gIR – A GPU BASED INTELLIGENT ROUTER

A PROJECT REPORT

Submitted By

S.PRIYA
V.SRIPRIYA
M.YAMINI

In partial fulfillment for the award of the degree

Of

Bachelor of Technology

In

INFORMATION TECHNOLOGY

DEPARTMENT OF INFORMATION SCIENCE AND TECHNOLOGY

ANNA UNIVERSITY,CHENNAI 600 025.

APRIL 2012

BONAFIDE CERTIFICATE

Certified that this project report titled **"gIR – A GPU BASED INTELLIGENT ROUTER"** is a *bona fide* work of **PRIYA S (20084496), SRIPRIYA V (20084530), YAMINI M (20084550)** who carried out the work under my supervision, for the partial fulfilment of the requirements for the award of the degree of *Bachelor of Technology* in *Information Technology*. Certified further that to the best of my knowledge and belief, the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or an award was conferred on an earlier occasion.

Place: Chennai.	**Dr.Ranjani Parthasarathi,**
	Professor,
Date:	Dept. of Information Science & Technology,
	Anna University,
Chennai - 600 025.	

COUNTERSIGNED

HEAD
Dept. of Information Science And Technology.
Anna University, Chennai 600 025

ACKNOWLEDGEMENT

First and foremost, we whole heartedly convey our sincere thanks and deep sense of gratitude to our guide Dr. Ranjani Parthasarathi, Professor, Department of Information Science and Technology, for her valuable guidance, suggestion and constant encouragement which paved the way for the successful completion of the project.

We owe our thanks to Dr. G. V. Uma, Professor and Head, Department of Information Science and Technology for giving us the opportunity to do this project.

We express our sincere gratitude to Dr. P. Narayanasamy,Dr. S. Swamynathan, Mr. N. Thangaraj, Mr. S. Sendhil Kumar the members of the Project Review Committee, for their valuable suggestions.We would like to thank all the teaching and non-teaching staff members of Department of Information Science and Technology for the help rendered during the course of the project. We also wish to thank our family and friends for their individual support and encouragement.

PRIYA .S

SRIPRIYA .V

YAMINI .M

TABLE OF CONTENTS

ABSTRACT (ENGLISH)

Modern routers need to provide increasing functionalities apart from traditional applications. With the growing internet traffic and complexity of packet processing task, the throughput of routers is affected. We propose gIR, a GPU based intelligent router to improve performance of router by off-loading computationally intensive tasks to GPU.

Recently, GPUs have gone beyond graphics processing to solve scientific applications; they are termed as General Purpose GPUs (GPGPU).The massively-parallel processing power and programming capability of GPU can be used to solve network related problems.

In gIR, we implement some of the core operations of router namely lookup, intrusion detection system for misuse and anomaly detection on a CUDA-enabled GPU. We exploit the inherent data parallelism in packet processing and task-level parallelism pertaining to lookup and intrusion detection system to improve and evaluate the performance of the system.

ABSTRACT (TAMIL)

நவீன திசைவிள் பாரம்பரிய பயன்பாடுகளைத் தவிர, அதிகரித்த செயல்பாடுகளை வழங்க வேண்டியுள்ளது. இணைய போக்குவரத்து வளர்ந்து வருவதின் காரணமாகவும், பாக்கெட் செயலாக்க பணி செக்கள் காரணமாகவும் ரவுட்டரின் செயல் பாதிக்கப்படுகிறது. நாங்கள் கிர் என்ற ஜி.பி.யூ சார்ந்த திசைவியை இத்திட்டத்தில் வழங்குகிறோம். இதில் திசைவியின் தீவிர பணிகளை ஜி.பி.யூவில் செலுத்தி, அதன் திறனை மேம்படுதுகிறோம்.

பொது நோக்கு ஜி.பி.யூ கிராபிக்காக மட்டும் இல்லாமல், அறிவியல் பயன்பாடுகளுக்க்காகவும் பயன்படுத்தப்பட்டு வருகிறது. ஜி.பி.யூவின் பெருமளவான இணையாகத் திறன் மற்றும் நிரலாக்கத் திறன், பிணைய தொடர்புடைய சிக்கல்களைத் தீர்க்க பயன்படுத்தப்பட்டு வருகிறது.

இத்திட்டத்தில், திசைவியின் முக்கிய பணிகளை CUDA-ஜி.பி.யூவில் செயல்படுத்த்துகிறோம். இதில் தரவு மற்றும்

பாக்கெட்டில் உள்ள இணையத்தை பயன்படுத்துவதோடு மட்டுமில்லாமல் இக்கண்டுபிடிப்பை மதிப்பீடும் செய்கிறோம்.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AC	Aho-Corasick
AES-CBC	Advanced Encryption Standard – Cipher Block Chaining
AH	Authentication Header
APNIDS	Anomaly Payload Network Intrusion Detection system
ART	Adaptive Resonance Theory
BISOM	Batch Incremental Self- Organizing Map
CIDR	Classless Inter-Domain Routing
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
CW-B	Commentz-Walter
DES-CBC	Data Encryption Standard – Cipher Block Chaining
DFA	Deterministic Finite Automata
DMA	Direct Memory Access
DPI	Deep Packet Inspection
ESP	Encapsulated Security Payload
FTP	File Transfer Protocol
GPGPU	General Purpose Graphic Processing Unit

GPU	Graphic Processing Unit
HIDS	Host Intrusion Detection System
HTTP	Hyper-Text Transfer Protocol
ICMP	Internet Control Message Protocol
IP	Internet Protocol
IPSec	Internet Protocol Security
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
ISAKMP	Internet Security Association and Key Management Protocol
LIP Hdr	Length of IP Header
LPM	Longest Prefix Matching
MDS	Misuse Detection System
NIDS	Network Intrusion Detection System
PCRE	Perl Compatible Regular Expressions
RE	Regular Expressions
RS	Rule Subset
SA	Security Association
SADB	Security Association Database
SIMD	Single Instruction Multiple Data
SIMT	Single Instruction Multiple Thread

SM	Stream Multiprocessor
SOM	Self Organizing Map
SP	Stream Processor
SPI	Security Parameter Index
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
WM	Wu-Manber

CHAPTER 1

INTRODUCTION

1.1 Motivation

Today’s networking trends and technologies put a high pressure on the packet processing operations of the core network. The trends and technologies include security threats, incoming of 4G network, and transition from v4 to v6 networks. The current processors become a bottleneck because there is much more data to process and the processing operations are increasingly complicated.

This problem may be solved by using specialized network processing equipments such as Network Processing Unit (NPU). Nonetheless, the solutions may not be cost effective and easy to manage because they require specific hardware and configurations.

This thesis approaches the problem differently by offloading network packet processing to Graphic Processing Units (GPU). GPUs have high computation power because they are designed for computation intensive applications such as premium quality video or interactive games. And, GPUs already exist in general PC configurations. As a result, the cost of ownership of this approach would be lower compared to the dedicated hardware solutions. Furthermore, because there is no specialized hardware configurations required, the system would be simpler to manage.

1.2 Problem Statement

A substantial increase of traffic in today’s Internet has posed a great challenge for complex packet processing operations. The ever-increasing traffic has resulted in higher link speeds, which in turn causes the current routers to compromise security. So, the design and implementation of future routers should strike a right balance between speed and security. Also, with organizations shifting to IPv6 networks, there is a need for routers to support both protocols in this transition stage. They should also support interoperability for both v4 and v6 networks.

1.3 Proposed Solution

Our work primarily focuses on speeding up the computationally intensive operations in packet processing by harnessing the massive parallel computing ability of GPU. Our router is designed to route traffic from heterogeneous networks by implementing Dual-IP. We block the traffic to blacklisted addresses and monitor the network for both known and unknown attacks, providing high level of security. We experiment with CUDA-enabled GPUs to boost the performance of the operations discussed above.

1.4 Motivation for a GPU Based Router

Normally GPUs were used for solving problems in graphics domain. Usage of GPUs for more general calculations required mapping of tasks to graphics domain and solving it there. This approach became difficult for problem domains that do not map well to computer graphics. This led to the development of GPUs for general purpose computing called GPGPUs.

GPGPUs are being used to solve complex mathematical operations. GPGPUs generally target applications having large data sets, high parallelism, minimum dependency between data elements, high arithmetic intensity. Some applications that use GPGPUs include weather forecasting, cryptography, computer vision etc.

Network processing is one such domain which is suitable for GPGPUs as it has large amount of packets to be processed with minimum dependency between them, thus providing a larger scope for parallelism. But the application of GPGPUs in network processing is still an open challenge. In this work, we restrict ourselves to accelerating the performance of routing and NIDS. The next section briefs about CUDA architecture.

1.5 Overview of CUDA

CUDA (Compute Unified Device Architecture) is NVIDIA's software suite of libraries and a compiler, and it provides an API to GPU functionalities.

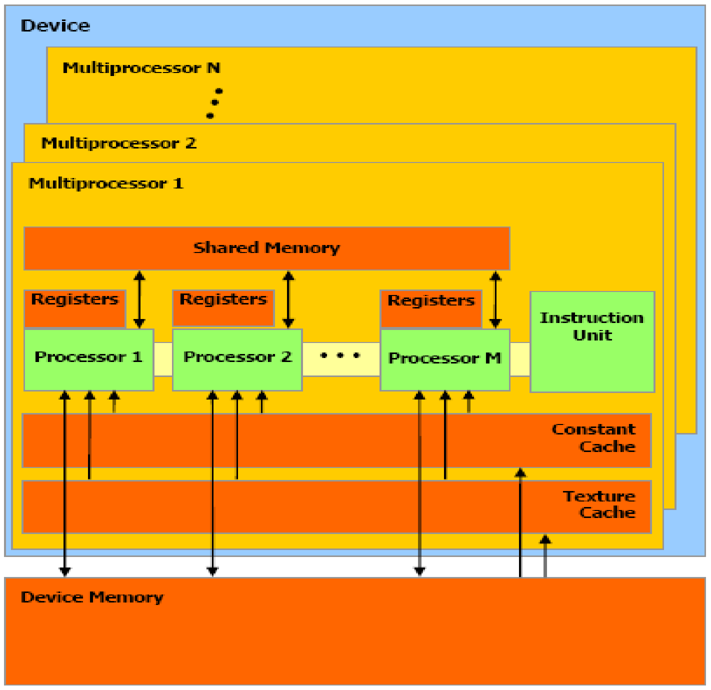

Figure 1.1 CUDA Architecture

In this work, we have used Nvidia GeForce GT 220 GPU. It has 48 cores with 6 Streaming Multiprocessors (SMs), each of which has 8 Stream Processors (SPs). All the threads running on SPs share the same program called kernel. Figure 1.1 shows the overall CUDA architecture.

An SM works as a collection of independent SIMTs (Single Instruction Multiple Threads). The basic execution unit of SM is a warp, a group of 32 threads sharing the same instruction pointer. All threads in a warp take the same code path. Warp switching is done by hardware, incurring zero context-switching overhead. Compared to SIMD (Single Instruction Multiple Data) architecture where all data elements are processed in same way, SIMT behavior of GPU gives more flexibility.

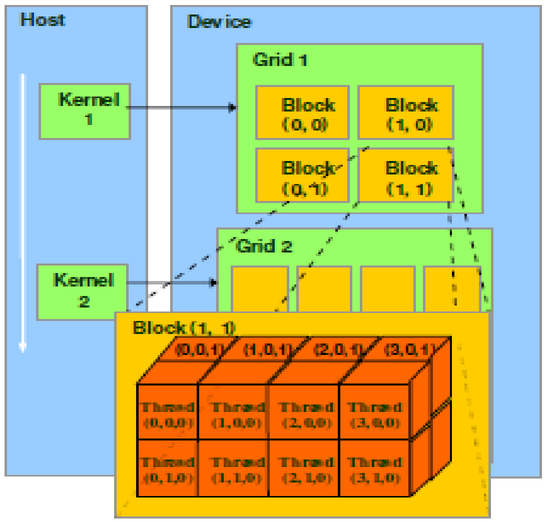

Figure 1.2 CUDA Thread Organization

CUDA organizes the parallel computation using the abstractions of threads, blocks and grids as indicated in Figure 1.2. Each SP handles one or more threads in a block. The block is a collection of threads and each SM handles one or more blocks in a grid. A grid is a collection of blocks and entire grid is handled by a single GPU chip.

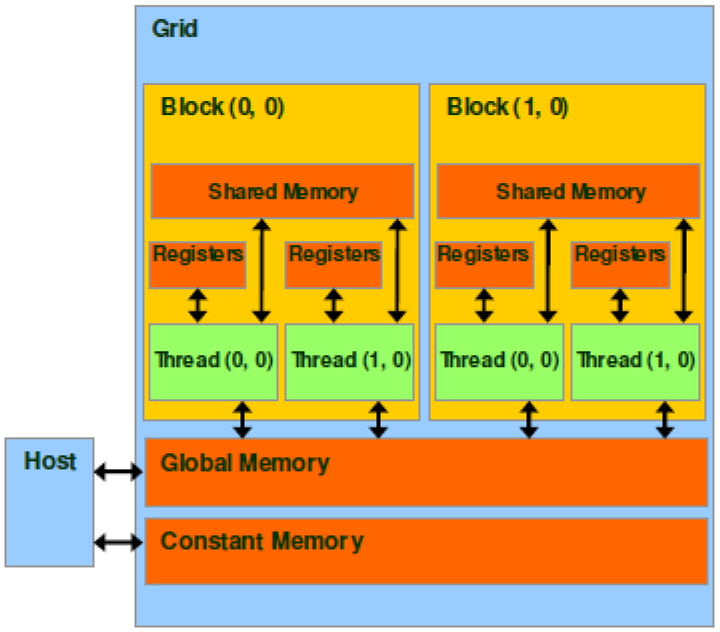

Figure 1.3 CUDA Memory Architecture

Figure 1.3 shows the CUDA memory architecture. Each CUDA device has several memories, which the programmers can use to achieve high execution speed in the kernels. The bottom of the picture shows global memory and the constant memory. The host code can read and write to these memories using suitable APIs. The constant memory allows read-only access by the device and provides faster and more data access path than the global memory.

