Evaluating the Effectiveness of Sybil Attacks Against Peer-to-Peer Botnets

by

Adam Louis Verigin
B.Eng., University of Victoria, 2008

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

MASTER OF APPLIED SCIENCE

in the Department of Electrical and Computer Engineering

Evaluating the Effectiveness of Sybil Attacks Against Peer-to-Peer Botnets

by

Adam Louis Verigin
B.Eng., University of Victoria, 2008

Supervisory Committee

Dr. S. W. Neville, Supervisor
(Department of Electrical and Computer Engineering)

Dr. M. McGuire, Departmental Member
(Department of Electrical and Computer Engineering)

Supervisory Committee

Dr. S. W. Neville, Supervisor
(Department of Electrical and Computer Engineering)

Dr. M. McGuire, Departmental Member
(Department of Electrical and Computer Engineering)

ABSTRACT

Botnets are networks of computers which have been compromised by malicious software which enables a remotely located adversary to control them and focus their collective power on specific tasks. Botnets pose a significant global threat, with tangible political, economic and military ramifications and have resultingly become a field of significant interest within the cyber-security research community. While a number of effective defence techniques have been devised for botnets utilizing centralized *command and control* (C&C) infrastructures, few of these techniques are suitable for defending against larger-scale *peer-to-peer* (P2P) botnets. In contrast, the sybil attack, combined with index poisoning is an established defence technique for P2P botnets. During a sybil attack, fake bots (*i.e.*, sybils) are inserted into the botnet. These sybils distribute fake commands to bots, causing them not to carry out illicit activities. Bots also then unwittingly redistribute the fake commands to other bots in the botnet.

This work uses packet-level simulation of a Kademlia-based P2P botnet to evaluate 1) the impact that the location of sybils within the underlying network topology can have on the effectiveness of sybil attacks and 2) several potential optimizations to the placement of sybils within the underlying network topology.

Contents

Supervisory Committee ii

Abstract iii

Table of Contents iv

List of Tables viii

List of Figures ix

Acknowledgements xi

Dedication xii

1 Introduction 1

1.1 What is a Botnet? . . . 1
1.2 The Security Risks of Malicious Botnets . . . 2
1.3 Botnet Mitigation Targets . . . 2
1.4 Command and Control Infrastructure . . . 4
1.4.1 Centralized C&C . . . 4
1.4.2 Peer-to-Peer C&C . . . 5
1.4.3 Peer-to-Peer Overlay Networks . . . 5
1.4.4 Unstructured Peer-to-Peer Overlay Networks . . . 6
1.4.5 Structured Peer-to-Peer Overlay Networks . . . 8
1.4.6 Summary . . . 8
1.5 Mitigating Botnets via C&C Infrastructure . . . 9
1.5.1 High Agression: Eradication . . . 9
1.5.2 Medium Agression: Takedown . . . 9
1.5.2.1 C&C Server Takedown . . . 9
1.5.2.2 ISP Takedown . . . 10
1.5.2.3 DNS Takedown . . . 10

1.5.3 Low Agression: Disruption . . . 11
1.5.3.1 Traffic Detection and Blocking . . . 11
1.5.3.2 The Sybil Attack and Index Poisoning . . . 12
1.5.4 Summary . . . 13
1.6 Botnet Research and Analysis . . . 15
1.6.1 *Ad hoc* Observation and Testing . . . 16
1.6.2 Emulation . . . 17
1.6.3 Simulation . . . 18
1.6.3.1 Epidemiological Models . . . 19
1.6.3.2 Random Graph Models . . . 20
1.6.3.3 Network Models . . . 20
1.6.4 Analytical Modelling . . . 21
1.7 Limitations of Prior Works . . . 22
1.8 Thesis Goals . . . 23
1.9 Outline . . . 23

2 Botnet Protocol and the Sybil Attack **24**
2.1 *peer-to-peer* (P2P) Botnet Protocol . . . 24
2.1.1 IDs, Keys and the XOR Distance Metric . . . 25
2.1.2 k-buckets . . . 26
2.1.3 Remote Procedure Calls . . . 29
2.1.4 Lookups . . . 29
2.1.5 Bot Lifecycle . . . 30
2.2 The Sybil Attack . . . 31
2.2.1 Sybil Behaviour . . . 33
2.2.2 Fastest Response Path . . . 33
2.2.3 Logical Placement Optimization . . . 34
2.2.4 Physical Placement Optimization . . . 34
2.2.4.1 Unrestricted Sybil Placement . . . 35
2.2.4.2 Uninformed Sybil Placement . . . 35
2.2.4.3 Fully-Informed Sybil Placement . . . 35
2.2.4.4 Partially-Informed Sybil Placement . . . 36
2.3 Measures of Effectiveness . . . 36
2.4 Chapter Summary . . . 36

3 Simulation Model **38**

3.1 Design Criteria . . . 38
3.1.1 Realistic Network Topology . . . 39
3.1.2 Facilitating Statistical Measurements . . . 40
3.1.3 Scalability . . . 40
3.1.4 Extensibility . . . 41
3.2 Existing Simulation Frameworks . . . 41
3.2.1 NS-2 . . . 41
3.2.2 NS-3 . . . 41
3.2.3 Möbius . . . 42
3.2.4 PRIME SSF . . . 42
3.2.5 PlanetSim . . . 43
3.2.6 OMNeT++ . . . 43
3.2.6.1 OverSim . . . 44
3.2.7 Summary . . . 45
3.3 Simulation Model . . . 45
3.3.1 OMNeT++ Overview . . . 45
3.3.2 Underlay Network Model . . . 47
3.3.2.1 Network Topology . . . 47
3.3.2.2 Churn Management . . . 49
3.3.2.3 Bots . . . 51
3.3.3 Overlay Model . . . 52
3.3.3.1 Initializing Bots . . . 52
3.3.3.2 Protocol Implementation . . . 52
3.3.4 Simulation Modes . . . 53
3.3.4.1 Generation . . . 53
3.3.4.2 Experimentation . . . 53
3.3.5 Instrumentation . . . 54
3.3.6 Sybil Placement Strategy Implementations . . . 55
3.3.6.1 Unrestricted Sybil Placement . . . 55
3.3.6.2 Uninformed Sybil Placement . . . 55
3.3.6.3 Partially-Informed Sybil Placement . . . 55
3.3.6.4 Fully-Informed Sybil Placement . . . 56
3.4 STARS Integration . . . 56
3.5 Extensions . . . 58
3.5.1 Realistic Network Topology . . . 58

3.5.1.1 Generators . . . 59
3.5.1.2 Topology Size Reduction . . . 61
3.5.1.3 Annotators . . . 62
3.5.1.4 Writers . . . 64
3.5.2 Saving & Loading Simulation State . . . 64
3.5.3 Churn Model Modifications . . . 66
3.5.4 Low Memory Routing Tables . . . 66
3.6 Chapter Summary . . . 67

4 Experiments **69**
4.1 Common Parameter Settings . . . 69
4.1.1 Network Topologies . . . 69
4.1.2 General Configuration and Settings . . . 70
4.2 Experiment Set 1: High Churn . . . 70
4.2.1 Generation . . . 71
4.2.2 No Sybils . . . 72
4.2.3 Unrestricted Sybil Placement . . . 72
4.2.4 Restricted Sybil Placement . . . 73
4.2.5 Time and Storage Requirements . . . 74
4.2.6 Evaluation . . . 74
4.2.6.1 Peer List Infection . . . 77
4.2.6.2 Value Retrieval . . . 84
4.2.7 Experiment Set 2: Low Churn . . . 91
4.2.7.1 Time and Storage Requirements . . . 92
4.2.7.2 Peer List Infection . . . 92
4.2.7.3 Value Retrieval . . . 95
4.3 Summary . . . 98

5 Conclusions and Future Work **100**
5.1 Conclusions . . . 100
5.2 Future Work . . . 102

A Appendix **104**
A.1 Serialization of Factory-Constructed Classes . . . 104

Bibliography **105**

List of Tables

Table 2.1 Botnet Protocol's Tunable Parameters 25
Table 2.2 Example of k-bucket coverage and utilization for a 20,000 bot botnet. 27

Table 3.1 Performance comparison of OverSim and thesis simulation model. 45
Table 3.2 Summary of Underlay Network Topology Parameters. 47
Table 3.3 Summary of Targeted Churn Parameters 51
Table 3.4 Generator Components Summary 62

Table 4.1 Summary of Network Topology Configurations 70
Table 4.2 Summary of Constant Botnet Protocol Settings 71
Table 4.3 Summary of Parameter Settings for All Experiment Configurations 72
Table 4.4 Summary of Generation-Specific Parameter Settings 73
Table 4.5 Normal Churn Process Settings 73
Table 4.6 Targeted Churn Settings for Unrestricted Sybil Placement Attacks 73
Table 4.7 Targeted Churn Settings during Restricted Sybil Placement Attacks 74
Table 4.8 CPU Time and Storage Requirements of High Churn CAIDA Network Simulations 75
Table 4.9 CPU Time and Storage Requirements of High Churn ReaSE Network Simulations 76
Table 4.10 Summary of Parameter Settings for All Low Churn Experiments 91
Table 4.11 CPU Time and Storage Requirements of Low Churn CAIDA Network Simulations 92
Table 4.12 CPU Time and Storage Requirements of Low Churn ReaSE Network Simulations 93

List of Figures

Figure 1.1 High-Level Architecture of a Botnet 3
Figure 1.2 Centralized Botnet Architecture 5
Figure 1.3 Peer-to-Peer Botnet Architecture 6
Figure 1.4 Overlay Network 7
Figure 1.5 Illustration of General Spectrum of Analysis Techniques . . . 15

Figure 2.1 Least-Recently-Seen Bucket Eviction Policy 28
Figure 2.2 The Activity Lifecycle of a Bot 31

Figure 3.1 Underlay Network Architecture 48
Figure 3.2 Internal Module Structure of the `subnet` Module 49
Figure 3.3 Internal Module Structure of the `Bot` Module 51
Figure 3.4 Interaction of Modules When Initializing a New Bot 52
Figure 3.5 Topology Generator Architecture 59
Figure 3.6 Comparison of Node Connectivity between Generated and Reduced-Size Network Topologies 63

Figure 4.1 Mean Peer List Infection Levels Across CAIDA Network Simulations with 4000 and 8000 Sybils 78
Figure 4.2 Mean Peer List Infection Levels Across CAIDA Network Simulations with 12000 and 16000 Sybils 79
Figure 4.3 Mean Peer List Infection Levels Across ReaSE Network Simulations with 4000 and 8000 Sybils 80
Figure 4.4 Mean Peer List Infection Levels Across ReaSE Network Simulations with 12000 and 16000 Sybils 81
Figure 4.5 Mean End-of-Simulation Peer-List Infection Levels for Each Restricted Sybil Placement 82
Figure 4.6 Suspected Effectiveness Spectrum for Informed Sybil Placement Strategies 84

Figure 4.7 Correct Value Retrival Levels for Simulations without Sybils . 85
Figure 4.8 Example of Value Retrieval Measurements 86
Figure 4.9 Correct Value Retrieval Levels Across CAIDA Network Simulations with 4000 and 8000 Sybils 87
Figure 4.10 Correct Value Retrieval Levels Across CAIDA Network Simulations with 12000 and 16000 Sybils 88
Figure 4.11 Correct Value Retrieval Levels Across ReaSE Network Simulations with 4000 and 8000 Sybils 89
Figure 4.12 Correct Value Retrieval Levels Across ReaSE Network Simulations with 12000 and 16000 Sybils 90
Figure 4.13 Mean Peer List Infection Levels for Low Churn Experiments . 94
Figure 4.14 Mean End-of-Simulation Peer-List Infection Levels for Each Restricted Sybil Placement 95
Figure 4.15 Correct Value Retrival Levels for Simulations without Sybils . 96
Figure 4.16 Correct Value Retrieval Levels Across Low Churn Simulations with 4000 Sybils 97
Figure 4.17 Reduction in Correct Value Retrieval Level for All Sybil Attacks with Low Churn 98

ACKNOWLEDGEMENTS

I would like to thank:

My wife, family and friends for their years of patience, support, encouragement and prayer.

Dr. S. W. Neville, for providing me with the opportunity to conduct this research and for his guidance, feedback throughout the process.

The B.C. Government and NSERC ISSNet for providing scholarship funding.

DEDICATION

Soli Deo Gloria.

Chapter 1

Introduction

1.1 What is a Botnet?

For the purpose of this thesis, a bot is a network-connected computer running malicious software which enables a remote user (*i.e.*, "botmaster") to control it with various commands which the bot then carries out independently, and a botnet is a group of bots connected by a *command and control* (C&C) infrastructure. The defining characteristic of botnets are the C&C infrastructures which botmasters use to distribute commands to bots, allowing a botnet's collective power to be focused on specific tasks. While botnets are not strictly malicious, the presence of malicious botnets is so pervasive that the term "botnet" is generally taken to mean "malicious botnet" [1, 2, 3]. This work follows this convention.

Botnets are commonly regarded as having originated with EggDrop, an *Internet relay chat* (IRC) bot that was first published in 1993[4, 5]. EggDrop was not intended for malicious purposes but instead to help maintain and police IRC channels. EggDrop contained a feature called "botnet" which allowed IRC administrators to link together multiple bots and leverage their collective power[4, 6]. The power of botnets was due to the fact that each bot within the botnet was able to independently carry out commands sent by the administrator and communicate with one another to coordinate activities (*e.g.*, sharing ban lists[6]).

Cyber-criminals soon realized the potential power of botnets and began using them for malicious purposes. The PrettyPark botnet became the first wide-spread malicious botnet in 1999[4, 7], targeting Microsoft Windows. Between 2002 and 2004 the number of malicious botnet variants increased rapidly with the publication of modular botnet code-bases. Since then, botnets have been identified that run on Linux[8], Apple's OSX [9], mobile operating systems[10, 11] and home routers[12].

1.2 The Security Risks of Malicious Botnets

The threat of botnets in not limited to the number of platforms they are present on. Malicious botnets, by their design, pose significant threats for three main reasons.

First, botnets are frequently spread by taking advantage of vulnerabilities in the user's operating system. This means that the botmaster often ends up with administrative privileges on the user's computer, granting the botmaster access to sensitive information (*e.g.*, user keystrokes, financial credentials, intellectual property assets, *etc.*) across large numbers of users[13].

Second, botnets give their botmasters access to large volumes of distributed computing power that can then be used to enact malicious behaviours such as *distributed denial-of-service* (DDoS) attacks, e-mail spamming, distributed password cracking, *etc.*[13]. The bots enacting these attacks also often generate large volumes of network traffic as an ensemble, negatively impacting the performance of the network infrastructure on which it is sent across[14]. As an example, Symantec reported that email spam accounted for approximately 75% of all emails in 2011 with botnets being responsible for over 80% of this traffic[15].

Finally, because botnets are able to execute arbitrary commands, botmasters can rent out portions of their botnets. Thus, for a fee, unsophisticated criminals are able to gain access to the services of a botnet for their purposes. As a result, botnets have become a key source of computational resources for cyber-crime[16].

Because of the pervasiveness of botnets and the threat they pose to infected computers, botnets have been recognized as a significant global threat, with tangible political, economic and military ramifications[17]. This has prompted responses from corporate and national organizations such as Microsoft[18], Homeland Security[17, 19], US-CERT[20] and ENISA[21]. As a result, botnets have also become an important area of cyber-security research[3].

1.3 Botnet Mitigation Targets

With the threat of botnets understood, the next main question is how to mitigate this threat. Figure 1.1 illustrates the high-level architecture of a botnet: the botmaster sends commands to bots via a C&C infrastructure. Following from this, there are three general botnet components that can be targeted for mitigation.

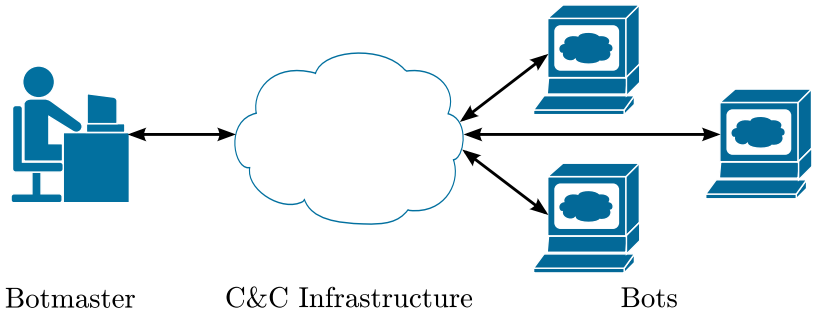


Figure 1.1: High-Level Architecture of a Botnet

The first mitigation target is the botmaster. The botmaster is effectively the head of a botnet, responsible for its actions, propagation and evolution. Botmasters are also aware of the architecture and operation of their botnet(s). Thus, if the botmaster is legally detained, they can be forced to shut down the botnet and make recompense for damages they have caused. However, the Internet is an international entity and national legal jurisdictions end every time a packet is routed across national borders. Thus, intelligent attackers route packets through multiple countries, requiring effective multi-jurisdictional cooperation if the botmaster's anonymity is to be overcome[22].

The second mitigation target is the set of vulnerable computers that form the "attack surface[22]" that cyber-criminals seek to exploit via malware. If these computers can be hardened through security patches and anti-virus software, the attack surface can be minimized. However, Symantec reported 403 million unique malware variants and an 81% increase in malicious attacks in 2011[15] and there were over 6500 *common vulnerabilities and exposures* (CVE) candidates for the year 2012[23]. This highlights the fact that this approach has itself proven to be a challenging problem for the security community and one that is far from solved.

The final mitigation target is the botnet's C&C infrastructure. As Section 1.1 highlighted, the C&C infrastructure is a key component of any botnet. Without it, the botmaster is no longer able to send commands to bots and focus their collective power; the botnet is rendered inert. However, disrupting the C&C infrastructure after the commands have been disseminated to bots is ineffective since bots act independently after receiving commands. Thus, mitigation strategies targeting the C&C infrastructure must seek to prevent bots from receiving commands. In practice, disrupting or dismantling botnet C&C infrastructures has proven to be an effective

means of mitigating and taking down botnets[24]. This is the mitigation approach that is explored in this thesis. It must be noted that the manner in which this mitigation approach is carried out is highly dependent upon the architecture of the C&C infrastructure. Thus, before expanding upon some of the botnet mitigation strategies of interest in this thesis, the various C&C architectures employed by botnets will be reviewed.

1.4 Command and Control Infrastructure

In discussing the trade-offs between different C&C architectures, the following terms will be used, in accordance with [25]:

- *Robustness* refers to a botnet's ability to retain its operational characteristics while subject to random node failures (*i.e.*, computers being turned on/off, random disinfection, *etc.*) without adjusting the tunable parameters of the botnet.
- *Resilience* refers to a botnet's ability to retain its operational characteristics while subject to deliberate and informed attacks without adjusting the tunable parameters of the botnet.
- *Diffuseness* refers to the average degree of intersection between the peer-lists of bots in a botnet. For example, in a highly diffuse botnet, there will be a low degree of intersection beetween peer-lists.

1.4.1 Centralized C&C

Early botnets used a centralized C&C infrastructure[26, 27], as shown in Figure 1.2. In the case of PrettyPark and most early botnets, this was achieved using a single IRC server to relay all C&C traffic. In other botnets, multiple IRC servers have been used, but the number of IRC servers is always much smaller than the number of bots in the botnet.

These centralized C&C structures are highly efficient. Each component is relatively specialized and there is little redundancy in the system. Botmasters are able to quickly recruit large numbers of bots to execute any command. However, centralized C&C structures are neither robust or resilient. First, because such botnets heavily rely on a small number of relay servers, they effectively have single points-of-failure that can easily be found and targeted. Second, there are distinct client and server

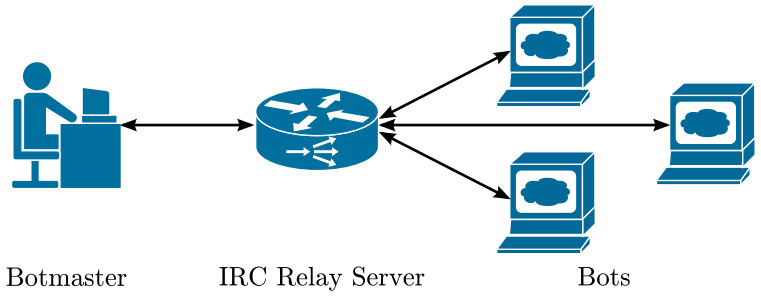


Figure 1.2: Centralized Botnet Architecture

roles, making it simple to identify where commands originate from. As a result, disrupting these networks proved relatively easy for the security community[1, 28].

1.4.2 Peer-to-Peer C&C

Having recognized these weaknesses, attackers responded by migrating to decentralized, *peer-to-peer* (P2P) C&C infrastructures[26], such as is depicted in Figure 1.3. P2P networks are generally very robust as most were designed for file sharing networks where nodes frequently only join the network for short periods of time. Furthermore, in a P2P network, nodes usually function as both clients and servers (*i.e.*, once a node has downloaded a file, they re-share it with other nodes). This provides three benefits for botnets. First, every bot in a botnet becomes part of the C&C infrastructure, meaning there are no longer single points of failure. Second, since there are no longer distinct roles for each computer in the botnet, it becomes harder to identify the origin of commands in the network. Third, if bots are able to independently calculate where commands will be stored, they will then be able to pull commands from the network rather than the botmaster having to push the commands to the full botnet, allowing botmasters the advantage of being able to covertly seed commands into the botnet. As a result of these benefits, P2P botnets have proven significantly more challenging for the security community[29] and are the focus of this work.

1.4.3 Peer-to-Peer Overlay Networks

An *overlay network* is a network built on top of another network[21]. The base network is often referred to as the *underlay network*. P2P networks, by their nature, form an overlay network: the sets of logical links between nodes form a logical overlay

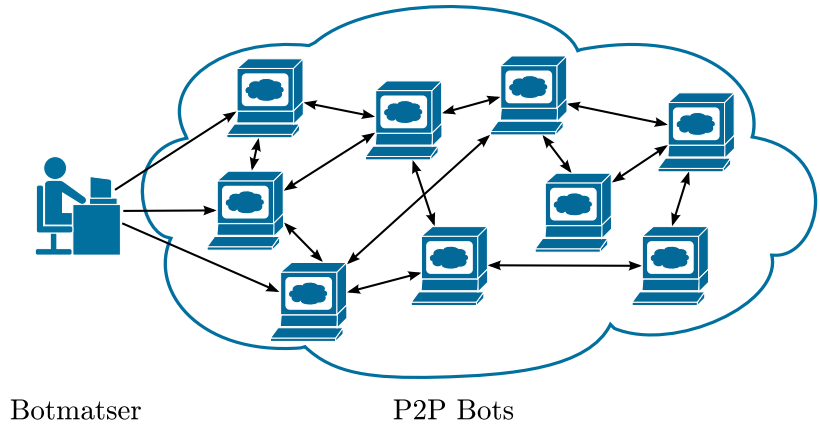


Figure 1.3: Peer-to-Peer Botnet Architecture

network in the application layer and the underlay network is the Internet over which the packets are routed. An example of such a network can be seen in Figure 1.4.

Multiple P2P protocols exist, and they are not all equally suited to efficient, robust, resilient botnet C&C infrastructures because they do not all use the same overlay network structure. The structure of P2P overlays is driven by the manner in which nodes form links with one another. Following from this, P2P overlay networks can be divided into two general categories: i) unstructured overlay networks and ii) structured overlay networks.

1.4.4 Unstructured Peer-to-Peer Overlay Networks

Unstructured P2P overlay networks are formed when links between nodes are established in an *ad hoc* fashion (*i.e.*, the P2P does not restrict which or how many peers a node chooses to connect itself to). This technique was used by early P2P networks (*e.g.*, Gnutella v0.4[30]), and was also used by the first known P2P botnet, Sinit[31]. Such networks tend to generate overlay networks that resemble Barabási-Albert models[32] due to preferential attachment[33] (*i.e.*, new nodes prefer to link to popular, highly connected peers). Barabási-Albert models are characterized by a small number of highly-connected nodes and an abundance of lowly-connected nodes.

Barabási-Albert graphs are known to be robust[25]; however, they are not well suited for botnet C&C because they are not resilient. Such networks can be severely crippled by targeting and removing only the highly connected nodes. Each time a highly-connected node is captured it also reveals a large portion of the overlay

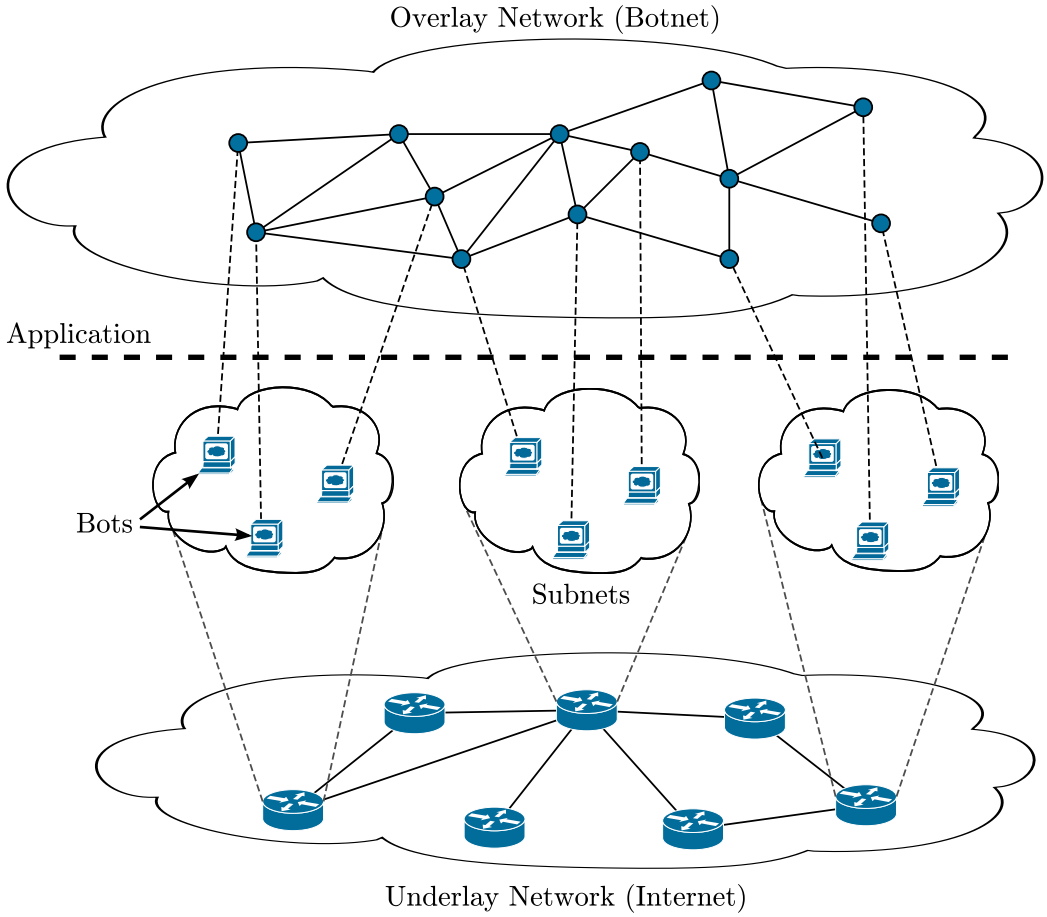


Figure 1.4: Overlay Network

topology. Furthermore, if these highly-connected nodes are not high-performance computers, they become a bottleneck and can severely degrade the performance of the botnet.

In an attempt to mitigate these issues, some P2P networks adopted a two-tier approach by creating a limited number of a second class of nodes which are often referred to as *superpeers*[34]. Superpeers are reliable, high-performance nodes that only provide directory store and search facilities[30]. Normal nodes connect to one or more superpeers to locate peers from which they can obtain data. Nodes then connect directly to the located peers to download the data. Thus, aside from the superpeers, these networks are still unstructured. While such networks address some of the performance issues of fully-unstructured P2P overlay networks, they do not address the resilience issue. Furthermore, this network structure reintroduces the

concept of clients and servers present in centralized C&C structures which further aids defenders in identifying which nodes to target.

1.4.5 Structured Peer-to-Peer Overlay Networks

In order to achieve the performance of two-tier networks with the completely-diffuse control structure of purely unstructured networks was a scientifically challenging problem. The resulting body of research lead to the emergence of *structured overlay networks* which are commonly implemented using *distributed hash tables* (DHTs)[35]. DHTs provide: i) a dictionary-like service that is partitioned across nodes in a network and ii) an efficient (typically $O(\log n)$[36]) entry-retrieval method for all nodes in the network. The dictionary consists of $\{key,value\}$-pairs. The *value* is the data hosts wish to store and retrieve. The *key* is the string which nodes use to search for and retrieve the associated value, and is typically the hash of the associated value or the value's file name. DHTs introduce a metric of distance between nodes in the network and keys in the dictionary so that values can be stored at nodes in "close proximity" to the associated key. This improves the performance of entry-retrieval and greatly improves the likelihood that searches will end successfully, even for unpopular values.

DHTs tend to result in overlay network structures similar to Erdős-Renyi graphs[37]. These graphs have been shown to be more resilient than Barabási-Albert graphs because they are more diffuse[25]. The Kademlia DHT protocol[38] creates particularly diffuse network graphs by placing an upper-limit on how many peers a node can retain in its peer-list. Since each node only has knowledge of a small portion of other nodes from the network, dissecting such a network by compromising individual nodes becomes increasingly difficult as the size of the network grows. However, this design also degrades the efficiency of the network[25].

1.4.6 Summary

The above C&C structures offer botmasters a general trade-off between effectiveness (*i.e.*, the ability to quickly recruit bots to a task) and resilience. From a defender's perspective, structured P2P C&C infrastructures pose a significant threat because they are resilient against targeted attacks. Protocols such as Kademlia increase this resilience by limiting the amount of information stored in each bot. Thus, this work focuses on P2P botnet protocols that make use of structured C&C overlay networks.

1.5 Mitigating Botnets via C&C Infrastructure

Botnet mitigation approaches that target the C&C infrastructure vary greatly depending upon the architecture of the C&C infrastructure. What follows is an overview of several different levels of aggression that defenders may use in attempting to actively mitigate bonets and an overview of some specific mitigation strategies that fall into these categories.

1.5.1 High Agression: Eradication

The highest level of aggression is *eradication* of the botnet[24]. This requires taking over control of the botnet and using the C&C infrastructure to tell each bot disinfect itself. This strategy has the ideal end result—the botnet is shut down and all infected computers are disinfected. However, this strategy requires defenders to have full knowledge of the botnet protocol and that bots be able to execute arbitrary commands. Furthermore, even if this strategy is proven technically feasible for a particular botnet, the legal and ethical questions regarding whether or not defenders should be allowed to disinfect remote computers have so far proven prohibitive[24, 39].

1.5.2 Medium Agression: Takedown

The next lower level of aggression is *take down* of the botnet's C&C infrastructure. This requires completely disabling all the components of the C&C infrastructure and any fall-back mechanisms. This strategy has a greater likelihood of preventing the botmaster from regaining control of the botnet. However, it also requires a greater amount effort on the part of the defenders since they must have full knowledge and understanding of the botnet's C&C infrastructure.

1.5.2.1 C&C Server Takedown

The classic approach to botnet takedown is to shut down and/or physically confiscate the C&C server(s). Leder *et al.*[40] outline three conditions that must all be met for this approach to work:

1. The botnet must use a centralized C&C infrastructure.
2. The location of the C&C servers must be discoverable.

3. The *internet service providers* (ISPs) providing service for the C&C server(s) must cooperate.

Unfortunately, each of these conditions can be countered. As Section 1.4 highlighted, modern botnets have been moving towards decentralized P2P C&C structures, either as a means of obfuscating the location of a fixed set of C&C servers (*e.g.*, the Storm botnet) or as a means of distributing commands(*e.g.*, the Nugache botnet)[24]. This counters both Conditions 1 and 2. Condition 3 can be countered by uncooperative ISPs which can delay law enforcement's access the the location of the C&C server with legal proceedings, allowing the botmaster time to relocate their server(s).

