Performance Analysis of Peer-To-Peer Botnets using "The Storm Botnet" as an Exemplar

by

Sudhir Agarwal
BEng, Siddaganga Institute of Technology, Tumkur, Karnataka, India, 2005

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science

in the Department of Computer Science

Performance Analysis of Peer-To-Peer Botnets using "The Storm Botnet" as an Exemplar

by

Sudhir Agarwal
BEng, Siddaganga Institute of Technology, Tumkur, Karnataka, India, 2005

Supervisory Committee

Dr. Sudhakar Ganti, Co-Supervisor
(Department of Computer Science)

Dr. Stephen Neville, Co-Supervisor
(Department of Electrical and Computer Engineering)

Dr. Kui Wu, Departmental Member
(Department of Computer Science)

Dr. Issa Traore, External Examiner
(Department of Electrical and Computer Engineering)

Supervisory Committee

Dr. Sudhakar Ganti, Co-Supervisor
(Department of Computer Science)

Dr. Stephen Neville, Co-Supervisor
(Department of Electrical and Computer Engineering)

Dr. Kui Wu, Departmental Member
(Department of Computer Science)

Dr. Issa Traore, External Examiner
(Department of Electrical and Computer Engineering)

ABSTRACT

Among malicious codes like computer viruses and worms, botnets have attracted a significant attention and have been one of the biggest threats on the Internet. Botnets have evolved to incorporate peer-to-peer communications for the purpose of propagating instructions to large numbers of computers (also known as bot) under the botmaster's control. The impact of the botnet lies in its ability for a bot master to execute large scale attacks while remaining hidden as the true director of the attack. One such recently known botnet is the Storm botnet. Storm is based on the Overnet Distributed Hash Table (DHT) protocol which in turn is based on the Kademlia DHT protocol. Significant research has been done for determining its operational size, behaviour and mitigation approaches.

In this research, the peer-to-peer behaviour of Storm is studied by simulating its actual packet level network behaviour. The packet level simulator is developed via the simulation framework OMNET++ to determine the impact of design parameters on botnets performance and resilience. Parameters such as botnet size, peer list size, the number of bot masters and the key propagation time have been explored. Furthermore, two mitigation strategies are considered: a) random removal strategy (disinfection strategy), that removes selected bots randomly from the botnet; b) Sybil

disruption strategy, that introduces fake bots into the botnet with the task of propagating Sybil values into the botnet to disrupt the communication channels between the controllers and the compromised machines. The simulation studies demonstrate that Sybil disruption strategies outperform random removal strategies. The simulation results also indicate that random removal strategies are not even effective for a small sized networks. The results of the simulation studies are particularly applicable to the Storm botnet but these results also provide insights that can be applied to peer-to-peer based botnets in general.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I wish to acknowledge all the people who have helped and guided me during my studies. I would like to express my sincere gratitude to my supervisors, Dr. Sudhakar Ganti and Dr. Stephen Neville, whose endless guidance, supervision and encouragement from the initial to the final level enabled me to develop an understanding of the subject.

I want to thank my thesis committee member, Dr. Kui Wu, for his time and valuable suggestions.

Lastly, I want to thank my family members for their understanding and endless support, throughout the duration of my studies.

DEDICATION

To my parents and family members!!!

Chapter 1

Introduction

1.1 Introduction

A *bot* is a machine on which malicious computer software has been installed such that it can be controlled autonomously and automatically. Botnets can be referred to as a group of bots (collection of compromised computers also know as Zombie computers) running the malicious software controlled by an attacker commonly known as the botmaster, under a common command-and-control infrastructure (C&C) [1, 2]. Researches estimate that 11 percent of the more than 650 million computes attached to the Internet were conscripted as bots [3]. The attacks not only degrade the information system infrastructure and assurance, but also the performance of the computer and network devices. It is evident that botnets are commonly used for fraudulent activities such as email spams, mass mailing, phishing attacks, DDoS (Distributed Denial-of-Service) attacks, stock scams, distribution of illegal content or other nefarious purposes [1, 4, 5, 6]. The sizes of these botnets vary from several hundred bots to tens of thousands of bots [7].

The defining characteristic of a botnet is the use of command and control (C&C) structure [5, 8]. To disseminate the botmasters' commands to their bot armies, Internet Relay Chat (IRC) has been used in the past as the C&C infrastructure protocol. IRC is a chat system that provides one-to-one and one-to-many instant messaging over the Internet [9]. The attacker can either use a public IRC server or build their own servers for the C&C purposes. Bots are configured to connect back to the C&C server listening on any configured port to receive commands and act as per these commands. Various detection schemes for identifying IRC based botnets have been

developed [4, 10, 11]. The longevity of these IRC botnets are relatively short as the bots C&C server can be easily detected, disrupted and also can be taken offline by law enforcement or other means [12]. An attacker can also use the centralized topology for distributing commands wherein the centralized server forwards messages between the bots. The major weakness in such centralized C&C structure is that its relatively easy to identify the server and its location and hence disrupt the entire botnet.

Most of the attackers have done a transition from IRC to distributed organization structure using Peer-to-Peer (P2P) architectures. To address the deficiencies of IRC botnets, the botmasters have transitioned to the use of peer-to-peer architectures and protocols. Botnets such as Slapper [13], Sinit [14], and Nugache [15] have implemented different kinds of P2P control architectures that have several advanced designs when compared to IRC botnets. P2P designs of IRC bots have many weaknesses. As a result, one single captured bot potentially could expose the entire system and would lead to the identification of the attacker. An ideal design for an advanced C&C botnet should have at least the following three properties: (a) to be as resilient as possible *i.e.,* when a substantial number of bots are removed either by disinfection, the botmaster should be able to maintain its control over the network; (b) rapid propagation of the commands and making it difficult to detect the botnet via its communication patterns; and (c) fault tolerant.

The Storm botnet is a well known real-world exemplar of a P2P botnet. Storm got its name from a particular self-propagation spam e-mail subject line about a weather disaster in early 2006: "230 dead as Storm batters Europe" [6]. It became a headline news item due to the massive amount of unsolicited e-mails it generated thereafter. The botnet has been given various labels by antivirus and security researchers such are Peacomm (Symantec), Peed (BitDefender), Tibs (BitDefender), Dorf (Sphos), Nuwar (ESET), and Zhelatin (F-Secure and Kaspersky) [5, 13, 15, 16]. The message contained the Storm binary as an attachment [5]. Those who opened the attachment got their computers infected and became a part of the ever-growing botnet. As of January 22, 2007, the Storm botnet accounted for 8% of all malware infections globally. Storm uses either social engineering techniques or drive-by-download methods for its propagation. In the former case, the Storm binary gets installed by clicking links such as those for greeting cards, video games or links to "important news of the day". In the later case, the binary gets installed when the user visits a website which exploits vulnerabilities of the user's web browser or its add-ons. The primary mission of the Storm has remained spam propagation. Table 1.1 provides a summary of some

of the common Storm outbreaks.

Date	Spam Tactic
Jan 17, 2007	European Storm Spam
April 12, 2007	Worm Alert Spam
June 27, 2007	E-card (applet.exe)
July 4, 2007	231st B-day
Aug 15, 2007	E-Card2
Aug 20, 2007	Login Information
Aug 28, 2007	Beta-Testing (setup.exe)
Sept 2, 2007	Labor Day (labor.exe)
Sept 5, 2007	Tor Proxy
Sept 6, 2007	Privacy
Sept 10, 2007	NFL Tracker
Sept 17, 2007	Arcade Games rule

Table 1.1: Common Storm Outbreaks

Frank Boldewin [17] and Porras *et al.,* [10] have performed reverse engineering and static analysis of the different released binaries of Storm namely Peacomm.C and labor.exe respectively. Based on their analysis, the process the binary follows to install itself on the victim's machine is summarized below. Readers interested in the functionality of the different binary components can refer to [10, 17, 18].

1. The first phase is the XOR decryption phase. The binary performs an XOR decryption which involves an anti-sandbox trick. This is done to crash or fool antivirus emulator engines. The anti-sandbox trick works as follows: the binary executes a system call to a legacy API windows functions (*e.g.,* FreeIconList). These API windows functions are rarely used and are so easily emulated by Antivirus (AV) Engines. The return value of the system call is added to the data to be encrypted before the XOR decryption operation is performed. The main reason is to hide the AV signature based malware detection since if the AV engine doesn't emulate the legacy API function, it will either crash or it will not succeed in decrypting the binary.

2. The next phase is the decryption phase which uses the TEA (Tiny Encryption Algorithm) [19]. The data of the portable executable (PE) section of the binary is decrypted using the TEA algorithm which uses a 128-bit key for decryption.

3. In the next step, various tasks are performed like file modifying, dropping, decrypting and unpacking the binary code and so forth. The first operation is the modification operation where in the binary modifies the system drivers such as tcpip.sys, key board drivers kbdclass.sys, cd rom driver cdrom.sys. The modification operation is followed by the deletion operation. Two files are dropped. The first one is a self-copy of spooldr.exe which is present in %systemroot% and the second file is the overlay containing the spooldr.sys driver, which gets detached to %systemroot%\system32. Next, the binary employs sophisticated root kit methods for disabling and preventing the execution of security product drivers.

4. The binary runs two virtual machine (VM) detection routines. The first one detects the presence of VMare by executing a privileged instruction that cause an exception to be generated when the instruction is executed in non-privileged mode in operating systems such as Windows and Linux, but no exception is generated with VMWare because VMWare emulator traps the exception. The second VM detection routine detects Microsoft Virtual PC by executing illegal opcodes which generate an exception in operating systems, whereas no exception is generated when the opcodes are executed in a Virtual PC [20].

5. The last phase or step is the initialization and synchronization phase. In the initialization phase the binary creates a security descriptor for the file [10] and in the synchronization phase the system synchronizes the system time using the NTP (Network Time Protocol) [5].

Based on the reverse engineering and static analysis, it is evident that the developers of Storm took care to both succeed in infecting the machines and in not being detected. References [10, 5, 17] provide additional details on Storm's installation process.

Storm botnet is based on the Overnet Distributed Hash Table (DHT) which in turn is based on the Kademlia DHT protocol [21]. It uses the fast-flux servers for secondary-stage binary distribution [22]. Storm also has a built-in defence mechanism which actively defends the botnet by executing DDoS attacks on those hosts that attempt to determine its internal workings. The authors of the Storm botnet continually update its design and architecture as the anti-virus companies have been able to detect its presence. Much of the research interest in Storm has primarily

been related to its detection, its removal, estimating the botnet's size and and its network behaviours. The size of the Storm botnet has been roughly estimated to be between 1 million to 5 million bots but there has been various reports on the size to be around 10 million [23]. Despite all the hype created, the enginnering design decisions associated with P2P botnets has not been well studied. This work develops a understanding of the engineering design decisions associated with Storm P2P botnet. A simulator model for Storm botnet is built via the open source simulation framework OMNET++ [24] and is known as the standard growth model of Storm botnet. In the remaining chapters, the simulator model for Storm botnet is referred as the standard growth model of Storm Botnet. The simulator model contains the full framework of the Storm. This model is used for a detailed analysis of Storm botnets key attributes.

Furthermore, we provide a detailed analysis of the botnet attributes and how they effect other bots present in the botnet. The following contributions are made in the thesis:

1. Determine the impact of number of bot masters, bot's peer list size and initial number of bots on <*key, value*> propagation and retrieval.

2. Determine the percentage of bots that received the <*key, value*> pair versus time.

3. Determine the mean time required by the bot to obtain a <*key, value*> pair based on the botnet's design attributes.

4. Study how design parameters effect the network load and the visibility of the botnet within the underlying network layer.

5. Determine the effect of random bot removal on the botnet.

6. Determine mean number of messages a bot sends to obtain the <*key, value*> pair from the network and determine how this size varies based on different design attributes.

7. Determine the impact of size of the botnet. Obtain results by reducing the total number of bots present in the system to half.

Two different mitigation strategies are evaluated. Random disinfection strategies and Sybil distruption strategies which disrupt the communication channel between the controller botmaster's and bots. We also determine the weaknesses of the design

parameters of the bot that could possibly effect the mitigation strategy in addition to disrupt the communication between the bots. Our studies indicate that Sybil attacks are more effective when compared to the random disinfection P2P-based strategies. Our results concur with the authors of [25] and indicate that Sybil attacks provides a higer degree of disruption to the functionality of the botnet. Results of botnet simulations are particularly applicable to Storm but the reported results are also likely to hold for other P2P structured botnets. The Storm botnet was chosen because its inner workings have become fairly well understood, and its engineering design constraints are relatively common across malicious peer structured botnets.

1.2 Thesis Organization

The remaining of the thesis is organized as follows. Chapter 2 describes the related work and provides an outline of how this work differs from the previous works. Chapter 3 contains information relating to the architecture (design and implementation) of the Storm botnet and its evaluated disinfection and distruption strategies. Chapter 4 contains information relating to the simulation setup, and the assumptions that were made. In addition, simulation results with respect to botnet design parameters and disinfection analysis are also discussed in detail. Chapter 5, concludes the work with a summary of contributions and possible directions for future work.

Chapter 2

Background and Related Work

Storm uses a modified version of Overnet P2P protocol for its command and control (C&C) infrastructure. An overview of Peer-to-Peer Overlay network, Overnet and Storm is presented below; in addition background on related research work and contributions are presented.

2.1 Terminology

In the following sections and chapters the terms client, peer, node, bot, contact and identifier are introduced. The client represents the running application of the peer. A peer is an active connecting point in the Kademlia network, which is applied by the client. A node or a bot is another peer with a fixed identifier in the Kademlia network. The contacts are known peers or nodes or bots, which are inserted into the routing table. Identifier is a quasi-unique binary number that identifies a node in the network. Furthermore a contact which is already known to a client, but cannot be ascertained whether it is alive or not, will be called a possible contact.

2.2 Peer-to-Peer Overlay Network

Peer-to-Peer (P2P) network overlays are of active research interest because they provide an efficient substrate for creating large-scale data sharing. Potentially they offer an efficient routing architectures that are self-organizing, scalable, fault tolerant, and robust. P2P overlay networks are generally classified into two categories: unstructured networks and structured networks [26].

2.2.1 Unstructured Overlay Network

An *unstructured overlay network* is formed by peers randomly joining and leaving the network. There is no fixed limit to the number of peers that a node may connect to and these peers do not follow a specific set of rules. The network operation is decentralized and follows no specific requirement for the network topology or for publication, replication and retrieval of data. The most popular application based on unstructured overlay network is Gnutella [27] and Freenet [28]. The original version of Gnutella used Flooding in order to retrieve a data item while protecting the anonymity of both authors and readers. While this method is highly robust and flexible, it is not scalable. No broadcast search or centralized location index is employed [28]. The advantage of these protocols is that popular data files can be found easily and transferred efficiently.

2.2.2 Structured overlay network

Structure P2P overlay network topologies are more tightly controlled and the data is not placed randomly on peer nodes. Each node connects to at most k peers, where k is a fixed parameter. The value of k depends on the specific overlay network topology. The network consists of a number of co-operating nodes that communicate with each other and store information about one another. The location of a peer in the Overlay network is not geographically determined which makes the queries sent by the nodes more efficient. Structured P2P networks generally use Distributed Hash Table (DHT) to organize their peers list and contact information [29]. Each node is uniquely identified with a node identifier. Data within the network is assigned to unique identifiers called keys. Each peer in the network maintains the contact information about other peers in its routing table. Based on the peer-to-peer protocol, different organization schemes are employed for the data objects and the contact information. The most popular structured overlay networks are Chord [30], Pastry [31], and Overnet [32].

2.3 Overnet P2P Protocol

Overnet is a decentralized structured peer-to-peer proprietary file sharing overlay network protocol. It implements a Kademlia-based [21] peer-to-peer distributed hash table (DHT) routing protocol. Overnet was implemented by Edonkey [33]. In late

2006, it was officially shut down [33] as a result of legal action from the Recording Industry Association of America (RIAA) and others, but benign Overnet peers still exist in the Internet.

An Overnet network consists of a number of cooperating nodes that communicate with one another and store information about one another. Each node has a 160-bit node *ID* identifier, a quasi-unique binary number that identifies it in the network. The *ID*'s are generated when each node first joins the network. The *ID* is transmitted with every message the node sends, permitting the recipient of the message to identify the sender's existence if necessary. Within the network, a block of data or the key's value, is also associated with a binary number of the same fixed length 160-bit. A node needing to search a value looks-up for it first at the node it considers closest to the key identifier. A node requiring to save a value, stores the information of the closest node to the key.

Each node in an Overnet network stores contact information about other nodes to route query messages. Every node keeps a list of *<IP address, UDP port, ID>* triplets for nodes of distance between 2^i and 2^{i+1} from itself in its i'th buckets which hold a maximum of k contacts, where $0 \leq i < 160$. These lists are referred to as k-buckets. Overnet defines the distance between two 160-bit identifiers, x and y, as their bitwise exclusive OR (XOR) interpreted as an integer, $d(x,y) = x \oplus y$. The buckets are organized by the distance between the nodes and the contacts in the bucket. Specifically, for bucket i, $0 \leq i < k$, we are guaranteed that

$$2^i \leq \text{distance(node, contact)} < 2^{i+1}$$

Given the large address space, this means that bucket zero has only one possible member, the key which differs from the node *ID* only in the high order bit and for all practical purposes is never populated. On the other hand, if node *ID*'s are evenly distributed, it is very likely that half of all nodes will lie in the range of bucket 159.

Each k-bucket is kept sorted by the time last accessed, ordered by the least-recently accessed at the head and the most-recently accessed at the tail. Each k-bucket contains at most k nodes, where k is a configurable parameter. When a node (the recipient) receives a message from another node (the sender), the recipient updates its k-bucket with the sender's *<IP address, UDP port, ID>* triplet as follows:

1. If the sender node *ID* is already in the recipient's k-bucket, the sender's triplet is moved to the tail of the k-bucket.

2. If the sender's triplet isn't in the recipients corresponding k-bucket then,

   (a) If the number of entries in the k-bucket is less than k, the sender's triplet is added to the tail of the k-bucket.

   (b) If the corresponding k-bucket is full, the recipient pings the least-recently accessed node which is at the head of the bucket. This is done in order to insert the sender's triplet into the k-bucket.

      i. If the least-recently accessed node responds, the recipient moves the least-recently accessed node to the tail of the list, and the new sender's contact is discarded.

      ii. If the least-recently accessed node fails to respond, the recipient removes the least recently accessed node from the corresponding k-bucket and adds the sender's triplet to the tail of the bucket.

Overnet implements the four message types provided by the Kademlia protocol which are outlined below:

1. PING: A node issues a PING message to probe a node to determine if it is on-line.

2. STORE: This message is used by the node to send the <*key, value*> pair to other nodes for later retrieval. If the node wishes to publish a <*key, value*> pair, it locates the k closest nodes that are closest to the key and sends them a STORE message, with a 160-bit key (*e.g.,* hash of a file identifier) and *value* (*e.g.,* an audio, video or some data file) being searched for.

3. FIND_NODE: This message is used by a node to search for k closest node *ID*'s (a 160-bit quantity). When a node receives a FIND_NODE message, it returns the <*IP address, UDP port, ID*> triplet of the k nodes it knows that are closest to the node ID. The recipient must return k <*IP address, UDP port, ID*> triplets if it has k or more entries in its bucket. It may only return fewer than k if it is returning all the contacts that it has knowledge of.

4. FIND_VALUE: A node can issue a search for a <*key, value*> pair via the FIND_VALUE message. If the corresponding key value is present at the recipient, the associated <*key, value*> pair data is returned; otherwise it returns the <*IP address, UDP port, ID*> triplet of the k nodes it knows of that are closest to the key.

2.4 Storm C&C Network Protocol

This section provides an overview of Storm operation. An outline of Storm's Command and Control (C&C) protocol is provided rather than presenting the functionality of its different binary components.

Storm uses the Overnet protocol, which in turn is based on the Kademlia DHT [21]. The main difference between Storm C&C architecture and Overnet P2P network is that Storm nodes XOR encrypt their messages and use a 128-bit identifier as opposed to 160-bit identifier. Peer identifiers are randomly generated using the MD4 cryptographic hash function [5]. Once the Storm binary is installed on the victim's machine, the binary generates a 128-bit ID and initializes its peer list file using the MD4 cryptographic hash function. The peer list file contains the <IP address, UDP port, ID> triplet informations, of other bots which are hard-coded in the binary. These triplet information are only for the inital settings of the peer list entries. Every Storm node contains its own randomly generated peer-list file. Storm nodes organize their routing tables as list of k-buckets. Specifically, for each $0 \leq i < 128$, the corresponding k-bucket holds up to k *<IP address, UDP port, ID>* triplets, where the value of k is typically 20, for nodes of distance between 2^i and 2^{i+1}. When an Overnet node receives any message from another node, it updates its appropriate k-bucket with the sender's node *<IP address, UDP port, ID>* triplet and returns the triplet of the k node it knows about, that are closest to the requested ID. These triplets can come from a single k-bucket or from multiple k-buckets if the closest k-bucket is not full.

In addition to receiving messages from another nodes and returning k triplets, Storm node periodically searches for a set of keys stored in the Overnet network. According to [5, 18], communication within the botnet proceeds as follows: Storm nodes generate 32 different 128-bit keys each day through a built-in algorithm of the form *f(d, r)*, which takes as input the current day (d) and a random number between 0 and 31 (r); and sends search queries to their contacts for these keys. The values associated with those keys contain an encrypted URL that Storm nodes decrypt and download using protocol such as HTTP. Store key value pairs may or may not exist in the network. The value associated with these keys are of the form "*.mpg;size=*;", where the asterisks are 16-bit numbers which are presumably used to compute IP address and a port of a re-director in the Storm fast-flux network, according to reverse engineering analysis performed by the authors of [10].

2.5 Related Work

Various mitigation strategies have been explored in the past to exploit the weaknesses of the protocol used by the Storm bot, in an effort to disrupt the communication between the bots. In the remaining part of this chapter we present the background on related research work, as well as introduce our contributions with respect to mitigation strategy.

Holz, Steiner, Dalhl, Biersack and Freiling [5], presented the first empirical study of P2P Storm botnet giving details about its propagation phase, malicious activities and other features. They provided a case study showing how to use Sybil's to infiltrate the Storm botnet. They used an active measurement technique to crawl the P2P network called the Overnet crawler. The crawler runs on a single machine and uses the breadth first search technique to find the peers currently participating in Storm or Overnet network. The goals of their work were: a) to estimate the number of compromised machines and to estimate the size of the Storm by infiltrating the botnet with Sybil's; b) disrupt the communication channel between the controller and the compromised machine using two different mitigation strategies, that is, Eclipse and polluting; and c) determine the effect of the pollution attack by polluting the keys used by the Storm. Their experiments showed that by polluting the keys that the Storm uses, they were able to disrupt the Storm botnet communication. Also the Eclipse attack, that they used to separate a part of the P2P network from the rest, is not feasible to mitigate the Storm botnet network as the Overnet keys are distributed throughout the entire hash table space, rather than be restricted to a particular zone.