The registers and shared memory, as shown in Figure 1.3, can be accessed in parallel. Registers are allocated to individual threads. Each thread can only access its own registers. Shared memories are allocated to thread blocks. All the threads in a block can access the variables in the shared memory allocated to the block. The shared memories are efficient means to allow threads to co-operate by sharing the results of their work.

1.6 Overview of Thesis Structure

Chapter 2 discusses the previous work carried out in various domains pertaining to our proposed system. Chapter 3 gives the detailed architecture of our system. Chapter 4 discusses the implementation and parallelization carried out in proposed system. Chapter 5 provides the results and analysis of our work. The scope for future experiment is discussed in Chapter 6.

CHAPTER 2

LITERATURE SURVEY

This chapter discusses the current state of art in routing and NIDS highlighting the advantages and disadvantages of each work and has attempted to address these issues in our work.

2.1 Routing

IP lookup mechanism is a key issue in designing a router. IP lookup is an important action in a router, which finds the next hop of each incoming packet with a longest-prefix-match address in the routing table.

Due to the large size of the routing tables, CIDR representation of addresses was used. CIDR decreases routing table size by allowing blocks of addresses to be grouped into single routing table entries. But CIDR approach requires routers to perform longest-prefix matching, instead of exact matching. Longest-prefix matching makes IP lookup complicated since it involves not only comparing the bit pattern, but also finding the appropriate prefix length.

With the advent of IPV6 addresses, the existing IPv4 lookup schemes are not scalable. Thus there is a need for generic lookup scheme that is easily scalable for both v4 and v6.

Several LPM algorithms have been proposed. Some of them are discussed below. The main idea of [11] is binary search on prefix length using hash tables. The idea in [12] is to develop bloom filters for unique prefix lengths. The destination address is hashed to find which bloom filter it belongs to and the corresponding prefix entries are traversed to find the next hop. [14] uses binary search on prefix intervals for finding the LPM. Other techniques for finding the longest matching prefix are all based on the trie data structure.

They adopt sophisticated ideas such as path compression [8], prefix expansion [2], level compression [9] to reduce the worst case memory access times per lookup. Path compression technique for lookup involves reducing the skip count (number of non-terminating nodes, traversed) while traversing the trie for finding a match. Prefix expansion technique operates by extracting multiple bits at a time and using them as an index of array pointers to traverse the child nodes. In Level Compression tries, whenever a trie node does not have terminal nodes for the next t levels, the fan-out trie is compressed into a single array of size 2^t and t bits are used to index into this array of pointers to nodes.

These LPM algorithms were particularly designed for 32-bit-long IPv4 address, and had poor scalability to the length of IPv6 addresses. Also, these schemes are variations of basic binary trie (a tree-like structure). As, a result decision for choosing a child node at each level is needed for traversal. This makes it difficult to implement these schemes in a parallelized manner.

2.2 NIDS

Intrusion Detection System is generally classified as Misuse Detection and Anomaly based detection system. Implementation of one of these is insufficient to form a complete detection System and often a combination yields better results. Current NIDS systems also suffer from evasion attacks by the implementation of IPSec.

Misuse Detection

The heart of misuse detection includes pattern matching with known signatures and rules. Often, this is considered to be processor intensive, taking into account the amount of packet data to be matched against a huge rule set. Several pattern matching algorithms have been proposed. Some of them are discussed below.

The Commentz-Walter (CW-B and CW-B1) algorithm [7] was proposed for NIDS system in which the worst-case running time complexity scales linearly with input length. But this algorithm suffers from heavy memory usage. The Wu-Manber (WM) algorithm [7] has been used in Snort for its fast sub-linear average-case speed; however it performs badly under an algorithmic complexity attack and has a quadratic (in n) worst-case running time. Further, both of these algorithms don't scale well for wide range of keyword lengths. Algorithms based on hash functions [13] were also proposed in which the hash value for pattern of length n and input text of length n was computed. But they don't scale up well as the number of patterns and pattern length increases. To overcome these deficiencies, AC (Aho-Corasick) algorithm was implemented in Gnort [3], a GPU-based intrusion detection system. Gnort uses DFA, where the transition table is stored as 2-D array of states and alphabets. This proliferate memory usage, which further worsens when the number of states in automaton increases.

Anomaly Detection

Most Anomaly Detection Systems are header based and are efficient in identifying only flooding attacks, port scanning etc. rather than examining each packet for stealthy attacks. So we need to inspect the payload to detect polymorphic worms and zero-day attacks.

Most of the header based detection system uses neural networks like ART-1, ART-2,SOM [4],[6] to provide real time intrusion detection. Of these ART-1 provides better detection rate due to their stability and plasticity. The main drawback in deploying these networks is their high training and testing time.

A number of research papers have also focused on payload-based intrusion detection. However, most of these systems are designed for particular port or service rather than for network traffic as whole. The high-dimensionality of network data inhibits the detection system to monitor entire network and hence these detection systems are deployed in most commonly used ports (port 80 for HTTP, port 23 for FTP). Thus there is a great possibility for intruder to launch attacks in rarely used ports.

Some work on payload intrusion detection namely PAYL [15], POISEON[5] uses statistical based model to create normal profile. PAYL uses byte frequency distribution model and Standard deviation to create normal payload model in training phase. In detection phase, the system uses Mahalanobis distance to calculate deviation of incoming packet payload from normal profile. POISEON is based on PAYL, in addition that it uses SOM to perform clustering of payload before passing into PAYL system. Both PAYL and POISEON have high detection rate but are subjected to Mimicry attacks. To overcome this, ANAGRAM[1] was proposed by considering significantly anomalous byte sequence rather than single byte frequency distribution. The mixture of higher-order N-Grams of a service payload was stored in memory using Bloom Filters. It is a semi-supervised system where separate model is created for normal and bad content traffic, thus reducing false positive rate and increasing detection rate.

With this analysis of the current solutions, we now propose the design of our system in the following chapter.

CHAPTER 3

DESIGN

This chapter presents the proposed framework for the router, the design philosophy and the detailed design of each module.

3.1 Proposed Framework

The Figure 3.1 shows the scenario for our framework. The network under consideration is IPv6 only. We propose a framework to build the edge-router with the following capabilities:

- Interoperability with both IPv4 and IPv6 networks
- Security against known and unknown attacks
- Increased throughput

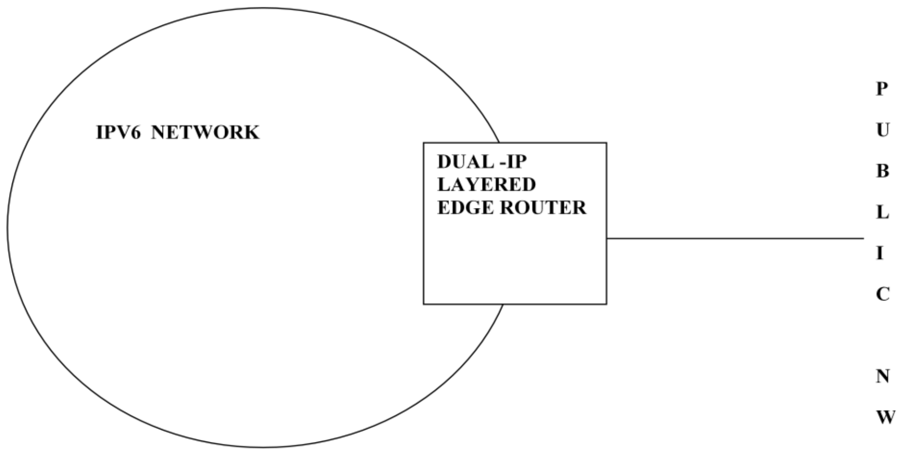

Figure 3.1 Scenario under study

The overall framework is shown in Figure 3.2.The inbound and outbound traffic are first checked against blacklisted entries. The legitimate packets of heterogeneous networks are routed with the help of Dual-IP transition functionality of the edge-router.

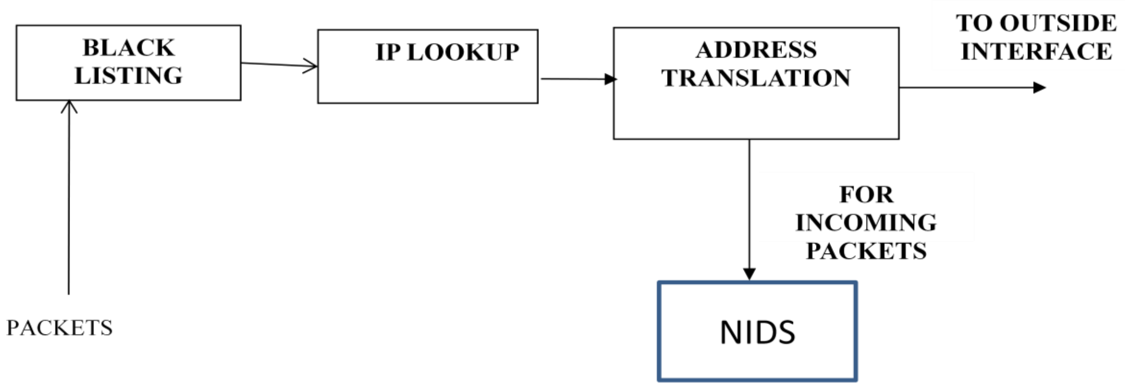

Figure 3.2 Proposed Solution

NIDS is deployed to secure internal network. The main issue associated with NIDS is the evasion attack by encryption. IPSec, that is optional in v4, has become mandatory in v6.Hence Deep packet inspection is not possible. We address this issue by implementing 2-key IPSec to strike a balance between privacy and security. The sniffed packets are then inspected for both known and unknown attacks. The combination yields better detection when compared to individual deployment.

3.2 Design Philosophy

The design technique used for our router is discussed in this section. In lookup we deploy bloom filters to perform LPM. A Bloom filter is essentially a bit-vector of length *m* used to efficiently represent a set of messages. Given a set of messages *X* with *n* members, the Bloom filter is indicated as follows. For each message *xi* in *X*, *k* hash functions are computed on *xi* producing *k* values each ranging from 1 to*m*. Each of these values address a single bit in the *m*-bit vector, hence each message *xi* causes *k* bits in the *m*-bit vector to be set to 1. The algorithm performs parallel queries on Bloom filters, in order to determine address prefix membership in sets of prefixes sorted by prefix length. Our approach is equally scalable for both V4 and V6 addresses. Also, we have implemented Dual-IP protocol to support co-existence of V4 and V6 networks.

For NIDS, we propose a complete detection system by combining both misuse and anomaly detection techniques to uncover both known and unknown attacks. Further - more we prevent evasion of NIDS by encryption using Two-key IPSec [10]. Two-Key IPSec is a form of selective encryption on IP packet by breaking the packet into multiple encryption zones. Each encryption zone is assigned a cryptographic key and a portion of the IP packet over which that key is used for encryption and decryption. Selective encryption enables a well-designed NIDS to detect network-wide events. In MDS we perform pattern matching using AC algorithm, wherein DFA's are constructed for Snort rules as proposed in Gnort. We further optimize this work by reducing the memory usage by constructing reduced transition tables. To improve the efficiency of the system we perform two stage decomposition of snort rule-set in CPU which reduces the number of rules to be matched against a packet in GPU. In anomaly based detection system We try to implement the traditionally sequential ART-1 algorithm in a parallel manner to reduce training time. The designs of individual modules are explained below.

3.2.1Blacklisting

Figure 3.3 shows how blacklisting is done. A blacklist containing a set of IP addresses is maintained. The source and destination address of each packet is extracted and is checked to see if it is blacklisted. If so, then the packet is dropped else it is routed.

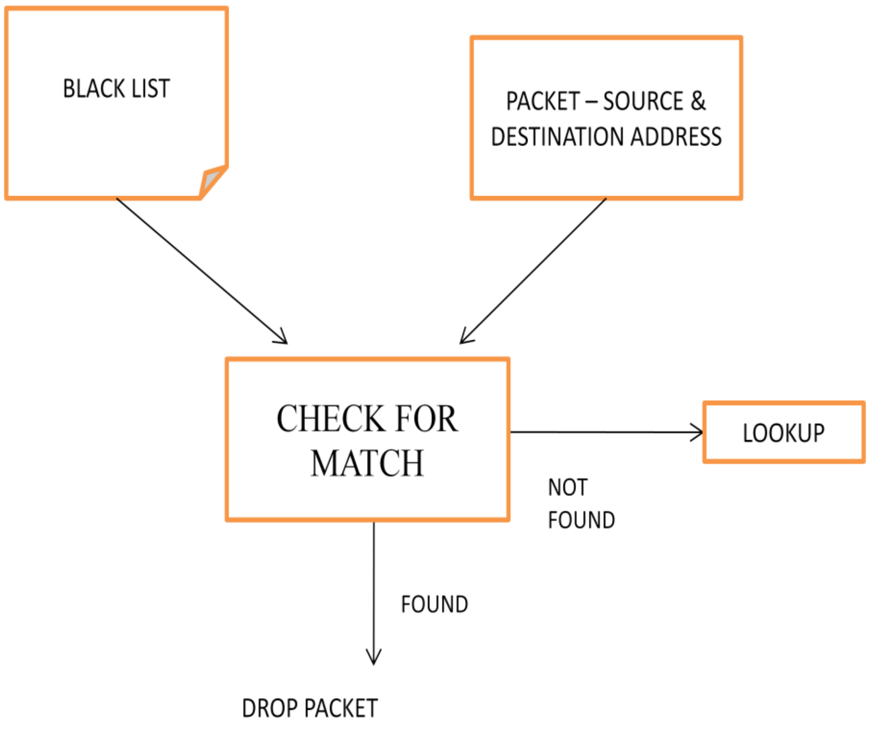

Figure 3.3 Blacklisting

3.2.2 IP-Lookup

Bloom filters are constructed for unique prefix length entries. The destination of the packet is hashed to check which bloom filter it belongs to. Then the corresponding entries of the prefix length are iterated to find the next hop. Fig 3.4 details the steps for performing lookup.