If the botnet's C&C servers contain software vulnerabilities or the botmaster uses poor security practices, it may be possible to defenders to remotely compromise and shut down the C&C servers. This removes Condition 3 from the above list, but is ethically questionable and may not be legally feasible.

1.5.2.2 ISP Takedown

An alternative to taking down the actual C&C server(s) is to take down the ISP that hosts the C&C server(s). This follows from recent studies which have shown that a large portion of global spam email is attributable to sources in a small number of ISPs[41, 42].

An instance of this type of takedown occurred in 2008 with the shutdown of the ISP McColo[43]. This ISP was suspected of housing the C&C servers for a number of major botnets. Following its takedown, spam levels were observed to drop significantly; however, this drop was short-lived, with spam levels returning to their previous levels within several months.

This highlights the main shortcoming with this mitigation strategy: if the botnet's C&C servers are distributed across multiple ISPs, then all of those ISPs must be taken down or this strategy will ultimately be ineffective.

1.5.2.3 DNS Takedown

Rather than using hard-coded IP addresses, many botnets use the *domain name system* (DNS) to dynamically resolve the IP address of their C&C servers. The key advantage of DNS is that the servers registered to a DNS name can be dynamically updated. Modern botnets often use layers of *fast flux DNS*[44], where numerous computers take turns registering with a DNS name for a brief period and act as

a proxy to the C&C servers, as a means of obfuscating the location of the C&C servers. DNS can also allow bots to dynamically generate DNS names as a fall-back mechanism.

If a botnet heavily relies on DNS, defenders may be able to reverse-engineer the list of DNS names used by the botnet. This knowledge can then be used to block or sink-hole (*i.e.*, route traffic to a device that logs the traffic and does not retransmit it) all traffic to and from these domains. Defenders can also work with DNS registrars to take down all the registered C&C domains and to register any unregistered domains with sinkhole servers owned by the defenders.

This strategy was successfully used by FireEye in the takedown of the Mega-D botnet[45] after an attempted ISP takedown failed. In general, however, if DNS registrars fail to comply with takedown requests, then the takedown attempt may fail. This makes DNS takedown difficult at a global scale. Furthermore, this strategy cannot be used against P2P botnets. The P2P protocols used by P2P botnets may not use DNS names to connect bots since not all Internet-connected hosts will have a DNS name. Even if every bot in a P2P botnet has a DNS name, those DNS names will likely be unique to each host, in which case a DNS takedown requires full knowledge of every peer in the botnet.

1.5.3 Low Agression: Disruption

The lowest level of aggression is *disruption* of the botnet's C&C traffic. In this case, bots continue to function as they normally would, but defenders prevent them from receiving commands from the botmaster. This can be achieved either by blocking the C&C traffic or by distributing fake commands. The main strength of this strategy is that it only requires partial knowledge of the botnet and its C&C infrastructure. Also, in some cases, it may be possible to use this approach to sever portions of the botnet reducing the overall effectiveness of the botnet. However, since the bots continue to function as normal, this strategy can only be used to temporarily halt or slow the botnet. If the C&C traffic is not fully disrupted the botmaster will eventually be able to counter the attack and regain full control of the botnet.

1.5.3.1 Traffic Detection and Blocking

One disruption strategy is to detect and then block botnet C&C traffic. This approach only requires partial knowledge about the botnet. Defenders must only know enough

about the botnet traffic to be able to distinguish it from other traffic. Once the C&C traffic is identified it can then be blocked or sink-holed.

There has been extensive work done regarding botnet detection. Silvia *et al.*[3] present an extensive survey of this work, but highlight that detection is still, "an arduous task," citing traffic encryption and evolving botnet designs as some of the causes of this difficulty. Host-based detection techniques, which attempt to identify botnet traffic and behaviour on an individual computer, often suffer from scalability issues: it is difficult to deploy such a system to every computer in a large network. Network-based detection techniques, which monitor for certain characteristics in network-wide traffic flows, must sift through large volumes of data. This may not be possible to do in real-time, reducing the effectiveness of this approach in large networks.

For this strategy to be effective in disrupting a botnet, a significant portion of the botnet's C&C traffic must be blocked. Since most botnets are globally distributed, this means that multiple ISPs in multiple countries must cooperate and coordinate their actions. Even if this strategy might prove effective technically, if the detection techniques employed require national and organizational entities to share sensitive information (*e.g.*, packet payloads), the entities may be unwilling to cooperate in the interest of preserving their own privacy or the privacy of their clients.

Another weakness with this technique is that there is a risk that traffic may be incorrectly classified. Traffic from legitimate hosts may be misclassified as botnet traffic and be blocked (*i.e.*, a false positive) or that botnet traffic may be misclassified as legitimate traffic and not be blocked (*i.e.*, a false negative). False-positive classifications can have large associated costs. For instance, if an organizational or national computer system fails because their traffic is blocked there may be security and legal repercussions. On the other hand, false-negative classifications permit botnet traffic to leak through. If too much traffic is permitted to leak through, this mitigation strategy will be ineffective.

1.5.3.2 The Sybil Attack and Index Poisoning

An alternative disruption strategy for P2P botnets is the *sybil attack*[46]. Many P2P networks rely on redundancy in order to improve their robustness (*e.g.*, replicating information to multiple peers, fragmenting tasks across multiple peers). The sybil attack aims to disrupt this redundancy by inserting sybils into the network. A sybil is a computer that joins the botnet and communicates with bots using the botnet's

P2P protocol; however, this computer forges multiple counterfeit identities, allowing it to pose as multiple bots in the P2P botnet. Subsequently, as legitimate bots select a set of peers for redundant remote operations, they can then be fooled into selecting a sybil bot multiple times, thus negating the redundancy of the operation.

In a P2P botnet, if sybils simply sink-hole commands then legitimate bots in the botnet will be less likely to find commands since a significant portion of their peers will never share commands. Alternatively, the sybil attack can be paired with *index poisoning*[47, 48] where fake commands are published into the botnet. When legitimate peers then search for these commands, they may receive the fake command instead of the real command. Thus, if sybils respond to queries with false commands, legitimate peers will start to re-share the fake commands throughout the network. This makes the fake commands available in more locations and increases the likelihood that subsequent searches by other peers will locate the fake commands instead of the real commands. This increases the effectiveness and efficiency of the sybil attack.

This strategy suffers from two main limitations. First, it requires a greater understanding of the botnet than detection and blocking since sybil bots must be able to participate in the botnet. Also, sybil bots must behave closely enough to normal bots to make them indistinguishable from normal bots; otherwise, the botmaster may be able to detect the presence of sybils and counter the attack. Second, not all botnets are susceptible to the sybil attack and/or index poisoning. If a botnet uses a some form of identity-based validation or reputation-based trust metric, it may be difficult or impossible to forge identities. This could prevent sybil attacks; however, this also makes it more difficult for new bots to be recruited into the botnet. Botmasters can also use asymmetric encryption keys to sign all commands, preventing index poisoning. However, if the sybil attack is proven feasible for a particular botnet it can be very effective, as demonstrated by Holz *et al.*[49] and their experimentation with this strategy against the Storm botnet.

1.5.4 Summary

A key point that must be highlighted from the above discussion is that there are no practically feasible takedown strategies for P2P botnets; the current approaches to botnet takedown are only feasible for botnets that use a relatively centralized C&C structure, even if that C&C structure is obfuscated (*e.g.*, by fast-flux DNS). The only presently-available strategy for mitigating P2P botnets are disruption strate-

gies. Thus, any work that can improve the efficiency and effectiveness of disruption strategies that target P2P botnets will provide valuable contributions to the current state of affairs for the defence community. Toward this end, this work focuses on the effectiveness of the sybil attack against P2P botnets.

1.6 Botnet Research and Analysis

Experiments for botnet research and analysis can generally be grouped into four approaches:

1. *Ad hoc* Observation and Testing
2. Emulation
3. Simulation
4. Analytical Modelling

These approaches are ordered according to their general fidelity and cost of execution in decreasing order (*i.e.*, *ad hoc* observation and testing has the highest general fidelity, analytic modelling has the lowest general fidelity) and according to their general controllability and repeatability of experiments in increasing order. This general trade-off between costly fidelity and the ability to run repeatable, controllable experiments is illustrated in Figure 1.5.

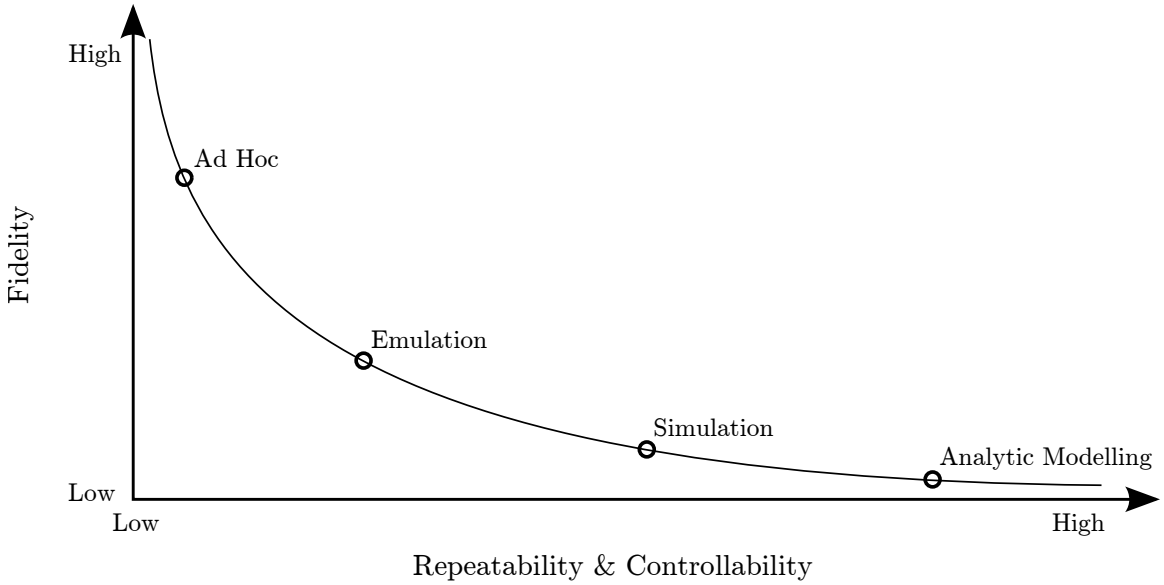


Figure 1.5: Illustration of General Spectrum of Analysis Techniques

An general development principle is to progress from analytical modelling to *ad hoc* observation and testing. The reasoning behind this is that i) each progressive approach is generally more costly than the previous and ii) if a theory is proven invalid at any point in this process, it will rarely, if ever, become correct in subsequent

approaches. Thus, by catching errors and deficiencies with the early approaches, the overall development cost of the solution is minimized.

1.6.1 *Ad hoc* Observation and Testing

Ad hoc observation and testing of botnets arose as the defence community started responding to the growing threat of botnets. In this approach, researchers generally observe a botnet running "in the wild" by either monitoring traffic from an infected computer (*e.g.*, using a honeypot[1, 50]) or by crafting a special version of the bot that explores the structure of the botnet[49].

The goals of research using this approach are usually:

- Characterizing the vulnerabilities that the malware exploited[51],
- Understanding how the botnet works[49, 51], and
- Measuring the size and geodemographic of the botnet[49, 51].

However, for the purpose of researching and developing mitigation strategies, this research approach has three major shortcomings:

1. The observer does not have control over the botnet and cannot alter the tunable design parameters of the botnet. As a result, any observations and mitigation strategies developed using this approach will be specific to the instance of the botnet under observation and are not necessarily generalizable to other instances of the same botnet protocol, other parameter tunings, or other botnet protocols.

2. The observer has no control over the background traffic across the whole of the botnet. This can lead to inconsistencies in measurements, as well as measurements that cannot be reliably reproduced.

3. The observer can affect the behaviour of the botnet, and can also affect the measurements collected by other observers if care is not taken to coordinate measurements.

In summary, *ad hoc* observation and testing can yield results with high fidelity to real-world botnets, but lacks the repeatability and controllability tenets of the scientific method which makes it less suited to general botnet research.

1.6.2 Emulation

Emulation improves upon the lack of repeatability and controllability of *ad hoc* observation and testing by providing a controlled environment (*i.e.*, testbed) in which the botnet executable can be run. For large-scale network research, there are two common types of testbeds: overlay testbeds (*e.g.*, PlanetLab[52]) and emulation testbeds (*e.g.*, Emulab[53], and DeterLab[54]).

Overlay testbeds use a geographically-diverse set of nodes (*i.e.*, bots) which form an overlay network on top of the Internet. Packets sent between nodes are routed across the actual routing infrastructure of the Internet, thus incurring realistic network delays. In contrast, emulation testbeds use an emulated network environment. As a result, emulation testbeds provide a greater degree of control and repeatability but at the cost of the additional hardware needed to emulate the network environment. Both types of testbeds often make use of virtualization to run multiple logical nodes on each physical node when conducting large-scale experiments. This can reduce the hardware costs of the testbed by an order of magnitude, but can affect the fidelity of the experiments if care is not taken to ensure that the underlying hardware of each physical node is not overloaded by the combined activity of each logical node[55].

The main shortcoming of emulation, with respect to botnet research, is the time required to run a sufficiently statistically rich set of experiments while exploring the design space of the botnet. Each experiment only allows researchers to observe a single configuration of the botnet's tunable parameters and the network environment. Thus, in order to explore the design space of the botnet, multiple experiments must be run with different parameters and environment configurations. Furthermore, each experiment is under the influences of a number of random processes that researchers do not have any control over because of the nature of physical hardware. This means that multiple repetitions of each experiment configuration are necessary in order to quantify the statistical distribution of behaviours under each configuration. Also, the work of Godkin[2] suggests that botnet behaviours are not necessarily stationary or ergodic, meaning that sufficient numbers of repetitions of each configuration must be run and analysed using statistical assessment techniques. The end result is that a significant number of experiments must be run to obtain scientifically valid results. However, since emulation experiments run in real time and the hardware costs of large-scale botnet experiments often restricts researchers to running only a single

experiment at a time, the total time to conduct a set of statistically rich experiments can be prohibitive.

1.6.3 Simulation

Simulation fundamentally differs from *ad hoc* experimentation and emulation in that is does not make use of actual botnet executables. Instead, simulation makes use of abstract models which combine researchers' knowledge and assumptions about a botnet into a set of mathematical, logical and symbolic relationships between entities representing the components of the botnet[56]. Instances of a model can then be used to simulate the behaviour of the botnet over a period of time. Once the model has been developed and validated, it can then be used to explore the behaviour of the botnet throughout the whole of the botnet's design space and to develop detection and mitigation strategies that are robust across the botnet's design space.

The simulation model is an abstraction of the real-world botnet. It is removed from real-world environments and contains simplified versions of many components of the real botnet or may also completely abstract away some components. This can greatly reduce the cost and, potentially, the time, of running experiments since a single computer can potentially simulate the activity of thousands of bots. However, the assumptions used to simplify the model can negatively impact the fidelity of the simulation. Thus, care must be taken to ensure that the resulting model closely approximates the behaviour(s) of interest in the real-world botnet. It is also worth noting that the cost savings of simulation is not necessarily in execution speed; sometimes simulations run slower than *ad hoc*/emulated experiments. However, in some cases this is balanced by reduced hardware requirements for individual simulations. In these cases, it may be feasible to distribute the execution of simulations across multiple machines decrease individual simulation run-times or it may be possible to run multiple experiments in parallel, reducing the time needed to run ensembles of simulations via developed frameworks such as STARS[57].

Similar to emulation, each simulation run only allows researchers to observe a single instance of the botnet under a single configuration of the environment and tunable botnet parameters. Repetitions of each individual configuration are also necessary because of random processes in the system. However, simulation differs from emulation in that the random processes are fully under the researchers' control. Simulations use values drawn from one or more *pseudo-random number generators*

(PRNGs), meaning identical simulations will generate identical results. The random processes can be varied between individual repetitions by seeding the PRNGs with different values, but researchers have full control over this. Another advantage related to PRNGs is that if a random process is not influenced by other random processes, it can be fully isolated by allocating it a separate PRNG. This then enables researchers to study the effects of certain random processes in isolation. As a result, simulation tends to be more conducive to rigorous statistical experimentation.

Because simulation models are abstract models, it is possible to construct different types of models for simulation. There are three common types of simulation models used for modelling botnets: epidemiological models, graph theory models and network models.

1.6.3.1 Epidemiological Models

In general, epidemiological models are used to simulate the spread of a virus throughout a population[58]. The model defines several states that each individual can be compartmentalized into (*e.g.*, susceptible, infected, *etc.*) and how individuals can transition between these states. The likelihood of an individual transitioning between states may be weighted according to various factors. In some cases it is possible to construct closed-form mathematical epidemiological models in which case analytical assessment becomes more appropriate than simulation.

When applied to botnets, epidemiological models are used to simulate the spread of botnets throughout a population of vulnerable computers and can also be used to simulate disinfection strategies[59, 60]. These models can also be enhanced to account for diurnal activity cycles and time zones[61], network topologies[60, 62] and peer relations within P2P botnets[59]. In [63], Song *et al.* combine an evolutionary game model with an epidemiological model. Using this model, they explore interactive strategies between multiple botnets and show that cooperative strategies allow botnets to survive with much lower contact rates.

The main weakness with these models is that they are primarily focused on accurately reproducing the spread of the botnet rather than the observable packet-level behaviour. As a result, they are not well suited to modelling detection and mitigation strategies that leverage network characteristics.

1.6.3.2 Random Graph Models

In the case of P2P botnets, random graph models arise naturally from the fact that their overlay structures resemble classical random graph models such as Barabási-Albert[32] and Erdős-Renyi[37] graphs (as was discussed in Sections 1.4.4 and 1.4.5). These graph models are well understood and suited to graph theory analysis.

In [25], Davis *et al.* use random graph models to compare the effectiveness and efficiency of various disinfection strategies for unstructured and structured P2P overlays. In [48] and [27], Davis *et al.* use random graph models to test the effectiveness of the sybil attack against P2P botnets, concluding that random sybil placement is just as effective as placing sybils logically close to commands in the botnet.

The largest weakness of random graph models is that they only model the overlay network, abstracting away the physical network layer. This means that these models provide no means of assessing the impact that network phenomena (*e.g.*, routing delays, traffic bottlenecks) have on the behaviour of the botnet and any relevant mitigation strategies.

1.6.3.3 Network Models

Network models, also called packet-level models, aim to accurately reproduce the observable packet-level behaviour of a botnet. As a result, these models must simulate both the application-level protocol of the botnet as well as the underlying network routing topology. This generally means network models require more resources than epidemiological and random graph models, and care must be taken to ensure that the network-layer of the model closely models real-world networks.

The advantage of network models is two-fold. First, they provide means of assessing the impact that network-level phenomena have on the behaviour of the botnet and any relevant mitigation strategies. Second, network models can be used to generate packet flows and other network-level information that can be gathered from real networking devices, meaning that these models can be used to assess detection and mitigation strategies that rely on observable network artefacts or information from both the application layer and the network.

In [64], Wei *et al.* develop a network model using the NS-2 simulator[65] to model the activity of distributed worms, focusing on the network-level characteristics necessary to provide a high level of fidelity for network phenomena. These characteristics include the structure of the network topology, link bandwidths between network nodes

and background traffic. In [66], van Ruitenbeek *et al.* develop a network model of a P2P botnet using Möbius[67] and demonstrate the effect that preventative measures and disinfections can have on the growth rate of the botnet. In [68], Kotenko *et al.* develop a packet-level simulation to evaluate cooperative defence strategies and their effectiveness against an DDoS attacks launched by an IRC botnet. In [69], Agarwal develops a network model based on the Storm botnet and uses it to explore the effect of altering the tunable parameters of the botnet protocol as well as how the botnet is affected by random node removals and a randomly-targeted sybil attack with index poisoning. This work is extended in [2] where Godkin uses this model to demonstrate that botnets exhibit non-stationary and non-ergodic behaviours, indicating that any work investigating botnets needs to conduct rigorous statistical analysis on collected data.

1.6.4 Analytical Modelling

Analytical modelling is similar to simulation in that it uses an abstract model in place of actual botnet executables. However, analytic modelling is restricted to using models comprised of solvable mathematical expressions. To achieve such models only a limited number of aspects of the botnet in question can be considered and these aspects themselves must be analytically tractable. If it is not possible to analytically solve the mathematical expressions a simulation approach becomes necessary.

As was mentioned in Section 1.6.3.1, some epidemiological models may be suitable for analytical assessment. Quite commonly, epidemiological models are combined with game theory models in order to evaluate different defence strategies. For example, in [70] Bensoussan *et al.* combine an evolutionary game model with an epidemiological model to explore the interaction between botnet operators and network defenders. Similarly, in [63] Song *et al.* combine an evolutionary game model with an epidemiological model to explore strategies where multiple botnets cooperate in order to increase their likelihood of survival. This work included both both analytical and simulation-based assessment.

As was highlighted above, the main limitation of analytical models is that they can only model a limited number of aspects of the botnet in question. For example, neither of the above-mentioned works include a model of the underlying routing network, in part because network traffic exhibits many complex behaviours that are not generally analytically tractable. This means that these models have the same shortcoming as

epidemiological and random graph simulation models: they lack the ability to assess the impact that network-level phenomena have on botnet behaviours and mitigation strategies.

1.7 Limitations of Prior Works

Because of the pervasive threat posed by botnets, botnet mitigation is clearly important; however, it still faces many challenges. In particular, botnets using structured P2P overlays for C&C raise two issues: i) structured P2P overlays are inherently resilient against targeted attacks and ii) there are currently no takedown strategies which can effectively target botnets using P2P C&C infrastructures. Thus, the only alternative left to defenders is to aggressively disrupt the botnet's C&C infrastructure in order to reduce the utility of the botnet to the botmaster, making it infeasible for the botmaster to maintain the botnet.

Disruption strategies reliant on accurately detecting botnet C&C traffic also still face a number of hard-to-solve problems. Botnet C&C traffic is a constantly moving target, often being encrypted and ever evolving, making it difficult to accurately detect it and detection approaches also often suffer from scalability issues. Even if these issues can be addressed, for these strategies to be effective in mitigating botnets, multiple ISPs in multiple countries must cooperate, which, due to political and privacy issues, is often an unrealistic expectation.

In contrast, when feasible, the sybil attack with index poisoning has been demonstrated as an effective mitigation strategy[49]. However, research into the effectiveness of this strategy and potential optimizations have been limited to *ad hoc* observation[49] and random graph model simulations[48, 27]. Each of these research approaches has its limitations. *Ad hoc* observation and testing is not a scientifically tractable research approach: researchers do not have control over the botnet or the background traffic and related network level phenomena, meaning this approach violates the repeatability and controllability tenets of the scientific method. Random graph model simulations, on the other hand, are scientifically tractable and are conducive to rigorous statistical analysis; however, these models have no means of assessing the impact of network-level phenomena or potential optimizations to the sybil attack that leverage knowledge about the underlying network topology. This leads Davis *et al.* to conclude that, "packet-level simulations may be required to accurately assess sybil attack rates"[27].

1.8 Thesis Goals

The goal of this work is to to explore how the placement of sybils within the underlay network topology impacts the effectiveness of the sybil attack against a P2P botnet. Also, by using knowledge of botnet traffic characteristics and bot placement within the underlay network, it should be possible to optimize the placement of sybils within the underlying network topology in order to maximize the effectiveness of the sybil attack.

Toward this end, this work develops a packet-level simulation of a Kaemlia-based botnet protocol based off of the work of Agarwal in [69]. This work extends the simulation model of Agarwal by including: i) a realistic *autonomous system* (AS)-level network topology model, ii) a state preservation mechanism and iii) a modified churn model. A realistic network topology is important because network topologies significantly impact the timing characteristics of packets as they are propagated through the network. Since this work looks at sybil placements with respect to these timing characteristics, it is important that they be accurately modelled. The state preservation mechanism is important in facilitating statistically rigorous testing as it decreases the run time of each individual simulation and helps to guarantee that ensembles of simulations begin with identical states botnet. Finally, the modified churn model is important for modifying the physical placement of sybils within the underlying network topology without affecting their logical placement within the botnet's overlay network.

1.9 Outline

The remainder of this thesis is organized as follows:

- **Chapter 2** presents a detailed description of the botnet protocol and sybil attack strategies used in this work.
- **Chapter 3** discusses the simulation model used and the extensions made to the work of [69].
- **Chapter 4** discusses the experiments used to evaluate the sybil attack strategies assessed in this thesis and their results.
- **Chapter 5** summarizes the findings of this thesis and outlines possibilities for future research.

Chapter 2

Botnet Protocol and the Sybil Attack

This chapter provides an overview of the Kademlia-based botnet protocol used in this work before discussing the details of Sybil attacks and the placement strategies that are be explored in this thesis.

2.1 P2P Botnet Protocol

This work uses a P2P botnet protocol based on the Storm botnet, which was based on the Kademlia DHT protocol[38]. The Storm botnet protocol was chosen because it is a well understood botnet protocol and there are previous works exploring the effectiveness of the sybil attack against the Storm botnet[48, 27, 49]. In contrast to the actual Storm botnet, which only used the Kademlia protocol to distribute the location of C&C servers and not actual commands[24], the botnet protocol used in this research distributes actual commands via the Kademlia protocol, making it a fully-decentralized botnet protocol.

This primary focus of the simulation model is the dissemination of commands to bots in the botnet via the C&C infrastructure and not the behaviours associated with the specific tasks carried out by bots. This follows from the fact that once a bot has received a command it can carry out that command independently of the botnet. Also, the Sybil attack aims to disrupt the C&C infrastructure and is generally not concerned with other aspects of the botnet's behaviour.

There are a number of tunable parameters that will be used in describing the botnet protocol used in this work. These parameters are summarized in Table 2.1. The remainder of the simulation model and the majority of model-specific details will be discussed in Chapter 3.

Parameter	**Description**
α	The degree of parallelism for all lookup procedures. [Default=3]
B	The size of bot IDs and keys used to identify values stored in the network. [Default=128]
k	The maximum number of peers stored in each k-bucket. [Default=20]
`numKeyValuePairs`	Number of $\{key,value\}$-pairs in each command set. A new command set it published into the network every `tRepublish`. [Default=32]
`numSeedLocations`	Number of bots that the full command set is initially distributed to. [Default=10]
`bootstrapSize`	The number of peer entries a bot receives for its initial peer-list.
`tRepublish`	Interval after which a new command set is published into the network. [Default=24h]
`tRefresh`	Interval after which any unaccessed k-buckets are refreshed. [Default=1h]
`tValueLookup`	Interval after which a bot attempts to search for one of the values it does not have. [Default=1h]
`tReplicate`	Interval after which a bot publishes every $\{key,value\}$-pair it has successfully retrieved. [Default=1h]

Table 2.1: Botnet Protocol's Tunable Parameters

2.1.1 IDs, Keys and the XOR Distance Metric

Each bot in the botnet is identified in the P2P overlay by a quasi-unique ID, B bits in length, which is randomly generated by each bot when they first join the botnet. The set of all possible IDs, $\{0, 1, \ldots, 2^B - 1\}$, is referred to as the overlay's *address space*. While Kademlia typically uses a 160-bit address space, the Storm botnet used a 128-bit address space[49]. Thus, this work also uses a 128-bit address space.

All bots in the botnet are trying to retrieve the current set of commands published by the botmaster. The number of commands in this command set is specified

by the `numKeyValuePairs` parameter, which is set at 32 since this is the same value used by the Storm botnet. All commands in the current command set are stored as {*key*,*value*}-pairs where the keys are also 128 bits in length, and thus are within the overlay's address space. The full command set is initially pushed (*i.e.*, seeded) to a small number of bots (specified by the `numSeedLocations` parameter) randomly distributed throughout the botnet. These bots immediately replicate each {*key*,*value*}-pair to the k bots closest to the key. All other bots are able to calculate the keys for each {*key*,*value*}-pair in the command set and attempt to pull (*i.e.*, look up) these commands by periodically searching for their associated key. After the time specified by `tRepublish`, the current command set becomes obsolete, a new command set is published into the botnet, and bots begin to search for the new command set.

Both IDs and keys play a central role in message routing in the Kademlia protocol. Kademlia uses the *exclusive-OR* (XOR) operator to define a metric of the distance between IDs and keys within the overlay's address space. Because IDs and keys are the same size, it is possible to calculate the distance between i) two IDs, ii) two keys or iii) an ID and a key. This XOR distance metric is used during lookups so that peers closest to the lookup target (*i.e.*, bot ID or command key) are queried first.

Another key part to the lookup procedure is that once the command set has been seeded into the network, those seed bots then replicate each {*key*,*value*}-pair to the k bots closest the key. This greatly increases the likelihood that subsequent command lookups by normal peers will find the commands since the lookups will converge toward the bots closest to the command.

2.1.2 k-buckets

Each bot in the botnet maintains a list of peers identified by a contact information tuple, typically <IP address, UDP port, bot ID>. Rather than storing the peer-list as a single, continuous list, it is divided into a set of sublists called k-buckets. The name k-bucket comes from the fact that each k-bucket can contain at most k peers, with k generally being set at 20[38]. This value of k is chosen in order to reduce the ammount of peer information retained by any given bot in the botnet while also ensuring that all k peers in a given bucket are not likely to fail within the span of an hour. For each $i \in [0, 128)$, each bot has a k-bucket that contains contact information for peers where $2^i \leq distance(ID_{bot}, ID_{peer}) < 2^{i+1}$. That is, the 127^{th} k-bucket contains peers whose ID differs from the bot's ID in the highest-order bit (and potentially other

bits), whereas the 0^{th} k-bucket contains peers whose ID only differs from the bot's ID in the least-significant bit.

If bot IDs are uniformly random across the whole of the overlay's address space, the probability of a peer falling within the distance range of the i^{th} bucket is 2^{i-128}. Thus, half of the bots in the botnet fall within the distance range covered by a bot's 127^{th} bucket, a quarter within the 126^{th} bucket, *etc.* As a result, there is a high likelihood that high-order buckets (*i.e.*, $127, 126, \ldots$) will be full while lower-order buckets (*i.e.*, $0, 1, \ldots$) will be empty. Table 2.2 illustrates this for a botnet of 20,000 bots: only the last 10 buckets (127-118) would normally be full, only 15 buckets (127-113) would normally have one or more entries, and each bot would normally only contain a total of 219 peer-list entries, just 1.01% of the whole botnet.

k-bucket Number	**Distance Range**	**Bots Covered**	**Bucket Entries**
127	$[2^{127}, 2^{128})$	10,000	20
126	$[2^{126}, 2^{127})$	5000	20
...	...	...	...
118	$[2^{118}, 2^{119})$	20	20
117	$[2^{117}, 2^{118})$	10	10
116	$[2^{116}, 2^{117})$	5	5
115	$[2^{115}, 2^{116})$	2	2
114	$[2^{114}, 2^{115})$	1	1
113	$[2^{113}, 2^{114})$	1	1
112	$[2^{112}, 2^{113})$	0	0
...	...	...	...
1	$[2^1, 2^2) = [2, 4)$	0	0
0	$[2^0, 2^1) = [1, 2)$	0	0
		Total	219

Table 2.2: Example of the k-bucket coverage and utilization for a typical bot in a botnet of 20,000 bots. The number of bots covered and the number of bucket entries are rounded to the nearest integer.

Every message sent within the botnet contains the ID of the sender so that the recipient can add the sender to its k-buckets. This is true regardless of whether the sender initiated communication or is replying to a request. To deal with this churn, Kademlia uses a *least-recently-seen* (LRS) eviction policy, illustrated in Figure 2.1.

