Master Dissertation

Performance Analysis of Different Hash Functions using Bloom Filter for Network Intrusion Detection Systems in 32-bit and 64-bit Computer Operation Mode.

By

TAN BENG GHEE

A Dissertation submitted for partial fulfilment of the requirement for the degree of Master of Science (Microelectronic Engineering)

August 2016

ACKNOWLEDGEMENT

First of all, I would like to thank my supervisor, Dr. Nor Muzlifah Mahyuddin for the patience and courage on guiding and providing constructive comment and knowledge throughout this time.

Secondly, thanks for my colleagues that covered me on my works while I was away to finish up this dissertation within the time frame.

Lastly, a very thankful appreciation to my wife, Soo Kheng for all the encouragement and the supports given.

ABSTRACT

A Network Intrusion Detection System (NIDS) is an application or device that screens the network traffics for malicious activities or any violation in the network policy. In current Gigabit per second (Gbps) networking speed, the NIDS needs to be very fast and efficient. Bloom Filter is a key component within the NIDS that contribute to the speed of the system. A Bloom Filter is an array of bits that determine whether a given structure of information belongs to it. A Bloom Filter pattern matching algorithm with fast hashing functions is developed for 32-bit and 64-bit computer system. The implemented hashes are Murmur2 Hash, City Hash, One-at-a-time Hash, SuperFast Hash, and Lookup3 Hash. The developed system's functionality is verified. Performance evaluation data shows that the Bloom Filter with SuperFast Hash is the fastest among all the Bloom Filter Variants that were under test. Experiment result also indicates that the Bloom Filter executes faster in 64-bit Mode as compared to 32-bit Mode, regardless of the hash. All the Bloom Filter Variants meet the projected false positive rate (0.1%) that were initialized. The Bloom Filter with City Hash recorded lowest false positive rate among all the Bloom Filter Variants.

ABSTRAK

Sistem Pengesanan Pencerobohan Rangkaian (NIDS) adalah sebuah aplikasi atau peranti yang memantau trafik rangkaian untuk mengesan aktiviti berbahaya atau sebarang pelanggaran dasar rangkaian. Dengan kelajuan rangkaian yang mencapai Gigabit sesaat (Gbps), NIDS perlu bertindak dengan cepat dan cekap. Penapisan Bloom adalah komponen utama dalam NIDS yang menyumbang kepada kelajuan sistem. Penapisan Bloom adalah struktur bit yang mampu menentukan sama ada data tertentu tergolong kepadanya. Algoritma Penapisan Bloom dengan menggunakan fungsi hash yang cepat dibangunkan untuk sistem komputer 32-bit dan 64-bit. Fungsi hash yang digunakan adalah Murmur2 Hash, City Hash, One-at-a-time Hash, SuperFast Hash dan Lookup3 Hash. Sistem yang dibangunkan telah disahkan berfungsi dengan tepat. Data penilaian prestasi menunjukkan bahawa Penapisan Bloom dengan SuperFast Hash adalah yang paling cepat di antara pelbagai Penapisan Bloom yang di bawah ujian. Keputusan eksperimen juga menunjukkan bahawa semua perlaksanaan Penapisan Bloom adalah lebih cepat dalam mod 64-bit berbanding dengan mod 32-bit. Semua Penapisan Bloom memenuhi kadar positif palsu (0.1%) yang diunjurkan. Penapisan Bloom dengan City Hash mencatatkan kadar positif palsu yang terendah di kalangan semua Penapisan Bloom.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

1. INTRODUCTION

1.1. Project Background

Internet is essential in global connectivity. This world-wide computer network connects billions of devices across the globe. Every connected device is assigned a unique Internet Protocol (IP) address to enable the communication within the Internet.

There are plenty of network security systems, one of them is called Network Intrusion Detection System (NIDS). It is built to detect malicious packets by monitoring the incoming and outgoing network packets [1]. Current computer network speed is measured at Gigabit per second (Gbps) due to rapid advancement of networking technologies. NIDS must be able to handle the high speed network traffic and execute complicated packet processing and filtering. NIDS design is becoming very challenging.

Bloom Filter pattern matching algorithm is very commonly used in NIDS, and is chosen to be the main focus in this project. The purpose of this research is to design and develop a Bloom Filter pattern matching algorithm using five fast hashing algorithms that works in the 32-bit and 64-bit computer system. The 64-bit computer system is very common nowadays because of the tremendous improvement in processor architecture for the past 10 years. A 64-bit computer system is capable of processing more data per CPU clock cycles as compare to a legacy 32-bit computer system. The hashing algorithm is also a key element that contribute to the Bloom Filter performance.

The Computer Processing Unit (CPU) hardware used in this project is Intel Microprocessor Core-i7. This model is designed and coded in Microsoft Visual Studio 2012 with multiple threading support. The false positive rates and the data processing throughput is evaluated for the functionality and performance characterization.

1.2. Problem Statement

Network security is very important to safeguard the data integrity within a network. The NIDS must be improved together with the rapid increment of network speed. A bad NIDS throughput will slow down the network which is very costly.

The hashing algorithm is an essential component in Bloom Filter, which is a pattern matching algorithm that is selected as the main algorithm for NIDS threat detection engine design. In this research, five hashing algorithms are chosen to be implemented, which are Murmur2 Hash, City Hash, One-at-a-time Hash, SuperFast Hash, and Lookup3 Hash.

In 2004, Intel ships updated versions of its Xeon and Pentium 4 processor families supporting the new 64-bit instruction set, and since then the computing environment has been slowly shifting to 64-bit systems. A 64-bit CPU offers higher data processing throughput. Multi-threading in an application can also be achieved through parallel computing in Operating System (OS).

