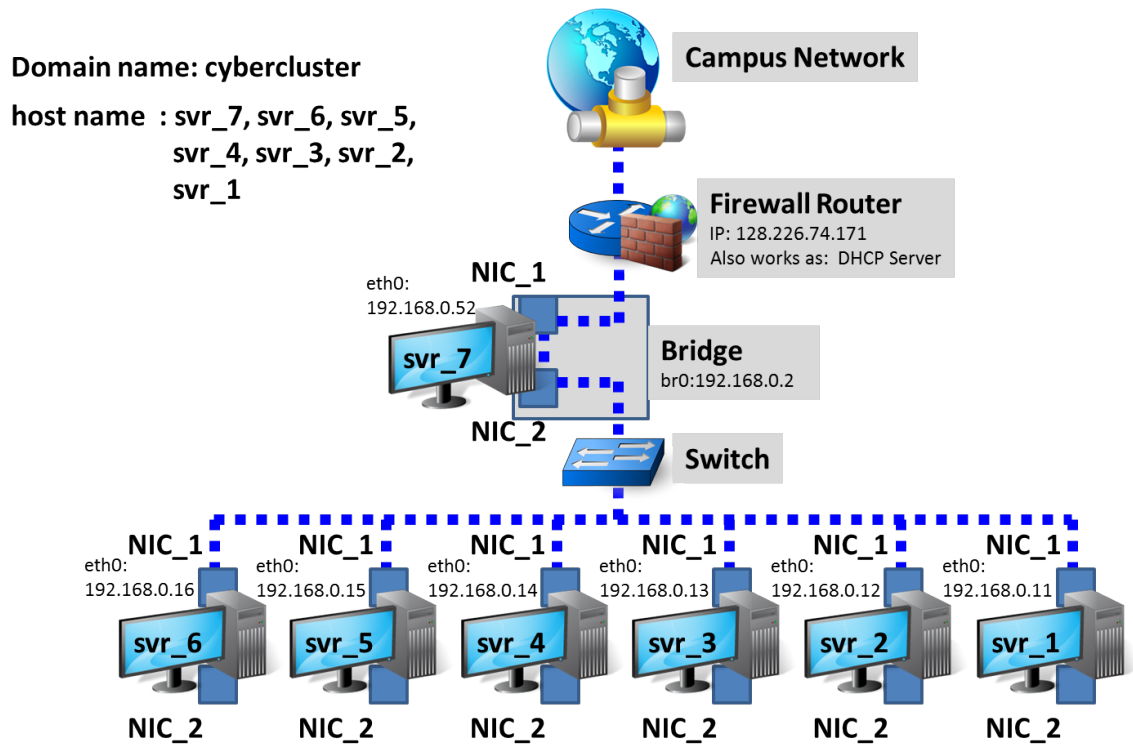


Figure B.11. The framework of our NetFPGA-based network testbed.

Domain name: cybercluster

host name : svr_T, svr_5, svr_4, svr_3, svr_2, svr_1



All IP addresses are

prefixed as:
192.168.X.X

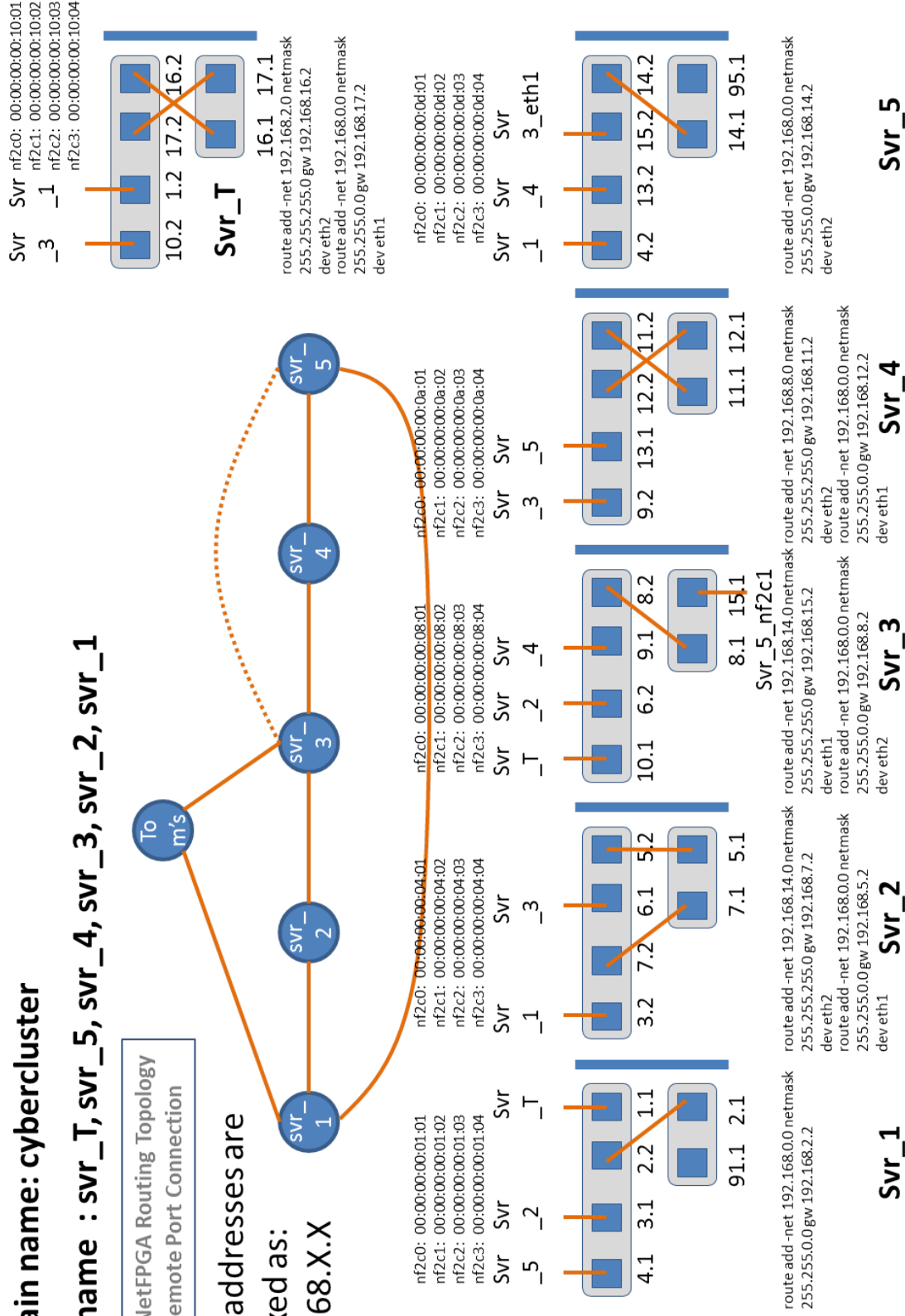


Figure B.12. Lab configuration of NetFPGA-based tutorial routers.