Davis, Fernandez, Neville and McHugh [25] explored the feasibility of the Sybil attack against Storm botnet. In a Sybil attack, the network is infiltrated with a large number of fake nodes; known as the Sybil's, in order to disrupt the communication between the bots. The authors outline a methodology for mounting practical Sybil attacks, on the Storm botnet. Their contributions can be summarized as: a) determine the number of Sybil nodes required to disrupt the communication between the bots; b) effect of the duration of Sybil attack in disrupting the botnet communication; and c) effect of the botnet design choices such as the size of a bot's peer list on the effectiveness of the attack. Their simulation studies showed that for a significant degradation of the Storm bot, substantial and sustained Sybil attacks are required. Moreover, an uninformed Sybil attack has near-zero impact on the botnet operators, whereas informed Sybil attack could be more effective but are more impractical

given that for a successful attacks global information regarding path response time is required. Their work is complementary to the work of the authors [5].

Singh, Ngan, Druschel and Wallach [34], studied the impact of Eclipse attacks on structured overlay P2P networks. The main goals of their work were to: a) study the impact of Eclipse attack;s and b) limitations of the known defences (secure routing [35]) with respect to Eclipse attacks. An Eclipse attack is one in which a set of malicious colluding overlay nodes arrange for a targeted correct node to be peered with only members of the malicious coalition. If successful, the attacker can mediate most of the overlay traffic and "eclipse" correct nodes from each other's view. Hereby, enabling censorship or denial of service attack. They also proposed on how to bind the in and out-degree of overlay nodes, and presented a defence strategy. Their experiments also showed that Eclipse attacks are not effective for all applications. As the Overnet keys are distributed throughout the entire hash table space, eclipse attacks are not feasible to mitigate.

Steiner, En-Najjary and Biersack [36] studied the KAD peer-to-peer file sharing application. The main goal of their study was to identify how KAD, a Kademlia based [21] DHT can be used and misused. The authors developed a crawler for KAD which used a simple breath first search technique to find the peers currently participating in KAD. Based on their findings they concluded: a) for the Sybil attack to be effective, the attacker needs to introduce thousands of Sybils in order to disturb the system; and b) Denial-of-service and Eclipse attacks can be introduced, so that the peer can connect to the machine that is intents to target.

Dumitriu, Knightly, Kuzmanovic, Stoica and Zwaenepoel [37] studied the resilience of P2P file sharing system against denial-of-service (DoS) attack by means of analytical modelling and simulation. Main goal of their work was to determine the effect of: a) file-targeted attack; and b) network-targeted attack. In file-targeted attacks, the attacker puts a large numbers of corrupted versions of a single file on the network. In the later, attackers respond to queries for any file with erroneous information. Their experiments showed that file-targeted attacks are highly dependent on the client's behaviour. The attack is successful only if the clients are unwilling to share the files and they do not remove or are slow to remove the corrupted file from their system.

Christin, Weigend and Chuang [38] conducted a measurement study to determine the impact of pollution and poisoning on content availability of four popular peer-to-peer file sharing networks, *e.g.,* Gnutella, EDonkey, Overnet and FastTrack. In

"poisoning" technique, the availability of a targeted item (movie, song, or software title) is reduced by injecting a massive number of decoys into the peer-to-peer network. In the case of pollution, content availability is reduced by injecting unusable copies of files in the network. Their experiments showed that poisoning and pollution can be highly affective depending on the injection rate of the polluted or poisoned version of a popular file. Content availability was shown to be highly reduced only when the polluted or poisoned version of the file was injected in the network on a massive scale.

Liang, Naoumov and Ross [39] developed a methodology for estimating the index poisoning levels and pollution levels in both structured and unstructured file-sharing systems in the network. "Index poisoning" attack is one in which availability of set of targeted files (movie, song, or software titles) is reduced by inserting massive number of false or poisoned records into the index for the targeted file. This causes the search results for the targeted title to be a large number of false indices, where these indices could be false file identifiers, IP addresses or port numbers. Their studies showed that both structured and unstructured P2P file-sharing systems are highly vulnerable to the index poisoning attack.

Defrawy, Karim, Gjoka, Minas, Markopoulou, and Athina [40] have investigated index poisoning attacks in BitTorrent.

Most of the studies that have been presented with respect to the disinfection and mitigation strategies have pursued a graph-centric analysis of peer-to-peer botnets. Authors of [5, 25] have provided their results more specific to Storm botnet. Our work is closely related to the work of the authors of [5, 25]. Most of the engineering constraints such as underlying network level issues (*e.g.,* timing) within peer-to-peer botnets are not accounted for in most of the related work. Our simulations results are based on the packet level simulations using an underlying network infrastructure and topology.

2.6 Contributions

In this research, a packet level simulator model of the Storm botnet with the Kademlia infrastructure is developed, inclusive of the underlying network topology to obtain understanding of its attributes and to provide insights that would be helpful to understand the Storm phenomenon. The researchers of botnet C&C infrastructure have not focused on issues such as the impact of network timing and botnet design parameters. To the best of our knowledge, our work with respect to the botnet design

parameters is unique as it has not been explored by any researchers. Prior P2P botnet researchers has largerly not addressed the following:

1. The impact of number of botmasters in a given Storm network.
2. The percentage of bots that receive the <*key, value*> pair as a function of time.
3. The mean time required by the bots to retrieve the <*key, value*> pair from the network.
4. The impact of the total number of bots in the network.
5. The impact of design parameters on networks load and its detectability.

Authors of [5, 25] have explored some of the parameters (duration of sybil attack and bot's peer size) that could affect the effectiveness of a attack against the botnet but have not considered a full range of design parameters.

In addition, this work also evaluates two mitigation strategies (random disinfection and Sybil distruption strategies) to disrupt the botnet C&C structure. Rather than just evaluating the effectiveness of these mitigation strategies we also determine the weaknesses of the design parameters and attributes of the bot, that could possibly affect the effectiveness of the mitigation strategy in addition to disrupt the botnet C&C structure.

Chapter 3

Using OMNET++: P2P Botnet Architecture and Implementation

Kademlia based networks can have more than one million simultaneous users, as it has become the most widely deployed DHT-based protocol. Storm botnet uses the Overnet protocol which in turn is based on the Kademlia DHT [21]. Open source discrete event simulator framework OMNeT++ version 3.3 [24] is used to develop a more flexible simulator framework for P2P Storm botnet. OverSim [41], a simulation framework based on OMNeT++, contains the implementation of structured Kademlia peer-to-peer based protocol. The Kademlia based protocol is largely undocumented and provides an abstract level of detail, making it hard to extend and simulate the dedicated Storm based botnet. The simulation model implemented provides a simple model of Storm botnet for simulations.

In the following subsections the architecture details, functions, engineering design parameters and the mitigation strategies for the P2P based Storm botnet is described. In Section 3.1 and 3.2, Storm botnet's key engineering design parameters, its architecture details are explained. Section 3.3 describes the Random disinfection and Sybil disruption mitigation strategies that are deployed to disrupt the botnets C&C structure.

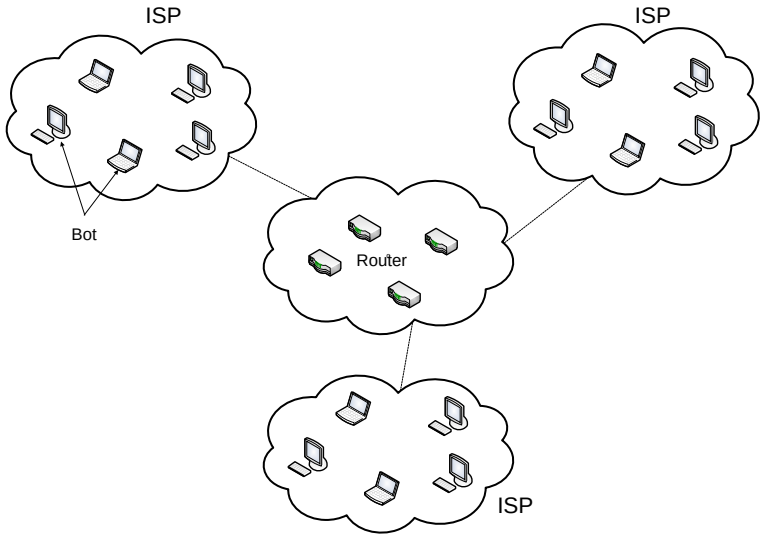

Figure 3.1: Design of Storm Botnet

3.1 Architecture and Implementation

The botnet's P2P architecture is not the same as the authors of [21] had designed. Botnet authors have been modifying its network architecture to make it difficult for the researchers to understand its internal working and to make it more resilient. Few architecture and implementation details that are required to develop this modified version is explained in the following sections.

3.1.1 Simulation Model Design

An overview of the architecture is illustrated in Fig. 3.1. The INET framework was developed by the OMNeT++ community, to accommodate OMNeT++ simulation mechanisms with network standards and protocols. INET is suited for simulations of wired, wireless and ad-hoc networks. It implements and supports many important network protocols such as IP, UDP/TCP, Ethernet, PPP, OSPF, RSVP-TE signalling, and 802.11. INET framework uses OMNET++ simulations concepts (such as queuing, timing, event handling, etc.) and implements protocol dependent features such as packet structures, signalling, routing, *etc.,* to support the generic simulation of various Internet models. The INET model is extended to support botnet's underlying P2P protocol. Technical details about the exact network simulations are presented in the next chapter.

The implemented Botnet simulator uses discrete event simulation (DES) framework based on OMNeT++, to simulate, exchange and process the network messages of the Kademlia Overlay network. The simulator design of botnet is composed of two sets of modules: Server module and Node Module. Components (modules) are defined by the high level language called NED [24]. Modules defined here are a combination of both simple as well as compound modules (collection of simple modules), which are directly implemented in C++.

Server Module

The Server Module acts as a service provider (ISP). The server module is characterized by five variables which are *maxBots*, *initialNoBots*, *tBotRemoval*, *tBotCreation* and *mitigationSybil*. Table 3.1 describes the four attributes that are used by the server module. The main task of the server module is the creation of bots. Bots are added into the botnet based on the *tBotCreation* parameter, an exponential distribution parameter. The birth rate of the bots is given an exponential distribution. At every *tBotCreation* time, a newly created bot with fully populated list of peers is added into the botnet. For the simulation, the Server Module initializes the peer list to a random set of peers, *i.e.,* the peer list is free to change over time, for the newly created bot. This peer list contains the <IP address, UDP port, ID> triplet informations, of other bots.

The *mitigationSybil* parameter categories a bot as a regular bot or a Sybil bot. If the *mitigationSybil* parameter value is set to true, then all the bots created by the corresponding server act as Sybil bots. The birth growth rate of the Sybil bots have the same exponential distribution as that of the normal bots. All the characteristics of the regular bot are applicable to the Sybil bots. Only difference between a Sybil bot and a regular bot being a flag differentiating it as a Sybil bot.

Node Module

The Storm consists of co-operating bots that communicate with one another and store <*key, value*> pair information. The node module is composed of a set of modules, each of which compartmentalizes a related group of functions and classes for node level Storm activities. The node module can be sub-divided into two distinct modules. Figure 3.2 shows the structure of the node module. UDP message processor module acts as an interface to send and receive messages between bots present in the

Attribute	Description	Example
maxBots	The number of bots to be created by the server during the simulation.	If this parameter value is set to 1000, then the number of active bots created by the server is not beyond 1000.
initialNoBots	The total number of bots to be created by the server at the beginning of simulation.	If this number is set to 1000 and the *maxBots* is set is 100, number of active bots present at the beginning of the simulation is 100.
tBotRemoval	Specifies the exponential removal time of the bot from the botnet.	If the *tBotRemoval* is 25, then at every exponential distribution of 25 seconds a randomly selected bot is removed from the botnet.
tBotCreation	Specifies the exponential bot creation time by the server.	If the *tBotCreation* is 5, then at every exponential distribution of 5 seconds a new bot is added into the botnet. The newly created bot has a fully populated list, uniformly selected bots from the botnet.
mitigationSybil	Categorizes a bot as a sybil (fake) bot or a normal bot.	If *mitigationSybil* is true, all the bots created by the server are Sybil bots.

Table 3.1: Server Module Attributes

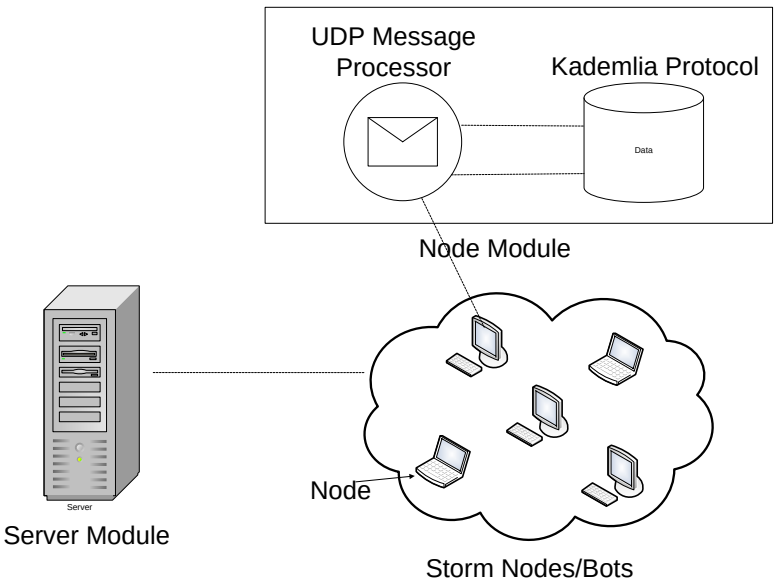

Figure 3.2: Structure of Storm Botnet

Attribute	Description
k	*k* is the maximum number of contacts stored in each *k*-bucket; this is normally 20.
alpha	*alpha* is a small number representing the degree of parallelism in network calls, usually 3.
keyLength	*keyLength* is the size in bits of the keys used to identify bots, store and retrieve data; in basic Kademlia this is 160, the length of an SHA1 digest (hash). For Storm, this value is 128.
key	Specifies the key that is stored or retrieved from the botnet, which is a binary number of length *keyLength*.
value	Specifies the block of data that is being stored or retrieved from a Kademlia network where the length of this data is *keyLength* bits.

Table 3.2: Kademlia Protocol Attributes

Attribute	Description
maxServers	The maximum number of servers present in the botnet. Each server node acts as a Internet service provider (ISP).
peerListSize	Initial peer list (list of contacts) size of each bot created.
maxBotMasters	Specifies the maximum number of botmasters present in the botnet.
sendKeyValueTime	Time instance at which botmaster's inject the <*key, value*> pair into the botnet. For example, to compensate for the bots leaving the botnet, Kademlia republishes each <*key, value*> pair into the botnet once every hour.
bucketRefreshTime	Time instance at which the flag contents of the *k* bucket are refreshed.
keyRefreshTime	Key change time. Time instance at which the bot looks up for a new <*key, value*> pair. For example, if *keyRefreshTime* is 1800, every 30 minutes a new key is looked up.
keyValueLookUpTime	Look up delay. Time instance for the <*key, value*> pair look up. For example, if the *keyValueLookUpTime* is 225, a non-botmaster sends next set of *alpha* bots into the botnet.

Table 3.3: Generic Attributes

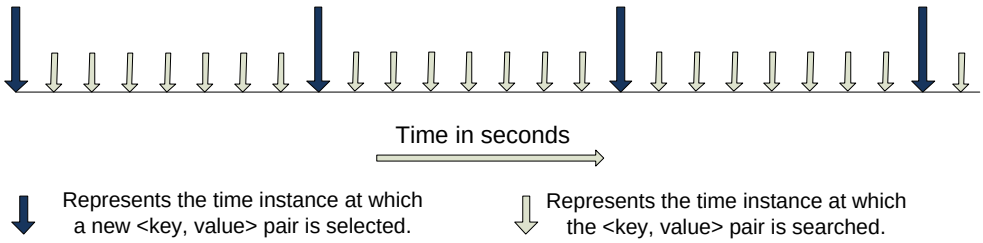

Figure 3.3: <*key, value*> pair Look Up parameters.

botnet, while Kademlia Protocol module contains the modified Kademlia protocol implementation. These modules are characterized by a set of attributes.

Tables 3.2 and 3.3 describes the most important attributes that are used by these two modules. Table 3.2 describes all the Kademlia protocol attributes that is used by the Kademlia Protocol module. Table 3.3 describes the generic attributes that are used by the UDP message processor and Kademlia Protocol module for simulations.

bucketRefreshTime parameters specifies the time at which the flag contents of the k-bucket are refreshed. When a bot details is obtained from the k-buckets, a flag value for the bot is set to true to ensure that the same bot is not contacted again for the <*key, value*> pair retrieval. To avoid pathological cases where no bot information is obtained from the k-bucket, each bot resets the flag of all k-bucket entries at every *bucketRefreshTime*. *keyRefreshTime* and *keyValueLookUpTime* parameters specify the <*key, value*> pair look up time. *keyRefreshTime* specifies the time instance at which a new <*key, value*> pair is looked up, whereas, *keyValueLookUpTime* specifies the time instance at the which the selected <*key, value*> pair is look up in the Internet. Figures 3.3 explains the *keyRefreshTime* and *keyValueLookUpTime* parameters.

The lifetime of the botnets is composed of three stages: (a) initialization, (b) look up, and (c) propagation. In order to understand these stages, the characteristics of the bots must be understood and these are described below.

3.1.2 Storm Network and *k*-bucket Details

The Storm botnet uses the UDP-based protocol built on the Overnet protocol that provides Storm much of its resiliency. This message protocol is used by the Storm masters and the bots present in the Internet for communication. Since the Storm uses UDP protocol to faciliate the message communication among bots, it does not

k-bucket Number	*Available Address Space*
127	$2^{\wedge}127$
126	$2^{\wedge}126$
125	$2^{\wedge}125$
124	$2^{\wedge}124$
3	$2^{\wedge}3$
2	$2^{\wedge}2$
1	$2^{\wedge}1$
0	$2^{\wedge}0$

Figure 3.4: Address space of k-buckets.

require the bots to establish a session in order to send and receive messages.

Storm has the possibility of creating 2^i unique bots in the botnet at the same time, where the value of i is 128. Figure 3.4 illustrates in address space of each k-bucket. Each bot has its own ID (identifier) generated using the uniform distribution function in order to ensure that the bots are uniformly distributed in the botnet. The bot ID is uniformly distributed as the bot ID determines its position (not geographical location) in the botnet. Data being stored or retrieved from the botnet must also have a key of the same length as the bot ID. Storm operations are based upon the use of exclusive OR or XOR metric. The XOR metric is applied to get the distance between any two keys or bot IDs.

The routing table of each bot in the Storm is organized such that the information about other bots that it knows of is stored in its several k-buckets. The total number of buckets that are present is equal to the length of the binary key which is based on the attribute *keyLength*. Moreover, the maximum number of bot details stored in each bucket is restricted to the attribute value k; this is normally 20. Whenever a

bot receives contact details from another bot present in the network, it updates its k-bucket. Based on the operation being performed by the bot, k-buckets are assigned ID ranges based on XOR distance performed between the bot ID's or between the bot ID and *key* ID.

3.1.3 Network Messages

The lifetime of botnets is composed of three stages. The first stage is the initialization stage. In this stage, all the botmaster's are initialized so that they can inject the <*key, value*> pair in the botnet. For the simulation, these botmasters are randomly selected points in the botnet with the task of enjecting the same <*key, value*> pairs into the botnet. The second stage is the lookup stage. In this stage, the bots establish connection with the other bots that are present within the same botnet. The third stage is the propagation stage where in the <*key, value*> pair is retreieved by the bots from the internet. Storm uses different message types for the execution of these three stages. The original Kademlia paper [21] specifies that the Kademlia protocol consists of four remote procedure calls (RPC's), or messages, but then goes on to specify procedures that must be followed in executing these. In order to differentiate the messages, the *message kind* attribute provided by the OMNeT++ simulation message class is used to specify the message type information. All the messages that a bot transmits contains its 128-bit ID. Simulation timers are used to control these messages. These timers either terminate or initiate operations like iteration, searching or publishing process at a predefined interval based on the message type. The procedures that is followed by the messages are described below:

Also I did not see where you clearly defined what you meant by a "botmaster" relative to the simulation - you should clearly define that you mean by this term and that you are not assuming N competing bot masters on a single botnet or a botnet split into N disjoint parts - instead the definition seems to be thtat the ¡key,value¿ pair store instruction is issued from N randomly selected points in the botnet

1. **PEER LIST Message**: The server module sends the PEER LIST message to the newly created bot. The PEER LIST message contains the <*IP address, UDP port, ID*> triplet information of other bots. Once this message is received by the bot, the bot updates its corresponding bucket based on the contacts present in the message. PEER LIST message is used along with PING PONG message. Figure 3.5 illustrates the scenario.

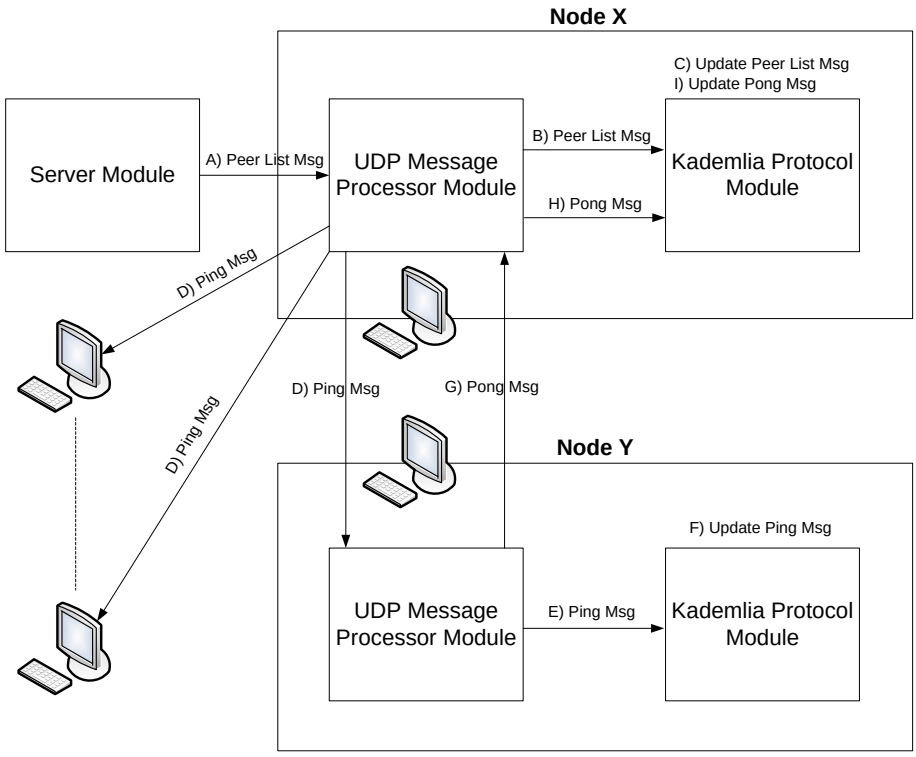

Figure 3.5: Peer List and Ping-Pong Message Flow Diagram

2. **PING PONG Message**: A bot, say X, issues the PING message to probe a bot, say Y, to determine if it is on-line. If the receiving bot Y is present in the Internet, it responds back to the sender bot X with a PONG message and updates its corresponding bucket with the contact information of the the bot X.. Figure 3.5 illustrates the scenario with respect to both PEER LIST message as well as PING PONG message. The following steps needs to be performed with respect to PEER LIST and PING PONG message:

   (a) The bot creator (server) sends PEER LIST message to the newly created bot say X.