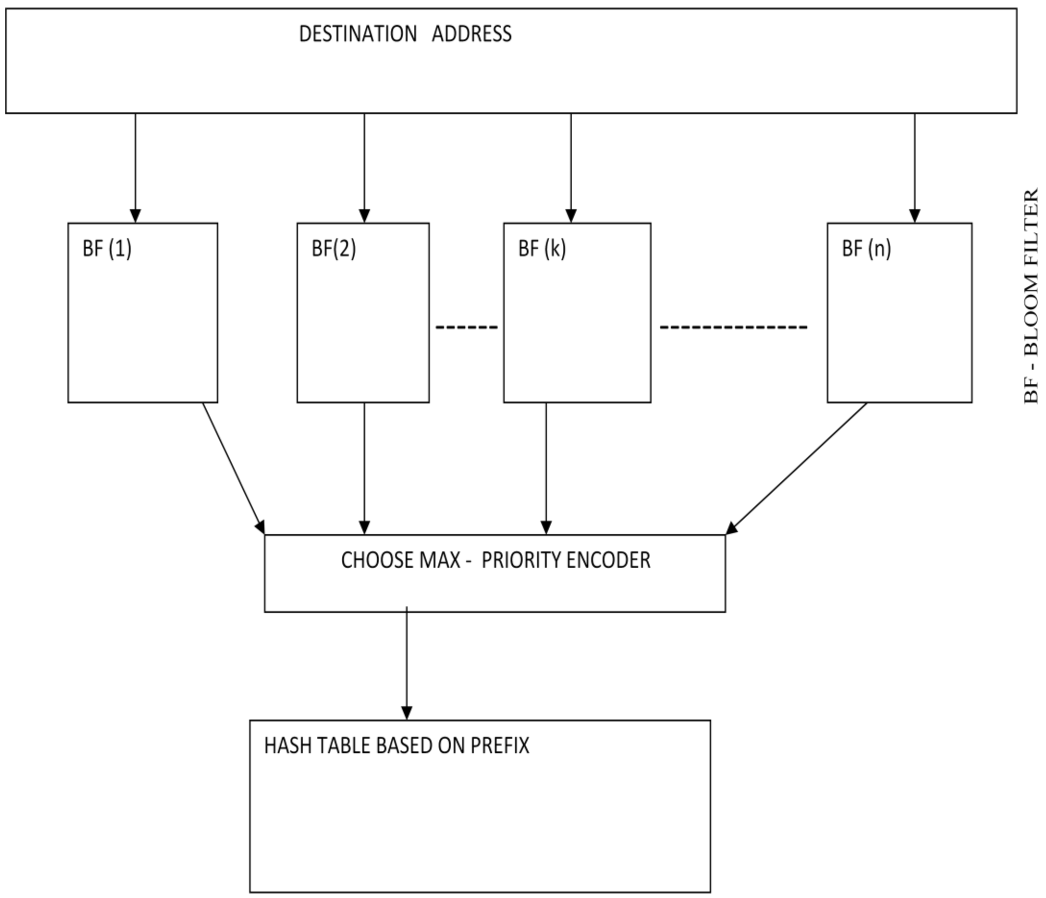

Figure 3.4 Lookup with Bloom Filters

The bloom filter implementation steps are as follows:

1. Group the prefixes in the table into sets according to their length.
2. For each available prefix construct a bloom filter
3. Set-up hash tables in the form <<prefix, next hop>>. The number of hash tables depend on the number of unique prefixes in the table.
4. Lookup()
    a. The input is the destination address.
    b. Extract the bits for each available prefix length
    c. Hash the destination address with the hash functions.

d. Check the bloom vector to see if the value in the corresponding index is 1.
e. If more than one prefix length matches, choose the maximum
f. Check the hash table corresponding to the prefix length to obtain the next hop.

3.2.3 Dual-IP Transition

Figure 3.5 shows the steps involved in dual-IP transition.

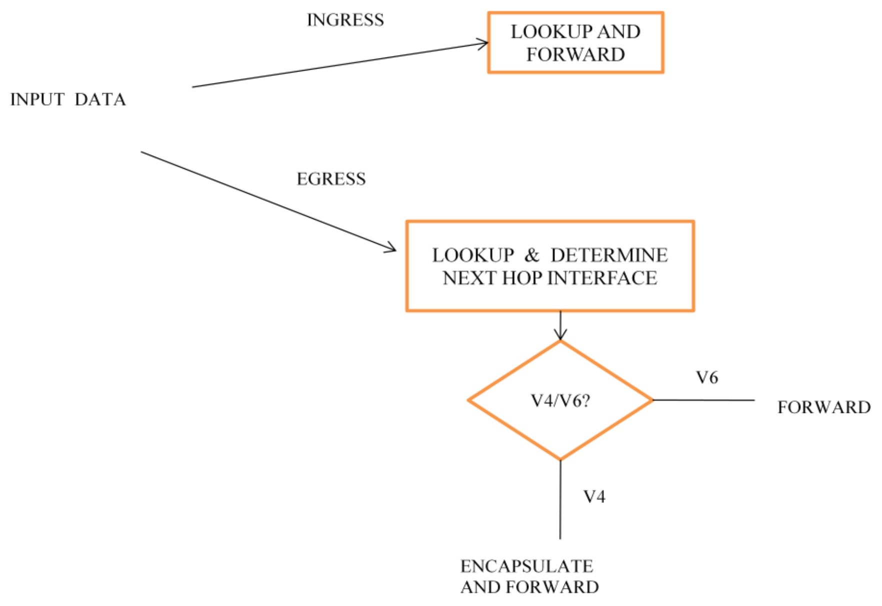

Figure 3.5 Dual IP Transition

In our implementation the network under consideration is a v6-only network. In dual-IP technique, the ingress packets are checked for v6 or tunneled v4. If it is tunneled, then the data portion is extracted to get the original v6 address. The packet is then routed via lookup. The v6 packets are routed normally. For egress packets, while lookup is done, the type of protocol of the next hop is determined. If the next hop is v4, then mapped address is provided and the necessary v4-packet is sent, else the v6-packet is sent.

3.2.4 NIDS

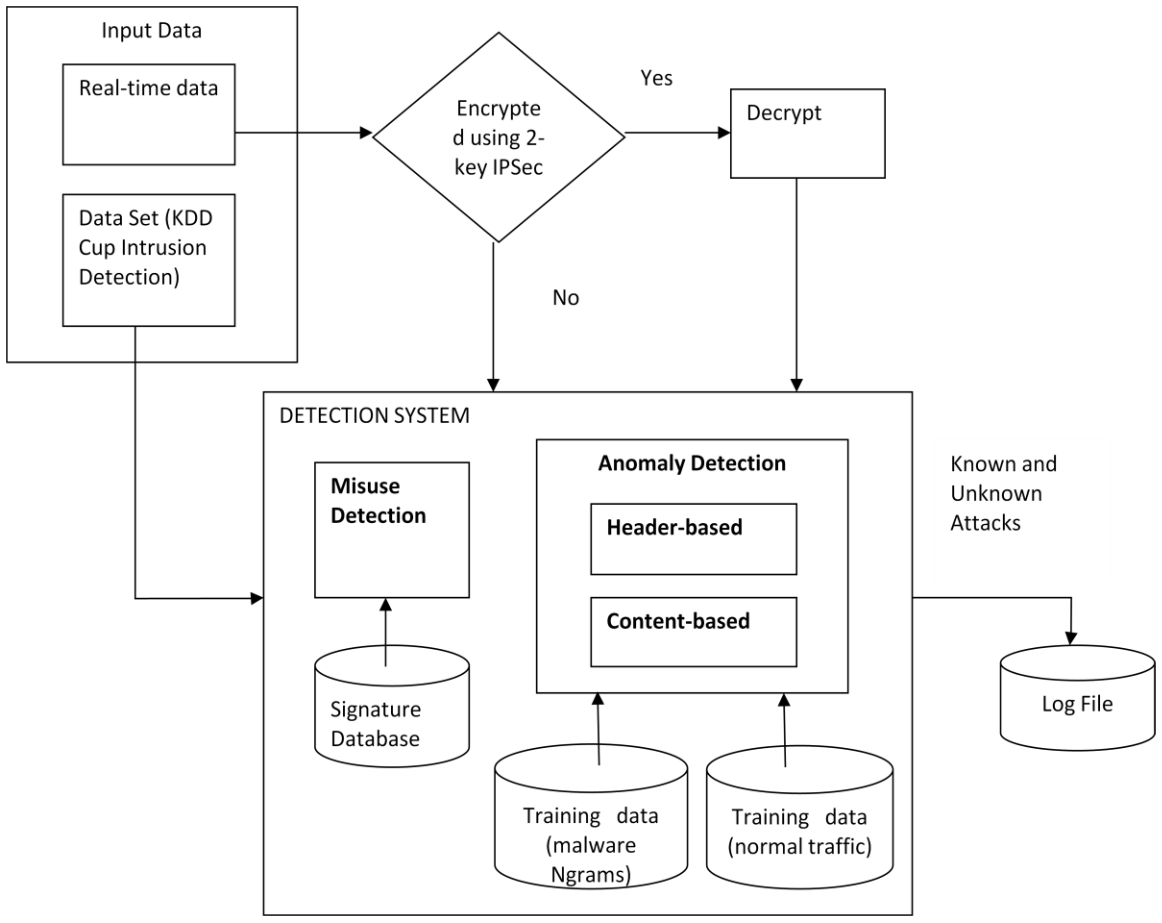
Input Data
Real-time data
Data Set (KDD Cup Intrusion Detection)
Encrypted using 2-key IPSec
Yes
No
Decrypt
DETECTION SYSTEM
Misuse Detection
Signature Database
Anomaly Detection
Header-based
Content-based
Training data (malware Ngrams)
Training data (normal traffic)
Known and Unknown Attacks
Log File

Figure 3.6 Network Intrusion Detection System

The overall architecture of NIDS as explained in section 3.2 is shown in Figure 3.6. The designs of individual modules in NIDS are discussed below.

3.2.4.1 IPSec

NIDS requires more than the source and destination addresses of a packet to detect events and they often examine packet contents. This information is hidden when traffic is encrypted with IPSec. In order to allow DPI of the incoming packet by NIDS, selective encryption is employed and we restrict our number of encryption zones to two. One allows inspection of the header and part of payload, the other includes the entire payload. Complete inspection of payload is possible only by HIDS.

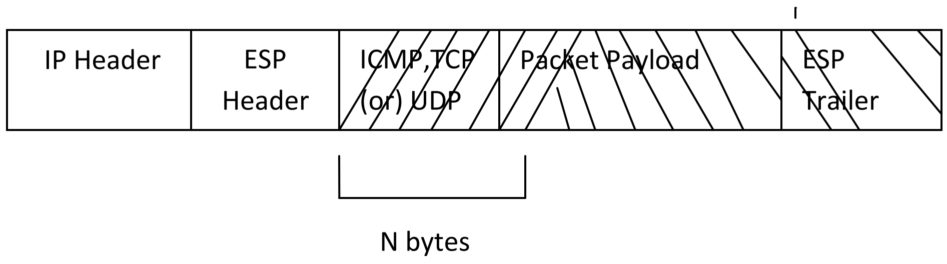

Figure 3.7 Transport Mode

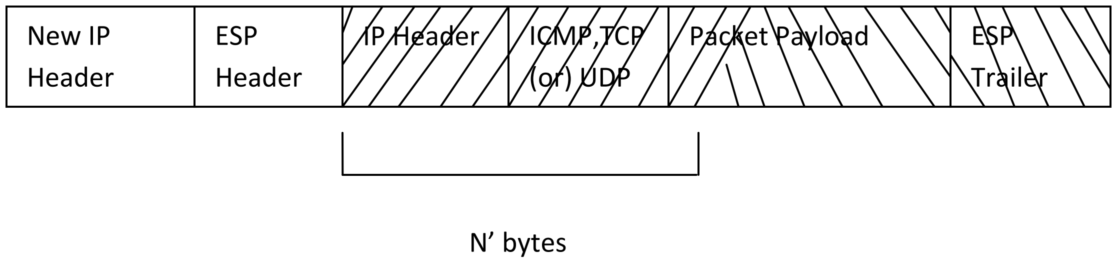

Figure 3.8 Tunnel Mode

IPSec operates in two modes:

Transport Mode

Transport mode is the default mode for IPSec, and it is used for end-to-end communications (for example, for communications between a client and a server). When transport mode is used, IPSec encrypts only the IP payload. Transport mode provides the protection of an IP payload through an AH or ESP header.

Tunnel Mode

When IPSec tunnel mode is used, IPSec encrypts the IP header and the payload. Tunnel mode provides the protection of an entire IP packet by treating it as an AH or ESP payload. With tunnel mode, an entire IP packet is encapsulated with an AH or ESP header and an additional IP header.

The two packet diagrams in Figure 3.7 and 3.8, the two hashed areas show the effect of using Two-Key IPsec to encrypt the bytes of the packet with two separate keys, K1 and K2. The first N bytes of the ESP payload are encrypted with the second key, K2. Note that for IPsec tunnel mode, N′ bytes are encrypted using K2, where N'= N + LIP Hdr and LIP Hdr is the length of the encapsulated IP header. K2 is shared among the end hosts and the network-based service. K1 is used to encrypt the remainder of the ESP payload and, as with traditional IPsec, is known only to the end points.

Figure 3.9 represents IPSec overview. IPSec module involves decryption of the incoming packet for inspecting the payload content. For decryption we need the algorithm used and the key to decrypt. The two end points first exchange the security association which identifies a particular packet using the 4 tuple [src addr | dest addr | SPI(security parameter index) | protocol] and makes a corresponding entry into an SADB(Security Association Database).The incoming packet is matched against an SADB entry and its corresponding algorithm and key values are retrieved for decryption. Security association and key exchange is done using ISAKMP protocol.