Appendix C

Publications

- **Journal Papers:**

1. **Chen, H.**, Gaska, T., Chen, Y., and Summerville, D. H. (2013). An optimized reconfigurable power spectral density converter for real-time shrew DDoS attacks detection. *Computers & Electrical Engineering*, 39(2), 295-308.
2. **Chen, H.**, Chen, Y., and Summerville, D. H. (2011). A survey on the application of FPGAs for network infrastructure security. *Communications Surveys & Tutorials, IEEE*, 13(4), 541-561.
3. Chen, Y., and **Chen, H.** (2009). Neuronet: An adaptive infrastructure for network security. *International Journal of Information, Intelligence and Knowledge*, 1(2), 143-168.

- **Conference Papers:**

1. **Chen, H.**, Chen, Y., Summerville, D. H., and Su, Z. (2013, April). An optimized design of reconfigurable PSD accelerator for online shrew DDoS attacks detection. In *INFOCOM, 2013 Proceedings IEEE* (pp. 1780-1787). IEEE.

2. **Chen, H.**, and Chen, Y. (2010, October). A comparison study of collaborative strategies for distributed defense against Internet worms based on small-world modeling. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2010 6th International Conference on* (pp. 1-10). IEEE.
3. **Chen, H.**, Summerville, D. H., and Chen, Y. (2009, October). Two-stage decomposition of SNORT rules towards efficient hardware implementation. In *Design of Reliable Communication Networks, 2009. DRCN 2009. 7th International Workshop on* (pp. 359-366). IEEE.
4. Craver, S., Chen, Y., **Chen, H.**, Yu, J., and Atakli, I. M. (2009, January). Blink: Securing information to the last connection. In *Consumer Communications and Networking Conference, 2009. CCNC 2009. 6th IEEE* (pp. 1-2). IEEE.
5. **Chen, H.**, and Chen, Y. (2008, June). A novel embedded accelerator for online detection of shrew DDoS attacks. In *Networking, Architecture, and Storage, 2008. NAS'08. International Conference on* (pp. 365-372). IEEE.

- **Book Chapters:**

1. **Chen, H.**, and Chen, Y. (2011). An Experimental Comparison of Collaborative Defense Strategies for Network Security. *Pervasive Computing and Networking*, 189.
2. **Chen, H.**, Feng, J., Chen, Y., and Summerville, D. H., Securing Network Infrastructure using Reconfigurable Hardware Devices, *Security Engineering Techniques and Solutions for Information Systems: Management and Implementations*, edited by N. Boudriga and M. Hamdi, published by Idea Group, Inc., ISBN 1615208038, March 2010.

3. Chen, Y., **Chen, H.**, and Ku, W.-S., Malicious Node Detection in Wireless Sensor Networks, *Security in RFID and Sensor Networks*, edited by Josh Y. Zhang and Paris Kitsos, published by Auerbach Publications, Taylor & Francis Group, ISBN 1420068393, 2009.

Bibliography

- [1] AL-KUWAITI, M., KYRIAKOPOULOS, N., AND HUSSEIN, S. A comparative analysis of network dependability, fault-tolerance, reliability, security, and survivability. *IEEE Communications Surveys & Tutorials* 11, 2 (2009), 106–124.
- [2] ALBERT, R., JEONG, H., AND BARABÁSI, A.-L. Internet: Diameter of the world-wide web. *Nature* 401, 6749 (1999), 130–131.
- [3] ALBERT, R., JEONG, H., AND BARABÁSI, A.-L. Error and attack tolerance of complex networks. *Nature* 406, 6794 (2000), 378–382.
- [4] ALDWAIRI, M., CONTE, T., AND FRANZON, P. Configurable string matching hardware for speeding up intrusion detection. *SIGARCH Comput. Archit. News* 33, 1 (Mar. 2005), 99–107.
- [5] ALLCHIN, D. Error types. *Perspectives on science* 9, 1 (2001), 38–58.
- [6] ALLMAN, M., BLANTON, E., PAXSON, V., AND SHENKER, S. Fighting coordinated attackers with cross-organizational information sharing. *IRVINE IS BURNING* (2006), 121.
- [7] ATTIG, M., DHARMAPURIKAR, S., AND LOCKWOOD, J. Implementation results of bloom filters for string matching. In *Field-Programmable Custom Computing Machines, 2004. FCCM 2004. 12th Annual IEEE Symposium on* (2004), pp. 322–323.
- [8] ATTIG, M., AND LOCKWOOD, J. SIFT: Snort intrusion filter for TCP. In *Proceedings of the 13th Symposium on High Performance Interconnects* (Washington, DC, USA, 2005), HOTI '05, IEEE Computer Society, pp. 121–127.
- [9] BABOESCU, F., AND VARGHESE, G. Scalable packet classification. *Networking, IEEE/ACM Transactions on* 13, 1 (2005), 2–14.
- [10] BAKER, Z. K., AND PRASANNA, V. K. A methodology for synthesis of efficient intrusion detection systems on FPGAs. In *Field-Programmable Custom Computing Machines, 2004. FCCM 2004. 12th Annual IEEE Symposium on* (2004), pp. 135–144.
- [11] BAKER, Z. K., AND PRASANNA, V. K. High-throughput linked-pattern matching for intrusion detection systems. In *Architecture for networking and*