   (b) UDP message processor module, of the newly created bot, receives the message and forwards it to its Kademlia Protocol module.

   (c) Kademlia Protocol module of X receives the message and stores the contact information of other bots, as present in the message, into its corresponding k-bucket by performing XOR operation between its own bot ID and received bot ID.

(d) The UPD message processor module of bot X sends PING message to all the contacts as present in the PEER LIST message.

(e) When a bot (say Y) receives the PING message from the bot (X), it forwards the PING message to its Kademlia Protocol module.

(f) The recipient's Kademlia Protocol module updates the appropriate k-bucket for the sender's bot ID. XOR operation is performed between sender's (X) bot ID and recipients (Y) bot ID.

(g) The recipient's (Y) UDP message processor module also sends back a PONG message to the sender bot (X).

(h) The sender bot (X) receives the PONG message from the recipient bot (Y) and forwards it to its Kademlia Protocol module.

(i) Kademlia Protocol module of X receives the message and stores the contact information of sender bot (Y), into its corresponding k-bucket by performing XOR operation between sender's bot ID (Y) and recipients bot ID (X).

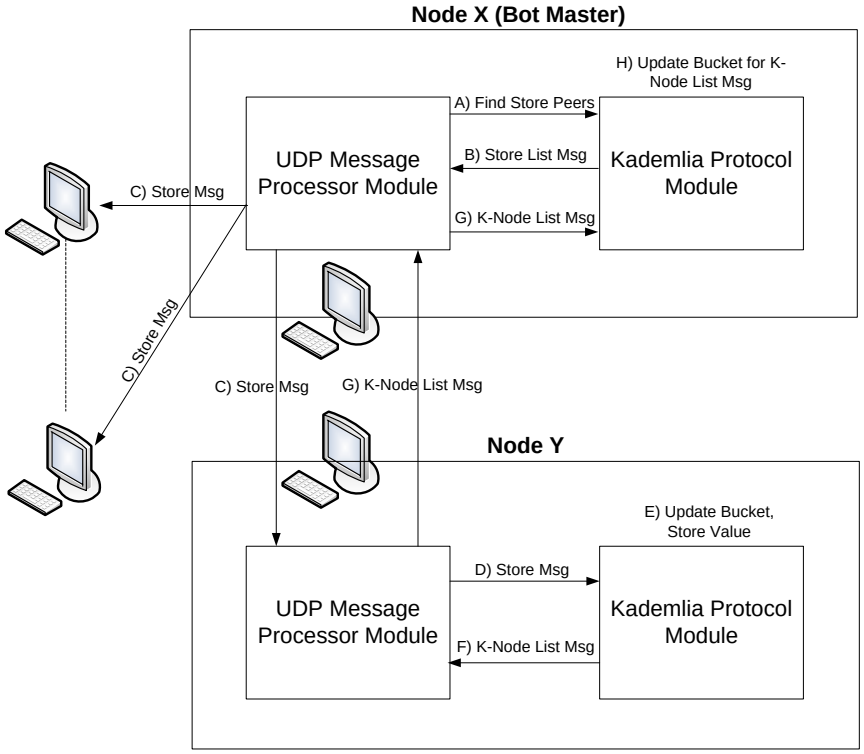

Figure 3.6: Store Message Flow Diagram

3. **STORE Message**: This message is used by the botmaster to distribute the <*key, value*> pair into the botnet. In order to publish the <*key, value*> pair, the botmaster locates the k closest bots that are closest to the key and sends them a STORE message. XOR operation is performed between the bot ID and the key ID to determine the bucket number from which the k closest bots entries are retreived. Apart from the botmasters, <*key, value*> pair can be distributed by bots if and only if it they have the <*key, value*> pair and a bot has requested <*key, value*> pair. The <*key, value*> pair is injected by the botmaster at every *sendKeyValueTime*, an engineering design parameter.

   Following steps are followed by the botmaster to send the STORE message. Figure 3.6, gives a general outline of the STORE message.

   (a) At every *sendKeyValueTime*, the botmaster (say X), sends a message to its Kademlia Protocol module to retrieve the closest set of bots for the key. In order to determine the k closest bots, XOR of botmaster's ID and the key ID is calculated to determine the corresponding bucket number. In order to ensure that the same bot is not contacted again for the <*key, value*> pair retrieval, bot details of previously contacted bots are not included for the k closest bot list. To avoid pathological cases where no bot information is present, each bot refreshes its k-buckets every hour. In this way, when the buckets are refreshed, the bot can reselect the previously selected bots for <*key, value*> pair distribution.

   (b) The Kademlia Protocol module of X sends the k closest list message to its UDP message processor module.

   (c) For each bot IDs obtained from the k closest list message, the UDP message processor module sends the STORE message to all the k closet bots. The STORE message contains the <*key, value*> pair in addition to the contact information of bot X.

   (d) When the bot (say Y) receives the STORE message from the bot X, it forwards this message to its Kademlia Protocol module.

   (e) The recipient's Kademlia Protocol module updates the appropriate k-bucket for the sender's bot ID (X). XOR operation is performed between key ID and recipients (Y) bot ID. A check is also performed to determine if the bot Y is a botmaster or not. If the bot Y is not a botmaster, then it

stores the <*key, value*> pair. If the bot Y is a botmaster, then the <*key, value*> pair is discarded.

(f) If the bot Y is a not a botmaster, message containing k closest contact details are sent to its UDP message processor module.

(g) The recipient's (Y) UDP message processor module then sends back the obtained k contact list message to the sender bot (X).

(h) The bot's (X) UDP message processor module receives the message from the bot Y, and forwards the message to its Kademlia Protocol module.

(i) Kademlia Protocol module of X receives the message and stores the contact information of sender bot (Y), into its corresponding k-bucket by performing XOR operation between recipient's bot ID (X) and key ID.

4. **FIND_NODE Message**: This message is used by bots to find the k closest bot IDs. When a bot receives a FIND_NODE message, it returns the <*IP address, UDP port, ID*> triplet of the k bots it knows of that are closest to the bot ID. The recipient must return k <*IP address, UDP port, ID*> triplets if possible. It may only return fewer than k triplets if it is returning all of the contacts that it has knowledge of. The FIND_NODE message is usually used along with FIND_VALUE message.

5. **FIND_VALUE Message**: This message is used by the bot to find the <*key, value*> pair. Botmasters do not use this message as they inject the original <*key, value*> pair into the Internet. In order for the bot to find the <*key, value*> pair, the bot performs a lookup to find *alpha* bots with IDs closest to the key. The lookup starts up by picking up *alpha*, an engineering design parameter, bots from its closest k-bucket and sending them the FIND_VALUE message. If the bucket has fewer than *alpha* entries, it just takes all available closest bots that it is aware of. When a bot sends a FIND_VALUE message, there are two scenarios that could arise. The first scenario being, the corresponding <*key, value*> pair is present with the recipient. And in the second scenario, the <*key, value*> pair is not present with the recipient. The FIND_VALUE message is sent to the closest bots repeated at every *keyValueLookUpTime* seconds, a engineering design parameter. <*key, value*> pair search process is halted immediately when any bot returns the value.

Following steps are followed by the bot to find the <*key, value*> pair. Figure 3.7 and 3.8, shows the general outline of the FIND_VALUE message based on the cases specified below:

Case 1: <*key, value*> pair is present with recipient.

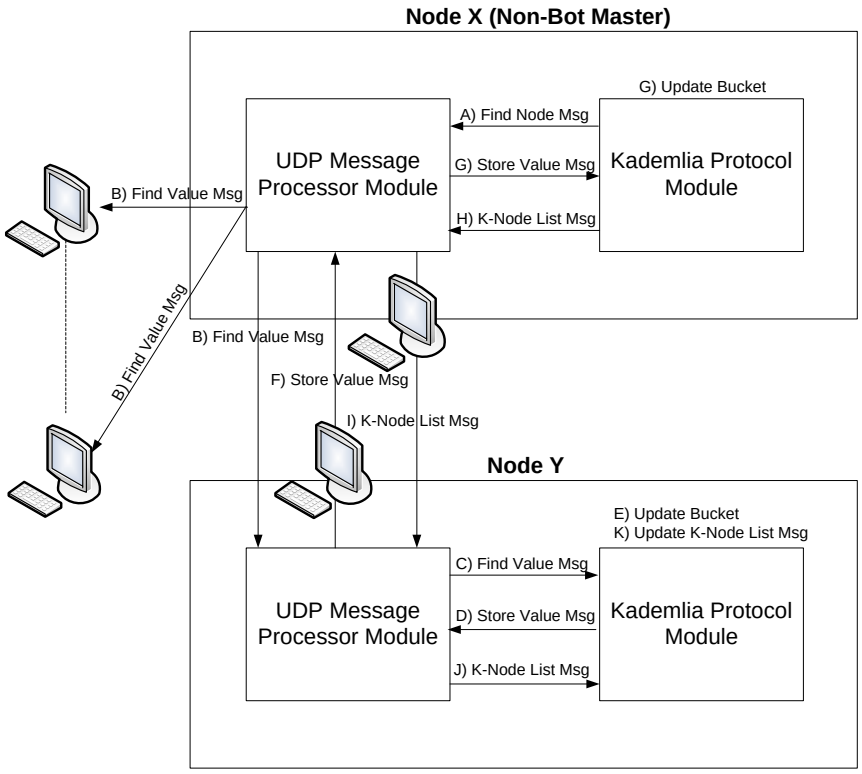

Figure 3.7: Flow Diagram of Find Node and Find Value Message with <*key, value*> pair present.

(a) At every *keyValueLookUpTime*, a non-botmaster (say X), sends a find FIND_NODE message to its UDP message processor module. This message contains *alpha* bot entries to which FIND_VALUE message should be sent. XOR operation is performed between sender's (X) bot ID and the key ID to determine the bucket number from which the *alpha* recipients are to be retrieved.

(b) For each bot ID present in FIND_NODE message, the UDP message processor module sends the FIND_VALUE message to the recipients. The FIND_VALUE message specifies the <*key, value*> pair that the sender bot

(X) is currently looking for, in addition the message contains the senders (bot's) contact details.

(c) When the recipient bot (say Y) receives the FIND_VALUE message from the sender bot (X), it forwards this message to the its Kademlia Protocol module.

(d) If the bot (Y) has the key value, then the recipient bot sends a STORE message to the sender bot (X). The Kademlia Protocol module of Y forwards this message to its UDP message processor module. The STORE message contains bots (Y) contact details as well as the the <*key, value*> pair value that the sender bot (X) is currently looking out for.

(e) The recipient's Kademlia Protocol module Y, updates the appropriate k-bucket for the sender's bot ID (X). XOR operation is performed between recipients bot ID and key ID.

(f) The recipients Y UDP message processor module sends the STORE message to the sender X.

(g) The recipient bot (X) receives the STORE message from the sender bot (Y) and forwards this message to its Kademlia Protocol module. The recipient's Kademlia Protocol module updates the appropriate k-bucket for the sender's bot ID (Y). XOR operation is performed between X's bot ID and key ID. A check is also performed to determine if the bot X is a botmaster or not. If it is a botmaster then the value of the key is not updated or stored. If the bot is a regular bot, then the <*key, value*> is stored. The bot X also sends the its k bot details (bucket to which the sender's bot details are added) are sent to its UDP message processor module.

(h) The recipient's (X) UDP message processor module then sends the message containing k contact information to the sender bot (Y). The k contact information is sent to the sender bot Y in order to impove the Botnet topology and subsequent *key* searches.

(i) The sending bot's (Y) UDP message processor module receives the message and forwards the k contact information message to the Kademlia Protocol module.

(j) Kademlia Protocol module receives the message and stores the contact in-

formation of sender bot (X), into its corresponding k-bucket by performing XOR operation between recipient's bot ID (X) and distributed key ID.

Case 2: <*key, value*> pair is not present with the recipient.

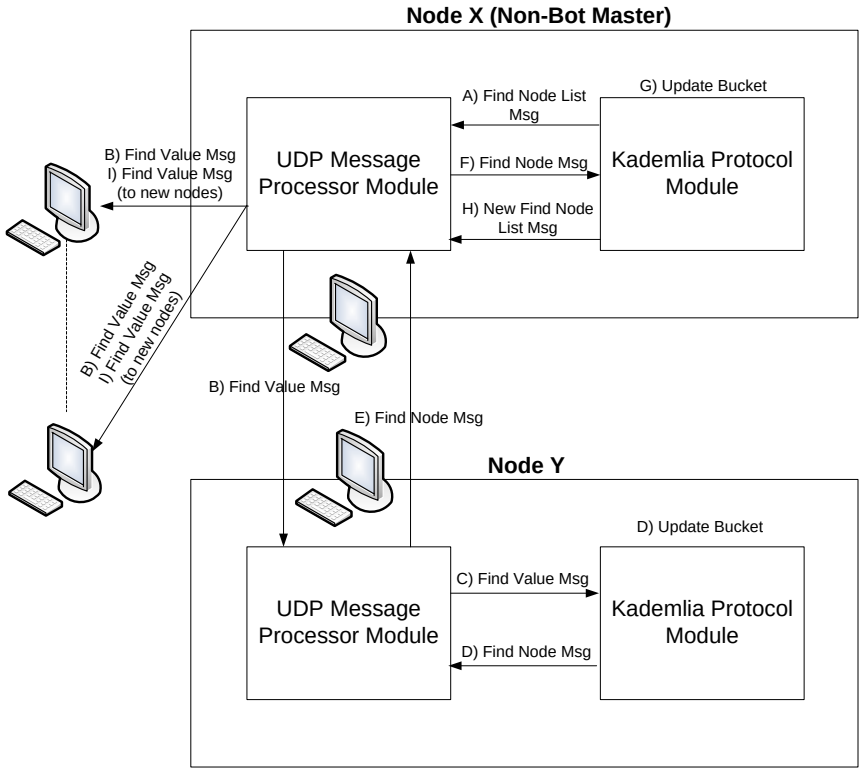

Figure 3.8: Flow Diagram of Find Node and Find Value Message with <*key, value*> pair not present.

(a) At every *keyValueLookUpTime*, a non-botmaster (say X), sends a find FIND_NODE message to its UDP message processor module. This message contains *alpha* bot entries to which FIND_VALUE message should be sent. XOR operation is performed between sender's (X) bot ID and the key ID to determine the bucket number from which the *alpha* recipients are to be retrieved.

(b) For each bot ID present in FIND_NODE message, the UDP message processor module sends the FIND_VALUE message to the recipient. The FIND_VALUE message specifies the <*key, value*> pair that the sender bot

(X) is currently looking for, in addition the message contains the senders (bot's) contact details.

(c) When the recipient bot (say Y) receives the FIND_VALUE message from the sender bot (X), it forwards this message to the its Kademlia Protocol module.

(d) If the bot Y doe not have the <*key, value*> pair, then the recipient bot Y sends a FIND_NODE message to the sender bot X. The Kademlia Protocol module of Y forwards this message to its UDP message processor module. XOR operation is performed between the recipients (Y) bot ID and key ID to determine the bucket from which the k recipients are to be retrieved. Also, the recipient's (Y) Kademlia Protocol module updates its appropriate k-bucket for the sender's bot ID (X). XOR operation is performed between recipients 128-bit bot ID and key 128-bit ID that is looked for.

(e) The recipients (Y) UDP message processor module sends the FIND_NODE message to the sender X.

(f) The recipient bot (X) receives the FIND_NODE message from the sender bot (Y) and forwards this message to its Kademlia Protocol module.

(g) The recipient's X Kademlia Protocol module updates the appropriate k-bucket for the sender's bot ID (Y). XOR operation is performed between recipient's (X) 128-bit key ID and key 128-bit ID and all the contact details present in the FIND_NODE message are added to the corresponding bucket.

(h) The recipient's bot X repeats the same process every *keyValueLookUpTime* seconds unless the <*key, value*> pair is not obtained. Even if the <*key, value*> pair is not stored in the botnet, bot's continue to search the <*key, value*> pair at every *keyValueLookUpTime* seconds. This is done as the bot's trying to retrieve the <*key, value*> pair have no information about the availability of the <*key, value*> pair.

3.2 Storm Operation

This section provides an overview on the Storm operation. The Storm consists of a number of co-operating bots that communicate with each other. Each bot present in

the network has its own ID (identifier). Bots present in the network can be categorized into two groups. Bot master and non-botmaster *i.e.,* bots. Botmaster can be defined as the attackers whose main task is to inject the <*key, value*> pair into the Internet. non-botmaster are the bots that are currently looking for the <*key, value*> pair. Each newly created bot receives a PEER LIST message from the server. The PEER LIST message has *peerSize* entries, which is used by the bot to initiate the initial look up process. The newly created bots has a fully populated list of uniformly selected bot details from the botnet. Storm bots organize their routing tables information in its k-buckets. These buckets are refreshed every *bucketRefreshTime* seconds. When a bot receives a query message from any other bot, it updates its appropriate k-bucket. Bots periodically searches for a set of keys stored in the botnet. The values associated with these keys are distributed or spread by the botmasters. To compensate for bots leaving the network, each botmaster republishes the <*key, value*> pair into the network every *sendKeyValueTime* seconds.

There are two different approaches that the Storm bots can use to retrieve <*key, value*> pair from the network. The first is to launch recursive search for the data. This is called recursive parallelism. In this method, if the data is not present with the receiving bot, it sends parallel asynchronous message to its nearest bot requesting for <*key, value*> pair. This method is recursive iterated unless the bot with <*key, value*> pair is found. Once the bot receives the value, it returns the <*key, value*> pair to the requesting bot. The second method is the iterative parallelism. For our simulations, we have implemented this method of search. In this method, if the search is not successful, the receiving bot returns k triplets to the requesting bot. These k triplets contain contact information of those bots that are closest to the key. The sequence of parallel search is continued until search time out period is reached or no bot in the set returns the value *i.e.,* the bot again sends query messages by sending FIND_VALUE messages to those bots that have already not been contacted. In order to search all the keys present in the network, the bot changes the key to be searched for every *keyRefreshTime* seconds. This is done in order to ensure that all <*key, value*> pair values are obtained. Search process begins by selecting *alpha* contacts from the bucket that are closest to the key that is currently used. If there are fewer than *alpha* bots present in the bucket, the bots selects the next available bucket. The bot then sends these parallel asynchronous FIND_VALUE messages to these *alpha* contacts every *keyValueLookUpTime* seconds and waits for its reply.

Bots get their next set of commands via a pull structure in which they search for

pre-specified keys at pre-specified times. Hence, the botmaster merely needs to inject (STORED) the desired <*key, value*> pair ahead all of the bots' automated search for it. This structure means that it is much harder to trace back where a given command was injected into the botnet, *i.e.,* it is much harder to find the botmasters. This is largely the opposite of how standard P2P networks are used. Once the bots receive their commands they act as purely independent nodes. But this is also why Storm ensures, via using Network Time Protocol (NTP), that the nodes all have similar notions of time. NTP sets their local clocks to the same values hence, they search for the same <*key, value*> pairs at more or less the same time and enact attacks, such as DDoS, at more or less the same time. Network Time Protocol does not guarantee that every bot will have the exact time but that they will have more or less the same notion of time due to its simplistic models of network delays.

In the next section, disinfection and disruption mitigation strategies have been explained. In order to differentiate between the basic growth model of the Storm Botnet and the mitigation strategies, basic birth process of the Storm is referred as Standard growth model of Storm Botnet.

3.3 Mitigation Strategies

In this section two different mitigation strategies are presented to disrupt the communication between the botmasters and bots. OMNET++ simulations are used to evaluate the effectiveness of these two mitigation mechanisms: Random disinfection mitigation strategy and Sybil distruption mitigation strategy. These are described in detail below. In the next chapter, mitigation strategy results and effectiveness of disinfection and distruption mitigation strategies have been presented. Moreover, the weakness of engineering design parameters that could disrupt the communication between the bots have been presented.

1. **Random Mitigation:** This is the simplest mitigation strategy. In this disinfection approach, the content availability of the set of keys in the P2P network is reduced by removing bots from the botnet. This method is equivalent to the random removal of bots from the botnet. The identity of the bots to be removed is not considered *i.e.,* there is no distinction made between the bots when the bot is removed. The bot to be removed from the botnet is randomly selected using uniform distribution function. The bot is completely removed from the

botnet and it is no longer part of the Internet. If the contact information of this bot is present in any other bot's k-buckets, based on the update process of the bucket, then this bots contact details will be gradually removed from the k-buckets and from the botnet.

2. **Sybil Mitigation:** In this method, the effect of Sybil attack as a disruption mitigation strategy on Storm botnet is determined. A Sybil attack is the one in which fake or bogus bots are introduced into the botnet. In the simulation, Sybil attack infiltrates the botnet with a large number of fake bots, with the intention of disrupting the communication between bots. In order to disrupt the communication between the bots, contact information of these fake bots are gradually inserted into k-buckets of regular bots gradually.

   In order to ensure that the fake bots can impact the botnet, the data associated with the *<key, value>* pair points to bogus content *i.e.,* the Sybil responds to the queries for any *<key, value>* pair with erroneous information. In the simulations, the server module (ISP Module) handles the task of generating these fake bots with unique IDs.These Sybil bots behave in the same way as that of non-Sybil bots. All the characteristics of the non-Sybil bot are applicable to the Sybil bots. In order to ensure that these Sybil bots are not recognized by the botmasters, the Sybil bots are inserted following exaclty the same approach as how non-Sybil nodes are inserted in the botnet. Only differences between a Sybil bot and a non-Sybil bot being a flag differentiating it as a Sybil bot and the fact that Sybil bots responds to the query message with erroneous information. The created Sybil bots remain active and participate in the underlying P2P Storm protocol *i.e.,* the Sybil bots use the same exponential distribution for its bots creation.

3.4 Summary

In this chapter the architecture and implementation details of the Storm botnet simulation have been outlined. The engineering design attributes used by the bot, and how the bots communicate with each other to retrieve the *<key, value>* pair from the botnet is explained. Two mitigation strategies have been presented to disrupt the communication channel between the botmaster and bots. In the next chapter, simulation results along with the data obtained to determine the effectiveness of various

engineering design parameters of Storm are presented. In addition, the effectiveness of Random disinfection strategy and Sybil disruption strategy used to disrupt the communication of the botnet has been discussed.