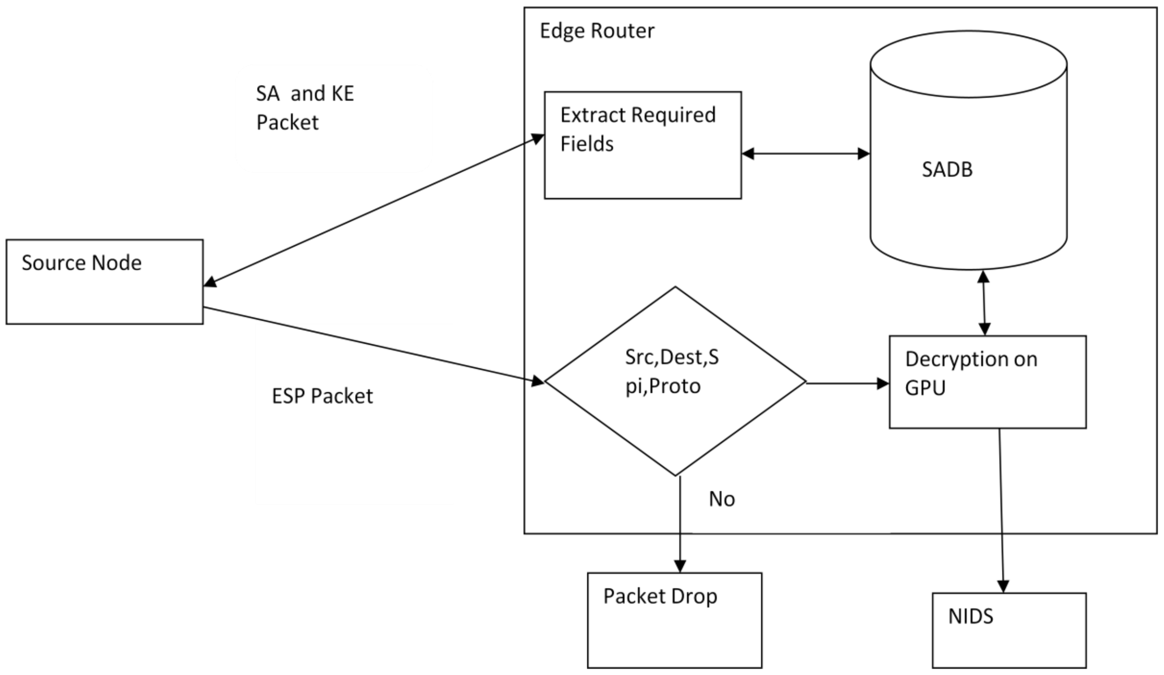

Figure 3.9 IPSec Overview

Security association packet, shown in Figure 3.10, contains SA header followed by proposals and transforms. There can be any number of proposals and transforms in an SA packet. Proposals contain information about the protocol, SPI etc and transforms contain the algorithm used and the length of the key. Key exchange part of the SA packet contains the key, start byte and offset to be decrypted in case of two key IPSec.

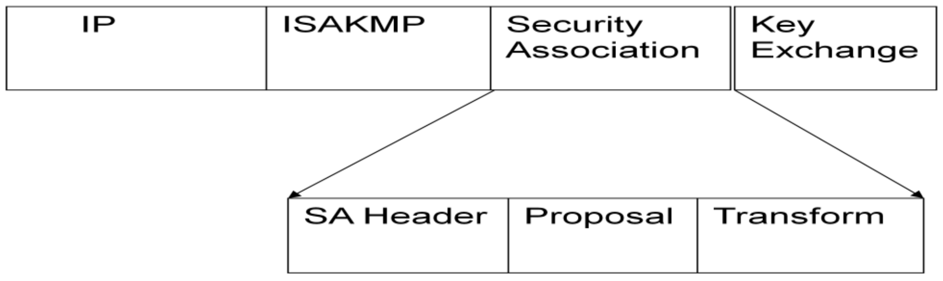

Figure 3.10 SA Packet Format

3.2.4.2 Misuse Detection System

Intrusion detection system involves inspecting the packet against known set of signatures which are considered as anomalous. In case the signature is found then packet is logged which can be used to generate alerts and for further inspection. This system involves three phases:

1.Pre-Processing of signatures

2.Detection Phase

3.Logging anomalous data.

Pre-Processing Phase

Signature set we are using is Snort version 2.9.Snort signatures consists of two parts

- Rule Header
- Rule Options

Snort signatures are many in number ranging around 30,000 and it also consist of redundancies. Hence it is a necessity to group snort rules based on the protocol in the rule header and other header fields like src addr,dest addr,src port and dest port. This two stage decomposition of rules removes the redundant patterns inherent in the rules and also reduces the no of rules to be matched against a packet. Figure 3.11 shows the SNORT rule set processing.

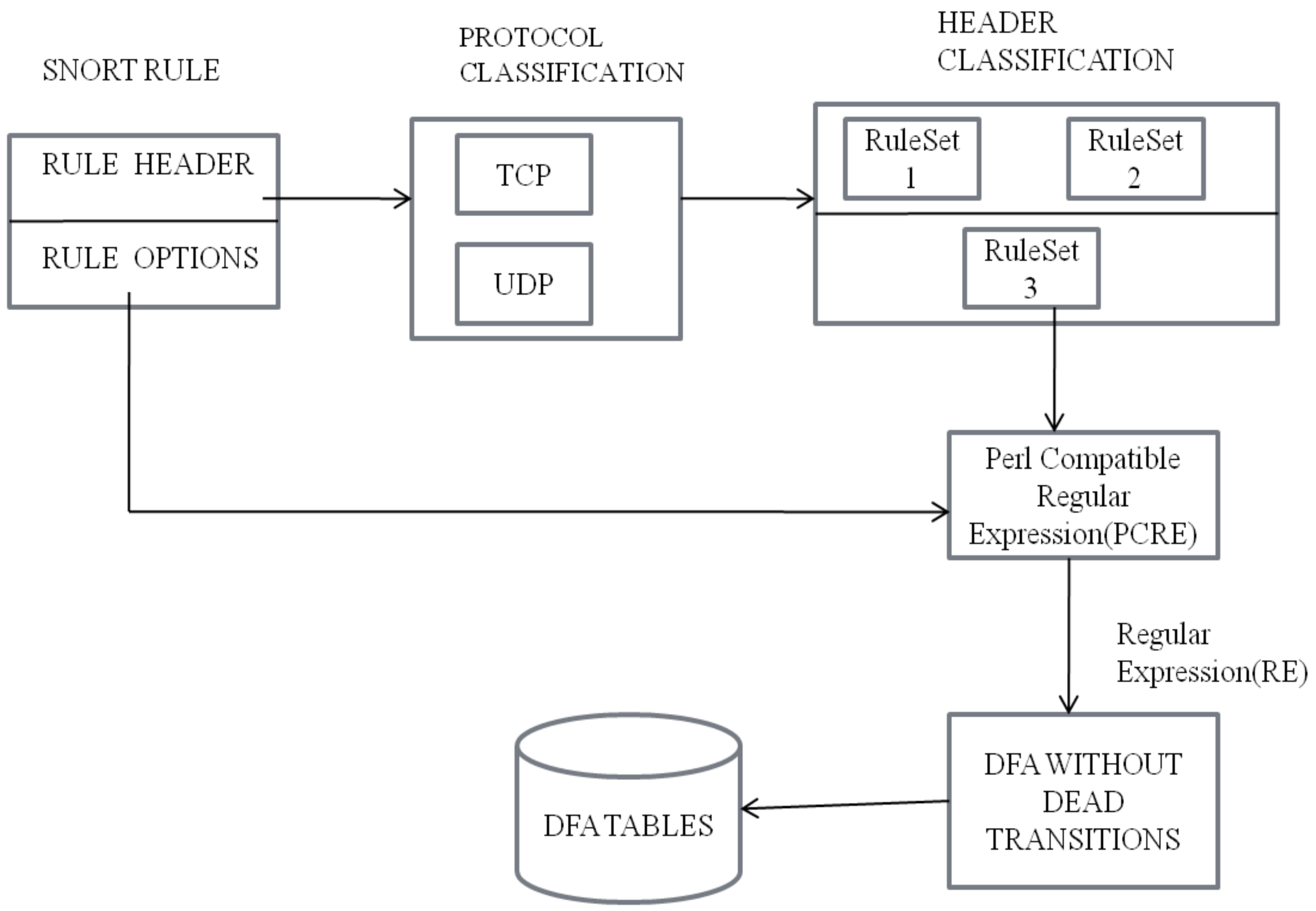

Figure 3.11 SNORT Pre-processing. RS- Rule Set; RE-Regular Expression

The snort rule looks like this.

Alert tcp $HOME_NET any -> $EXTERNAL_NET any(msg : “” ,content :””,PCRE:””)

Figure 3.12 shows a typical SNORT rule structure. PCRE (Perl based regular expressions) are processed for the construction of DFA transition tables. PCRE based regular expressions are converted using Regexbuddy (Tool) and DFA is constructed from the regular expression with dead transitions removed.

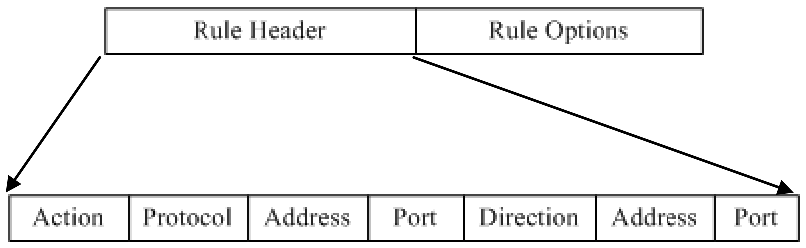

Figure 3.12 SNORT Rule Structure

Detection Phase

Figure 3.13 shows the MDS model. The incoming packets are classified based on the protocol and the header fields and the rule-set to which the packet belongs is identified .The packet is then compared against all the DFA's pertaining to that rule-set. Each packet is processed against a set of DFA's in parallel and each byte within the packet is inspected in parallel to record multiple patterns occurring at different byte positions within the packet.

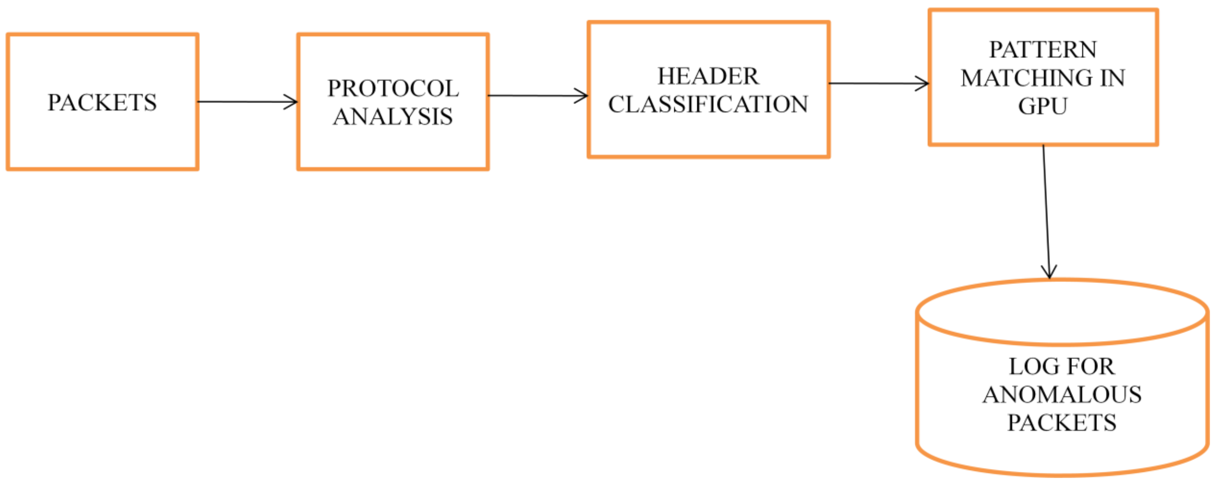

Figure 3.13 Misuse Detection System

Logging Anomalous data

In case of a match found in the detection phase the packet number along with the content of the packet, the rule matched against it and the message contained within the rule in rule options, and the start byte of the signature is recorded in a file.

