Worm Detection in Software Defined Networks

Thesis submitted in partial fulfillment of requirements
For the degree of
Master Of Computer Science and Engineering
of
Computer Science and Engineering Department
of
Jadavpur University

by

Om Prakash Kumar
Regn. No. - 128990 **of** $2014 - 15$
Exam Roll No. - M4CSE15-04

under the supervision of

Mridul Sankar Barik
Assistant Professor

Department of Computer Science and Engineering
Jadavpur University
Kolkata, West Bengal, India
2016

Certificate from the Supervisor

This is to certify that the work embodied in this thesis entitled **"Worm Detection In Software Defined Networks"** has been satisfactorily completed by **Om Prakash Kumar** (Registration Number 128990 of $2014-15$; Class Roll No. 001410502004; Examination Roll No. $M4CSE15-04$). It is a bona-fide piece of work carried out under my supervision and guidance at Jadavpur University, Kolkata for partial fulfilment of the requirements for the awarding of the **Master Of Computer Science and Engineering** degree of the Department of Computer Science and Engineering, Faculty Of Engineering and Technology, Jadavpur University, during the academic year $2014-16$.

Mridul Sankar Barik,
Assistant Professor,
Department of Computer Science and Engineering,
Jadavpur University.
(Supervisor)

Forwarded By:

Prof. Debesh Kumar Das,
Head,
Department of Computer Science and Engineering,
Jadavpur University.

Prof. Sivaji Bandyopadhyay,
Dean,
Faculty of Engineering & Technology,
Jadavpur University.

Department Of Computer Science and Engineering
Faculty Of Engineering And Technology
Jadavpur University, Kolkata - 700 032

Certificate of Approval

This is to certify that the thesis entitled **Worm Detection In Software Defined Networks** is a bona-fide record of work carried out by **Om Prakash Kumar** (Registration Number 128990 of $2014 - 15$; Class Roll No. 001410502004; Examination Roll No. $M4CSE15 - 04$) in partial fulfilment of the requirements for the award of the degree of **Master Of Computer Science and Engineering** in the **Department of Computer Science and Engineering, Jadavpur University**, during the period of June 2015 to May 2016. It is understood that by this approval, the undesigned do not necessarily endorse or approve any statement made, openion expressed or conclusion drawn therein but approve the thesis only for the purpose of which it has been submitted.

Examiners:

(Signature of The Examiner)

(Signature of The Supervisor)

Department of Computer Science and Engineering
Faculty of Engineering And Technology
Jadavpur University, Kolkata - 700 032

Declaration of Originality and Compliance of Academic Ethics

I hereby declare that the thesis entitled **"Worm Detection In Software Defined Networks"** contains literature survey and original research work by the undersigned candidate, as a part of his degree of **Master Of Computer Science and Engineering** in the **Department of Computer Science and Engineering, Jadavpur University**. All information have been obtained and presented in accordance with academic rules and ethical conduct.
I also declare that, as required by these rules and conduct, I have fully cited and referenced all materials and results that are not original to this work.

Name: Om Prakash Kumar

Examination Roll No.: M4CSE15-04

Registration No.: 128990 of 2014 – 15

Thesis Title: Worm Detection In Software Defined Networks

Signature of the Candidate:

ACKNOWLEDGEMENT

I am pleased to express my gratitude and regards towards my Project Guide **Shri Mridul Sankar Barik**, Assistant Professor, Department Of Computer Science and Engineering, Jadavpur University, without whose valuable guidance, inspiration and attention towards me, pursuing my project would have been impossible.

I am grateful towards all the members of **Centre for Distributed Computing Lab, Jadavpur University,** Biswajit Halder and Promit Banik, Masters Student of Jadavpur University, for cooperating with me in all possible ways and providing me with a good working environment throughout the duration of the work.

Last but not the least,I express my regards towards my friends and family for bearing with me and for being a source of constant motivation during the entire term of the work.

Om Prakash Kumar
MCSE Final Year
Exam Roll No. - M4CSE15-04
Regn. No. - 128990 of 2014 – 15
Department of Computer Science and Engineering,
Jadavpur University.

Abstract

Malware, or malicious software such as viruses, worms, trojan horses or rootkits, pose a grave challenge to the computer user community by obtaining unauthorized access to computer resources. Among various malware, worms interest computer security researchers immensely due to their ability to infect millions of computers in a short period of time and cause hundreds of millions of dollars in damage. Unlike other malware, worms can replicate themselves over the Internet without requiring any human involvement, which makes their damage potential very high.

Traditional network do not have any direct mechanism for protecting from these harmful worms. Software Defined Networking (SDN) is a new approach to networking which provides an abstraction layer for the physical network. It is considered a future technology, which has the potential to reduce complexity and costs, especially in large data centers. While SDN can also help improving network security by providing a centralized control instance, the security of the control plane itself remains an issue. As SDN is not yet an established technology, having been used mostly in experimental networks, little data about anomalies in control traffic exist. A key concept of SDN is to provide APIs for third-party applications. This makes the network more manageable and flexible. However, it also increases the risk of malware which can compromise the network. Security researchers strive to prevent, detect and contain malware attack such as worms, as well as model their propagation patterns in Software-Defined Networking. This term paper presents an idea about working and getting familiar with Worm Propagation detection and Software-defined Networking.

Recently, Software defined networking (SDN) is emerging and intensively discussed as one of the most promising technologies to make networks programmable and virtualizable. SDN promises to dramatically simplify network management, network operations, reduce cost, accelerate service delivery and enable innovation through network programmability. SDN is a promising solution for network virtualization that decouples control from forwarding or data plane. Although openness and programmability are primary features of SDN, Security is of core importance for real-world deployment.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computer Worms are reproducing programs that run independently and travel across network connections. The main difference between viruses and worms is the method in which they reproduce and spread. A virus is dependent upon a host file or boot sector, and the transfer of files between machines to spread, while a worm can run completely independently and spread itself through network connections which means it does not required any human interactions. Self-replicating malware has been an issue in computer security for many years, in the past few years, with the widespread adoption of the Internet, worms and viruses have become serious pests: spreading around the world in a matter of hours with the capacity to carry highly damaging payloads. Such malware is growing more sophisticated, as the authors of new worms learn from the successes and mistakes of the past. Detection of Worm propagation in a network in a simple or traditional network infrasturcture is complex hence a new concept of networking as Software Defined Network has been emerged by researcher where detection of any malware activity is less complex and easy compare to network currently we use.

Software Defined Networking (SDN) is a new approach to networking, it enables programmable, dynamic and flexible network architectures. SDN has the potential to dramatically reduce operational costs in large data centers. The main concept is the separation of the data plane and the control plane. The logical processing of network packets is taken over by a centralized instance, called the controller. Thus, SDN exploits the fact that software is more flexible than hardware. In large networks, the configuration of network elements is error-prone, SDN provides the advantage that configurations are centralized in one component. Furthermore, SDN bears the potential for network virtualization. Several concepts such as Virtual LANs (VLAN) were developed in the past, but the consequence was an increased complexity, especially in wide network infrastructures. Existing SDN software solutions allow network virtualization based on flexible criteria, such as network packet header fields. This provides a network abstraction layer which allows multiple tenants to share a single physical network.

1.1 Problem Statement

Additionally, SDN controllers provide APIs for third-party applications, allowing customizable network management. However, the centralization of the control plane entails risks: the controller is a critical component, and, if compromised, affects the availability of network services. Security in networks has been widely addressed in research, and several tools and techniques exist for anomaly detection. However, these techniques are limited to detecting and handling security threats in user traffic. With the emergence of SDN, new approaches should be developed on how to detect Worm. The first problem is to define what a Worm in the context of SDN is. Since SDN is not yet an established technology, no data is available regarding attacks that target the control plane. Hence, one question treated in this thesis is: what can be considered as Worm in the context of SDN control traffic? Another part of this thesis consists of elaborating possible attack scenarios and implementing them as SDN applications, in order to later analyze the generated traffic. This leads to the following research questions: how can this traffic be distinguished from normal control traffic? Which algorithms are suitable for detecting these Worms? The outcome of this work is a data analysis, a prototype and an evaluation of algorithms.

1.2 Thesis Overview

The rest of the thesis is structured as follows:

Chapter 2 presents about the concept of Computer Worm, as what is worm after becoming active different life phases a worm travel, Some examples of worm including Good as well as Bad intent worms than different propagation models which has been studied related to worm propagation to understand worms spreading behavior and to creates a detection mechanism against computer worm. Finally in this chapter I will explain two types of detection mechanism based on Signature and Anomaly behavior.

Chapter 3 presents about the concept of Software Defined Network(SDN), its history, benefits of SDN over conventional networking, architecture of SDN, various implementation issues with SDN, and OpenFlow protocol specification and various security solutions which are using Software Defined Networking concept.

Chapter 4 present experimental setup based on mininet prototyping and various experiment of network creation, controller, and security in SDN.

Chapter 5 concludes the whole thesis and discuss about future works.

Chapter 2

Worm

2.1 Introduction

A worm is defines as a piece of malicious code that duplicates and propagates itself across a network by exploiting security flaws of machines in the network and launch the most destructive attacks against computer network. The key difference between a worm and a virus is that a worm is autonomous, that is the spread of active worm does not need any human interactions. As a worm is fully automated, hence their behavior is usually repetitious and predictable, it make possible to detect worm.

2.2 Worm Life Phase

The life of a worm, after it released, typically includes the following phases :

- Target finding
- Worm transferring/Propagation
- Worm activation and
- Worm Payload.

Target finding describe how does a worm find new host to infect, Worm transferring scheme deals with How does it transmit itself to the target, Worm activation refers to mechanism by which the worms operates on the target, whereas Worm payloads describe what the worm carries to reach its goal.

2.2.1 Target Finding Scheme

The first step of a worm's life to find target, in which worm find new hosts to infect. There are many different methods [25] to find the next victim like **Blind Scan**, **Hit-List**, **Topological**, **Passive**,

Web-Search.

Blind Scan is One of the simple way in which the worm has no prior knowledge about targets. There are three types of Blind scanning, which are **random**, **sequential** and **permutation** scanning. In **random** Scanning the attacker selects a target at random, probes the target and then continues the cycle by generating a new random target. In **sequential** scan, attacker chooses to sequentially scan from a local IP address that means a worm will scan (i+1)th host after scanning ith host . In **Permutation** scanning unlike random attacker, avoid probing the same address multiple times and to coordinate the scanning of many infected hosts.

The second way of finding target **Hit-list** in which attacker use a pre scanned list of vulnerable address. In this way the worm knows exactly the address of target. The hit-list can be contained inside the worm or stored somewhere externally for worms to query.

Worms which employ **topological** scanning gather potential targets from the local machine. This includes the email addresses in a user's contact list, URLs in the user's browsing history. Topological worms use this information to gain knowledge of the topology of the network and use that as the path of infection. This makes the attack more accurate since the scanning and infection may look like normal traffic as each infected host needs to contact few others hosts and does not need to scan the entire network. Furthermore, topological worms can spread very fast, especially on networks with highly connected applications [25].

In **Passive** target finding technique the worm passively wait for incoming or outgoing connections and extract information from these connections to determine new targets. In this method worm patiently waits for target instead of seeking target aggresively. and hence due to waiting attitude this scanning is much slower than the previous techniques but can be harder to detect by intrusion detection systems (IDS). For example, Gnuman is a passive worm that acts as a Gnutella node waiting for queries to copy itself, and CRClean is a passive worm install itself on machine to counter attack from Code Red II infected host [8].

2.2.2 Worm Propagation Scheme

After the next victim is found, a copy of the worm is sent or propagate to the target. There are different schemes for worm propagation [25]:

- Self-Carried
- Second Channel and
- Embedded.

A **self-carried** worm actively transmits itself as part of the infection process. That means the worm payload is transferred in a packet by itself. In **Second channel** Worms like blaster [25] after finding the target, first goes into the target and then downloads the payload and run rest of the worm. An **embedded** worm sends itself along as part of a normal communication channel, either appending to or replacing normal messages. As a result, the propagation does not appear

as anomalous when viewed as a pattern of communication. It is hard for anomaly-based detection systems to detect. The contagion strategy [25] is an example of a passive worm that uses embedded propagation.

Apart from three propagation scheme mentioned above, **botnets** [8] have been utilized to propagate worms, spams, spyware, and launching denial-of-service attacks. A botnet use a botmaster to control a group of compromised host and different protocol such as http or point-to-point (P2P) to implement communication channel to issue commands. Witty worm is an example of worm propagated by botnets, which infected 12,000 vulnerable hosts in 45 min.

2.2.3 Worm Activation

Four methods are discussed by *Weaver ET al.* [22, 25] by which worms are activated.

- Human Activated
- Activated Based on Human Activity
- Activated By a Scheduled Processes, and
- Self Activated.

Human activatedworms need a human to manually execute the worm and frequently known as viruses. Worms which are activated when a user clicks on an email, such as the Melissa virus, or which copy infected files onto a shared folder, such as the Nimda worm, fall into this human activated category.

The second worm activation method is **human activity-based** activated worms, which are activated by a user's actions which wouldn't normally be expected to execute a worm, such as via a user's login scripts, or when a CD or memory card is inserted into the computer.

Scheduled process activated worms are activated by a legitimate automated process which hasn't been properly secured, such as a legitimate program which automatically updates itself from an infected web server.

Self activating worms are the most worrisome and begin execution immediately after being transmitted to the target. These worms generally exploit vulnerability in a running application. Buffer overflow vulnerabilities are a common target, and there have been attempts to detect packets containing buffer overflows. Code Red I and II are examples of self activating worms.

2.2.4 Worm Payload

The term **payload** of a worm refers to the behavior or actions taken by the worm, it means actual worm code. Worm sends the payload in a straightforward unchanged fashion. Worms can allow an attacker to remotely control the infected host. In this case, the attacker can execute arbitrary code

and can cause the infected host to take any desired action. This could include mounting a DoS attack against a target, collecting data from the infected host, or erasing, modifying or encrypting files on the infected host. A worm can cause the host to act as a mail relay or a web proxy. Mail relays are used by spammers to cloak the source of spam and web proxies would be used to cloak the source of undesirable websites, such as phishing sites.

2.3 Worm Examples

2.3.1 Worm With Good Intent

The first implementation of worms was by John F Shock and Jon A Hupp, researchers of Xerox PARC in 1978 [26]. The purpose of creating worm was to improve the performance of the network by finding idle processors on the network and assigns them tasks, sharing the processing load and so improving the use of the processor across an entire network. They were self-limited so that they would spread no farther than intended.

The Nachi family [26] of worms tried to download and install patches from Microsoft's website to fix vulnerabilities in the host system by exploiting those same vulnerabilities. In practice, although this may have made these systems more secure, it generated considerable network traffic, rebooted the machine in the course of patching it, and did its work without the consent of the computer's owner or user. Regardless of their payload or their writers' intentions, most security experts regard all worms as malware. Several worms, like XSS worms [5], have been written to research how worms spread. For example, the effects of changes in social activity or user behavior and communities clustered structure and size. CRclean (Code-Red Clean) [3] was a proof-of concept worm that was never released. It was designed to wait for attacks from hosts infected with Code Red II and then infect the attacking host, disable Code Red II, and filter out subsequent Code Red II infection attempts.

2.3.2 Worm With Bad Intent

The first internet worms, the **Morris worm**, whose effect gained the wide attention of the media. Morris worm was launched in November 1988 by Robert Tappan Morris, who was a student at Cornell University at that time. It is the first known worm to exploit the buffer overflow vulnerability. It targeted send mail and finger services on DEC VAX and Sun 3 hosts [8]. The Morris worm was not intended to cause any harm, but was designed to discover the number of the hosts on the Internet. The worm was supposed to run a process on each infected host to respond to a query if the host was infected by the Morris worm or not. If the answer was yes, the infected host should have been skipped; otherwise, the worm would copy itself to the host. However, a flaw in the program caused the code to copy itself multiple times to already infected machines, each time running a new process, slowing down the infected hosts to the point that they became unusable. It is a Topological Worm (6-10of the Internet's approximately 60,000 (at that time) hosts were infected within 16 hours of the worm's deployment [14]. Morris Worm Scan the local subnet as target discovery, after finding the target it uses Self carried Propagation mechanisms which exploit a fingered Buffer Overflow after than it is self activated and perform a dictionary attack against

the local /etc/passwd [14] file to obtain password for local accounts. After it get password worm get access of that account after log in.

Code Red I [8, 14] was first seen in July19, 2001 affecting more than 359,000 running Microsoft's Internet Information Server (IIS) Web service were infected by Code-Red I v2 worm in less than 14 hours. Code Red I use a blind scan scheme that scans port 80 on random IP addresses to find other vulnerable machines, and then Exploiting a buffer overflow in Microsoft IIS Web Server. In the Days 1-19 of each month The infected websites will display: "HELLO! Welcome to http://www.worm.com! Hacked By Chinese!" massage on English language, servers tries to open connections to infect randomly chosen machines using 100 threads. In Day 20-27 it stops trying to spread and launches a denial-of-service attack on the IP address of www1.whitehouse.gon.

Code Red II [8,14,27] was released one month later, Use the same buffer overflow vulnerability as Code Red, but was otherwise completely different. Code Red II no longer launches a DoS attack against predefined IP addresses; instead, it first determined if Code Red II was already installed, and if not it installed a backdoor, went dormant for a day and then rebooted the machine. It then began to spread. Installing the backdoor allowed a remote user to execute arbitrary code at a later date. It still employs blind scan but focuses more on the local subnet, and targets mainly systems with Chinese language settings. Code Red I sends its payload in monomorphic format and has a signature starting with "GET /default.ida?NNNNNNN." Code Red II has a similar signature, but replaces N with X. Both versions of Code Red are self-carried and transfer via TCP connections [8].