Chapter 4

Simulation and Analysis

Up until now, the enginnering design decisions associated with P2P botnets has not been well studied. Most of the studies have focused on the disinfection and disruption mitigation strategies. No large scale simulation based studies have been published taking into account the actual packet level network behaviour. In the previous chapter, the architecture details of Storm and the mitigation strategies that could be used to disrupt the C&C structure of botnet have been presented. In this chapter, simulation results along with the data obtained to determine the effectiveness of various engineering design parameters of Storm has been presented. The goal is to identify the botnet behaviour under different botnet setups and to show how the engineering design parameteres influence the Storm botnet.

This chapter is organized as follows. Section 4.1, describe in detail the underlying network infrastructure and the simulation scenarios based on which results are obtained. In Section 4.2, the findings of the simulation results are presented. Simulation results along with the data that is obtained for Storm's standard growth model is presented. In addition, results with respect to the disinfection and disruption strategies are also provided.

4.1 Simulation Platform and Parameters

In order to understand the technical details of the simulation set up, the basic simulation setup in discussed in Section 4.1.1. This section provides the generic description of routers connectivity. In section 4.1.2, details on simulation parameters that are used by the server and peers is provided. The last subsection, provides detailed de-

scription on the simulation scenarios that are used to simulate the Storm standard growth model and mitigation strategies.

4.1.1 Basic Simulation Setup

For the simulations, the Storm botnet is divided into a number of service providers and routers. The storm botnet contains 20 routers which are randomly connected to each other, with a degree of connectivity of 2 or 3. In addition, the botnet consists of 10 service providers each of which is connected to a router. The numbers of bots present in the botnet is fixed and are equally divided among the service providers. The simulated storm networks have a total of 81000 and 40500 bots. For example, if the simulation is done for 81000 bots, then each service provider would have a maximum of 8100 active bots at any point of time during the simulation. Due to memory constraint, the storm botnet could be simulated with 81000 bots. In order to determine the impact of Storm size, the number of bots present in the botnet is reduced to half *i.e.,* 40500. All the routers implement the OSPF (Open Shortest Path First) routing algorithm with RED (Random Early Detection) Queuing technique for routing, as provided by the INET framework. In order to simulate the network scenario, a network delay is introduced when messages are transferred between the bots. This delay is introduced by default by the OSPF routing algorithm. Due to the fact that there are many simulations parameters that are analyzed for the Storm, all the simulation results are limited to this one network scenario.

4.1.2 Simulation Parameters

In the previous chapter, all the attributes with respect to the bot were outlined. These attribtues can be placed in two categories: (a) those with fixed values (Table 4.1), and (b) those with configurable values (Table 4.2). The performance of the Storm botnet against different Table 4.2 parameter settings is explored via the constructed simulator.

4.1.2.1 Attributes with constant values

Table 4.1 provides the constant values of the attributes that are used by the botnet overlay protocol. As explained earlier, the network consists of 10, *maxServers*, for all our simulations. Each server is responsible for creating a specific number of bots based

Attribute	Value
k	20
alpha	3
keyLength	128
maxServers	10
sendKeyValueTime	3600 sec
bucketRefreshTime	3600 sec
keyRefreshTime	1350 sec
tBotCreation	exponential(5)
key	f6abc415f098c7dc6924e524bcacde24
value	abd1f098c456df232890cbf0a3490f3f

Table 4.1: Attributes with Fixed Values.

on the *maxBots* parameter. Every bot participating in the network generates an ID of length *keyLength* bits, when it first joins the network. Bots are added into the botnet based on the an exponential distribution parameter *tBotCreation*. The birth rate of the bots follows an exponential distribution. At every exponential *tBotCreation* time, a newly created bot with fully populated list of peers is added into the botnet. The bot transmits its bot ID with every message it sends. The value of the attributes *k* and *alpha* is set to 20 and 3 respectively, based on the Kademlia paper [21]. Storm network generates 32 different 128-bit keys each day using a random function *f(d, r)* which takes as inputs the current day *d* and a random number *r* between 0 and 31 [5]. Due to the fact that different <*key, value*> pairs are generated every day and bots look up for these newly generated <*key, value*> pair every day, simulation time is restricted to 24-hours.

There are few more constant attributes that are assumed. There are 32 different <*key, value*> pairs used by the Storm. Due to the fact that without the loss of generality, the performance fo the botnet can be evaluated with respect to a small number of Storm <*key, value*> pair, the simulation is specific to one <*key, value*> rather than 32 different <*key, value*> pairs. The <*key, value*> pair value are randomly selected and are set to "f6abc415f098c7dc6924e524bcacde24" and "abd1f098c456df232890cbf0a3490f3f" respectively. The *keyRefreshTime* attribute specifies the time instance at which bots look up for the new <*key, value*> pair. This value is assumed the value to be 1350 seconds. This means that 32 different keys are searched twice in a day by each bot present in the network. The next attribute is the *bucketRefreshTime* time. As explained in the previous chapter, every hour the

flag contents of k-bucktes are removed. The last constant attribute is the *sendKeyValueTime*, time instance at which the botmasters inject the <*key, value*> pair into the botnet. The task of botmasters in the botnet is to distribute the <*key, value*> pair in the network at every *sendKeyValueTime* seconds. This value is set to 3600. Kademlia paper [21] states that nodes republishes the <*key, value*> pair into the botnet once every hour.

4.1.2.2 Varying Attributes

Attribute	**Value**
mitigationSybil	true, false
maxBots	4050, 8100
initialNoBots	1%, 20%
peerListSize	200, 300
maxBotMasters	5, 10
tBotRemoval	NoRemoval, exponential(25)
keyValueLookUpTime	225, 900, 3600

Table 4.2: Simulation Attributes with Varying Values.

Table 4.2, shows those attribute with configurable values.

1. The attribute *mitigationSybil* categorizes a bot as a Sybil bot or normal bot. If the attribute value is set *true*, all the bots created by the server are Sybil bots and vice versa. For the standard growth model of the Storm, *mitigationSybil* value is set to *false*. In this case, the server acts as normal server. Whereas, in case of Sybil mitigation strategy this value is set to *true*.

2. *maxBots* attribute specifies the maximum number of bots that are created by the servers during the simulation. The total number of bots present in the network is equal to *maxBots* * *maxServers*. As explained earlier, for the simulation, simulated networks have a total of 81000 and 40500 bots.

3. *initialNoBots* attribute specifies the initial percentage of bots created by the server at the start of the simulation. The possible values for this attribute for the simulations are 1% and 20% of the total number of bots.

4. In order to determine the effect of peer list size on standard growth model of Storm and the mitigation strategies, the possible values selected for the *peerListSize* attribute set to 200 and 300. The number varies with different version of the Storm binary.

5. *maxBotMaster* attribute specifies the total number of botmasters present in the botnet. The botmasters are randomly spread among the servers. In order for the botmaster to remain invisible in the Internet, the total number of botmasters present in the system must be small. For the reason, *maxBotMaster* parameter value is chosen to be 5 and 10.

6. *tBotRemoval* attribute specifies the removal percentage of bots from the network. For the standard growth model, *tBotRemoval* parameter is set to 0 as in this model no bots are removed from the network. Value of Exponential (25) constitutes the 10% removal of bots from the network and is used for the disinfection process.

7. In order to retrieve the <*key, value*> pair from the network, a bot searches for the key every *keyValueLookUpTime* seconds. The bot sends FIND_VALUE search queries to *alpha* bots into the botnet. The possible values for the simulation are 225, 900 and 3600 seconds. 3600 seconds corresponds to one time lookup every hour, where as 225 corresponds to 16 lookup every hour. In order to determine the effect of *keyValueLookUpTime*, the search for the <*key, value*> pair is done with different values.

4.1.3 Simulation Scenarios

In order to analyze bot attributes, mitigation strategies and determine how botnet is affected in general, simulation scenario are categorized into three categories: (a) standard growth model, (b) disinfection mitigation strategies, and (c) disruption mitigation strategies. These scenarios cover all the possible combinations of the attributes as specified in the table 4.2. For each combination of the parameters, 6 simulation runs are performed with different seeds. Due to time constraints, only 6 simulation are runs for different combinations of parameters. However, we have found out that for each data set the difference among data for different seeds is not statistically significant as they fall within a 95% confidence interval.

4.2 Simulation Results

In this section, detailed analysis and discussion of the simulation results along with the data that is obtained for Storm's standard growth model and mitigation strategies have been provided. Rather than providing results separately for all the attributes, results are specified with respect to Storm's standard growth model and mitigation strategies. The analysis of the simulations results addresses the following:

1. The effect of bots' peer-list size on the <*key, value*> pair retrieval.
2. The impact of initial number of bots on <*key, value*> propagation and retrieval.
3. The impact of the number of botmasters in the botnet.
4. The mean time required by the bot to obtain a <*key, value*> pair based on the botnets design attributes.
5. The impact of size of the botnet.
6. Percentage of bots that received the <*key, value*> pair verses time.
7. Effect of engineering design parameters on the botnet load and visibility.
8. Mean number of messages a bot sends to obtain the <*key, value*> pair from the network.

In the following sections, some entries of the tables have been highlighted. The entries in bold signify the maximum values present in the table whereas, minimum values have been highlighted using italics.

4.2.1 Analysis based on <*key, value*> pair count

Table 4.3 and 4.4 summarizes the number of bots that retrieve the <*key, value*> pair value as the bots search for a key during the 24 hour simulation period for peer list sizes of 200 and 300 respectively. Tables 4.5 and 4.6 depict the same scenario in terms of percentage of bots that retrieve the <*key, value*> value pair.

Figures 4.1 and 4.2 show the result for Standard growth model of Storm with different combinations of parameters for a botnet size of 81000 bots. Based on the different combination of parameters, Figures 4.1 and 4.2 depict that the maximum number of bots that retrieve the <*key, value*> pair from the botnet is around 91%

Peer Size = 200			Bot Masters 5				Bot Masters 10			
Key Value Look Up Time (sec)	Botnet Size	Strategy	Bot Removal 0%		Bot Removal 10%		Bot Removal 0%		Bot Removal 10%	
			Initial Bots: 1%	Initial Bots: 20%	Initial Bots: 1%	Initial Bots: 20%	Initial Bots: 1%	Initial Bots: 20%	Initial Bots: 1%	Initial Bots: 20%
225	81000	Growth Model	68243	62313	65435	67619	**68935**	62469	65681	67729
		Sybil (True Value)	31586	30151	31574	31589	31882	30775	**32525**	32138
		Sybil (Any Value)	60893	58661	59018	58857	60954	58727	58969	58980
225	40500	Growth Model	30145	29769	26605	26900	30010	29775	26506	27803
		Sybil (True Value)	13805	13412	11725	12405	14850	13828	12181	12426
		Sybil (Any Value)	26951	26376	23418	24200	27653	26324	23447	24295
900	81000	Growth Model	67157	61597	62136	66642	67001	61729	64227	66484
		Sybil (True Value)	31357	30777	28122	29085	32095	31340	31524	28558
		Sybil (Any Value)	60287	58698	54612	55766	60269	58720	57986	54654
900	40500	Growth Model	29254	28958	25030	26677	28958	28838	25268	25674
		Sybil (True Value)	13585	12976	10830	11506	13862	13269	12087	13117
		Sybil (Any Value)	26242	25660	21783	22831	26263	25310	23020	24226
3600	81000	Growth Model	66603	61322	57826	62171	66865	61229	63475	65926
		Sybil (True Value)	31333	30675	30736	28711	31802	31189	31465	31439
		Sybil (Any Value)	59734	58359	56619	54692	59623	58344	56815	56751
3600	40500	Growth Model	28371	28254	23167	25652	28657	28264	24187	25920
		Sybil (True Value)	13088	12648	*10795*	11843	13830	13339	12190	12613
		Sybil (Any Value)	25425	24719	*21110*	22581	25521	24651	21931	23192

Table 4.3: Number of bots with <*key, value*> pair for peer list size 200

Peer Size = 300			Bot Masters 5				Bot Masters 10			
Key Value Look Up Time (sec)	Botnet Size	Strategy	Bot Removal 0%		Bot Removal 10%		Bot Removal 0%		Bot Removal 10%	
			Initial Bots: 1%	Initial Bots: 20%	Initial Bots: 1%	Initial Bots: 20%	Initial Bots: 1%	Initial Bots: 20%	Initial Bots: 1%	Initial Bots: 20%
225	81000	Growth Model	73434	72291	64174	64255	73215	**73668**	66546	68297
		Sybil (True Value)	31774	35688	32199	32110	31164	35579	32747	32771
		Sybil (Any Value)	61320	63828	59961	59787	60083	63599	59813	60008
225	40500	Growth Model	30083	30397	27378	27738	30237	31195	27351	26731
		Sybil (True Value)	13869	13405	11993	12655	13899	13581	13137	13706
		Sybil (Any Value)	27087	26642	24051	25215	27276	26861	24699	26148
900	81000	Growth Model	73191	72739	63090	64041	72908	72721	65512	67573
		Sybil (True Value)	32101	**35991**	31713	30011	31707	35896	32476	32182
		Sybil (Any Value)	61132	63729	58700	56854	60802	63672	58823	58593
900	40500	Growth Model	29197	29591	25070	25776	29396	30739	26128	27446
		Sybil (True Value)	13691	13109	10993	12912	13992	13572	12718	12270
		Sybil (Any Value)	26437	25811	22214	24603	26812	26042	23800	23714
3600	81000	Growth Model	72598	72887	64143	66142	71905	72984	64427	66793
		Sybil (True Value)	32010	35548	29329	31571	31802	31189	31465	31439
		Sybil (Any Value)	60056	63510	55300	57507	60131	63487	57243	57447
3600	40500	Growth Model	28680	28566	22268	24752	28810	28685	25289	26698
		Sybil (True Value)	13476	13192	11091	*10765*	13613	13411	12182	11957
		Sybil (Any Value)	25726	24858	21955	*21652*	25684	25205	22647	22714

Table 4.4: Number of bots with <*key, value*> pair for peer list size 300

Peer Size = 200			Bot Masters 5				Bot Masters 10			
			Bot Removal 0%		Bot Removal 10%		Bot Removal 0%		Bot Removal 10%	
Key Value Look Up Time (sec)	Botnet Size	Strategy	Initial Bots: 1%	Initial Bots: 20%	Initial Bots: 1%	Initial Bots: 20%	Initial Bots: 1%	Initial Bots: 20%	Initial Bots: 1%	Initial Bots: 20%
225	81000	Normal Infection	84.25	76.90	80.78	83.48	**84.70**	77.12	81.08	83.61
		Sybil (True Value)	38.99	37.67	38.98	38.99	39.36	37.99	**40.15**	39.67
		Sybil (Any Value)	75.17	72.42	72.86	72.66	75.25	72.50	72.80	72.81
225	40500	Normal Infection	74.43	73.50	65.69	66.42	74.10	73.50	65.44	68.65
		Sybil (True Value)	34.08	33.11	28.97	30.63	36.66	34.14	30.07	30.68
		Sybil (Any Value)	66.54	65.12	53.78	59.75	68.28	64.99	57.90	59.98
900	81000	Normal Infection	82.91	76.04	76.70	82.27	82.71	76.20	79.29	82.07
		Sybil (True Value)	38.71	37.99	34.71	35.90	39.62	38.69	38.91	35.25
		Sybil (Any Value)	74.42	72.46	67.42	68.86	74.40	72.49	71.58	67.47
900	40500	Normal Infection-Half	72.23	71.51	61.80	65.87	71.50	71.20	62.39	63.39
		Sybil (True Value)	33.50	32.05	26.74	28.41	34.23	32.76	29.85	32.38
		Sybil (Any Value)	64.79	63.35	53.78	56.37	64.84	62.49	56.84	59.81
3600	81000	Normal Infection	82.25	75.70	71.39	76.75	82.54	75.59	78.36	81.39
		Sybil (True Value)	38.68	37.87	37.94	35.44	39.26	38.50	38.84	38.81
		Sybil (Any Value)	73.74	72.04	69.90	67.52	73.60	72.03	70.14	70.06
3600	40500	Normal Infection-Half	70.05	69.76	57.20	63.33	70.50	69.78	59.72	64.00
		Sybil (True Value)	32.14	31.23	*26.66*	29.24	34.15	32.93	30.00	31.15
		Sybil (Any Value)	62.77	61.03	*52.12*	55.75	63.01	60.56	54.15	57.24

Table 4.5: Percentage of bots with <*key, value*> pair for peer list size 200

Peer Size = 300			Bot Masters 5				Bot Masters 10			
Key Value Look Up Time (sec)	Botnet Size	Strategy	Bot Removal 0%		Bot Removal 10%		Bot Removal 0%		Bot Removal 10%	
			Initial Bots: 1%	Initial Bots: 20%	Initial Bots: 1%	Initial Bots: 20%	Initial Bots: 1%	Initial Bots: 20%	Initial Bots: 1%	Initial Bots: 20%
225	81000	Growth Model	90.65	89.24	79.22	79.37	90.38	**90.94**	82.15	84.31
		Sybil (True Value)	39.22	44.05	39.75	39.64	38.47	43.92	40.42	40.45
		Sybil (Any Value)	75.70	78.80	74.02	73.81	74.17	78.51	73.84	74.08
225	40500	Growth Model	74.27	75.05	67.60	68.49	74.65	77.02	67.53	66.05
		Sybil (True Value)	34.25	33.09	29.61	31.24	36.66	34.14	30.17	30.68
		Sybil (Any Value)	66.88	65.78	59.38	62.25	67.35	66.32	60.98	64.56
900	81000	Growth Model	90.35	89.70	77.88	79.06	90.01	89.77	80.87	83.42
		Sybil (True Value)	39.63	**44.43**	39.22	37.05	39.14	44.31	40.09	39.73
		Sybil (Any Value)	75.47	78.67	72.46	70.19	75.06	78.60	72.62	72.33
900	40500	Growth Model	72.09	73.06	61.90	63.64	72.58	75.90	64.50	67.76
		Sybil (True Value)	33.80	32.36	27.14	31.88	34.23	32.76	29.85	32.38
		Sybil (Any Value)	65.28	63.73	54.85	60.74	66.20	64.30	58.76	58.55
3600	81000	Growth Model	89.62	89.98	78.18	81.65	88.77	90.10	79.54	82.46
		Sybil (True Value)	39.51	43.88	36.20	38.97	39.12	44.22	38.09	39.97
		Sybil (Any Value)	74.14	78.40	68.27	70.99	74.23	78.37	70.67	70.92
3600	40500	Growth Model	70.81	70.53	54.89	61.11	71.13	70.82	62.44	65.92
		Sybil (True Value)	33.27	32.57	27.38	*26.58*	33.61	33.11	30.08	29.52
		Sybil (Any Value)	63.52	61.37	54.21	*53.46*	63.41	62.23	55.91	56.08

Table 4.6: Percentage of bots with <*key, value*> pair for peer list size 300

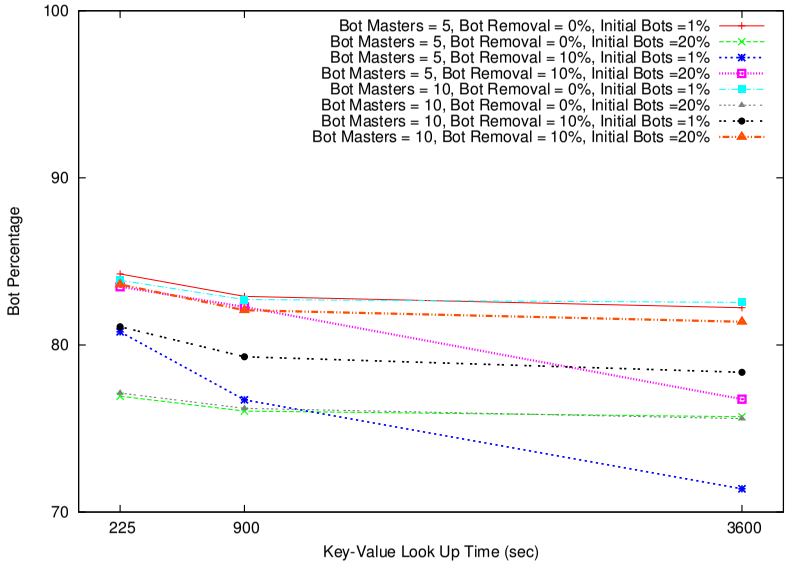

Figure 4.1: Standard Growth Model for 81000 bots: Percentage of bots with <*key, value*> pair for peer list size of 200.

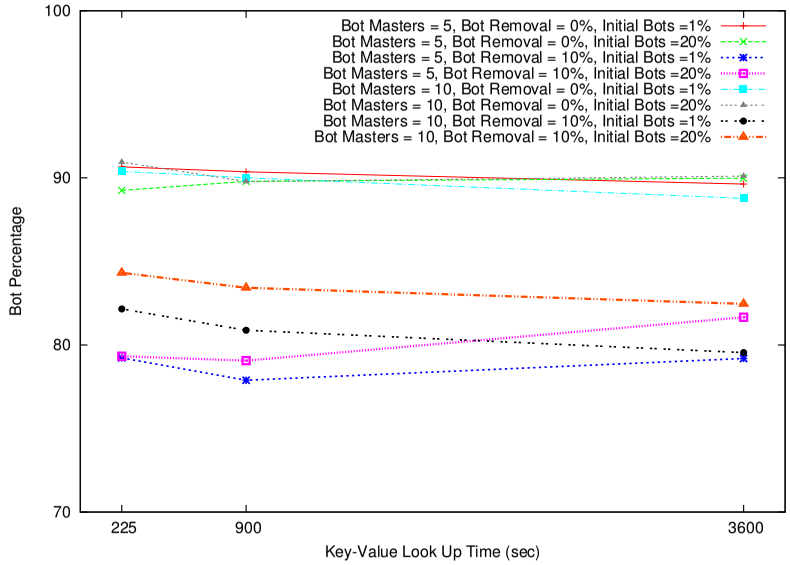

Figure 4.2: Standard Growth Model for 81000 bots: Percentage of bots with <*key, value*> pair for peer list size of 300.

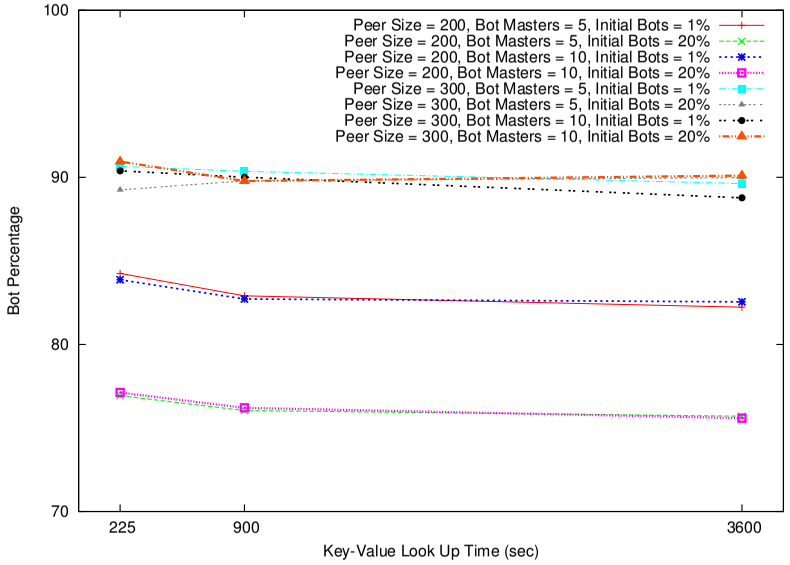

Figure 4.3: Standard Growth Model for 81000 bots: Percentage of bots that retrieve the <*key, value*> pair without random disinfection strategy.