3.2.4.3 Anomaly Header-based NIDS

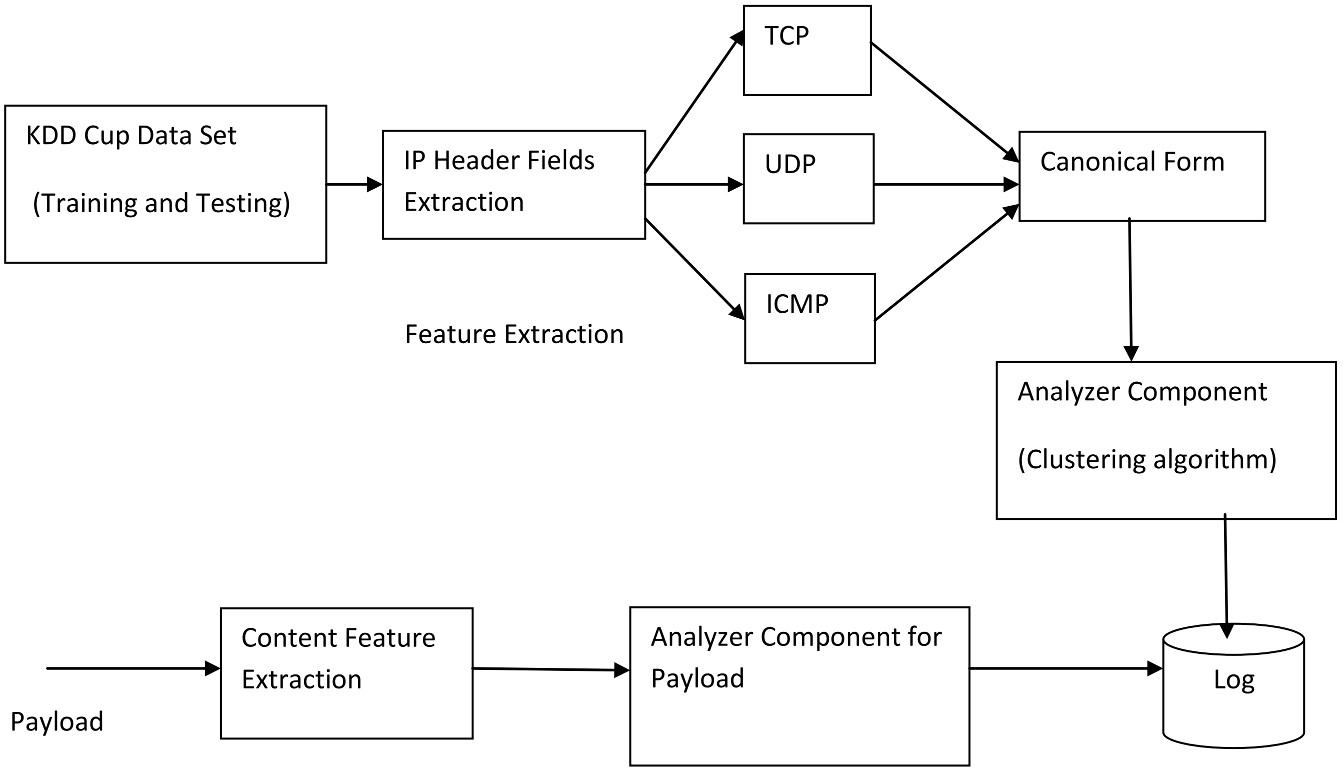

Figure 3.14 Anomaly Detection System

Fig 3.14 shows the steps involved in anomaly detection. The header fields of packet are processed to obtain connection-oriented features. Here, we choose 41 features as described by KDD cup dataset. These features are converted to canonical form according to the clustering algorithm used in the system.

In this work, we use unsupervised clustering algorithm, ART-1 for clustering data as it is found to have high detection rate and low false positive rate. Our objective here is to improve the training speed of such sequential algorithm to form a real-time detection system. Detailed description of the algorithm and the methodology used in parallelizing is provided in the next chapter. The initial preprocessing activity for this analyzer component is done in CPU. The features are normalized and assigned a unique binary value based on interval. The system is trained and tested using KDD cup dataset and the results are logged for future analysis.

3.2.4.4 Anomaly Payload NIDS

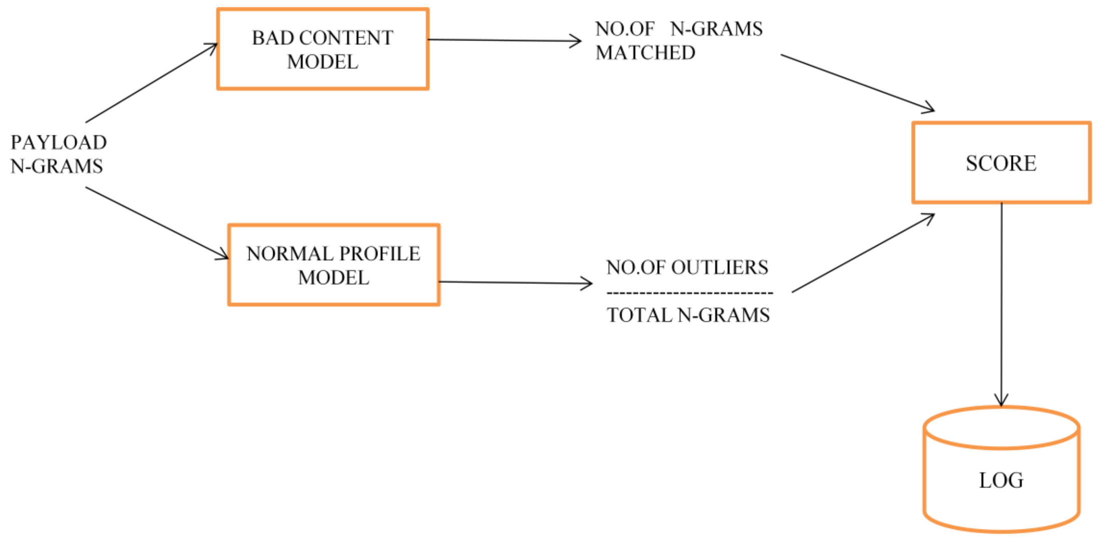

Figure 3.15 Payload Based Detection

The Payload Based Detection System is shown in Fig 3.15 .In our system, N-gram of payload is used in training and detecting attacks. The N-grams of normal traffic data are to be stored in memory and compared with incoming payload. We come up with an SOM based system to remember N-grams. Here, we create separate SOM model for Normal and bad content traffic. In theory, SOM efficiently classifies the data but are slower when compared to other peer neural networks. To overcome this, we utilize the computational capability of CUDA to speed-up the clustering process.

In order to fully exploit the parallel processing ability of GPU, we use modified version of SOM namely Batch Incremental SOM(***BISOM***). BISOM facilitates data parallelism thus enabling detection system to learn N-Grams in parallel. Furthermore, we exploit inherent parallelism within the algorithm. In the detection phase, the Euclidean distance is computed for each n-gram of payload with both models. The score is computed based on number of outlier n-gram/total number of n-gram and number of bad-content n-gram.

The design for our proposed model has been discussed. Next, we will discuss the steps involved in its implementation on CUDA.

CHAPTER 4

IMPLEMENTATION

This chapter describes the implementation aspects of our proposed design and how effectively it can be mapped to CUDA.

4.1 Mapping to CUDA

In this work we have programmed using CUDA-C which is an extension to ANSI C that NVIDIA has created to compile programs containing CUDA kernel code into executables that can use the GPU as a target device. The important considerations to be made for the purpose of mapping are as follows:

1. Thread Organization

2. Memory Organization

Thread Organization

As already discussed in chapter 1, CUDA architecture is organized as Grids, blocks and threads. A Grid is a collection of blocks. A block is a collection of threads. During implementation, a decision has to be made on how these are organized to obtain peak performance. This decision depends on the nature of the algorithm under consideration.

Memory Organization

As discussed in Chapter 1, CUDA memory architecture has different types of memory like constant, global, shared etc. During implementation, suitable memory has to be chosen for storing the data, by considering the pros and cons of each memory. The choice depends on the size and type data and also the algorithm that processes it.

Execution of a CUDA program involves launching kernels. A kernel is a piece of code which is callable from host and executed on device. The GPU kernel execution takes four steps:

i. DMA transfer of data from host(CPU) to device(GPU)
ii. Host instructs GPU to launch kernel
iii. GPU executes threads in parallel. Block scheduling is done by GPU
iv. DMA Transfer of results from device to host

Figure 4.1 shows how the CUDA program life cycle takes place

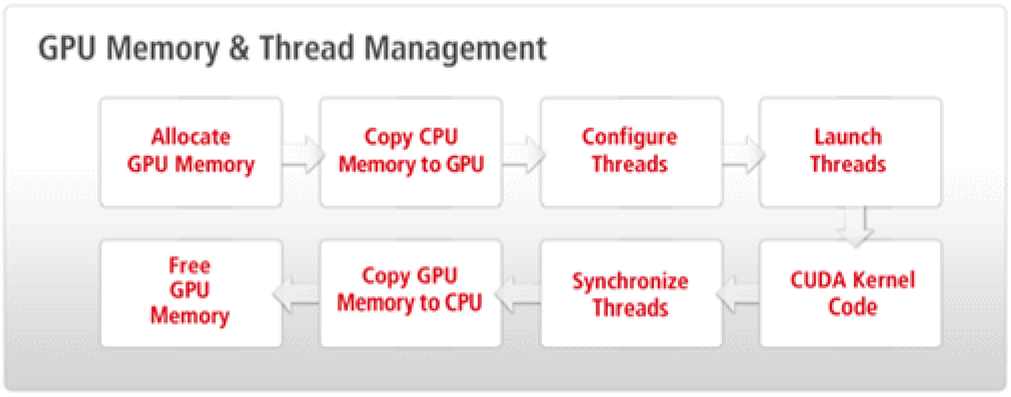

Figure 4.1 CUDA Program Cycle

The next section describes how we have mapped our work to CUDA. We have indicated how parallelization in done in each module and have provided the reasons for the same. We have also provided suitable diagrams for better understanding.

The project flow is shown in Fig 4.2. Initially packets are buffered in CPU memory and then transferred in batch during the kernel launch. The kernel executes the operation and transfers the results to CPU.

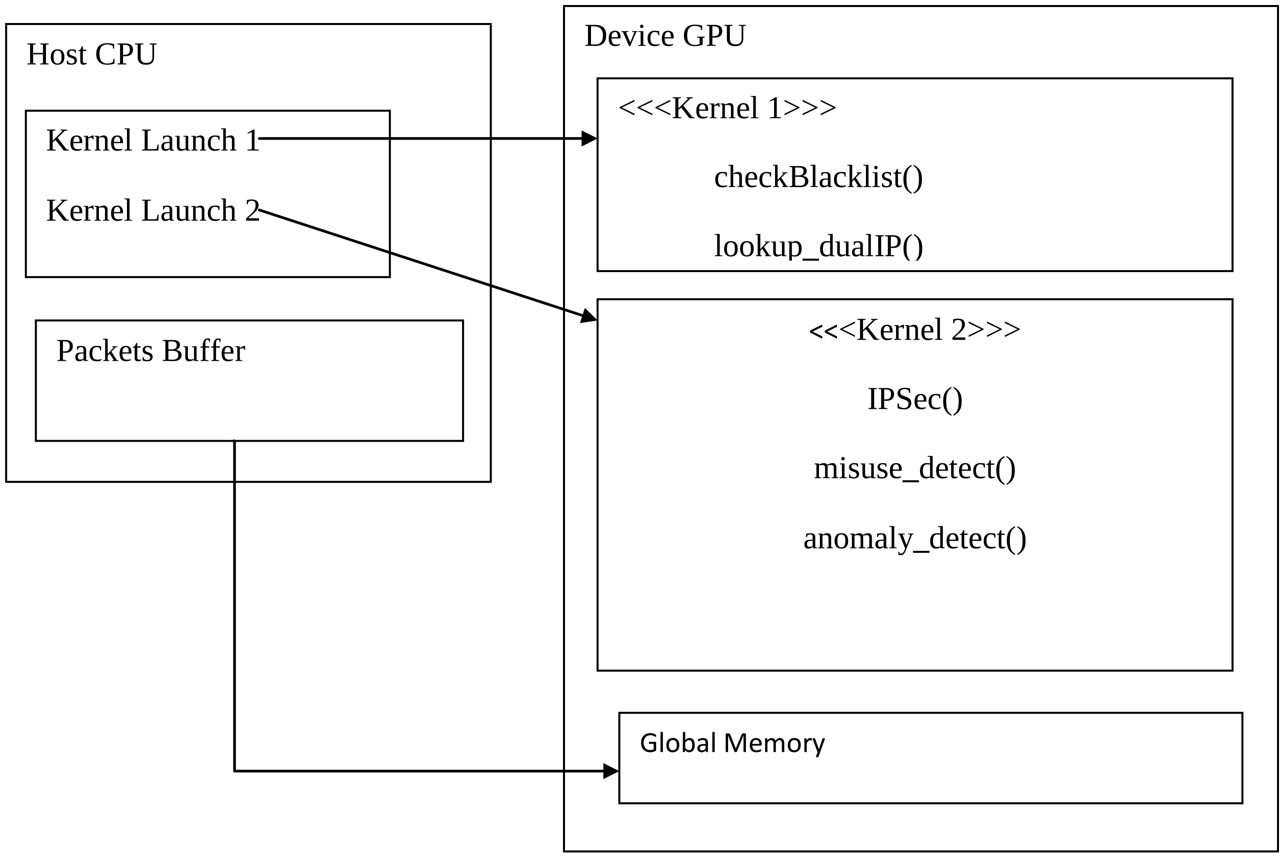

Figure 4.2 Kernel Launch In GPU

4.2 Parallelization in Routing

In routing module parallelization using CUDA is done in checking for blacklisted address and in lookup with Dual-IP. In check-Blacklist function, the packet’s IP address is compared with the blacklist entries and decision of whether the packet should be routed or dropped is taken. Here, two levels of parallelism are employed - data level and task level by launching 2-D threads (x,y). The y-threads are chosen to handle packets and x-threads to handle comparison with blacklisted entries. This logic is based on the concept of warp (collection of threads executed in parallel) execution. The threads are arranged in column-major order. By allowing y-threads to process a packet, we claim that complete processing of single packet is done with a warp execution.

Coming to memory optimizations, the blacklisted entries are placed in constant memory rather than global for faster access rate. Coalescing memory access is obtained as the threads access consecutive memory locations for packet data.