This project is inspired by the necessity to increase the performance and the efficiency of the NIDS pattern matching algorithm used to detect network intrusion. In any typical NIDS system, 75% of the CPU processing time has been occupied by the pattern matching detection engine to detect potential attack [1]. This demonstrates that pattern matching is a computational exhaustive process and will impact the overall system and networking performance.

1.3. Objectives

The focus of this project is to design and develop a signature based Bloom Filter matching algorithm using fast hashing algorithm that works in both the 32-bit and 64-bit computer system. The objectives are:

i. To design, optimize and verify the Bloom Filter algorithm.
ii. To implement fast hashing algorithms in the Bloom Filter that runs on modern multi-threading 32-bit and 64-bit CPU.
iii. To evaluate the functionality, as well as the speed performance and the false positive rates of the system with each hashing implementation.

1.4. Scope of Research

The scope of this project is to design and develop a Bloom Filter that fits in Network Intrusion Detection System using fast hashing algorithms that works in both the 32-bit and 64-bit computer system.

Microsoft Cryptographic Service Provider has been chosen to generate random sample packets on the local host to evaluate the functionality and the performance of the model.

1.5. Thesis Organization

Chapter 2 on this thesis comprises the basic technical concept of this project and the Literature Review. Overview of Network Intrusion Detection System, Hashing and Bloom Filter are discussed. It also includes the discussion on the previous works that are related to the improvement of hashing algorithm and the Bloom Filter based pattern matching algorithm.

Chapter 3 covered the steps in designing the Bloom Filter system, the implementation with a varieties of hash functions, the verification and the evaluation of the developed system. The software and hardware requirement for this work is listed, and the detail verification and performance evaluation flow of the developed system are discussed.

Chapter 4 contains the results and the relevant discussions for this work. The verification result is presented. The performance evaluation of the developed system is collected and analyzed.

Finally, Chapter 5 provides conclusion and the overall result of this work. Suggestions of the future work are presented.

CHAPTER 2

2. LITERATURE REVIEW

2.1. Introduction

In this section, the basic bloom filter and hashes that are commonly adopted in intrusion detection system is discussed; focusing on improvement on the bloom filter and hash performance in modern application. Area that is covered in conducting this research includes the bloom filter structure improvement and hash algorithm improvement, for them to perform optimally in Intel processor, which provides an effective filter to reduce the database query which is known to be very costly.

2.2. Intrusion Detection System

The definition of "Intruder" is any personnel that is trying to gain access to the data and information that resides on a network or system without appropriate authorization by exploiting the vulnerabilities of the system security [2] [3]. This intrusion can seriously impact overall security and functionality of the system.

An intrusion detection system (IDS) is a system that monitors network traffic or system activities for malicious activities or violations of defined policy, and generate electronic reports to a managing host. IDS is first introduced in the early

1980s by James P. Anderson; published a study on the new approaches that improve computer security auditing and surveillance [2] [4] [5]. There is a variety of IDS and they have different approaches in detecting suspicious traffic, and IDS device can be either host based (HIDS) or network based (NIDS). A host-based IDS (HIDS) resides on a client computer or a single workstation, and protects only that specific host whilst a network-based IDS (NIDS) is a separate device that is attached to a network that monitor and analyze the network traffic [3].

NIDS is a network security system that detects attack within the same network from authorized users. It is the software that computerizes the traffic monitoring process and analyze every occurring events within a computer network for any potential incidents which poses threats to the security of the system [6].

Performance of NIDS is generally measured by two fundamentals and most commonly used metrics, which are True Positive (TP) rate and False Positive (FP) rate [7]. TP rate is the probability of the NIDS to trigger an alarm when there is a genuine intrusion. FP rate is the probability of the NIDS to trigger an alarm when no intrusion occurs.

2.2.1. Varieties of Intrusion Detection Systems

Generally, there are two different types of Intrusion Detection Systems. One of them is signature based intrusion detection system. The other one is anomaly based intrusion detection system [3] [8].

Signature based intrusion detection systems search the network traffic for a packet sequence or series of malicious bytes using string or pattern matching algorithm [9]. The advantage of this method is that the signatures are very easy to understand. If the network administrators know the network behavior that needs to be identified, then the IDS is very simple to design and develop. The shortcoming of this technique is that it is only able to detect the intrusion with signatures previously stored in a database, whilst the novel intrusion signatures cannot be identified. A signature must be stored for every intrusion that has been identified.

The anomaly based intrusion detection system works based on defining network behaviors. If the network behaves in accordance with the predefined behavior, then it is accepted as genuine transaction; else the IDS will trigger an alarm signaling an anomaly detection [8]. The classification of an intrusion is based on the rules or heuristics definition, rather than patterns or signatures matching as in signature based IDS, and this system attempts to detect any type of misbehavior that deviates from normal operation. The anomaly based detection is able to detect novel intrusion which signature based systems failed to detect. This system, however, must be well defined and configured for it to be able to recognize the abnormal traffic.

2.2.2. Design and Architecture of Intrusion Detection Systems

Firewall is always the first line of defense against external intruders in any computer network system. Firewall is a network security system comprises with both software and hardware that monitors and controls all the incoming and outgoing network traffic, based on predefined security rules [3]. A firewall is usually located in a trusted zone, and creates a barrier between a secure internal network and an external network, such as the Internet, which is assumed to be insecure and untrusted. Firewalls are typically classified as either network firewalls or host-based firewalls. A network firewall is a software appliance that is running on general purpose hardware, or computer appliance that monitor traffic between networks. Host-based firewall provides a layer of software on one dedicated host that is assigned to specifically control all the network traffic flowing in and out of that machine.