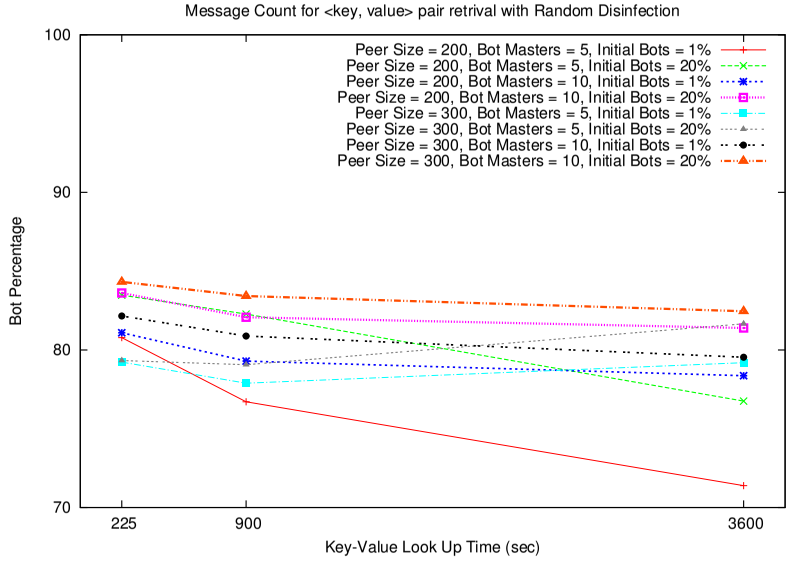

Figure 4.4: Standard Growth Model for 81000 bots: Percentage of bots that retrieve the <*key, value*> pair with random disinfection strategy.

whereas, the minimum number of bots that retrieve the <*key, value*> pair is around 71%. An important observation is that the *keyValueLookUpTime* parameter does not impact the bot count. Even when the <*key, value*> pair value is searched every 225 seconds (16 times in an hour) the number of bots that receive the <*key, value*> pair is almost equal to that when the search query message is sent every hour. This is due to the fact that every bot refreshes the flag contents of each k-bucket every hour and given the fact that each bot organizes its contacts into k-buckets, only the last few k-buckets have contact information other bots. The Figure also reveals that number of botmasters do not impact the botnet. Comparison of a botnet having 5 botmasters with that of a botnet having 10 botmasters indicate that percentage of bots that retrieve the <*key, value*> pair fall approximately within the 71% to 91% range.

Comparison of Figures 4.1 and 4.2 indicate the impact of the peer size on the standard growth model of Storm. Figure 4.1 indicate that for a peer list size of 200, the maximum number of bots that retrieve the <*key, value*> pair from the botnet is around 84% whereas, the minimum number of bots that retrieve the <*key, value*> pair is around 71%. Moreover, Figure 4.2 indicate that for a peer list size of 300, the maximum number of bots that retrieve the <*key, value*> pair from the botnet is around 91% whereas, the minimum number of bots that retrieve the <*key, value*> pair is around 77%. It can be clearly seen that, the percentage of bots that received the <*key, value*> pair with peer size 300 is higher by approximately 8% as compared to that of peer list size of 200. To summarize, it can be concluded from these Figures that in order for the bot to remain invisible during the key search process, the number of search queries sent by the bot should be minimal as this reduces the network load induced by the Storm.

Figures 4.3 and 4.4 demonstrate the impact of the random disinfection strategy. Figure 4.4 indicate that with the random mitigation strategy, the percentage of nodes that receive the <*key, value*> pair ranges between 71%- 84%, whereas, for the Standard growth model of Storm, this range is approximately between 77%-91%. Figure 4.4 also indicate that with random disinfection strategy, there is less variance with respect to the number of bots that receive <*key, value*> pair as compared to with the standard growth model. For different combination of parameters such as number of botmasters, peer list size, initial percentage of bots and look up time, variation of the number of bots that retrieve the <*key, value*> from the botnet is around 4% to 5%. To summarize we can conclude that a random disinfection strategy having 10%

removal percentage of bots from the botnet, the number of nodes that receive the <*key, value*> pair is reduced by approximately 5%-7%.

Figures 4.5 and 4.6 show the results for standard growth model of Storm with different combination of parameter for a botnet size of 40500 bots. The number of bots present is reduced to half in order to determine the effect of botnet size. Comparison of Figures 4.5 and 4.6 with 4.1 and 4.2, indicates that when botnet size is reduced by half, percentage of nodes that receive the <*key, value*> pair is reduced from 71%-90% to approximately 54%-77%. These Figures also indicate that when the botnet size is reduced to half, parameters such as initial number of nodes, peer list size and number of botmasters present in the botnet do not effect the number of bots that retreive the <*key, value*> pair.

Comparison of Figure 4.7 with 4.8, indicates the effect of removal disinfection strategy on a botnet size of 40500 bots. Even with a 10% removal of bots from the network, the percentage of bots that receive the <*key, value*> pair is reduced from 70%-77% to 54%-69%. Comparison of Figures 4.3 and 4.4 with 4.7 and 4.8, indicate that when the botnet size is reduced to half, random disinfection strategy having 10% removal of bots from the network, does not significantly impact on the system. The percentage of bots that retrieve the <*key, value*> is reduced by approximately 8% to 15%. In addition, the Figures also indicate that parameters such as number of botmasters, initial number of bots, peer list size and key value look up time have no impact on the network. Based on the results obtained, it can conclude that random disinfection strategy is not even effective for a smaller size of botnet.

Figures 4.9, 4.10, 4.11 and 4.12 show the results of Sybil Mitigation strategy for different combinations of parameters for peer list size of 200 and 300 respectively. Figures 4.9 and 4.10 specify the number of bots that receive the true <*key, value*> pair, whereas Figures 4.11 and 4.12 indicate the number of bots that receive any <*key, value*> pairs for peer list size of 200 and 300 respectively. Figures 4.9 and 4.10 indicates that the maximum number of bots receive the true <*key, value*> pair ranges from 34% to 44%. Moreover, the Figures also indicate that with only Sybil mitigation (with no random removal of nodes), this range is between 38%-44%. Comparison of Figures 4.1 and 4.2 with 4.9 and 4.10 indicate that with the introduction of 10% Sybil bots in the network, the percentage of bots that receive the true <*key, value*> pair reduces from 71%-91% to 34%-44% *i.e.,* reduction by approximately 37%-47% in the number of bots that retrieve the true <*key, value*> pair. This is due to the fact that a bot initiating a key search process, stops the look up process only after

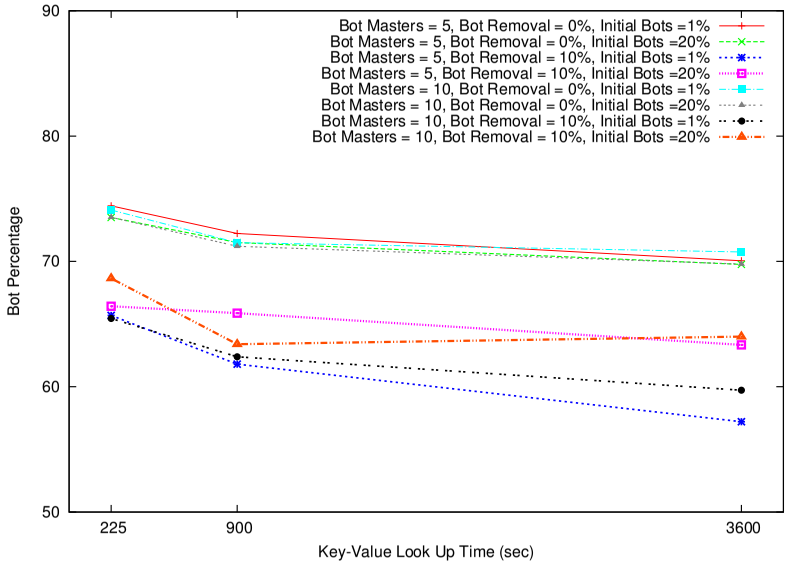

Figure 4.5: Standard Growth Model for 40500 bots: Percentage of bots with <*key, value*> pair for peer list size of 200.

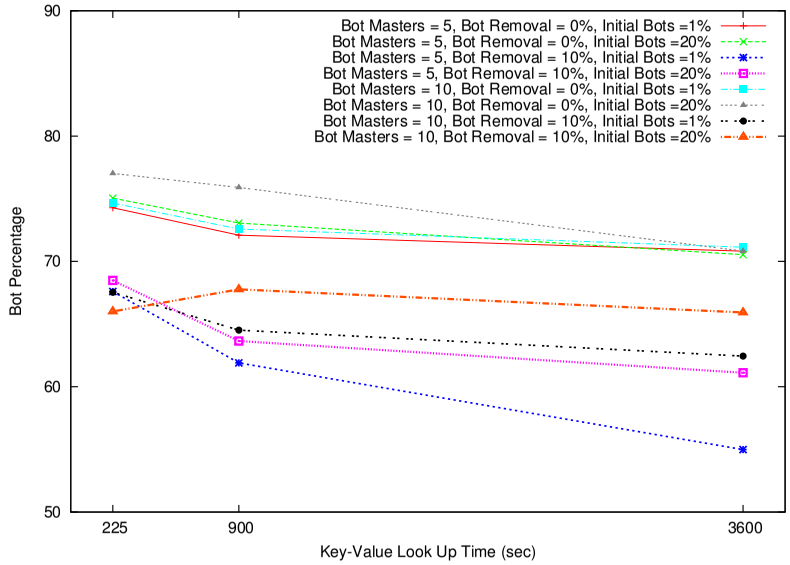

Figure 4.6: Standard Growth Model for 40500 bots: Percentage of bots with <*key, value*> pair for peer list size of 300.

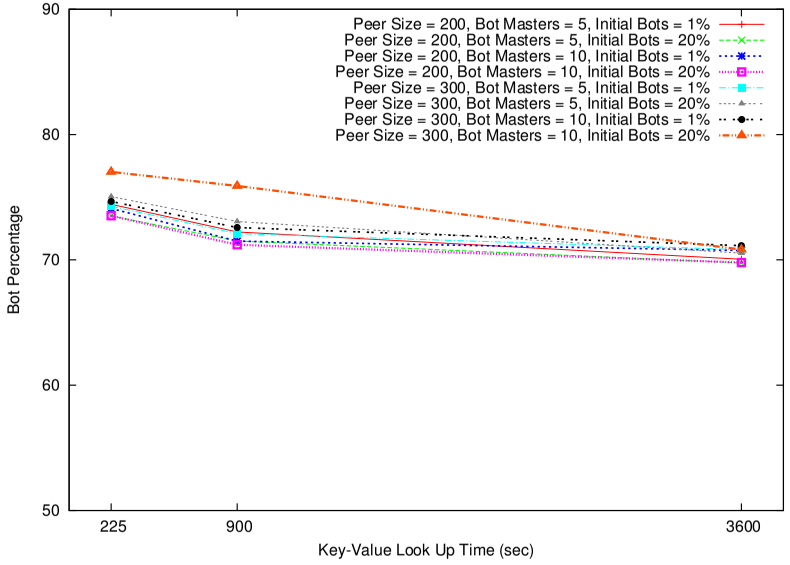

Figure 4.7: Standard Growth Model for 40500 bots: Percentage of bots that retrieve the <*key, value*> pair without random disinfection strategy.

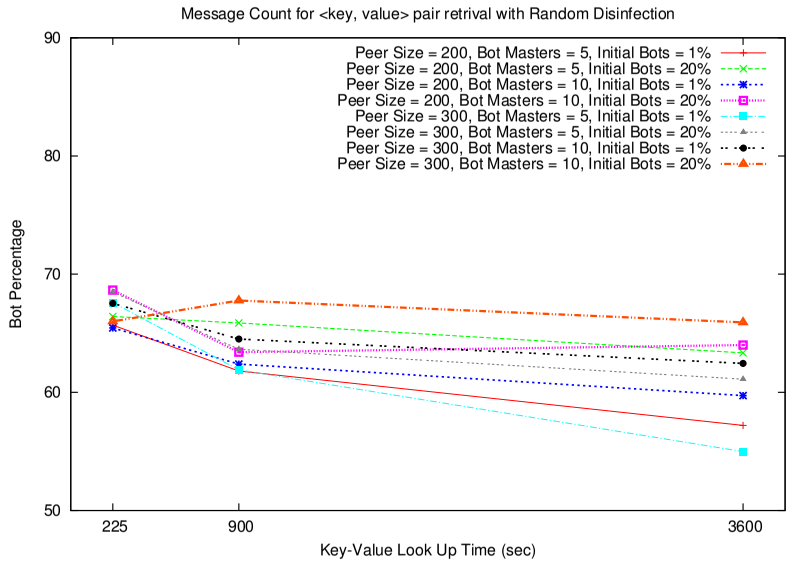

Figure 4.8: Standard Growth Model for 40500 bots: Percentage of bots that retrieve the <*key, value*> pair with random disinfection strategy.

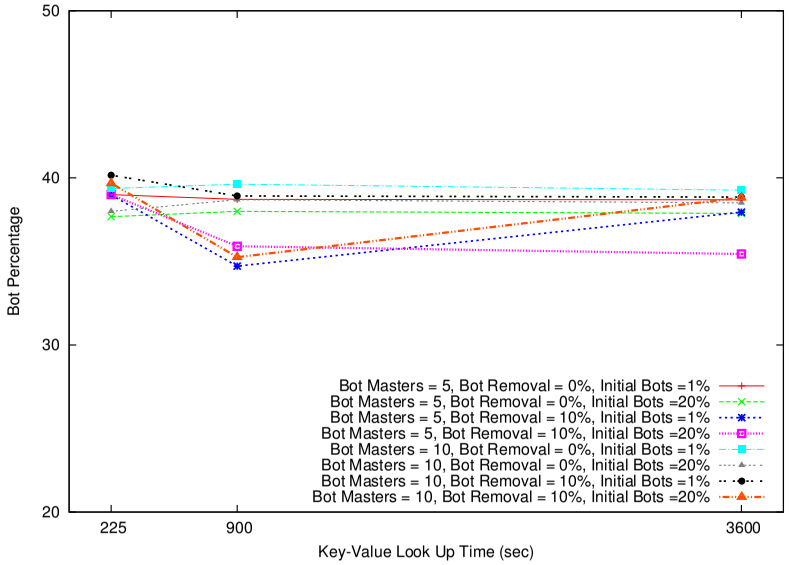

Figure 4.9: Sybil Mitigation for 81000 bots: Percentage of bots that retrieve true <*key, value*> pair for peer list size of 200.

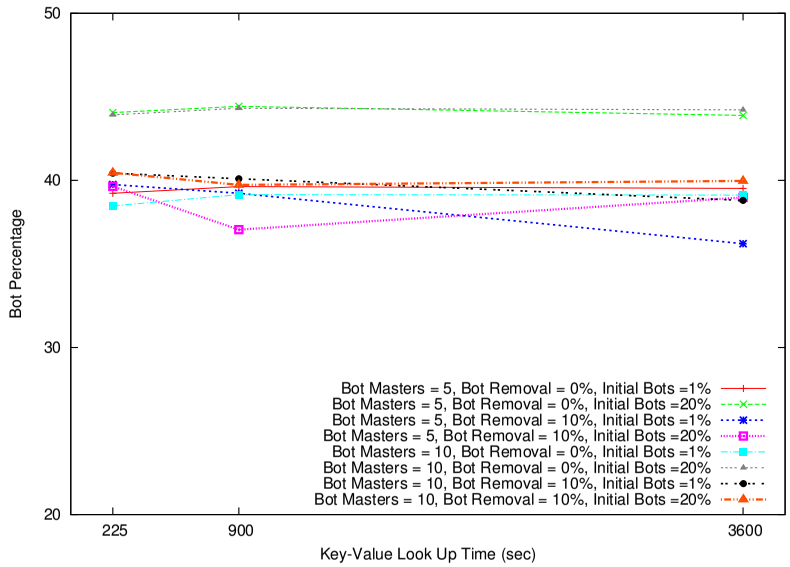

Figure 4.10: Sybil Mitigation for 81000 bots: Percentage of bots with true <*key, value*> pair for peer list size of 300

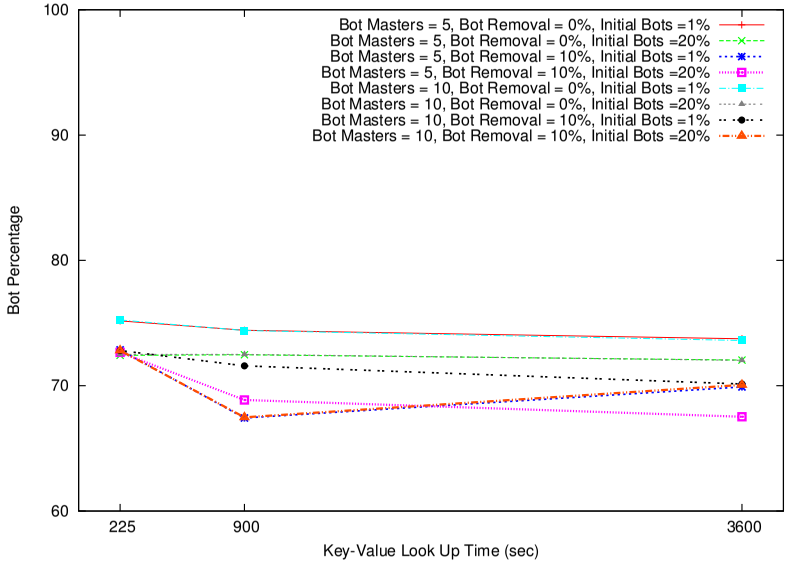

Figure 4.11: Sybil Mitigation for 81000 bots: Percentage of bots with any <*key, value*> pair for peer list size of 200.

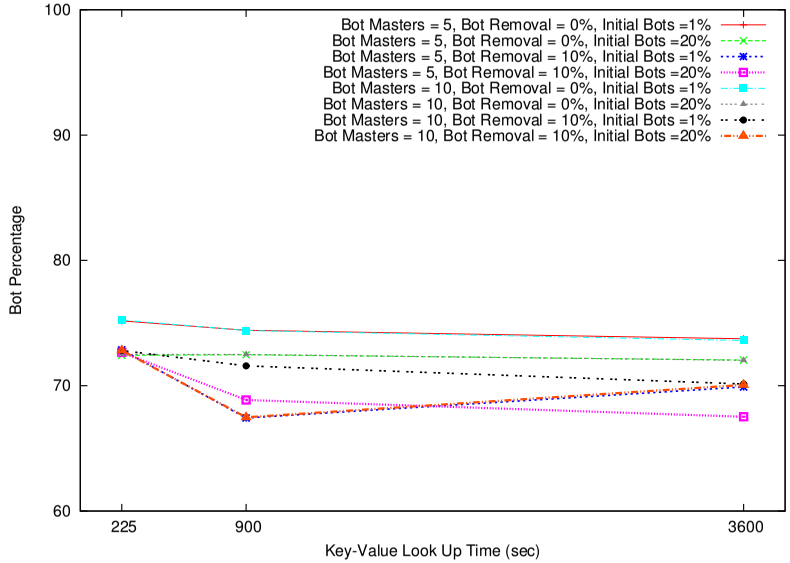

Figure 4.12: Sybil Mitigation for 81000 bots: Percentage of bots with any <*key, value*> pair for peer list size of 300.

the retrieval of the <*key, value*> pair. After the retrieval of the <*key, value*> pair, only taks of the bot is to responds to other bots that request for <*key, value*> pair value. Comparison of Figures 4.1 and 4.2 with 4.11 and 4.12 indicate that with the introduction of 10% Sybil bots into the botnet, the number of bots that retrieve any <*key, value*> pair from the botnet is reduced. The percentage of bots that receive any <*key, value*> pair is reduced from 71%-91% to 67%-79%.

Figures 4.13, 4.14, 4.15 and 4.16 show the results of Sybil Mitigation strategy for different combinations of parameters for a botnet size of 40500. Figures 4.13 and 4.14 specify the number of bots that receive the true <*key, value*> pair, whereas Figures 4.15 and 4.16 indicate the number of bots that receive any <*key, value*> pair for peer list size of 200 and 300 respectively. Figures 4.13 and 4.14 indicate that the maximum number of bots that receive the true <*key, value*> pair ranges from 26%-36%. Comparison of Figures 4.5 and 4.6 with 4.13 and 4.14 indicate that with the introduction of 10% Sybil bots in the botnet, the percentage of bots that receive the true <*key, value*> pair reduces from 54%-77% to 26%-37% *i.e.,* a reduction by approximately 28%-40% in the number of bots that receive the <*key, value*> pair. Comparison of Figure 4.5 and 4.6 with 4.15 and 4.16 also indicates that the total number of bots that receive any <*key, value*> pair in both the scenarios have a similar <*key, value*> pair count. This is due to the fact that a node initiating a key search, stop the look up process when it retrieves the <*key, value*> pair. After the retrieval of the <*key, value*> pair, only taks of the bot is to responds to other bots that request for <*key, value*> pair value.

With respect to the bot parameters, Storm attributes do not effect the total number of bots that receive the <*key, value*> pair. Introduction of random disinfection method along with Sybil mitigation further reduces the actual <*key, value*> pair count and also either the Sybil or actual <*key, value*> pair count.

Based on these observations it can be concluded that the Sybil distruption strategy is more effective when compared to the random mitigation even for a small sized network. This is due to the fact that as the number of bots grows in the botnet, the Sybil bot is able to insert its contact information in other bots k-bucket.