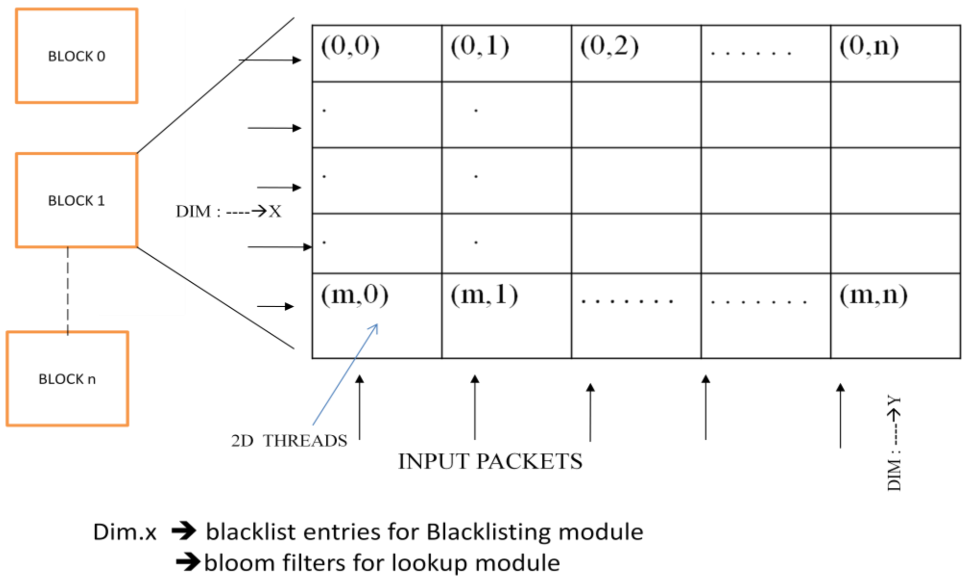

Figure 4.3 Block Launch For Kernel

Figure 4.3 shows how parallelization in done within a block for routing. We first group the prefixes based on their length. Bloom filters are developed for the unique prefix lengths and the corresponding prefix entries are hashed and stored. Now, the destination IP address of the packets is hashed to determine which bloom filters (prefix length) they belong to. We then iterate the corresponding prefix entries to determine next hop. For outgoing packets we check if the next hop protocol is v4 or v6. If it is V4 we send the appropriate v4 packet, else we route as such.

Here parallelism is achieved in the same manner as the blacklist function. Instead of blacklist entries in the x-dimension, we use bloom filter. The match vector, which gives the longest prefix length, is placed in shared memory because it is updated by all the threads. Hash tables that contain next hop entries are placed in constant memory due to their fast access rate.

4.3 Parallelization in NIDS systems

In NIDS, parallelization using CUDA is done in IPSec, Misuse Detection System, in anomaly header detection and anomaly payload Detection System. Computationally intensive tasks like cryptographic algorithm, pattern matching and clustering algorithms used in these domains are off loaded to GPU.

4.3.1 IPSec

IPSec aims at providing security and privacy by encrypting packets. It employs several cryptographic algorithms for this purpose. In our study, we need to perform partial decryption of 2-key IPSec encrypted packets to allow deep packet inspection by NIDS system. We restrict ourselves to the implementation of AES-CBC and DES-CBC decryption algorithms.

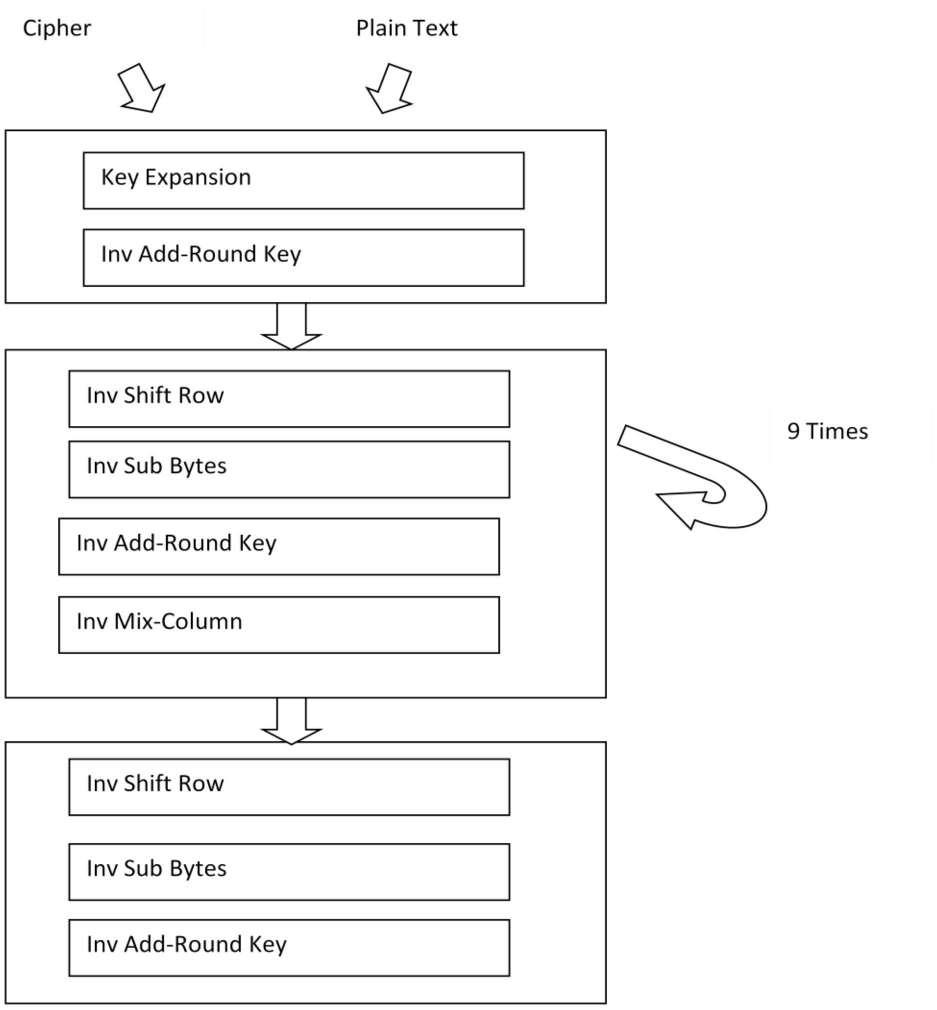

Figure 4.4 AES-CBC

The steps involved in AES-CBC decryption for 128-bit key is shown in Figure 4.4. The algorithm has 10 rounds of processing of cipher text. Initially 128-bit key is expanded in Key Expansion phase to generate keys for 10 rounds. The cipher text is subjected to following operations for 9 rounds: Inverse Shift Row, Inverse Substitution Bytes, Add Round Key, Inverse Mix-Column. The last round performs all the operations except Inverse Mix-Column and the final plain-text is obtained.

DES-CBC:

The deciphering algorithm for DES is detailed below.

```
De-cipher(cipherBlock[64],RoundKeys[16,48],plainBlock[64])
{
  Permute(64,64,cipherBlock,inBlock,InitialPermutationTable);
  Split(64,32,inBlock,leftBlock,rightBlock);
  For(round =16 to 1)
  {
     Mixer(leftBlock,rightBlock,RoundKeys[round]);
     If(round != 1)
        Swapper(leftBlock,rightBlock);
  }
  Combine(32,64,leftBlock,rightBlock,outBlock);
  Permute(64,64,outBlock,FinalPermutationTable);
}
```

Parallelization in GPU

Here, the deciphering algorithms AES and DES are executed in parallel. This is important in the context of IPSec, as different packets are encrypted with different algorithms. For this, we assess the scope of parallelism in each algorithm and arrive at a model that gives peak performance for both when combined.

As shown in Figure 4.5, the proposed model handles packets in parallel, with each y-thread handling a packet and its corresponding decryption algorithm. The x-threads handles byte blocks within a packet in parallel, thus providing algorithm-level parallelization. The number of bits handled by an x-thread depends on the algorithm: 128 bits for AES and 64 bits for DES. Coalescing memory access is obtained as the threads access consecutive memory locations for packet data.

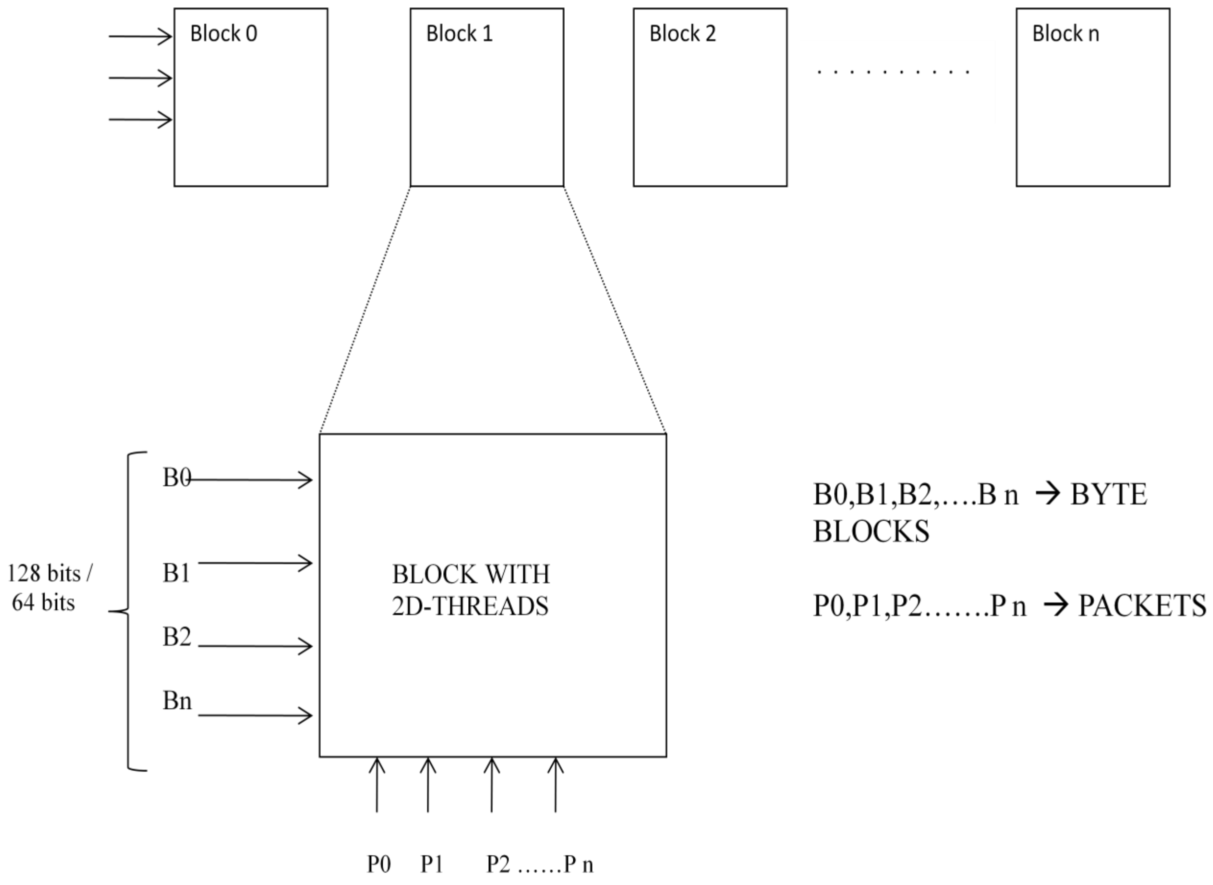

Figure 4.5 View of a Block for IPSec

4.3.2 Misuse Detection system

In misuse detection system the packet payload contents are inspected against known signature of attacks. The preprocessing phase involves DFA construction from Rule Set of Snort, as discussed in previous chapter. These DFAs are transferred to constant memory of GPU. The storage of DFAs in constant memory is adopted as DFAs are static and need to be accessed frequently by each byte of payload. The procedure for the traversal of DFA is given below.

Pseudo code

```
For each byte in packet
    Ss=startState
    For each transition in ss ( i from 1 to max_trans)
        If(pckt_data[in]==dfa[ss][i][0])
          Ss=dfa[ss][i][1]
         In++;
         If(ss==accept_state)
             Pattern Found
             Break
        End if
      End if
   End for
End for
```

In order to uncover all the attack signatures in packet, we intend to traverse every byte of packet against DFA rather than identifying only the first matching signature. For this, each byte begins with the start state of DFA and is traversed with its subsequent bytes as transition symbols. The byte sequences that lead to final state of DFA indicates that the attack signature has been matched. The matched byte sequences and attack details are recorded for future analysis.

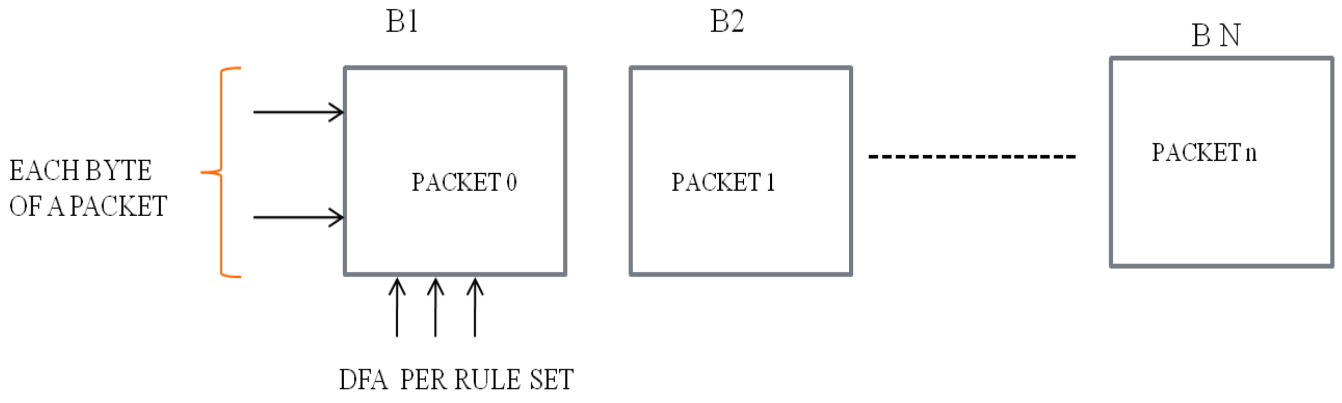

Figure 4.6 View of Block for MDS

Parallelization in GPU

We can see from the procedure that DFAs are to be traversed by each byte of the packet which amounts to huge quantum of DFA traversal. To overcome the overhead associated with this process, we come up with the idea of parallel DFA execution with parallel byte matching. Mapping to CUDA architecture, each packet is handled by a block which gives data-level parallelism as shown in Figure 4.6. Threads are launched in 2-D blocks with x-threads deciding the byte to be processed and y-threads identifying the DFA to be acted upon.