- communications systems, 2005. ANCS 2005. Symposium on* (2005), pp. 193–202.
- [12] BAKER, Z. K., AND PRASANNA, V. K. Automatic synthesis of efficient intrusion detection systems on FPGAs. *Dependable and Secure Computing, IEEE Transactions on* 3, 4 (2006), 289–300.
 - [13] BARABÁSI, B. A.-L., AND BONABEAU, E. Scale-free. *Scientific American* (2003).
 - [14] BARFORD, P., KLINE, J., PLONKA, D., AND RON, A. A signal analysis of network traffic anomalies. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurement* (Marseille, France, 2002), ACM, pp. 71–82.
 - [15] BELLOVIN, S. M., CLARK, D. D., PERRIG, A., AND SONG, D. A clean-slate design for the next-generation secure internet. In *NSF workshop at CMU* (2005).
 - [16] BELLOWS, P., AND HUTCHINGS, B. JHDL-an HDL for reconfigurable systems. In *FPGAs for Custom Computing Machines, 1998. Proceedings. IEEE Symposium on* (1998), pp. 175–184.
 - [17] BISPO, J., SOURDIS, I., CARDOSO, J., AND VASSILIADIS, S. Regular expression matching for reconfigurable packet inspection. In *Field Programmable Technology, 2006. FPT 2006. IEEE International Conference on* (Dec 2006), pp. 119–126.
 - [18] BONESANA, I., PAOLIERI, M., AND SANTAMBROGIO, M. D. An adaptable FPGA-based system for regular expression matching. In *Design, Automation and Test in Europe, 2008. DATE '08* (2008), pp. 1262–1267.
 - [19] BOS, H., AND HUANG, K. A network instruction detection system on IXP1200 network processors with support for large rule sets. In *Technical Report 2004-02* (2004), Leiden Univeristry.
 - [20] BRAUN, F., LOCKWOOD, J., AND WALDVOGEL, M. Protocol wrappers for layered network packet processing in reconfigurable hardware. *Micro, IEEE* 22, 1 (2002), 66–74.
 - [21] BRODER, A., AND MITZENMACHER, M. Network applications of bloom filters: A survey. *Internet Mathematics* 1, 4 (2003), 485–509.
 - [22] BROWN, S., FRANCIS, R., ROSE, J., AND VRANESIC, Z. *Field-Programmable Gate Arrays*, 1st ed., vol. 180 of *The Springer International Series in Engineering and Computer Science Ser.* Springer-Verlag New York, LLC, New York, 1992.

- [23] BURRUS, C. S., AND PARKS, T. W. *DFT/FFT and Convolution Algorithms: Theory and Implementation*, 1st ed. John Wiley & Sons, Inc., New York, NY, USA, 1991.
- [24] CAI, M., HWANG, K., KWOK, Y.-K., SONG, S., AND CHEN, Y. Collaborative internet worm containment. *Security & Privacy, IEEE* 3, 3 (2005), 25–33.
- [25] CAI, M., HWANG, K., PAN, J., AND PAPADOPOULOS, C. Wormshield: Collaborative worm signature detection using distributed aggregation trees. In *tech. report TR 2005-12* (2005), Univ. of Southern California.
- [26] CAIDA. Dataset on the Witty Worm - march 19-24, 2004. http://www.caida.org/data/passive/witty_worm_dataset.xml.
- [27] CAIDA. The UCSD network telescope. http://www.caida.org/projects/network_telescope/.
- [28] CALLADO, A., SZAB, C. K. G. Z., GERO, B. Z. P. T., KELNER, J., FERNANDES, S., AND SADOK, D. A survey on internet traffic identification. *IEEE Communications Surveys & Tutorials* 11, 3 (2009).
- [29] CARDOSO, L. S. Internet security and critical infrastructures. http://www.eurescom.de/message/messagesep2004/Internet_security_and_critical_infrastructure.asp.
- [30] CERT. Overview of attack trends. Tech. rep., CERT Coordination Center, Carnegie Mellon University, 2002.
- [31] CHAKRABARTI, A., AND MANIMARAN, G. Internet infrastructure security: a taxonomy. *Network, IEEE* 16, 6 (2002), 13–21.
- [32] CHANG, R. K. C. Defending against flooding-based distributed denial-of-service attacks: a tutorial. *Communications Magazine, IEEE* 40, 10 (2002), 42–51.
- [33] CHEN, H., AND CHEN, Y. A novel embedded accelerator for online detection of shrew DDoS attacks. In *Proceedings of the 2008 International Conference on Networking, Architecture, and Storage* (Washington, DC, USA, 2008), NAS '08, IEEE Computer Society, pp. 365–372.
- [34] CHEN, H., CHEN, Y., AND SUMMERVILLE, D. A survey on the application of FPGAs for network infrastructure security. *Communications Surveys Tutorials, IEEE* 13, 4 (2011), 541–561.
- [35] CHEN, H., SUMMERVILLE, D. H., AND CHEN, Y. Two-stage decomposition of snort rules towards efficient hardware implementation. In *Design of Reliable Communication Networks, 2009. DRCN 2009. 7th International Workshop on* (2009), IEEE, pp. 359–366.

- [36] CHEN, H., SUMMERVILLE, H. D., AND CHEN, Y. Two-stage decomposition of IDS rules towards efficient hardware implementation. Tech. rep., SUNY - Binghamton, Department of Electrical and Computer Engineering, January 2009.
- [37] CHEN, Y., AND CHEN, H. Neuronet: An adaptive infrastructure for network security. *International Journal of Information, Intelligence and Knowledge* 1, 2 (2009), 143–168.
- [38] CHEN, Y., AND HWANG, K. Collaborative change detection of ddos attacks on community and isp networks. In *Collaborative Technologies and Systems, 2006. CTS 2006. International Symposium on* (2006), IEEE, pp. 401–410.
- [39] CHEN, Y., AND HWANG, K. Collaborative change detection of DDoS attacks on community and ISP networks. In *Proceedings of the International Symposium on Collaborative Technologies and Systems* (Washington, DC, USA, 2006), CTS '06, IEEE Computer Society, pp. 401–410.
- [40] CHEN, Y., AND HWANG, K. Spectral analysis of tcp flows for defense against reduction-of-quality attacks. In *Communications, 2007. ICC'07. IEEE International Conference on* (2007), IEEE, pp. 1203–1210.
- [41] CHEN, Y., AND HWANG, K. Spectral analysis of TCP flows for defense against reduction-of-quality attacks. In *Communications, 2007. ICC '07. IEEE International Conference on* (2007), pp. 1203–1210.
- [42] CHEN, Y., HWANG, K., AND KU, W.-S. Collaborative detection of DDoS attacks over multiple network domains. *Parallel and Distributed Systems, IEEE Transactions on* 18, 12 (2007), 1649–1662.
- [43] CHEN, Y., HWANG, K., AND KU, W.-S. Distributed change-point detection of ddos attacks: Experimental results on deter testbed. In *DETER* (2007).
- [44] CHEN, Y., HWANG, K., AND KWOK, Y.-K. Filtering of shrew DDoS attacks in frequency domain. In *Local Computer Networks, 2005. 30th Anniversary. The IEEE Conference on* (2005), pp. 8 pp.–793.
- [45] CHENG, C.-M., KUNG, H. T., AND TAN, K.-S. Use of spectral analysis in defense against dos attacks. In *Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE* (2002), vol. 3, pp. 2143–2148 vol.3.
- [46] CHO, Y. H., AND MANGIONE-SMITH, W. H. Deep packet filter with dedicated logic and read only memories. In *Field-Programmable Custom Computing Machines, 2004. FCCM 2004. 12th Annual IEEE Symposium on* (2004), pp. 125–134.
- [47] CHO, Y. H., AND MANGIONE-SMITH, W. H. Programmable hardware for deep packet filtering on a large signature set, 2004.