Capable hackers will still able to gain unauthorized access to network bypassing the firewall, and this inspires an essential second line of defense in the form of IDS. IDS generally monitors the network and analyzes the traffic and activities of the computer system [2]. Figure 2.1 illustrates the common firewall and NIDS equipped in any computer network security system.

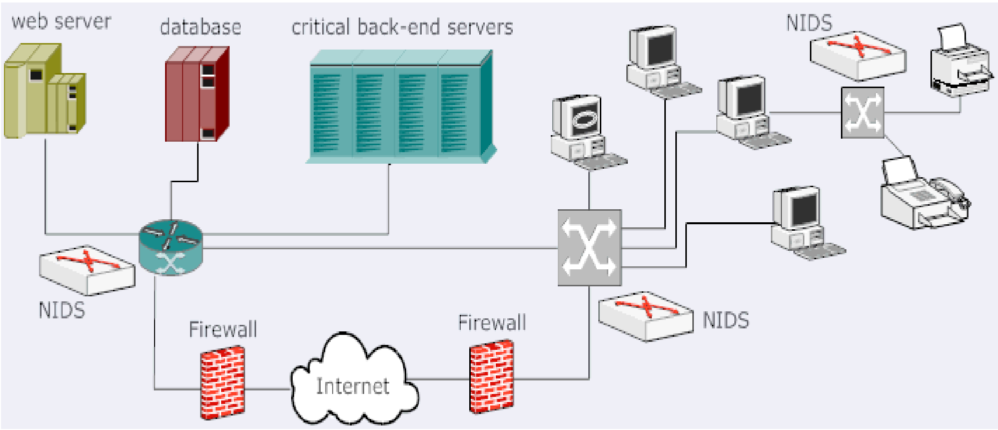


Figure 2.1 Firewall and NIDS in Computer Network Security System [10]

Subsequently, Figure 2.2 describes the basic working flow of an IDS. It is generally consist of three key modules; which is Data Collection Module, Detection Module and Response Module.

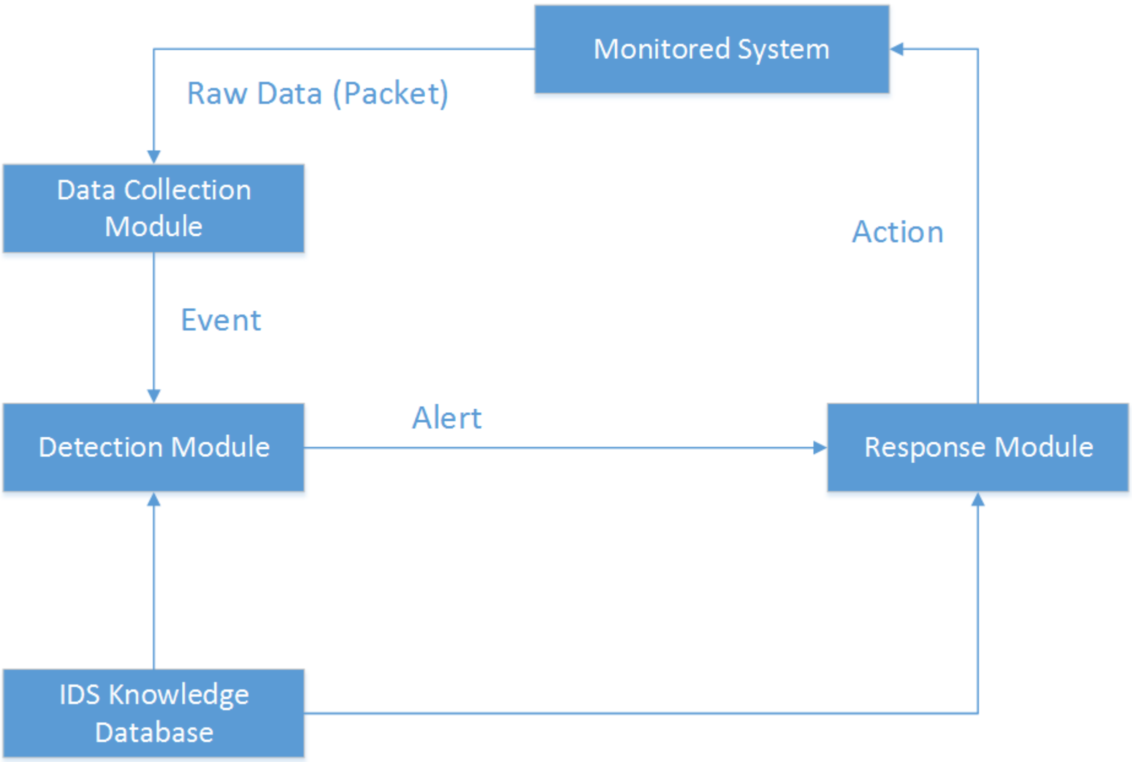


Figure 2.2 Basic architecture of IDS

The main function of Data Collection Module is to collect raw data of the network packet which is captured by the Network Interface Card (NIC) of the monitoring system; it then processes, and parses the packet information to the Detection Module. Figure 2.3 below shows the detail steps on the working flows of the Data Collection Module [2] [11].