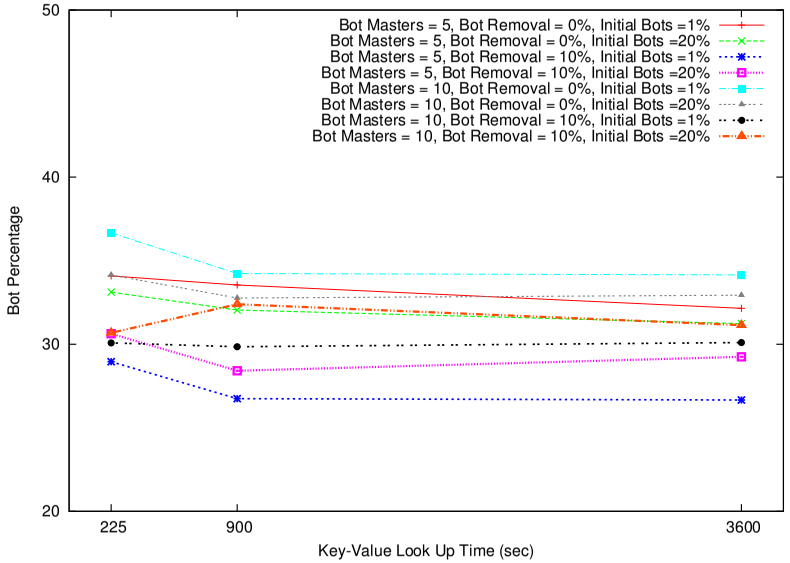

Figure 4.13: Sybil Mitigation for 40500 bots: Percentage of bots with true <*key, value*> pair for peer list size of 200.

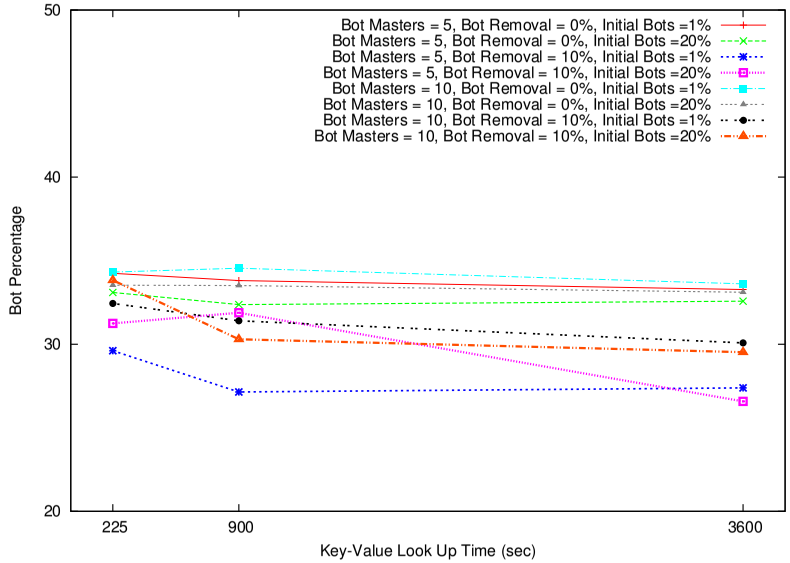

Figure 4.14: Sybil Mitigation for 40500 bots: Percentage of bots with true <*key, value*> pair for peer list size of 300.

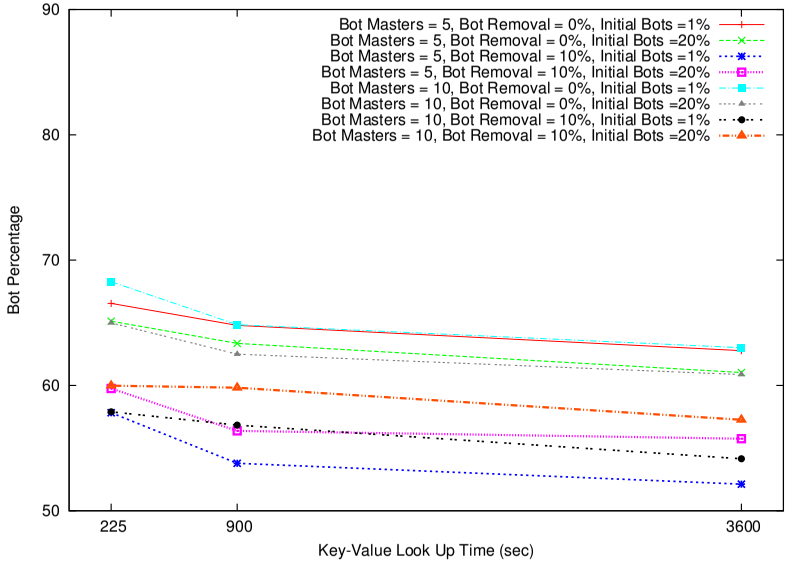

Figure 4.15: Sybil Mitigation for 40500 bots: Percentage of bots with any <*key, value*> pair for peer list size of 200.

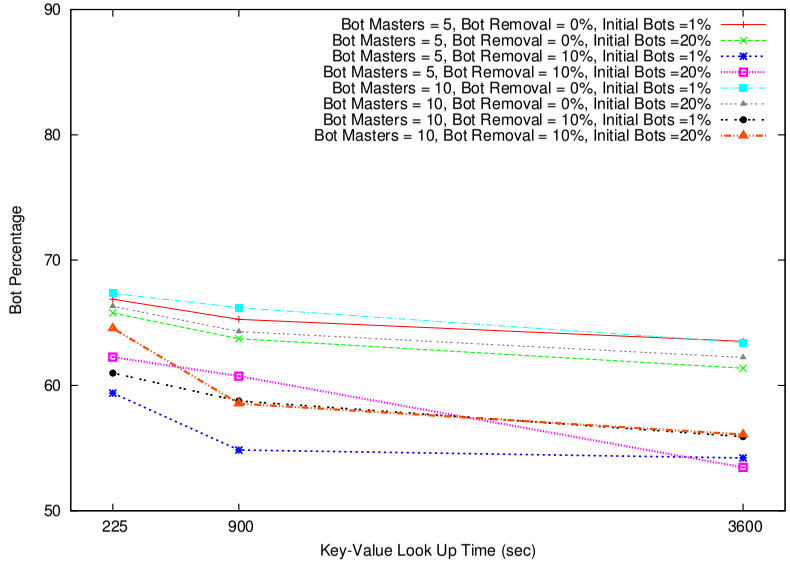

Figure 4.16: Sybil Mitigation for 40500 bots: Percentage of bots with any <*key, value*> pair for peer list size of 300.

4.2.2 Analysis based on <*key, value*> pair retrieval time

Tables 4.7 and 4.8 show the time required by the bots in seconds to retrieve the <*key, value*> pair value as the bot searches for a key during the 24 hour simulation period for peer list sizes of 200 and 300 respectively.

Figures 4.17 and 4.18 depict the mean time required by the bot to retrieve the <*key, value*> pair for different combinations of parameters for peer list sizes of 200 and 300 respectively. The maximum number of nodes present in the system is 81000. Based on the combinations of parameters, Figures 4.17 and 4.18 show that the minimum amount of time required by the bot to retrieve the <*key, value*> is around 72 minutes and the maximum time required is around 102 minutes. This is due to the fact that the botmasters inject the <*key, value*> pair into the botnet at every hour, thus the mean amount of time required by the bots to retrieve the <*key, value*> pair is greater. Based on the Figures, it is clear that key retrieval time is dependent on the parameter *keyValueLookUpTime*. Figures indicate that the time required by the bot to retrieve the <*key, value*> pair is less when the search is done every 225 seconds. This is due to the fact that each bot (those searching for the value) sends FIND_VALUE message into the botnet every *keyValueLookUpTime* seconds. Smaller the value of the *keyValueLookUpTime* parameter, the number of FIND_VALUE messages that are sent into the botnet is increased. Based on the results, the <*key, value*> retrieval time is maximum for the following two combinations: 10 botmasters, 10% bot removal, inital number of bots equal to 20%, look up time set as 900 sec and peer size set as 200; botmasters set as 10, 10% bot removal, initial number of bots equal to 1% and look up time as 3600 seconds and peer size set as 300.

Comparison of Figures 4.17 and 4.18 also demonstrates the impact of peer list size. It can be clearly seen that for different peer list sizes the <*key, value*> pair retrieval time does not fluctuate much. This is due to the fact that bot initiating a search sends FIND_VALUE message every *key value look up time* seconds to the nearest bots with respect to the *key ID*. The receiving bot either returns the *value* or sends k closest contacts. The search terminates once the bot receives the <*key, value*> pair. This implies that <*key, value*> pair retrieval is dependent on *look up time*. Analysis of number of bots that retrieve the <*key, value*> pair, as specified in previous section, indicates that the parameter *keyValueLookUpTime* does not impact the number of bots that retrieve the <*key, value*> pair to a greater extent. Combining these two results it can be concluded that, a bot should search for the <*key, value*> pair with

Peer Size = 200			Bot Masters 5				Bot Masters 10			
Key Value Look Up Time (sec)	Botnet Size	Strategy	Bot Removal 0%		Bot Removal 10%		Bot Removal 0%		Bot Removal 10%	
			Initial Bots: 1%	Initial Bots: 20%	Initial Bots: 1%	Initial Bots: 20%	Initial Bots: 1%	Initial Bots: 20%	Initial Bots: 1%	Initial Bots: 20%
225	81000	Growth Model	84.26	81.43	85.06	84.46	84.55	81.65	83.03	83.28
		Sybil (True Value)	93.15	93.71	97.48	92.78	90.95	92.90	92.00	90.25
		Sybil (Any Value)	75.90	76.10	77.43	74.90	74.00	75.51	75.20	74.53
225	40500	Growth Model	82.58	82.06	85.98	81.98	84.23	80.38	81.38	81.60
		Sybil (True Value)	88.43	88.65	90.26	94.55	91.24	86.17	86.48	86.85
		Sybil (Any Value)	71.45	69.50	70.35	71.53	73.78	69.00	69.33	*67.68*
900	81000	Growth Model	92.37	89.63	92.55	91.51	92.35	87.68	93.20	100.45
		Sybil (True Value)	101.12	104.31	104.00	100.60	100.81	102.38	99.95	103.81
		Sybil (Any Value)	83.36	85.26	83.58	82.60	83.48	85.43	82.33	82.75
900	40500	Growth Model	95.35	91.68	98.58	94.35	95.76	90.20	93.85	90.92
		Sybil (True Value)	99.65	96.76	102.15	101.42	95.72	95.35	101.42	97.14
		Sybil (Any Value)	82.07	77.98	81.13	79.18	80.65	78.40	81.25	89.07
3600	81000	Growth Model	99.22	94.30	98.48	93.90	97.80	92.65	96.97	95.22
		Sybil (True Value)	106.96	108.96	109.65	106.76	102.90	105.55	106.77	104.30
		Sybil (Any Value)	91.07	93.27	92.68	90.63	89.28	91.56	91.16	90.13
3600	40500	Growth Model	98.10	96.71	100.38	99.16	97.61	95.96	98.93	97.20
		Sybil (True Value)	105.16	102.50	107.71	**114.68**	101.06	101.80	103.85	103.38
		Sybil (Any Value)	89.23	85.97	89.37	92.47	87.92	86.48	89.07	88.02

Table 4.7: <*key, value*> pair time (minutes) for peer list size 200

Peer Size = 300			Bot Masters 5				Bot Masters 10			
Key Value Look Up Time (sec)	Botnet Size	Strategy	Bot Removal 0%		Bot Removal 10%		Bot Removal 0%		Bot Removal 10%	
			Initial Bots: 1%	Initial Bots: 20%	Initial Bots: 1%	Initial Bots: 20%	Initial Bots: 1%	Initial Bots: 20%	Initial Bots: 1%	Initial Bots: 20%
225	81000	Growth Model	81.32	84.05	78.10	81.42	80.97	78.18	78.72	82.68
		Sybil (True Value)	86.65	92.48	86.48	91.98	87.48	91.65	86.27	84.70
		Sybil (Any Value)	69.42	74.90	68.98	71.83	71.65	73.77	69.38	68.03
225	40500	Growth Model	80.43	76.78	77.38	77.25	78.52	78.72	76.70	74.25
		Sybil (True Value)	84.52	84.85	83.36	87.43	85.40	83.07	83.60	86.60
		Sybil (Any Value)	66.32	*64.18*	64.38	64.65	67.12	63.38	65.63	65.05
900	81000	Growth Model	88.65	85.52	86.45	84.98	72.53	81.92	85.85	86.15
		Sybil (True Value)	93.98	97.42	97.72	94.28	94.50	96.08	93.23	96.50
		Sybil (Any Value)	77.27	81.07	78.93	77.35	77.47	80.58	77.02	78.97
900	40500	Growth Model	88.47	85.63	88.48	83.02	86.52	84.73	86.60	85.17
		Sybil (True Value)	92.73	92.87	94.62	101.28	92.10	90.62	91.22	91.87
		Sybil (Any Value)	75.52	73.78	74.57	78.27	75.50	72.80	74.27	72.30
3600	81000	Growth Model	94.32	88.98	96.13	102.23	94.07	87.68	92.55	90.55
		Sybil (True Value)	99.92	102.33	**120.73**	102.08	99.10	101.03	101.00	99.95
		Sybil (Any Value)	85.97	87.15	94.70	86.70	85.38	86.88	85.90	85.60
3600	40500	Growth Model	93.13	90.27	94.22	90.23	93.28	89.20	91.60	91.60
		Sybil (True Value)	101.75	96.37	102.00	103.58	98.63	97.98	96.95	102.05
		Sybil (Any Value)	86.08	81.78	83.23	82.98	84.42	81.93	82.57	83.57

Table 4.8: <*key, value*> pair time (sec) for peer list size 300

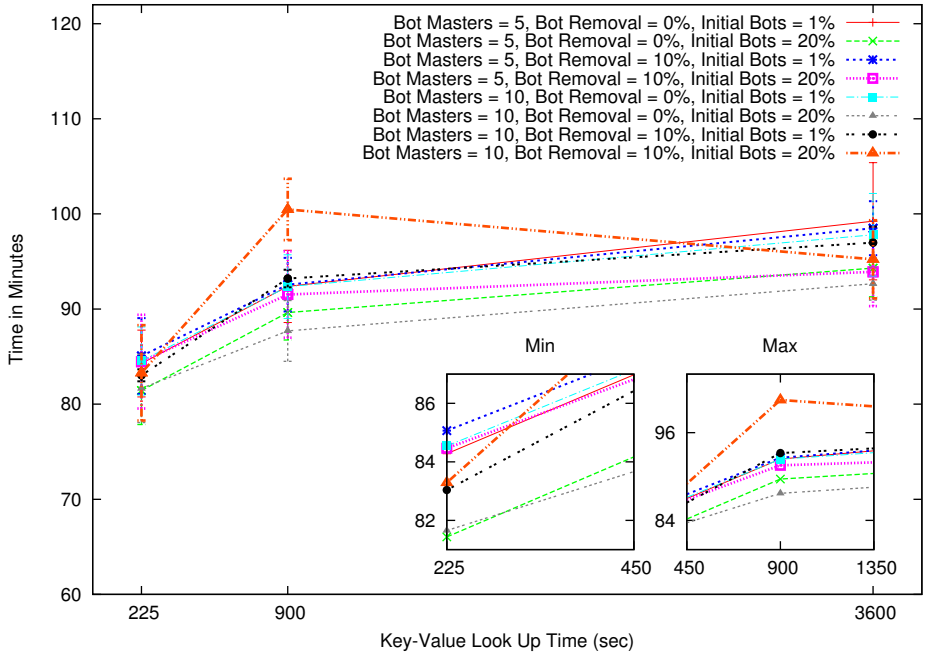

Figure 4.17: Standard Growth Model for 81000 bots: <*key, value*> pair retrieval time, with peer list size set at 200.

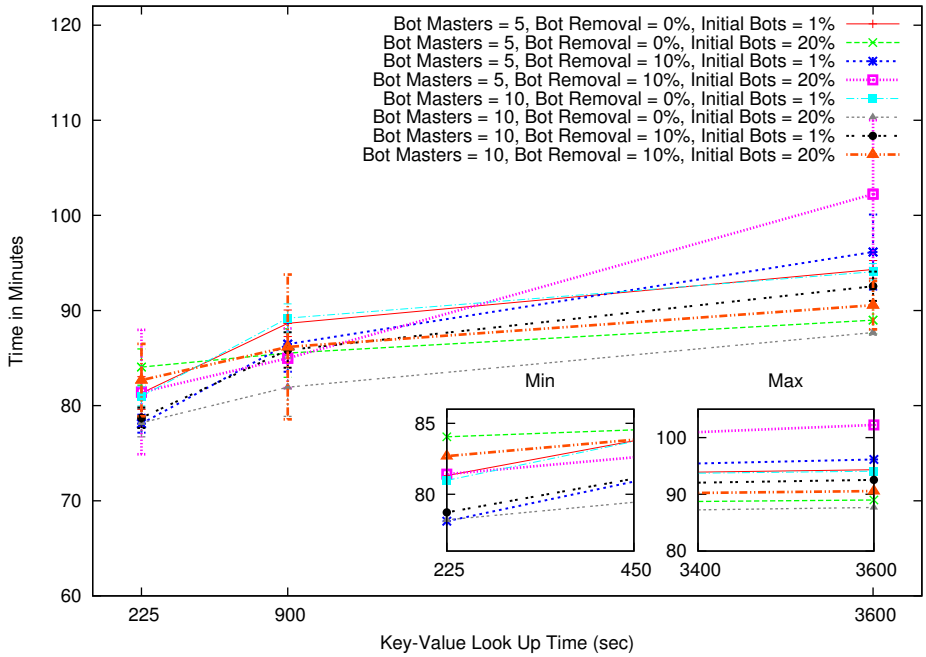

Figure 4.18: Standard Growth Model for 81000 bots:: <*key, value*> pair retrieval time, with peer list size set at 300.

a higher *keyValueLookUpTime* This is due to the fact that the number of messages sent by the bot to retrieve the <*key, value*> pair is reduced, leading to decrease in the network load and enhancing its invisibility in the Internet.

In order to determine the impact of botnet size with respect standard growth model of Storm, botnet size is reduced to half *i.e.,* 40500 bots. Comparison of data indicates that even when the botnet size is reduced to half, the time required by the bot to retrieve the <*key, value*> pair falls within the same range *i.e.,* the minimum amount of time required by the bot to retrieve the <*key, value*> is around 74 minutes and the maximum time required is around 100 minutes. To summarize, it can be concluded that time required by a bot to retrieve the <*key, value*> pair is independent of the botnet size. This analysis also supports the fact that there are many variations of bots and the size of these networks is limited.

Figures 4.19, 4.20, 4.21 and 4.22 show the result of Sybil Mitigation strategy for different combinations of parameters for a botnet size of 81000 bots. Figures 4.19 and 4.20 indicate the time required by the bot to obtain the any <*key, value*> pair whereas, Figures 4.21 and 4.22 indicate the time required by the bot to obtain the true <*key, value*> pair from the network. Comparison of Figures 4.19 and 4.20 with 4.21 and 4.22 show that for Sybil Mitigation strategy the amount of time required to obtain true <*key, value*> pair from the network is larger than the time required to obtain any <*key, value*> pair. The main aim of a Sybil bot is to insert its contact information in the bot's k-bucket and respond to any FIND_VALUE message with the fake <*key, value*> pair. The increase in the retrieval time of the true <*key, value*> pair indicates that the Sybil bots are able to insert their contact information in bots' k-buckets leading to the increase in the retrieval time of the true <*key, value*> pair.

Comparison of Figures 4.17 and 4.18 with 4.21 and 4.22 also indicates that with Sybil disruption strategy, time required by the bot to obtain the true <*key, value*> pair from the botnet is increased as compared to the standard growth model of Storm. The maximum amount of time required is increased from 102 minutes to 121 minutes, *i.e.,* approximately an increase of 20 min in the <*key, value*> retrieval. The increase in the retrieval time of the true <*key, value*> pair is due to the fact that the Sybil bots are able to insert their contact information in other bots' k-buckets. Comparison of Figures 4.17 and 4.18 with 4.19 and 4.20 indicates that with a mitigation strategy the amount of time to retreive the any <*key, value*> pair is decreased by 10 to 12 mintues.

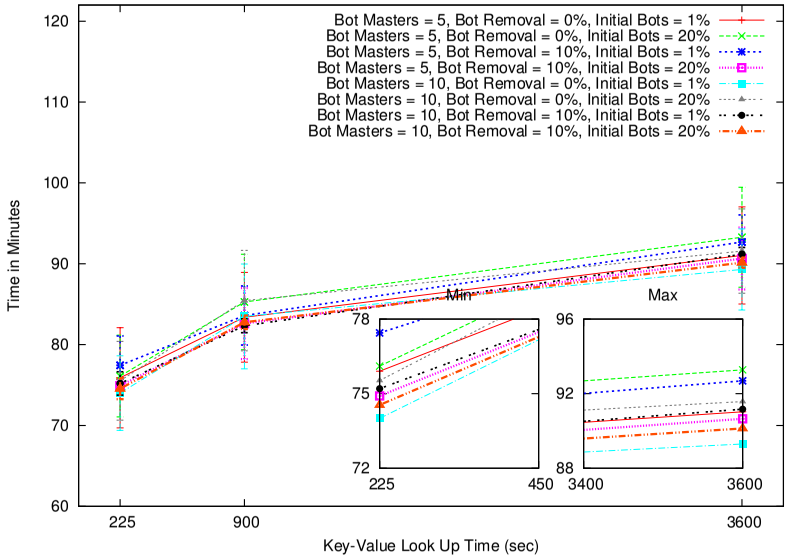

Figure 4.19: Sybil Mitigation for 81000 bots: any <*key*, *value*> pair retrieval time, with peer list size set at 200.

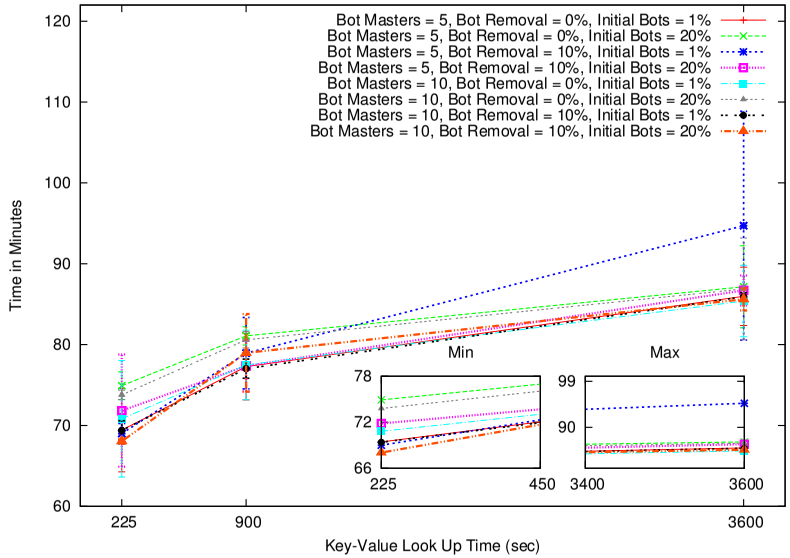

Figure 4.20: Sybil Mitigation for 81000 bots: any <*key*, *value*> pair retrieval time, with peer list size set at 300.