Nimda [8, 14] used four exploits to infect web servers running IIS, IE web browsers and Office 2000 programs. It infected IIS web servers, which then infected visitors who use IE. It also copied itself to network drives, shared the computer's folders, and created a guest account with Administrator privileges. It attached itself to explorer.exe to hide itself. It emailed itself to email addresses in the user's contact list. It was self-modifying, so hashes wouldn't identify it. Similar to the Code Red II worm, the Nimda worm allowed remote access to the infected host, as well as decreasing the security of that host by sharing the C drive.

Slammer [10, 14] was a fast-spreading worm that contained no malicious payload. Slammer exploited connectionless UDP service, rather than connection-oriented TCP. It was fully contained in a 404-byte (code size 376 byte and UDP Header 28 byte) UDP packet (Entire Worm fit in a single packet) and used a crippled implementation of random scanning. It took advantage of buffer overflow vulnerability in SQL Server. The Slammer worm (also called Sapphire worm) consists of an IP scanner combined with an exploit for MS SQL Server, written in 376 bytes of code. Worm infected 75,000+ hosts in 10 minutes (despite broken random number generator). In the first minute, the infected population doubled in size every 8.5 ($\pm$1) seconds. The worm achieved its full scanning rate (over 55 million scans per second) after approximately three minutes, after which the rate of growth slowed down somewhat because significant portions of the network did not have enough bandwidth to allow it to operate unhindered. Most vulnerable machines were infected within 10-minutes of the worm's release. Although worms with this rapid propagation had been predicted on theoretical grounds, the spread of Sapphire provides the first real incident demonstrating the capabilities of a high-speed worm. By comparison, it was two orders magnitude faster than the Code Red worm, which infected over 359,000 hosts on July 19th, 2001.

The **Witty worm** [8] launched in March 19, 2004 targeted an overflow vulnerability in eEye's Internet Security System's protocol analysis module for ICQ communications. Witty is a self-carried UDP worms that use a blind target finding scheme. It sends out UDP packets to 20,000 random generated IP addresses on random destination ports from source port 4000, with a random packet size ranging between 768-1307 bytes. The code size of Witty is only 637 bytes. It hitlist and reached saturation after infecting 12000 hosts in under 75 minutes.

2.4 Worm Propagation Model

In the real world, the knowledge of an outbreak is a necessary criterion for the subsequent development of a cure for the unknown disease. In the perspective of network security, one must first be aware that there is a worm outbreak in order to analyze the malicious traffic and then prepare a suitable patch. The worst kind of worm is one that is never detected, since such a worm can do extensive damage without the user ever being aware of it. However, even though such a worm will leave little evidence about their presence on the target hosts, in order to propagate, it will still generate a surge in the network traffic in the destination port. Therefore, one way to detect such an outbreak could be through detection of a surge in the global network traffic on specific port(s). Since noise is a significant factor in the Internet traffic, one must know the exact propagation characteristics of a worm to decide whether there is any worm activity or not. Due to these reasons, propagation modeling remains a very important tool in the war against worms. *Wang et. al* [24] has described Important propagation models based on various network topology as

- Homogeneous network in which each node has same degree (like Mesh topology) and each infected node has an equal opportunity to infect any vulnerable node.
- Random network or Non-Homogeneous network in which a random number generator is used to assign a link between two nodes.
- Small world network is a mathematical graph, which inset between a regular network and a random network which occurs after replacing a fraction p of the link d dimensional lattice with new random links. In this network a host used their neighbor to reach any other hope with minimum hops or count.
- Power low networks represented with $f_d \propto d^{\alpha}$ where f_d is frequency of out-degree d and α is constant called power-law exponent. In this network most nodes have minimum topology degree.
- Real World topology which means mostly Internet, Social networks. Propagation model of a topology-based worms can be effective and correct in any kind of network topology.

are as follows.

2.4.1 Classical Epidemic Model (the SI Model)

In classical simple epidemic model [24, 27] (also called SI model), there are only two possible states for each host: susceptible (vulnerable) or infectious. Since infection attempts have no effect on the non-vulnerable hosts, they are not considered in this model. The model also assumes that once a host is infected, it stays in that infectious state forever - there is no "infected but not infectious" state. Thus state transition of any host can only be: susceptible $\longrightarrow$ infectious. The classical simple epidemic model for a finite population is:

$$\frac{dI(t)}{dt} = \beta I(t)[N - I(t)] \tag{2.1}$$

Where I(t) is the number of infected hosts at time t; N is the size of population; and β is the infection rate. At beginning, t = 0, I(0) hosts are infectious and the other N - I(0) hosts are susceptible. Solving the differential equations for N = 2^{13} , I(0) = 1, and $\beta = 1/2^{10}$, [27] have obtained a sigmoid curve as shown in the Figure 2.1. The beginning of most of the worm epidemics matches the early part of this curve, until the effects of bandwidth limitation and network congestion set in.

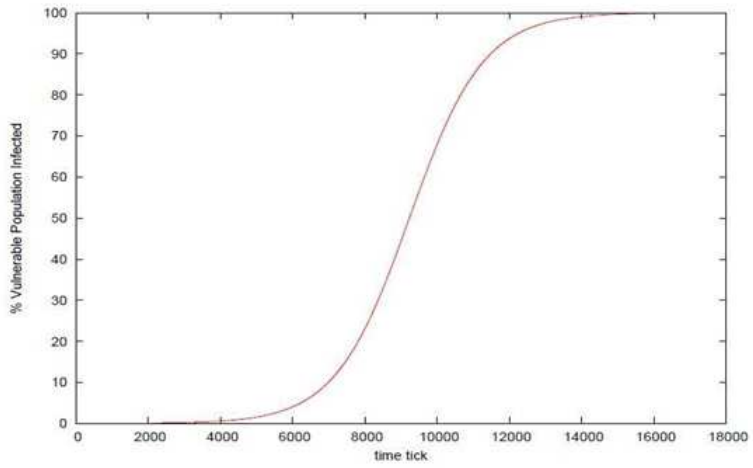

Figure 2.1: Classic Epidemic Model, [27]

2.4.2 Uniform Scan Worm Model

In Uniform Scan Worm model [24], worm scan all IP addresses uniformly as a worm has no prior knowledge of vulnerable host distribution. Once a host is infected by a worm it remains in infectious state forever. A infected host scans an average of $\eta\delta$ scans during a time interval δ where η is average scan rate. Therefore on average, an infected host has probability q to hit a specific IP address in scanning space during time interval δ.

$$q = 1 - (1 - \frac{1}{\Omega})^{\eta\delta} \approx \frac{\eta\delta}{\Omega}, \frac{1}{\Omega} \ll 1 \tag{2.2}$$

Where Ω is collection of IP addresses and $1/\Omega$ is probability to hit any address scanned by a worm.

When δ is too small than the probability that two scans sent out by an infected host will hit the same vulnerable host is negligible. Hence number of infected host at time t+δ will be

$$I(t+\delta) = I(t) + I(t).[N - I(t)]\frac{\eta\delta}{\Omega} \tag{2.3}$$

Taking $\delta \rightarrow 0$, and using epidemic model (1), the uniform scanning model can be represented by (4)

$$\frac{dI(t)}{dt} = \frac{\eta}{\Omega} I(t)[N - I(t)] \tag{2.4}$$

At time t=0, I(0) represent number of initially infected hosts and [N - I(0)] is number of all susceptible host.

A flash worm is purposed as future worm, which contains a complete hit-list (list of all vulnerable host distributed all over the world), infects a target, and pass half of its address space to that infected address. Using this mechanism, no address is scanned more than once and it possibly infects all vulnerable hosts in tens of seconds. For this reason we cannot ignore time delay ϵ which is time interval between when a worm start scans and when it infects a vulnerable host. Hence the flash worm can be modeled by (5):

$$\frac{dI(t)}{dt} = \frac{\eta}{N} I(t)[N - I(t)], I(t - \epsilon), forallt{<}\epsilon \tag{2.5}$$

2.4.3 Random Constant Spread (RCS) Model

Staniford et al. [23] presents a RCS model to model propagation of Code Red I v2 Worm, Let a(t) = I(t)/N be the fraction of the population that is infectious at time t . Substituting I(t) in equation (2.1) with a(t) we get

$$\frac{da(t)}{dt} = ka(t)[1 - a(t)] \tag{2.6}$$

With solution:

$$a(t) = \frac{e^{k(t-T)}}{1 + e^{k(t-T)}} \tag{2.7}$$

Where, k= βN, and T is a constant of integration that fixes the time position of incident. Equation (2.6)-(2.7) shows that for early t, a(t) grows exponentially.

2.4.4 Generalized Epidemic Model (the SIR Model)

Kermack - Mckendrick [27] extended the simple epidemic model by considering the removal process of infectious hosts. In this extended model, it assumes that during an epidemic, some infectious hosts either recover or die; however, once a host recovers from the disease, will never infected again. The hosts that recover or die from the disease are put in the "removed" state (which is an addition to the simple model). Thus each host can be in only one of following three states at any time: susceptible, infectious, or removed. Any host in the system either undergoes the state transition "Susceptible (S) $\longrightarrow$ Infectious (I) $\longrightarrow$ Removed (R)" or stays in "susceptible" state forever. In this model, I(t) and R(t) denote the number of hosts that are infectious and removed at time t. β is pair-wise rate of infection and γ as the rate of removal of infectious hosts. Then Kermack - McKendrick model can be represented by (8):

$$\frac{dI(t)}{dt} = \beta I(t)[N - I(t) - R(t)] - \frac{dR(t)}{dt} \tag{2.8}$$

$$\frac{dR(t)}{dt} = \gamma I(t) \tag{2.9}$$

Where N is the vulnerable host population size. we can see that from the model that in order for an epidemic to happen, the number of infectious hosts must rise initially. Thus, at the beginning, we must have $\frac{dI}{dt} > 0$, which implies βI(t)S(t) > γI(t), or S(t) > $\frac{\gamma}{\beta}I(t)$.

This $\frac{\gamma}{\beta}I(t)$ is known as the epidemiological threshold, which is defined as the number of secondary infections caused by a single primary infection. Stated differently, it determines the number of people infected by contact with a single infected person before his death or recovery. When this threshold is below 1, each person who gets infected will infect less than one person before recovering or dying, so the outbreak will eventually decrease.

2.4.5 Realistic Scenario - Two-Factor Worm Model

Although the Kermack - Mckendrick model does extend the original SI model, there are a few reasons it cannot be applied to Internet straightaway. First, the Kermack -Mckendrick model considers removal of infectious hosts only, while in reality patching, filtering and similar countermeasures remove both infectious and susceptible hosts from the total vulnerable population. Moreover, this model assumes that the infection rate is constant, which is not necessarily true for a rampantly spreading worm like Code Red. The reason for the latter event is that large-scale worm propagation causes excessive load on the routers, sometimes even overwhelming them as their ARP caches fill up. And once one router goes down, other routers need to recomputed new paths and update their routing tables, which again cause even more load, and further degradation of performance. While the degradation of performance of routers and the resulting disruption of service may very well be one of intended result of the worm, this also has a negative effect on the worm propagation itself. Because now not all scan packets would reach their destinations, the overall scanning rate (and thus scanning efficiency) of the worm decreases. We do not pursue the details of this model any further as the goal of this model is to obtain the propagation curves in presence of practical constraints like congestion and recovery. However, these conditions do not necessarily apply, or hinder, the progress of a smart worm that scans at a rather slower rate and is thus able to reduce its network

footprint drastically [27].

In two-factor model, there are two removal processes as removal of infectious hosts and removal of susceptible hosts. Denote R(t) and Q(t) as number of removed hosts from the infectious population and susceptible population. The pair-wise infection rate β is modeled as a function of time t, β(t). Then the two-factor model can be represented by (10) :

$$\frac{dI(t)}{dt} = \beta I(t)[N - I(t) - R(t) - Q(t)] - \frac{dR(t)}{dt} \tag{2.10}$$

$$\frac{dR(t)}{dt} = \gamma I(t)$$

Where N is finite population size; I(t) denotes the number of infectious hosts at time t; β(t) is pair-wise infection rate at time t; and γ represents rate of removal of infectious hosts.

2.4.6 Analytical Active Worm Propagation Model

Another discrete-time and continuous state deterministic approximation model, called Analytical Active Worm Propagation (AAWP) model [2], has been proposed by Chen et al. to model the spread of active worms that employ random scanning. It assumes worm scans in a fully connected network and no hosts can be repeatedly infected. In this model worm scan the whole IPv4 address (N=2^{32}) with equal likelihood. That means probability to hit a scan is $1/2^{32}$. Denote m_t as the number of vulnerable (including infected hosts) hosts; and n_t as number of infected hosts at time t (t$\geqslant$0). At time t=0, m_0 is equal to N and n_0 is equal to h. we assume s is the scanning rate, and number of new infected hosts in each time tick is equal to (m_i - n_i) [1 - $(1 - 1/2^{32})^{sn_t}$].Assume d represents death rate and p denotes patching rate. Thus, in each time tick the number of vulnerable hosts without being infected and number of healthy hosts will be (d+p)n_t. Therefore, on average the number of infected hosts in next time tick t+1 can be represented by (11):

$$n_t + 1 = n_t + (m_t - n_t)[1 - (1 - \frac{1}{2^{32}})^{sn_t}] - (d + p)n_t \tag{2.11}$$

In each time tick, the total number of vulnerable hosts including infected hosts is (1 - p)m_t, and thus, at time tick t, $m_t = (1 - p)^t m_0 = (1 - p)^t N$. Therefore we can derive (12) as :

$$N_t + 1 = (1 - d - p)n_t + [(1 - p)^t N - n_t][1 - (1 - \frac{1}{2^{32}})^{sn_t}] \tag{2.12}$$

Where t$\geqslant$0 and n_0 = h. Differences between the AAWP model and the epidemic model are:

1. The epidemic model uses a continuous-time differential equation, while the AAWP model is based on a discrete-time model. We believe that the AAWP model is more accurate. Because in the AAWP model, a computer cannot infect other hosts before it is infected completely. But in the epidemic model, a computer begins devoting itself to infecting other hosts even though only a "small part" of it is infected. Therefore, the speed that the worm can achieve and the number of hosts that can be infected may be very different.

2. The epidemic model does not consider the time that it takes the worm to infect a host, while the AAWP model does. Different worms have different infection abilities that are reflected by the scanning rate (or the birth rate) and the time spent to infect a host. The time required to

infect a host always depends on the size of the worm's copy, the degree of network congestion, the distance between source and destination, and the vulnerability that the worm exploits. It can be shown that the time to infect a host is an important factor for the spread of active worms.

3. In the AAWP model, we consider the case that the worm can infect the same destination at the same time, while the epidemic model ignores the case. In fact, it is not uncommon for a vulnerable host to be hit by two (or more) scans at the same time.

Both models are deterministic and try to get the expected number of infected hosts, given the size of the initially-infected hosts, the total number of vulnerable hosts, the scanning rate/birth rate, and the death rate. The epidemic model can easily deduce the closed form, while the AAWP model predicts the spread of random-scanning worms more accurately.

2.4.7 Bluetooth Worm Model

G. Yan and S. Eidenbenz present this model to propagation of Bluetooth worm which captures not only behavior of Bluetooth Worms but also the impact of mobility patterns on the propagation of Bluetooth worms. It assumes that all individual Bluetooth devices are homogeneously mixed and advances time in a discrete fashion. It derives the duration of an infection cycle $T_{(cycle)}$(t), analyzing a single infection cycle, and number of new infections out of the infection cycle α(t). The average density of infected devices in the network at time $t_k + 1$ is defined by (13):

$$i(t_{k+1}) = i(t_k).\frac{\rho(t_k)}{i'(t_k) + (\rho(t_k) - (i'(t_k))e^{-\alpha'.\rho(t_k)/(\rho(t_k)-i'(t_k))})} \tag{2.13}$$

Where i'(t_k) is the maximum value between i(t) and 1/Sinq(t) to ensure at least one infected device in the radio signal covers. $\rho(t_k)$ is the average device density at time t_k. It use α' to achieve a better estimation of worm propagation, defined by (14):

$$\alpha' = \frac{\rho(t_k) - i(t_k)}{\rho(t_k)}.\alpha t_k + \frac{i(t_k)}{\rho(t_k)}.\alpha(t_x) \tag{2.14}$$

At early phase, α' is closed to αt_k and at last phase of worm propagation, α' is close to $\alpha(t_x)$.Here t_x is the latest time when an infected device starts their infection cycle after time t but before time $t_k + 1$. This model shows that the Bluetooth worm propagates very slowly at early stage and spread quickly once the density of the infected devices reach 10 percent.