At all times, each k-bucket is ordered from *least-recently-seen* (LRS) to *most-recently-seen* (MRS). When a message is received, the XOR distance between the

sender's ID and the recipient's ID is calculated, and the appropriate k-bucket is selected. If the sender is already within the k-bucket, the sender's entry is moved to the MRS position. If there is no entry for the sender and the bucket has fewer than k entries, the sender's contact information is inserted in the MRS position. Otherwise, if the bucket is full, the recipient PINGs the LRS peer. If the LRS peer fails to respond in a timely manner, it is evicted and the sender is inserted in the MRS position. Otherwise, the LRS bot is promoted to the MRS position and the sender's information is discarded.

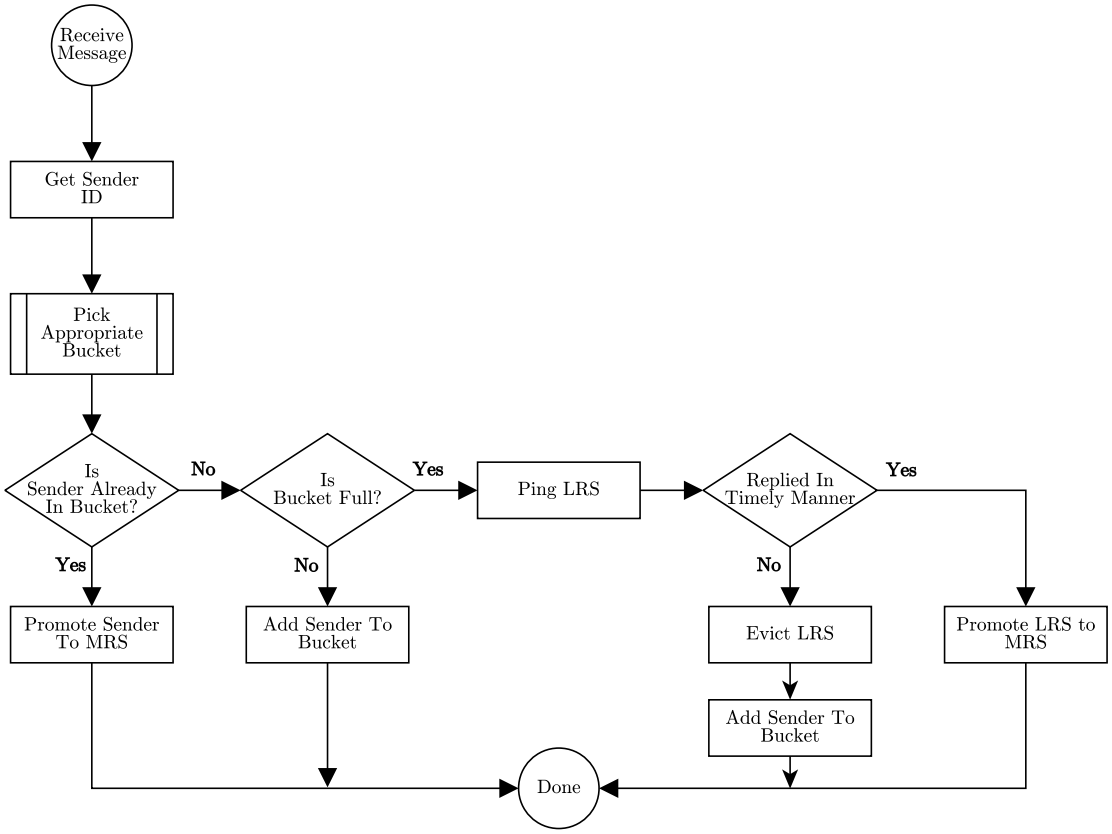


Figure 2.1: Least-Recently-Seen Bucket Eviction Policy

This LRS eviction policy serves two purposes. First, Maymounkov *et al.*[38] demonstrated that, in P2P file-sharing networks, the longer a peer has been online the more likely it is to remain online. Thus, the LRS eviction policy causes bots to favour peers which are more likely to remain online, improving the likelihood that a bot will always have active peers. Second, this reduces the churn within buckets,

preventing certain denial-of-service attacks, and also providing some resilience against sybil attacks. Sybils cannot simply flush bots' peer-lists by flooding the network with numerous, short-lived identities; instead, sybils must maintain identities for longer periods of time before they become effective at penetrating the botnet.

In order for a bot to participate in the botnet, it must have active bots in its peer-lists. Thus, for a bot to join the botnet, it must somehow obtain an initial peer-list. This process is known as "bootstrapping". The actual mechanics of the bootstrapping process are outside the scope of this work; bots simply receive contact information for a randomly-selected subset of the active peers in the botnet when they are create. The length of this subset is specified by the `bootstrapSize` parameter.

2.1.3 Remote Procedure Calls

From the Kademlia protocol, the botnet overlay protocol defines four *remote procedure calls* (RPCs):

1. **PING-PONG** - Tests whether or not another bot is still alive. The initiating bot sends a PING and waits for the recipient bot to reply with a PONG.
2. **STORE** - Instructs the recipient bot to store the specified {*key*,*value*}-pair for later retrieval by other bots.
3. **FIND NODE** - Searches for a list of bots closest to a key or ID in the network. The initiating bot specifies the key/ID and the recipient bot replies with a list of the k closest peers if at all possible. Fewer than k peers may be returned only if the recipient bot is returning all peers it has knowledge of.
4. **FIND VALUE** - Searches for a value by its associated key. The initiating bot specifies a key. If the recipient bot has the value, it replies with a STORE message containing the {*key*,*value*}-pair; otherwise, it responds identically to the FIND NODE RPC.

2.1.4 Lookups

The above RPC are used to perform lookups. While the original Kademlia paper[38] describes the lookup procedure as recursive, it is actually iterative[71]. The initiating bot begins by selecting the α (typically 3) closest peers from its k-buckets and inserting them into a *shortlist*. Each peer is sent a lookup request (*i.e.*, FIND NODE or FIND VALUE) in parallel and marked as *probed*. Each peer replies with a list of the

k closest peers from their k-buckets and the initiating bot updates its shortlist with all of these entries. When a bot replies to a lookup request, it is marked as *active*. If a probed bots fails to reply promptly, that bot is moved from the shortlist to a *waitlist*. If it eventually replies, it is then moved back to the shortlist and marked as *active*.

During the next iteration, the closest α unprobed peers from the shortlist are probed. This iteration begins once all previous lookup requests have finished or timed-out. Iterations continue until the first k entries of the shortlist have been marked active, or until the shortlist has been exhausted.

Using this general lookup procedure, three specific types of lookups can be performed:

1. **Node Lookup** - Locates the k closest bots to any key or ID. All requests sent during this lookup are FIND NODE messages.
2. **Value Lookup** - Used to retrieve a command stored in the botnet. All requests sent during this lookup are FIND VALUE messages. This lookup terminates immediately if a peer replies with a STORE message. At this point, the closest active peer in the shortlist (without the value) is also sent a STORE message.
3. **Value Publication** - Used to publish a {*key*,*value*}-pair into the botnet. The initiating bot performs a **Node Lookup**. Once the lookup terminates, STORE messages are sent to the k best candidates from the shortlist.

2.1.5 Bot Lifecycle

On connecting to the network, a bot is initialized with an initial peer-list as outlined in Section 2.1.2. Once the bot is bootstrapped into the botnet, it refreshes each of its k-buckets by performing a node lookup for a random ID that would fall into each bucket. Once these refreshes are complete, the bot has successfully joined the botnet and enters the running state.

While running, bots repeatedly carry out three actions. First, if any bucket has not been updated for `tRefresh` seconds, the bot issues a node lookup to refresh that bucket. Second, every `tValueLookup` seconds, if the bot has not retrieved all commands from the current command set it issues a value lookup for one of the remaining commands, selected at random. Finally, every `tReplicate` seconds, if the bot has successfully found any commands, it performs a value publication for each.

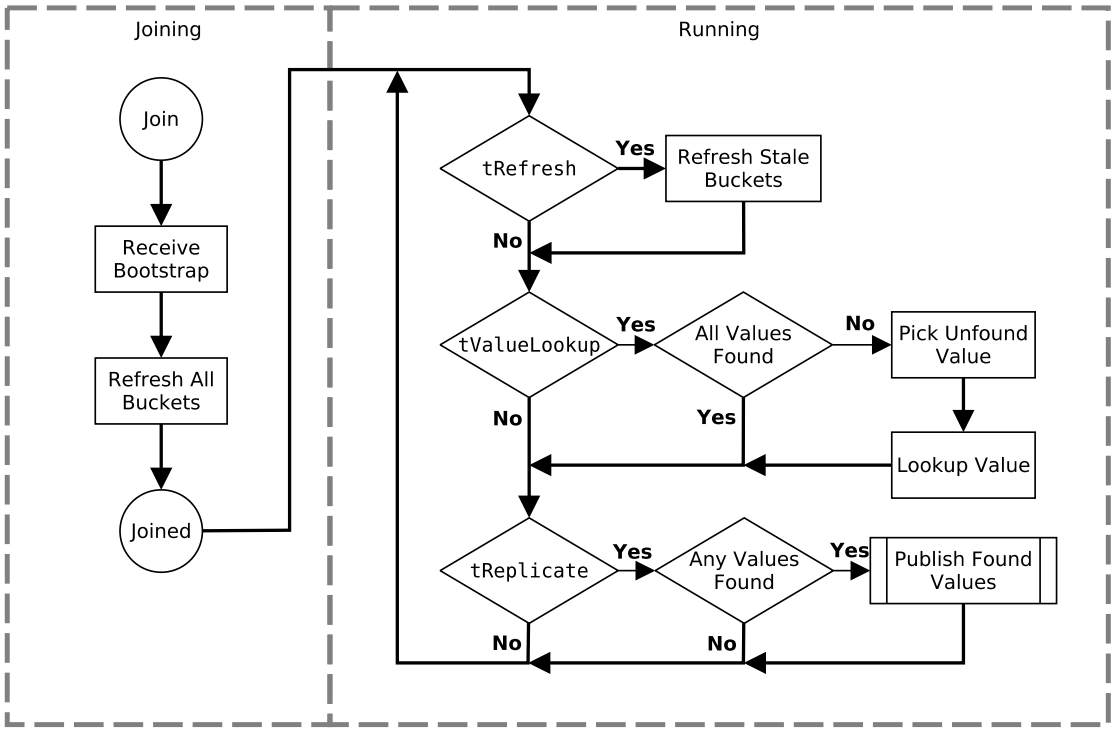


Figure 2.2: The Activity Lifecycle of a Bot

This lifecycle, illustrated in Figure 2.2, is slightly different for seed bots. Seed bots receive the full set of commands directly from the botmaster. Thus, after joining the botnet, they immediately perform value publication lookups for each command to ensure that the commands are located where normal bots are looking for them.

2.2 The Sybil Attack

As Section 1.5.3.2 outlined, the sybil attack[46] is a botnet mitigation strategy that can prove effective against P2P botnets. In fully-decentralized botnets, such as the Storm botnet, there is no system in place to regulate how bots create their identity (*i.e.*, overlay ID). Instead, bots are trusted to randomly generate and use a single quasi-unique identity since there is generally a low likelihood that two bots will generate the same ID. For example, if 20,000 values are randomly chosen from an address space of 2^{128} with a uniform probability of each value being selected, the probability

of any of these IDs colliding (according to the "Birthday Problem"[72]) is:

$$1 - \prod_{i=1}^{20000} \frac{2^{128} - i}{2^{128}} \approx 2 \times 10^{-24}$$

The sybil attack exploits the above assumption. A *sybil* is a computer that joins the botnet and communicates with legitimate bots using the botnet's P2P protocol. The defining characteristic of a sybil is that it creates multiple identities, allowing a single sybil to masquerade as multiple bots. This allows a sybil to gain a disproportionately-large influence over the botnet. Thus, the *sybil attack* is the insertion of one or more sybils into a botnet in order to gain influence over the botnet.

The rate at which the sybil attack gains influence over a botnet using a Kademlia-based protocol is directly proportional to the level of bot churn. This is due to the eviction policy bots use when maintaining their peer-lists (as described in Section 2.1.2): sybils will not be inserted into the peer-list of an existing bot until some of that bot's peer-list entries contain peers which have become inactive. Thus, if most bots are only active for short periods of time, sybils will gain influence over the botnet relatively quickly.

On its own, the sybil attack is not a mitigation strategy, but is rather a means of gaining influence over a botnet. Thus, in order to be effective as a mitigation strategy, the sybil attack must be combined with a disruption strategy which can leverage the influence gained through the sybil attack. As Section 1.5.3.2 highlighted, a common combination is to use the sybil attack with *index poisoning*[47, 48].

All P2P systems rely on some form of index in order to facilitate data retrieval. In the Storm botnet, the index is the set of commands stored as {*key*,*value*}-pairs by each bot in the botnet. In an index poisoning attack, sybils poison this index by sharing fake commands so that bots participating in the botnet will receive these fake commands instead of the legitimate commands. Since the index structure is distributed, bots which receive the fake commands will then replicate them to other bots, increasing the likelihood that bots will find the fake commands. When combined with the sybil attack, index poisoning has been shown to be an effective mitigation strategy against P2P botnets[49].

For the remainder of this thesis the combination of the sybil attack and index poisoning will simply be referred to jointly as a "sybil attack" for the sake of brevity.

2.2.1 Sybil Behaviour

For this work, the actual behaviour of each sybil identity is largely identical to normal bots. The main differences are: i) sybils do not initiate value lookups and ii) sybils always respond to FIND VALUE requests with the SYBIL VALUE. In terms of implementation, the actual value of the SYBIL VALUE is not significant beyond the fact that it is distinct from the correct values for all commands. However, the same SYBIL VALUE is used in all sybil replies in order to facilitate analysis. In a real sybil attack, the SYBIL VALUE would likely be a "no operation" command.

While other potential optimizations could be made to this sybil implementation, the goal of this work is not to measure the absolute effectiveness of the sybil attack. Rather, the goal of this work is to measure the relative effectiveness of the sybil attack when using different sybil placement strategies.

2.2.2 Fastest Response Path

In [48], Davis *et al.* highlight how the effectiveness of the sybil attack is innately constrained by the fastest response path. During a successful value lookup, the fastest response path is the path to the first non-sybil bot that responds with a STORE message. Any replies after this point are ignored. Thus, for sybils to be effective, they must be placed closer, in a response-path sense, than the fastest response path.

Within the botnet protocol of this work, there are two main factors that affect the fastest response path. First, lookups are iterative and do not iterate until some or all of the probed bots have responded or timed out. Thus, bots probed in early iterations are able to respond before bots in subsequent iterations. Second, within an iteration, bots which are close in the underlay network are typically be able to respond faster to lookup requests due to lower packet latency.

Both of the above factors provide ways in which the placement of sybils during a sybil attack can be optimized. The logical positioning of sybils within the P2P overlay can be optimized so that sybils are more likely to be selected in early iterations of command lookups. Likewise, the physical positioning of sybils within the underlay network can be optimized so that the average packet latency between sybils and bots is minimized, enabling sybils to respond faster than other bots within the same iteration.

2.2.3 Logical Placement Optimization

In [27], Davis *et al.* explored one possible optimization to the logical positioning of sybils: placing sybils logically close to where commands are initially be published in the P2P overlay. They concluded that this was no more effective than randomly placing sybils within the overlay. This would seem to indicate that while this placement strategy increases the likelihood that sybils are on the lookup path for commands, this placement strategy does not place these sybils closer, in a response-path sense, than the random placement strategy.

For this placement strategy to be effective in poisoning the botnet's index, sybils must be selected in early iterations of a given lookup. Similar to the k-bucket discussion in Section 2.1.2, if the keys of the commands published into the botnet are uniformly distributed throughout the botnet's address space, half of the commands on average fall under the address space of a bot's last k-bucket. There are 32 commands published at any given time, meaning that 16 are covered by the last k-bucket. Since this k-bucket is limited to only 20 entries, there is a very low likelihood that the k-bucket contains entries for sybils near each of the values in that k-bucket's address space. This means that other bots still have a chance of being selected for early iterations over sybils for at least some of the command lookups. Also, if the keys of commands are uniformly distributed throughout the botnet, a sybil positioned close to one command is further from other commands and thus is less likely to be on the lookup path for those other commands.

2.2.4 Physical Placement Optimization

In contrast to [27], this thesis explores an optimization to the physical placement of sybils within the underlay network. The underlay network model used in this work, which will be described in detail in Chapter 3, uses an AS-level network topology structure. Within this model, the physical placement of sybils in the underlay network is fully characterized by which AS each sybil is placed within.

Ideally, it would be possible to place sybils within every AS in the network topology (*i.e.*, unrestricted placement). However, with the number of ASes in the Internet topology already over 34,000 and still growing[73], the hardware requirements for such an attack are increasingly expensive. Furthermore, because of the multi-national nature of the Internet, legal and political issues further restrict which ASes sybils can be deployed within. Thus, it is much more likely that sybils will be deployed within

a subset of all possible ASes. These ASes will be referred to as *targeted ASes*, and this general approach will be synonymously referred to as both a *targeted sybil attack* and a *restricted* sybil placement strategy.

The goal of this work is to select the optimal set of ASes to target for sybil placement in order to maximize the effectiveness of the sybil attack. When each sybil identity is created, it is placed into a randomly-selected target AS. This work compares the effectiveness of 4 different sybil placement strategies: i) unrestricted placement, ii) uninformed (*i.e.*, random) placement, iii) fully-informed placement and iv) partially-informed placement.

The following sections will outline each placement strategy. The details of how each strategy is implemented will be discussed in Chapter 3.

2.2.4.1 Unrestricted Sybil Placement

The unrestricted sybil placement strategy assumes that it is possible to place sybils in any AS. Because this is technically not a targeted sybil attack, this placement strategy is distinct from the other strategies discussed below. The unrestricted placement strategy will be evaluated to demonstrate the effectiveness of the sybil attack without technical, political or legal limitations, *i.e.*, to generate an effective upper-bound on the effectiveness of the sybil attack.

2.2.4.2 Uninformed Sybil Placement

The uninformed sybil placement strategy is the naïve approach to selecting which ASes to target: target ASes are chosen purely at random under a uniform distribution. Such a strategy requires no knowledge of where bots are in the network topology, but also likely results in placing sybils in non-ideal locations. As such, this strategy should provide a lower-bound on the effectiveness of the sybil attack.

2.2.4.3 Fully-Informed Sybil Placement

The fully-informed sybil attack effectively uses a heat map of botnet traffic, prioritizing ASes through which high volumes of botnet traffic flow. In terms of execution, this approach requires observing botnet traffic at a global scale, calculating the heat map, and then placing sybils using the information gathered by the heat map. The goal of this approach is to place sybils so that they are centred between the most active bots in the botnet.

The procedure of detecting and monitoring botnet traffic across thousands of ASes makes this strategy impractical for real-world use; however, this strategy provides an effective upper bound on the sybil placement strategies that exploit using accessible network traffic observations.

2.2.4.4 Partially-Informed Sybil Placement

In the partially-informed sybil placement strategy a small set of bots (*e.g.*, 10) are compromised by defenders. Within the simulation, these bots are chosen randomly from the botnet. Using the information from their peer-lists, the routes between each compromised bot and its peers is traced (*e.g.*, using `traceroute`[74]). The number of botnet links traversing each AS are used as a measure of that AS's centrality within the botnet and sybils are deployed in the most-central ASes.

2.3 Measures of Effectiveness

To compare the effectiveness of the sybil attack when using each of the above sybil placement strategies, this work uses two measures: i) the number of sybil entries present in each bot's peer-list (*i.e.*, peer-list infection level) and ii) the number of bots actively participating in the botnet which have been able to successfully retrieve the correct value for each command (*i.e.*, value retrieval level). The peer-list infection level provides a measure of the influence gained by the sybil attack. As such it is not necessarily a direct measure of the effectiveness of the sybil attack, but rather a *proxy* measure for the effectiveness of sybil attacks. The value retrieval level provides a direct measure of the sybil attack's ability to poison the botnet's index and to prevent legitimate bots from retrieving commands from the botmaster.

2.4 Chapter Summary

P2P botnet protocols such as the protocol described in this chapter trust bots to create and use a single quazi-unique identity because they are fully decentralized and must allow new, randomly recruited bots to join the botnet. As a result, such a botnet is susceptible to the sybil attack.

Previous research has sought to quantify the effectiveness of the sybil attack against P2P botnets and to explore optimizations to the placement bots in the P2P

overlay network. However, these works have largely been limited to random graph models which are not able to assess the impact of i) network-level phenomena nor ii) the placement of sybils within the underlay network. Thus, this thesis outlines several different sybil placement strategies for optimizing the placement of sybils within the underlay network.

Finally, this chapter outlines the measures of effectiveness that are used to compare the performance of the sybil attack for each of the placement strategies.

Chapter 3

Simulation Model

In order to compare the efficiency and effectiveness of mitigating P2P botnets with different sybil placement strategies, a high-degree of control is needed over the botnet in question so that the sybil placement process can be as isolated as possible from other processes such as bot churn, background traffic, *etc*. Of the research approaches outlined in Chapter 1, packet-level simulation is the best suited approach because i) it allows for stochastic simulation of the botnet while still providing complete control over the processes in play (*e.g.*, botnet protocol, churn, background traffic, *etc*.) and ii) it allows the routing infrastructure and timing to be modelled.

This chapter will present an evaluation of existing tools for packet-level simulation and then discuss the implementation of a simulation model using one of these tools. Since the simulation model used in this work is based off of the work of Agarwal in [69], this chapter will then highlight the major contributions made to the simulation model throughout the course of this research. This chapter will also provide a detailed discussion of the implementation of each of the sybil placement strategies outlined in Section 2.2.4.

3.1 Design Criteria

There are a number of existing tools for network simulation. This section outlines several design criteria used in evaluating the strengths and weaknesses of each of these tools.

A preface to these criteria is that the hardware available for this research was a cluster of 41 IBM Blade servers. Each of these machines offers two 3GHz Intel Xeon

processors, 8GB of RAM and 60GB of storage. The chosen simulation tool must make the best possible use of this hardware.

3.1.1 Realistic Network Topology

The first and most important criterion is support for realistic network topologies. The simulation model's network topology is responsible for modelling network phenomena such as routing delays and traffic bottlenecks. Furthermore, research has shown that network simulations are sensitive to the topology used[75]. Thus, to ensure a high level of fidelity for simulations, the network topology model must accurately reflect the structure and attributes of the Internet.

There are generally two approaches to generating network topology structures. The first approach is to generate a random Internet-like topology structure using an informed topology generator[76]. The second approach is to use a measured internet topology structure, such as those measured by the CAIDA Internet Mapping and Annotation project[77].

Measured topologies have the advantage that they represent the actual graph structure of the Internet, not just a graph with several measurably-similar properties. This is important since the behaviour of network simulations has been proven to vary depending on the underlying network routing topology[75] and mitigation strategies must ultimately be effective on the Internet and not a theoretical network topology. However, by also supporting generated topologies, the model can be used to quantify how behaviours vary across multiple network topologies. If the effectiveness of a particular botnet mitigation strategy varies widely across different network topologies this may limit its applicability when used in-the-wild. Thus, it is desirable to support both means of generating topology structures.

Another consideration with network topology structures is whether to use an AS-level topology or a router-level topology. Router-level topologies provide a higher level of detail than AS-level topologies, but at the cost of increased resource demands. Also AS-level topology information is much easier to measure since this can be gathered from publicly-visible *border gateway protocol* (BGP) information while ISPs generally do not publish their router-level topologies because of privacy concerns[78]. Because of this there are existing data sets that measure the AS-level topology of the Internet, whereas there are no such data sets at a router level. As a result, this work focuses on AS-level topologies.

Beyond the mere structure of the network topology, it is also important that the properties of links between nodes in the network topology, such as bandwidth and propagation delay, are accurately modelled[64].

3.1.2 Facilitating Statistical Measurements

As was discussed in Section 1.6.3, stochastic simulation tends to be conducive to statistical testing since it provides better controllability and repeatability than using real systems while providing a generally greater level of fidelity than analytical models. The downside of using simulation, with respect to analytical modelling, is that numerous simulation runs must be conducted and analysed in a statistically rigorous manner. However, there are two ways that a simulation framework can help reduce the time and effort necessary to conduct all of these simulation runs.

First, the framework can reduce the execution time of each individual simulation run by supporting the saving and loading of simulation state. This allows the initial state of the network to be generated once and then used for multiple repetitions without repeating the time and work of generating that state. Second, the execution time of ensembles of simulations can be reduced by using a distributed automation framework to run multiple simulations in parallel in order to leverage all 41 IBM Blade servers used in this work.

The state preservation mechanism is also important for statistical testing because it facilitates altering the initial state of the botnet separately from the random processes (*i.e.*, the seeds of the PRNGs) of each repetition. If a previous botnet state can be loaded by a simulation, then the initial botnet state is established separately from the random processes (*i.e.*, PRNG seeds) of each repetition. Without this mechanism, it is more difficult to achieve this level of control.

3.1.3 Scalability

The chosen simulation tool must scale well to networks of realistic sizes. This means that the run-time execution rate and memory usage of the simulation tool must not degrade severely as both the size of the underlying network topology and the size of the overlay network are increased.

3.1.4 Extensibility

The simulation tool should minimize the effort necessary to implement the application-layer botnet protocol. Toward this end, open-source tools are preferable to close-source tools since they can be customized to accommodate unanticipated needs. However, since open-source projects also commonly draw on community-driven development, it is also important that the framework components be verifiably correct.

There is a general trade-off between generalization and performance: frameworks try to provide components that will be useful to a wide audience under a wide variety of circumstances, but because of this fewer assumptions and limitations can be placed on these components, meaning that fewer optimizations can be made to them. For this work, scalability is considered more important than extensibility due to the large number of simulation runs needed to conduct statistically rigorous testing.

3.2 Existing Simulation Frameworks

Using the above set of design criteria, this section evaluates a number of existing network simulation frameworks that are publicly available.

3.2.1 NS-2

NS-2[65] is a well established, open-source, discrete-event network simulation framework. Having received support through DARPA and corporate initiatives, NS-2 comes pre-packaged with a large number of well-tested components that facilitate simulation of numerous network protocols in both wired and wireless networks. The open-source nature of NS-2 also makes it extensible to novel protocols and problems.

Despite its popularity and wide-spread use, NS-2 suffers from a number of early design decisions which limit its capability. In [79], Weingartner *et al*. demonstrate that as the number of nodes in a benchmark network simulation increased, most comparable simulation frameworks saw linear increases in run-time whereas NS-2 saw exponential increases.

3.2.2 NS-3

NS-3[80] is the successor to NS-2, designed with the goal of addressing the scalability issues of NS-2. While NS-3 has generally achieved these goals, it is not backwards-

compatible with NS-2. This means that NS-3 cannot leverage the suite of well-tested components from NS-2. Development of an equivalent suite of tested components for NS-3 remains an ongoing issue.

3.2.3 Möbius

Möbius[67] is a modelling tool designed for system-level performance and dependability testing. Its key design goal is to support multiple modelling formalisms and solution methods. The supported modelling types include *stochastic activity networks* (SANs), place/transition networks (*i.e.*, Petri nets) and queue-based networks. Using these model types, Möbius is able to perform three types of solving: i) analytical solving, ii) numerical, state-space solving based on compact *multivalued decision diagram* (MDD) Markov processes and iii) distributed discrete-event solving techniques.

The main shortcoming of Möbius with respect to this work is that it is generally aimed at high-level modelling. As a result, Möbius does not have a large suite protocol of models which are suitable for accurate packet-level simulation. As a result, it is not well suited for this work.

3.2.4 PRIME SSF

PRIME SSF[81] stands for Parallel Real-time Immersive network Modelling Environment Scalable Simulation Framework. This framework is designed to conduct simulations that interact with real-world systems by using a combination of simulation and emulation.

While integrating with physical devices has the potential to increase the fidelity of experiments conducted with this tool, it does so at the expense of control, repeatability and scalability. The behaviour of physical devices and networks is nondeterministic which prevents exactly replication of simulation results. Also, the use of physical devices prevents access to their internal state and settings, reducing the information and control that researchers have access to. Finally, the use of physical devices increases the hardware resources needed to conduct each experiment, making large-scale network simulation less feasible.

3.2.5 PlanetSim

PlanetSim[82] is an open-source overlay network simulation framework. It uses a layered design to isolate the implementation of the overlay protocol from any underlay concerns, but still carries out a full packet-level simulation. Furthermore, the framework comes with reference implementations of both structured (*e.g.*, Chord, Symphony and SkipNet) and unstructured (*e.g.*, Gnutella) overlay networks. Finally, because the project is open-source, the framework can be extended to provide new overlay protocols and to support different underlay networks.

While well suited to the general goals of this simulation, this project has a number of significant disadvantages. The chief limitation of this project is its lack of support for realistic underlay network: the project only has support for several basic random graph topologies and as of its last publication had only started to introduce a churn model. Also, the project appears to be largely dormant, with the last publication written in 2009[83].

3.2.6 OMNeT++

OMNeT++ [84] is an open-source discrete-event simulation engine. A core part of this project is the INET framework which provides implementations of a large number of wired and wireless network devices and protocols. A number of NS-2 modules have been ported for use with OMNeT++, providing well established references against which the OMNeT++ implementations can be validated, lending the credibility of NS-2 to OMNeT++. OMNeT++ also has a strong community base which has been growing steadily for nearly a decade. This, combined with the open-source nature of the project has contributed to a comprehensive suite of reference modules and documentation which greatly facilitates the creation of new models using this framework. OMNeT++ has also been shown to scale well, closely matching the performance of NS-3 [79], which makes it well suited to large-scale network simulations. Finally, there are several automation frameworks available for distributed, multiple-replications-in-parallel execution of OMNeT++ simulations, including Akaroa[85] and STARS[57]. As a result, OMNeT++ is well suited to the purposes of this work and is used to develop the botnet simulation model.

3.2.6.1 OverSim

OverSim[86] is an open-source overlay and P2P network simulation framework for OMNeT++ that provides reference implementations of a number of structured and unstructured P2P overlay protocols. However, there there are three main issues with OverSim with regard to the design criteria of Section 3.1.

First, at the onset of this work OverSim had no support for realistic network topologies. The only supported topology was a multi-tiered random graph structure. Nodes within a tier were structured as an Erdős-Renyi graph and a portion of nodes within each tier were also randomly connected to nodes in adjacent tiers. At the time of this writing, OverSim has added support for more realistic network topologies via the ReaSE topology generator[78] which is able to generate network topologies which are directly compatible with the OverSim framework. However, ReaSE still has the limitation that it only supports generating arbitrary, random topologies; it is not possible to use ReaSE to import a measured AS-level topology structure of the Internet and augment this with realistic link characteristics.

Second, OverSim has no support for saving simulation state. As Section 3.1.2 highlighted, this results in wasted effort in generating the initial network state for each repetition of a given experiment and creates difficulties when conducting statistical testing.

Finally, while OverSim is generic and extensible enough to support numerous P2P overlay protocols, this comes at the cost of scalability. Table 3.1 compares the rate of execution and memory usage between comparable simulations conducted using OverSim and the simulation model that was developed in this work. OverSim is running a 20,000 node Kademlia overlay while the thesis model is running a 20,000 node Kademlia-based botnet. Both simulations use a 2500 node AS-level underlay network topology that was generated using the ReaSE topology generator. The thesis model completed a 24 hour simulation after approximately 3 days while the OverSim simulation crashed after approximately 7 days having only simulated approximately 30 minutes. This crash is due to bugs arising from the fact that OverSim does not implement plain Kademlia, but instead implements a variant of Kademlia called S/Kademlia[71] and it is not trivial to disable the additional features of this variation. At the end of these simulations, the OverSim model consumed more than 2.5 times the memory and ran over 100 times slower. While not a full comparison of the two

tools, this still shows that OverSim requires significant additional CPU time and memory compared to the simulation model used in this work.