- [48] CHO, Y. H., AND MANGIONE-SMITH, W. H. Deep network packet filter design for reconfigurable devices. *Trans. on Embedded Computing Sys.* 7, 2 (2008), 1–26.
- [49] CISCO. Cisco visual networking index: Global mobile data traffic forecast update. *White Paper, February* (2014).
- [50] CLARK, C. R., AND SCHIMMEL, D. E. Efficient reconfigurable logic circuits for matching complex network intrusion detection patterns. In *Proc. 11th ACM/SIGDA Int. Conf. Field-Programm. Logic Appl. (FPL)* (2003), p. 956.
- [51] CLARK, C. R., AND SCHIMMEL, D. E. Scalable pattern matching for high speed networks. In *Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines* (Washington, DC, USA, 2004), FCCM '04, IEEE Computer Society, pp. 249–257.
- [52] COLAJANNI, M., GOZZI, D., AND MARCHETTI, M. Enhancing interoperability and stateful analysis of cooperative network intrusion detection systems. In *Proceedings of the 3rd ACM/IEEE Symposium on Architecture for networking and communications systems* (Orlando, Florida, USA, 2007), ACM, pp. 165–174.
- [53] COOKE, E., JAHANIAN, F., AND MCPHERSON, D. The zombie roundup: understanding, detecting, and disrupting botnets. In *Proceedings of the Steps to Reducing Unwanted Traffic on the Internet on Steps to Reducing Unwanted Traffic on the Internet Workshop* (Cambridge, MA, 2005), USENIX Association, pp. 6–6.
- [54] COVINGTON, G. A., GIBB, G., LOCKWOOD, J. W., AND MCKEOWN, N. A packet generator on the netfpga platform. In *Field Programmable Custom Computing Machines, 2009. FCCM'09. 17th IEEE Symposium on* (2009), IEEE, pp. 235–238.
- [55] CULLER, D. E., GUPTA, A., AND SINGH, J. P. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann Publishers Inc., 1997.
- [56] DANTU, R., CANGUSSU, J., AND YELIMELI, A. Dynamic control of worm propagation. In *Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004. International Conference on* (2004), vol. 1, IEEE, pp. 419–423.
- [57] DAS, A., NGUYEN, D., ZAMBRENO, J., MEMIK, G., AND CHOUDHARY, A. An FPGA-based network intrusion detection architecture. *Information Forensics and Security, IEEE Transactions on* 3, 1 (2008), 118–132.
- [58] DHARMAPURIKAR, S. Design and implementation of a string matching system for network intrusion detection using FPGA-based bloom filters. In *Proc.*

of 12 th Annual IEEE Symposium on FieldProgrammable Custom Computing Machines (2004).

- [59] DHARMAPURIKAR, S., KRISHNAMURTHY, P., SPROULL, T., AND LOCKWOOD, J. Deep packet inspection using parallel bloom filters. *Micro, IEEE* 24, 1 (Jan 2004), 52–61.
- [60] DHARMAPURIKAR, S., AND LOCKWOOD, J. W. Deep packet inspection using parallel bloom filters. *Micro, IEEE* 24, 1 (2004), 52–61.
- [61] DHARMAPURIKAR, S., AND LOCKWOOD, J. W. Fast and scalable pattern matching for network intrusion detection systems. *Selected Areas in Communications, IEEE Journal on* 24, 10 (2006), 1781–1792.
- [62] EGOROV, S., AND SAVCHUK, G. SNORTRAN: An optimizing compiler for Snort rules. Tech. rep., Fidelis Security Systems, Inc., 2002.
- [63] EICK, S. G., LOCKWOOD, J. W., LOUI, R., MOSCOLA, J., AND WEISHAR, D. J. Transformation algorithms for data streams. In *Aerospace Conference, 2005 IEEE* (2005), pp. 1–10.
- [64] FENG, W., BALAJI, P., BARON, C., BHUYAN, L. N., AND PANDA, D. K. Performance characterization of a 10-gigabit ethernet TOE. In *High Performance Interconnects, 2005. Proceedings. 13th Symposium on* (2005), pp. 58–63.
- [65] FRINCKE, D., AND WILHITE, E. Distributed network defense. In *Proceedings of the 2001 IEEE Workshop on Information Assurance and Security* (2001), Citeseer, pp. 236–238.
- [66] GAMER, T., SCHARF, M., AND SCHÖLLER, M. Collaborative anomaly-based attack detection. In *Self-Organizing Systems*. Springer, 2007, pp. 280–287.
- [67] GARCIA, P., COMPTON, K., SCHULTE, M., BLEM, E., AND FU, W. An overview of reconfigurable hardware in embedded systems. *EURASIP J. Embedded Syst.* 2006, 1 (2006), 13–13. 1288236.
- [68] GOKHALE, M., DUBOIS, D., DUBOIS, A., BOORMAN, M., POOLE, S., AND HOGSETT, V. Granidt: Towards gigabit rate network intrusion detection technology. In *Field-Programmable Logic and Applications: Reconfigurable Computing Is Going Mainstream*, M. Glesner, P. Zipf, and M. Renovell, Eds., vol. 2438 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2002, pp. 404–413.
- [69] GORDON, L. A., LOEB, M. P., LUCYSHYN, W., AND RICHARDSON, R. 2004 CSI/FBI computer crime and security survey. *Computer Security Journal* 20, 3 (2004), 33–51.