2.4.8 Local Preference Model

This model is based on continuous time model to describe the spread of localized scanning worms. In this model worm has a probability p of uniformly scanning IP addresses that have the same first n bits and probability of (1 - p) of uniformly scanning other addresses. Suppose that the worm scanning space contains K networks where all IP addresses have the same first n bits and each network has N_k (k = 1,2,?..,K) initially vulnerable hosts. I_k(t) denotes the number of infected hosts in the k-th network at time t; and β' and β" denotes pair-wise rates of infection in local scan and remote scan, respectively, hence:

$$\beta\text{'} = \frac{p\eta}{2^{32-n}},\ \beta\text{"} = \frac{(1-p)\eta}{(K-1)2^{32-n}}$$

$$\frac{dI_k(t)}{dt} = [\beta' I_k(t) + \sum_{j \neq k} \beta\text{"} I_j(t)].[N_k - I_k(t)] \tag{2.15}$$

Where η represents average number of scans an infected host sends out per unit of time. This model supposes only the first m networks (m < K) have uniformly distributed vulnerable hosts, i.e., $N_1 = ?.. = N_m = \text{N/m}$, $N_{m+1} = ?. = N_k = 0$. Thus Worm propagation follows (16) :

$$\frac{dI_k(t)}{dt} = [\beta' + (m-1)\beta\text{"}.I_k(t)[N_k - I_k(t)], k = 1, 2, ...m \tag{2.16}$$

Suppose $I_k(0) = I_1(0) > 0$, k = 2,3,??.,m. We then have:

$$\frac{dI(t)}{dt} = \frac{[\beta' + (m-1)\beta\text{"}]}{m}.I(t)[N - I(t)] \tag{2.17}$$

Equation (15) describes the number of newly infected hosts at time tick t with respect to the entire Internet.

2.4.9 Local Analytical Active Worm Propagation Model

It is also called as LAAWP model, is a discrete time model extended from AAWP model. It characterizes the propagation of worms employing the localized scanning strategy to probe subnets. The worm scans a random address with a probability of p_0. For an address with same first octet the probability is p_1, while with same first two octet, scans with probability p_2. This model assumes localized worms scan a subnet containing 2^{16} IP addresses instead of the whole Internet. This address is divided into three parts according to first two octets. Subnet 1 is special subnet which has a larger hit-list and b_1 denotes the average number of infected hosts in subnet 1 and k_1 denotes the average number of scans hitting subnet 1. 2^8 - 1 subnets belongs to subnet 2, which has b_2 infected hosts and k_2 scans on average and other 2^{16} - 2^8 subnets belongs to subnet 3 with b_3 infected hosts and k_3 scans on average. Hence the number of infected hosts in next time tick is represented by (18):

$$b_{i+1} = b_i + (\frac{N}{2^{16}} - b_i)n_i[1 - (1 - \frac{1}{2^{16}})^{k_i}] \tag{2.18}$$

Where i=1,2 or 3. k_i (i=1,2 or 3) indicates total number of scans in different subnets coming from the local subnet. The k_i (i=1,2 or 3) is as follows:

$$k_1 = p_2\ sb_1 + p_1\ s\lfloor b_1 + (2^8 - 1)\ b_2 \rfloor\ /2^8 + p_0\ s\lfloor b_1 + (2^8 - 1)\ b_2 + (2^{16} - 2^8\ )\ b_3 \rfloor / 2^{16}$$
$$k_2 = p_2\ sb_2 + p_1\ s\lfloor b_1 + (2^8 - 1)\ b_2 \rfloor\ /2^8 + p_0\ s\lfloor b_1 + (2^8 - 1)\ b_2 + (2^{16} - 2^8\ )\ b_3 \rfloor / 2^{16}$$
$$k_3 = p_2\ sb_3 + p_1\ sb_3 + p_0\ s\lfloor b_1 + (2^8 - 1)\ b_2 + (2^{16} - 2^8\ ) b_3 \rfloor / 2^{16}$$

.

The LAAWP model adopts deterministic approximation to reflect the spread of worms that preferentially scan targets close to their addresses with a higher probability.

2.4.10 Email Worms Simulation Model

This falls under Topology-based model (describe structure dependent propagation of worms). This model used to study propagation of Email worms, considered the probability of opening an email attachment and email checking frequency, and then compared internet email worm propagation on power law topologies, small world topologies and random graph topologies. According to distribution of Yahoo! Email groups checked by author [24], they believe the internet email network conforms to a heavy-tailed distribution and model the email network topology as a power law network following $F(\alpha)= K^{-\alpha}$ with constant α as a power law exponent that determines the degree of nodes in the network. A larger maximum topology degree requires a larger power law exponent and a larger expected value of topology degree demands a smaller power law exponent. This model uses $\alpha = 1.7$ to generate the power law network with the total number of hosts $|V| = 100000$ and an average degree of 8. The highest degree for this power law network is 1833 and lowest degree is 3.

Email worms depend on two factors either emails users' interaction to spread, when an user checks an email with a malicious attachment. This model has used $C \sim N\ (0.5,0.3^2)$ as user's behavior or opening probability. And email checking time which is an important parameter to the propagation speed of the email worm. In this model email checking time T follows a Gaussian distribution: $T \sim N(40,20^2)$. This model discusses two different cases according to their infection assumptions: non-reinfection and reinfection, whether a user in the infectious state can be infected again. If the victim can be infected each time they are visited by worms, it is assumed to be reinfection scenario. Otherwise, infected users send out worm copies only once even if they open a worm attachment again, known as non-reinfection scenario. Author has purposed an algorithm which considers the propagation of reinfection email worms which shows that each time tick an infected host send worm copies to other host and check whether target user is vulnerable (not healthy) then user is checking email or not? After getting confirmation that target user has checked the malicious attachment email worm infect the user and sends link or a copy of itself to all users who is linked with new infected user.

According to discrete-time email simulator author gets Figure 2.2 (taken from [24]), showing the propagation of email worms on a power-law network under the non-reinfection and reinfection scenarios.

Figure shows that spreading speed in reinfection case is faster and the number of infected hosts at the end of propagation is higher than the non-reinfection case.

2.4.11 Logic 0-1 Matrix Model

This model uses a logic matrix (denoted by T) approach to model the spreading of P2P worms. Here T represents the topology of a P2P overlay network. It adopts two constants of logic type (True or 1, False or 0) as the value of matrix variables. The logic constant 'T'/1 indicate the existence of a directed link between two nodes in the network, and the logic constant 'F'/0 indicate there is no directed link. The i-th row of a topology logic matrix represents all outbound link of node i; and the j-th column of the topology logic matrix represents all inbound links of node j. The logic 0-1 matrix model propagation of P2P worms under three different distributions: infectious state (denoted by logic vector S), vulnerability status (denoted by logic vector V) and quarantine

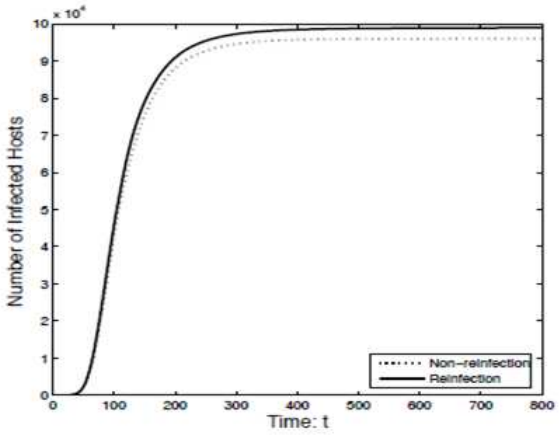

Figure 2.2: Propagation on a power-law network: reinfection vs. non-reinfection, [24]

status (denoted by logic vector Q). Where the logic vector S_g represents the current state g of the logical P2P overlay network and the logic vector S_{g+1}represents next state of the logical P2P overlay network, we have (19):

$$S_{g+1} = S_g + S_g^{new} \tag{2.19}$$

Here, 1-entries in the vector S_g^{new} represent the transition to infectious at state g+1. S_g^{new} Varies in consideration of different distributions of S, V, and Q. If all nodes are vulnerable to the worm and no nodes are quarantine, then we have (20):

$$S_{g+1} = S_g + S_g T \tag{2.20}$$

If all nodes are not vulnerable to worm and no nodes are quarantined, then we have (21):

$$S_{g+1} = S_g + S_g TV \tag{2.21}$$

If all nodes are vulnerable and some nodes are quarantined, then we have (22):

$$S_{g+1} = S_g + S_g T\overline{Q} \tag{2.22}$$

Where $\overline{Q}$ stands for distribution of those unquarantined nodes.

2.4.12 OSN (Online Social Networks) Worms Model

This model purposed worm models based on two categories as application platform based model and sending message based model of Facebook, most popular social network service providers. In Facebook, two users' accounts appear in each other's friend list if they have confirmed their status to be friend. Thus topology of this network is treated as undirected graph and is constructed by a power-law distribution in the models.

Facebook application platform based model: Facebook provides an application platform that can be utilized by attackers to publish malicious applications. Users of Facebook can install application to their accounts through this platform. If a user added a malicious application, their account is infected and an invited message is sent to all their friends to invite them to install the same application, which helps to spreading of the worm application. The probability of installing one application for user i is:

$$P_{user}(i,t) = \frac{AppS_i(t)^{\rho} + init_{user}}{\sum_{j=1}^{N_{user}}(AppS_j(t)^{\rho} + init_{user})} \tag{2.23}$$

Where $AppS_(i)$(t) is the number of applications that user i has installed at time step t. The parameter ρ reflects the effect of preferential installation. $init_{user}$ is used to show initial probability of P_{user}(i,t) of a user who does not install any application. As we know there are many new installations every day, hence the probability of selecting an application by user i from the application list is:

$$P_{app}(k,t) = \frac{Install_k(t) + init_{user}}{\sum_{j=1}^{N_{app}}(Install_j(t) + init_{user})} \tag{2.24}$$

Where $init_app$ defines the initial probability P_{app} (k,t)of an application without any installation. An invitation message is sent to all friends of a user after installing a malicious application by that user. Assume each user has received c invitaitons at time step t. Then probability of infecting a user is:

$$P_{virus} = \frac{\alpha}{(1 - \frac{Install_{N_{app}(t)}}{N_{user}} . \frac{APPS_i(t)}{N_{app}})c} \tag{2.25}$$

Where σ is the percentage of user who accepted the invitations.

Sending Message Based model: When a user of Facebook receive malicious emails and click them, those users are infected and worm sent one copy of itself to their friends. At each time a user can log-in to Facebook with a log-in time T_login (i), which follows an exponential distribution N(μTl(t),σ^2_{Tl}). T_{online} (i) represents the time a user spend on Facebook, Which follows a Gaussian distribution N(μT_o(t),σ^2_{To}). All online users may open the malicious email with a probability of

P_click , which follows a Gaussian distribution N(μ_p(t),σ_p^2).

2.4.13 Spatial-Temporal Model

A spatial-temporal random process was used to describe the statistical dependence of malware propagation in arbitrary topologies. This spatial-temporal model is stochastic discrete time model that reflects the temporal dependence and the spatial dependence in the propagation of malware. The temporal dependence means that the status of node i (infected or susceptible) at time t+1 depends on status of node i at time t and the status of its neighbors at time t. The temporal dependence of node i can be shown as (26) and (27):

$$P(X_i(t+1)=0|X_i(t)=0)=\delta_i \tag{2.26}$$

$$P(X_i(t+1)=0|X_i(t)=0,X_{N_i}(t)=xN_i(t))=\beta_i(t) \tag{2.27}$$

Where X_i (t) denotes the status of a network node i at time t (t represents discrete time): X_i(t)=1, if node i infected at X_i(t)=0, if node i is susceptible at time t. X_{N_i}(t) denotes the status of all neighbors of node i at time t and vector x_{N_i} (t) is realization of X_{N_i}(t). A susceptible node can be compromised by its infected neighbor and can be infected at next time step t+1 with a birth rate β_i(t). Otherwise, node i is infected and has a death rate δ_i to recover at the next time step t+1.

R_i(t) Denotes the probability that node i recovers from infected to susceptible status at time t+1, is:

$$R_i(t)=P(X_i(t+1)=0|X_i(t)=1)=\delta_i P(X_i(t)=1) \tag{2.28}$$

If node i is susceptible at time t, the probability that node I remains susceptible at next time step can be defined as:

$$S_i(t)=P(X_i(t+1)=0|X_i(t)=0)=\sum\nolimits_{xN_i(t)}[P(X_{N_i})(t)=xN_i(t)|X_i(t)=0)(1-\beta_i(t))] \tag{2.29}$$

Where a joint probability P(X_{N_i}(t)= x_{N_i}(t)|X_i (t)=0) representing the status of neighbors of node i at time t, characterizes the spatial and temporal statistical dependence according to the network topology and the interaction between nodes Based on (28) and (29), the probability that node i is infected at time t+1 can be represented by (30).

$$P(X_i(t+1)=1)=1-R_i(t)-S_i(t)P(X_i(t)=0) \tag{2.30}$$

Formula (25) reflects an iteration process of malware propagation according to the status of a node at time t and the status of all neighbors of this node i at time t, which characterizes the spatial and temporal statistical dependencies. The expected number of infected nodes at time t, n(t), can

be computed:

$$n(t) = E\lfloor \sum_{i=1}^{M} X_i(t) \rfloor = \sum_{i=1}^{M} P(X_i(t) = 1) \tag{2.31}$$

The *Independent Model* assumes that the status of all nodes at time t is spatially independent. This means no propagation cycles are formed when worms propagates via some intermediate nodes because the infected probability of a node is not influenced by its neighbors. Thus, independent model neglects the spatial dependence. The state evolution of node I in the independent model can be represented by (32):

$$P(X_i(t+1) = 1) = 1 - R_i(t) - S_i^{ind}(t)P(X_i(t) = 0) \tag{2.32}$$

Where

$$S_i^{ind}(\mathrm{t}) = \prod_{j \in N_i} \lfloor 1 \text{ - } \beta_{ji} P(X_j(t) = 1) \rfloor$$

The *Markov Model* assumes that the status of a node is related to its neighbors, but its neighbors cannot be influenced by each other at the same time. This assumption formed propagation cycles via a single intermediate node. If the status of node i's neighbors at the same time step is spatially independent given the status of node i, then the state evolution of a node in the Markov model can be represented by (33):

$$P(X_i(t+1) = 1) = 1 - R_i(t) - S_i^{mar}(t)P(X_i(t) = 0) \tag{2.33}$$

Where

$$S_i^{mar}(\mathrm{t}) = \prod_{j \in N_i} \lfloor 1 \text{ - } \beta_{ji} P(X_j(t) = 1 | X_i(t) = 0) \rfloor$$

2.4.14 Comparison of Worm Propagation Models

The Classical simple epidemic is the most widely used model for investing the propagation of scan-based worms using a continuous-time differentia equation. Others model such as uniform scan worm model and the RCS model are derived from classical simple epidemic model Which assumes two stats for all hosts: susceptible and infectious, and will stay in infectious state forever when a host is infected. Nut this model is not suitable for the case where the infected and infectious nodes are patched or removed. Hence classic general epidemic model (Kermack-McKendrick model) has been proposed to extend simple epidemic model by introducing a new remove state of infectious host. Again this model has neglect the human countermeasures and decrease in infection rate which lead to develop a two-factor worm model.

The above models adopt a continuous-time differential equation to observe and predict worm spreading in the network. Modeling worm propagation at each discrete time tick is more accurate than using continuous time. The AAWP model, The LAAWP model and the Bluetooth worm model are constructed using discrete event process. The AAWP model characterizes the spread of active worms that employ random scanning. LAAWP is extended from AAWP and takes into account the characteristics of local subnet scanning worms spreading. The Bluetooth models study the propagation of Bluetooth worms. It captures not only the behavior of the Bluetooth protocol

but also the impact of mobility patterns on the propagation of Bluetooth worms.

Above models were used to study propagation of worms in homogeneous network .Topological Worm model has been proposed to study the propagation of worm in topological network. Email worms propagation model is used to study the propagation of email worm propagation over a power-law topology. However, this model describes the email worm propagation tendency instead of modeling the dynamic spreading procedure between each pair of nodes. Two cases has been discussed either non-reinfection or reinfection, but this model is not capable of accurately eliminating the errors caused by reinfection.

The logic 0-1 matrix model employs a logic matrix to represent link between each pair of hosts and model the spreading of peer-to-peer worms over a pseudo power-law topology.

Social network have become attractive targets for worms and OSN model has been proposed to characterize the behavior of a worm spreading on the application network of Facebook. This model assumes a user starts infecting other at every moment once the user is infected. In practice however, infected users spread worms only as they periodically accept invitations and install malicious application or check newly received message and open malicious links. As a result they have neglected a realistic temporal delay process.

The above models assume computer users behave independently, that is, the status of all hosts at the same time step is spatially independent. However, In real scenario, the propagation of topology-based worms required human activation. A Spatial-Temporal model has been proposed in which a spatial-temporal has been used to describe the statistical dependence of worm propagation in arbitrary topologies. Although this model can outperform the previous models through capturing temporal dependence and detailed topology information.

2.5 Worm Propagation Prevention

Worm Propagation prevention can be achieved in first two life phase either in Target discovery as mentioned in section 2.2.1 (Target finding scheme) using either IP Scan whether it is Blind Scan (Random, Permutation, Sequential) or Email Scanning (Topological Search) or Web-Search, or code delivery (Worm transferring) as discussed in [14] using Network File System, Email, Web Clients or command shell we need some prevention ideas.

To prevent from IP Scanning the basic scheme is to turn off the unneeded services (ports) that when a infected host send TCP SYN to create connection to target host than as service or port is turned off it will not response and after timeout infected host will get TCP RST response but sending many time TCP SYN and receiving TCP RST congest traffic between infected host and target host and it is not possible to turn off all unneeded services as very often services remains on even if they are not needed hence we need to deploy a firewall as shown in Figure 2.3 (taken from [14]) somewhere between infected host and target hosts which will intercept SYN packets sent from any host and discard if found request from infected host as infected host send request multiple time in same manner.