	OverSim	Thesis Model
Execution Rate (% of Real-Time)	0.22%	30%
Memory usage	4729MB	1800MB

Table 3.1: Comparison of OverSim and the simulation model used in this thesis when simulating 20,000 overlay nodes on a 2500 AS network topology.

3.2.7 Summary

Of the tools summarized above, OMNeT++ was chosen for this research because of its large base of validated reference components and scalability. While the OverSim framework would provide a means of quickly implementing a working simulation model, the issues outlined in Section 3.2.6.1 render it unsuitable for rigorous statistical evaluation of large-scale botnets. As a result, this work makes use of a custom OMNeT++ simulation model which is discussed in detail in Section 3.3.

While several automation tools exist for OMNeT++, the simulation model was previously integrated with STARS in [2]. This research proceedes using STARS for simulation automation because of this existing support.

3.3 Simulation Model

This section will outline the architecture of the simulation model used in this work. While this model is based off of the simulation model developed by Agarwal in [69], a number of changes and extensions have been made to it throughout the course of this work. Thus, this section will discuss the overall architecture of the resulting model used in this work and Section 3.5 will highlight and discuss the changes and extensions made.

3.3.1 OMNeT++ Overview

OMNeT++ is an object-oriented discrete-event simulation framework[84]. The simulation model is composed of *modules* which communicate and interact by sending *messages* to one another via *gates*.

Modules represent the physical and logical components of the system. There are two types of modules: i) simple modules and ii) compound modules. Simple modules hold state and implement active behaviours while compound modules are used to group and connect modules (both simple and compound) into larger, more abstract entities. Thus, compound modules are used to form the hierarchy of modules within the simulation model. The OMNeT++ *network definition* (NED) language is used to declare modules types and module properties while any associated functionality is implemented using a C++ interface. Thus, compound modules only consist of a NED file whereas simple modules consist of both NED and C++ files. For compound modules, the NED file also declares how the sub-modules are connected.

Messages represent individual events in the model. Messages can be scheduled to occur either immediately or at any time in the future and can also contain data. As a result, messages can be used for two general tasks: i) triggering events at a specific time and ii) passing information between modules. Intermodule messages can either be sent directly to the gate of the receiving module or between gates that have been connected by a series of one or more *channels*. Channels can have associated properties and functionality, useful for modelling propagation delays, *etc*. Similar to modules, channel types and properties are declared using the NED language while any functionality is implemented in C++.

In addition to the NED and C++ components, simulations runs can use one or more *configuration* (INI) files to set module and channel properties for an experiment. A single INI file can specify multiple parameter configurations and the number of repetitions of each parameter configuration to be conducted. OMNeT++ also supports exact replication of simulations via the Mersenne Twister[87] PRNG.

When running a simulation, OMNeT++ offers both *graphical user interface* (GUI) and *command line interface* (CLI) simulation environments. The Tk/TCL-based GUI is useful for visualizing and debugging small-scale simulations. However, the GUI generally results in lower speed of execution and can become quite sluggish when rendering large numbers (*i.e.*, thousands) of modules. As a result, the CLI environment is preferable for executing large-scale simulations as well as batches of simulations. For operating systems without a GUI, the CLI is the only simulation environment available.

There are a number of component libraries available for OMNeT++. The INET framework [88] is particularly noteworthy as it provides implementations of standard wired and wireless networking protocols and devices. The INET framework is used

in this research to ensure correct implementations of the protocols and devices used in the underlay network environment.

3.3.2 Underlay Network Model

The underlay network model represents the physical network over which bots send packets. This portion of the model is also concerned with the location of bots within the underlay network and their churn. Table 3.2 summarizes the parameter settings that characterize the underlay network model.

Attribute	Description
`numberOfAses`	Number of ASes in the network topology
`numberOfSubnets`	Number of Subnets in the network topology
`initialBots`	Initial number of bots in each subnet. Only used if not starting from saved botnet state.
`maxBots`	Maximum number of bots that can be in a single subnet.

Table 3.2: Summary of Underlay Network Topology Parameters.

3.3.2.1 Network Topology

The core of the underlay network model is the network routing topology. This consists of a static AS-level network topology and a number of connected subnets which represent ISPs, each of which can contain a number of bots. The internal routing structure of each subnet and how each bot is connected to that routing structure (*i.e.*, wired *vs.* wireless) is abstracted away for scalability. This overall architecture is illustrated in Figure 3.1. Each AS can have at most 1 attached subnet. ASes with no attached subnet are referred to as *transit ASes* whereas ASes with an attached subnet are referred to as *destination ASes* since they form the set of routable destinations within the AS-level topology.

Each AS is implemented in OMNeT++ as a single `router` module. These `router`s are interconnected with channels that specify the available bandwidth and routing delays of the link. In real network systems large ASes are implemented as a *point-of-presence* (PoP). Rather than inferring a structure for all PoPs and to improve scalability, the simulation model accounts for PoPs by adding additional delays to all links to and from large ASes. Because the network topology is static, all `router`s

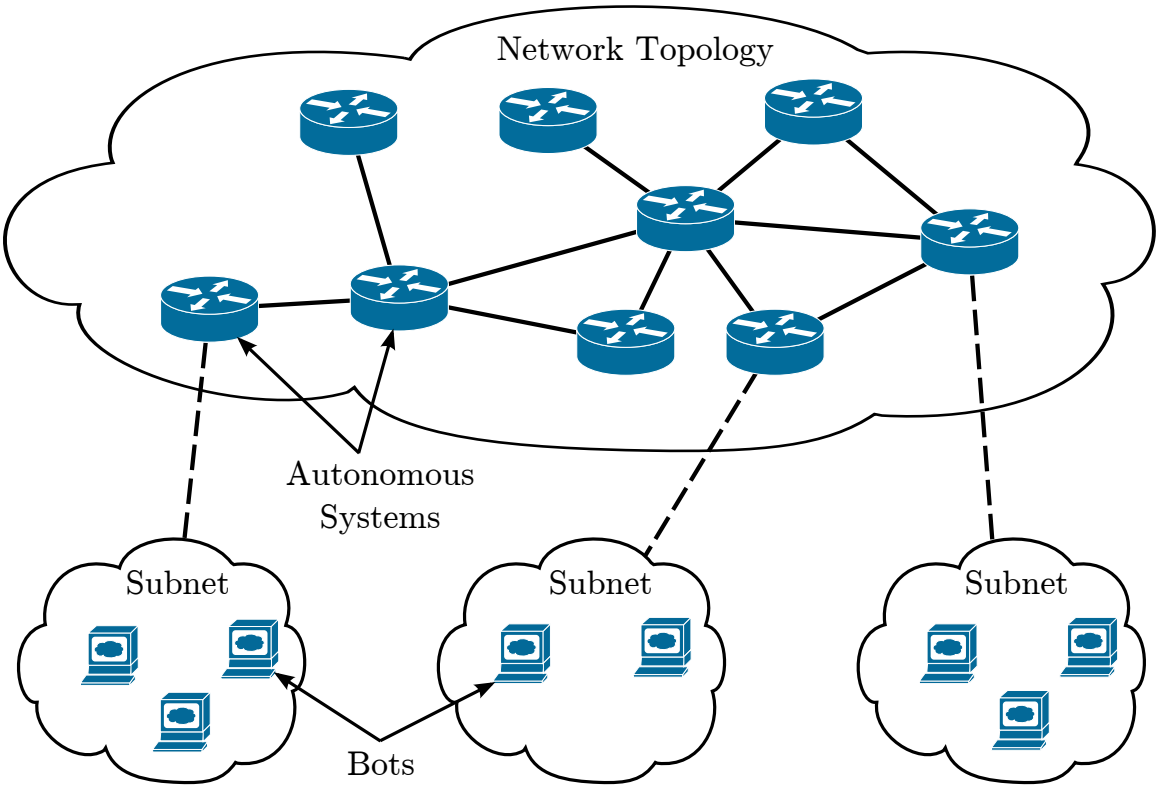


Figure 3.1: Underlay Network Architecture

use static routing tables instead of dynamic routing protocols in order to improve scalability. These static routing tables are stored in files which are loaded at the beginning of the simulation, allowing these files to be used for multiple simulation runs.

Each `subnet` module is connected to one of the ports of the `router` module representing the AS it belongs to. Within a `subnet` module, the routing infrastructure is abstracted away. As illustrated in Figure 3.2, subnets contain a `networking` module responsible for modelling the transport-level protocol used by bots and a `message mapper` module which is responsible for i) modelling intra-subnet routing delays and ii) transferring packets between the `networking` module and the `bot cluster` module. The `bot cluster` module is simply a container module that contains the modules used to model each individual bot. Because all bots within a subnet share a single `networking` module, all bots use the same transmission protocol: the *user datagram protocol* (UDP). This protocol reduces the complexity and overhead in simulating

botnet traffic and ensures that measured traffic characteristics are not merely an artefact of a connection-oriented protocol or congestion control mechanisms.

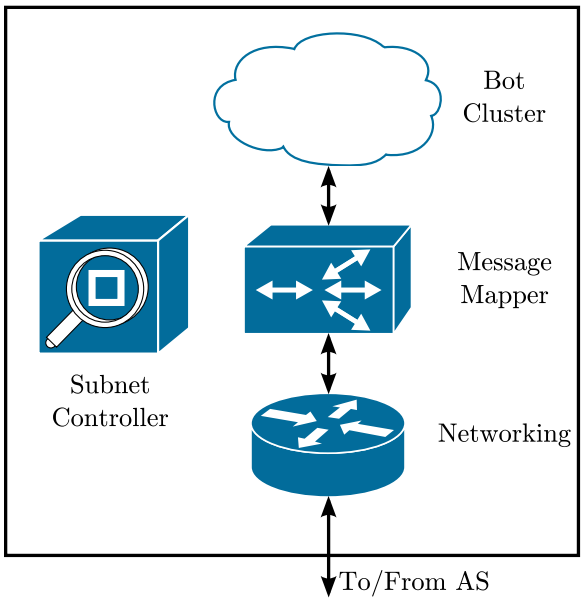


Figure 3.2: Internal Module Structure of the `subnet` Module

Subnets also contain a `subnet controller` module which serves two functions. First, each `subnet controller` maintains a list of all bots in the subnet and provides a means of looking up bot modules by their IP address. This lookup functionality is used by the `message mapper` to look up bots when delivering inbound packets. Second, each `subnet controller` manages the creation, initialization and removal of bots within the subnet. In creating a new bot, the `subnet controller` delegates all protocol-specific initialization tasks (*i.e.*, generating bot parameter settings, bootstrapping, *etc.*) to protocol-specific modules. This process will be described in more detail in Section 3.3.3. `Subnet controller`s do not actually trigger churn events—they only carry out the mechanics of creating and removing bots. Churn events are triggered by a global `churn manager`.

3.3.2.2 Churn Management

The `churn manager` is responsible for triggering all churn events throughout the whole underlay network. The churn process is characterized by:

- `tBotCreation`: The distribution describing the inter-arrival times of bots joining the botnet (*i.e.*, the birth-rate).

- `tBotRemoval`: The distribution describing the inter-arrival times of bots leaving the botnet (*i.e.*, the birth-rate).

Whenever a churn event is triggered it is then forwarded to a randomly selected subnet that can handle the event. For bot creation, this means that there must be room in the subnet for more bots. For bot removal, this means that there must be bots in the subnet that can be removed.

There is also a second, *targeted* churn process that can run parallel to the normal churn process. It functions identically to the normal churn process, except that i) it only sends churn events to subnets that have been marked as targeted, ii) it manages a separate set of hosts (*i.e.*, hosts created by the targeted churn process can only be removed by the targeted churn process) and 3) all modules involved in initializing bots use a separate set of PRNGs when initializing targeted bots.

For this work, the targeted churn process is used to handle sybil bot churn as a separate, isolated random process from normal bot churn. As a result, it is possible to isolate the effect that adding sybils can have on a botnet; subsequent runs with and without sybils, if seeded identically, will only vary because of the behaviour of the sybils, not because of side-effects from creating sybils. Furthermore, it is possible to alter the physical placement of sybils within the underlay network without altering their logical placement in the overlay network.

Each `subnet` module has two parameters which determine if it is targeted or not: i) `targetingPriority` and ii) `numberOfTargets`. `numberOfTargets` is always set uniformly across all `subnet`s whereas `targetingPriority` can vary from `subnet` to `subnet`. In general, if `targetingPriority` is non-negative and less than `numberOfTargets`, the `subnet` is targeted. The only exception to this is if all `subnet`s have a negative `targetingPriority` but the `numberOfTargets` is greater than 0, in which case the number of targets specified by `numberOfTargets` is randomly selected from all `subnet`s. Using this, the various targeting strategies from Section 2.2.4 can be implemented, as summarized in Table 3.3.

Of special note is the "Informed" case. This is used for both the partially and fully informed cases. Each subnet is assigned a unique priority: the highest priority is 0 and the lowest priority is `numSubnets` $- 1$. The targeting priorities are calculated by external tools and saved in the simulation's configuration files. This allows the priorities to be calculated for a particular botnet once and then re-used in multiple simulation runs with differing numbers of targets. The details of how these priorities are calculated will be discussed in Section 3.3.6.

Strategy	Targeting Priority	Number of Targets	Notes
No Targets	-1	-1	Default configuration.
Unrestricted	0	1	All `subnets` are targeted.
Uninformed	-1	N	N `subnets` are chosen at random.
Informed	$\{0, 1, \ldots,$ `numSubnets` $- 1\}$	N	Top N `subnets` are chosen.

Table 3.3: Summary of Targeted Churn Parameters

3.3.2.3 Bots

The `bot` module, as illustrated in Figure 3.3, encapsulates the application-layer of each bot and is composed of an `overlay protocol` module and a `host message processor` module. All messages to and from the `overlay protocol` module are first filtered by the `host message processor` module. For inbound packets, this module filters out packets destined for the bot's IP address but not the bot's overlay ID which can happen as IP addresses are reused by new bots because of churn. For outbound packets, the `host message processor` adds the necessary framework-related fields to the packet so that it can be properly routed to the destination `bot` module. The `overlay protocol` module implements the host-level behaviour of the overlay protocol discussed in Section 2.1.

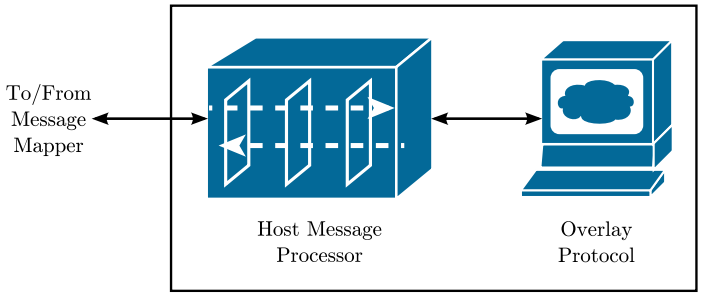


Figure 3.3: Internal Module Structure of the `Bot` Module

3.3.3 Overlay Model

There are two functional components to the overlay network model: i) initializing new bots and ii) the implementation of the host-level botnet protocol.

3.3.3.1 Initializing Bots

Bot initialization is managed by the `subnet controller` module which delegates protocol-specific initialization to a number of modules in the `overlay controller` module, as pictured in Figure 3.4. The `parameter generator` generates and directly sets any parameters needed by the `bot` module's `overlay protocol` module. The `bootstrap generator` module creates the initial list of peers used to bootstrap the new bot into the botnet. Finally, the `key-value pair generator` module generates the set of commands that bots will attempt to look up.

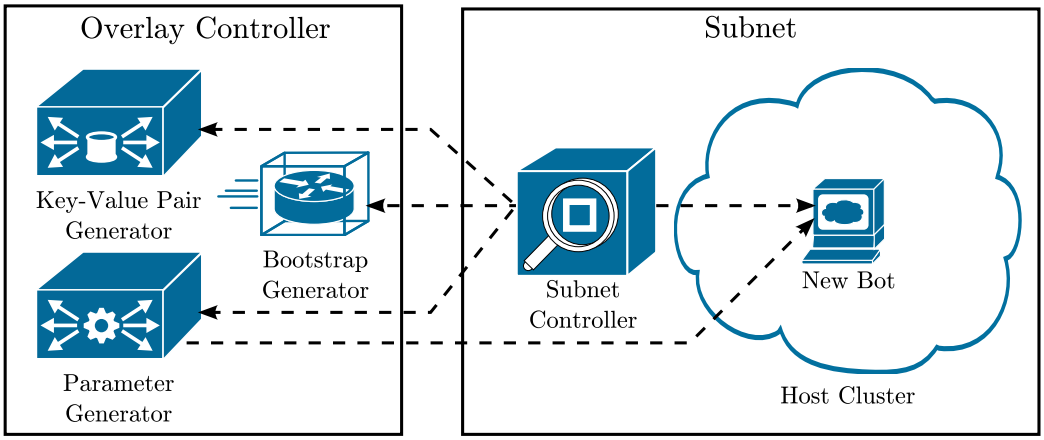


Figure 3.4: Interaction of Modules When Initializing a New Bot

3.3.3.2 Protocol Implementation

There are three distinct types of bots within the simulation: i) normal bots, ii) seed bots and iii) sybil bots. Each bot's `overlay protocol` module implements the botnet's overlay protocol which was discussed in detail in Section 2.1. However, seed and sybil bots function differ slightly from normal bots.

Seed bots receive the full list of commands that bots are searching for immediately after they are created. Once seed hosts have received this list, they then attempt to publish each command within the botnet using value publication lookups (discussed

in Section 2.1.4). Also, since these bots have all the commands from the current command set, they do not perform value lookups.

From Section 2.2.1, sybil behaviour differs from normal bot behaviours in that: i) sybils do not initiate value lookups and ii) sybils always respond to FIND VALUE requests with SYBIL VALUE. As Section 2.2 outlined, sybils are characterized by their use of multiple identities. However, to simplify the implementation of sybils, each sybil identity is modelled using a separate `bot` module rather than as a single `bot` module with multiple identities. This way all types of bots only manage a single identity, minimizing the differences between the protocol implementation of each. Also, within a subnet, the physical infrastructure is abstracted away, meaning that the concept of individual computers within the subnet is not modelled. The only notable side-effect of modelling sybils this way is that every sybil identity has a separate IP address as this is necessary for the sake of routing packets within the simulation model. As a result, analysis can only produce measures relative to identities, not computers.

3.3.4 Simulation Modes

The simulation model supports two general modes of operation: i) *generation* and ii) *experimentation*.

3.3.4.1 Generation

Generation mode is designed to generate and save the state of a realistically-structured botnet. An initial number of bots are uniformly distributed throughout the underlay network and churn is configured so that bots are added faster than they are removed, resulting in an overall growth in the size of the botnet. The botnet runs and grows according to its P2P overlay protocol until the botnet reaches a user-specified size. At this point the simulation state is saved. Under generation mode all instrumentation is disabled in order to minimize the run time of the simulation.

3.3.4.2 Experimentation

Experimentation mode is designed for loading a previously saved simulation state and running that network for a set duration under a specified setting of the tunable parameters of the model. During these runs, various instrumentation is enabled to measure behaviours and observable characteristics of the botnet. In this work, the

experimentation mode is combined with a number of different simulation configurations in order to implement the sybil placement strategies outlined in Section 2.2.4. Instrumentation is discussed in detail in Section 3.3.5.

Because of the large number of simulations needed for statistically-rigorous measurements, the simulation model is integrated with the STARS framework[57]. The details of this integration process are discussed in Section 3.4. While there is STARS support for both generation and experimentation modes, it is of most benefit to the experimentation mode since the majority of simulation runs use in this mode.

3.3.5 Instrumentation

While OMNeT++ has built in support for scalar and vector instrumentation, these implementations are too limited for this research. First, measurement files are written in a text-based format and cannot be compressed on-the-fly. If a simulation generates large volumes of data, these files can easily exhaust all available disk space before the simulation finishes. Second, all vector measurements are grouped into one file and scalars into another file. This increases the complexity of analysing data since each measurement series must be separated before preforming analysis. This also requires extra care when processing large measurement files so as to not exhaust all available memory.

For these reasons, the simulation model leverages two alternative forms of instrumentation. For time-series vectors, data is collected using the instrumentation tools provided by the STARS framework while other data is collected using custom instrumentation. The STARS instrumentation writes binary-formatted files that can be compressed on-the-fly and which are compatible with the STARS MATLAB analysis tools. The custom instrumentation writes *comma-separated variable* (CSV) formatted files which can also be compressed on-the-fly and are analysed using custom analysis scripts.

While it is undesirable to have two forms of instrumentation, this strategy leverages the verified STARS analysis scripts where possible, using custom instrumentation and analysis only for data series which are not compatible with the STARS analysis scripts.

3.3.6 Sybil Placement Strategy Implementations

Section 2.2.4 outlined four sybil placement strategies: i) unrestricted placement, ii) uninformed (*i.e.*, random) placement, iii) partially-informed placement and iv) fully-informed placement. The following sections will outline how each placement strategy is implemented in the simulation model.

3.3.6.1 Unrestricted Sybil Placement

As Section 3.3.2.2 outlined, this placement strategy is built in to the `churn manager` module. By specifying the correct parameter settings, sybils are distributed randomly into all subnets in the underlay network.

3.3.6.2 Uninformed Sybil Placement

Similar to the unrestricted sybil placement strategy, this placement strategy is built into the `churn manager` module; however, the number of subnets to be targeted must be explicitly specified via the `numberOfTargets` parameter.

3.3.6.3 Partially-Informed Sybil Placement

The partially-informed sybil placement strategy requires access to the peer-lists of bots in the botnet. Because of how the botnet state is saved, the only means of extracting this information is to launch an OMNeT++ simulation which loads the state. At this point, the peer-lists from each bot are written to file in a format which is easy to load for analysis and the OMNeT++ simulation terminates.

Afterwards, the topology generation tool (discussed in Section 3.5.1) is used to reload or regenerate the network topology. The peer-list information from each bot is then loaded and a random set of these peer-lists is selected for tracing. Routes between the selected bots and each of their peers are traced and recorded at each AS along the route, provided the AS is a destination AS; transit ASes cannot be targeted since bots cannot be placed into them. Afterwards, each subnet is prioritized by the number of routes traced through the associated AS. This information is appended to the INI file used to launch the above OMNeT++ simulation. At this point, it is finally possible to run an OMNeT++ simulation with targeted sybil placement.

3.3.6.4 Fully-Informed Sybil Placement

The fully-informed sybil placement strategy requires a record of how many botnet packets pass through each destination AS. This information is gathered while running the botnet without sybils, meaning that this placement strategy requires two simulation runs: i) an information gathering run without sybils and ii) a measurement run with targeted sybil placement.

During the first run, as each packet traverses a destination AS, the `router` module logs the sender and recipient bots. At the end of the simulation, this data is aggregated into flow records for each subnet since each destination AS is associated with a single subnet. These flows are characterized by: i) the sender, ii) the recipient and iii) the number of packets sent between these bots. Because of the sheer volume of data collected during a simulation run, all packet flows with fewer than 3 packets are ignored. The subnet with the highest number of observed botnet packets is assigned the highest priority. Each sender-recipient pair observed by this subnet is then removed from consideration in the remaining subnets. This process is repeated with the remaining subnets until all subnets have been prioritized.

The priorities are saved so they can be used in multiple simulations with different numbers of target ASes. At the beginning of each of these runs, the priorities are appended to the simulation's INI along with the number of subnets to target.

3.4 STARS Integration

The *statistically rigorous simulation* (STARS) framework[57] is an MPI-based framework for automating distributed, parallel execution of simulations and statistically-rigorous analysis of generated measurements. In [2], Godkin provides a detailed discussion of how STARS performs its statistically-rigorous analysis. As that work highlights, the statistical analysis tools used by STARS require stationary, time-series data. The work in this thesis, however, violates both of these constraints. The sybil attack seeks to disrupt the botnet's communication and fracture the botnet's overlay network which effectively means that it seeks to introduce non-stationary behaviours. Also, the STARS analysis scripts are only suitable for simple time-series data whereas much of the data necessary to analyse the structure and connectivity of the overlay network graph is complex and only saved periodically. Thus, for this thesis, the STARS framework was used primarily for simulation automation.

When STARS is deployed across a cluster of computers, one computer is designated as the *management* node while all other computers function as *worker* nodes. By default, the management node also doubles as a worker node. Users submit *processes* which contain a series of *tasks* (*i.e.*, simulations and/or analysis). The management node manages: i) the distribution of tasks and their related resourced to worker nodes and ii) the collection of results from worker nodes when they finish tasks.

STARS requires several user-supplied components in order to integrate a simulation with the framework. First, STARS requires a *resource package*. The resource package contains all of the resources (*i.e.*, executables, configuration files, *etc.*) necessary to run any OMNeT++ simulations and analysis. Second, each batch of simulations requires a *workfile*. This specifies the experiment configuration which includes both STARS and OMNeT++ configuration settings. Finally, STARS requires several Python modules which specify: i) how to interpret the workfile and ii) the mechanics of running each simulation and processing any results. These modules are packaged with the resource package.

In [2], Godkin provides a high-level discussion of integrating STARS with an earlier version of the simulation model used in this work. For that work, minimal effort was required to integrate the simulation model with STARS because the run procedure and measurements were both well-suited to STARS. However, for this thesis, there were five types of run configurations:

1. Generation
2. No Sybils (with Router Logging)
3. Uninformed Sybil Attack
4. Partially-informed Sybil Attack
5. Fully-informed Sybil Attack

Each configuration uses a base OMNeT++ INI file, minimizing the amount of additional configuration that needs to be specified by the workfile. Most of these run configurations require a distinct run procedure (outlined in Section 3.3.6) and generate different results. Thus, it makes sense to provide separate modules for each. However, between a number of the configurations, portions of the procedure and results are the same. Thus, it makes sense to organize these modules hierarchically, leveraging inheritance to reduce code duplication between modules.

Because of how STARS loads modules, this hierarchy requires that STARS load the modules in a specific order (*i.e.*, load dependencies first, then dependants). However, STARS provides no means of specifying the order in which to load modules. Thus, this work contributes a mechanism for specifying which order to load modules in. A file named "`load_order`" is included with the Python modules. If STARS detects this file, it first loads the modules specified in this file in the order that they are listed before loading any additional modules included in the resource package.

Another issue that arose with STARS is that it only manages the deployment of the resource package to worker nodes. If an analysis task wishes to use other resources, it must manually retrieve them after being deployed to a worker. This functionality is built into the MATLAB analysis scripts, but not into any of the default Python modules. Furthermore, if the workers are responsible for managing extra resources, there is no way to coordinate when they will attempt to pull these resources. This can potentially lead to network congestion issues if multiple workers try to retrieve large resource files at the same time. This work contributes a second type of task: an *analysis task*. Analysis tasks can specify additional resources that they require. When the task is deployed to a worker, the management node then deploys these additional resources to the worker node. The management handles resource deployment in a serial fashion, preventing network congestion problems. After the task has finished, the resources are cleaned up.

3.5 Extensions

While the simulation model above is based off of the model developed by Agarwal in [69], it has been heavily refactored through the course of this work. While many of the changes were simple splitting modules to separate responsibilities or clarify logic, there were also several significant architectural changes. This section will discuss these contributions.

3.5.1 Realistic Network Topology

In order to generate realistic network topologies, a tool was written using the Python programming language[89] to interface with various AS-level topology generators and convert their output into a format that could be used by the simulation model. When writing this tool, there were three main goals. The first goal was to generate a

realistic AS-level topology, including network characteristics such as link bandwidths, delays, error rates, *etc*. The second goal was to make it as simple as possible to incorporate multiple topology generators. The third goal was to automate as much of the simulation setup as possible. The resulting architecture can be seen in Figure 3.5.

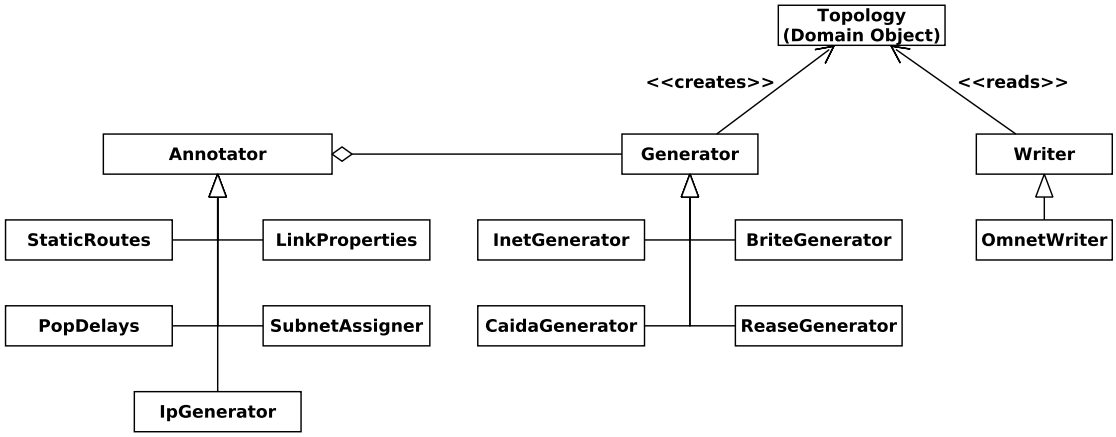


Figure 3.5: Topology Generator Architecture

The general flow of processing is that a `Generator` interacts with an underlying topology source (*i.e.*, an external program or file) to generate a topology structure. The generator object translates the output of its underlying source into *topology domain objects* (*i.e.*, classes representing the topology, autonomous systems, subnets and links). Since not all underlying topology sources generate the same level of detail, `Annotators` are then used to fill in missing details such as link characteristics. `Annotators` are also used to generate some simulation-specific details such as IP addresses for all autonomous systems and subnets, and static routes between nodes in the network topology. Finally, the annotated topology is passed to a `Writer` which outputs the topology in a specific format (*e.g.*, files that can be consumed by a simulation).

The following sections will outline the mentioned components and the design decisions behind them.

3.5.1.1 Generators

`Generator` components are responsible for generating the network topology's structure. As was discussed in Section 3.1.1, one of the primary goals for the simulation

model was to support the use of both measured and generated network topologies. However, measured and generated topologies require different work-flows: measured topologies are typically stored in a set of files which must be read and parsed while generated topologies require running an external topology generation program, capturing the output and then parsing it. Thus, the developed `generator` components encapsulate the specific work-flow for each underlying topology source. In total, four `generator`s were developed: the `CaidaGenerator`, the `InetGenerator`, the `BriteGenerator` and the `ReaseGenerator`.

The `CaidaGenerator` is the only developed `generator` for measured topologies, providing support for measurements from the CAIDA Internet Mapping and Annotation project[77]. This project provides a catalogue of Skitter-formatted[90] measurements of the IPv4 routed /24 AS links of the Internet from January 2000 until the present. A single measurement file may not observe the whole Internet routing topology, so the `CaidaGenerator` supports aggregating multiple measurement files into a single topology.

The `InetGenerator` is the simplest `generator` for generated topologies, using the Inet-3.0 topology generator[91]. It is worth noting that while the Inet-3.0 topology generator is named very similarly to the OMNeT++ INET framework, the two projects are unrelated. Inet-3.0 is simple to run since all configuration options can be specified as command-line arguments, and the output is also simple to parse. Thus, the `InetGenerator` simply builds and executes the appropriate command and then captures and parses the output. However, Inet-3.0 only generates an AS-level topology structure; it does not generate any other network characteristics. This means that `annotator`s must be used to fill in these details. It is also worth noting that Inet-3.0 does not appear to have been actively developed or supported since 2002[92].