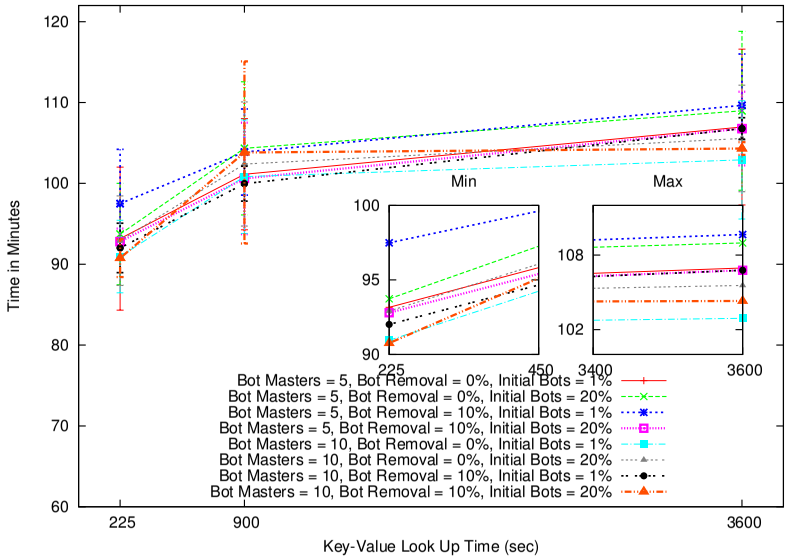

Figure 4.21: Sybil Mitigation for 81000 bots: true <*key*, *value*> pair retrieval time, with peer list size set at 200.

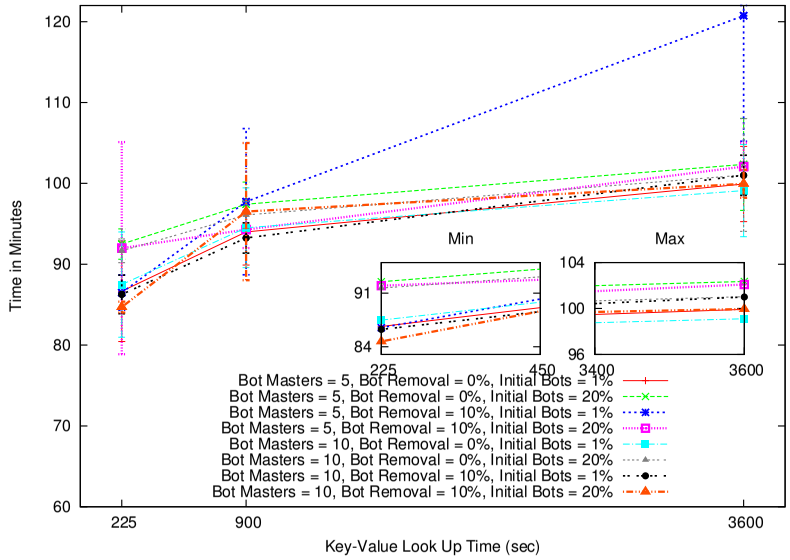

Figure 4.22: Sybil Mitigation for 81000 bots: true <*key*, *value*> pair retrieval time, with peer list size set at 300.

For Sybil disruption strategy, when the botnet size is reduced to half, *i.e.,* 40500, the botnet results are similar to that of the Sybil botnet with botnet size of 81000 bots. The amount of time required to retrieve the true <*key, value*> pair increases, whereas the amount of time to retrieve any <*key, value*> pair decreases when compared to the Standard growth model of Storm with botnet size of 40500 bots. To summarize, it can be concluded that even for a smaller bonet size, Sybil distruption strategy is very effective as this mitigation strategy not only increases the amount of time to retrieve the true <*key, value*> pair, but also decreases the amount of time that is required to retrieve any <*key, value*> pair from the botnet.

4.2.3 <*key, value*> Retrieval Statistics

In the previous sections, results on the duration required by the bot to retrieve the <*key, value*> pair from the botnet has been provided. In this section, <*key, value*> pair retrieval statistics versus time and <*key, value*> retrieval statistics versus the simulation evolution time has been provided.

These statistics would indicate the instance of time at which the <*key, value*> pair retrieval is maximum and minimum, and determine the engineering design parameters that could effect the <*key, value*> pair retrieval.

4.2.3.1 <*key, value*> pair Retrieval Statistics versus Time

Figures 4.23 and 4.24 show the <*key, value*> pair retrieval statistics per time interval for the standard growth model of Storm with a botnet size of 81000 bots and having 1% initial number of bots, for peer list size of 200 and 300 respectively. Figures 4.23 and 4.24 indicate that a large percentage of bots require 90 minutes to 150 minutes from the time of its birth to retrieve the <*key, value*> pair from the botnet. However, an interesting observation is that the percentage of bots that retrieve the key, value pair from the botnet reduces during betwen the 90 to 150 mintues. This is due to the fact that the <*key, value*> pairs are injected into the botnet at every hour.

Comparison of Figures 4.23 and 4.24 indicates that peer list size has an impact on the the <*key, value*> retrieval statistics. Figures indicate there is an increase in the number of bots that retrieve the <*key, value*> pair at the 90 minute for a peer list size of 300. Moreover, the Figures also indicate that most of the bots retrieve the <*key, value*> pair within a maximum of 180 minutes. However, there are bots in the network that take more than 4 hours to retrieves the <*key, value*> pair, though this

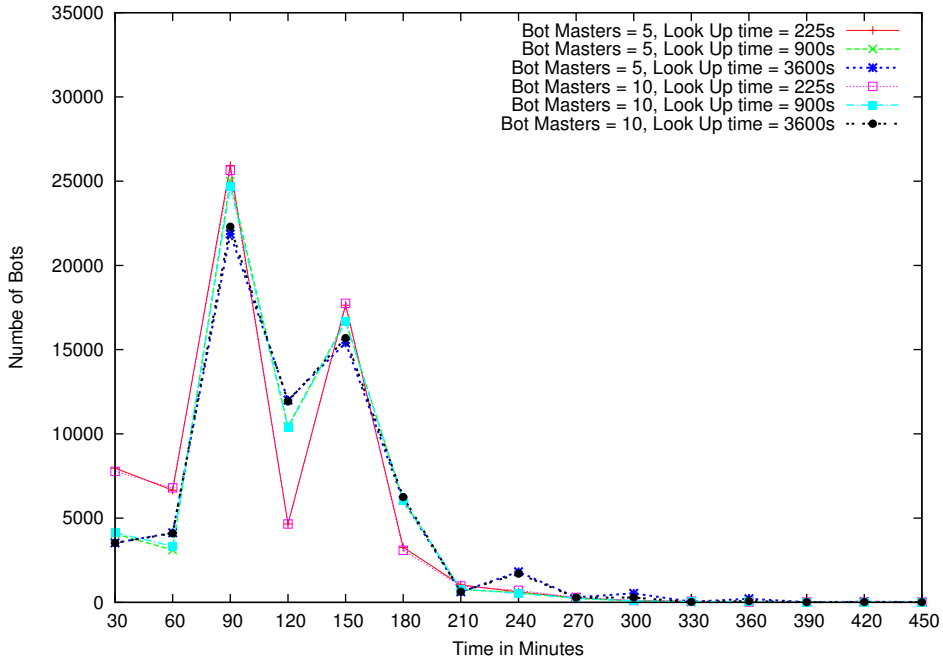

Figure 4.23: Histogram of number of bots retrieving <*key, value*> pair per time interval for Standard growth model of Storm with peer list size of 200.

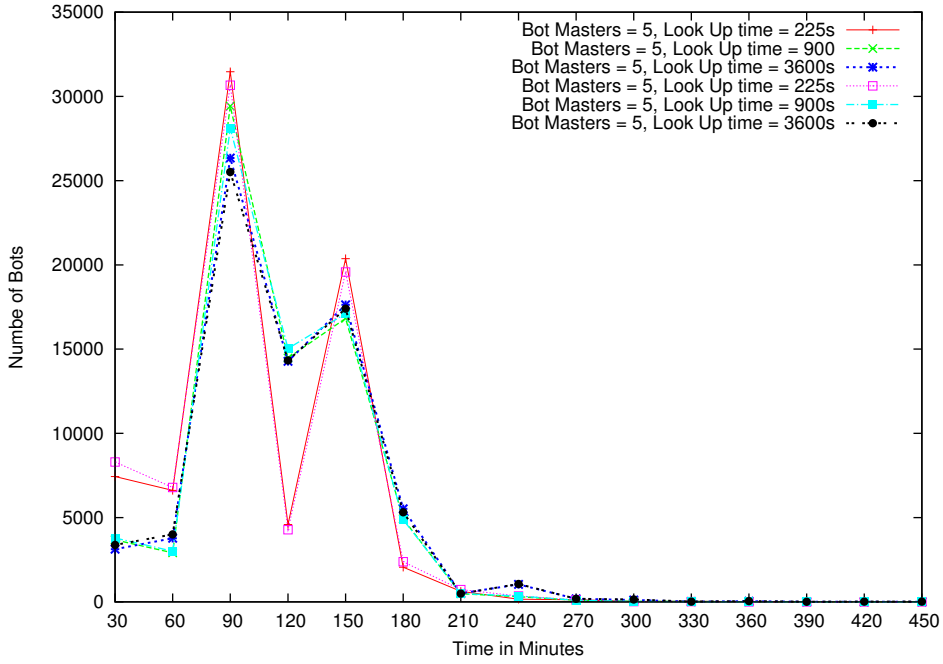

Figure 4.24: Histogram of number of bots retrieving <*key, value*> pair per time interval for Standard growth model of Storm with peer list size of 300.

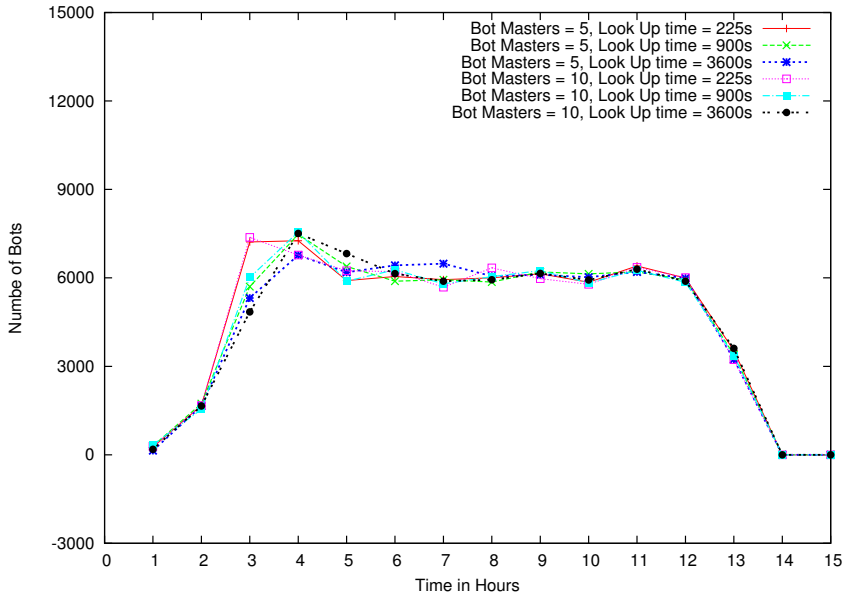

Figure 4.25: Histogram of number of bots retrieving <*key, value*> pair versus evolution of the simulation for Standard growth model of Storm with 1% inital number of bots.

number is relatively small (few hundreds of bots). For different design parameters and mitigation strategies, the <*key, value*> pair retrieval statistics have little impact on these curves.

4.2.3.2 <*key, value*> pair Retrieval Statistics Versus Evolution of the Simulation

Figure 4.25 represents the histogram of number of bots retrieving the <*key, value*> pair per simulation time for the standard growth model of Storm with 1% initial number of bots and peer size set to 200. This Figure indicates that when the initial number of nodes in the system is small, the number of bots that retrieve <*key, value*> pair per hour follow a similar distribution. Simulation results with different peer size have shown the same results. When the number of bots present in the botnet is reduced to half, the curve patter remains the same.

Figure 4.26 represents the histogram of number of bots retrieving the <*key, value*> pair per simulation time for the standard growth model of Storm with 20% initial number of bots and peer size set to 200. Comparison of Figures 4.25 and 4.26 indicate that when the initial percentage of bots present in the botnet is larger, there is an

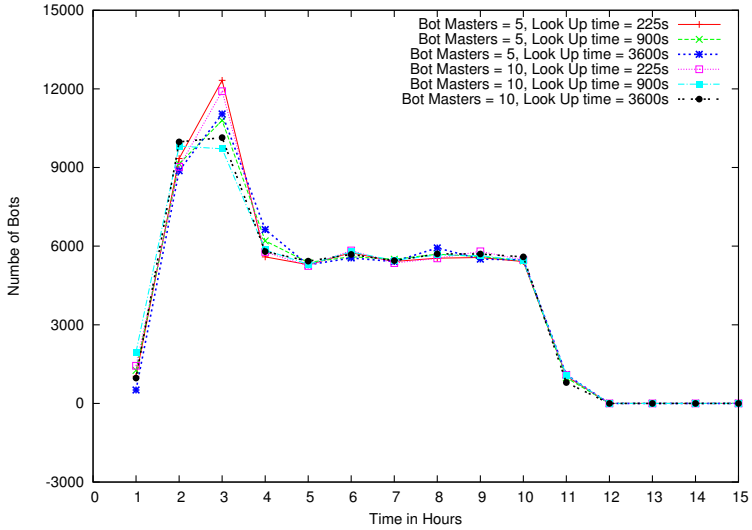

Figure 4.26: Histogram of number of bots retrieving <*key, value*> pair versus evolution of the simulation for Standard growth model of Storm with 20% inital number of bots.

increase in number of bots that retrieve the <*key, value*> pair during the initial hours of the simulation. This is due to the fact that with a higher initial percentage of bots present in the system, the bot is able to insert its contact details into other bots k-bucket. The Figure also indicates that, the number of bots that retrieve the <*key, value*> pair per hour follow a similar distribution after a certain time duration. Comparison of Figures indicate that the initial percentage of bots has an impact on the botnet. Simulation results with different parameters and random disinfection strategies have little impact on the histogram curve.

Figure 4.27 represents the histogram of number of bots retrieving the <*key, value*> pair per simulation time for Sybil distruption strategy with 20% initial number of bots and peer size set to 200. Comparison of Figures 4.26 and 4.27 indicates that for Sybil Mitigation strategy, initial percentage of bots present at the start of the simulation has no impact on the histogram patter of the <*key, value*> pair retrieval even during the initial few hours of the simulation. Comparison of Figures 4.25 and 4.27 indicate that Sybil mitigation strategy with 20% intial number of bots and standard growth model with 1% initial number of bots have a similar histogram pattern. Simulation results with combinations of parameters and random removal strategy have indicated a similar curve pattern.

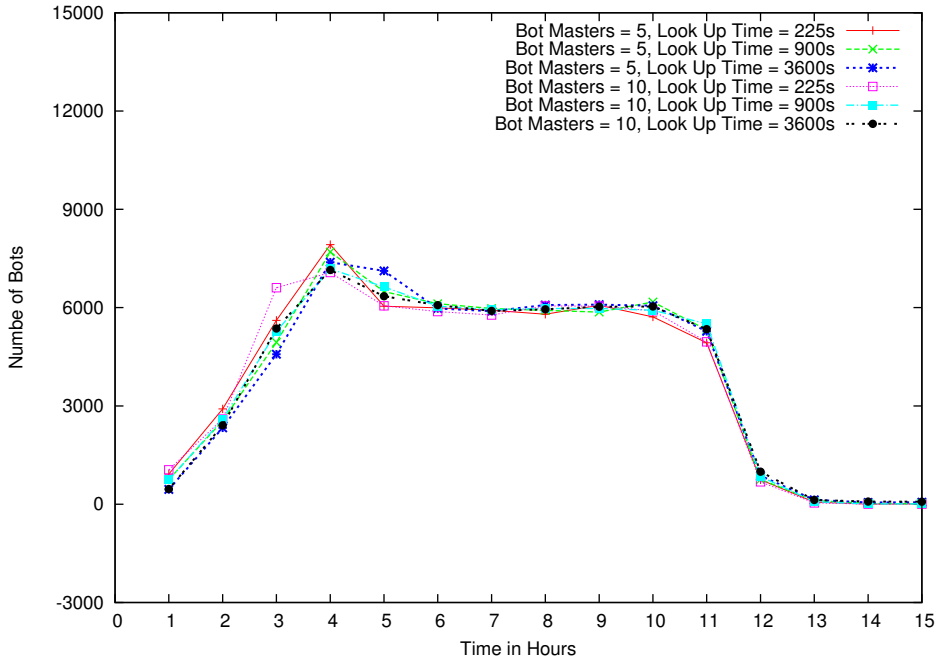

Figure 4.27: Histogram of number of bots retrieving any <*key, value*> pair versus evolution of the simulation for Sybil Distruption strategy with 20% inital number of bots.

4.2.4 Message Count Analysis

This section provides results with respect to the number of messages that a bot sends and receives from other bots to retrieve the <*key, value*> pair. Analysis of the number of messages that a bot sends and receives from other bots gives an indication on how the different combinations of parameters effect the network load induced by the botnet. Tables 4.9 and 4.10 summarize the results that are obtained for peer list sizes of 200 and 300 respectively as the bot searches for the key during the 24 hour simulation period.

Figures 4.28 and 4.29 show the results for the analysis of message count with respect to the Standard growth model of bots with peer size of 200 and 300 respectively. The mean number of messages that a bot sends and receives ranges from 278 to 523 depending on the combinations of parameter values.

The results can be summarized as follows:

1. Comparison of Figures 4.28 with 4.29 indicate that the key value look up time parameter has an impact on the message count. Mean number of messages sent

Peer Size = 200			Bot Masters 5				Bot Masters 10			
Key Value Look Up Time (sec)	Botnet Size	Strategy	Bot Removal 0%		Bot Removal 10%		Bot Removal 0%		Bot Removal 10%	
			Initial Bots: 1%	Initial Bots: 20%	Initial Bots: 1%	Initial Bots: 20%	Initial Bots: 1%	Initial Bots: 20%	Initial Bots: 1%	Initial Bots: 20%
225	81000	Growth Model	412	412	380	374	413	412	373	373
		Sybil (True Value)	397	**480**	368	366	395	481	366	365
		Sybil (Any Value)	378	425	353	350	377	425	351	350
225	40500	Growth Model	389	379	367	364	390	378	361	362
		Sybil (True Value)	371	362	352	354	370	361	350	350
		Sybil (Any Value)	355	346	337	339	353	346	336	336
900	81000	Growth Model	383	383	358	347	383	382	352	354
		Sybil (True Value)	369	448	349	346	369	447	339	348
		Sybil (Any Value)	353	399	336	333	353	398	326	335
900	40500	Growth Model	264	353	340	338	363	352	337	338
		Sybil (True Value)	348	337	327	326	346	336	323	324
		Sybil (Any Value)	334	324	315	314	332	323	312	312
3600	81000	Growth Model	336	340	322	413	335	337	307	306
		Sybil (True Value)	326	397	302	306	325	398	301	300
		Sybil (Any Value)	314	358	293	297	313	359	292	291
3600	40500	Growth Model	312	308	296	296	313	308	293	294
		Sybil (True Value)	303	297	287	289	303	297	286	286
		Sybil (Any Value)	294	288	279	281	293	288	*278*	278

Table 4.9: Message Count for peer list size 200

Peer Size = 300			Bot Masters 5				Bot Masters 10			
Key Value Look Up Time (sec)	Botnet Size	Strategy	Bot Removal 0%		Bot Removal 10%		Bot Removal 0%		Bot Removal 10%	
			Initial Bots: 1%	Initial Bots: 20%	Initial Bots: 1%	Initial Bots: 20%	Initial Bots: 1%	Initial Bots: 20%	Initial Bots: 1%	Initial Bots: 20%
225	81000	Normal Infection	507	450	497	446	506	492	490	488
		Sybil (True Value)	509	520	474	477	483	**523**	475	473
		Sybil (Any Value)	490	477	459	461	465	480	459	457
225	40500	Growth Model	503	490	472	474	501	486	471	474
		Sybil (True Value)	481	470	459	460	483	470	457	457
		Sybil (Any Value)	464	454	444	445	466	454	442	442
900	81000	Growth Model	471	435	463	463	472	450	455	453
		Sybil (True Value)	475	489	445	452	475	488	443	446
		Sybil (Any Value)	459	452	432	438	459	452	430	433
900	40500	Growth Model	465	458	443	442	464	453	438	438
		Sybil (True Value)	449	441	430	431	450	440	424	429
		Sybil (Any Value)	435	428	418	418	436	427	413	417
3600	81000	Growth Model	420	413	411	380	420	451	408	407
		Sybil (True Value)	427	457	412	402	427	457	402	401
		Sybil (Any Value)	416	427	402	393	416	426	393	393
3600	40500	Growth Model	411	406	398	396	412	406	390	393
		Sybil (True Value)	404	397	385	390	403	397	382	388
		Sybil (Any Value)	394	388	377	382	393	388	*375*	380

Table 4.10: Message count for peer list size 300

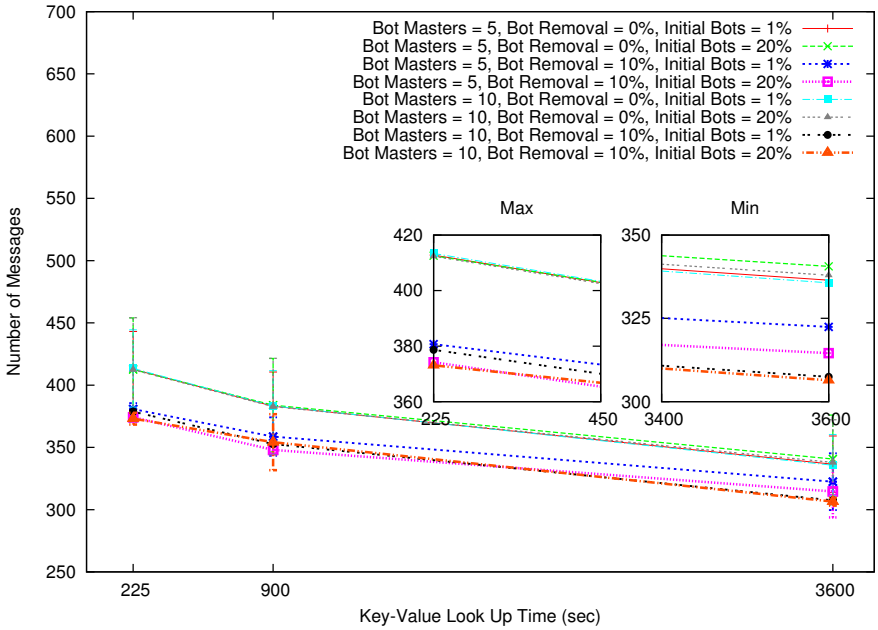

Figure 4.28: Standard Growth Model for 81000 bots: Mean Message count for <*key, value*> pair retrieval, with peer list size set at 200.

or received by a bot in order to retrieve the pair value is lower when the search is done every hour when compared to the search that is done 16 times in an hour. An interesting observation is that even with lesser number of messages sent or received by the bot to retrieve the <*key, value*> pair from the botnet, the percentage of bots that receive the <*key, value*> pair is almost the same when compared to the search that is done 16 times in an hour. This indicates that a lower *keyValueLookUpTime* parameter value increases the botnet load on the Internet, in addition it does not effect the percentage of bots that retrieve the <*key, value*> pair.