- [70] GORMAN, D. M., MEZIC, J., MEZIC, I., AND GRUENEWALD, P. J. Agent-based modeling of drinking behavior: a preliminary model and potential applications to theory and practice. *American Journal of Public Health* 96, 11 (2006), 2055–2060.
- [71] GRANDISON, T., AND SLOMAN, M. A survey of trust in internet applications. *IEEE Communications Surveys & Tutorials* 3, 4 (2000), 2–16.
- [72] GUCCIONE, S., LEVI, D., AND DOWNS, D. A reconfigurable content addressable memory. In *Proceedings of the 15 IPDPS 2000 Workshops on Parallel and Distributed Processing* (2000), Springer-Verlag, pp. 882–889.
- [73] GUPTA, P., AND MCKEOWN, N. Packet classification on multiple fields. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication* (Cambridge, Massachusetts, United States, 1999), ACM, pp. 147–160.
- [74] HASHIM, F., KIBRIA, M., AND JAMALIPOUR, A. Detection of DoS and DDoS attacks in ngmn using frequency domain analysis. In *Communications, 2008. APCC 2008. 14th Asia-Pacific Conference on* (2008), pp. 1–5.
- [75] HAUCK, S. The roles of FPGAs in reprogrammable systems. *Proceedings of the IEEE* 86, 4 (1998), 615–638.
- [76] HE, X., PAPADOPOULOS, C., HEIDEMANN, J., MITRA, U., AND RIAZ, U. Remote detection of bottleneck links using spectral and statistical methods. *Comput. Netw.* 53, 3 (2009), 279–298.
- [77] HUBERMAN, B. A., AND ADAMIC, L. A. Internet: growth dynamics of the world-wide web. *Nature* 401, 6749 (1999), 131–131.
- [78] HUNTER, P. Hardware-based security: FPGA-based devices. *Computer Fraud & Security* 2004, 2 (2004), 11–12.
- [79] HUTCHINGS, B., FRANKLIN, R., AND CARVER, D. Assisting network intrusion detection with reconfigurable hardware. In *Field-Programmable Custom Computing Machines, 2002. Proceedings. 10th Annual IEEE Symposium on* (2002), pp. 111–120.
- [80] HUTCHINGS, B. L., FRANKLIN, R., AND CARVER, D. Assisting network intrusion detection with reconfigurable hardware. In *Field-Programmable Custom Computing Machines, 2002. Proceedings. 10th Annual IEEE Symposium on* (2002), pp. 111–120.
- [81] IGURE, V. M., AND WILLIAMS, R. D. Taxonomies of attacks and vulnerabilities in computer systems. *IEEE Communications Surveys & Tutorials* 10, 1 (2008), 6–19.

- [82] INTERNETWORLDSTATS.COM. World internet users statistics usage and world population. <http://www.internetworldstats.com/stats.htm/>.
- [83] JANAKIRAMAN, R., WALDVOGEL, M., AND ZHANG, Q. Indra: A peer-to-peer approach to network intrusion detection and prevention. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on* (2003), IEEE, pp. 226–231.
- [84] JEONG, H., TOMBOR, B., ALBERT, R., OLTVAI, Z. N., AND BARABÁSI, A.-L. The large-scale organization of metabolic networks. *Nature* 407, 6804 (2000), 651–654.
- [85] JUNG, H.-J., BAKER, Z. K., AND PRASANNA, V. K. Performance of FPGA implementation of bit-split architecture for intrusion detection systems. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International* (2006), p. 8 pp.
- [86] KANNAN, J., SUBRAMANIAN, L., STOICA, I., AND KATZ, R. H. Analyzing cooperative containment of fast scanning worms. In *Proceedings of Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI)* (2005), pp. 17–23.
- [87] KAUFMAN, C., PERLMAN, R., AND SPECINER, M. *Network security : private communication in a public world*, 2 ed. Prentice Hall series in computer networking and distributed systems. Prentice Hall PTR, Upper Saddle River, N.J., 2002.
- [88] KEIZER, G. Mydoom DoS attack on Microsoft falters, February 03 2004.
- [89] KENT, S., AND SEO, K. RFC 4301: Security architecture for the internet protocol, 2005.
- [90] KERNIGHAN, B. W., AND LIN, S. An efficient heuristic procedure for partitioning graphs. *Bell system technical journal* 49, 2 (1970), 291–307.
- [91] KOJM, T. ClamAV. <http://www.clamav.net>.
- [92] KUROSE, J., AND ROSS, K. *Computer Networking: A Top-Down Approach Featuring the Internet*. Addison-Wesley Longman Publishing Co., Inc., 2002.
- [93] KUZMANOVIC, A., AND KNIGHTLY, E. W. Low-rate tcp-targeted denial of service attacks: the shrew vs. the mice and elephants. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications* (New York, NY, USA, 2003), SIGCOMM '03, ACM, pp. 75–86.
- [94] LAKSHMAN, T. V., AND STILIADIS, D. High-speed policy-based packet forwarding using efficient multi-dimensional range matching, 1998. 285283 203–214.