The `BriteGenerator` uses BRITE: the Boston University Internet topology generator [93]. BRITE is able to generate AS-level and router-level topologies, offers rudimentary support for generating link bandwidths and delays and can import pre-generated topologies from a number of different file formats. Among the supported formats, BRITE claimed to support importing Skitter-formatted files, such as those produced by the CAIDA Internet Mapping and Annotation project[77]. Unfortunately, at the time of this work the implementation for this format is broken, and the BRITE code-base is no longer maintained[94], prompting the specific development of the `CaidaGenerator`. BRITE requires that configuration options be specified in a file. Thus, the `BriteGenerator` creates a temporary file to contain this information

which is removed after running the BRITE executable. Also, the output from BRITE is stored in a file. Thus, the `BriteGenerator` reads the contents of this file into a buffer and then removes the generated file. While BRITE supports generating link bandwidth and delay values according to constant, uniform, exponential or heavy-tailed distributions, these values are not not weighted by the degree of connectivity of the nodes being linked. The work of Wei *et al.*[95, 64] suggests this is a necessary step for the values to accurately reflect those of the Internet. Thus, these values are generated by an `annotator` instead of BRITE.

The the last developed `generator` is the `ReaseGenerator` which uses ReaSE[78] as its underlying topology source. ReaSE is the most recent of the considered topology generators and it includes a number of desirable features such as AS-level and router-level topologies, link bandwidth generation and background traffic generation. It is also the only of the three considered topology generators that is still actively supported and developed. The only downside of ReaSE is that it is designed to work directly with either the OMNeT++ INET framework or the OverSim framework. As a result, it generates NED-formatted files which are non-trivial to parse. However, for AS-level topology configurations, parsing the output is still manageable. Similar to BRITE, ReaSE requires configuration options to be specified in a file and generates output in a file. Thus, the `ReaseGenerator` follows the same work-flow as the `BriteGenerator`.

Inet-3.0, BRITE and ReaSE use text-based output formats which can be saved to a file. Thus, their associated `generator` components support an additional work-flow to read previously generated output from a file.

3.5.1.2 Topology Size Reduction

The network topologies generated by the `CaidaGenerator` from recent snapshots of the Internet topology contain approximately 32,000 ASes. However, because of the memory constraints of the hardware used in this work, it is only possible to use network topologies of approximately 2500 ASes. As a result, it is necessary to reduce the size of the topology. In this work, this is achieved by removing the least-connected ASes from the topology, effectively retaining the core nodes of the network topology.

A consequence of this is that a similar procedure must be followed for generated topologies as is illustrated in Figure 3.6. Figure 3.6a shows the node connectivity distribution for a CAIDA topology that has been reduced from 32,000 ASes to 2500

ASes. Figure 3.6b shows the connectivity distribution for a 2500 AS topology generated using ReaSE. Notice how the topology lacks the highly-connected nodes (*i.e.*, nodes with more than 400 connections) that are present in the CAIDA topology. However, if ReaSE is used to generate a 32,000 AS topology which is then reduced to 2500 ASes by removing the least-connected ASes from the topology, as is shown in Figure 3.6c, the node connectivity better matches the node connectivity of the reduced CAIDA topology.

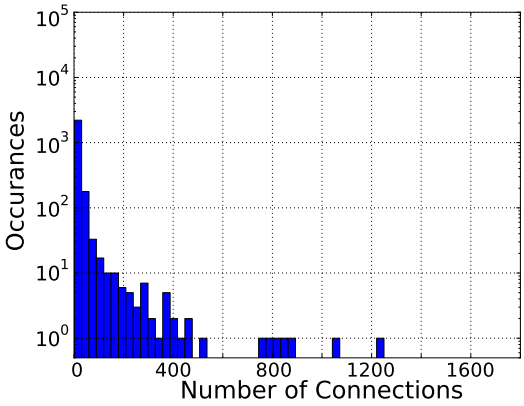
3.5.1.3 Annotators

While ReaSE supports the generation of a number of realistic link characteristics, the other topology generators and measured network topologies do not. Because of this variation in their capabilities, the developed Python tool must to be able to easily enable or disable the generation of certain characteristics. To facilitate this, a set of `Annotators` can be used to "fill-in" missing values. These components are also used to provide information that is present in none of the topologies, such as static routes between nodes in the network. Table 3.4 summarizes all of the `Annotators`.

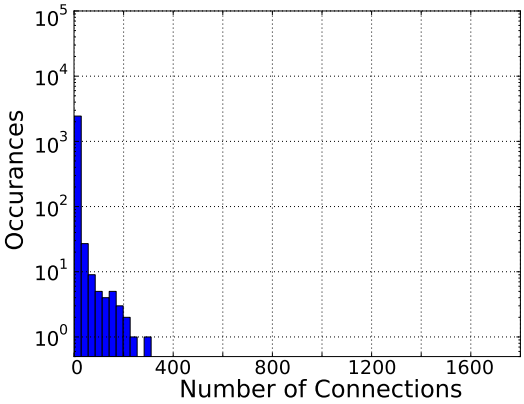
Name	**Functionality**
`IpGenerator`	Generates IPv4 addresses for every AS and ranges of IPv4 addresses for every subnet.
`LinkPropertyGenerator`	Generates bandwidths and routing delay characteristics for every AS-link.
`PopLinkAdjuster`	Adjusts the delay of links to and from large ASes.
`SubnetAssigner`	Connects subnets to the least-connected ASes in the topology.
`StaticRouterGenerator`	Generates static routes for every routable destination (*i.e.*, subnet) using Djikstra's algorithm.

Table 3.4: Generator Components Summary

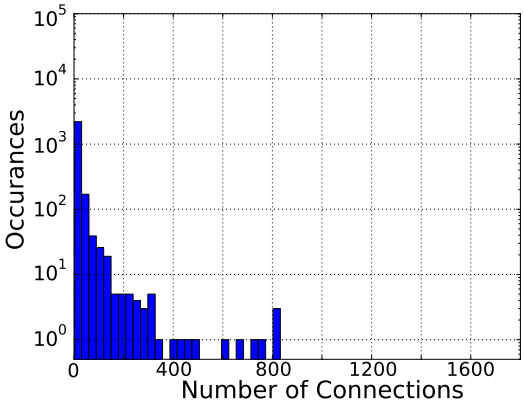
Of special note, the `LinkPropertyGenerator` is of critical importance since it is responsible for generating realistic link characteristics. While there are a number of tools for measuring available link bandwidths, there has been little work done to measure these values across the whole of the Internet. The iPlane project[96] measures the bandwidth between various links by participating in popular BitTorrent swarms. Unfortunately these measurements only provide rough estimates for a portion of the Internet topology. The Pathneck[97] project, which estimated link bandwidths for



(a) CAIDA Topology (32,000 ASes reduced to 2500 ASes)



(b) Generated ReaSE Topology (2500 ASes)



(c) Reduced ReaSE Topology (32,000 ASes reduced to 2500 ASes)

Figure 3.6: Comparison of Node Connectivity between Generated and Reduced-Size Network Topologies

a large number of AS links, previously had a dataset available. While this data is no longer available, Wei *et al.* provide a summary of the values in [64]. Thus, the developed `LinkPropertyGenerator` makes use of these same values.

3.5.1.4 Writers

Once the topology has been fully annotated, it can be output in a useful format by one of two `Writer` components. The first writer component is the `OmnetWriter` which outputs the topology as NED files which can be used directly by the simulation model outline in Section 3.3. The second writer is the `PythonWriter` which uses Python's built-in `pickle` module to serialize the topology domain objects so that the topology can later be reloaded for analysis and visualization.

3.5.2 Saving & Loading Simulation State

Another significant contribution of this work was the addition of a state saving and loading mechanism. This functionality is essential in facilitating rigorous statistical analysis. First, it eliminates redundant work since the generation of an initial network state only needs to be done once for all simulations that use that same initial network state. Second, multiple simulation runs can begin with an identical initial botnet state, even if the tunable parameters of the botnet or random processes of the simulation are altered in a way that would change the growth of the botnet.

The serialization functionality was implemented using Boost Serialization library[98]. This is a general-purpose C++ serialization library, and is one of the most fully-featured serialization libraries available. However, through the course of this work, it became apparent that there is one unfortunate conflict between the Boost serialization library and the OMNeT++ framework. When deserializing objects through pointers deserialization must be invoked on unassigned pointers. Boost allocates the appropriate amount of memory, assigns it to the pointer that deserialization was invoked on and then uses *placement new* to initialize that block of memory. Thus, Boost serialization requires objects to be fully constructable through their constructor. However, the OMNeT++ framework makes extensive use of the *factory pattern*[99] to determine the correct object type and to initialize and maintain framework-specific state that is beyond the control of the caller (*i.e.*, assigning object tracking IDs, managing string pools, allocating type-specific parameters, *etc.*). Thus, to correctly deserialize factory-built objects through pointers, the factory must first be used to create the ob-

ject and initialize any uncontrollable state, and then deserialization must be invoked on a pointer to this partially initialized object to restore any controllable state. This requires overriding a portion of the serialization library code through *template specialization*. Since this code was largely identical for each instance, it was formalized as a macro which can be seen in Appendix A.1. This approach also requires that the deserialization process is managed by an external object or function rather than directly invoking Boost serialization.

In the simulation model, there is a global `BotnetArchiver` module that is responsible for managing the process of saving and loading of state for the whole simulation model. There is also a `TopologyArchiver` which manages the saving and loading of the state for the network topology (*i.e.*, ASes and links). The procedure for loading state is described in Algorithm 1. The procedure for saving is identical, but is simpler as it does not require the construction of objects.

Algorithm 1: Procedure for Loading Simulation State

```
Load global simulation parameters
Load the number of ASes in the topology
foreach AS do
    Construct the AS module
    Load any controllable parameters (e.g., static routes)
    foreach link to another AS do
        Create link
        Load parameters
    end
    if there is a link between the AS and a subnet then
        Construct the topology-portion of the link
        Load link parameters
    end
end
Construct and load the OverlayController and its sub-modules
Load the number of subnets
foreach subnet do
    Construct and load the Subnet module and its submodules
    Complete the link to the subnet's associated AS
    Construct each host in the subnet and load its state
end
```

3.5.3 Churn Model Modifications

The churn model outlined in Section 3.3.2.2 is a significant change from the original churn model used by Agarwal in [69].

First, the original churn model was distributed while the new model is centralized. Originally each `SubnetController` was responsible for both triggering churn events and managing the execution of those events. This seemed advantageous since each subnet could have a separately-controlled churn process; however, none of the experiments conducted using this churn model ever leveraged this ability. Thus, the distributed churn process effectively increased the complexity of the `SubnetController` module without offering any utilized advantage. Also, the global churn rate was proportional to both the churn rate of each subnet and the number of subnets. To maintain a constant global churn rate if the number of subnetworks changed, the churn rate for each subnet had to be manually adjusted in the configuration file. By centralizing the churn management, the process is simpler to understand and does not vary depending on the number of subnets. Furthermore, the centralized churn process can still be weighted to created different churn amounts of churn in each subnet.

Second, the original churn model only had a single churn process. Thus, by introducing sybils into the simulation, the churn and placement of normal bots was altered. The new churn model features two separate churn processes: i) normal churn and ii) targeted churn. As was discussed in Section 3.3.2.2, the introduction of the separate targeted churn process allows sybils to be introduced into the network without affecting the placement of normal bots. Also, since both the churn and the parameter generation processes for targeted churn use different PRNGs, the physical placement of sybils can be varied without altering their logical placement. This means that the effects of adding sybils and varying their physical placement can be studied in isolation. This is essential for the purposes of the conducted research in this thesis.

3.5.4 Low Memory Routing Tables

One issue that arose when running simulations was that the memory consumption continuously increased for the entire duration of the simulation. This was traced back to the routing table model used by the OMNeT++ INET framework. In this model, routes are stored in an arbitrary order. Thus, the only means of finding the best matching route, given a destination IP address, is to perform a linear search through

the whole list. In order to improve the run-time performance, every time a routing table successfully locates a route, a pointer to the route is cached in a C++ `map`, indexable by the destination IP address and, future lookups can search through the cached entries with logarithmic complexity.

The memory consumption problem arose because this cache is never cleared. In a P2P network where each peer contacts numerous other peers in other subnets, this means that the same route is cached numerous times, once for each destination in that subnet. This then occurs at each routing node in the network. As a result, with a large number of routable destinations (*i.e.*, subnets) and a high level of bot churn, these cached routes can consume several gigabytes of RAM over the course of a simulation.

During this work, a low-memory version of the routing table model was implemented. This optimization was possible because the simulation model uses a static network topology and thus all routes are static. As a result, it is possible to sort all the static routes at each routing node according to the network prefix (*i.e.*, the logical OR of the IP address and subnet mask) at the start of the simulation. Once sorted, a binary search can be used to look up routes with logarithmic complexity. There is one small caveat with this approach: if the search fails to locate a route, the first entry in the routing table must be checked. If it is the default gateway route (*i.e.*, the network prefix is 0.0.0.0), then it is considered to match the destination IP address.

Since this yields the same performance as the route cache in the default routing table model, the low-memory routing tables do not need to cache retrieved routes and do not incur the same memory penalties. Furthermore, the default routing tables eventually end up with more routes in the cache than in the routing table (since each route is cached for each destination). Thus, the low-memory routing tables also reduced the time needed to run simulations by approximately 6%.

3.6 Chapter Summary

This chapter discussed the packet-level simulation model that is used in this research. While multiple simulation tools exist, OMNeT++ was chosen because it provides a rich set of verified protocol and device implementations within a flexible, scalable environment. While this simulation model is based off of the work of Agarwal[69], a number of extensions and revisions were contributed throughout the course of this work in order to support i) realistic underlay networks, ii) saving and loading simu-

lation state iii) improved scalability for large networks and iv) a churn model which is able to support the sybil placement strategies explored in this thesis.

Because of the large numbers of simulations required by this research, the simulation model is integrated with the STARS automation framework. This framework allows for parallel execution of simulations across the cluster of computers available for this research, greatly reducing the time needed to execute batches of simulations.

Chapter 4

Experiments

This chapter will lay out the parameter settings of two sets of experiments conducted during this research with summaries and discussions of the measurements collected during those experiments.

4.1 Common Parameter Settings

The hardware available for this research is a cluster of 41 Blade servers. Each of these machines has two 3GHz Intel Xeon processors, 8GB of RAM and 60GB of storage. Since each Blade server has two processors, this work limited simulation memory usage to approximately 4GB, enabling each Blade server to run two simulations simultaneously.

Because the interests of this research are similar to those of Davis *et al*. in [27], this work uses the same size of botnet: 20,000 bots. This leaves two other settings which impact the overall memory usage of each simulation: i) the size of the underlay network and ii) the number of sybils used during sybil attacks. Empirical testing was used to determine that a network topology of 2500 ASes with 2000 subnets would allow for approximately 16,000 sybils. This allows sybils to make up as much as 50% of the total botnet size.

4.1.1 Network Topologies

This work considers two underlay network topologies: i) a measured network topology from the December 2012 measurements of the CAIDA Internet Mapping and Annotation project[77] and ii) a network topology generated using the ReaSE topol-

ogy generator. Each parameter setting is run on both of these networks. Only one botnet state is generated for each underlying network topology. Although it would be highly desirable to generate and run experiments with multiple botnet states, the time and storage requirements for the necessary number of simulation runs, which will be discussed in more detail in Section 4.2.5, is not feasible.

Attribute	**CAIDA**	**ReaSE**
Measurement Set	December 2012	N/A
`numberOfAses`	2500	2500
`numberOfSubnets`	2000	2000

Table 4.1: Summary of Network Topology Configurations

4.1.2 General Configuration and Settings

In general, each simulation run simulates 24 hours of botnet activity. Also, 10 repetitions are run for each scenario (*i.e.*, placement strategy and particular configuration of parameter settings). While this number of repetitions may not be sufficient for thorough statistical analysis of each scenario, it should provide a general sense of how the botnet reacts to each sybil placement strategy. For all simulations, the PRNGs seed sets are selected based on the repetition number. This way the random processes vary between repetitions but should be identical (where possible) across scenarios. However, some random processes (*e.g.*, intra-subnet packet delays) cannot be repeated identically across all scenarios since adding sybils to the botnet inherently affects packet flows to and from each subnet.

Across all experiments, all of the botnet protocol parameters outlined in Table 2.1 are not be varied and are set as detailed in Table 4.2. Of special note, `tValueLookup` is set to 15 minutes instead of the default of 1 hour. This means that bots attempt 4 value lookups per hour instead of 1 so that if a bot is alive for 24 hours, it is possible for it to retrieve all 32 {*key*,*value*}-pairs published into the botnet.

4.2 Experiment Set 1: High Churn

This section outlines the scenario-specific settings for the first set of experiments conducted during this research. Note that this section also outlines the parameter

Parameter	Setting
α	3
B	128 bits
k	20
`numKeyValuePairs`	32
`numSeedLocations`	10
`bootstrapSize`	200
`tRepublish`	24 hours
`tRefresh`	1 hour
`tValueLookup`	15 minutes
`tReplicate`	1 hour

Table 4.2: Summary of Constant Botnet Protocol Settings

settings used when generating the initial botnet states that are used in all other simulation runs. These initial botnet states are reused by the second experiment set.

In total, this experiment set covers 6 types of simulations: one type for generating the initial botnet state, one type for running the botnet without sybils, and four types for running the botnet with each of the placement strategies outlined in earlier chapters. The following sections will outline the settings used for each type of scenario. A summary of the parameter settings for all scenarios is presented in Table 4.3.

4.2.1 Generation

Only two generation simulations are needed: one for each network topology. As outlined in Section 4.1.1, each network topology contains 2500 ASes and 2000 subnets. Also, in both cases, the desired botnet size is 20,000 bots. Initially, half of the bots are evenly distributed between all subnets (*i.e.*, `initialBots` = 5). For each subnet, `maxBots` is set to 2000 bots. This is mostly due to the fact that this number closely relates to the IP address range that is used during all subsequent simulation runs that use the saved botnet state, and those runs need to place numerous sybils into a particular subnet. Finally, during these simulations normal churn is set so that an average of 400 bots (*i.e.*, 2% of the final botnet size) are added per minute and 1 bot is removed per minute (*i.e.*, 0.005% of the final botnet size). The period between these churn events is distributed according to an exponential distribution.

	Generation	No Sybils	Unrestricted	Uninformed	Partially Informed	Fully Informed
Botnet Size	20,000	20,000	20,000	20,000	20,000	20,000
Churn (% of botnet size per min)	+2%, −0.005%	±2%	±2%	±2%	±2%	±2%
# of Sybils	N/A	N/A	4k, 8k, 12k, 16k	4k, 8k, 12k, 16k	4k, 8k, 12k, 16k	4k, 8k, 12k, 16k
# of Targeted Subnets	N/A	N/A	2000	250, 500, 1000	250, 500, 1000	250, 500, 1000
Duration (Sim Time)	N/A	24h	24h	24h	24h	24h
Network Topologies	Both	Both	Both	Both	Both	Both
Repetitions	1	10	10	10	10	10
Total Simulations	2	20	80	240	240	240

Table 4.3: Summary of Parameter Settings for All Experiment Configurations

4.2.2 No Sybils

The "No Sybil" simulations are requisite for the fully-informed sybil placement strategy. During these simulations there are no sybils and thus no targeted churn. Normal churn is set so that, per minute, an average of 2% of the botnet (400 bots) are added and, on average, an equal number are removed. There is only one scenario for each network, resulting in a total of 20 simulations.

4.2.3 Unrestricted Sybil Placement

The unrestricted sybil placement simulations use the same churn processes outline in Section 4.2.2. 10 repetitions are run for each saved botnet state for 4 different numbers of sybils: 4000, 8000, 12,000 and 16,000. Between both initial botnet states, a total of 80 simulation runs are required.

Attribute	Value
`numberOfAses`	2500
`numberOfSubnets`	2000
`initialBots`	5
`maxBots`	2000
`tBotCreation`	exponential(0.15) seconds
`tBotRemoval`	exponential(1) second

Table 4.4: Summary of Generation-Specific Parameter Settings

Attribute	Value
`tBotCreation`	exponential(0.15) seconds
`tBotRemoval`	exponential(0.15) seconds

Table 4.5: Normal Churn Process Settings

In each case, the sybils are added by the targeted churn process according to an exponential distribution of inter-arrival times so that, on average, all sybils are added over a 2 hour period:

$$\texttt{tTargetedBotCreation} = \text{exponential}\left(\frac{2h}{\texttt{numberOfSybils}}\right)$$

Also, no sybils are removed (*i.e.*, targeted bot removal isdisabled). This is achieved by setting `tBotRemoval` to 48 hours since this schedules the first removal for after the end of the simulation. Once all sybils have been added into the botnet, targeted churn is altogether disabled for the remainder of the simulation.

Attribute	4000 Sybils	8000 Sybils	12000 Sybils	16000 Sybils
`tTargetedBotCreation`	exp(1.8s)	exp(0.9s)	exp(0.6s)	exp(0.45s)
`tTargetedBotRemoval`	48h	48h	48h	48h

Table 4.6: Targeted Churn Settings for Unrestricted Sybil Placement Attacks

4.2.4 Restricted Sybil Placement

The uninformed, partially-informed and fully-informed sybil placement strategies only differ in how they select which subnets to target. Otherwise, the simulations for these strategies use identical parameter settings.

These simulations use the same settings for normal and targeted churn as the unrestricted sybil placement simulations; however, these placement strategies are also dependent upon the number of subnets that can targeted for sybil placement. Thus, for each of the 4 numbers of sybils inserted into the botnet, simulations are run for 3 values of `numberOfTargets`: 250, 500 and 1000. These values represent 12.5%, 25% and 50% of the targetable subnets, respectively. In total, there are 12 scenarios for each of the restricted sybil placement strategies. At 10 repetitions of each scenario for each initial botnet state, this results in 240 simulations for each of the 3 restricted sybil placement strategies.

Attribute	4000 Sybils	8000 Sybils	12000 Sybils	16000 Sybils
`tTargetedBotCreation`	exp(1.8s)	exp(0.9s)	exp(0.6s)	exp(0.45s)
`tTargetedBotRemoval`	48h	48h	48h	48h
`numberOfTargets`	250, 500, 1k	250, 500, 1k	250, 500, 1k	250, 500, 1k

Table 4.7: Targeted Churn Settings during Restricted Sybil Placement Attacks

4.2.5 Time and Storage Requirements

Tables 4.8 and 4.9 summarize the CPU time and storage requirements for all of the experiments outlined above. Note that the CPU time values exclude the CPU time required for data analysis whereas the storage values include analysis results. The time needed for each simulation run varies proportionally to both the number of sybils and the number of targeted subnets. Storage varies in a similar fashion, but the variance due to the number of targeted simulations is much lower than the variation due to the number of sybils. In total, these simulations require 3.6TB of storage and would have required over 10 years to complete if run serially on a single computer. By distributing the workload in the available computing cluster, these simulations were completed over a period of approximately 4 months.

4.2.6 Evaluation

Because of the high number of simulations conducted during this research, this thesis makes a number of assumptions that are used during evaluation in order to summarize the resulting measurements.

	Targeted Subnets	No Sybils	4000 Sybils	8000 Sybils	12,000 Sybils	16,000 Sybils	Total Time (All Runs)
No Sybils	0	3d 5h 9.77GB	—	—	—	—	32d 2h 97.68GB
Unrestricted	2000	—	3d 23h 3.69GB	5d 4h 4.14GB	6d 14h 4.66GB	7d 19h 5.23GB	235d 0h 177.2GB
Uninformed	250	—	3d 17h 3.62GB	4d 14h 4.02GB	5d 14h 4.50GB	6d 15h 5.02GB	205d 0h 171.56GB
	500	—	3d 18h 3.63GB	4d 18h 4.05GB	5d 22h 4.56GB	7d 4h 5.10GB	215d 20h 173.43GB
	1000	—	3d 20h 3.64GB	4d 22h 4.09GB	6d 6h 4.61GB	7d 12h 5.18GB	225d 0h 175.14GB
Partially Informed	250	—	3d 18h 3.64GB	4d 15h 4.04GB	5d 19h 4.54GB	6d 22h 5.06GB	210d 20h 172.84GB
	500	—	3d 17h 3.64GB	4d 18h 4.07GB	5d 23h 4.57GB	7d 3h 5.12GB	215d 10h 174.03GB
	1000	—	3d 22h 3.67GB	5d 2h 4.12GB	6d 10h 4.65GB	7d 22h 5.22GB	233d 8h 176.50GB
Fully Informed	250	—	3d 16h 3.66GB	4d 14h 4.05GB	5d 15h 4.52GB	6d 16h 5.04GB	205d 10h 172.68GB
	500	—	3d 18h 3.67GB	4d 19h 4.08GB	5d 22h 4.57GB	7d 3h 5.11GB	215d 20h 174.25GB
	1000	—	3d 20h 3.69GB	5d 20h 4.16GB	6d 7h 4.64GB	7d 16h 5.19GB	236d 6h 176.82GB
						Total	6y 7d 22h 1.80TB

Table 4.8: Summary of CPU Time and Storage Requirements for High Churn CAIDA Network Simulations.

	Targeted Subnets	No Sybils	4000 Sybils	8000 Sybils	12,000 Sybils	16,000 Sybils	Total Time (All Runs)
No Sybils	0	2d 12h 9.78GB	—	—	—	—	25d 97.79GB
Unrestricted	2000	—	3d 0h 3.69GB	3d 23h 4.15GB	5d 0h 4.67GB	6d 5h 5.24GB	181d 16h 177.50GB
Uninformed	250	—	2d 18h 3.63GB	3d 11h 4.02GB	4d 6h 4.50GB	5d 2h 5.02GB	155d 10h 171.78GB
	500	—	2d 20h 3.64GB	3d 15h 4.06GB	4d 13h 4.56GB	5d 11h 5.11GB	164d 14h 173.71GB
	1000	—	2d 22h 3.66GB	3d 18h 4.10GB	4d 15h 4.61GB	5d 16h 5.18GB	169d 14h 175.47GB
Partially Informed	250	—	2d 21h 3.66GB	3d 16h 4.06GB	4d 12h 4.55GB	5d 11h 5.09GB	165d 173.68GB
	500	—	2d 22h 3.67GB	3d 20h 4.10GB	4d 19h 4.62GB	5d 21h 5.18GB	174d 4h 175.64GB
	1000	—	3d 0h 3.68GB	3d 22h 4.14GB	5d 1h 4.67GB	6d 11h 5.25GB	184d 4h 177.40GB
Fully Informed	250	—	2d 18h 3.62GB	3d 12h 4.02GB	4d 6h 4.50GB	5d 3h 5.02GB	156d 6h 171.59GB
	500	—	2d 20h 3.64GB	3d 15h 4.06GB	4d 13h 4.56GB	5d 13h 5.10GB	165d 10h 173.63GB
	1000	—	2d 23h 3.66GB	3d 19h 4.10GB	4d 20h 4.63GB	6d 2h 5.19GB	176d 16h 175.77GB
						Total	4y 51d 6h 1.8TB

Table 4.9: Summary of CPU Time and Storage Requirements for High Churn ReaSE Network Simulations.

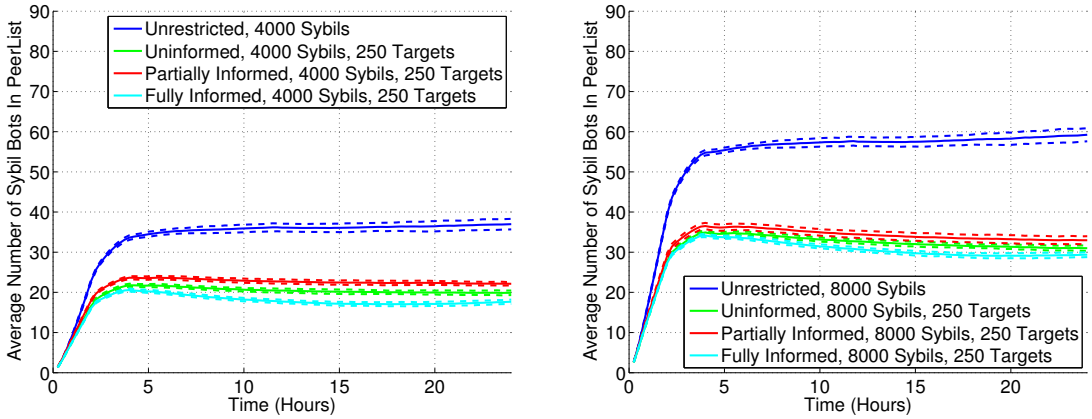
Primarily, this work assumes that the performance of the botnet is ergodic across all repetitions of a particular configuration of tunable parameters. This assumption is rooted in the fact that all repetitions of a particular scenario start from the same botnet state. This assumption is leveraged in order to summarize the measurements from an ensemble of repetitions. These summaries are generally be presented as an ensemble average with an envelope of ± 1 standard deviation. These statistics are not intended to infer that the underlying distributions are Gaussian; they are provided to give a general summary of how the measured behaviours of the botnet vary under particular configurations of the tunable parameters of the botnet.

Regarding value-related measurements, this work also assumes that value lookups for each {*key*,*value*}-pair are independent, meaning that value-related measurements can be aggregated across the ensemble of all {*key*,*value*}-pairs and the ensemble of all repetitions of a particular scenario. Value lookups for different {*key*,*value*}-pairs only interact in so much as they alter the peer-list of the initiating bot and each bot contacted during the lookup procedure. However, there are similar side-effects even when multiple bots perform lookup requests for the same {*key*,*value*}-pair as well as when bots perform any other regular activities such as refreshing buckets and replicating retrieved values to the k closest bots in the botnet. In total, the additional lookups generated by searching for multiple values only generate a small portion of the overall peer-list maintenance within the botnet. Thus, this work makes the assumption that lookups for each {*key*,*value*}-pair are independent of one another.

4.2.6.1 Peer List Infection

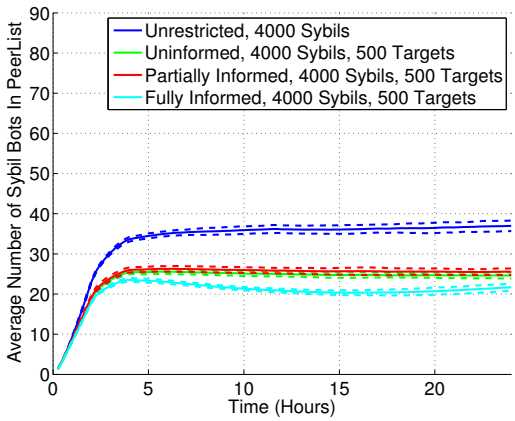
Figures 4.1 through 4.4 present a summary of the mean peer-list infection levels for each experiment scenario (excluding scenarios without sybils). These measurements represent the mean number of entries in each bots' peer-list that point to a sybil instead of a legitimate bot. Note that the measurements for the unrestricted sybil placement strategy are repeated across all plots for a particular network topology and number of sybils since this strategy is independent of the number of targeted subnets.