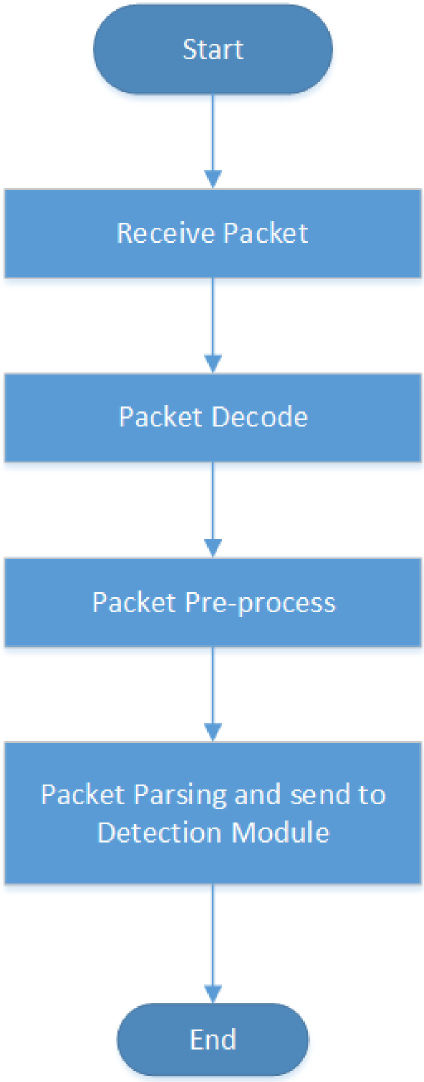


Figure 2.3 Flow chart of Data Collection Module

Normally, the packet capturing is implemented in the Operating System (OS) kernel using NIC attached on the monitored system, simplify the immediate processing and filtering of the captured packets while accessible from the user and application level. A very commonly used packet capturing process is libpcap for UNIX-based OS library kernel [12], and winpcap for Microsoft Windows based OS [13]. The captured packet will be sent to the decoder, where it will be decoded. Subsequently, its protocol format will be verified and the checksums will be validated, as well as duplicating the integrated attributes (address, protocols, ports number etc.) to a data structure that stored internally. This is followed by pre-processed stage where these packets are handled from the low-level protocol format. The packets are transformed into higher level format with abstract details comprising an event allowing the rules and signatures operation in the application layer within Detection Module.

Detection Module is the second phase of the IDS after Data Collection Module. Detection Module is the key component of an IDS. Figure 2.4 illustrates the flow chart of the Detection Module system. The packets processed by Data Collection Module are sent to Detection Module to be verified with the pre-configured rules and policies that are stored in Knowledge Database Module.

If IDS detects any intrusion, the incident must be recorded by the system. This is the responsibility of the Response Module. Typically, all incident will be recorded into a standard computer message logging. It is stored in the monitored system.

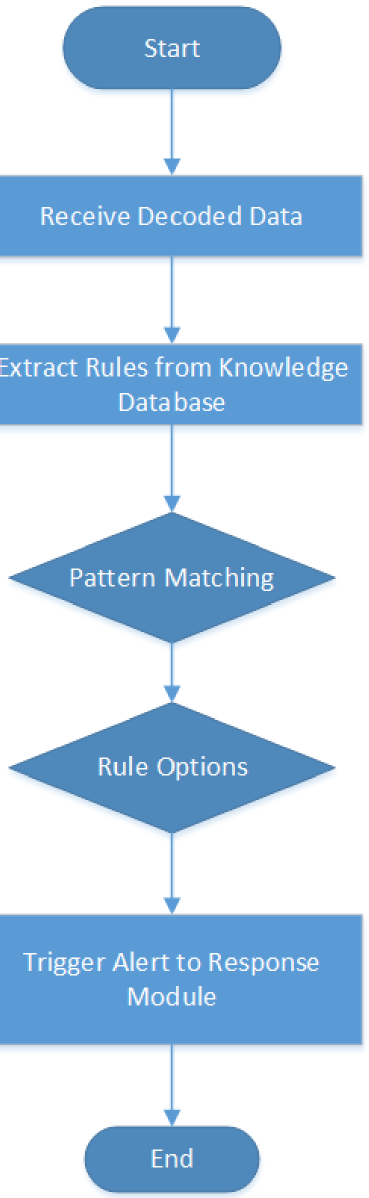


Figure 2.4 Flow chart of Detection Module

2.3. Computer Central Processing Unit

A central processing unit (CPU) is the electronic circuitry within a computer that carries out the instructions of a computer program by performing the basic arithmetic, logical, control and input/output (I/O) operations specified by the instructions. The term has been used in the computer industry at least since the early 1960s [14]. Traditionally, the term "CPU" refers to a processor, more specifically to its processing unit and control unit (CU), distinguishing these core elements of a computer from external components such as main memory and I/O circuitry [15].

2.3.1. Operations of CPU

The fundamental operation of most CPUs, regardless of the physical form they take, is to execute a sequence of stored instructions that is called a program. The instructions to be executed are kept in computer physical memory. Nearly all CPUs follow the fetch, decode and execute steps in their operation, which are collectively known as the instruction cycle.

After the execution of an instruction, the entire process repeats, with the next instruction cycle normally fetching the next-in-sequence instruction because of the incremented value in the program counter. If a jump instruction was executed, the program counter will be modified to contain the address of the instruction that was jumped to and program execution continues normally. In more complex CPUs, multiple instructions can be fetched, decoded, and executed simultaneously.

Some instructions manipulate the program counter rather than producing result data directly; such instructions are branching instructions and facilitate program behavior like looping and conditional program execution (by using conditional jump).

2.3.2. Intel CPU Architecture

x86 Instruction Set Architectures (ISA) is based on the Intel 8086 CPU and its Intel 8088 variant, and it is backward compatible. Intel 8086 was designed in 1978 as a 16-bit extension of Intel 8-bit 8080 microprocessor, with added memory segmentation as the solution for memory addressing to cover more than a plain 16-bit address. This "x86" term became popular due to several successors to the Intel 8086 processor which the model name ended in "86", including 80186, 80286, 80386 and 80486 processors.