4.3.3 Anomaly Header-based Detection System

The implementation of ART-1 for analyzer component of AHBDS is detailed here. The preprocessing of data for this component is done in CPU as described earlier. The ART-1 network diagram is shown in the Figure 4.7. The F1 layer is the input layer which accepts binary input. The F2 layer constitutes the clusters. The nodes in these layers are connected by top-down and bottom-up weights. For each input, the activation component of each cluster is computed by multiplying bottom-up weights with input signal. The winning cluster is found and the weights are updated.

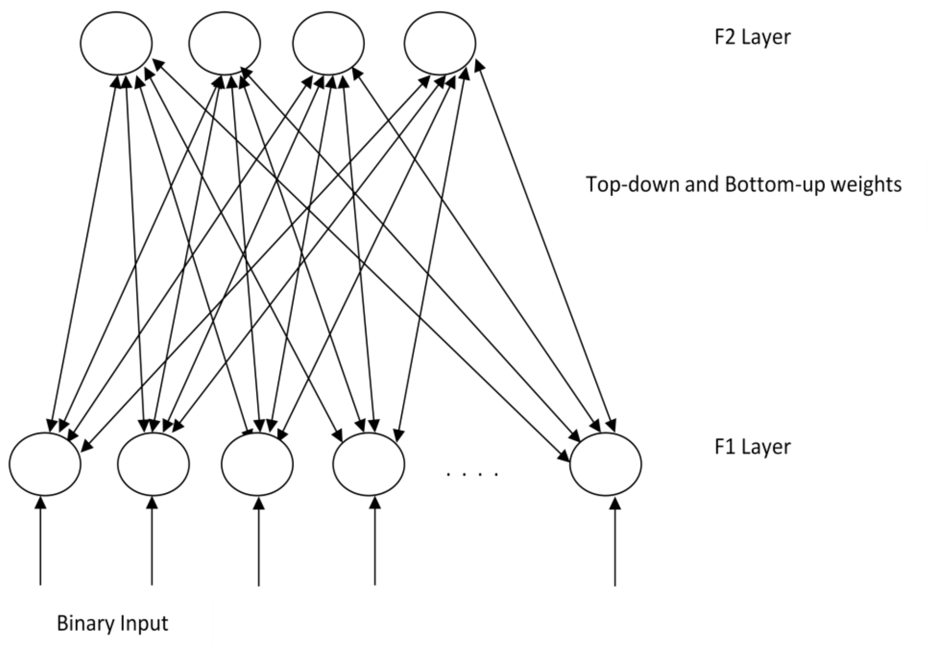

Figure 4.7 ART-1

The steps involved in ART-1 are given below.

ART-1 clustering Algorithm:

N: =Maximum number of clusters

M: =Input Vector Dimension

S: =Input Vector

Y: = Activation component

```
P: =vigilance parameter

STEP 0: Initialize Top-down and Bottom-up weights

STEP 1: while stopping condition is false, do STEP 2.

    STEP 2: For each training input  do STEP 3- 8

        STEP 3: Set activation of all F2 units to zero

        STEP 4: Set activations of F1 to input vector S

        STEP 5: Send signal from F1 to F2  x(i)=S(i)

        STEP 6: Calculate Activation Component

            y(i,j): = ∑ b(i,j) *x(i)                  }

        STEP 7: For node with maximum y, test for Reset

            Compute x(i): =s(i)*t(i,j)
            If Norm(x)/Norm(s) < P
                Inhibit y
            Go to previous step
            Else
                Winner=y

        STEP 8: Update top-down and bottom-up weights of winning cluster

            b(i,j) : = ( L * x(i) )/( L – 1 + norm(x) )      }
            t(i,j): = x(i)                                   }
```

ART-1 using CUDA

In anomaly detection system the F1 nodes tend to increase since there are many connection features are to be considered. Hence the complexity is laid in calculating activation component of all F2 nodes and updating weights of winner. Here, we tried to reduce the running time by parallelizing possible tasks in algorithm. The braces in algorithm indicate the steps that are performed in parallel. Data parallelism is not applicable in this case since there are weight updations after each input.

Here a single block is launched with (N,M) 2-D threads. N*M multiplication and summation in algorithm is done in parallel. To prevent thread starvation we have used a scale factor to control number of threads. Based on scale factor each thread performs series of multiplication and local summation. The bottom-up and top-down weights of all clusters is kept in consequent memory locations. This will enhance performance through coalesce access of weights in global memory. The results of implementation are presented in next chapter.

4.3.4Anomaly Payload-Based Detection System (APBDS)

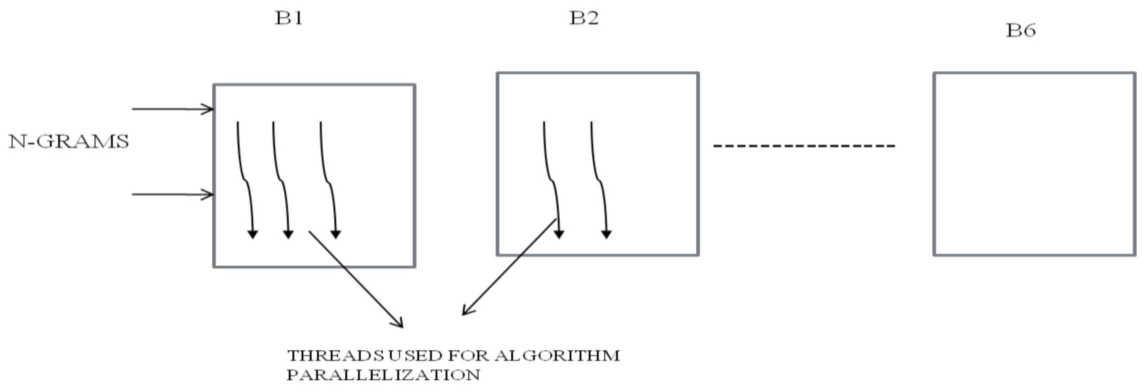

Figure 4.8 Block view of APBDS system

APBDS design was introduced in the previous chapter. This chapter focuses on BISOM implementation in CUDA. The algorithm for Batch-Incremental SOM is explained as follows.

```
STEP 1: Initialize weight vectors
STEP 2: For each epoch
        STEP 3: Interpolate new values of sigma
        STEP 4: Initialize numerator and denominator to zero
        STEP 5: For each N-Gram
                For each cluster                  }
                    Compute distance              }
                Find winning node                 }

                For each cluster                                      }
                    Accumulate numerator and denominator              }
        STEP 6: For each cluster                                      }
                Update weight vectors                                 }
```

The scope of parallelism within algorithm is indicated with braces. BISOM is similar to ART except that the weight updations are done at end of epoch. In our implementation, one dimension of thread is used in handling N-grams (data parallelism) and other is used for parallelizing computations of each step as shown in Figure 4.8. Further, N-grams of all payloads are processed without any distinction on packet to which it belongs, thus employing load-balancing technique. The resulting malicious and outlier N-gram details are sent as result to CPU. The log file is updated by both detection systems for further analysis by network administrator.

Thus the router has been implemented by effectively mapping the tasks on to the GPU. The evaluation of the effectiveness of this approach is discussed in the next chapter.

CHAPTER 5

PERFORMANCE EVALUATION

In this chapter we present the results of our work. All the modules have been implemented on an Nvidia GT220 graphics card. Extensive experiments have been carried out to identify the right amount of parallelism and right mapping on to the GPU. For this purpose, we have varied the number of threads and blocks and have evaluated the performance. By varying these parameters, we identify the best combination for optimal performance of these operations.

Table 5.1 shows the time taken for varying block-thread combinations for the lookup module for different packet counts.

Table 5.1 Lookup Performance with Blocks and Threads Variation

2D-Threads	**Blocks**	**Time in ms for different packet count**		
		1000	**5000**	**10,000**
150	6	0.224	0.676	1.294
	12	0.172	0.522	0.942
	18	0.183	0.460	0.884
225	6	0.216	0.594	1.04
	12	0.181	0.461	0.883

	18	0.154	0.433	0.841
300	6	0.171	0.511	0.930
	12	0.154	0.454	0.826
	18	0.159	0.423	0.812
450	6	0.144	0.464	0.887
	12	0.123	0.526	0.852
	18	0.160	0.464	0.806
510	6	0.165	0.543	0.832
	12	0.154	0.425	0.813
	18	0.159	0.436	0.774
525	6	1.22	5.687	5.842
	12	0.927	5.83	7.600
	18	0.956	3.577	6.533

For better comparison of speed up due to block-thread variation, the speed-up values for 1000 packets are given in Table 5.2 by treating the time for 150-6 block-thread combination as the base model. These values are plotted in Figure 5.1. From the graph it can be inferred that the peak performance is achieved for 450 threads and 12 blocks.

Table 5.2 Speed-Up Factor for 1000 Packets

Threads	Blocks	Speed-Up Factor
150	6	1
	12	1.302
	18	1.224
225	6	1.037
	12	1.037
	18	1.237
300	6	1.454
	12	1.31
	18	1.454
450	6	1.409
	12	1.556
	18	1.821
510	6	1.4
	12	1.358
	18	1.409
525	6	0.184
	12	0.242
	18	0.234

The effect of block-thread variation is shown in Figure5.1, 5.2 and 5.3 for the lookup, MDS and IPSec respectively. It can be inferred that the variation is associated with the resource utilization of GPU.

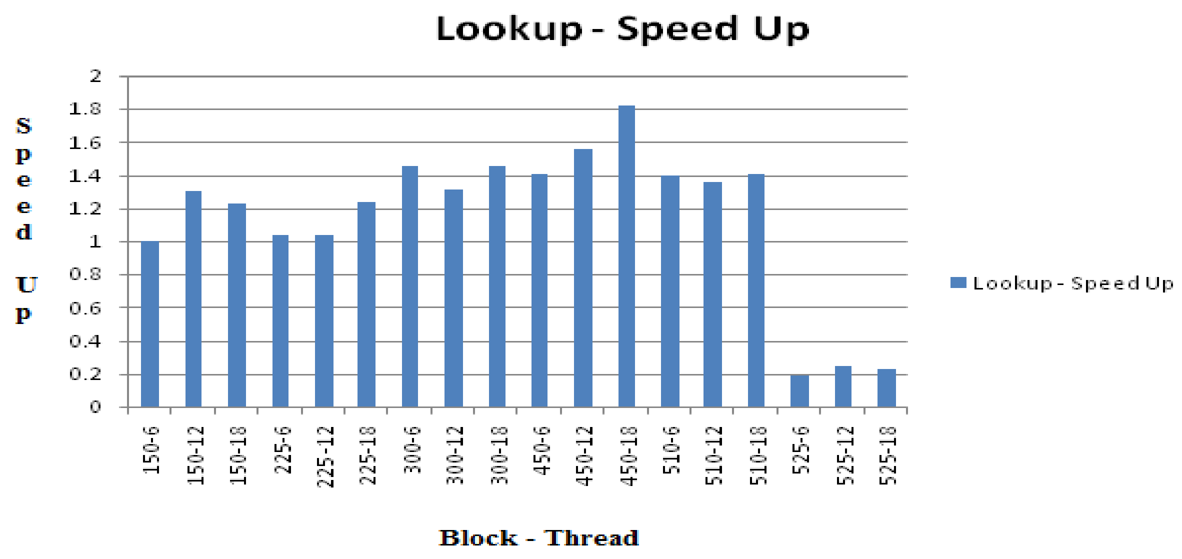

Figure 5.1 Speed up Factor in Lookup for 1000 packets

A close look indicates that optimal execution time is achieved at a point where resources are utilized effectively. The peak performance is achieved for 450 threads and 12 blocks (1000 packets) in case of lookup and for 100 blocks and 256 threads in case of MDS. For IPSec, the peak value is achieved for 6 blocks and 256 threads.

The difference in optimal number of blocks and threads in each of the case depends on the number of blocks executed in a SM. The number of blocks executed in parallel in a SM depends on resource usage by the algorithms.

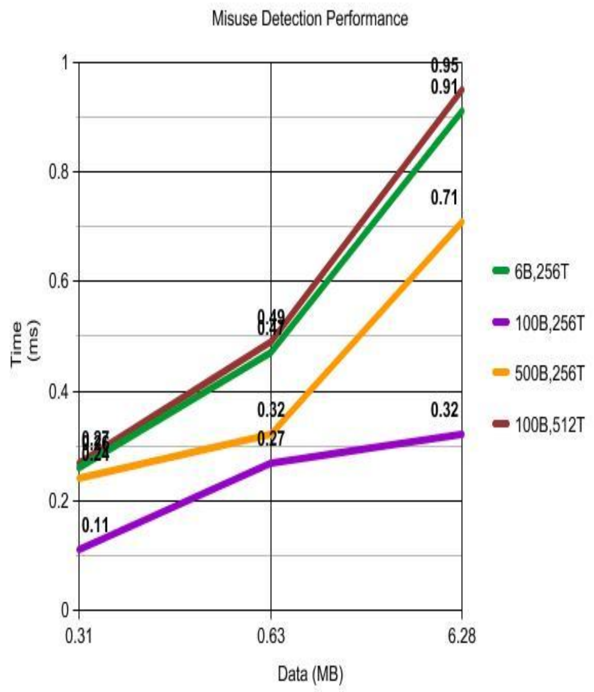

Figure 5.2 MDS Performance

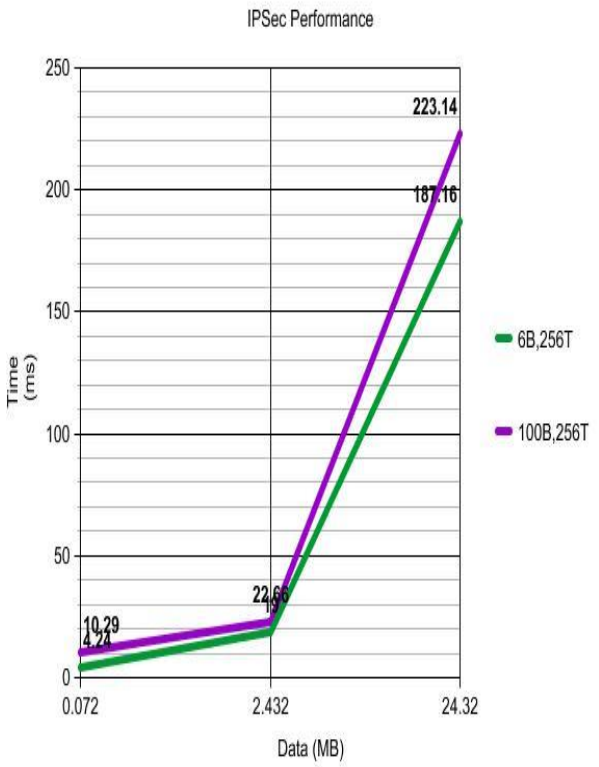

Figure 5.3 IPSec Performance

Table 5.3 Results of Anomaly Header-Based System

Input Dimension	Per Thread Processing Data	Time in ms
100	1	94.55
	4	2.24
	8	2.18
	16	2.52
128	1	119.04
	8	2.88
	16	1.78
	32	2.58

Table 5.3 shows the results of anomaly header-based system. The input dimensions are the number of nodes in the F1 layer. The maximum number of clusters is 128. Each cluster is processed in parallel and within each cluster the calculations for activation component is computed in parallel. Per- thread processing data indicates the number of computations done by each thread. The time decreases when the data is distributed among threads. The increase at certain value indicates per-thread over head in data processing.