- [95] LAWSON, M. V. *Finite Automata*. Chapman & Hall/CRC, Boca Raton, 2004.
- [96] LI, P., SALOUR, M., AND SU, X. A survey of internet worm detection and containment. *IEEE Communications Surveys & Tutorials* 10, 1 (2008), 20–35.
- [97] LI, S., TORRESEN, J., AND SORAASEN, O. Improving a network security system by reconfigurable hardware. In *proc. of 22nd Norchip Conference* (Oslo, Norway, 2004).
- [98] LI, S., TORRESEN, J., AND SORASEN, O. Exploiting stateful inspection of network security in reconfigurable hardware. In *Field-Programmable Logic and Applications*, vol. 2778/2003. Springer Berlin / Heidelberg, 2003, pp. 1153–1157.
- [99] LI, Z., HU, G., YAO, X., AND YANG, D. Detecting distributed network traffic anomaly with network-wide correlation analysis. *EURASIP Journal on Advances in Signal Processing* 2009 (2008).
- [100] LIN, P.-C., LI, Z.-X., LIN, Y.-D., LAI, Y.-C., AND LIN, F. C. Profiling and accelerating string matching algorithms in three network content security applications. *IEEE Communications Surveys & Tutorials* 8, 2 (2006), 24–37.
- [101] LIN, P.-C., LIN, Y.-D., LEE, T.-H., AND LAI, Y.-C. Using string matching for deep packet inspection. *Computer* 41, 4 (2008), 23–28.
- [102] LIN, Y.-D., WEI, H.-Y., AND YU, S.-T. Building an integrated security gateway: Mechanisms, performance evaluations, implementations, and research issues. *IEEE Communications Surveys & Tutorials* 4, 1 (2002), 2–15.
- [103] LO, C.-T. D., TAI, Y.-G., AND PSARRIS, K. Hardware implementation for network intrusion detection rules with regular expression support. In *Proceedings of the 2008 ACM symposium on Applied computing* (Fortaleza, Ceara, Brazil, 2008), ACM, pp. 1535–1539.
- [104] LOCKWOOD, J., NAUFEL, N., TURNER, J., AND TAYLOR, D. Reprogrammable network packet processing on the field programmable port extender (FPX). In *FPGA* (2001), pp. 87–93.
- [105] LOCKWOOD, J. W. An open platform for development of network processing modules in reprogrammable hardware. *IEC DesignCon01* (2001).
- [106] LOCKWOOD, J. W., MCKEOWN, N., WATSON, G., GIBB, G., HARTKE, P., NAOUS, J., RAGHURAMAN, R., AND LUO, J. Netfpga—an open platform for gigabit-rate network switching and routing. In *Proceedings of the 2007 IEEE International Conference on Microelectronic Systems Education* (Washington, DC, USA, 2007), MSE '07, IEEE Computer Society, pp. 160–161.
- [107] LOCKWOOD, J. W., MOSCOLA, J., KULIG, M., REDDICK, D., AND BROOKS, T. Internet worm and virus protection in dynamically reconfigurable hardware. In *Military and Aerospace Programmable Logic Device (MAPLD)* (2003), p. 10.

- [108] LOCKWOOD, J. W., MOSCOLA, J., REDDICK, D., KULIG, M., AND BROOKS, T. Application of hardware accelerated extensible network nodes for internet worm and virus protection, 2003.
- [109] LOINIG, J., WOLKERSTORFER, J., AND SZEKELY, A. Packet filtering in gigabit networks using FPGAs. In *Austrochip 2007 - Proceedings of the 15th Austrian Workshop on Microelectronics (2007)* (2007), pp. 53 – 60.
- [110] LONG, B., ZHANG, Z. M., PHILIP, S. Y., AND XU, T. Clustering on complex graphs. In *AAAI* (2008), pp. 659–664.
- [111] LUO, X., AND CHANG, R. K. C. On a new class of pulsing denial-of-service attacks and the defense. In *In Network and Distributed System Security Symposium (NDSS)* (2005), pp. 61–79.
- [112] MADHUSUDAN, B., AND LOCKWOOD, J. Design of a system for real-time worm detection. In *Proceedings of the High Performance Interconnects, 2004. on Proceedings. 12th Annual IEEE Symposium* (2004), IEEE Computer Society, pp. 77–83.
- [113] MARKATOS, E. P., ANTONATOS, S., POLYCHRONAKIS, M., AND ANAGNOSTAKIS, K. G. Exclusion-based signature matching for intrusion detection. In *Proceedings of the IASTED International Conference on Communications and Computer Networks (CCN)* (2002), pp. 146–152.
- [114] MATUSIAK, R. Implementing fast fourier transform algorithms of real-valued sequences with the TMS320 DSP family. In *Application Report. SPRA291 - August 2001 texas instruments* SPRA291 - August 2001 texas instruments, 2001.
- [115] MERMET, J. P. *Fundamentals and Standards in Hardware Description Languages: Proceedings of the NATO Advanced Study Institute, in Ciocco, Barga, Italy, April 16-26, 1993*. Kluwer Academic Publishers, 1993.
- [116] MILGRAM, S. The small world problem. *Psychology today* 2, 1 (1967), 60–67.
- [117] MIRKOVIC, J., ROBINSON, M., AND REIHER, P. Alliance formation for ddos defense. In *Proceedings of the 2003 workshop on New security paradigms* (2003), ACM, pp. 11–18.
- [118] MITRA, A., NAJJAR, W., AND BHUYAN, L. Compiling PCRE to FPGA for accelerating Snort IDS. In *Proceedings of the 3rd ACM/IEEE Symposium on Architecture for networking and communications systems* (Orlando, Florida, USA, 2007), ACM, pp. 127–136.
- [119] MOORE, D., SHANNON, C., VOELKER, G. M., AND SAVAGE, S. Internet quarantine: Requirements for containing self-propagating code. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies* (2003), vol. 3, IEEE, pp. 1901–1910.

- [120] MOSCOLA, J., LOCKWOOD, J., LOUI, R. P., AND PACHOS, M. Implementation of a content-scanning module for an Internet firewall. In *Field-Programmable Custom Computing Machines, 2003. FCCM 2003. 11th Annual IEEE Symposium on* (2003), pp. 31–38.
- [121] NAOUS, J., GIBB, G., BOLOUKI, S., AND MCKEOWN, N. Netfpga: reusable router architecture for experimental research. In *Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow* (2008), ACM, pp. 1–7.
- [122] NECKER, M., CONTIS, D., AND SCHIMMEL, D. TCP-stream reassembly and state tracking in hardware. In *Field-Programmable Custom Computing Machines, 2002. Proceedings. 10th Annual IEEE Symposium on* (2002), pp. 286–287.
- [123] NETWORKCOMPUTING.COM. Rolling review kickoff: Network behavior analysis systems. <http://www.networkcomputing.com/careers-and-certifications/rolling-review-kickoff-network-behavior-analysis-systems/d/d-id/1226292?>
- [124] NI, J., LIN, C., CHEN, Z., AND UNGSUNAN, P. A fast multi-pattern matching algorithm for deep packet inspection on a network processor. In *Parallel Processing, 2007. ICPP 2007. International Conference on* (2007), pp. 16–16.
- [125] OH, J.-T., PARK, S.-K., JANG, J.-S., AND JEON, Y.-H. Detection of DDoS and IDS evasion attacks in a high-speed networks environment. *International Journal of Computer Science and Network Security* 7, 6 (2007), 124–131.
- [126] OIKONOMOU, G., MIRKOVIC, J., REIHER, P., AND ROBINSON, M. A framework for a collaborative ddos defense. In *Computer Security Applications Conference, 2006. ACSAC'06. 22nd Annual* (2006), IEEE, pp. 33–42.
- [127] PAXSON, V. Bro: A system for detecting network intruders in real-time. In *Computer Networks* (1999), pp. 2435–2463.
- [128] PAXSON, V., ASANOVIC, K., DHARMAPURIKAR, S., LOCKWOOD, J., PANG, R., SOMMER, R., AND WEAVER, N. Rethinking hardware support for network analysis and intrusion prevention. In *Proceedings of the 1st USENIX Workshop on Hot Topics in Security* (Vancouver, B.C., Canada, 2006), USENIX Association, pp. 11–11.
- [129] PAXSON, V., SOMMER, R., AND WEAVER, N. An architecture for exploiting multi-core processors to parallelize network intrusion prevention. In *Sarnoff Symposium, 2007 IEEE* (2007), pp. 1–7.
- [130] PENG, M., AND AZGOMI, S. Content-addressable memory (CAM) and its network applications. In *International IC Taipei Conference Proceedings* (Taipei, 2001).