The x86 instruction set is designed with variable instruction length, primarily Complex Instruction Set Computing (CISC) design with emphasis on backward compatibility. However, this instruction set is not a typical CISC, it is basically an extended version of the simple eight-bit 8008 and 8080 architectures.

2.3.3. 64-bit CPU Architecture

In computer architecture perspective, 64-bit computing is the system that uses the processors that have data path widths, integer size, and memory address widths of 64 bits (eight bytes). A 64-bit CPU and ALU architectures are the designs that use the registers, address buses, and data buses in 64-bit size. From software standpoint, 64-bit application means it is compiled with source code that uses 64-bit virtual memory addresses.

The term 64-bit defines a computer system in which it use 64-bit CPUs. 64 bits is a size that describes some modules within the computer architecture, like data buses, memory and the execution units, and also by the extension of the software that is executed on the system. Figure 2.6 shows the state diagram of the x86-64 operating modes.

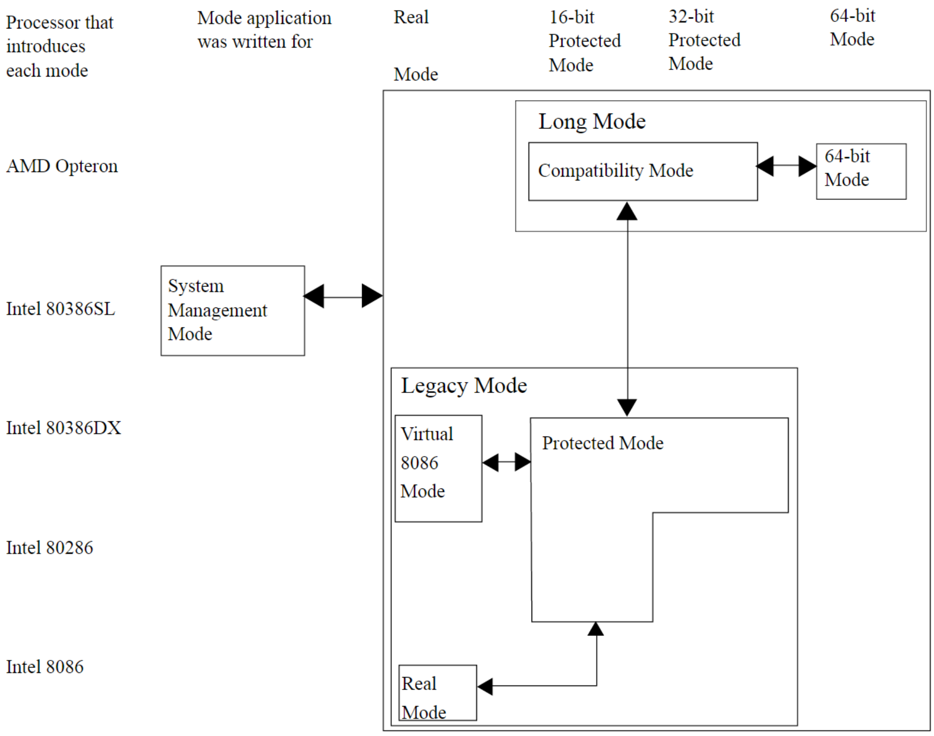


Figure 2.5 The x86-64 operating modes [16]

The generally understanding is that any 64-bit computer architecture has addressing and general purpose registers that are 64 bits wide. This allows direct access for any 64-bit data types and memory addresses. A 64-bit register can store a maximum value of 2^{64} (18 quintillion or 1.8×10^{19}). Thus, a 64-bit processor with 64-bit memory addresses can access up to 2^{64} bytes (16 exbibytes) of memory.

2.3.4. Advantages of 64-bit CPU Architecture

Many operating systems needs to be intensively altered to benefit from the new architecture changes, due to the needs of the OS to handle memory addressing hardware. Generally, an operating systems compiled for 64-bit architectures would be backward compatible, it can support both 32-bit and 64-bit applications.

On the x86-64 architecture 64-bit hardware, most 32-bit operating systems and applications can be executed without any compatibility issues. Due to larger address space available of 64-bit architectures, it is much easier to handle the usage of large data sets in applications like complex mathematic computing and large databases management. The memory space limit is so much higher for a 64-bit operating systems.

Certain 64-bit architectures like Intel x86-64 supports more general purpose registers than their 32-bit architecture predecessor. This results in significant improvement of speed for the codes that is executed in tight loops, where the processor does not have to fetch data from the memory space of internal data cache if these data can be fit into all the available registers. Figure 2.6 shows the available registers in x86-64 architecture.

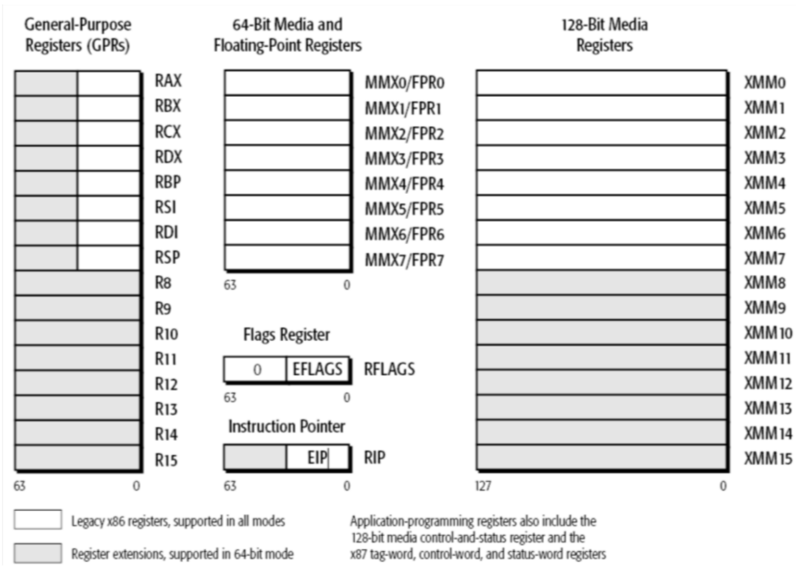


Figure 2.6 The 64 bit registers in x86-64 architecture [17]

In summary, a 64-bit processor performs best with 64-bit software with 64-bit operating systems. The 64-bit processor design takes care of the backward compatibility, most 32-bit software is well handled. It is to note that a 64-bit software is not compatible with 32-bit processor.