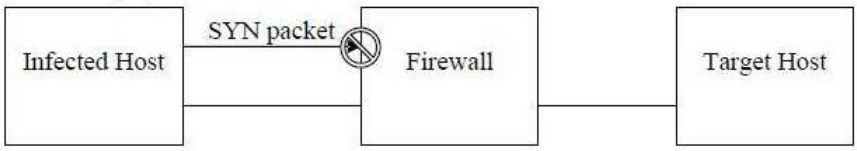

Figure 2.3: Firewall Operation, [14]

To Prevent of hostile code delivery using network filesystem We need to stop to mounting filesystem but this is not true in case of pre-mounted filesystem. Hence we need an authentication mechanism based on IP access control with username and password which allow an authorized IP address to gain access of a file if the owner of that IP address provide correct username and password and every user should choose a strong password and make changes to their password in certain period of time to prevent from dictionary attacks. This scheme is also not good enough if infected host is authorized user in that case mounted file should be read only and we need a technology which is able to monitor flow of data over network, and must understand the network filesystem protocol that is being used to deliver the code.

A worm transfer hostile code via Email using SMTP (port 25) protocol for sending or receiving mail, SMTP allow to send or receive a mail between any two devices without requiring much authentication. As we know a worm send its code via email from one infected host to multiple target host Figure 2.4 (taken from [14]).

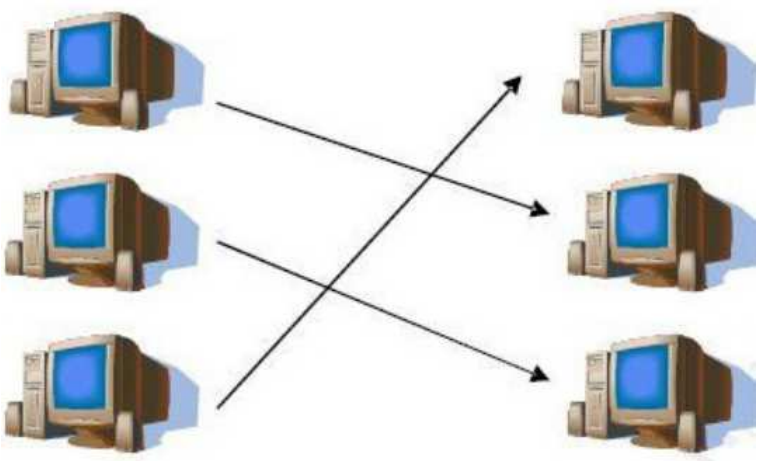

Figure 2.4: An Email delivery Configuration, [14]

To prevent this first we have to reduce this multiplicity of delivery paths to a small, finite no. of path and System administrator configure a firewall which block outgoing connection to port 25 originating from system controlled by them and forces client to send mail via designated mail-relay.

This will prevent an infected system to send email directly to target system. If a worm provides its own SMTP engine other than message mechanism then code delivery will be stopped. System administrator allow the same mechanism/rule of blocking for incoming connection to port 25 to make sure that it is receiving email from a designated mail exchange Figure 2.5 (taken from [14]).

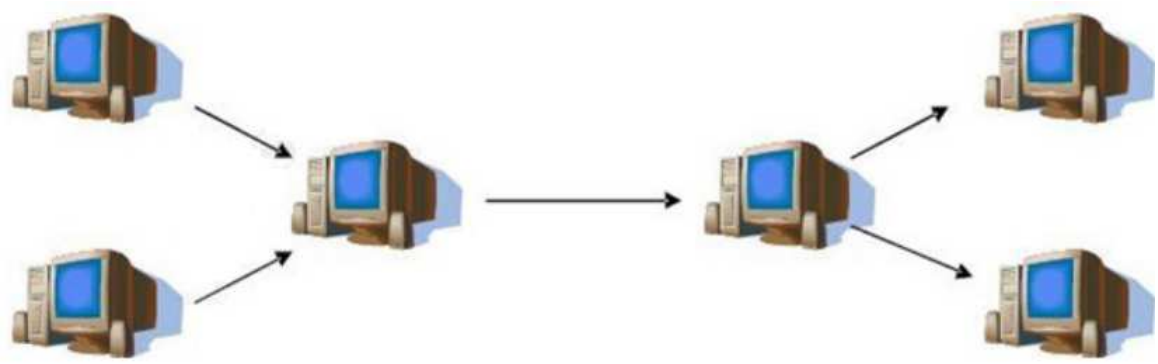

Figure 2.5: Mail delivery using designated relay and Mail exchange, [14]

It significantly reduce the number of potential delivery paths that system administrator can monitor. Now the challenge is to detect hostile code traversing from one of those paths via SMTP and prevent it to from being delivered. As all email is delivering from administrator designed mail-relay this is the first place to deploy an anti-virus gateway/Firewall which can scan large volume of email and detect virus or worms etc., and discard if a worm is found.

The same prevention concept is used for worm code transfer through web-clients using HTTP (port 80) or HTTPS (port 443) in this case a HTTP proxy server is designed for communication causes web content delivery Shown in Figure 2.6 (taken from [14]).

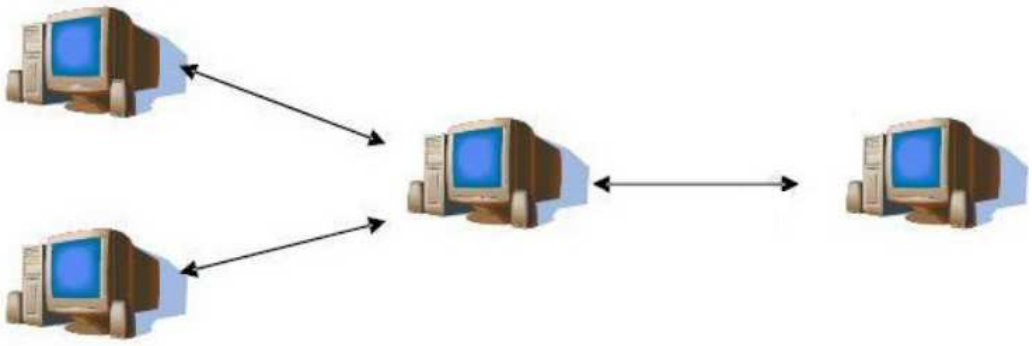

Figure 2.6: Web Browsing based on Web Proxy Server, [14]

We need technology at the web proxy which can monitor HTTP data flowing from server to the

client and which can intercept malicious content. Many anti-virus gateway/firewall system have an HTTP filtering component that perform the same task. Various worms like Morris and Spida use command shell or command shell equivalence to deliver malicious code to target system in the form of command sequences to be executed on target system. In this case first infected system make contact with a network service that provides access to a command shell in some fashion either through a command shell like services such as Telnet or RSH, directly or XP cmdshell stored in MS SQL server, indirectly. To prevent from this type of delivery we have to prevent random system from being able to access the service in first place but it becomes more complicated due to its nature of services. System administrator use command shell service to perform remote administration of the system under their control. We use the concept of Virtual Public Network (VPN) service which allow a client to access some specific services only. We deploy a VPN access devices, which is used to tunnel traffic to the rest of the internal network, with VPN client software, which establishes an encrypted connection with VPN access device. We deploy VPN access with a Firewall that prevent anyone else from accessing individual other than the authorized person. But attacks from within the administrator domain is still a problem as we know that firewall is unable to prevent malicious/threats resides within network. In this case delivery of hostile code prevented by command shell services itself using strong authentication enforcing good password.

2.6 Worm Detection Scheme

In this section various scheme which is used to detect active worm, has been described, basically Most of the worm detection mechanism can be described as network-based or host-based. The key distinction between the two is where each is implemented. Host-based solutions are implemented on the servers which run the services under scrutiny, and are useful for observing and changing the way the services operate when threats are detected. Host-based solutions can be implemented as modifications to the network stack, modifications to the operating system, modifications to the application, or even implemented by running the desired programs within an emulated environment. Network-based IDS collect data directly from the network, and are installed at gateways between internal networks and the Internet. At these locations, the network-based IDS can analyze both incoming and outgoing traffic for malicious traffic. They can also be placed between sub networks of larger organizations, such as between the networks of separate departments. All Intrusion Detection are classified between Signature-Based Detection and Anomaly-Based Detection.

2.6.1 Signature-Based Detection

Signature-based detection scheme collect signature from all known worm and is normally used for detecting known attacks. A Signature is a string of character that appear in payload of worm packets as part of the attack. This type of IDs does not care about how worm find their target, propagate, become active or any normal traffic, it just maintain a database to look into worm payload to check whether it is worm infected or not.

Signature-based IDs has many limitation at first to maintain Signature database is a tough job as it may contains thousands of signature of each kind of worm, When this scheme is used it compare all Signature entries of the database with packet this make the scheme resource consuming

and slow to detect. Since every packet is examined, signature-based systems can catch worms that employ self-carried or second channel propagation schemes. Signature based scheme can't detect Embedded worm because payload of embedded worm vary from worm to worm depending on the embedding method used to create worm.

2.6.2 Anomaly-Based Detections

Signature detectors, use pre-generated signatures to detect traffic or behavior that is known to be malicious or undesirable. The signatures can be either man-made, or automatically generated, though in either case, an ideal signature should match only the malicious traffic and no legitimate traffic. Signature detectors can detect known attacks but are unable to detect new attacks. Since fast-spreading worms can infect most of the Internet before man-made signatures can be crafted. Automated signature generating systems are needed. These signature-generating systems are typically called anomaly -based detectors. Most anomaly-based detectors do not care about the payload format or content; instead, they check the headers of packets to define the type of connection to which the packet belongs. Anomaly detectors detect unusual patterns, or anomalies, in network traffic or in the behavior of a running program. The anomaly being detected depends strongly on which anomaly detector is being used. Examples of some current methods include detecting segments of executable code, unusual byte frequencies. Once an anomaly is detected, most anomaly detectors can then generate a signature which can be used by signature based IDS. Some Important Existing anomaly detections scheme are Destination-Source Correlation, Honeypots, described in next section.

2.6.2.1 Dynamic Quarantine Defence

Firewall or routers can inspect packets content according to the signature of known worms and packet will be dropped automatically when firewalls or routers find out the signature of a worm but its impossible to detect unknown worm as no signatures is available that means we have to rely on behavior based anomaly detection to detect an unknown worm, and behavior based detection methods has a common problem of high false alarm rate, which automatically block many legitimate connection hence *Zou et. al.* [28] purposed Dynamic Quarantine Defence method to detect an unknown warm based on principle "assume guilty before proven innocent".

Whenever a host show suspicious behavior than that host will be quarantine and after a short time (called quarantine time T) that quarantine will be released. If the worm anomaly detection program we use in the system can determine which service port has suspicious activities than the quarantine means only block traffic on the suspicious port without interfering normal connections on other ports.

In the real implementation, security staff should inspect quarantined hosts as quickly as possible But for fast spreading worms, due to the slow human?s manual response and limited human resources, the inspection by security staffs cannot catch up with the increasing speed of the number of alarmed hosts. Therefore, in order not to severely interfere normal activities, the quarantine on a host will be released automatically after a while even if the host has not been inspected by security

staffs yet.

This dynamic quarantine method has two advantages: first, a falsely quarantined healthy host will only be quarantined for a short time, thus its normal activities will not be interfered too much; second, because now we can tolerate higher false alarm rate than the normal permanent quarantine, we can set the worm anomaly detection program to be more sensitive to a worm?s activities. Thus we can detect and quarantine more infected hosts and detect them earlier.

2.6.2.2 DEWP(Detector for Early Worm Propagation)

Chen et. al. [1] has developed DEWP(Detector for Early Worm Propagation) based on three essential requirements for such a system.

1. system must detect worm propagation at a very early stage to suppress the worm before it gets out of control.
2. a worm defense system needs to create worm signatures without human interference in order to react and quarantine worm propagation rapidly.
3. it is important to reduce the false alarm rate of worm detection because false positives potentially lead to denial-of-service to legitimate traffic.

DEWP detects worm intrusions, creates worm signatures, and contains worm propagation automatically, without human interference. DEWP applies a novel worm detection algorithm by matching destination port numbers between incoming and outgoing connections. This idea is from the following two observations on worm traffic. First, a worm usually exploits particular security vulnerability corresponding to specific network port numbers. Second, the nature of worms is that an infected host will probe other vulnerable hosts exploiting the same vulnerability. Therefore, routers seeing unusually high levels of bi-directional probing traffic with the same destination port number can infer a new worm has arisen.

Figure 2.7 (taken from [1]) shows the two components that make up DEWP: the worm detector and packet filter. DEWP detects worm intrusions with two steps: destination port matching and destination address counting. After it discovers a worm attack and the corresponding destination port number, DEWP deploys packet filter to block worm probing traffic.

2.6.2.3 Behavior Approach to Worm Detection

Ellis et. al. [4] presents a new approach to the automatic detection of worms using behavioral signatures. A behavioral signature describes aspects of any particular worm?s behavior that are common across the manifestations of a given worm and that span its nodes in temporal order. Characteristic patterns of worm behaviors in network traffic include

1. sending similar data from one machine to the next,
2. tree-like propagation and reconnaissance, and

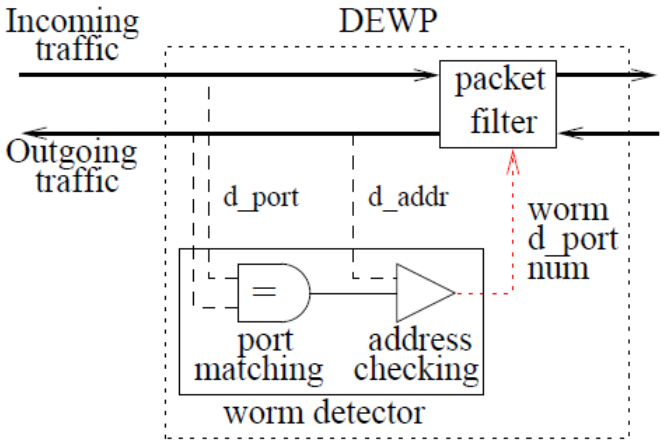

Figure 2.7: DEWP architecture, [1]

3. changing a server into a client.

Ellis et. al. [4] used The ACN framework (abstract communications network - is a network-theoretic model of computer networks and the data flows that cross them) to express worm propagation network , behavioral signatures, and NAAs.

The ACN consists of nodes that represent three elements (hosts, routers, and sensors) and four types of links (hard- ware, routing, data flow, and spans). The hosts include both user workstations and servers. The routers compute the routing tables, enforce routing policy, and perform the actual switching of the message flow. The sensors collect a subset of the traffic on the network. The (symmetric) hardware links represent a physical communications medium whereby data can flow from one computational element to another. The routing links represent the direction that data gets routed from one host to another. The data flow links rep- resent conversations between computational elements. The primary focus in this paper is on the data flow network, F.

F captures the semantics of end-to-end communication between two hosts. For each conversation between two hosts in the computer network there exists one link between those two hosts in F. It is possible for there to be multiple links between two nodes just as there may be multiple conversations between two hosts in a computer network. Each link in F preserves the features of the respective conversation, including the headers, content, and timing information.

Figure 2.8 (taken from [4])represents a simple ACN with six hosts, two routers, and three types of links. The black boxes represent the routers, the open circles represent the hosts and the star represents a sensor. Each link type represents a separate component subnetwork. The most significant difference between the subnetworks is in the semantics of their links.

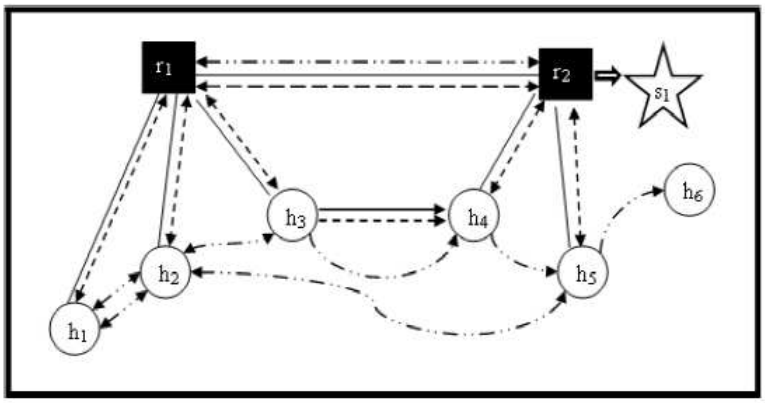

Figure 2.8: A sample ACN, [4]

The link in Figure 2.8 (taken from [4]) between s1 and r2 represents a spanning link. A sensor that sees all packets that pass through a router spans the router. Any router that is spanned by a sensor is a monitored router. A sensor has access to all features of a data flow link that passes through the monitored router it spans.

The worm propagation network is constructed to represent the worm spread across a network from infected host to infected host. Let (N, L) be the data flow network (F) of the ACN, (N,N $\times$ N). The worm propagation network, W = (I,Q), is a related to (N,L). I is the set of infected nodes in N. Q is a set of links, (a, b), where a worm on a infected b. For each link in Q, there is a sequence of links in L with the appropriate features to constitute the infection vector (IV); there may be multiple IV s in W.