- [131] PENG, T., LECKIE, C., AND RAMAMOHANARAO, K. Survey of network-based defense mechanisms countering the DoS and DDoS problems. *ACM Comput. Surv.* 39, 1 (2007), 3.
- [132] PESCATORE, J., REYNOLDS, M., AND HALLAWELL, A. How to limit damage from the MyDoom worm. Tech. rep., Gartner, Inc., 2004.
- [133] PROAKIS, J. G., AND MANOLAKIS, D. K. Digital signal processing: Principles. In *Algorithms and Applications (4th Edition)*. Prentice Hall, 2006.
- [134] PTACEK, T. H., AND NEWSHAM, T. N. Insertion, evasion, and denial of service: Eluding network intrusion detection. Tech. rep., Secure Networks Inc, 1998.
- [135] RICHARDSON, R. Csi survey 2007: The 12th annual computer crime and security survey. 2007. *Computer Security Institute: San Francisco, CA*.
- [136] RICHARDSON, R. 14th annual csi computer crime and security survey. *CSI Computer Crime and Security Survey* (2009).
- [137] RICHARDSON, R. 15th annual 2010/2011 computer crime and security survey. *Computer Security Institute, New York, NY* (2011).
- [138] RICHARDSON, R., AND DIRECTOR, C. Csi computer crime and security survey. *Computer Security Institute 1* (2008), 1–30.
- [139] RINGBERG, H., SOULE, A., AND CAESAR, M. Evaluating the potential of collaborative anomaly detection. *Unpublished report* (2008).
- [140] ROESCH, M. Snort - lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX conference on System administration* (Seattle, Washington, 1999), USENIX Association, pp. 229–238.
- [141] ROESCH, M., AND GREEN, C. Snort users manual 2.8.3. http://www.snort.org/docs/snort_manual/.
- [142] SCARFONE, K., AND MELL, P. Guide to intrusion detection and prevention systems (IDPS). Tech. rep., National Institute of Standards and Technology, 2007.
- [143] SCHNACKENGERG, D., HOLLIDAY, H., SMITH, R., DJAHANDARI, K., AND STERNE, D. Cooperative intrusion traceback and response architecture (citra). In *DARPA Information Survivability Conference & Exposition II, 2001. DISCEX'01. Proceedings* (2001), vol. 1, IEEE, pp. 56–68.
- [144] SCHUEHLER, D. V., AND LOCKWOOD, J. Tcp-splitter: A TCP/IP flow monitor in reconfigurable hardware. In *High Performance Interconnects, 2002. Proceedings. 10th Symposium on* (2002), pp. 127–131.

- [145] SCHUEHLER, D. V., AND LOCKWOOD, J. W. A modular system for FPGA-based TCP flow processing in high-speed networks. In *14th International Conference on Field Programmable Logic and Applications (FPL)* (2004), pp. 301–310.
- [146] SCHUEHLER, D. V., MOSCOLA, J., AND LOCKWOOD, J. Architecture for a hardware based, TCP/IP content scanning system. In *High Performance Interconnects, 2003. Proceedings. 11th Symposium on* (2003), pp. 89–94.
- [147] SECURITYFORUM.ORG. Threat horizon 2014: Managing risks when threats collide. <http://www.crypto-gen.com/pdf/2014-executive-summary.pdf/>.
- [148] SHALUNOV, S., AND TEITELBAUM, B. TCP use and performance on internet2. In *ACM SIGCOMM Internet Measurement Workshop* (San Francisco, USA, 2001).
- [149] SIDHU, R., AND PRASANNA, V. K. Fast regular expression matching using FPGAs. In *Field-Programmable Custom Computing Machines, 2001. FCCM '01. The 9th Annual IEEE Symposium on* (2001), pp. 227–238.
- [150] SINGH, S., ESTAN, C., VARGHESE, G., AND SAVAGE, S. Automated worm fingerprinting. In *Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation - Volume 6* (San Francisco, CA, 2004), USENIX Association, pp. 4–4.
- [151] SINHA, R., PAPADOPOULOS, C., AND HEIDEMANN, J. Internet packet size distributions: Some observations. Technical Report ISI-TR-2007-643, USC/Information Sciences Institute, Los Angeles, 2007.
- [152] SIPSER, M. *Introduction to the Theory of Computation*. International Thomson Publishing, 1996.
- [153] SONG, H., AND LOCKWOOD, J. W. Efficient packet classification for network intrusion detection using FPGA. In *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays* (Monterey, California, USA, 2005), ACM, pp. 238–245.
- [154] SONG, H., SPROULL, T., ATTIG, M., AND LOCKWOOD, J. Snort offloader: A reconfigurable hardware NIDS filter. In *Field Programmable Logic and Applications, 2005. International Conference on* (2005), IEEE, pp. 493–498.
- [155] SOULE, A., RINGBERG, H., SILVEIRA, F., REXFORD, J., AND DIOT, C. Detectability of traffic anomalies in two adjacent networks. In *Passive and Active Network Measurement*. Springer, 2007, pp. 22–31.
- [156] SOURDIS, I., BISPO, J., CARDOSO, J. M., AND VASSILIADIS, S. Regular expression matching in reconfigurable hardware. *Journal of Signal Processing Systems* 51, 1 (2008), 99–121.