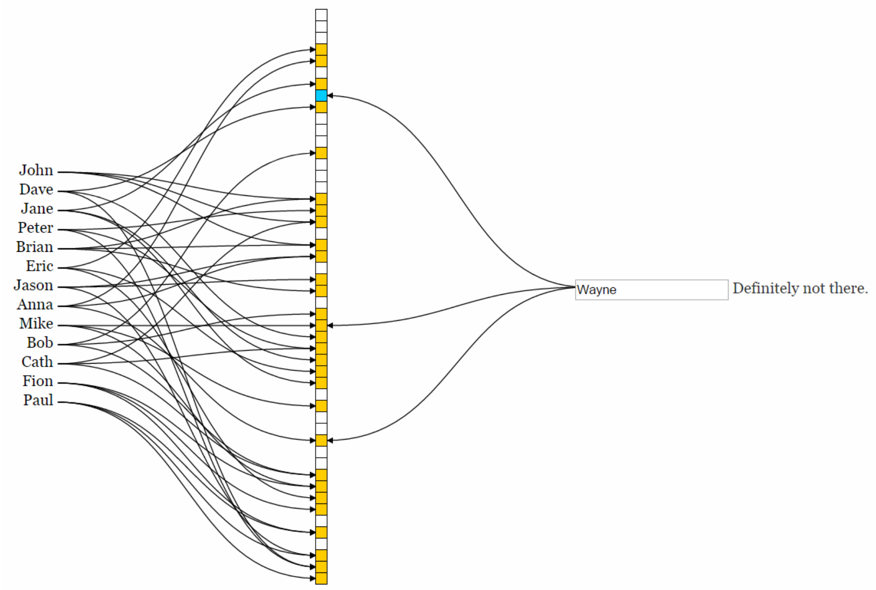
2.4. Bloom Filter Pattern Matching Algorithm

A Bloom filter is a set of data structure designed to determine rapidly and memory-efficiently, whether an element is present in a set [18]. Bloom Filter is commonly applied for pattern matching applications and software. For the Bloom Filter used in NIDS, the network packet is the sample piece of data and the predefined threat signature database is the set.

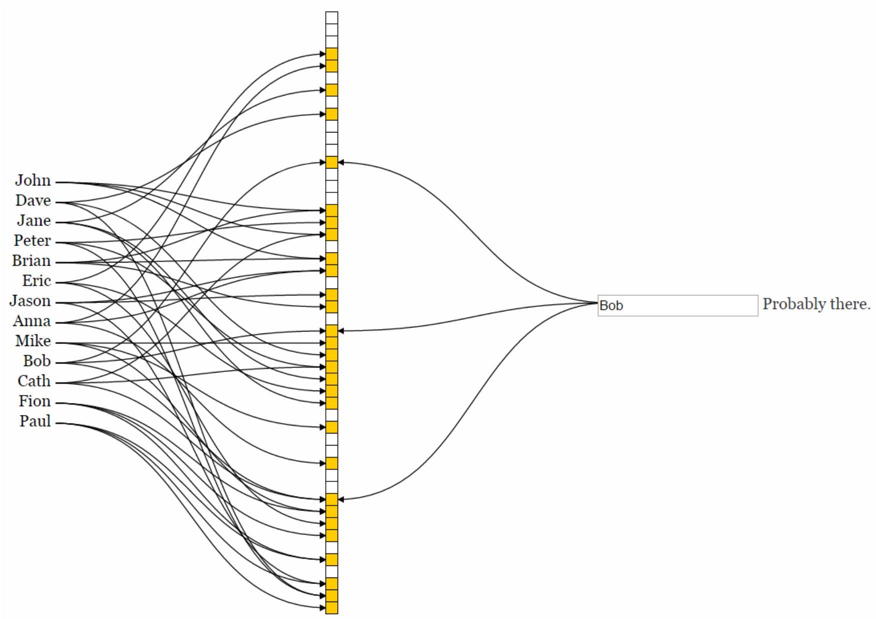
Bloom Filter comprises 2 major components; Bloom array and hash function. Bloom array is a bits array with m bits of length and it is always initialized to zero. There are also a total of k hash functions, where each function hashes some element to one of the m array positions in a uniform random distribution. A predefined set that consists of n elements must first be programmed by hashing each elements of set through the k hash functions. The output of hash functions is used as indices (or addresses in this case) for the Bloom array position, and all the bits at the corresponding addresses is set to 1.

In NIDS, a proper Bloom Filter design can determine whether the piece of data belongs to the set of a threat in database. However, there are chances of false positive occurrence where the data is identified to be a member to the set (a threat) but in fact it is not a threat. Bloom Filter will never generates a false negative due to the definitive output of the hash functions, hence there will not be any failure in NIDS to detect any intrusion with predefined signature [18] [19].