Figure 4.5 summarizes the mean end-of-simulation peer-list infection levels for each restricted sybil placement strategy. The values are presented as a percentage of the unrestricted sybil placement strategy since it is the highest-performing placement strategy in each case. When the restricted placement strategies target all 2000 targetable subnets they become equivalent to the unrestricted placement strategy, so

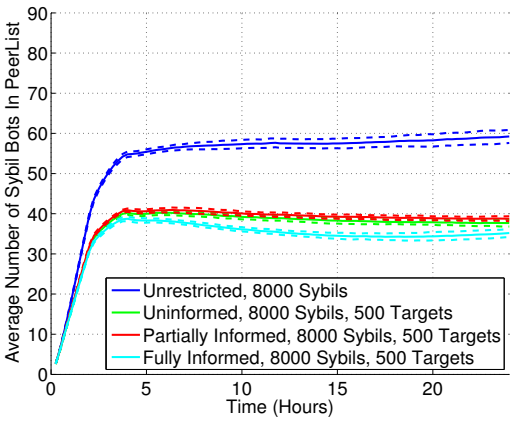


(a) 4000 Sybils, 250 Targets

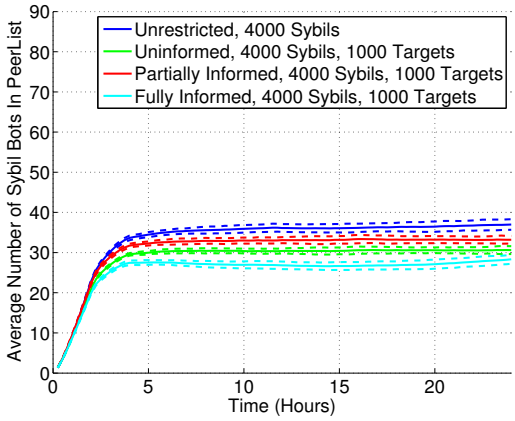
(b) 8000 Sybils, 250 Targets



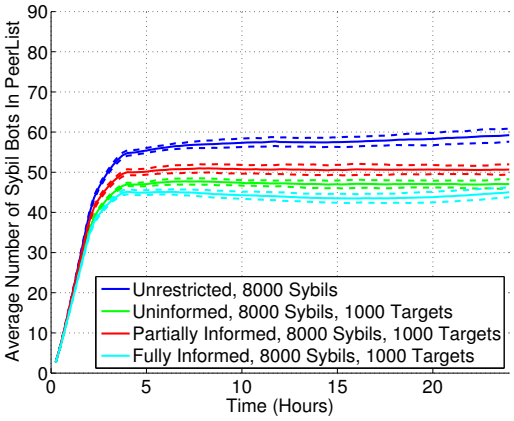
(c) 4000 Sybils, 500 Targets



(d) 8000 Sybils, 500 Targets

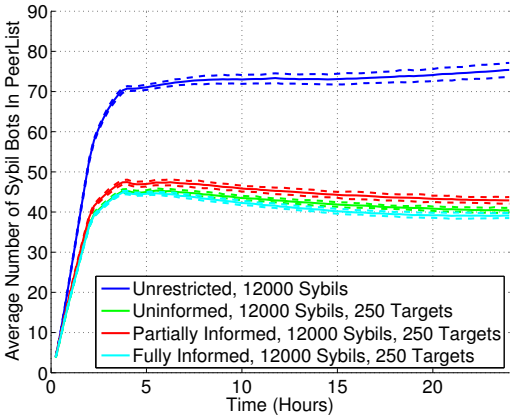


(e) 4000 Sybils, 1000 Targets

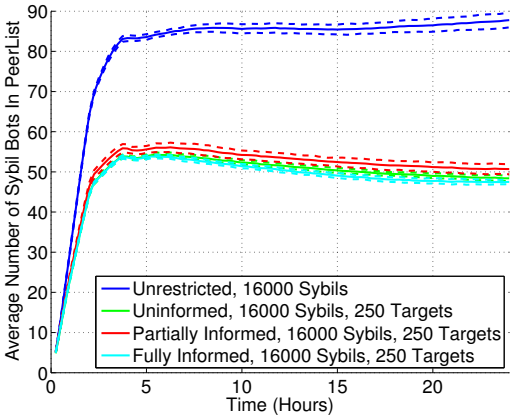


(f) 8000 Sybils, 1000 Targets

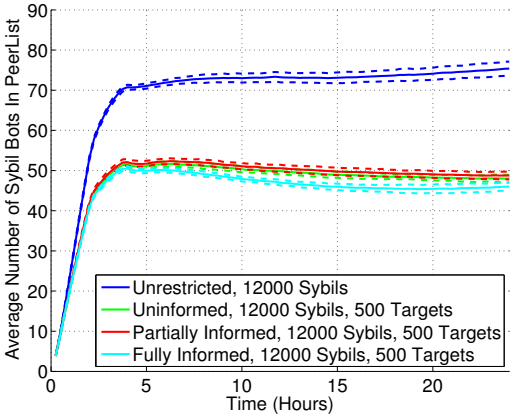
Figure 4.1: Mean Peer List Infection Levels Across CAIDA Network Simulations with 4000 and 8000 Sybils



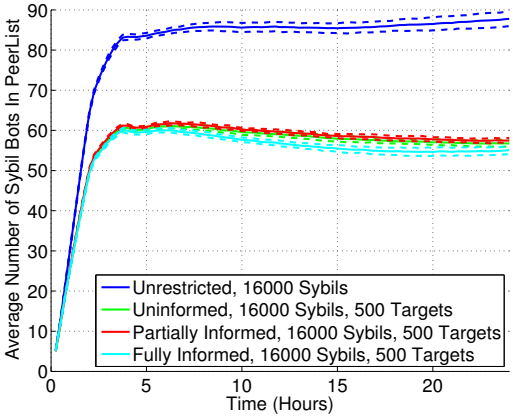
(a) 12000 Sybils, 250 Targets



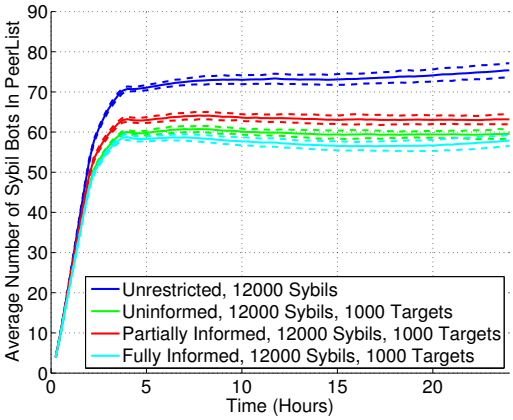
(b) 16000 Sybils, 250 Targets



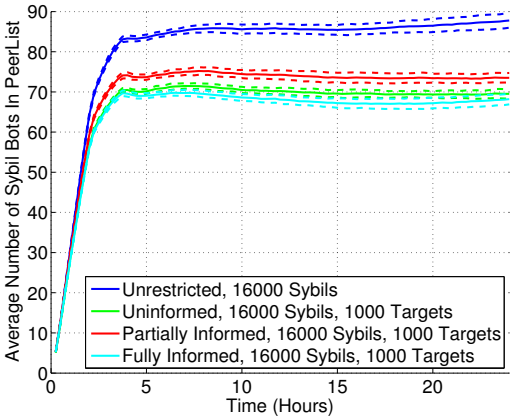
(c) 12000 Sybils, 500 Targets



(d) 16000 Sybils, 500 Targets

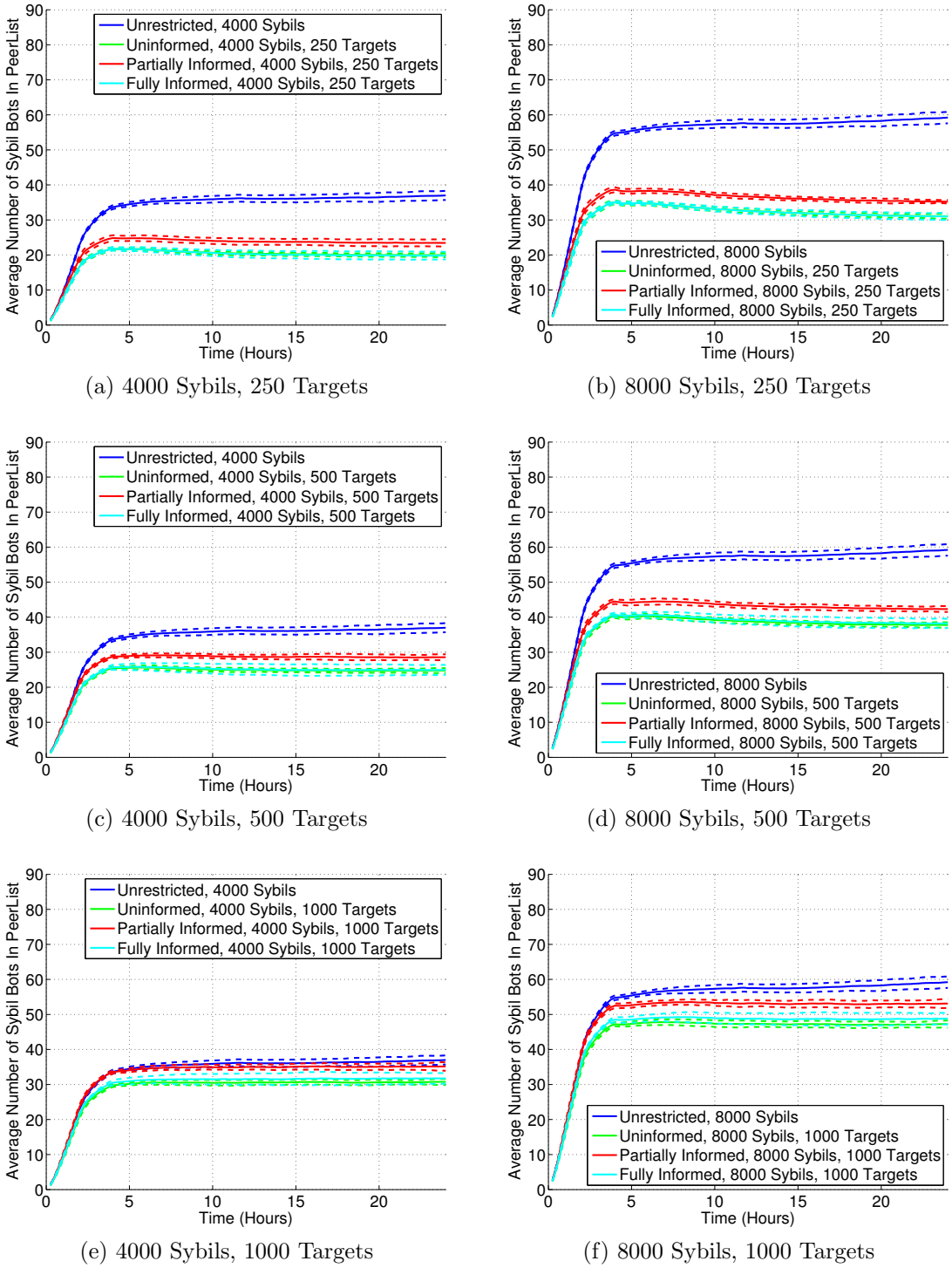


(e) 12000 Sybils, 1000 Targets



(f) 16000 Sybils, 1000 Targets

Figure 4.2: Mean Peer List Infection Levels Across CAIDA Network Simulations with 12000 and 16000 Sybils



(a) 4000 Sybils, 250 Targets

(b) 8000 Sybils, 250 Targets

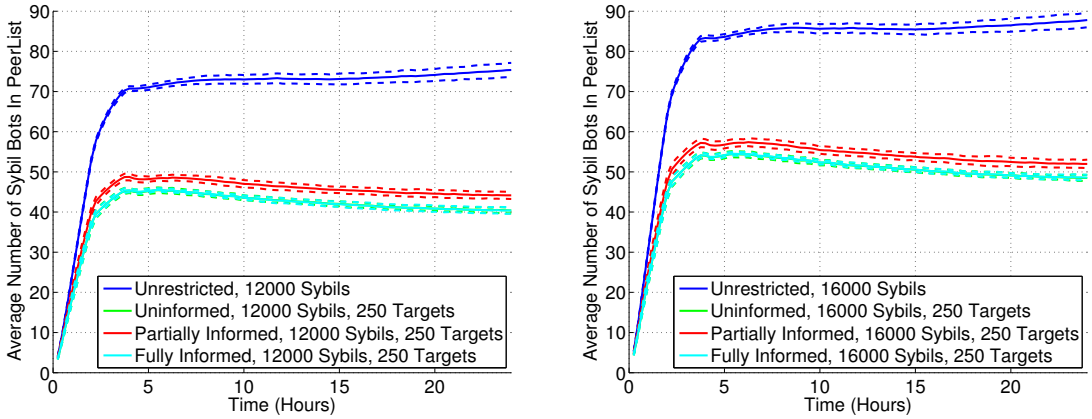
(c) 4000 Sybils, 500 Targets

(d) 8000 Sybils, 500 Targets

(e) 4000 Sybils, 1000 Targets

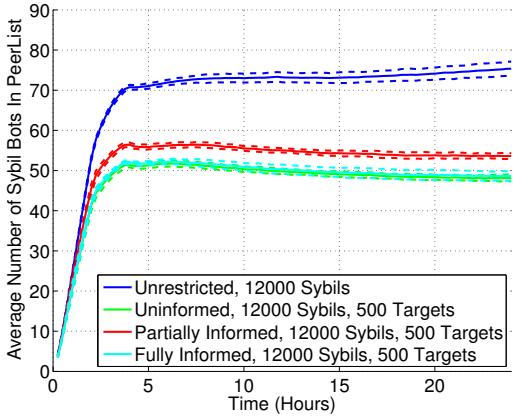
(f) 8000 Sybils, 1000 Targets

Figure 4.3: Mean Peer List Infection Levels Across ReaSE Network Simulations with 4000 and 8000 Sybils

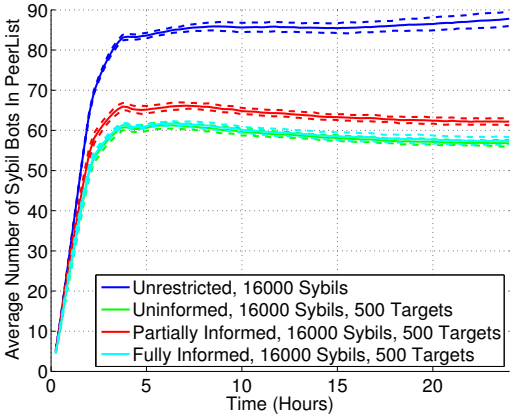


(a) 12000 Sybils, 250 Targets

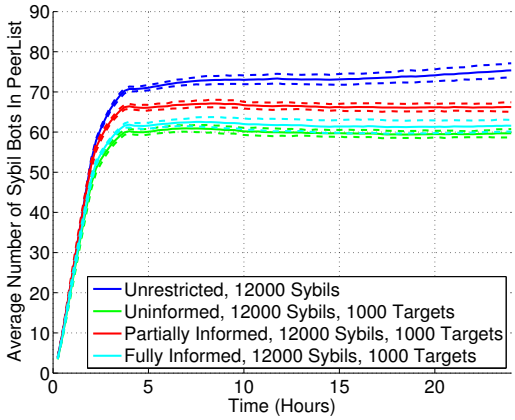
(b) 16000 Sybils, 250 Targets



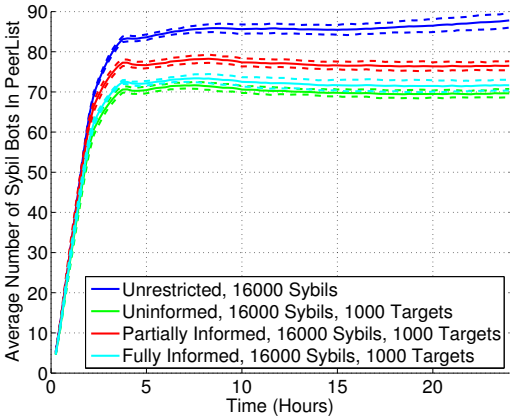
(c) 12000 Sybils, 500 Targets



(d) 16000 Sybils, 500 Targets

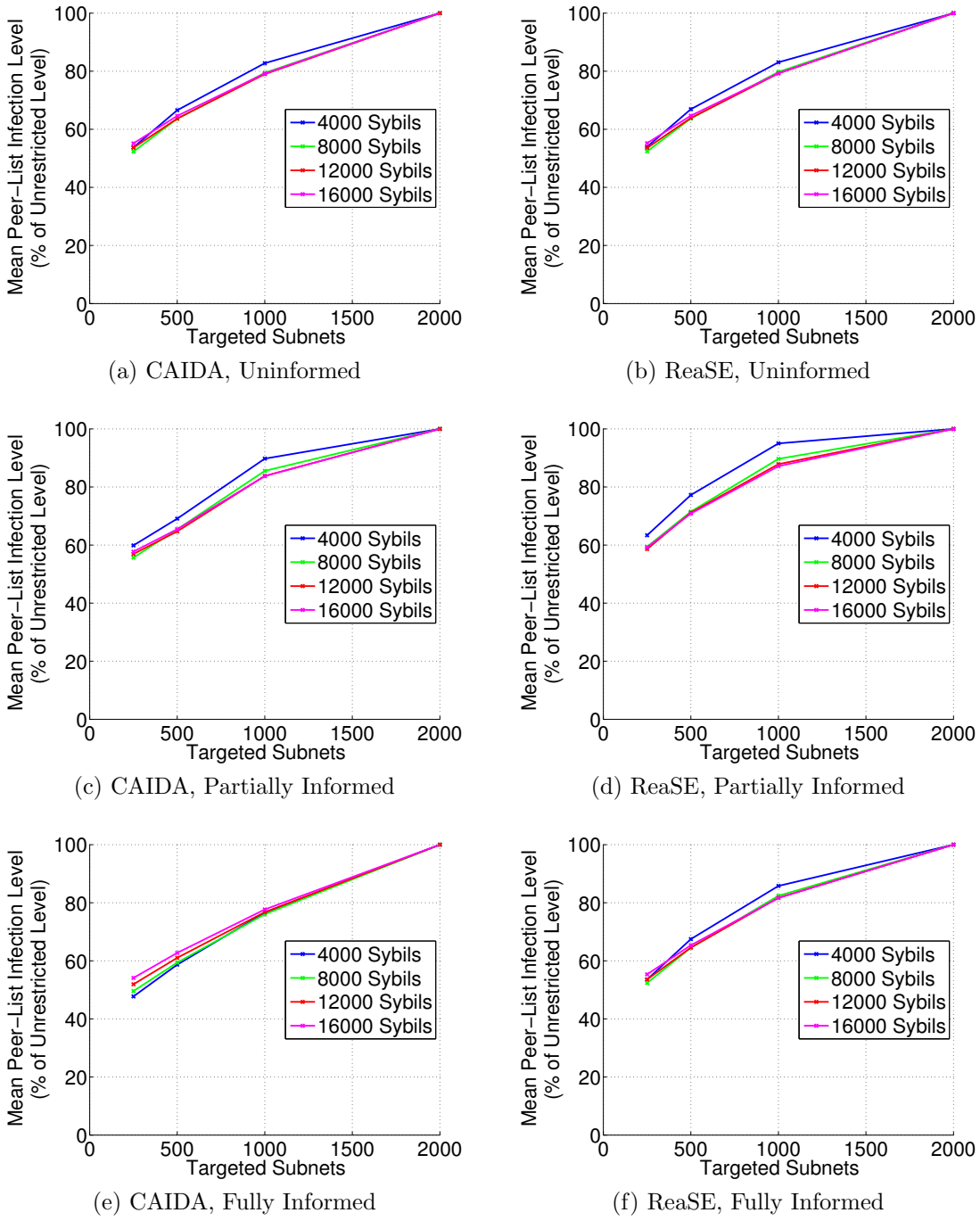


(e) 12000 Sybils, 1000 Targets



(f) 16000 Sybils, 1000 Targets

Figure 4.4: Mean Peer List Infection Levels Across ReaSE Network Simulations with 12000 and 16000 Sybils



(a) CAIDA, Uninformed

(b) ReaSE, Uninformed

(c) CAIDA, Partially Informed

(d) ReaSE, Partially Informed

(e) CAIDA, Fully Informed

(f) ReaSE, Fully Informed

Figure 4.5: Mean end-of-simulation peer-list infection levels for each restricted sybil placement strategy as percentage of levels for unrestricted placement strategy. The last point in each plot (at 2000 Targeted Subnets) is 100% since at this point the restricted placement strategies become equivalent to the unrestricted placement strategy.

each plotted series is shown to converge to 100% at 2000 targeted subnets. These figures omit the ±1 standard deviation envelope for clarity.

In every scenario, the unrestricted sybil placement strategy has the highest peer-list infection level. Of the restricted sybil placement strategies, the partially-informed strategy always has the highest peer-list infection level. Interestingly, in all of the CAIDA network scenarios, the uninformed placement strategy's' peer-list infection level is equal to or greater than the peer-list infection level of the fully-informed strategy while the opposite is true with the ReaSE network scenarios; however, the difference between these two strategies is at most 8% of the unrestricted infection level and between all 3 restricted placement strategies the difference is at most 13.2%.

From this, it seems reasonable to tentatively draw two conclusions. First, if the uninformed and fully informed attacks represent bounds for the range of placement strategies informed by traffic volumes, both of these values represent lower-bounds for the effectiveness of the sybil attack and the optimum information strategy lays somewhere in between these two attacks, as illustrated in Figure 4.6. This is likely a result of how each strategy places sybils. The uninformed placement strategy will place sybils into leaf ASes with higher probability, making sybils effective against bots in those ASes but generally less effective against the majority of bots in the botnet. In contrast, by informing sybil placement based on traffic volumes, sybils will tend to be placed in ASes more central to the network topology. While this may make sybils able to reach a larger number of bots with a moderate packet latency, because of the diffuse, probabilistic nature of Kademlia-style graphs there remains a reasonable probability that some legitimate bot will be closer. Second, given that the range of the sybil attacks' effectiveness, in terms of peer-list infection levels, is up to 13.2%, it does not seem that the gains achieved by informing the sybil placement strategy justify the additional cost and effort necessary to implement them in the real world. However, both of these conclusions are based on peer-list infection measurements which is only a proxy variable for the actual effectiveness of the sybil attack; the value retrieval analysis in Section 4.2.6.2 is a more direct measurement of the effectiveness of the sybil attack and must also be considered.

From these figures, it also appears that the influence gained by the sybil attack is affected by the location of sybils within the physical network topology. This is likely due to the fact that, by pooling sybils into a small number of subnets, when sybils attempt to respond to requests, there will usually be a legitimate bot in a position where it can respond before the sybils. While the sybils may be placed so that they

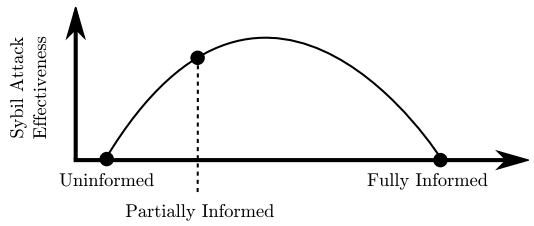


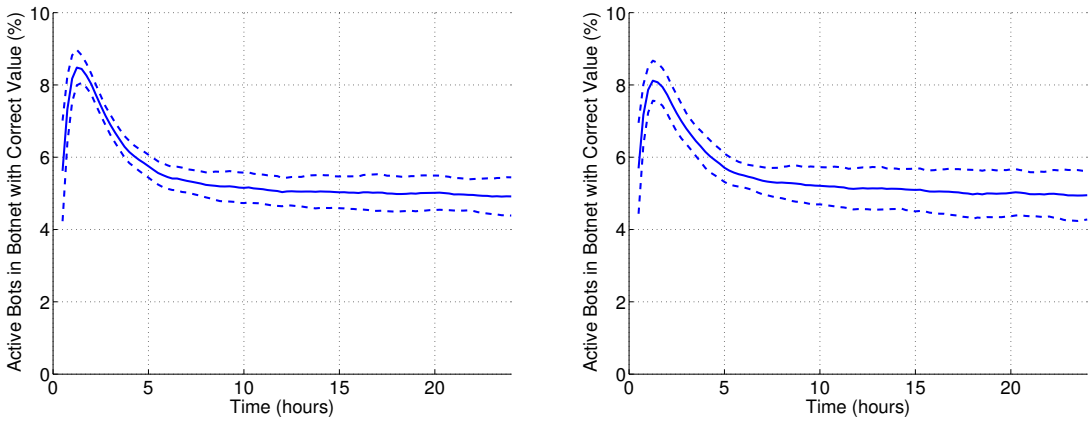
Figure 4.6: Suspected Effectiveness Spectrum for Informed Sybil Placement Strategies

are relatively close (in terms of the underlay network topology) to a large number of bots, bots will still likely have legitimate peers which are still closer than these sybils which allows these peers to respond to requests before sybils.

4.2.6.2 Value Retrieval

Figure 4.7 shows the value retrieval summary statistics for all simulation runs without sybils. As was discussed in Section 4.2.6, the summary statistics are averaged across both the ensemble of {*key*,*value*}-pairs and repetitions for each scenario. These measurements show how many of the bots still active in the botnet have successfully retrieved the correct value for the {*key*,*value*}-pair (displayed as a percentage of the full botnet size). Both network topologies demonstrate near-identical behaviour: at the beginning of the simulation, the value retrieval level climbs rapidly; however, after approximately 1 hour, the level peaks at approximately 8% and then decreases before settling-off at approximately 5% of the botnet. The initial peak is likely due to how the initial botnet state is generated: the death process during generation runs much slower than during all experimentation runs, meaning that the botnet starts off highly connected, but the churn process reduces the level of connectivity of the botnet. Still, these measurements demonstrate that churn can have a significant impact on the botnet's ability to recruit bots to a particular task. In this case, while the active botnet is approximately 20,000 bots, only approximately 1000 bots are ever simultaneously performing a given task. Thus, the reported size of the botnet is far higher than the size of the botnet which can effectively be utilized. Unfortunately, this may also indicate that the churn level used in this experiment set is artificially high. This is the main motivation for the second experiment set presented in Section 4.2.7.

Figure 4.8 shows the value retrieval levels for all {*key*,*value*}-pairs during one repetition of the uninformed sybil placement strategy with 2500 targeted subnets

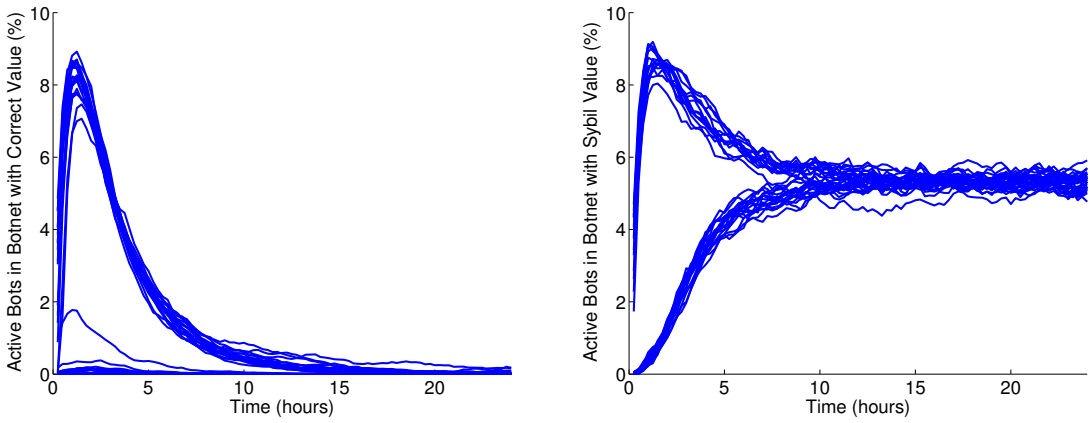


(a) CAIDA Network (b) ReaSE Network

Figure 4.7: Correct Value Retrival Levels for Simulations without Sybils

and 4000 sybils. This particular scenario was chosen arbitrarily for demonstration purposes; however, all attack scenarios in this experiment set exhibit a behaviour that can be observed in Figure 4.8. Due to the fact that i) sybils are seeded into the botnet from the start of the simulation and ii) sybils immediately attempt to publish their values into the botnet, there is an initial race between the seed hosts and the first sybils to publish their values into the botnet. As is evident from Figure 4.8, the sybils are able to win this race for a subset of {*key*,*value*}-pairs during each simulation. In Figure 4.8a, this is reflected in that some of the {*key*,*value*}-pairs are never widely distributed throughout the botnet. For these {*key*,*value*}-pairs, the corresponding series in Figure 4.8a shows that the sybil value is instead widely distributed from the beginning of the simulation and onward.

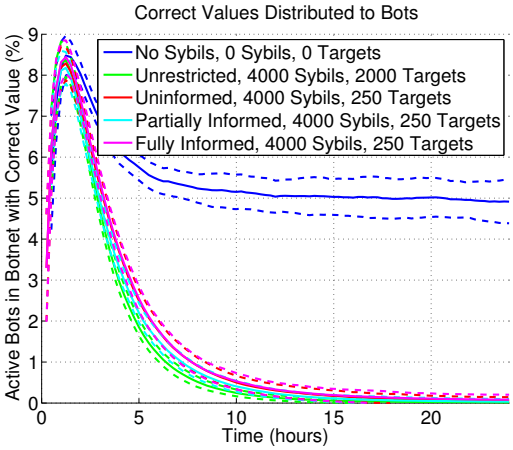
This behaviour clearly leads to multiple distinct modes of behaviour, meaning that the whole ensemble of value retrieval measurements is not ergodic. The remainder value retrieval analysis will thus only focus on measurement instances where the seed hosts manage to successfully publish the {*key*,*value*}-pair into the botnet. These instances are of particular interest because they demonstrate the performance of the sybil attack when sybil values are published mid-way through a 24-hour command cycle which is the scenario most widely researched in the literature. For this experiment set, a {*key*,*value*}-pair is considered to be successfully published if the peak correct value retrieval level is at least 70% of the maximum level observed during the ensemble of measurements for a particular scenario.



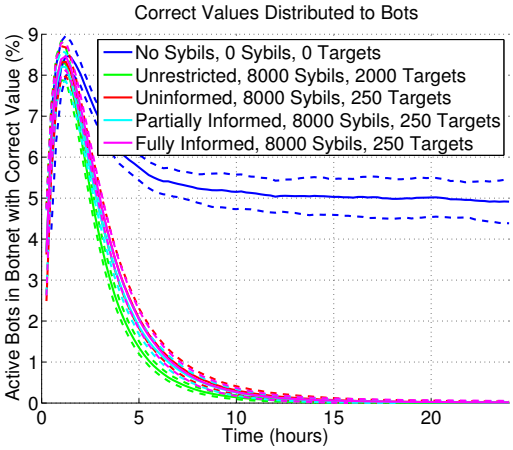
(a) Correct Values Retrieved (b) Sybil Values Retrieved

Figure 4.8: Example of value retrieval measurements for all {*key*,*value*}-pairs during a sybil attack (Uninformed sybil placement, 2500 targeted subnets, 4000 sybils).

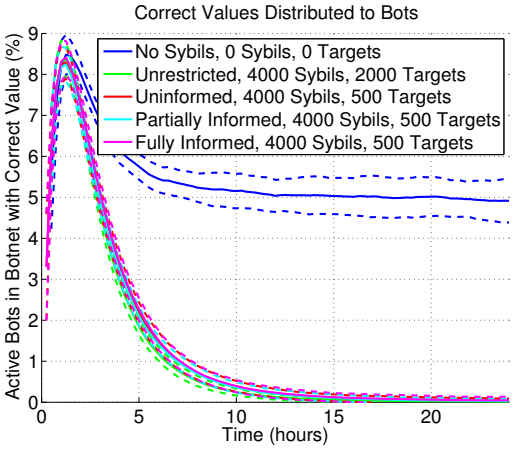
Figures 4.9 through 4.12 show the measured value retrieval levels for all experiments in this experiment set. In each case, the sybil attack is able to reduce the correct value retrieval level to below 0.5% of the botnet size. In fact, in a number of cases the sybil attack fully disrupts the botnet (*i.e.*, reduces the retrieval level to 0) before the end of the simulation. It is not feasible to compare the effectiveness of the different sybil placement strategies by the overall reduction in correct value retrieval levels at the end of each simulation since there is no statistically significant difference between them.