2. Comparisons of Figures 4.28 with 4.29 also indicate the effect of peer list size on the message count. The Figures indicate that the message count is higher for bots that have a larger peer list size. This is based on the fact that the Storm implements the PING_PONG Message. When a newly created bot is added into the botnet, it sends PING Message to all the bot contacts present in its peer list. As the simulation time progresses the number of bots present in the botnet is increased, leading to the increase in the number of bots that respond to the PING PONG message.

3. Figures 4.28 and 4.29 also indicate the effect of random disinfection strategy. Results indicate that even with 10% random removal of nodes from the botnet, message count is reduced approximately by only 6% to 8%. Moreover, the Figures indicate that number of bot masters does not impact the message count. This is due to the fact that the bot masters inject the <*key, value*> pair into the botnet at every hour.

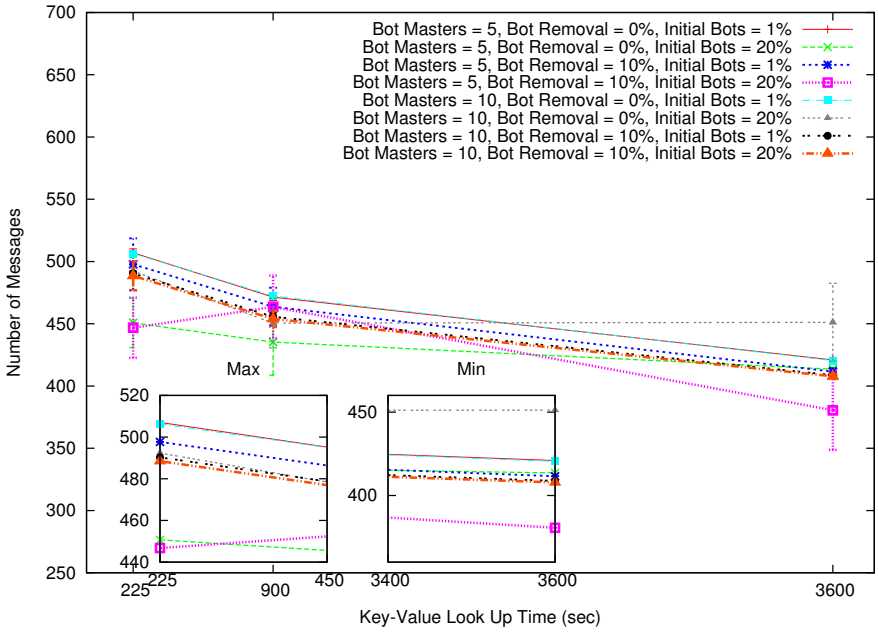

Figure 4.29: Standard Growth Model for 81000 bots: Mean Message count for <*key, value*> pair retrieval, with peer list size set at 300.

When the number of bots present is the botnet is reduced to half, results indicate that the message count is reduced only by 5% to 10%. This result indicates that the botnet size does not impact the mean message count. Comparison of the results with that of larger network indicate that almost the same numbers of messages are required irrespective of the botnet size.

Figures 4.30 and 4.31 show the mean number of messages that a bot sends and receives in order to retrieve the true <*key, value*> pair for peer sizes of 200 and 300 respectively, whereas Figures 4.32 and 4.33 indicate the mean number of messages that a bot sends and receives to retrieve any <*key, value*> pair for peer sizes of 200 and 300 respectively for Sybil mitigation strategy.

Comparison of Figures 4.30 and 4.31 with 4.28 and 4.29, indicates that for Sybil mitigation strategy, the parameter *keyValueLookUpTime* has an impact on the mean number of messages required by the bot to obtain the true <*key, value*> pair. Figures 4.30 and 4.31 also indicate that for a look up time of 3600 seconds, the number of messages required by the bot to retrieve the true <*key, value*> pair is increased, whereas, for a smaller look up time *e.g.,* 225 seconds, the number of messages remains either the same or is slightly reduced depending on the combinations of parameters. Comparison of Figures 4.32 and 4.33 with 4.28 and 4.29 indicates that the number of messages required by the bot to obtain any <*key, value*> pair is reduced by a very small percentage. The Figures also indicate that when the initial percentage of bots persent in the network is set to 20% and the peer list size is set to 200, the number of messages required to obtain the actual <*key, value*> pair is increased for the Sybil mitigation strategy.

For Sybil mitigation with botnet size of 40500, comparison of tables 4.9 and 4.10 indicates that the number of messages required to retrieve the <*key, value*> is the same as that of standard growth model having a botnet size of 40500 nodes, with the exception that the number of messages required to obtain either the any <*key, value*> pair from the botnet is slightly reduced when compared to standard growth model. Also, the results indicate that the initial percentage of bots does not impact the botnet C&C structure.

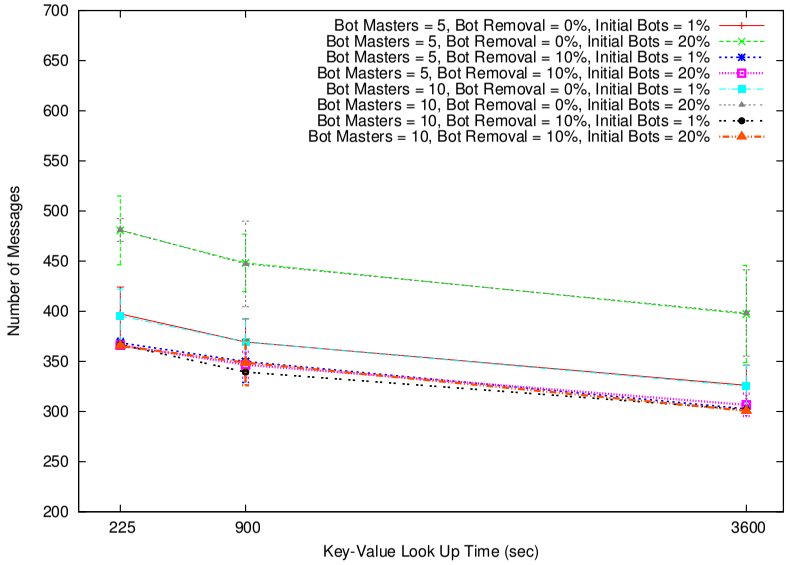

Figure 4.30: Sybil Mitigation for 81000 bots: Mean Message count for true <*key, value*> pair retrieval, with peer size set at 200.

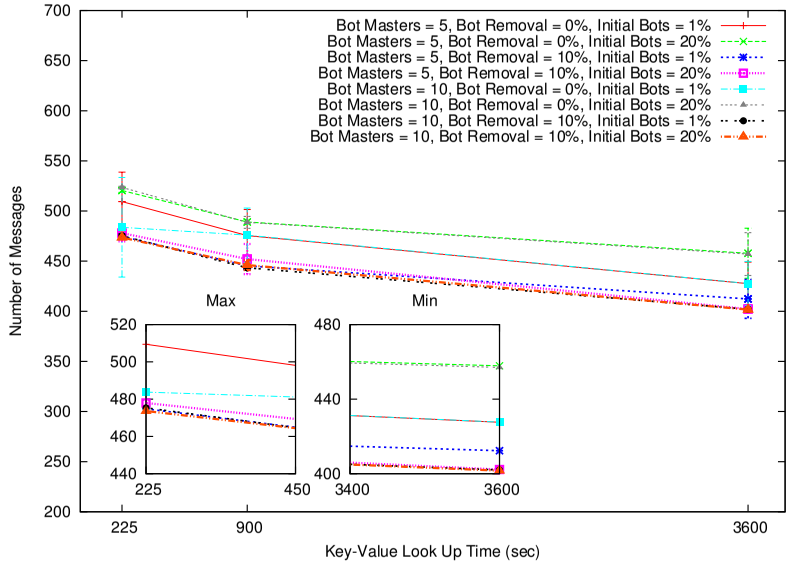

Figure 4.31: Sybil Mitigation for 81000 bots: Mean Message count for true <*key, value*> pair retrieval, with peer size set at 300.

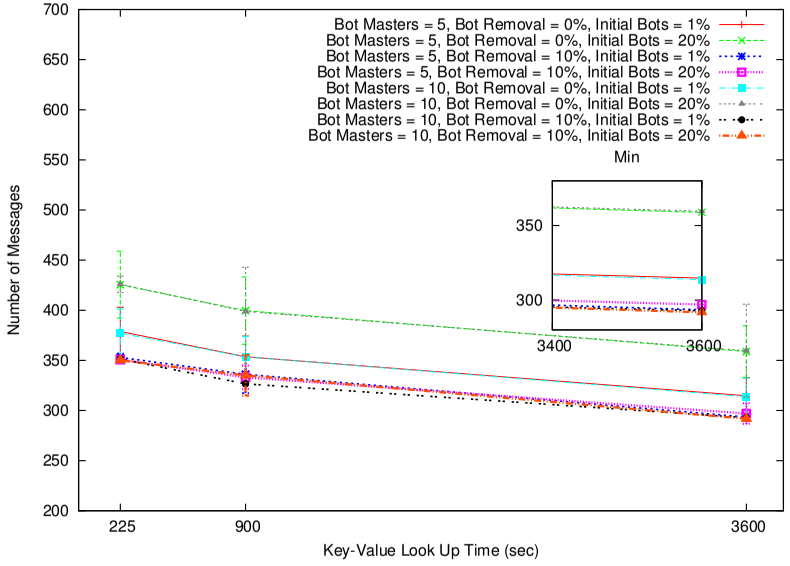

Figure 4.32: Sybil Mitigation for 81000 bots: Mean Message count for any <*key, value*> pair retrieval, with peer size set at 200.

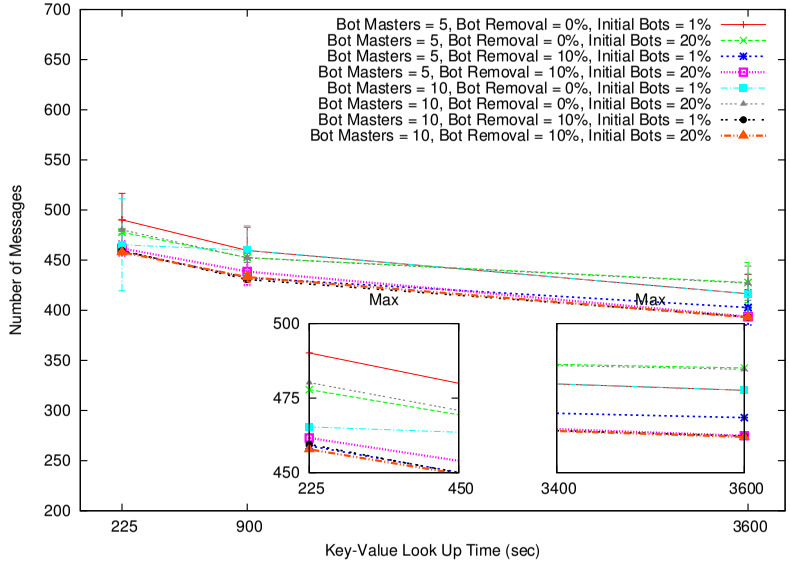

Figure 4.33: Sybil Mitigation for 81000 bots: Mean Message count for any <*key, value*> pair retrieval, with peer size set at 300.

Chapter 5

Conclusions and Future Work

In this thesis, the peer-to-peer behavior of the Storm botnet is studied by incorporating the actual packet level network behavior with a focus on large scale simulation. Significant research has been done for estimating the botnet's size, behaviour and mitigation approaches but the enginnering design decisions associated with P2P botnets has not been well studied. The peer-to-peer behaviour of Storm is studied by simulating its actual packet level network behaviour. The packet level simulator is developed via the simulation framework OMNET++ to determine the impact of engineering design decisions associated with Storm botnet. Parameters such as botnet size, peer list size, the number of botmasters and the key distribution time have been explored. Two different mitigation strategies are implemented. Random disinfection strategies and Sybil distruption strategies which disrupt the communication channel between the controller botmasters and bots. We analyze the design parameters of the bot that could possibly effect the mitigation strategy in addition to disrupt the communication between the bots.

In order to determine the key parameters and impact of the random disinfection strategies and the Sybil disruption strategies, simulations were performed and results were gathered for different combinations of parameters. Due to the fact that there are quiet a good number of parameters, simulations were done with different combinations of parameters for botnet size of 81000 and 40500. The number of bots was reduced to half, in order to determine the effect of botnet size on bots design attributes, and the mitigation strategy. The simulation results in Section 4.2 show the impact of bot parameters with respect to the standard growth model of Storm botnet as well as with the disinfection and disruption strategies. The simulation results show that Sybil disruption mitigation strategy outperforms random disinfection strategy. Results of

botnet simulations are particularly applicable to Storm but the reported results are also likely to hold for other P2P structured botnets. The Storm botnet was chosen because its inner workings have become fairly well understood, and the engineering design constraints are relatively common across malicious peer structured botnets.

Hence this thesis research brings us to some inferences metioned below:

- For the standard growth model of Storm botnet, the maximum percentage of bots that receive the <*key, value*> pair is around 90%, whereas, when the botnet size is reduced to half, the percentage of bots that receive the <*key, value*> pair is reduced to 75%. This indicates that as the botnet size grows, percentage of bots that retrieve the <*key, value*> pair increases, indicating that retrieval of <*key, value*> pair is dependent on the botnet size.

- For the standard growth model as well as for the mitigation strategies, parameters *peer list size* and *initial number of bots* impact the <*key, value*> pair retrieval from the Internet. On the contrary, parameter *key value look up time* and *number of bot controllers* do not impact the <*key, value*> retrieval.

- Sybil disruption strategy is more effective as compared to the random disinfection strategy. Simulation results indicate that even for a smaller network, Sybil disruption strategy significantly reduces the number of bots that retrieve the correct <*key, value*> pair. Moreover, Sybil disruption strategy also increases the retrieval time of the correct <*key, value*> pair from the Internet.

- Distribution of <*key, value*> pair into the botnet per hour follows more or less constant retrieval rate. Even when the initial number of bots present in the botnet is higher, the system gradually returns to a constant retrieval rate. It is also noted that, a larger percentage of bots retrieve the <*key, value*> pair within 2 hours of their birth.

- Propagation of the <*key, value*> pair in the botnet is independent of bot ID and *key* ID, as most of the bots lie in the Kademlia k-bucket range starting from 120 to 127, even with an evenly distribution of bots in the botnet.

5.1 Future Work

To understand the peer-to-peer based botnets in greater depth, there are many interesting issues that are need to be addressed. Few of these issues are:

- In the work, the performance of the P2P based botnets have been analysed based on specific combinations of parameter values. Simulation results have provided insights on some of the bots characteristics; nevertheless, there are many other parameters such as Kademlia's bucket size, degree of parallelism, key refresh time, key distribution time that could possibly impact the $<key, value>$ distribution. This work only considered a fixed bot growth rate, there is the need to determine how the on-going birth and death processes could possibly affects bot behavior.

- In this work, random disinfection strategies and Sybil distruption strategies have been implemented and analyzed. In our research we have not considered the possibility that the higher injection rate of Sybil bots in the botnet can make the Sybil bots detectable by the botmaster. The Sybil distruption strategy requires the addition of specific percentage of Sybil bots into the network. Alternate mitigation strategies needs to be identified that would not only help to reduce the visibility of fake bots to the botmaster but also help to find botmasters that are present in the botnet.

- For the simulations, a fixed number of randomly connected routers are used. There is a need to gain an understading on how the standard growth model and the mitigation strategies behaves under the Internet style scale-free network.

These results about the design parameters, along with further improvements would provide significant insights to gain a better understandging of the trade off with respect to the botnet design parameters and mitigation strategies.

Bibliography

[1] E. Cooke, F. Jahanian, and D. Mcpherson, "The zombie roundup: Understanding, detecting, and disrupting botnets," in *Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*, pp. 39–44, June 2005.

[2] A. Ramachandran, N. Feamster, and D. Dagon, "Revealing botnet membership using dnsbl counter-intelligence," in *SRUTI'06: Proceedings of the 2nd conference on Steps to Reducing Unwanted Traffic on the Internet*, (Berkeley, CA, USA), p. 8, USENIX Association, 2006.

[3] C. Davis, J. Fernandez, S. Neville, and J. McHugh, "Structured peer-to-peer overlay networks: Ideal botnets command and control infrastructures?," InProceedings of the 13th European Symposium on Research in Computer Security (ESORICS08), October 2008.

[4] M. Abu Rajab, J. Zarfoss, F. Monrose, and A. Terzis, "A multifaceted approach to understanding the botnet phenomenon," in *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, (New York, NY, USA), pp. 41–52, ACM, 2006.

[5] T. Holz, M. Steiner, F. Dahl, E. Biersack, and F. C. Freiling, "Measurements and mitigation of peer-to-peer-based botnets: A case study on storm worm," in *LEET* (F. Monrose, ed.), USENIX Association, 2008.

[6] "Storm chaos prompts virus surge." `http://news.bbc.co.uk/1/hi/technology/6278079.stm`, Accessed: 19 January 2007.

[7] D. Dagon, G. Gu, C. Lee, and W. Lee, "A taxonomy of botnet structures," pp. 325–339, Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual, December 2007.

[8] P. Wang, S. Sparks, and C. C. Zou, "An advanced hybrid peer-to-peer botnet," in *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, (Berkeley, CA, USA), p. 2, USENIX Association, 2007.

[9] C. Kalt, "Internet relay chat: Client protocol." `http://www.mirc.com/help/rfc2812.txt`, April 2000.

[10] P. Porras, H. Sadi, and V. Yegneswaran, "A multi-perspective analysis of the storm (peacomm) worm," *Computer Science Laboratory, SRI International*, October 2007.

[11] J. R. Binkley and S. Singh, "An algorithm for anomaly-based botnet detection," In Porceedings of the 2nd conference on Steps to Reducing Unwanted Traffic on the Internet(SRUTI'06), pp. 43-48, July 2006.

[12] "Federal Bureau of Investigation (FBI), Operation Bot Roast." `http://www.fbi.gov/pressrel/pressrel07/botnet061307.htm`, February 2007.

[13] I. Arce and E. Levy, "An analysis of the slapper worm," *Security Privacy, IEEE*, vol. 1, pp. 82–87, February 2003.

[14] J. Stewart, "Sinit p2p trojan analysis." `http://www.secureworks.com/research/threats/sinit/`, December 8, 2003.

[15] "Bot software looks to improve peer age." `http://www.securityfocus.com/news/11390`, May 2006.

[16] J. B. Grizzard, V. Sharma, C. Nunnery, B. B. Kang, and D. Dagon, "Peer-to-peer botnets: overview and case study," in *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, (Berkeley, CA, USA), p. 1, USENIX Association, 2007.

[17] F. Boldewin, "Peacomm.c - cracking the nutshell." `http://www.reconstructer.org`, September 2007.

[18] J. Stewart, "Storm worm ddos attack." `http://www.secureworks.com/research/threats/storm-worm`, February 8, 2007.

[19] R. M. Wheeler, David J.; Needham, "Tea, a tiny encryption algorithm," In Proceeding of 2nd International Workshop of Fast Software Encryption, LNCS, Vol. 1008, pp. 363-366, 1995.

[20] P. Ferrie, "Attacks on virtual machine emulators," Symantec Advanced Threat Research, December 2006.

[21] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 53-65, 2002.

[22] "Know your enemy: Fast flux service networks," *Tech Report: Honeynet Project*, 2007.

[23] "World's most powerful supercomputer goes online." `http://seclists.org/fulldisclosure/2007/Aug/520`, August 2007.

[24] A. Varga, "Omnet++ comuunity site." `http://www.omnetpp.org`.

[25] C. Davis, J. Fernandez, S. Neville, and J. McHugh, "Sybil attacks as a mitigation strategy against the storm botnet," In: Proceedings of the 3rd International Conference on Malicious and Unwanted Software (MALWARE 2008), October 2008.

[26] M. Castro, M. Costa, and A. Rowstron, "Abstract debunking some myths about structured and unstructured overlays," *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design and Implementation. USENIX Association*, pp. 85–98, 2005.

[27] P. Kirk, "Gnutella protocol development." `http://rfc-gnutella.sourceforge.net/index.html`, 2003.

[28] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A distributed anonymous information storage and retrieval system," pp. 46–66, 2000.

[29] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *Communications Surveys Tutorials, IEEE*, vol. 7, pp. 72–93, Quarter 2005.

[30] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for internet applications," *Networking, IEEE/ACM Transactions*, vol. 11, pp. 17–32, February 2003.

[31] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pp. 329–350, Springer-Verlag, 2001.

[32] K. Kutzner and T. Fuhrmann, "Measuring large overlay networks - the overnet example," in *Konferenzband der 14. Fachtagung Kommunikation in Verteilten Systemen (KiVS 2005)*, (Kaiserslautern, Germany), February 2005.

[33] "edonkey network." `http://en.wikipedia.org/wiki/EDonkey`.

[34] A. Singh, T.-W. Ngan, P. Druschel, and D. S. Wallach, "Eclipse attacks on overlay networks: Threats and defenses," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pp. 1–12, April 2006.

[35] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach, "Secure routing for structured peer-to-peer overlay networks," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 299–314, December 2002.

[36] M. Steiner, T. En-Najjary, and E. W. Biersack, "Exploiting kad: possible uses and misuses," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 5, pp. 65–70, October 2007.

[37] D. Dumitriu, E. Knightly, A. Kuzmanovic, I. Stoica, and W. Zwaenepoel, "Denial-of-service resilience in peer-to-peer file sharing systems," *SIGMETRICS Perform. Eval. Rev.*, vol. 33, no. 1, pp. 38–49, 2005.

[38] N. Christin, A. S. Weigend, and J. Chuang, "Content availability, pollution and poisoning in file sharing peer-to-peer networks," in *EC '05: Proceedings of the 6th ACM conference on Electronic Commerce*, (New York, NY, USA), pp. 68–77, ACM, February 2005.

[39] J. Liang, N. Naoumov, and K. W. Ross, "The index poisoning attack in p2p file sharing systems," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pp. 1–12, April 2006.

[40] K. El Defrawy, M. Gjoka, and A. Markopoulou, "Bottorrent: misusing bittorrent to launch ddos attacks," in *SRUTI'07: Proceedings of the 3rd USENIX workshop on Steps to reducing unwanted traffic on the internet*, (Berkeley, CA, USA), pp. 1–6, USENIX Association, 2007.

[41] I. Baumgart, B. Heep, and S. Krause, "Oversim: A flexible overlay network simulation framework," in *IEEE Global Internet Symposium*, pp. 79–84, May 2007.