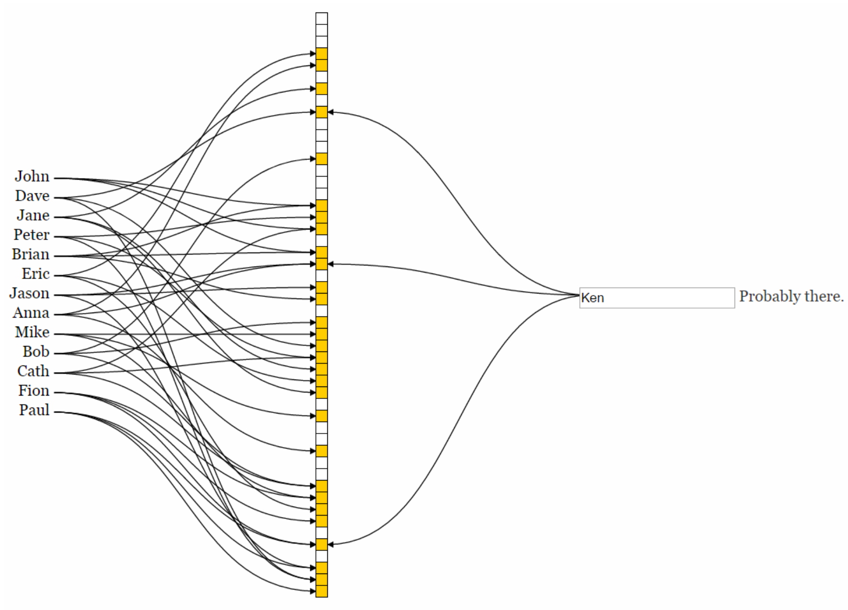
Figure 2.7 shows the three typical cases when a query went through the Bloom Filter, which are True Negative (Figure 2.7a), True Positive (Figure 2.7b), and False Positive (Figure 2.7c).



(a)



(b)



(c)

Figure 2.7 Three typical response for a query on Bloom Filter (a) True Negative (b) True Positive (c) False Positive

The probability of a false positive occurrence can be defined in the mathematic equation Eq. 2.1 [18]:

$$p = \left(1 - \left(1 - \frac{1}{m}\right)^{nk}\right)^{k} \approx \left(1 - e^{-\frac{nk}{m}}\right)^{k} \tag{2.1}$$

k = number of hash functions

m = Bloom Filter array length

n = size of Bloom Filter set

The generalized Bloom Filter (*k* greater than 1) allows many bits to be set. And if the parameters (*k* and *m*) are chosen properly, then the false positive rate will be kept low. Eq. 2.2 shows the optimum *k* value, for given *m* and *n* in order to obtain a minimum false positive rate.

By choosing suitable values for *m* and *k*, the false positive probability can be decreased. The optimal values for *k*, for given *m* and *n* in order to minimize the false positive is determined by the Eq. 2.2 below.

$$k = \frac{m}{n} \ln 2 \tag{2.2}$$

From the Eq. 2.2, the minimum false positive rate can be determined to be:

$$2^{-k} \approx 0.6185^{\frac{m}{n}} \tag{2.3}$$

The Eq. 2.4 can be used to determine an optimum *m* value for a given *n*:

$$m = -\frac{n \ln p}{(\ln 2)^2} \tag{2.4}$$

2.4.1. Operations of Bloom Filter

There are three fundamental operation of Bloom Filter. The first is Bloom array initialization. An empty Bloom filter is a bit array of m bits, all set to 0. All the bits must be initialized to 0 because any bit set indicate a hash output.

The second operation is element insertion. Every element that need to be added to the Bloom Filter needs to be fed to each of the *k* hash functions to get *k* array positions. All the *k* bit position of the array is set to 1.

The third operation is element query. The purpose of this operation is to test whether the given element is in the set. Similar to the element insertion, the element needs to be fed to each of the k hash functions to get k array positions. In order to query if the given element is a member of the set, the element is fed to each of the k hash functions to determine the k array positions. If any of the corresponding bits is 0, then it can be concluded that the element is definitely not a member of the set. If all of the corresponding bits are 1, then it is almost certain that the element is in the set, or it could be the bits that have by chance been set during the insertion of another elements, which is categorized as false positive. In a standard Bloom filter, it is impossible to differentiate between the two cases, some advanced techniques can be deploy to address this problem.

Figure 2.8 illustrates the three fundamental operations of a simple Bloom Filter with $m = 8$ and $k = 3$.