The worm propagation network, W = (I,Q), is constructed from the communication network, (N, L). W is assumed to start with a set of initially infected nodes, I_{t_0} , and an empty set of links, Q_{t_0} , at time t = t_0. A node b and link (a, b) are added to I and Q, respectively, when an infected node a first propagates to it via an IV and successfully infects it. The infection time of a node is the time at which the first infection process completes or t0, if it is in I_{t_0} . Features of the links in L that constitute the IV preserve the links in Q. The start time of an infection link (a, b) in Q is the time when the communications that cause b to be infected are initiated by a and its end time is the infection time.

Figure 2.9 (taken from [4]) shows a worm with an IV that includes an ICMP echo request (followed by a reply) followed by a TCP connection to port 80. If either transaction in the IV is not completed or if the transactions are not in order, the infection does not spread. This worm has one infection vector, IV , which is necessary for propagation. It consists of:

- an ICMP echo request from a to b, followed by
- an ICMP echo reply from b to a, followed by

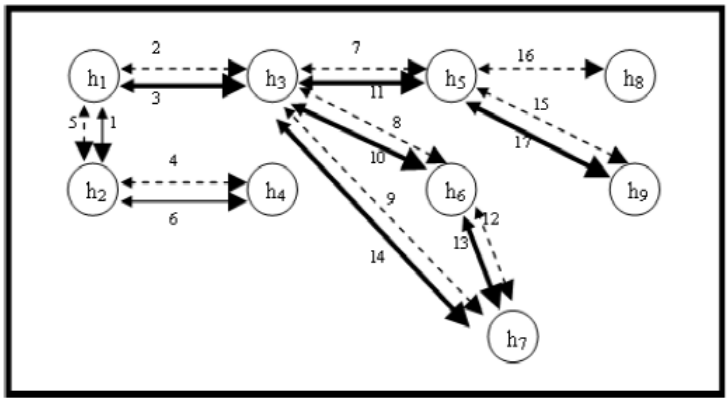

Figure 2.9: Simplified ACN showing worm spread, [4]

- a TCP SYN from a to b, followed by
- a TCP SYN/ACK from b to a, followed by
- an exploit and worm code from a to b.

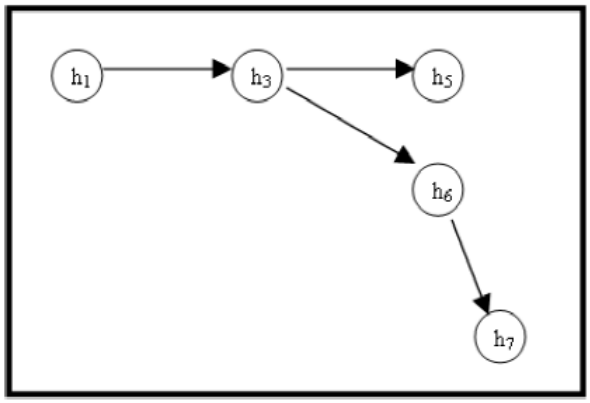

Figure 2.10: Worm propagation network of Figure 2.9, [4]

Figure 2.10 (taken from [4]) shows W constructed from the ACN shown in Figure 2.9 (taken from [4]) and the IV described above.A few of the nodes in Figure 2.9 are not in W. h_2 is not in W because the IV was not completed: the TCP port 80 data flow precedes the ICMP data flow, which does not constitute an IV . This indicates that non-worm traffic in the network may resemble worm traffic. h_4 is not in W because h_2 was not infected when they communicated, even though there were ICMP and TCP data flows. h_8 is not in W because the IV from h_5 is incomplete. The link (h_3, h_7) is not in W because h_7 was already infected (h_6 infected it first). Although the links

between h_5 and h_9 constitute a valid IV , h_9 is still not added to W. The reason in this case is due to the system conditions not being satisfied (e.g., h_9 is not vulnerable).

Ellis et. al. [4] presented three type of behavior signature for a worm as changes from server to client, α-in-α-out, and fan out. The other (descendant relations) is an inductive signature because it applies to a behavior that occurs across nodes. All of the behavioral signatures here are designed to answer the question "Is host x infected?" Although a descendant relation is an inductive signature, it is still measured with respect to a specific host.

In first type of signature When a server becomes infected with an active worm, the worm attempts to infect other hosts. To do this, it initiates connections to other services that the worm knows how to exploit. As it knows how to exploit the service whereby it infected the current host, it may use the same exploit to infect other hosts. When this happens the infected host changes in behavior; it acts as a server of a service until it is infected, after which it acts as a client of the service.

In second type of signature For a worm to spread, an infected host needs to complete a well-defined behavior−one of its (IV s)−with respect to a target host. The α-in-α-out signature is "a sends content to b that b later sends to c".

In third type of signature (Inductive signature) The descendant relation is used with idea as worms are fixed in the logic they execute and the number of infection vectors they employ. Consequently, as a worm spreads from an initially infected node, the number of descendants grows: the number of descendants at each depth grows as does the depth of the most distant descendant. For a worm to spread rapidly, each infection must infect more than one other host on average; that is, each worm must infect more than one other host, or it will fail to spread exponentially.

In third part of this mechanism NAAs concept has been introduced as An NAA is a description of how a specific enterprise distributes network applications across its network. This includes a description of where clients, servers, and peers of the various network applications are to be located across a network.the NAA significantly affects the sensitivity of behavioral signatures.

2.6.2.4 EarlyBird System

Singh et. al. [21] has developed Earlyword automated method for detecting new worms based on traffic characteristics common to most of them: highly repetitive packet content, an increasing population of sources generating infections and an increasing number of destinations being targeted.The system can be deployed at a vantage point in the network to scrutinize every packet that passes through it, and provide various levels of alerts to the user.

EarlyBird system firs computed a 64-bit signature using a combination of the contents of the packet payload and the destination port number for every packet passing through the vantage point the system is deployed.If the signature is found to occur more than $OccuranceRate$(threshold) number of times, than system instantiate counters to count the number of distinct sources, distinct destinations, and distinct source-destination pairs for the signature under consideration.the system generates alerts when:

1. Packets with similar content are being sent to a number of hosts (destination IP addresses).
2. Packets with similar content are being sent from a large number of hosts (source IP addresses).
3. Packets with similar content are being sent from a number of hosts to a large number of hosts (source-destination IP address pairs).

2.6.2.5 DSC(Destination-Source Correlation)

DSC [6] is a powerful two-phase local worm detection algorithm. Instead of only focusing on scanning symptoms, DSC aims to identify the worm victim behavior, based on both scanning pattern and infection pattern so that it can overcome some disadvantages of existing worm detection techniques. In the first phase of DSC, it correlate both incoming and outgoing packets to find infection pattern. Then in the second phase it checks for anomalous scanning patterns typical of worms. DSC is the first completely behavior-based model to detect worms.

DSC keep track of SYN and UDP traffic of source and destination, it record the address of the inside destination host and the scanning source from the monitored network. If a host get a packet on port i, then start sending packet destined for port i, a counter is incremented, when the counter reaches a certain threshold, it becomes suspect and alert is generated.

Detecting infection-like behavior in hosts requires keeping state. Hence Bloom Filter is used in DSC. Bloom filter is a bit vector, initially all zero. As entries are made to the filter, k independent functions are used to hash data into indicators, and the corresponding bit is set to one. To see if data has been entered into a Bloom filter, on merely performs the k hash operations on the data, and verifies all corresponding bits are set to one.

DSC use Bloom filter to store the destination addresses for scanning traffic (SYN and UDP) directed at the network. For scanning traffic originating form the network, it if the source host appears in the corresponding filter. If so, record this infection like behavior in a watch list. If the same host repeatedly sends out more packets and exceeds a trained threshold, it issue an alert.

After issuing an alert(i.e. finding an infection pattern) it checks for anomalous scannig pattern based on some predefined threshold to differentiate between normal communication outgoing connection and abnormal communication outgoing connection because there are some infection-like patterns in legitimate traffic, such as telnet/ssh chains and P2P connections.

The DSC algorithm is able to detect almost all types of scans, as long as the scan is frequent enough (based on the threshold) and the infection of the worm is targeting the same port. The major issue of DSC is that it can only capture scans from worms targeting the same port. It can not detect normal applications which can produce infection-like traffic and may not have an immediate stable scan rate. For example, hosts running gnutella may receive TCP/6346 traffic and also send data to other clients through TCP/6346 elsewhere. DSC has another limitation when it comes to bipartite, or dual worms (Worms that use two or more attack).

2.6.2.6 WDS using Sequential Hypothesis Testing and Credit Based approach

A worm infected host target to a specific port of the target or specific services that is run by target and due to this selective port strategy most of the time get rejected due to either many IPV4 address are dead (or not assigned to active host) or many Ipv4 addresses are protected by firewall as firewall block the port addresses or many active host may not running the target services. Author [19] implement Worm Detection System(WDS) with the concept of probability of successfull connectio of a benign host is higher than a malicious host, and for that WDS need FCC called as First Contact connection (FCC is a packet either TCP or UDP) send to a host with which the sender is contacting first time.

when running WDS we maintain a seperate state information of each host we track which host have been previously contacted by l (we manage Previous Contact Host,PCH set for each host) and also each local host l has a queue of FCC attempts that l has sent but still in pending state.

Algorithm states that

1. When WDS observe a packet(TCP SYN or UDP) sent by a local sender l, it will check wheather destination adress d is in l's previously contacted host(PCH) host or not if it isn't than WDS adds d to the PCH set and adds a new entry to the end of FCC queue with d as destination address and status PENDING.
2. When WDS observe an incoming packet addressed to local host l and source address is present inside l's FCC queue, packet is treated as response to the FCC request and status of FCC record is updated as SUCCESS unless the packet is TCP RST packet.
3. In case if FCC queue has status as PENDING and timeout occurs than FCC queue record will be updated as FAILURE.

Now WDS will calculate likelihood ratio for l based on status of each host present inside FCC queue of l and if likelihood ratio is greater threshold(predefined) than host l will be infected by worm otherwise it is an benign host.

As we know a worm possess autonomous behavior due to which a worm infected host can send thousand of connection request at a time, to limit the rate of issue of FCC request by a host WDS used Credit based approach for that WDS initially assign every host a credit value of 10 and when a host will send FCC reques creddit will be decreased by 1 and when a host received a SUCCESS status credit value will be increased by 2.

There are some issue related with this approach as may be Worm author has written the code in a manner like Worm will show benign behavior to gather more credit which will be used later to send more FCC request hence to avoid that every second credit value of a host is updated with maximum of 10 and 2/3rd of current credit.

Now second problem is that may be a benign host has sent 10 FCC request and currently it has 0 credit value so that it would get starve for a lack of first contact connection successes hence to get rid out of this problem WDS will allow a allowance of 1 credit to host in case if credit of that host will be 0 for 4 second because WDS can detect a Worm infected host within 3 second.

Chapter 3

Software Defined Network

3.1 Introduction

In today world of networking we are totally dependent of traditional network which is complex to manage. Software-Defined Network (SDN) has gained a lot of attention from academic researchers and industry in recent years, because it addresses the lack of programmability in existing networking architecture and enables easier and faster network innovation. Software-Defined Networking is a new networking architecture which has been proposed as a facilating technology for network evolution and network virtualization. Open Network Foundation is the main organization which has developed SDN, ONF is a non-profit consortium of network operators, service providers and vendors that promotes the SDN architecture and drives the standardization process of its major elements [17].

In [17] ONF defines SDN as a technology where "network control is decoupled from forwarding and is directly programmable". It concentrates the network intelligence in software-based central controllers, which aims to bring better and more efficient control, customizability and adaptability. As a result the networking devices are turned into packet forwarding equipment that can be programmed via an open interface.

Figure 3.1 (taken from [17]) represents SDN framework, which consists of three layers. The lowest layer is the infrastructure layer, also known as data plane or forwarding plane as it is mainly responsible for data forwarding as well as managing local information and gathering statistics. Next layer which is one above of data plane is known as control layer, also called as control plane or brain of controller as it is responsible for programming and managing the forwarding plane. It makes use of information provided by data plane to defines network operations and routing. It also compromises one or more controllers communicate with the forwarding network elements through standardized interface, referred as southbound interface.

OpenFlow, which is one of the mostly used southbound interfaces, mainly considers switches will be discussed later. The application layer can receive an abstracted and global view of the network from the controllers and use that information to provide appropriate guidance to the control layer. The interface between the application layer and the control layer is referred to as the northbound

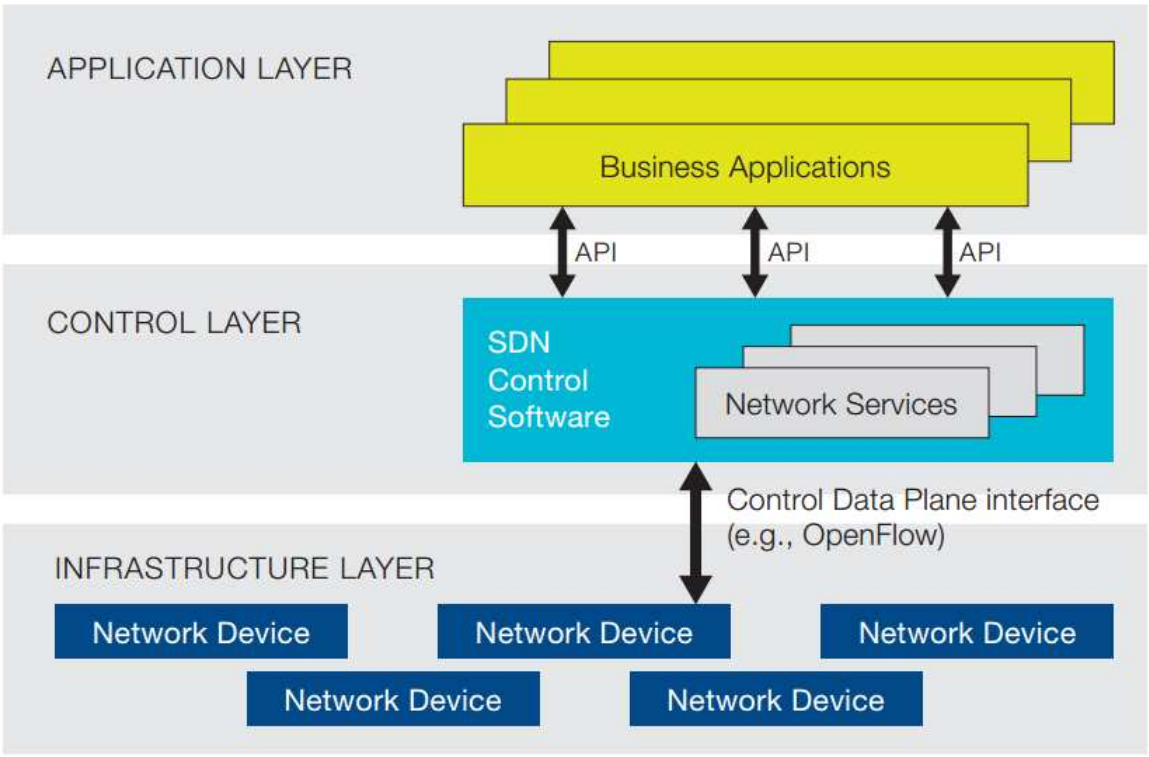

Figure 3.1: A three-tier architecture of Software Defined Networking, [17]

interface [11, 20].

The main benefits that the SDN technology might offer are listed in [17] and summarized below:

- Centralized unified control of network devices from different vendors
- Better automation and control, as an abstraction of the real network is created
- Simplified and quicker implementation of innovations, as the network control is centralized and there is no need every individual device to be reconfigured
- Improved network reliability and security, because of fewer configuration errors and unified policy enforcement, provided by the automated management and the centralized control
- Ability to easily adapt the network operation to changing user needs, as centralized network state information is available and can be exploited

3.1.1 Southbound Interface

The southbound interface is the connection between the SDN switch and the controller, the protocol used is commonly OpenFlow. This communication channel is necessary to provide a functioning SDN network. The integrity and authenticity of the data traffic must be ensured. An attacker able to manipulate the OpenFlow data sent to the switch could gain control over the entire network infrastructure. Hence, the southbound interface should use encryption and integrity protection. Since the controller and switch communicate via a TCP channel, the OpenFlow specification recommends TLS for securing the data traffic [7, 11].

One another option as Southbound interface is Forwarding and Control Element Separation (ForCES) [11] standardized by Internet Engineering Task Force (IETF) since 2003. ForCES is also a framework, not only a protocol; the ForCES framework also separates the control plane and data plane, but is considered more flexible and more powerful than OpenFlow. Forwarding devices are modeled using logical function blocks (LFB) that can be composed in a modular way to form complex forwarding mechanisms. Each LFB provides a given functionality, such as IP routing. The LFBs model a forwarding device and cooperate to form even more complex network devices. Control elements use the ForCES protocol to configure the interconnected LFBs to modify the behavior of the forwarding elements.

3.1.2 Northbound Interface

The northbound application programming interfaces (APIs) are open source-based, represent the software interfaces between the software modules of the controller platform and the SDN applications running atop the network platform it is also called Controller Switch interface [11]. Network applications writers use these APIs to expose universal network abstraction data models and functionality.

3.2 OpenFlow Protocol

OpenFlow is a standardized protocol that defines the communication between the control and the data forwarding plane in the SDN architecture. It moves the control out of the networking devices (routers, switches, etc.) into the centralized controller. The protocol uses the concept of flows that use match rules to determine how the packets will be handled. The protocol is configured on both sides - the device and the controller.