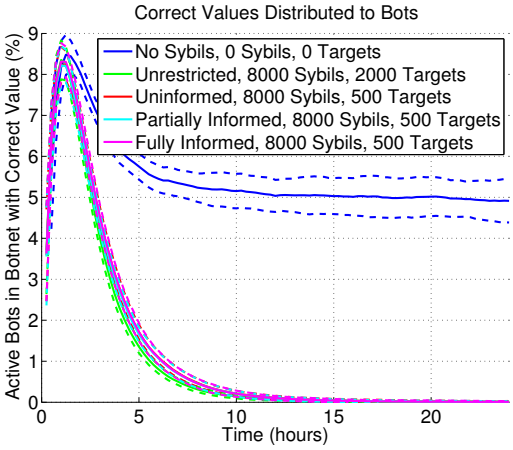
(a) 4000 Sybils, 250 Targets



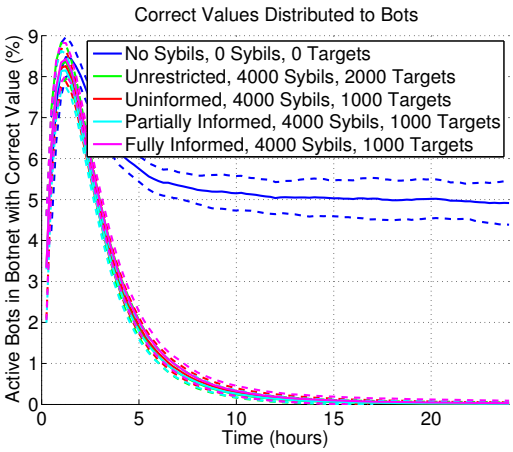
(b) 8000 Sybils, 250 Targets



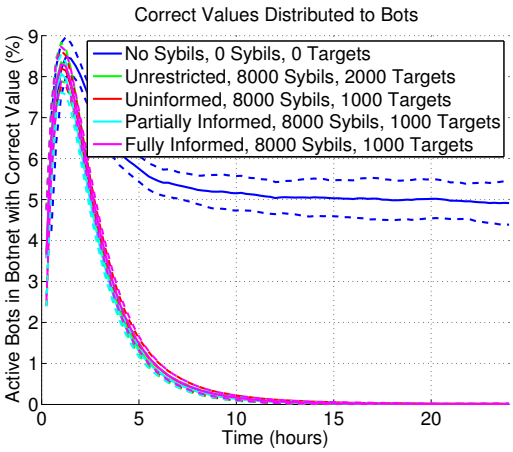
(c) 4000 Sybils, 500 Targets



(d) 8000 Sybils, 500 Targets

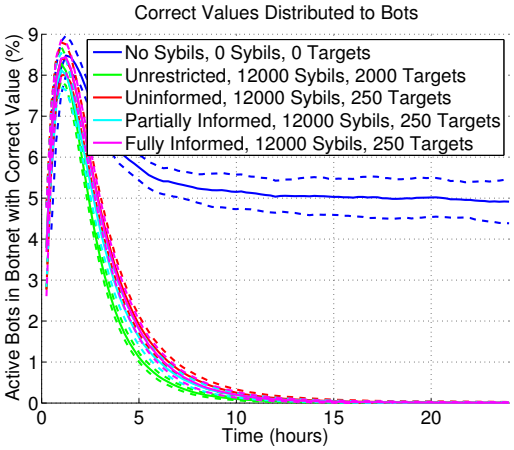


(e) 4000 Sybils, 1000 Targets

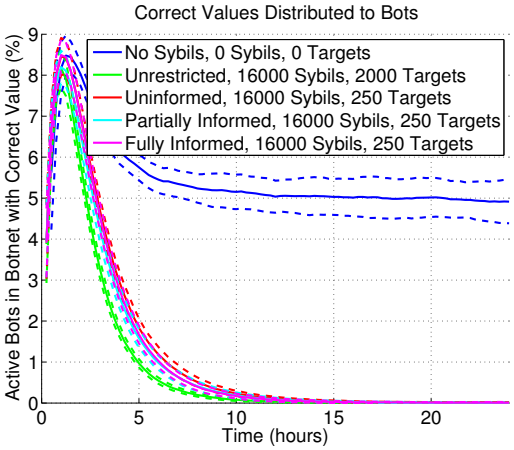


(f) 8000 Sybils, 1000 Targets

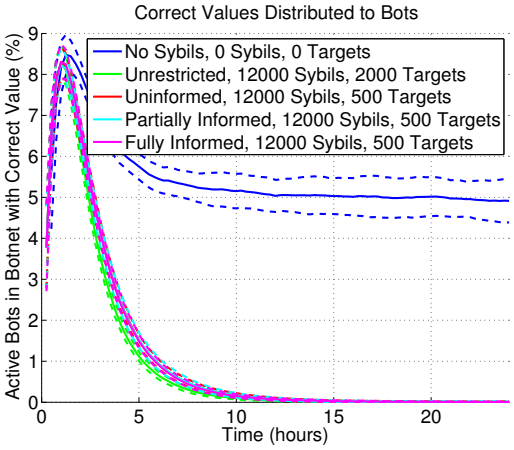
Figure 4.9: Correct Value Retrieval Levels Across CAIDA Network Simulations with 4000 and 8000 Sybils



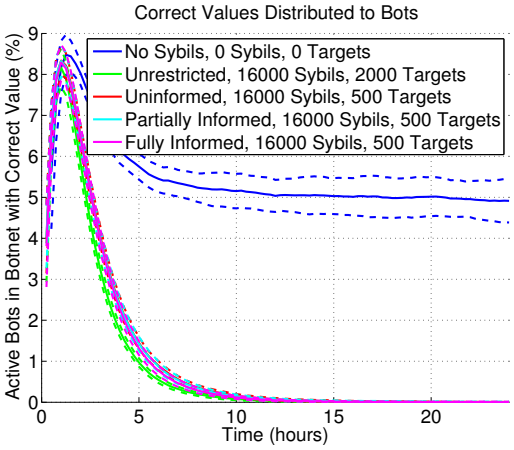
(a) 12000 Sybils, 250 Targets



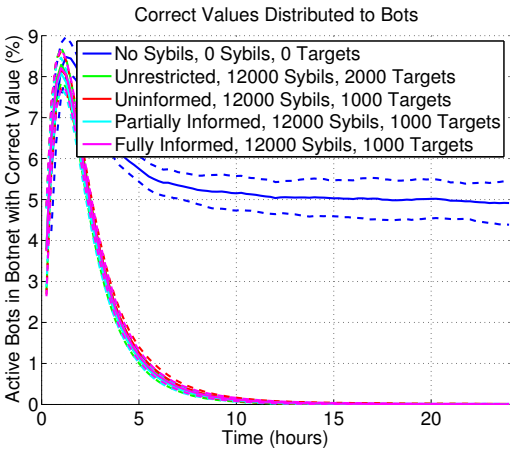
(b) 16000 Sybils, 250 Targets



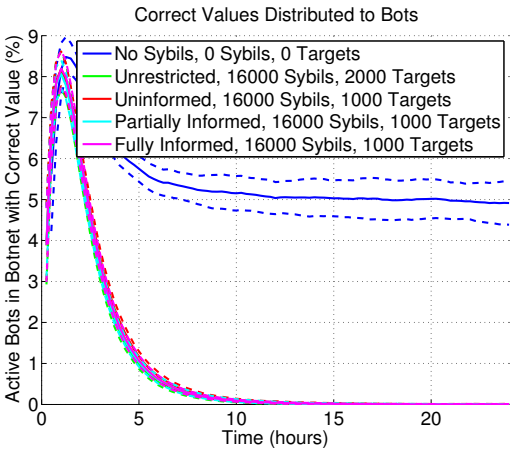
(c) 12000 Sybils, 500 Targets



(d) 16000 Sybils, 500 Targets

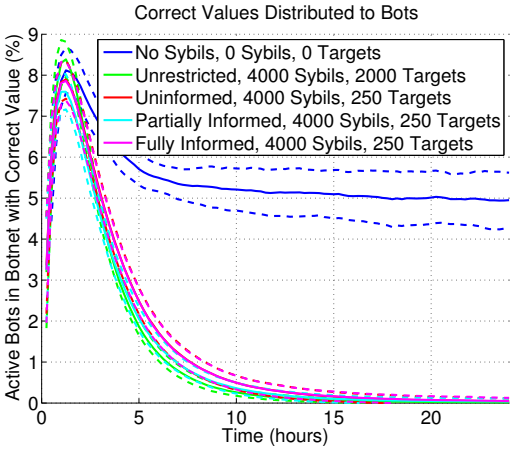


(e) 12000 Sybils, 1000 Targets

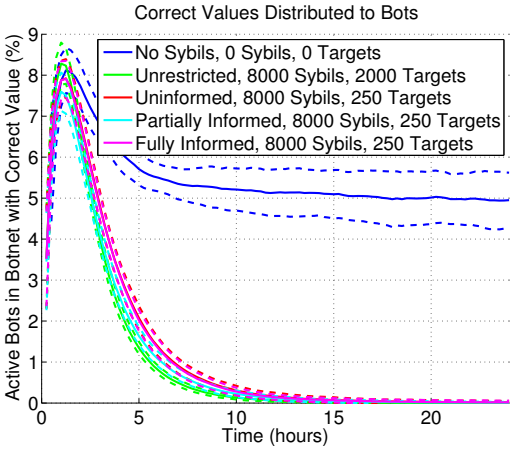


(f) 16000 Sybils, 1000 Targets

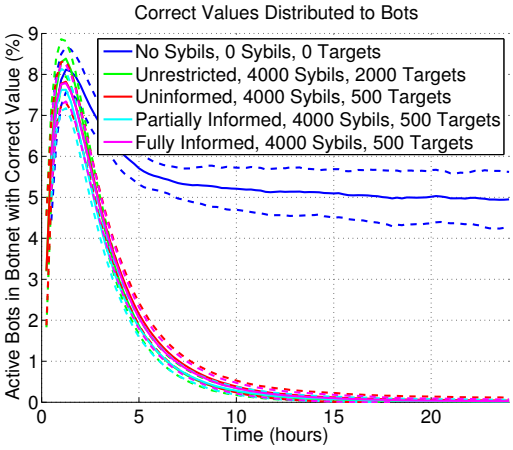
Figure 4.10: Correct Value Retrieval Levels Across CAIDA Network Simulations with 12000 and 16000 Sybils



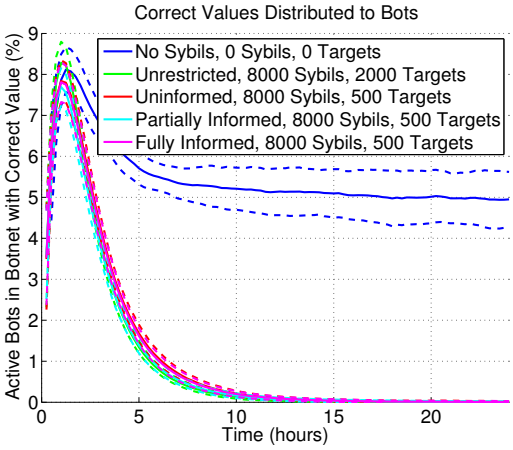
(a) 4000 Sybils, 250 Targets



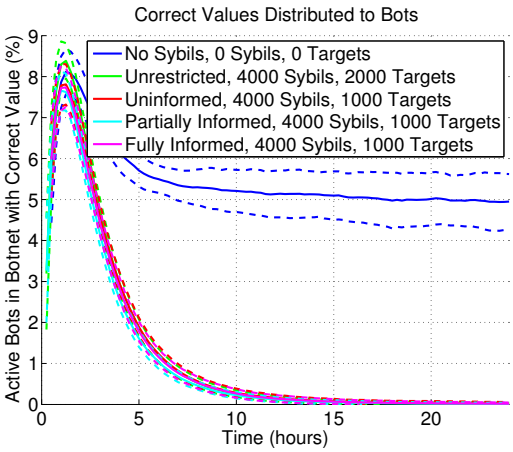
(b) 8000 Sybils, 250 Targets



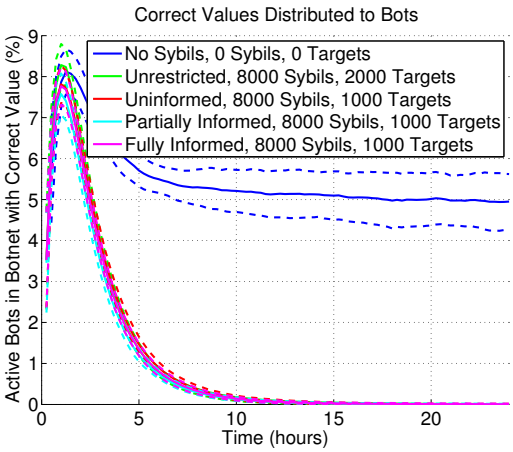
(c) 4000 Sybils, 500 Targets



(d) 8000 Sybils, 500 Targets

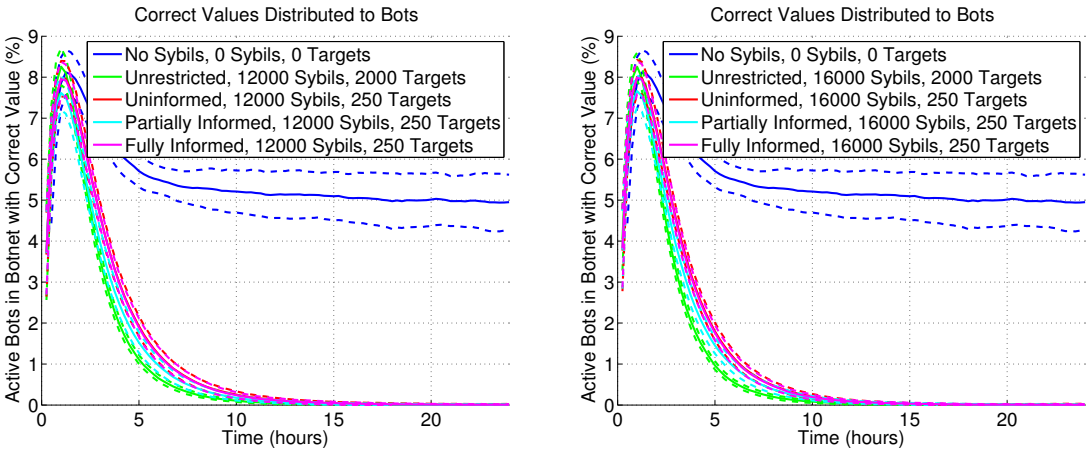


(e) 4000 Sybils, 1000 Targets



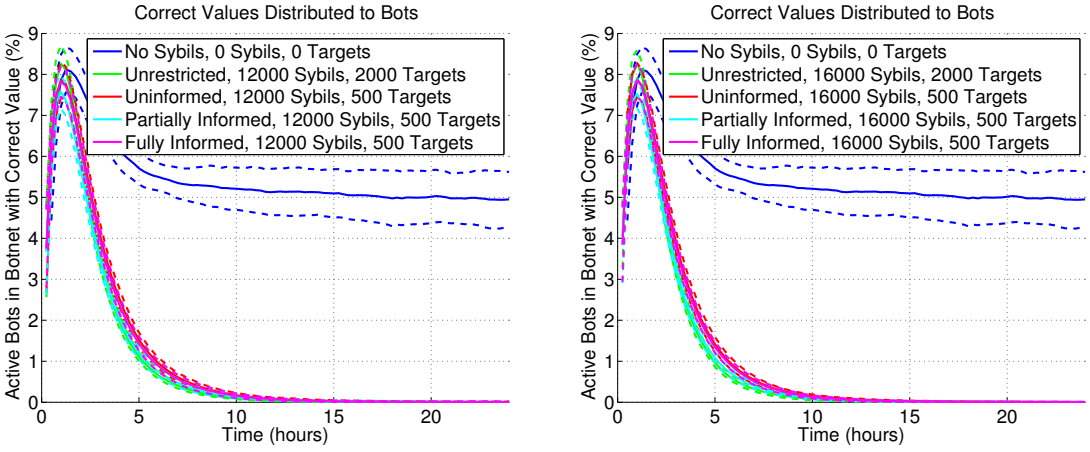
(f) 8000 Sybils, 1000 Targets

Figure 4.11: Correct Value Retrieval Levels Across ReaSE Network Simulations with 4000 and 8000 Sybils



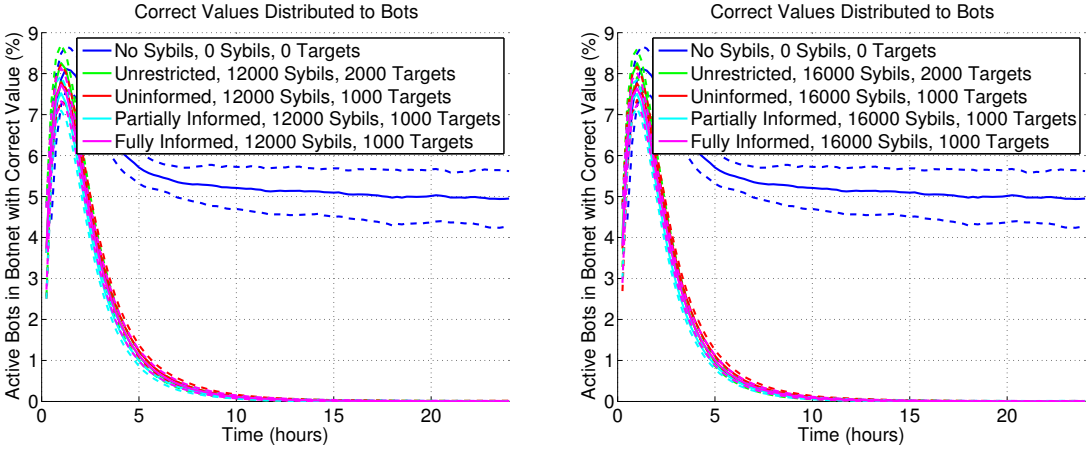
(a) 12000 Sybils, 250 Targets

(b) 16000 Sybils, 250 Targets



(c) 12000 Sybils, 500 Targets

(d) 16000 Sybils, 500 Targets



(e) 12000 Sybils, 1000 Targets

(f) 16000 Sybils, 1000 Targets

Figure 4.12: Correct Value Retrieval Levels Across ReaSE Network Simulations with 12000 and 16000 Sybils

4.2.7 Experiment Set 2: Low Churn

From the results of the previous section, it appears that by using a churn rate of ±2%/min (*i.e.*, approximately 400 bots/min), the sybil attack is able to nearly fully disable the botnet in every situation. This makes is difficult to compare the effectiveness of the different sybil placement strategies. Thus, a second set of experiments will be conducted in order to examine how effective the sybil attack is when the churn rate is reduced. During these experiments, all parameter settings will be kept identical to those outline in Section 4.2 except for the churn rate which will be reduced by an order of magnitude to ±0.2%/min (*i.e.*, approximately 40 bots/min). Also, due to time constraints, `numberOfSybils` will only be set at 4000 rather than 4000, 8000, 12,000 and 16,000.

	No Sybils	**Unrestricted**	**Uninformed**	**Partially Informed**	**Fully Informed**
Botnet Size	20,000	20,000	20,000	20,000	20,000
Churn (% of botnet size per min)	±0.2%	±0.2%	±0.2%	±0.2%	±0.2%
# of Sybils	N/A	4k	4k	4k	4k
# of Targeted Subnets	N/A	2000	250, 500, 1000	250, 500, 1000	250, 500, 1000
Duration (Sim Time)	24h	24h	24h	24h	24h
Network Topologies	Both	Both	Both	Both	Both
Repetitions Per Scenario	10	10	10	10	10
Total Simulations	20	20	60	60	60

Table 4.10: Summary of Parameter Settings for All Low Churn Experiments

4.2.7.1 Time and Storage Requirements

Tables 4.11 and 4.12 summarize the CPU time and storage requirements of the experiment set outlined above. As in Section 4.2.5, the CPU time requires exclude the CPU time required for analysis whereas the storage values include analysis results. In total, these experiments require approximately 1.8TB of storage and would have required nearly 4.5 years to complete if run serially on a single computer. By distributing the workload, these simulations were instead run over the course of a month. Although this experiment set only contains 25% of the experiments of the first experiment set, it requires approximately 50% of the CPU time and storage, meaning that the mean per-simulation CPU time and storage requirements of this experiment set are double that of the first experiment set.

	Targeted Subnets					**Total Time (All Runs)**
	0	**250**	**500**	**1000**	**2000**	
No Sybils	6d 16h 18.18GB	—	—	—	—	66d 16h 181.80GB
Unrestricted	—	—	—	—	8d 18h 7.36GB	87d 12h 73.58GB
Uninformed	—	8d 7h 7.48GB	8d 10h 7.41GB	8d 14h 7.40GB	—	252d 22h 222.78GB
Partially Informed	—	8d 11h 7.45GB	8d 13h 7.43GB	8d 21h 7.39GB	—	258d 18h 222.74GB
Fully Informed	—	8d 3h 7.35GB	8d 10h 7.35GB	8d 17h 7.32GB	—	252d 12h 220.14GB
					Total	2y 188d 8h 921.04GB

Table 4.11: Summary of CPU Time and Storage Requirements for All Low Churn CAIDA Network Simulations

4.2.7.2 Peer List Infection

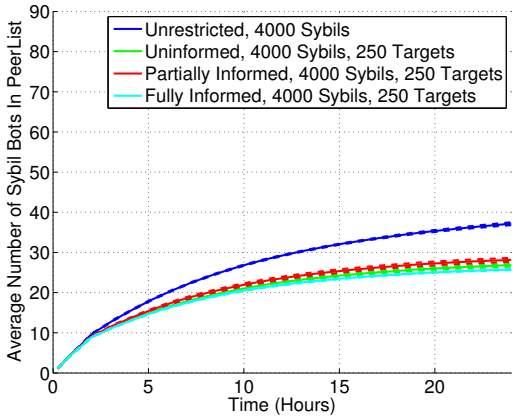
Figure 4.13 presents a summary of the mean peer-list infection levels for each experiment scenario (excluding scenarios without sybils) in a fashion similar to Section 4.2.6.1. Figure 4.14 summarizes the mean end-of-simulation peer-list infection levels for each restricted sybil placement strategy, again, in a similar fashion to Section 4.2.6.1.

	Targeted Subnets					Total Time (All Runs)
	0	**250**	**500**	**1000**	**2000**	
No Sybils	5d 8h 18.06GB	—	—	—	—	53d 8h 180.64GB
Unrestricted	—	—	—	—	6d 14h 7.39GB	65d 20h 73.85GB
Uninformed	—	6d 4h 7.43GB	6d 7h 7.40GB	6d 8h 7.35GB	—	187d 22h 221.73GB
Partially Informed	—	6d 12h 7.43GB	6d 15h 7.41GB	6d 17h 7.36GB	—	198d 8h 221.99GB
Fully Informed	—	6d 1h 7.36GB	6d 2h 7.33GB	6d 5h 7.32GB	—	183d 8h 219.99
					Total	1y 323d 18h 918.20GB

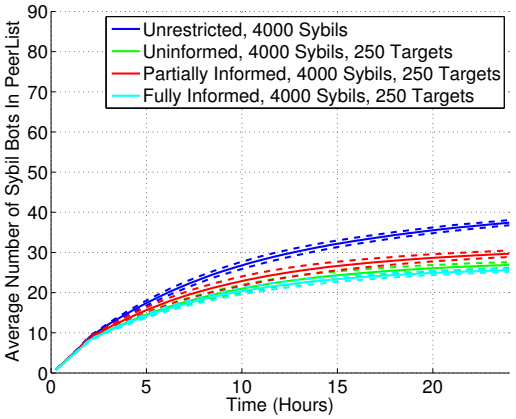
Table 4.12: Summary of CPU Time and Storage Requirements for All Low Churn ReaSE Network Simulations

In the high churn experiments the peer-list infection level rose rapidly until approximately the 4 hour mark, after which the level remained relatively constant. In the low churn experiments, the peer-list infection level rises at a much slower rate and never reaches a constant level. This difference is due to the fact that bots using the Kademlia protocol prefer to retain established peer relationships. With a lower rate of churn, it takes longer for these established peer relationships to be severed, slowing the rate at which sybils are able to infect peer-lists.

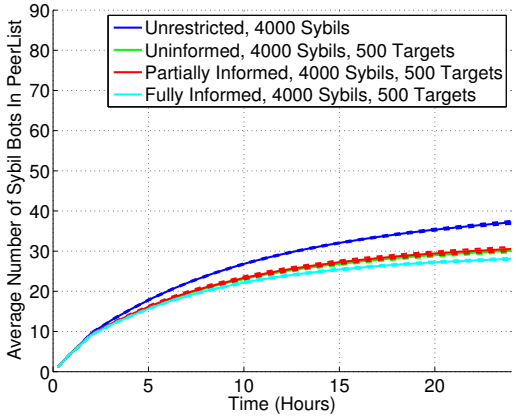
A comparison of the sybil placement strategies shows a number of trends similar to the results of the high churn experiments. The unrestricted sybil placement strategy has the highest mean peer-list infection levels. Also, of the restricted placement strategies the partially informed sybil placement strategy consistently has the highest mean peer-list infection level. However, in the low churn experiments the uninformed strategy always outperforms the fully informed strategy, even on the ReaSE network topology. As before, the overall gain resulting from using an informed sybil placement strategy is quite small, peaking at approximately a 10% increase in the mean number of infection peer-list entries. This again seems to indicate that there is not a significant gain yielded by informing the placement of sybils.



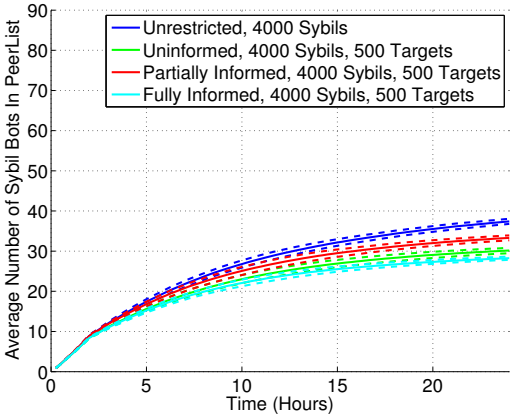
(a) CAIDA Network, 250 Targets



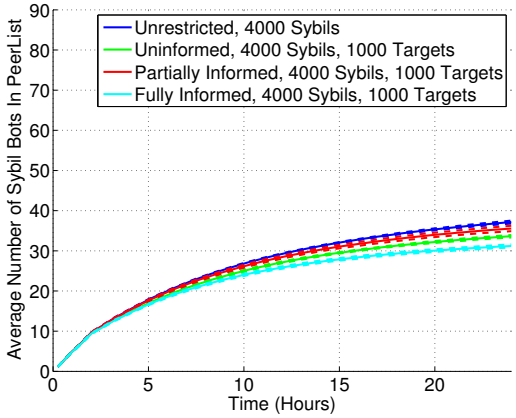
(b) ReaSE Network, 250 Targets



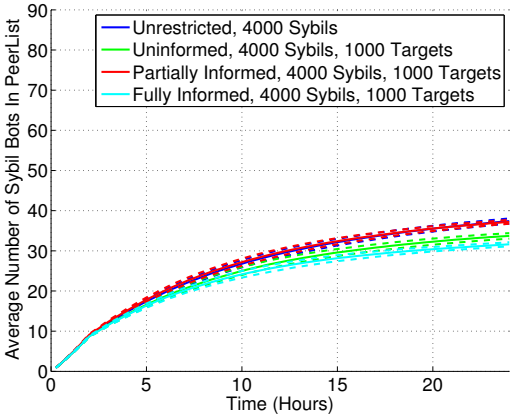
(c) CAIDA Network, 500 Targets



(d) ReaSE Network, 500 Targets

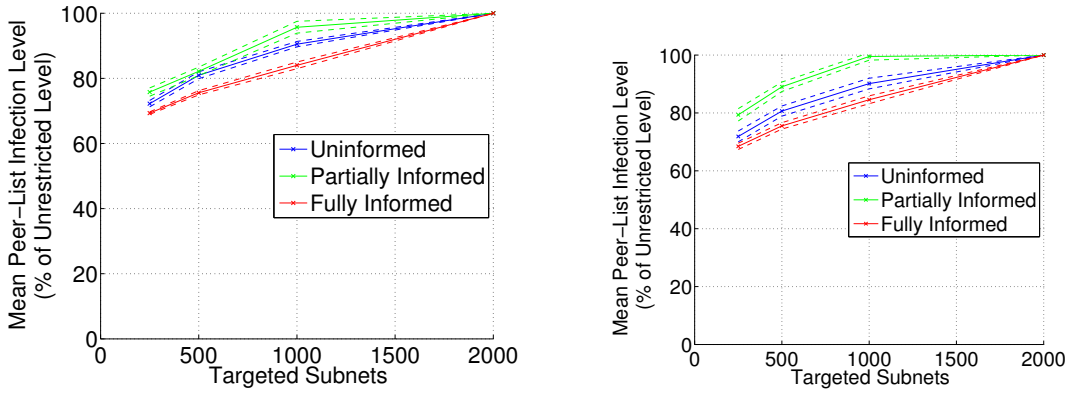


(e) CAIDA Network, 1000 Targets



(f) ReaSE Network, 1000 Targets

Figure 4.13: Mean Peer List Infection Levels for Low Churn Experiments



(a) CAIDA Network

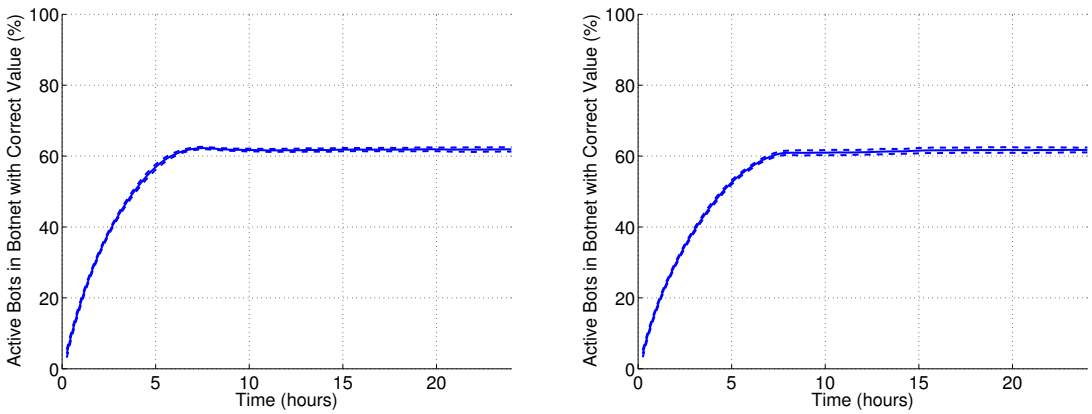
(b) ReaSE Network

Figure 4.14: Mean end-of-simulation peer-list infection levels for each restricted sybil placement strategy as percentage of levels for unrestricted placement strategy. The last point in each plot (at 2000 Targeted Subnets) is 100% since at this point the restricted placement strategies become equivalent to the unrestricted placement strategy.

4.2.7.3 Value Retrieval

Figure 4.15 shows the value retrieval level summary statistics for all simulation runs without sybils. In contrast to the high churn experiments which settled at steady state value retrieval level of approximately 5% of the botnet, the low churn experiments settle at approximately 60% of the botnet. This highlights how the performance of the botnet is sensitive to the level of bot churn: in these experiments a ten-fold increase in bot churn reduced the mean value retrieval level by approximately the same ammount. It is important to note that this work only provides measurements for two churn rates, and it is possible that the above-observed relationship is non-linear throughout this span of churn rates. Regardless, even with a reduced churn level, only 2/3 of the active bots in the botnet can be recruited to a specific task.