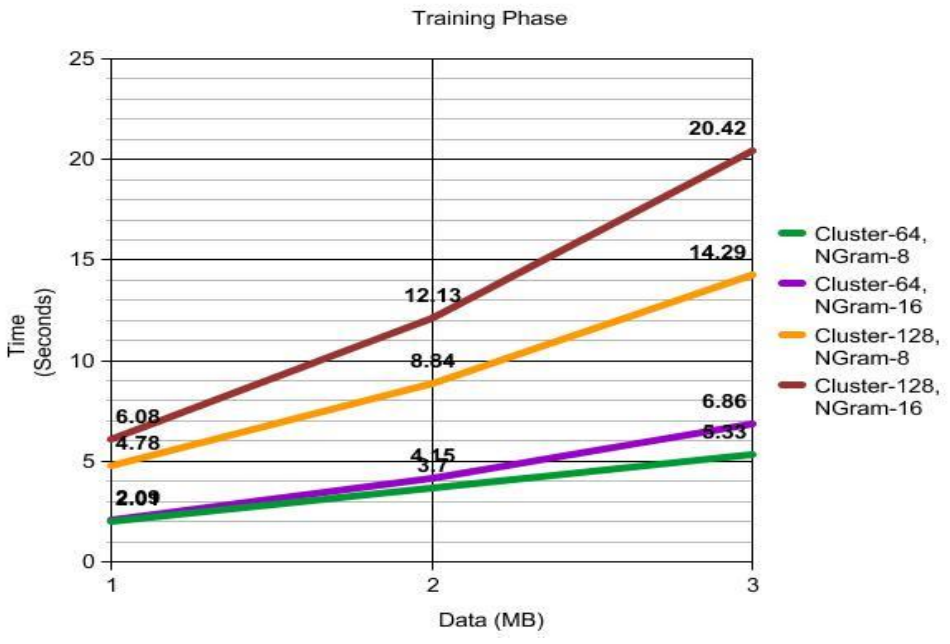

Figure 5.4 APNIDS Training Phase

It can be inferred from Figure 5.4 and 5.5 that the execution time of training and detection phase of APNIDS depends on the number of clusters and length of the N-Gram. In both cases the execution time increases with the number of clusters. In case of training the increase in length of N-Gram increases the time. This is due to the fact that the length of N-Gram determines the number of nodes in F1 layer which in turn influences number of weight updations required. With detection, the increase in length of N-Gram reduces the number of N-Grams to be detected. Also, there is no weight updation in this phase. This improves the detection time.

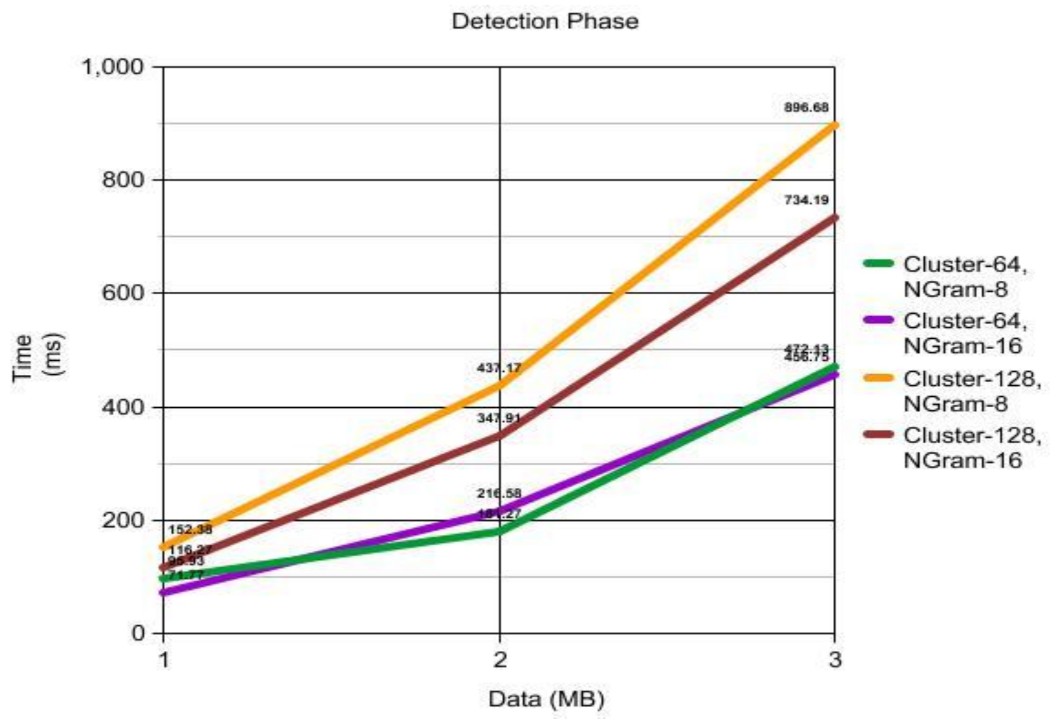

Figure 5.5 APNIDS Detection Phase

In order to the show the power of GPU for network processing tasks, we also provide a comparison with AMD ATHLON processor for the lookup task as shown in Table 5.4.

Table 5.4 : Comparison for Lookup – ATHLON vs. GPU

Hardware Type	**Technique Used**	**Data Entry**	**Time (in sec)**
AMD ATHLON 1.1 GHZ MACHINE: 512MDDR33 memory	Divide and Conquer technique	1 MILLION DATA PACKETS	0.649
GT220 CUDA cores : 48 Compute capability – 1.2	Bloom-Filters technique		0.000733

The prototype software implementation of AMD ATHLON [16] is single process, single thread running on a machine with single CPU. Thus, the algorithm executes sequentially, which contributes to the long lookup time. But, GPU can run several blocks and several threads within block in parallel. So, our execution time is less.

Though AMD ATHLON uses divide and conquer technique, its complexity comes from storage of large number of segment tables and linear lookup in those tables. The GPU also has memory constraints. So, a space efficient technique, bloom filters is used. Also, the latency is reduced by utilizing constant, texture and shared memory of the GPU. So, our performance is 930 times better.

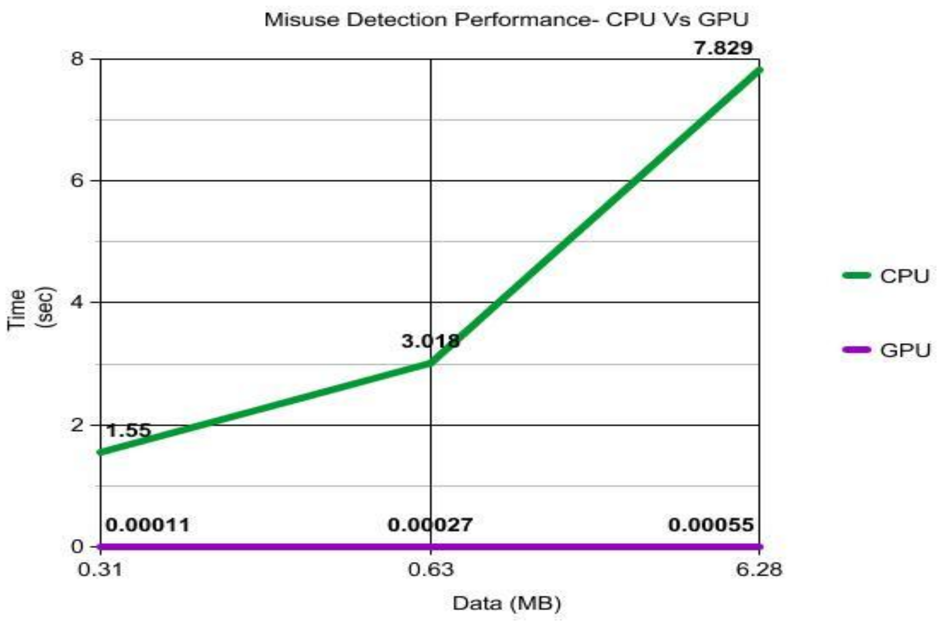

Figure 5.6 Comparison of MDS – CPU vs. GPU

Figure 5.6 indicates the comparison of CPU and GPU for Misuse detection system. It can be seen that GPU outshines the performance of CPU by a factor of 12,000.The performance data indicates that GPU is effective in leveraging the speed of various network operations.

CHAPTER 6

CONCLUSION AND FUTURE WORK

We have shown is this thesis that GPU can be used for speeding up networking operations. We have implemented routing and NIDS functionality on an edge router. We have off-loaded the computationally intensive tasks like longest prefix matching, pattern matching, cryptographic algorithms and neural network algorithms used in the domain to GPU. We have applied suitable parallelization at each stage and have shown the results. Our analysis of the results shows that GPU’s parallel implementation scales at a higher rate when compared to other processors. The results also depicts that GPU can be used for better designing of future routers.

Our NIDS system on GPU inspects ingress traffic only. But the co-relation between the ingress and egress traffic can improve detection. Our work was done using a single GPU. Future work can extend this to multiple GPUs. The issues with current GPUs are being addressed by new and upcoming architectures like Nvidia’s Fermi. More focus can be given for deploying network operations on these latest GPUs.

REFERENCES

[1] Amini. M, Jalili. R, and Shahriari .H.R, “RT-UNNID: A Practical Solution to Real-Time Network-Based Intrusion Detection Using Unsupervised Neural Networks,” Elsevier Computers and Security Journal, Vol. 25, no. 6, September 2006.

[2] Amini. M, Jalili .R, “Network-based intrusion detection using unsupervised adaptive resonance theory (ART),” Proc. Fourth Engineering of Intelligent Systems Conf., Madeira, Portugal, 2004.

[3] Bolzoni .D, Etalle .S, Hartel .P, “POSEIDON: a 2-tier Anomaly-based Network Intrusion Detection System,” Fourth IEEE International Workshop on In Information Assurance, 2006, pp. 144-156.

[4] Cheng Zhong, Guo-Liang Chen, "A Fast Determinate String Matching Algorithm for the Network Intrusion Detection Systems," Machine Learning and Cybernetics International Conf. ,2007, vol. 6, pp. 3173-3177.

[5] Dharmapurikar. S, Krishnamurthy. P, Taylor D.E, "Longest prefix matching using bloom filters," Networking, IEEE/ACM Transactions on , vol.14, no.2, pp. 397- 409, April 2006.

[6] Giorgos Vasiliadis, Spiros Antonatos, Michalis Polychronakis, Evangelos P.Markatos, and Sotiris Ioannidis, “Gnort: High Performance Network Intrusion Detection Using Graphics Processors”, Recent Advances in Intrusion Detection,vol. 5230, Springer, 2008, pp. 116-134.

[7] James Kelly, “An Examination of Pattern Matching Algorithms for Intrusion Detection Systems”, Ottawa-Carleton Institute for Computer Science, Carleton University, August 2006.

[8] Ke Wang, Stolfo. S.J, “Anomalous Payload-based Network Intrusion Detection,”Recent Advances in Intrusion Detection-RAID Conf., 2004, pp. 203-222.

[9] Lampson. B, Srinivasan. V, Varghese. G, "IP lookups using multiway and multicolumn search," Networking, IEEE/ACM Transactions on , vol.7, no.3, pp.324-334, June 1999.

[10] McLain .C, Studer .A, and Lippmann .R, "Making network intrusion detection work with IPsec," Massachusetts Institute of Technology Lincoln Laboratory, Tech. Rep. 1121, 2007.

[11] Morrison. D.R, "PATRICIA--Practical Algorithm to Retrieve Information Coded in Alphanumeric," J. ACM, vol. 15, no.4, Oct.1968.

[12] Nilsson. S and Karlsson. G, "IP-Address Lookup Using LC-Tries," IEEE J. SAC, vol. 17, no. 6, June 1999, pp. 1083–1092.

[13] Silva .B, Marques N.C, “A Hybrid Parallel SOM Algorithm for Large Maps in Data-Mining,” Proc. 13th Portuguese Conference on Artificial Intelligence (EPIA 2007), Workshop on Business Intelligence, Portugal, 2007.

[14] Srinivasan. V and Varghese. G, "Fast Address Lookups Using Controlled Prefix Expansion," ACM Transactions on Computer Systems, Vol. 17, no. 1, February 1999.

[15] Waldvogel. M, "Fast longest prefix matching: Algorithms, analysis and applications," Ph. D. Thesis, Swiss Federal Institute of Technology,Zurich, 2000.

[16] Zhenqiang Li, Xiaohong Deng, Hongxiao Ma, Yan Ma, “Divide-and-Conquer: A Scheme for IPv6 Address Longest Prefix Matching,” IEEE Workshop on High Performance Switching and Routing - HPSR , 2006.