3.2.1 Switch Components

The forwarding device in an OpenFlow scenario is an OpenFlow switch. Each switch has an internal flow table, which is basically a set of rules that a switch is using to process incoming packets on the data plane. The number of flow tables varies by OpenFlow version. The flow table is managed by the controller, which can add, delete or modify flows. For this purpose, the controller and the switch are connected via a secure channel, usually TLS/SSL. This is shown in Figure 3.2 (taken

from [16]).

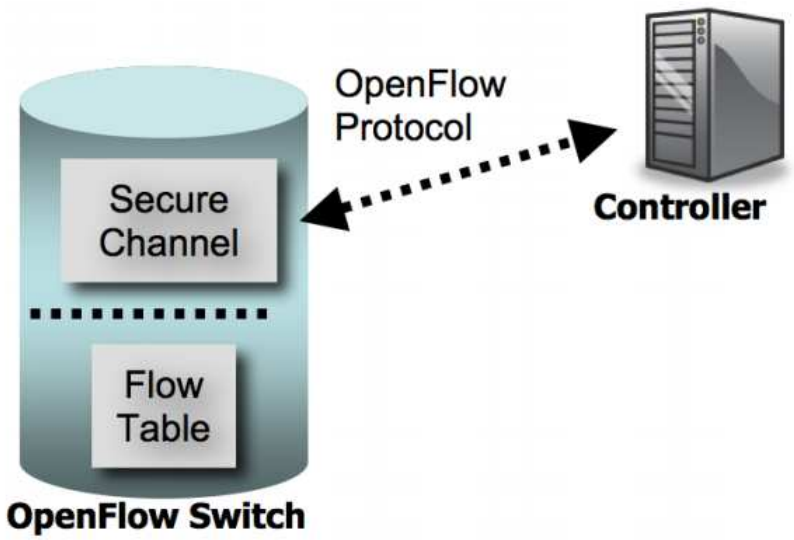

Figure 3.2: An OpenFlow switch communicates with a controller over a secure connection using the OpenFlow protocol, [16]

3.2.2 Flow Table

A flow table entry consists of the following components Figure 3.3 (taken from [16]):

Figure 3.3: A flow entry consists of header fields, counters, and actions., [16]

1. header fields that arriving packets are matched against (e.g. input port, layer 2 source and destination fields, IP addresses)
2. actions that should be applied to matching packets (e.g. output packet to physical port(s), send packet to controller or drop packet).
3. counters that are updated every time a packet matches (e.g. byte count, packet count)

Most commonly, incoming packet header fields will be matched against source and destination fields including MAC addresses, IP addresses or TCP ports. The OpenFlow specification (v 1.0) defines the following fields (Figure 3.4 [16]):

Ingress Port	Ether source	Ether dst	Ether type	VLAN id	VLAN pri-or-ity	IP src	IP dst	IP proto	IP ToS bits	TCP/ UDP src port	TCP/ UDP dst port

Figure 3.4: Fields from packets used to match against flow entries, [16]

Those fields either have values assigned or they can be wildcarded. Additionally, the flow entry contains a priority. If a packet matches more than one flow entry, "the entry that specifies an exact match (i.e., it has no wildcard) is always the highest priority" [16]. If the match for two entries is the same, then the entry with the highest priority is chosen. Besides installing flow table entries, there are other message types exchanged between the controller and the switch. If a switch receives a packet that does not match any entry in the flow table, the switch will send the packet to the controller as a PACKET_IN message. The controller can then decide how to handle this packet, e.g. by adding a new entry to the flow table.

3.2.3 Message Type

The OpenFlow specification defines three classes of messages: asynchronous, controller-to-switch and symmetric [16]. The following abstract is based on the OpenFlow switch specification version 1.0. There are three types of symmetric messages.

1. **HELLO** : messages are exchanged on connection startup
2. **ECHO_REQUEST and ECHO_REPLY** : messages are used to determine latency or bandwidth
3. **VENDOR** : messages are used to provide additional (vendor dependent) functionality.

Asynchronous messages are sent by the switch when a specific event occurs. There are four different types [16]:

1. **Packet-in** : if an arriving packet does not match any flow table entry (or if an entry exists with output port CONTROLLER), a PACKET_IN message is sent to the controller. The packet is buffered if the switch has enough memory and supports buffering. In that case a BUFFER_ID is set in the packet-in message. Otherwise, the full packet will be included.
2. **Flow-removed** : a flow can be removed for different reasons, i.e. a timeout (hard timeout or idle timeout) or the flow is explicitly removed by the controller. Switches do not always send FLOW_REMOVED messages by default. However, the controller can set the SEND_FLOW_REM flag in a FLOW_MOD message, forcing a switch to do so.
3. **Port-status** : the switch should send this messages on port configuration changes.
4. **Error** : the switch notifies the controller of errors.

Controller-to-switch messages are initiated by the controller and do not necessarily require an answer from the switch [16]. Six types are defined in the OpenFlow specification:

1. **Features** : on connection, the controller sends a FEATURE_REQUEST message, the switch must answer with a FEATURE_REPLY specifying its capabilities.
2. **Configuration** : the controller can set and request configuration parameters; the switch only responds to requests.
3. **Modify-State** : those messages are used by controller to create, modify or delete flows and to modify switch port properties.
4. **Read-State** : the controller can request statistics (i.e. byte count, packet count) from flow table, ports or flow table entries.
5. **Send-Packet** : the controller may send packets and order the switch to output them to a specific port.
6. **Barrier** : those messages are used to request notifications for completed operations.

3.2.4 Connection Setup

Switch establish the communication at a user-configurable IP address, using a specified port. When an OpenFlow connection is first established, each side of the connection must immediately send an OFTP_HELLO message with version field set as highest Openflow protocol version supported by sender. now receiver will calculate smaller version between it sent and it received. After getting negotiated version which is supported by recipient, then connection successfully established otherwise, the recipient reply with an OFTP_ERROR message with type field of OFPET_HELLO_FAILED, and then terminate the connection.

3.3 Controller

The controller is the most important part, responsible for maintaining all the network protocols and policies and distributing appropriate instructions to the network devices. An OpenFlow controller is responsible for determining how to handle packets without valid flow entries. It manages the switch flow table by adding and removing flow entries over the secure channel using the OpenFlow protocol. The openflow switches communicate with single or multiple controller. In Single Controller only one controller control the entire operation that is why it is also called single point of failure. Use of multiple controllers improves reliability incase if one controller fail than other can operate continue. SDN controller can be implemented in either centralized, distributed or multi-layer structure.

NOX was the first OpenFlow controller. It provides an interface for additional modules and is written in C++ and Python. The core components include topology discovery, layer 2 and layer 3 switching. The first version was published in 2007. Since 2008, NOX is open source. Newer versions of NOX are implemented exclusively in C++.

Controller	Language	Open Source	Developer	Status	Overview
NOX	Python/C++	Yes	Nicira	Not Active	The first OpenFlow controller written in Python and C++
POX	Python	Yes	Nicira	Active	General open-source controller written in python
Beacon	Java	Yes	Stanford	Not Active	A cross-platform, modular, Java-based OpenFlow controller that supports event based and threaded operations
Floodlight	Java	Yes	BigSwitch	Active	A Java-based OpenFlow controller (Supports v1.3), based on the beacon implementation, that works with physical and virtual switches
Ryu	Python	Yes	NTT,OSRG group	Active	An SDN operating system that aims to provide logically centralized controller and APIs that to create a new network management and control applications. Ryu fully supports OpenFlow v1.0, v1.2, v1.3 and Nicira extensions
OpenDaylight	Java	Yes	Linux Foundation	Active	A java based Open Flow Controller open to anyone, including end users and customers, and it provides a shared platform for those with SDN goals to work together to find new solutions.

Table 3.1: Table 3.1 List of Controllers

Table 3.1 shows list of some most popular controllers implemented on some specific languages:

Apart from tables Some innovative IT organization has also created its own controller like HP has created HP Virtual Application Networks SDN Controller [15], a unified control point in an OpenFlow-enabled network, simplifying management, provisioning, and orchestration. This enables delivery of a new generation of application-based network services. It also provides open application program interfaces (APIs) to allow third-party developers to deliver innovative solutions to dynamically link business requirements to network infrastructure via either custom Java programs or general-purpose RESTful control interfaces. The VAN SDN Controller is designed to operate in campus, data center, or service provider environments. It is an Extensible, scalable, resilient controller architecture.

Cisco has created Ciso One [13]. The Cisco ONE Controller was released in 2013, representing a major milestone in Cisco's execution and delivery of the ONE framework. It's a software package that acts as an interface between northbound and southbound APIs, and is made up of three components as part of a multi-faceted approach:

- **Controllers and agents** : a production-ready OpenFlow Controller and OpenFlow agent.
- **Programmatic Interfaces** : a strong collection of application programming interfaces (APIs) that are exposed directly on switches/routers to support existing OpenFlow requirements. AND
- **Virtual network overlays** : a suite of products that delivers virtual overlays, virtual services, and resource orchestration capabilities in the data center.

It supports the SDN-standard OpenFlow. Cisco ONE Controller customers can use the hundreds of APIs, in Cisco's One Platform Kit (onePK), to access the data inside their network. OnePK exposes existing features and capabilities within Cisco's switches and routers.

3.3.1 Pox Controller

In my research work i am working on Pox controller which is a controller based on Python. It is also called as NOX's successor, POX, Which was built as a friendlier alternative and has been used and implemented by a number of SDN developers and engineers. Compared to NOX, POX has an easier development environment to work with and well written APIs and documentation. It also provides a web based GUI. Some important features of POX controller are :

- Openflow interface based on Python
- Pox has reusable sample components for path selection, topology discovery etc.
- Pox runs anywhere and specifically targeting Linux, Mac Os and Windows
- Pox supports same GUI and Virtualization tools used by Nox
- Pox perform well compared to Nox.

3.4 SDN Advantage

The rapid development of new trends in computing, such as server virtualization, cloud services, immense variety of mobile data applications, etc., determine the need of new network architectures that can successful cope with the changing environment. Some of the key points summarized in [17] and listed below:

- **Changing traffic patterns** : a main characteristics is the shift from the traditional client-server applications, where one client accesses one server, to application where multiple databases and servers are accessed before the data is returned to the client.
- **Rise of the mobile data services** : the use of mobile devices (smartphones, notebooks, tablets, etc.) increases immensely. These devices have to be accommodates as a part of the corporate networks in order to meet compliance requirements and protect the privacy of corporate data and intellectual property.
- **Growth of cloud services** : the access on demand of applications, infrastructures or others IT resources introduces the need of scalability and adaptability of computing storage and network resources.
- **Big data processing** : presents the need for parallel processing on multiple servers, which need reliable connection to each other and high available bandwidth.

As a result the traditional network design struggle to meet the requirements of the rapidly changing demand patterns, because of the constraints of the current architectures, such as:

- **Static Nature** : the large number of the employed protocol, each of which meets specific needs of the network design, make changes in large networks rather troublesome. One of the primary reasons for this is the fact that the network management tools operate on device level and therefore any topology change triggers manual reconfiguration process of multiple devices. To avoid the chance of service interruption, such changes are usually avoided. This static nature of the contemporary network architectures severely contradicts to the dynamic nature of the traffic patterns.
- **Incoherent policies** : to implement a new consistent policy throughout the network may require reconfiguration of thousands of devices, which sometimes might be impossible. The incoherence might lead to security vulnerability, non-compliance with regulations, etc.
- **Scalability Limitations** : the huge rise of traffic demand might require changes in scale of the network architecture. The rise in the number of devices, however, highly increases complexity.
- **Vendor dependence** : network operators strive to meet customer requirements by constantly evolving the provided services and introducing high-customization. This might require specific configuration of the employed devices, which can be impossible because of the lack of open interfaces.

The existing incompliance between network capabilities and customer requirements determine the need for diverse solution. SDN architecture is being developed to meet the emerging conditions.

3.5 SDN Applications

The SDN architecture is claimed to greatly simplify network management and provide an immense number of new services via the programmable software modules. A summary of the application scenario that will benefit from employing the OpenFlow architecture are described in [7, 11] and briefly summarized below.

- **Enterprise networks** : the centralized control function of SDN can be particularly beneficial for enterprise networks in different ways. For example, network complexity can be reduced by removing middleboxes and configuring their functionality within the network controller. Different network functions implemented via SDN include NAT, firewalls, load balancers and network access.
- **Data centers** : power consumption management is a major issue in data centers, as they often operate below capacity in order to be able to meet peak demands. The deployment was motivated by the need of customized routing and traffic engineering, as well as scalability, fault tolerance and control that could not be achieved with traditional WAN networks.
- **Infrastructure-based wireless access networks** : an SDN solution for enterprise wireless LAN networks is proposed in [11]. The solution builds an abstraction of the access point infrastructure that separates the association state from the physical access point. The purpose is to ensure proactive mobility management and load balancing.

3.6 Security Solution using Software Define Networking

3.6.1 Anomaly Detection on SDN

Mehndi [9] has shown how four prominent traffic anomaly detection algorithms (Threshold Random Walk with Credit Based Rate Limiting, Rate limiting, Maximum Entropy detector and Netad) can be implemented in an SDN context using Openflow compliant switches and NOX as a controller. The basic idea in all of these controllers is to centralize the observation of network state, decide appropriate policies based on this state observation and then enforce these policies by installing flow entries in the switches. whenever a connection attempt succeeds. As a result, only the relevant packets (e.g. TCP SYNs and SYNACKs) within a connection are sent to the controller while all other packets are processed at line rate in the switch hardware.

3.6.2 HP Sentinel Security

HP has innovated Sentinel Security [18] application for HP Virtual Application Network to secure network including Campus, branches, Data center and Cloud Computing environments. Sentinel

application use the concept like redirection of DNS queries from client to the Sentinel application and check the hostname against its own threat dataset (HP Tipping Point DVLabs RepDV) containing over 700,000 known malware, spyware and botnet threats and prevent illegitimate access if a match is found.

3.6.3 defenceFlow

Redware [18] has designed defenceFlow to protect network against known and emerging network threats. defenceFlow support traditional NetFlow and Software defined network (SDN)/ OpenFlow networks, used to removing network and application DDoS attacks. It collect network traffic statistics based on OpenFlow and create normal traffic baseline using collected statistics and analyze the information and control the infrastructure layer.

3.6.4 Check Point

Check point [12] has unveiled a computer networking security architecture and methodology called Software-defined Protection that combines network security devices and defensive protection. An SDP infrastructure is designed to be modular, agile and most importantly secure, which is partitioned into three interconnected layers. Enforcement layer which inspects traffic and enforces protection within well defined network segments. The control layer is most important as it generates security policies and deploys those protection to enforcement points. The Management layer orchestrates the infrastructure and integrates security with business processes.

SDP [12] architecture supports traditional network security and access control policy for enterprises implementing technologies such as Software-defined Networking. It is also called Next Generation Threat Prevention which counter many known and unknown threats as it contains Integrated Intrusion Prevention System (IPS), Network based Anti-Virus, Threat Emulation and Anti-Bot. Check Point access control is based on next generation firewall which contains many security policy with capabilities likes Next Generation Firewall and VPN, User Identity Awareness, Application control and Data and Content Awareness. SDP use Data Loss Prevention (DLP) to Protect Data.

Chapter 4

Experimental Setup

4.1 Introduction

In this chapter, first of all we will describe the components which we have used to detect computer worm in our thesis work like OVS-Switch, POX-Controller, Mininet, TCPReplay, libpcap, Testbed setup to replay captured traffic using TCPReplay, dataset, than in experimental setup we will describes algorithm which we have used to detect computer worm infected host, Flow Chart follwed by Results.

4.2 Installation of Components

4.2.1 OVS-Switch

OpenvSwitch (OVS) is a , multilayer virtual switch licensed under the open source Apache 2.0 license and it has been designed to support massive network automation through programmatic extension. we have installed OVS because OVS is an open source software switch implementing OpenFlow and it runs in the kernel and user space in Linux. And we need all these OVS internal components and utilities in my experimental work. Process to install OVS-Switch are :

First,install dependencies for building OVS.

```
$ sudo apt-get install build-essential libssl-dev linux-headers-$(uname -r)
```

Build OVS from the source as follows. The steps below will build OVS user-space tools as well as its kernel module.

```
$ wget http://openvswitch.org/releases/openvswitch-1.9.3.tar.gz
$ tar xvfvz openvswitch-1.9.3.tar.gz
$ cd openvswitch-1.9.3
$ ./configure --with-linux=/lib/modules/‘uname -r‘/build
```

```
$ make
```

now install OVS user-space components under /usr/local/share/ using:

```
$ sudo make install
```

The next step is to test the OVS kernel module (before installing it).For that, load the kernel module in the kernel first.

```
$ sudo insmod ./datapath/linux/openvswitch.ko
```

now Verify that the OVS kernel module is loaded successfully.

```
$ lsmod | grep openvswitch
```

Once openvswitch.ko is successfully loaded in the kernel, now install the kernel module as follows.

```
$ sudo make modules_install
```

now Create a skeleton OVS configuration database.

```
$ sudo mkdir /etc/openvswitch
$ sudo ovsdb-tool create /etc/openvswitch/conf.db ./vswitchd/vswitch.ovsschema
```

start OVS database server using:

```
$ sudo ovsdb-server --remote=punix:/usr/local/var/run/openvswitch/db.sock
- remote=db:Open_vSwitch,manager_options --pidfile --detach
```

Initialize OVS configuration database.

```
$ sudo ovs-vsctl --no-wait init
```

Start OVS daemon.

```
$ sudo ovs-vswitchd --pidfile --detach
```

4.2.2 POX Controller

POX is a Python-based SDN controller. POX can be installed from GitHub quickly through the command:

```
$ git clone http://github.com/noxrepo/pox
$ cd pox
```

POX is invoked by running **./pox.py** followed by the module name in the format **directory.filename.** if we write

```
$ ./pox.py forwarding.l2_learning
```

than Pox controller will goes up on default port with port no 6633
and if we want to run Pox controller in some other port number than we have to use openflow.of 01 –port=port number as

```
$ ./pox.py forwarding.l2_learning openflow.of 01 --port=6655
```

4.2.3 Mininet Setup

Basically TRW-CB algorithm required to process only TCP/UDP packets and for which i am using TCPReplay to replay captured dataset but still i am implementing Mininet to Ping between host and get a result that TRW-CB algorithm is rejecting all packets other than IP(TCP/UDP).