- [157] SOURDIS, I., AND PNEUMATIKATOS, D. Fast, large-scale string match for a 10gbps FPGA-based network intrusion detection system. In *Lecture notes in computer science*, vol. Volume 2778/2003. Springer Berlin / Heidelberg, Berlin; New York, 2003, pp. 880–889.
- [158] SOURDIS, I., AND PNEUMATIKATOS, D. Pre-decoded CAMs for efficient and high-speed NIDS pattern matching. In *Field-Programmable Custom Computing Machines, 2004. FCCM 2004. 12th Annual IEEE Symposium on* (2004), pp. 258–267.
- [159] SPEROTTO, A., SCHAFFRATH, G., SADRE, R., MORARIU, C., PRAS, A., AND STILLER, B. An overview of IP flow-based intrusion detection. *IEEE Communications Surveys & Tutorials* 12, 3 (2010).
- [160] SPITZNAGEL, E., TAYLOR, D., AND TURNER, J. Packet classification using extended tcams. In *Proceedings of the 11th IEEE International Conference on Network Protocols* (2003), IEEE Computer Society, p. 120.
- [161] SRINIVASAN, V., SURI, S., AND VARGHESE, G. Packet classification using tuple space search. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication* (Cambridge, Massachusetts, United States, 1999), ACM, pp. 135–146.
- [162] SURESH, D. C., GUO, Z., BUYUKKURT, B., AND NAJJAR, W. A. Automatic compilation framework for bloom filter based intrusion detection. In *Lecture notes in computer science*. Springer-Verlag, Berlin; New York, 2006, pp. 413–418.
- [163] TAYLOR, D. E. Survey and taxonomy of packet classification techniques. *ACM Comput. Surv.* 37, 3 (2005), 238–275.
- [164] TAYLOR, D. E., AND TURNER, J. S. Scalable packet classification using distributed crossproducting of field labels. In *IEEE INFOCOM 2005, 24th Annual Joint Conference of the IEEE Computer and Communications Societies* (2005), pp. 269–280.
- [165] TRIMBERGER, S. M. *Field-Programmable Gate Array Technology*. Kluwer Academic Publishers, 1994.
- [166] TUMMALA, A. K., AND PATEL, P. Distributed IDS using reconfigurable hardware. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International* (2007), pp. 1–6.
- [167] VIGNA, G., ROBERTSON, W., VISHAL, K., AND KEMMERER, R. A. A stateful intrusion detection system for World-Wide Web servers. In *Computer Security Applications Conference, 2003. Proceedings. 19th Annual* (2003), pp. 34–43.

- [168] VINCENT ZHOU, C., LECKIE, C., AND KARUNASEKERA, S. Decentralized multi-dimensional alert correlation for collaborative intrusion detection. *Journal of Network and Computer Applications* 32, 5 (2009), 1106–1123.
- [169] VUUREN, D. v. Death, taxes and worms white paper. Tech. rep., Dimension Data Holdings plc, 2004.
- [170] WAGNER, A., DÜBENDORFER, T., PLATTNER, B., AND HIESTAND, R. Experiences with worm propagation simulations. In *Proceedings of the 2003 ACM workshop on Rapid Malcode* (2003), ACM, pp. 34–41.
- [171] WASSERMAN, S. *Social network analysis: Methods and applications*, vol. 8. Cambridge university press, 1994.
- [172] WATTS, D. J., AND STROGATZ, S. H. Collective dynamics of small-worldnetworks. *nature* 393, 6684 (1998), 440–442.
- [173] WESTE, N. H. E., AND ESHRAGIAN, K. *Principles of CMOS VLSI Design: A Systems Perspective*, 2 ed. Addison-Wesley, 1993.
- [174] WILSON, T. Targeted attacks on the rise. <http://www.darkreading.com/security/perimeter/showArticle.jhtml?articleID=208804471>.
- [175] WONG, S., VASSILIADIS, S., AND COTOFANA, S. Future directions of (programmable and reconfigurable) embedded processors. In *In Embedded Processor Design Challenges, Workshop on Systems, Architectures, Modeling, and Simulation - SAMOS* (2002), Marcel Dekker, Inc.
- [176] XILINX. ChipScope Pro 12.3 Software and Cores - User Guide. http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_3/chipscope_pro_sw_cores_ug029.pdf/, 2010.
- [177] XU, T., ZHANG, Z., YU, P. S., AND LONG, B. Evolutionary clustering by hierarchical dirichlet process with hidden markov state. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on* (2008), IEEE, pp. 658–667.
- [178] YU, F., KATZ, R. H., AND LAKSHMAN, T. V. Gigabit rate packet pattern-matching using TCAM. In *Network Protocols, 2004. ICNP 2004. Proceedings of the 12th IEEE International Conference on* (2004), pp. 174–183.
- [179] YUSUF, S., AND LUK, W. Bitwise optimised CAM for network intrusion detection systems. In *Field Programmable Logic and Applications, 2005. International Conference on* (2005), pp. 444–449.
- [180] YUSUF, S., LUK, W., SLOMAN, M., DULAY, N., AND LUPU, E. A combined hardware-software architecture for network flow analysis. In *IEEE International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA '05)* (Las Vegas, USA, 2005).

- [181] YUSUF, S., LUK, W., SLOMAN, M., DULAY, N., LUPU, E. C., AND BROWN, G. Reconfigurable architecture for network flow analysis. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 16, 1 (2008), 57–65.
- [182] ZHANG, G., AND PARASHAR, M. Cooperative defence against ddos attacks. *Journal of Research and Practice in Information Technology* 38, 1 (2006), 69–84.
- [183] ZHOU, C. V., KARUNASEKERA, S., AND LECKIE, C. A peer-to-peer collaborative intrusion detection system. In *Networks, 2005. Jointly held with the 2005 IEEE 7th Malaysia International Conference on Communication., 2005 13th IEEE International Conference on* (2005), vol. 1, IEEE, pp. 6–pp.
- [184] ZHOU, C. V., LECKIE, C., KARUNASEKERA, S., AND PENG, T. A self-healing, self-protecting collaborative intrusion detection architecture to trace-back fast-flux phishing domains. In *Network Operations and Management Symposium Workshops, 2008. NOMS Workshops 2008. IEEE* (2008), IEEE, pp. 321–327.
- [185] ZHOU, Z., XIE, D., AND XIONG, W. A p2p-based distributed detection scheme against ddos attack. In *Education Technology and Computer Science, 2009. ETCS'09. First International Workshop on* (2009), vol. 2, IEEE, pp. 304–309.