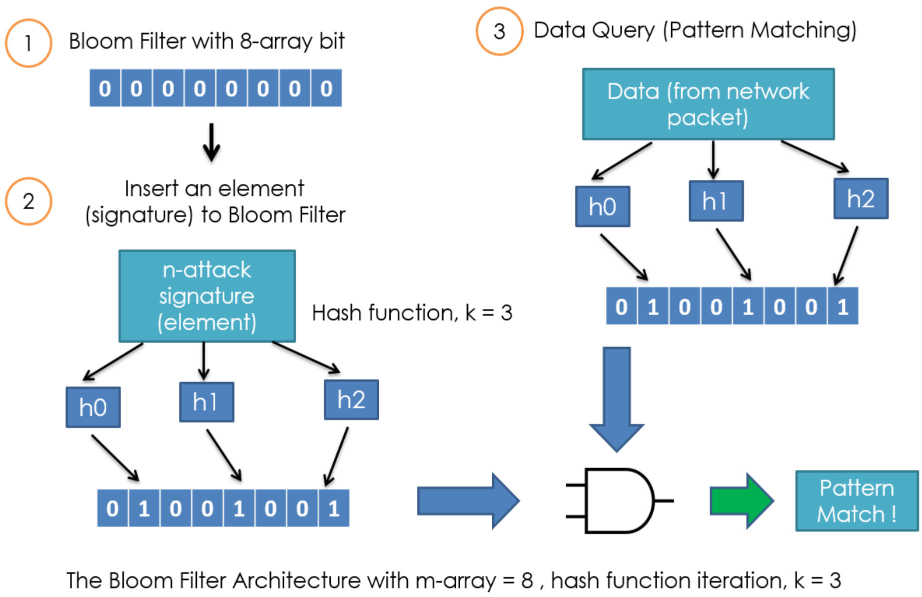


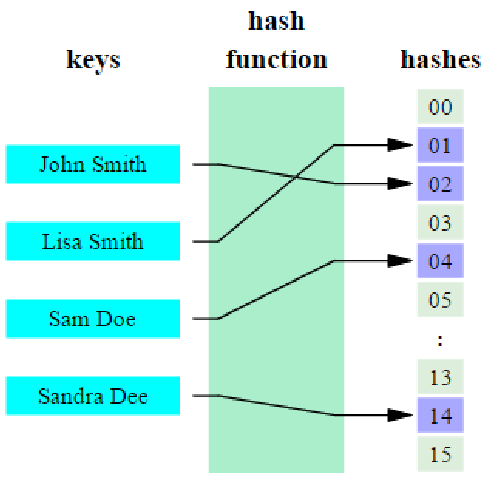
Figure 2.8 Three fundamental operations of a simple Bloom Filter

2.5. Hash Functions

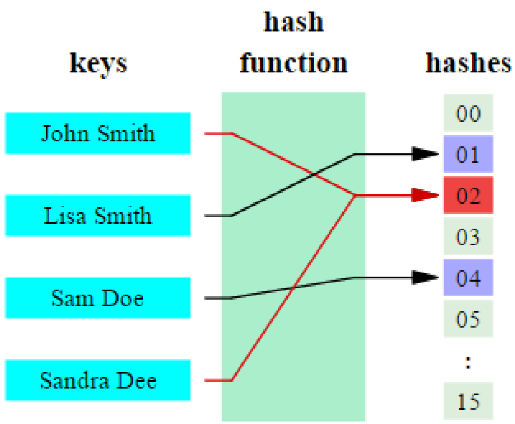
A hash function is a mathematical formula that is used to map a random size of sample data to a fixed size data output. Typically, the output of a hash function is called hash values, or simply hashes. A very common usage of hash function is called hash table, which is a widely used data structure in computer software for quick data lookup.

A good hash function is important for a good hash table performance. If a poor hash function is chosen, it likely leads to a clustering behavior, which means the probability of collision (different keys are mapped to the same hash) is significantly greater than the expected possibility from a random function.

Hashing is basically doing a one-way mapping between two sets of data. A perfect hash function is injective; it maps each valid input to a different hash value. For such hash function, the desired entry can be directly located in a hash table, without any additional searching. Figure 2.9 (a) shows the mapping for perfect hash function. A collision is defined as the occurrence of identical hash output of different input values. Practically, hash functions are rarely perfect and collisions is not uncommon. These non-ideal mapping is not unique for every entry, there are more than one entry in the input set that will be mapped to the same hash value as shown in Figure 2.9 (b).



(a)



(b)

Figure 2.9 Mapping of (a) Perfect (Ideal) and (b) Imperfect (Non-ideal) hash functions [20], [21]

The choice of hash functions determines the efficiency and accuracy of a Bloom Filter [12] [13]. A collision of the hash values in Bloom Filter array will produce false positives result. This will impact the Bloom Filter efficiency. To keep the false positive rate low, the hash function implemented in Bloom Filter needs to be very close to perfect for low collision rate. The hardware design must be simple as a complex hardware will lead to lower throughput. The hash function should be flexible in the output length for simple porting of Bloom Filter, as the array size might vary in different applications.

2.5.1. Murmur2 Hash

Murmur2 Hash is a non-cryptographic hash function suitable for general hash-based lookup. The name comes from two basic operations, multiply (MU) and rotate (R), used in its inner loop.

Murmur2 Hash yields a 32-bit or 64-bit output value. The algorithm is simple in term of number of generated assembly instructions, and produces a good output distribution. Murmur2 Hash is not specifically designed to be difficult to reverse by an adversary, hence making it inappropriate for cryptographic usage.

2.5.2. One-at-a-time Hash

Jenkins One-at-a-time hash is one of the well perform hash in terms of probability of collision [22]. Figure 2.10 shows the avalanche behavior of this hash. This image was made using Bret Mulvey's AvalancheTest. A single bit in the input is corresponded to each row, and a bit in the output is corresponded to each column. A good mixing behavior is presented by a green square, weak mixing behavior shows a yellow square, and no mixing would be indicated by red. From the figure, it can be seen that only a few bits in the last byte are weakly mixed. The performance is much better than a variety of commonly used hash functions.

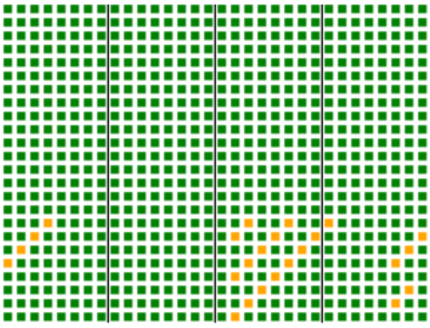


Figure 2.10 Avalanche behavior of Jenkins One-at-a-time hash [23]

2.5.3. SuperFast Hash

Bob Jenkins One-at-a-time hash is improved by Paul Hsieh [24] in which a new version of hash known as SuperFast hash is created. The hash is kind of a mixture