Mininet is a network emulator which creates a network of virtual hosts, switches, controllers, and links. Mininet hosts run standard Linux network software, and its switches support OpenFlow for highly flexible custom routing and Software-Defined Networking.

Mininet supports research, development, learning, prototyping, testing, debugging, and any other tasks that could benefit from having a complete experimental network on a laptop or other PC.

Mininet provides a simple and inexpensive network testbed for developing OpenFlow applications. It Enables multiple concurrent developers to work independently on the same topology and Supports system-level regression tests, which are repeatable and easily packaged it also supports arbitrary custom topologies, and includes a basic set of parametrized topologies. Mininet Provides a straightforward and extensible Python API for network creation and experimentation. Mininet networks run real code including standard Unix/Linux network applications as well as the real Linux kernel and network stack. Mininet can be easily installed from github by following commands.

```
$ sudo apt-get install git
$ git clone git://github.com/mininet/mininet
```

We can easily create our custom topology in mininet. A custom topology(named as custom topo.py here and asaved under /home/user/mininet/mininet/custom directory) can be run in mininet using the following command.

```
$ sudo mn -custom /mininet/custom/custom topo.py -topo mytopo -mac -controller,
remote port=5678
```

Here remote and port is optional. Remote is used to specify that the controller is a remote controller i.e. the POX controller. And port is used to mention through which port the controller

should be communicated while it is not running in default port 6633.

4.2.4 libpcap and TCPReplay

libpcap is a system-independent interface for user-level packet capture. libpcap provides a portable framework for low-level network monitoring. Applications include network statistics collection, security monitoring, network debugging, etc.

Tcpreplay is a suite of BSD GPLv3 licensed tools written by Aaron Turner for UNIX operating system using which we can replay previously captured traffic which is in libpcap format to test a variety of network devices. using TCPReplay we can classify our traffic as client or server rewrite headers of layer 2,3 and 4 and finally replay the traffic onto network and through other devices such as switches, routers, firewalls, NIDS and IPS's.

before installing TCPReplay we have to install libpcap as captured traffic is in libpcap format.Hence to install libpcap first we have to download .tar.gz files of libpcap from
`http://www.tcpdump.org/ release/libpcap-1.7.4.tar.gz`
and than unzip the file and perform setup operation mentioned below:

- `$ tar -zxvf libpcap-1.7.4.tar.gz`
- `$ cd libpcap-1.7.4`
- `$ ./configure`
- `$ make`
- `$ make install`

now libpcap is installed in our system, before installing TCPReplay there are some dependencies file to support TCPReplay are required, which i have installed using:

```
$ sudo apt-get install flex
$ sudo apt-get install byaac
```

now download latest version of TCPReplay .tar.gz file from
`https://github.com/appneta/tcpreplay/releases/ download/v4.1.1 /tcpreplay-4.1.1.tar.gz`
than follow the process given below:

- `$ tar -zxvf tcpreplay-4.1.1.tar.gz`
- `$ sudo apt-get install build-essential libpcap-dev`
- `$ cd tcpreplay-4.1.1`
- `$ ./congigure`
- `$ make`
- `$ sudo make install`
- `$ sudo make test`

4.3 Threshold Random Walk with Credit-based Rate Limiting TRW-CB

The TRW-CB [9] algorithm detects the Worm infected host and SYN Flooding from a host by using the fact that the probability of a connection being a success should be much higher for a benign host than for a malicious one. This can be achieved by performing the likelihood ratio test. For each internal host the algorithm maintains a queue for connection initiations (TCP SYNs) also called as FCC(First Contact Connection) queue which are yet to receive a response (SYN-ACK) and a set of Previously contacted host as PCH. When the connection receives a TCP RST or a connection timeout the connection dequeues it from the queue and increases the likelihood ratio of the host, which initiated the connection. When there is a successful connection the likelihood ratio is decreased. When the likelihood ratio for a particular host reaches a certain threshold then it is declared as infected.

4.3.1 Worm Detection with TRW-CB on the POX Controller

The TRW-CB [Threshold Random Walk with Credit-based Rate Limiting] [9] is implemented on the POX controller to detect Computer worm and prevent denial of service (DOS)attack with SYN Flooding.The algorithm either uses the connection initiations or replies to them . According to OpenFlow 1.0 any new packet, which does not match a flow entry in the switch, is forwarded to the controller. The TRW-CB instance on the controller monitors the packets if the connection is successful then the flows are established in the switch to forward the rest of the packets in that session. Consider a internal host Host1 connected to an OpenFlow switch with a POX controller and a TRW-CB instance is executing on the controller. Host 2 is a external webserver (as shown in Fig 4.1).

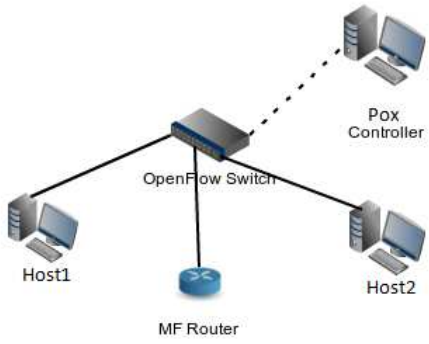

Figure 4.1: Two hosts connected with Switch and controller

1. Host1 sends a TCP- SYN or first UDP packet to a external host Host2. The packet reaches the switch, since there are no matching flows in the flow table the packets are forwarded to the controller.(shown in fig 4.2)

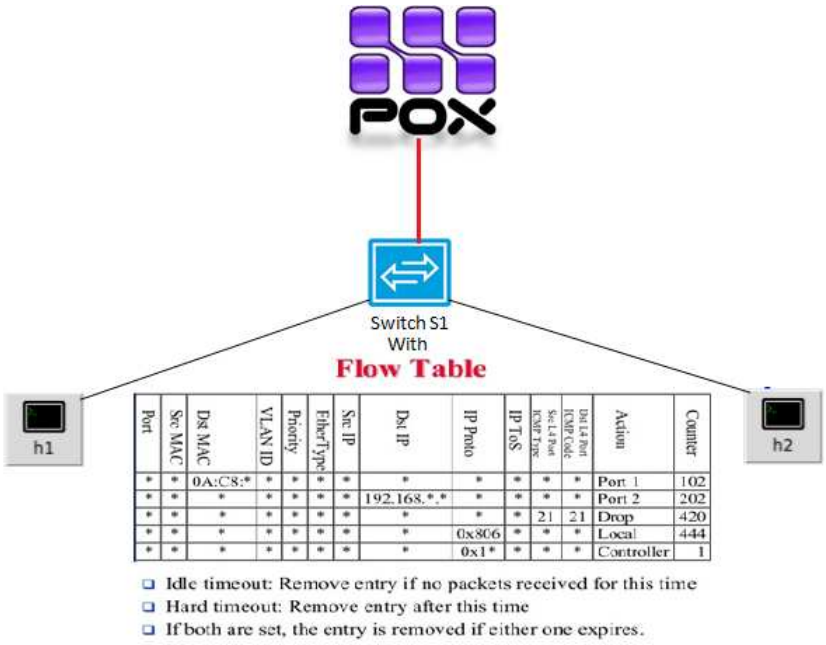

Figure 4.2: Packet forwarded to Controller if no matching flows found

2. **Phase1** : The TRW-CB on the POX controller perform following operations (shown in Fig 4.3)
   - Add Host2 to list of hosts (PCH) contacted by Host1
   - Add connection request to FCC queue of Host 1 with PENDING status

   and than forwards this packet through the switch without setting flows.

3. **Phase2** : The external host Host2 can respond in two ways with TCP SYN-ACK/UDP packet or OTHERS(FIN,RST etc.)/Timeout
   - **TCP SYN-ACK/UDP**: When a SYN-ACK/UDP is received from Host2 to Host1 the switch again forwards to the controller due to no matching flows in the flow table. The TRW-CB instance on the controller perform following operations (shown in fig 4.4)
     - Remove address of Host2 from connection request queue (FCC queue) of Host1
     - Decrease likelihood ratio for Host1
   - **Timeout or Others(RST)** : When a connection times out or RST packets occurs from Host2 to Host1 the TRW-CB does not install any flow entries in the switch and increases the likelihood ratio for Host1.

   and than installs two flows into the switch, one from Host1 to Host2 and the other from Host2 to Host1.

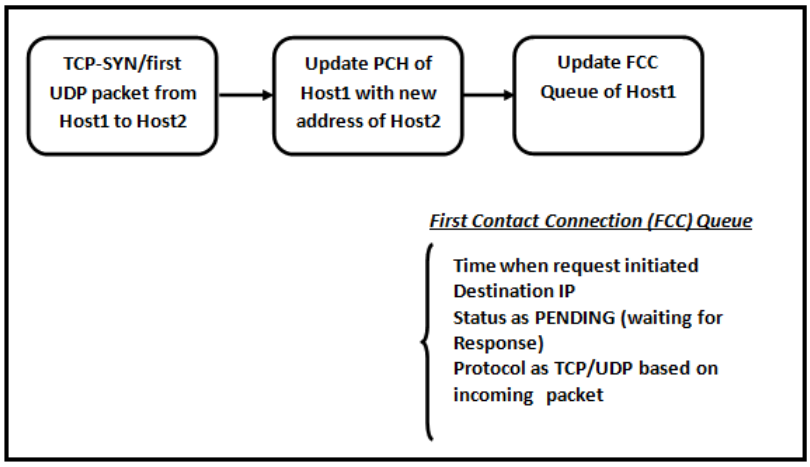

Figure 4.3: Flow Chart of Phase1 Operation of TRW-CB

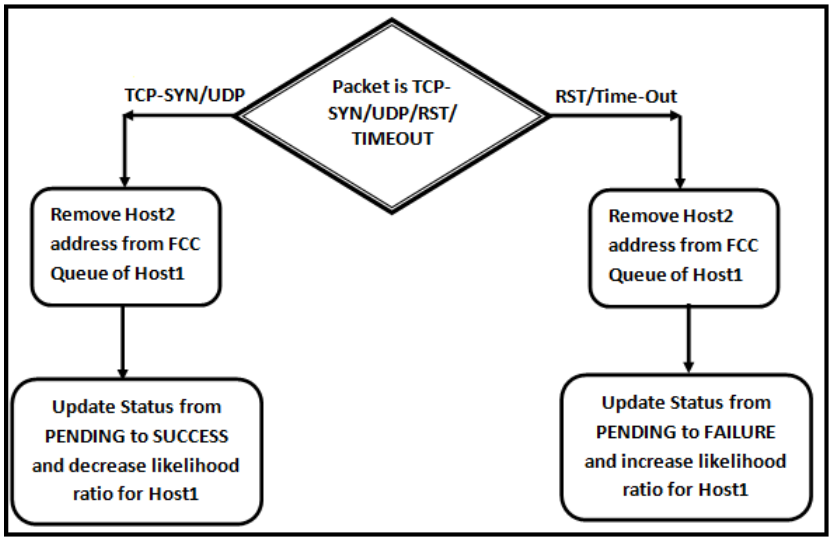

Figure 4.4: Flow Chart of Phase2 Operation of TRW-CB

When the likelihood ratio for a particular host reaches a threshold limit, the host is declared as infected otherwise benign.

4.4 DataSet Description

In order to effectively evaluate our algorithm we need some captured traffic which contains both benign as well as attack traffic. benign traffic capturing was not an issue but the problem was to get Worm infected traffic or traffic generated by Computer worm infected host and hence we have used captured traffic used by [9] in his research work which contains benign as well as attack traffic also. Attack traffic contains TCP SYN and UDP flood packets.

4.4.1 Testbed Setup

For accuracy evaluation, I have created a test bed consists of one host machine which ran OVS switch which implements Openflow protocol. A Pox controller was located on the same machine and communicated with the OpenvSwitch daemon through a Unix domain socket. Another machine running in Ubuntu VMs on same host machine which was used to replay traffic from our merged datasets and send it to our server. Beside this mininet is installed in one of the host to test the TRW-CB module in a large logical network using the POX controller as remote controller (shown in Fig 4.5).

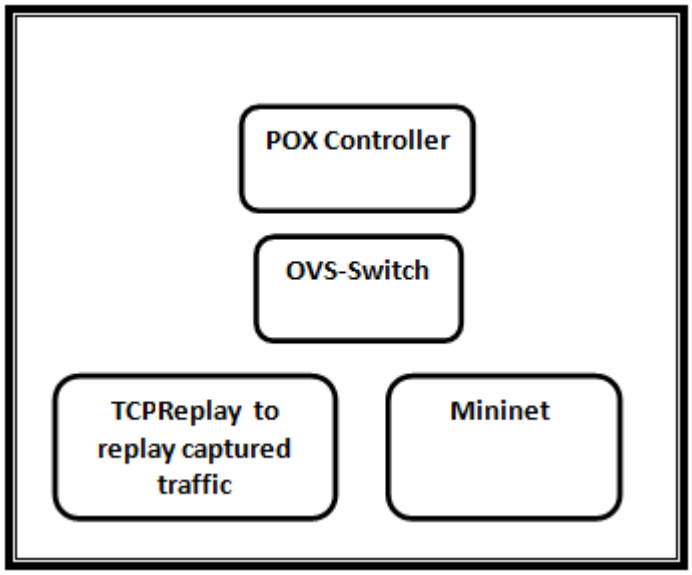

Figure 4.5: TestBed Setup: A host containing Pox,Ovs,mininet and VM machine

4.5 Flow Chart

The flow chart below explains my implementation of the TRW-CB algorithm. It shows what happens to the packet from the moment it arrives at the switch port to the moment it leaves the

switch to its final destination as figures 4.6 illustrates. When a switch send a packet to the controller due to absence of flow rule controller will check wheather it is Ip packet or not if it is IP packet than it will be checked that source is local to Subnet or not once verified Payload type will be examined if controller get TCP/UDP packet than forward to next step where packet payload will be checked wheather it is TCP SYN (or first udp packet from sender to destination) if yes than POX controller will perform phase1 of TRW-CB algorithm and instruct switch to send packet without installing any flow rules. up to payload type pox controller us checking 3 times for (IP packet type,Local or not and Payload type), at any stage if condition will be false based on packet information packet will be dropped by POX controller.

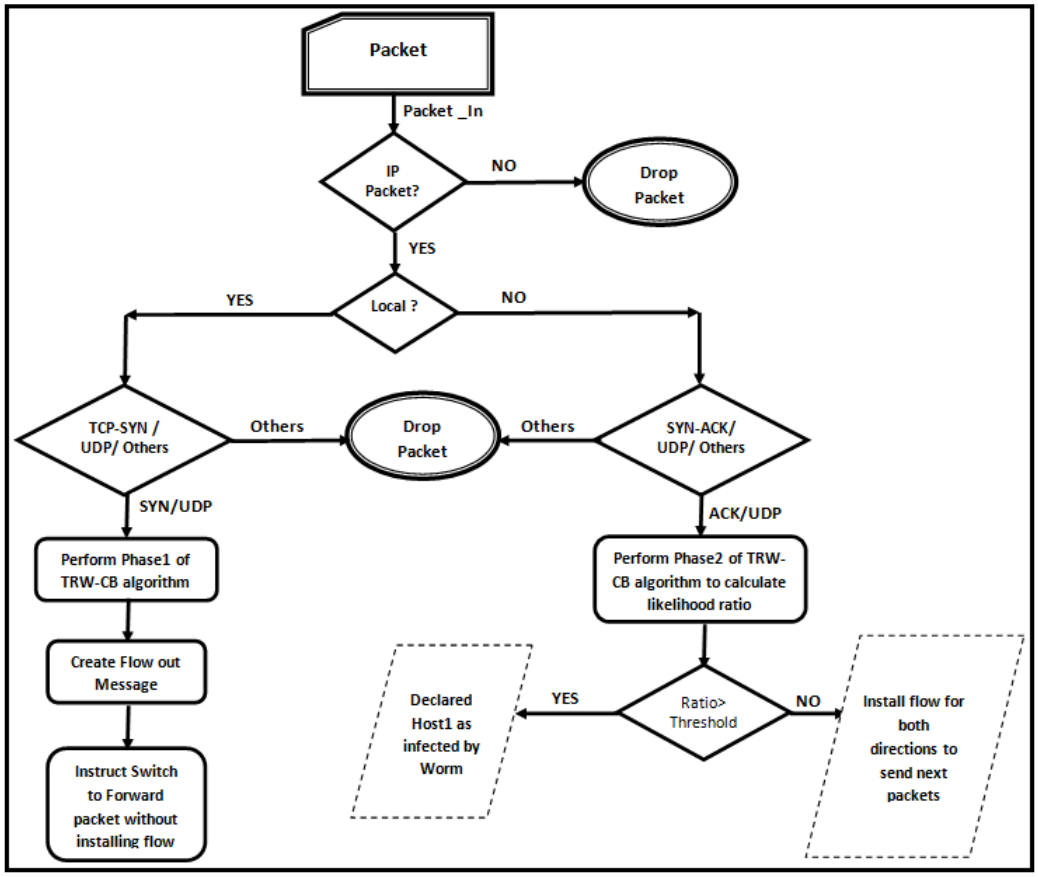

Figure 4.6: Flow Chart of Packet_In events handling

If packet is SYN-ACK or others type than POX controller will perform phase1 of TRW-CB algorithm and than calculate likelihood ratio and finally compare likelihood ratio with predefined threshold vale to determine wheather host is worm infected or not? if likelihood ratio will be greater than sender is declared as worm infected and drop packets else install flow rules both direction to send next coming packets from same sender to same receiver.

4.6 Result

First We have check whether TRW-CB is rejecting non-IP packets or not and for that we have created a mininet with 2 host as h1(host1) and h2(host2) with IP addresses 10.0.0.1 and 10.0.0.2 correspondingly, connected by one switch and a controller in which TRW-CB algorithm is running(shown in Fig 4.7).

```
cdcju@sdn1-PC:~$ sudo mn --custom ~/mininet/custom/3s4h.py --controller remote,p
ort=6655
[sudo] password for cdcju:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
From 10.0.0.1 icmp_seq=4 Destination Host Unreachable
From 10.0.0.1 icmp_seq=5 Destination Host Unreachable
From 10.0.0.1 icmp_seq=6 Destination Host Unreachable
From 10.0.0.1 icmp_seq=7 Destination Host Unreachable
From 10.0.0.1 icmp_seq=8 Destination Host Unreachable
From 10.0.0.1 icmp_seq=9 Destination Host Unreachable
^C
--- 10.0.0.2 ping statistics ---
10 packets transmitted, 0 received, +9 errors, 100% packet loss, time 9049ms
pipe 3
mininet>
```

Figure 4.7: ping between hosts in mininet, where host is unreachable as TRW-CB rejecting them

now when we open another terminal where TRW-CB algorithm running in Pox controller, we can see clearly that our algorithm is rejecting ARP packets received from 10.0.0.1 which is IP address of h1 pinging to h2 with IP 10.0.0.2(shown in fig 4.8)

we have seen that for ARP packets our algorithm is working fine now we have to check for other packets like UDP/TCP/Non-IP packets other than TCP/UDP (there are too many packets which is IP type but for this algorithm we have considered only UDP/TCP as IP type rests are of Non-IP type). First I have configured OVS switch according to our need and connected POX controller to the (switch shown in Fig 4.9).

```
cdcju@sdn1-PC:~$ cd pox
cdcju@sdn1-PC:~/pox$ ./pox.py log.level --DEBUG misc.TRW  openflow.of_01 --port=6655
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
DEBUG:misc.TRW:Enabling Threshold Random-Walk algorithm
DEBUG:core:POX 0.2.0 (carp) going up...
DEBUG:core:Running on CPython (2.7.6/Jun 22 2015 18:00:18)
DEBUG:core:Platform is Linux-3.13.0-85-generic-i686-with-Ubuntu-14.04-trusty
INFO:core:POX 0.2.0 (carp) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6655
INFO:openflow.of_01:[00-00-00-00-00-03 1] connected
DEBUG:misc.TRW:Connection [00-00-00-00-00-03 1]
INFO:openflow.of_01:[00-00-00-00-00-02 2] connected
DEBUG:misc.TRW:Connection [00-00-00-00-00-02 2]
INFO:openflow.of_01:[None 3] closed
INFO:openflow.of_01:[00-00-00-00-00-01 4] connected
DEBUG:misc.TRW:Connection [00-00-00-00-00-01 4]
DEBUG:misc.TRW:NON IP / ARP packet received
DEBUG:misc.TRW:NON IP / ARP packet received
DEBUG:misc.TRW:NON IP / ARP packet received
DEBUG:misc.TRW:NON IP / ARP packet received
DEBUG:misc.TRW:NON IP / ARP packet received
DEBUG:misc.TRW:NON IP / ARP packet received
DEBUG:misc.TRW:NON IP / ARP packet received
DEBUG:misc.TRW:NON IP / ARP packet received
DEBUG:misc.TRW:NON IP / ARP packet received
DEBUG:misc.TRW:NON IP / ARP packet received
DEBUG:misc.TRW:NON IP / ARP packet received
DEBUG:misc.TRW:NON IP / ARP packet received
DEBUG:misc.TRW:NON IP / ARP packet received
DEBUG:misc.TRW:NON IP / ARP packet received
DEBUG:misc.TRW:ARP packet from 10.0.0.1 dropped
DEBUG:misc.TRW:ARP packet from 10.0.0.1 dropped
DEBUG:misc.TRW:ARP packet from 10.0.0.1 dropped
DEBUG:misc.TRW:NON IP / ARP packet received
DEBUG:misc.TRW:NON IP / ARP packet received
DEBUG:misc.TRW:NON IP / ARP packet received
DEBUG:misc.TRW:ARP packet from 10.0.0.1 dropped
DEBUG:misc.TRW:ARP packet from 10.0.0.1 dropped
DEBUG:misc.TRW:ARP packet from 10.0.0.1 dropped
DEBUG:misc.TRW:ARP packet from 10.0.0.1 dropped
```

Figure 4.8: TRW-CB in Pox controller rejecting non-IP/ARP packets received from mininet

now replay traffic using TCPReplay from another terminal and send it to OVS switch (shown in Fig 4.10).

after than when controller receives any packets based on their type TRW-CB perform actions and forward/reject packets or detect host showing worm infected traffic (shown in Fig 4.11).

```
cdcju@sdn3-pc:~$ sudo ovs-vsctl show
[sudo] password for cdcju:
9cdbf4c6-08bb-489f-ae6e-53ffb63cd279
    Bridge "br1"
        Controller "tcp:0.0.0.0:6660"
        Port "br1"
            Interface "br1"
                type: internal
        Port "vnet0"
            Interface "vnet0"
        Port "eth0"
            Interface "eth0"
    ovs_version: "2.0.2"
cdcju@sdn3-pc:~$
```

Figure 4.9: OVS-Switch configuration, where POX controller is running

```
cdcju@sdn3-pc:~$ sudo tcpreplay -i  eth0 -t -K udpFlood_1_1_1synper10sec.pcap
File Cache is enabled
Actual: 30 packets (16620 bytes) sent in 0.000076 seconds.
Rated: 218684200.0 Bps, 1749.47 Mbps, 394736.84 pps
Flows: 30 flows, 394736.84 fps, 30 flow packets, 0 non-flow
Statistics for network device: eth0
        Successful packets:        30
        Failed packets:            0
        Truncated packets:         0
        Retried packets (ENOBUFS): 0
        Retried packets (EAGAIN):  0
cdcju@sdn3-pc:~$
```

Figure 4.10: Replay captured traffic using TCPReplay

```
cdcju@sdn3-pc:~/pox$ ./pox.py forwarding.TRW log.level --DEBUG openflow.of_01 --port=6660
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
DEBUG:core:POX 0.2.0 (carp) going up...
DEBUG:core:Running on CPython (2.7.6/Jun 22 2015 18:00:18)
DEBUG:core:Platform is Linux-3.13.0-85-generic-i686-with-Ubuntu-14.04-trusty
INFO:core:POX 0.2.0 (carp) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6660
INFO:openflow.of_01:[00-27-0e-18-9c-08 1] connected
DEBUG:forwarding.TRW:Connection [00-27-0e-18-9c-08 1]
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 74 < 99
DEBUG:forwarding.TRW:NON-IP packet received
DEBUG:forwarding.TRW:NON-IP packet from 68:5d:43:53:ac:b4 dropped
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 74 < 99
DEBUG:forwarding.TRW:NON-IP packet received
DEBUG:forwarding.TRW:NON-IP packet from 68:5d:43:53:ac:b4 dropped
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 74 < 99
DEBUG:forwarding.TRW:NON-IP packet received
DEBUG:forwarding.TRW:NON-IP packet from 68:5d:43:53:ac:b4 dropped
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 94 < 185
INFO:forwarding.TRW:Packet coming from 68:5d:43:53:ac:b4 is IP type
INFO:forwarding.TRW:Please Wait:Checking wheather source is Local or not
INFO:forwarding.TRW:Sorce 10.0.0.6 is internal
INFO:forwarding.TRW: UDP packet coming from internal host
INFO:forwarding.TRW:packet coming from 10.0.0.6 to 10.0.0.255
INFO:forwarding.TRW:local host 10.0.0.6 is sending for the firsr time
INFO:forwarding.TRW:Local host with IP 10.0.0.6
INFO:forwarding.TRW:new host 10.0.0.255 is contacting for the firsr time
INFO:forwarding.TRW:This host 10.0.0.6 has already exceeded its unAcked connection limit.
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 74 < 99
DEBUG:forwarding.TRW:NON-IP packet received
DEBUG:forwarding.TRW:NON-IP packet from 68:5d:43:53:ac:b4 dropped
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 74 < 99
DEBUG:forwarding.TRW:NON-IP packet received
DEBUG:forwarding.TRW:NON-IP packet from 68:5d:43:53:ac:b4 dropped
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 74 < 99
DEBUG:forwarding.TRW:NON-IP packet received
DEBUG:forwarding.TRW:NON-IP packet from 68:5d:43:53:ac:b4 dropped
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 94 < 185
INFO:forwarding.TRW:Packet coming from 68:5d:43:53:ac:b4 is IP type
INFO:forwarding.TRW:Please Wait:Checking wheather source is Local or not
INFO:forwarding.TRW:Sorce 10.0.0.6 is internal
INFO:forwarding.TRW: UDP packet coming from internal host
INFO:forwarding.TRW:packet coming from 10.0.0.6 to 10.0.0.255
^CINFO:core:Going down...
INFO:openflow.of_01:[00-27-0e-18-9c-08 1] disconnected
```

Figure 4.11: Packet_In events handled by POX controller

Chapter 5

Conclusion

5.1 Conclusion

This thesis is divided into three parts ,where in first two parts we did a survey of i) computer worm including different life phase an active worm possess, categorization of different type of computer worm according to their behavior either good or bad, different worm propagation model developed by researchers followed by different type of propagation prevention and detection mechanism. ii.) a review into Software Defined Networking as new networking paradigm with its interfaces and Openflow protocol, different type of controllers (may be active or not in current), SDN advantages over traditional network and different security solutions developed by researchers and different IT organizations.
Finally we conclude this thesis with last part as iii) experimental setup with explanation of TRW-CB algorithm to detect computer worm in Software-Defined network using POX controller. Different type of Worm detection algorithm has been developed in traditional network by researchers. As SDN is called future of networking we have to make SDN more secure after learning different propagation mechanism used by computer worm in traditional network.
In this thesis, our work demonstrate that Software Defined Networks using Openflow and POX controller rejects packets other than TCP/UDP as worm transfer their payload using TCP/UDP packets only.

5.2 Future Work

In this thesis, some issues has been solved but still there are some problems we have to sort out to detect any type of worm (known worm or zero-day worm), based on behavior of computer worm. As a future work, we can do different research related to developing Worm detection system in SDN like :

1. Different Worm use different worm propagation methods using those propagation worm we have to create one strong algorithm to detect computer worm. Some worms spread itself too fast whether some are too slow and developing one algorithm to detect both fast as well as slow worm is a challenge for researchers.

2. make detection more accurate in any network (Home, Office or ISP).

3. after detecting a computer worm, TRW-CB will communicate to ISP where malicious traffic will be inspect by human operator to verify the existence of an attack.

4. it could be used to correlate threats from multiple home networks for detecting global network security problems e.g. botnet.

Bibliography

[1] Xuan Chen and John Heidemann. Detecting early worm propagation through packet matching. https://www.isi.edu/ johnh/PAPERS/Chen04a.pdf, 2004.

[2] Zesheng Chen, Lixin Gao, and K. Kwiaty. Modeling the spread of active worms. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 3, pages 1890–1900 vol.3, 2003.

[3] S. Coursen. Good viruses have a future. http://www.securityfocus.com/columnists/23, 2015.

[4] Daniel R. Ellis, John G. Aiken, Kira S. Attwood, and Scott D. Tenaglia. A behavioral approach to worm detection. In *Proceedings of the 2004 ACM Workshop on Rapid Malcode*, WORM '04, pages 43–53. ACM, 2004.

[5] M.R. Faghani and Uyen Trang Nguyen. A study of xss worm propagation and detection mechanisms in online social networks. *Information Forensics and Security, IEEE Transactions on*, 8(11):1815–1826, 2013.

[6] Guofei Gu, M. Sharif, Xinzhou Qin, D. Dagon, Wenke Lee, and G. Riley. Worm detection, early warning and response based on local victim information. In *Computer Security Applications Conference, 2004. 20th Annual*, pages 136–145, 2004.

[7] Manar Jammal, Taranpreet Singh, Abdallah Shami, Rasool Asal, and Yiming Li. Software defined networking: State of the art and research challenges. *Computer Networks*, 72:74 – 98, 2014.

[8] Pele Li, M. Salour, and Xiao Su. A survey of internet worm detection and containment. *Communications Surveys Tutorials, IEEE*, 10(1):20–35, 2008.

[9] Syed Akbar Mehdi, Junaid Khalid, and Syed Ali Khayam. Revisiting traffic anomaly detection using software defined networking. In *Proceedings of the 14th International Conference on Recent Advances in Intrusion Detection*, RAID'11, pages 161–180, Berlin, Heidelberg, 2011. Springer-Verlag.

[10] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the slammer worm. *Security Privacy, IEEE*, 1(4):33–39, 2003.

[11] B. A. A. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka, and T. Turletti. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys Tutorials*, 16(3):1617–1634, Third 2014.

[12] Check Point White Paper. Software-defined protection enterprise security blueprint. https://www.checkpoint.com/downloads/product-related/Ebook/Software-defined%

[13] CISCO White Paper. Software-defined networking:why we like it and how we are building on it. http://www.cisco.com/c/dam/en_us/solutions/industries/docs/gov/cis13090_sdn_sled_white_paper.pdf.

[14] Glenn Gebhart Sans Institute White Paper. Worm propagation and countermeasure. https://www.sans.org/reading-room/whitepapers/malicious/worm-propagation-countermeasures-1410, 2004.

[15] H. P. Technical White Paper. Realizing the power of sdn with hp virtual application networks. http://h17007.www1.hp.com/docs/interopny/4AA4-3871ENW.pdf.

[16] Open Networking Foundation White Paper. Openflow switch specification version 1.0.0 (wire protocol 0x01). http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf, 2009.

[17] Open Networking Foundation White Paper. Software-defined networking:the new norm for networks. https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf, 2012.

[18] Radware White paper. Defenseflow:the sdn application that programs networks for dos security. http://www.infinigate.co.uk/fileadmin/user_upload/UK/Company/Radware/pdfs/defenseflow-sdn-application-that-programs-networks-for-dos-security-solution-brief.pdf.

[19] Stuart E. Schechter, Jaeyeon Jung, and Arthur W. Berger. Fast detection of scanning worm infections. In *Recent Advances in Intrusion Detection: 7th International Symposium, RAID 2004, Sophia Antipolis, France, September 15-17, 2004. Proceedings*, pages 59–81, 2004.

[20] Adib Habbal Shivaleela Arlimatti, Suhaidi Hassan and Suki Arif. Software defined network and openflow: A critical review. *ARPN Journal of Engineering and Applied Sciences*, 10(3), 2005.

[21] Sumeet Singh, Cristian Estan, George Varghese, and Stefan Savage. The earlybird system for real-time detection of unknown worms. Technical Report CS2003-0761, University of California, San Diego, August 2003.

[22] Craig Smith, Ashraf Matrawy, Stanley Chow, and Bassem Abdelaziz. Computer worms: Architectures, evasion strategies, and detection mechanisms, 2009.

[23] Stuart Staniford, Vern Paxson, and Nicholas Weaver. How to own the internet in your spare time. In *Proceedings of the 11th USENIX Security Symposium*, pages 149–167. USENIX Association, 2002.

[24] Yini Wang, Sheng Wen, Yang Xiang, and Wanlei Zhou. Modeling the propagation of worms in networks: A survey. *Communications Surveys Tutorials, IEEE*, 16(2):942–960, 2014.

[25] Nicholas Weaver, Vern Paxson, Stuart Staniford, and Robert Cunningham. A taxonomy of computer worms. In *Proceedings of the 2003 ACM Workshop on Rapid Malcode*, WORM '03, pages 11–18. ACM, 2003.

[26] Wikipedia. Computer worm — wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Computer_worm, 2015.

[27] Cliff Changchun Zou, Weibo Gong, and Don Towsley. Code red worm propagation modeling and analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, pages 138–147. ACM, 2002.

[28] Cliff Changchun Zou, Weibo Gong, and Don Towsley. Worm propagation modeling and analysis under dynamic quarantine defense. In *Proceedings of the 2003 ACM Workshop on Rapid Malcode*, WORM '03, pages 51–60. ACM, 2003.