Figure 4.16 shows the measured value retrieval levels for all low churn scenarios. Similar to the peer-list infection levels, the value retrieval levels decrease at a far slower rate in these simulations than in the high churn experiments. At best, the low churn experiments manage to reduce the value retrieval level by approximately 45%. This is likely due to the lower peer-list infection levels during these experiments. It is also interesting to note that on the ReaSE network topology the partially informed sybil placement strategy with only 500 targets performs nearly equivalent to the

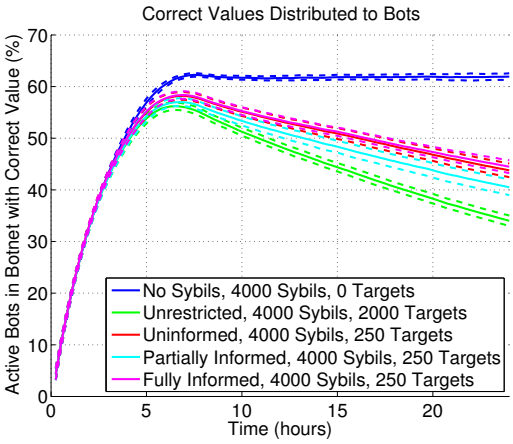


(a) CAIDA Network (b) ReaSE Network

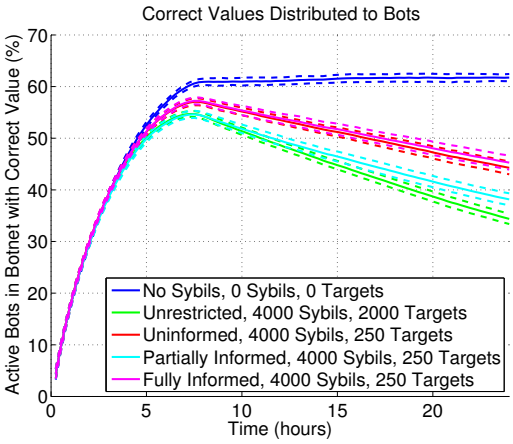
Figure 4.15: Correct Value Retrival Levels for Simulations without Sybils

unrestricted strategy; however, this is not the case on the CAIDA network topology, meaning this performance gain is not consistent across both network topologies.

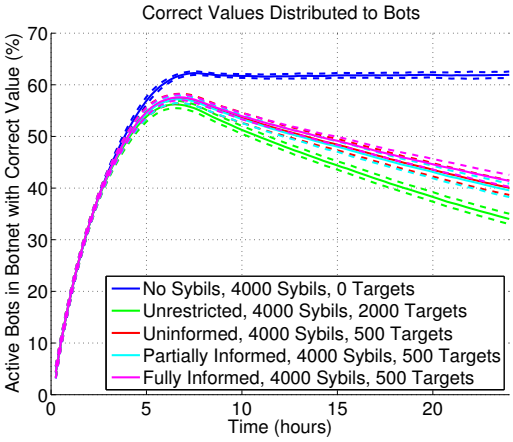
Figure 4.17 summarizes the end-of-simulation reduction in the value retrieval level for each simulated scenario. While on the ReaSE network topology the partially informed sybil placement strategy is able to reduce the value retrieval level by up to 10% more than the uninformed strategy, this is not true on the CAIDA network topology. This confirms the earlier suspicions that the gains yielded by the informed sybil placement strategies (with respect to the uninformed strategy) are not significant enough to balance the additional effort of gathering the necessary information and coordinating where sybils are placed. From a defender's perspective, this is still a valuable outcome: an uncoordinated sybil attack is still nearly as effective as a coordinated effort.



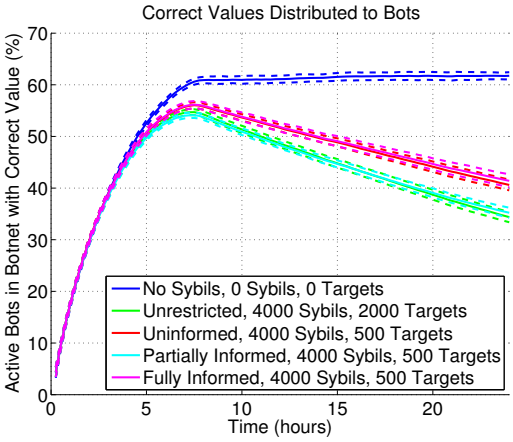
(a) CAIDA Network, 250 Targets



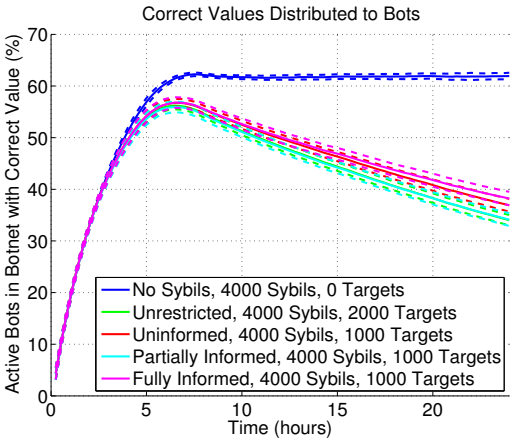
(b) ReaSE Network, 250 Targets



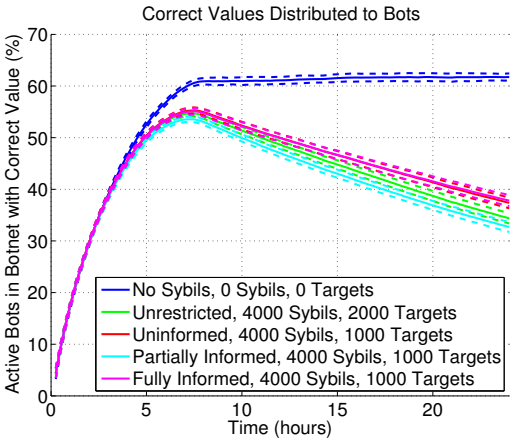
(c) CAIDA Network, 500 Targets



(d) ReaSE Network, 500 Targets

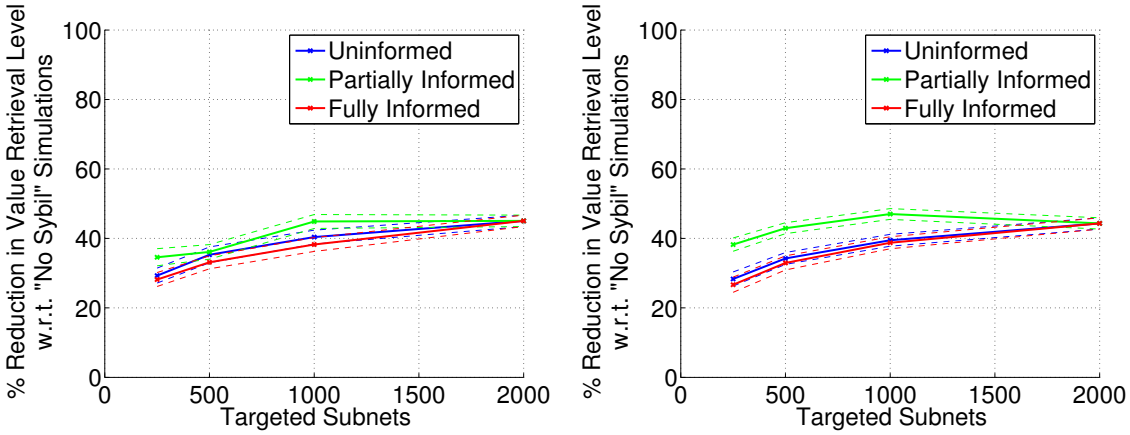


(e) CAIDA Network, 1000 Targets



(f) ReaSE Network, 1000 Targets

Figure 4.16: Correct Value Retrieval Levels Across Low Churn Simulations with 4000 Sybils



(a) CAIDA Network (b) ReaSE Network

Figure 4.17: Reduction in Correct Value Retrieval Level for All Sybil Attacks with Low Churn

4.3 Summary

This chapter presented a series of experiments to evaluate each of the sybil placement strategies outlined in Chapter 2 and as well as results and analysis of the obtained measurements. While the partially informed sybil placement strategy consistently performed the best of the restricted sybil placement strategies, the gains in peer-list infection level and reduction of value retrieval level with respect to the uninformed sybil placement strategy were i) not consistent across both network topologies and ii) were not high enough to justify the additional effort that such an approach would require to implement in the real world. While it is unfortunate that neither of the informed sybil placement strategies provided significant improvements, from a defender's perspective these results are still of value: it appears that agencies working to defend against P2P botnets by using the sybil attack do not need to invest extra effort into coordinating their attacks and monitoring where the optimal network locations for sybils are.

Aside from the specific sybil placement strategies evaluated by this work, the performance of the sybil attack was observed to be sensitive to the number of subnets that sybils were placed in. By distributing the same number of sybils to a larger number of subnets throughout the network topology, the value retrieval level of the botnet could be reduced by as much as 10%. This seems to indicate that the the proximity of sybils to bots (with respect to their location in the underlying network topology) impacts the ability of sybils to respond before other bots, thus impacting

the effectiveness of the sybil attack. Furthermore, this indicates that future research regarding the effectiveness of the sybil attack against Kademlia-based botnet protocols needs to account for the position of sybils in the underlying network topology.

This work also observered that the performance of the botnet under normal operation was sensitive to bot churn; by decreasing the churn rate by an order of magnitude, the value retrieval level increased by an order of magnitude. While the experiments in this work show the botnet's value retrieval level to vary approximately inversely proportional to the rate of bot churn, this relationship is likely to vary for other values of churn than those used during the experiments of this research. Regardless, this indicates that while bot masters may be able to recruit large numbers of bots to their botnet, if the rate of churn of these bots is high the effective number of bots which can be recuited to a particular task may be drastically smaller than the size of the botnet.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

Botnets, by virtue of their design, pose a significant global threat, with tangible political, economic and military ramifications. The decentralized nature of P2P botnets pose a unique set of challenges because many classic botnet defence strategies are either only effective against centralized botnets or do not scale well to the global scale of most botnets. The sybil attack, in combination with index poisoning, is a mitigation strategy specifically targeted at P2P botnets and has been demonstrated to be effective[49]. However, much of the literature regarding the application of the sybil attack to P2P botnets has either been conducted in an *ad hoc* fashion against botnets in the wild, which lacks the repeatability and controllability tenets of the scientific method, or has utilized research approaches which abstract away the network layer of the botnet. This work contributed a packet-level simulation of a Kademlia-based botnet and used this simulation model to evaluate several strategies for placing sybils within the underlying network topology.

Primarily, this work observed that the physical placement of sybils within the underlying network topology of a P2P botnet impacts the overall effectiveness of the sybil attack, both in terms of value retrieval level (*i.e.*, the number of active bots which have successfully retrieved a command) and mean peer-list infection level (*i.e.*, the mean number of sybil entries in the peer-lists of legitimate bots). During the low churn experiments of this work, the reduction in value retrieval level was observed to vary between 30% and 40%, simply by varying how many and which ASes sybils were

placed within. Similarly, the mean peer-list infection levels varied by 40% in the high churn experiments and 30% in the low churn experiments.

These observations are important for two reasons. First, they suggest that future research regarding the sybil attack against P2P botnets need to account for where both bots and sybils are within the underlying network topology. Second, regarding real-world deployment of sybil attacks, while it would ideally be possible to place a sybil in every autonomous system of the botnet's underlying network topology, this is not practically feasible as this amounts to placing a computer in each autonomous system which, at the current scale of the Internet, amounts to over 30,000 computers. This work indicates that both the number of ASes that sybil are placed into and the network location of that subset of ASes can significantly impact the effectiveness of the sybil attack.

Second, this work observed that using an using an informed strategy to select which AS to place sybils into does not provide a reliable increase in the effectiveness of the sybil attack. While the reduction in value retrieval level could be increased by an additional 10% by using an informed sybil placement strategy, these gains were not consistent across both of the network topologies considered in this work. Given the additional cost and effort necessary to gather the necessary information and coordinate actions between multiple agencies, it does not appear that these informed sybil placement strategies are an economically viable alternative to an uninformed (*i.e.*, random) strategy. From a defender's perspective, this means that executing a uncoordinated sybil attack with randomly placed sybils is still the most economically efficient sybil placement strategy and that such a strategy will not necessarily perform significantly worse than an informed strategy.

Third, this work observed that bot churn (*i.e.*, bots joining and leaving the botnet) can have a significant impact on the effectiveness of a Kademlia-based P2P botnet and the effectiveness of the sybil attack against such a botnet. Under normal operation, increasing the rate of bot churn by an order of magnitude resulted in an order of magnitude decrease in the value retrieval level of the botnet. While it quite possible that this relationship is non-linear, it still indicates that Kademlia-based botnets are sensitive to bot churn. Also, even at the lower rate of churn, the mean value retrieval level of the botnet was only approximately 60%. Thus, while a botnet may be reported to have hundreds of thousands of bots, this does not necessarily mean that the botmaster can recruit all of these bots to a particular task in a timely fashion. Churn also had similar effects upon the sybil attack. With high levels of churn,

even the least aggressive sybil attack scenarios resulted in near-complete disabling of botnet within 24 hours, whereas with low levels of churn the sybil attack was able to reduce the value retrieval level by 40% at most after 24 hours.

Finally, this work observered non-ergodic behaviours in the value retrieval measurements. This suggests that much richer testing is required to properly characterize the statsistical behaviours of P2P botnets than what is presented in this thesis.

In summary, the results of this work indicate that i) network-level positioning and phenomena are factors that must be accounted for when researching P2P botnets and the effectiveness of the sybil attack against such botnets, and ii) the effective power of a P2P botnet is not simply a function of the number of bots in the botnet but is a complex combination of multiple factors including the rate of churn of bots in the botnet.

5.2 Future Work

This work has provided an initial insight into the effects of network-level positioning and phenomena upon the effectiveness of the sybil attack against a Kademlia-based P2P botnet; however, the suite of simulations in this work are far from an exhaustive exploration of this design space.

This work has been primarily concerned with the physical placement of sybils within the underlying network topologies of P2P botnets while the work of Davis *et al.* in [27] was concerned with the logical placement of sybils within the overlay network. Both works have concluded that these strategies do not consistently improve the effectiveness of the sybil attack against P2P botnets. However, it is possible that strategies leveraging information and positioning in both layers may yield consistent improvements to the effectiveness of the sybil attack. Thus, further research into such approaches may yet yield more effective sybil placement strategies.

Also, although this research conducted over 1000 simulation runs, these simulations only covered two network topologies, one initial botnet state for each of these topologies, and 10 repetitions of each scenario on these initial botnet states. It would be desirable to conduct a more thorough suite of simulations using a wider array of network topologies, initial botnet states and operational parameters of the botnet. It would also be desirable to conduct simulations of larger-scale botnets and to observer how observed behaviours vary with the scale of the botnet.

Furthermore, this work utilized rudimentary statistical summaries of the measurement data generated by simulations due to the rather limited number of repetitions of each scenario. It would be desirable to conduct a more thorough statistical analysis of these measurements, especially given that this research observed non-ergodic behaviours. The main constraints that have restricted this research are the CPU time and storage requirements for such a suite of experiments: the experiments conducted during this research required approximately 5 months and nearly 5.5TB of storage. A more thorough set of simulations would need to utilise *high performance computing* (HPC) resources in order for larger suites of experiments to be practically feasible.

This work also observed interesting behaviours related to the performance of the botnet and sybil attack with respect to the level of bot churn; however, this work only conducted experiments for two levels of bot churn. It would be desirable to conduct experiments for a wider array of churn rates to better quantify its effects.

Finally, this work only considered a single botnet protocol. While the Kademlia protocol is generally well engineered and presents a number of complex issues to defenders, it would be desirable to conduct experiments with other P2P botnet protocols to quantify the effectiveness of the strategies presented in this work against such botnet protocols.

Appendix A

Appendix

A.1 Serialization of Factory-Constructed Classes

```
#ifndef __ARCHIVE_HELPER_H__
#define __ARCHIVE_HELPER_H__

#include <boost/archive/text_iarchive.hpp>

// For classes that must be created via a factory. Use this macro
// in the associated source file to override the default behavior
// when deserializing objects.
#define CALLER_ALLOCATES_WHEN_DESERIALIZING(T) \
namespace boost { \
  namespace archive { \
    namespace detail { \
      template<> \
      void pointer_iserializer<boost::archive::text_iarchive,T>::load_object_ptr( \
        basic_iarchive &ar, \
        void * & x, \
        const unsigned int file_version \
      ) const { \
        boost::archive::text_iarchive & ar_impl = \
          boost::serialization::smart_cast_reference< \
          boost::archive::text_iarchive &>( ar ); \
        BOOST_TRY { \
          ar.next_object_pointer( (T*)x ); \
        } \
        BOOST_CATCH(...) { \
          BOOST_RETHROW; \
        } \
        BOOST_CATCH_END \
        ar_impl >> boost::serialization::make_nvp( NULL,*reinterpret_cast<T*&>(x) ); \
      } \
    } \
  } \
}

#endif /* __ARCHIVE_HELPER_H__ */
```

Bibliography

[1] E. Cooke, F. Jahanian, and D. McPherson, "The zombie roundup: Understanding, detecting, and disrupting botnets," in *Proceedings of the USENIX SRUTI Workshop*, ser. SRUTI'05. Berkeley, CA, USA: USENIX Association, 2005, pp. 39–44. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251282.1251288

[2] T. Godkin, "Statistical assessment of peer-to-peer botnet features," M.A.Sc. Thesis, University of Victoria, 2013.

[3] S. S. Silva, R. M. Silva, R. C. Pinto, and R. M. Salles, "Botnets: a survey," *Computer Networks*, 2012.

[4] J. Canavan, "The evolution of malicious IRC bots," in *Virus Bulletin Conference*. Citeseer, 2005, pp. 104–114.

[5] C. Li, W. Jiang, and X. Zou, "Botnet: Survey and case study," in *Innovative Computing, Information and Control (ICICIC), 2009 Fourth International Conference on*. IEEE, 2009, pp. 1184–1187.

[6] R. Pointer and The Eggheads Development Team. Botnet sharing and linking: What is a botnet? [Online]. Available: http://www.eggheads.org/support/egghtml/1.6.17/botnet.html#sect2

[7] Symantec. (1999, May) PrettyPark.Worm. Website. [Online]. Available: http://www.symantec.com/security_response/writeup.jsp?docid=2000-121508-3334-99

[8] D. Goodin. Linux webserver botnet pushes malware. [Online]. Available: http://www.theregister.co.uk/2009/09/12/linux_zombies_push_malware/

[9] D. Web. (2012, April) Doctor Web exposes 550 000 strong Mac botnet. Website. [Online]. Available: http://news.drweb.com/show/?i=2341&lng=en&c=14

[10] C. Mullaney. (2012) Android.bmaster. [Online]. Available: http://www.symantec.com/security_response/writeup.jsp?docid=2012-020609-3003-99

[11] P. Porras, H. Saidi, and V. Yegneswaran, "An Analysis of the iKee.B (Duh) iPhone Botnet," SRI International, Tech. Rep., 2009. [Online]. Available: http://mtc.sri.com/iPhone/

[12] I. Paul. (2009, May) Nasty new worm targets home routers, cable modems. [Online]. Available: http://www.pcworld.com/article/161941/nasty_new_worm_targets_home_routers_cable_modems.html?tk=rss_main

[13] N. Ianelli and A. Hackworth, "Botnets as a vehicle for online crime," *CERT Coordination Center*, pp. 1–28, 2005.

[14] P. Traynor, M. Lin, M. Ongtang, V. Rao, T. Jaeger, P. McDaniel, and T. La Porta, "On cellular botnets: measuring the impact of malicious devices on a cellular network core," in *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 2009, pp. 223–234.

[15] P. Wood, G. Egan *et al.*, *Symantec Internet Security Threat Report*. Symantec, 2012, vol. 17.

[16] "CSBN-2: Cyber security assessment Netherlands," National Cyber Security Centre, June 2012. [Online]. Available: http://english.nctv.nl/Images/cybersecurityassessmentnetherlands_tcm92-480116.pdf

[17] "HSARPA BAA-11-02: Cyber security research and development broad agency announcement." [Online]. Available: https://www.fbo.gov/index?s=opportunity&mode=form&id=40161dd972cd60642ecaaa955e247067

[18] D. Anselmi, J. Kuo, R. Boscovich *et al.*, "Microsoft security intelligence report," 2010.

[19] D. Maughan, "Federal R&D landscape and DHS S&T overview," in *Proceedings of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research*, ser. CSIIRW '11. New York, NY, USA: ACM, 2011, pp. 4:1–4:1. [Online]. Available: https://buildsecurityin.us-cert.gov/sites/default/files/files/Rhyne-RAndD_AndDHS_SAndT.pdf

[20] M. McDowell. (2006) Understanding hidden threats: Rootkits and botnets. US-CERT. [Online]. Available: http://www.us-cert.gov/cas/tips/ST06-001.html

[21] D. Plohmann, E. Gerhards-Padilla, and F. Leder, "Botnets: 10 tough questions." European Network and Information Security Agency, 2011. [Online]. Available: http://www.enisa.europa.eu/activities/Resilience-and-CIIP/critical-applications/botnets/botnets-10-tough-questions

[22] K. Geers, *Strategic Cyber Security*. CCD COE Publications, 2011, ch. Cyber Security and National Security.

[23] (2013, May) Common vulnerabilities and exposures. Website. [Online]. Available: http://cve.mitre.org/cve/cve.html

[24] D. Dittrich, "So you want to take over a botnet," in *Proceedings of the 5th USENIX conference on Large-Scale Exploits and Emergent Threats*, ser. LEET'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 6–6. [Online]. Available: http://dl.acm.org/citation.cfm?id=2228340.2228349

[25] C. Davis, S. Neville, J. Fernandez, J.-M. Robert, and J. McHugh, "Structured Peer-to-Peer Overlay Networks: Ideal Botnets Command and Control Infrastructures?" in *Proc. 13th European Symposium on Research in Computer Security (ESORICS '08)*. Springer-Verlag, 2008, pp. 461–480.

[26] J. Grizzard, V. Sharma, C. Nunnery, B. Kang, and D. Dagon, "Peer-to-peer botnets: Overview and case study," in *Proceedings of the First Workshop on Hot Topics in Understanding Botnets*, ser. HotBots'07, Berkeley, CA, USA, 2007, pp. 1–1.

[27] C. Davis, J. Fernandez, and S. Neville, "Optimising sybil attacks against p2p-based botnets," in *Malicious and Unwanted Software, 2009. MALWARE 2009. 4th International Conference on*, October 2009, pp. 78 –87.

[28] G. Gu, J. Zhang, and W. Lee, "Botsniffer: Detecting botnet command and control channels in network traffic," in *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS08)*. San Diego, CA, 2008.

[29] C. Rossow, D. Andriesse, T. Werner, B. Stone-Gross, D. Plohmann, C. J. Dietrich, and H. Bos, "P2PWNED: Modeling and evaluating the resilience of peer-

to-peer botnets," in *Proceedings of the 34th IEEE Symposium on Security and Privacy (S&P)*, San Francisco, CA, May 2013.

[30] M. Kao, "P2p," in *Encyclopedia of Algorithms*. Springer, 2008, pp. 611–624.

[31] R. Schoof and R. Koning, "Detecting peer-to-peer botnets," University of Amsterdam, Tech. Rep., February 2007.

[32] A. Barabási and R. Albert, "Emergence of scaling in random networks," *science*, vol. 286, no. 5439, pp. 509–512, 1999.

[33] M. Ripeanu, "Peer-to-peer architecture case study: Gnutella network," in *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*. IEEE, 2001, pp. 99–100.

[34] B. Beverly Yang and H. Garcia-Molina, "Designing a super-peer network," in *Data Engineering, 2003. Proceedings. 19th International Conference on*. IEEE, 2003, pp. 49–60.

[35] S. El-Ansary, S. Haridi, and J. Wu, "An overview of structured P2P overlay networks," in *Handbook on theoretical and algorithmic aspects of sensor, ad hoc wireless, and peer-to-peer networks*. Auerbach Publications, August 2005, pp. 665–684.

[36] A. Ghodsi, "Distributed k-ary System: Algorithms for distributed hash tables," PhD Dissertation, KTH—Royal Institute of Technology, Stockholm, Sweden, Oct. 2006.

[37] P. Erdős and A. Rényi, "On the evolution of random graphs," *Magyar Tud. Akad. Mat. Kutató Int. Közl*, vol. 5, pp. 17–61, 1960.

[38] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the xor metric," in *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, ser. IPTPS '01. London, UK, UK: Springer-Verlag, 2002, pp. 53–65. [Online]. Available: http://dl.acm.org/citation.cfm?id=646334.687801

[39] D. Dittrich, F. Leder, and T. Werner, "A case study in ethical decision making regarding remote mitigation of botnets," in *Financial Cryptography and Data Security*. Springer, 2010, pp. 216–230.

[40] F. Leder, T. Werner, and P. Martini, "Proactive botnet countermeasures: an offensive approach," *The Virtual Battlefield: Perspectives on Cyber Warfare*, vol. 3, pp. 211–225, 2009.

[41] A. Ramachandran and N. Feamster, "Understanding the network-level behavior of spammers," in *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 4. ACM, 2006, pp. 291–302.

[42] M. Van Eeten, J. Bauer, H. Asgharia, S. Tabatabaie, and D. Rand, "The role of internet service providers in botnet mitigation an empirical analysis based on spam data," in *Telecommunications Policy Research Conference*, 2010. [Online]. Available: http://ssrn.com/abstract=1989198

[43] S. DiBenedetto, D. Massey, C. Papadopoulos, and P. J. Walsh, "Analyzing the aftermath of the mccolo shutdown," in *Applications and the Internet, 2009. SAINT'09. Ninth Annual International Symposium on.* IEEE, 2009, pp. 157–160.

[44] J. Nazario and T. Holz, "As the net churns: Fast-flux botnet observations," in *Malicious and Unwanted Software, 2008. MALWARE 2008. 3rd International Conference on.* IEEE, 2008, pp. 24–31.

[45] P. Lin, "Anatomy of the mega-d takedown," *Network Security*, vol. 2009, no. 12, pp. 4 – 7, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1353485810700052

[46] J. Douceur, "The sybil attack," in *Peer-to-Peer Systems*, ser. Lecture Notes in Computer Science, P. Druschel, F. Kaashoek, and A. Rowstron, Eds. Springer Berlin, 2002, vol. 2429, pp. 251–260. [Online]. Available: http://dx.doi.org/10.1007/3-540-45748-8_24

[47] J. Liang, N. Naoumov, and K. Ross, "The index poisoning attack in p2p file sharing systems," in *IEEE INFOCOM*, vol. 6, 2006.

[48] C. Davis, J. Fernandez, S. Neville, and J. McHugh, "Sybil attacks as a mitigation strategy against the storm botnet," in *Malicious and Unwanted Software, 2008. MALWARE 2008. 3rd International Conference on*, October 2008, pp. 32 –40.

[49] T. Holz, M. Steiner, F. Dahl, E. Biersack, and F. Freiling, "Measurements and mitigation of peer-to-peer-based botnets: a case study on storm worm," in *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, ser. LEET'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 9:1–9:9. [Online]. Available: http://dl.acm.org/citation.cfm?id=1387709.1387718

[50] S. Staniford, V. Paxson, N. Weaver *et al.*, "How to own the internet in your spare time," in *Proceedings of the 11th USENIX Security symposium*, vol. 8, 2002, pp. 149–167.

[51] P. Porras, H. Saïdi, and V. Yegneswaran, "A foray into conficker's logic and rendezvous points," in *Proceedings of the 2nd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more*, ser. LEET'09. Berkeley, CA, USA: USENIX Association, 2009, pp. 7–7. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855676.1855683

[52] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "Planetlab: an overlay testbed for broad-coverage services," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 3, pp. 3–12, 2003.

[53] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," in *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*. Boston, MA: USENIX Association, Dec. 2002, pp. 255–270.

[54] T. Benzel, R. Braden, T. Faber, J. Mirkovic, J. T. Wroclawski, and S. Schwab, "The deter project: Advancing the science of cyber security experimentation and test," in *Technologies for Homeland Security (HST), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1–7.

[55] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau, "Large-scale virtualization in the emulab network testbed," in *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, 2008, pp. 113–128.

[56] J. Banks, J. Carson, B. Nelson, and D. Nicol, *Discrete-Event System Simulation*, 4th ed. Prentice Hall, 2005.

[57] E. Millman, D. Arora, and S. Neville, "Stars: A framework for statistically rigorous simulation-based network research," in *Advanced Information Networking and Applications (WAINA), 2011 IEEE Workshops of International Conference on.* IEEE, 2011, pp. 733–739.

[58] H. W. Hethcote, "Three basic epidemiological models," in *Applied Mathematical Ecology.* Springer, 1989, pp. 119–144.

[59] M. Khosroshahy, M. K. Mehmet Ali, and D. Qiu, "The sic botnet lifecycle model: A step beyond traditional epidemiological models," *Computer Networks*, 2012.

[60] P. Porras, L. Briesemeister, K. Skinner, K. Levitt, J. Rowe, and Y.-C. A. Ting, "A hybrid quarantine defense," in *Proceedings of the 2004 ACM workshop on Rapid malcode.* ACM, 2004, pp. 73–82.

[61] D. Dagon, C. Zou, and W. Lee, "Modeling botnet propagation using time zones," in *Proceedings of the 13th annual network and distributed system security symposium (NDSS06)*, 2006.

[62] J. Rrushi, E. Mokhtari, and A. A. Ghorbani, "Estimating botnet virulence within mathematical models of botnet propagation dynamics," *Computers & Security*, vol. 30, no. 8, pp. 791–802, 2011.

[63] L.-P. Song, Z. Jin, and G.-Q. Sun, "Modeling and analyzing of botnet interactions," *Physica A: Statistical Mechanics and its Applications*, vol. 390, no. 2, pp. 347–358, 2011.

[64] S. Wei, J. Mirkovic, and M. Swany, "Distributed worm simulation with a realistic internet model," in *Principles of Advanced and Distributed Simulation, 2005. PADS 2005. Workshop on.* IEEE, 2005, pp. 71–79.

[65] "Network simulator ns-2," Website, Nov 2011. [Online]. Available: snam.isi.edu/nsnam/index.php/User_Information

[66] E. Van Ruitenbeek and W. H. Sanders, "Modeling peer-to-peer botnets," in *Quantitative Evaluation of Systems, 2008. QEST'08. Fifth International Conference on.* IEEE, 2008, pp. 307–316.

[67] J. Peccoud, T. Courtney, and W. H. Sanders, "Möbius: an integrated discrete-event modeling environment," *Bioinformatics*, vol. 23, no. 24, pp. 3412–3414, 2007.

[68] I. Kotenko, A. Konovalov, and A. Shorov, "Agent-based simulation of cooperative defence against botnets," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 6, pp. 573–588, 2012. [Online]. Available: http://dx.doi.org/10.1002/cpe.1858

[69] S. Agarwal, "Performance analysis of peer-to-peer botnets using the storm botnet as an example," M.Sc. Thesis, University of Victoria, 2010.

[70] A. Bensoussan, M. Kantarcioglu, and S. C. Hoe, "A game-theoretical approach for finding optimal strategies in a botnet defense model," in *Decision and Game Theory for Security*. Springer, 2010, pp. 135–148.

[71] I. Baumgart and S. Mies, "S/kademlia: A practicable approach towards secure key-based routing," in *Parallel and Distributed Systems, 2007 International Conference on*, vol. 2. IEEE, 2007, pp. 1–8.

[72] E. W. Weisstein. (2013) Birthday problem. From MathWorld—A Wolfram Web Resource. [Online]. Available: http://mathworld.wolfram.com/BirthdayProblem.html

[73] B. Edwards, S. Hofmeyr, G. Stelle, and S. Forrest, "Internet topology over time," Tech. Rep. arXiv:1202.3993, Feb 2012.

[74] V. Jacobson and S. Deering. (1989) Traceroute. [Online]. Available: ftp://ftp.ee.lbl.gov/traceroute.tar.gz

[75] K. Anagnostakis, M. Greenwald, and R. Ryger, "On the sensitivity of network simulation to topology," in *Modeling, Analysis and Simulation of Computer and Telecommunications Systems, 2002. MASCOTS 2002. Proceedings. 10th IEEE International Symposium on*, 2002, pp. 117 – 126.

[76] H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, and W. Willinger, "Network topology generators: Degree-based vs. structural," in *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4. ACM, 2002, pp. 147–159.

[77] Internet mapping and annotation. [Online]. Available: http://www.caida.org/research/topology/internet_mapping/

[78] T. Gamer and M. Scharf, "Realistic simulation environments for ip-based networks," in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, ser. Simutools '08. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, pp. 83:1–83:7. [Online]. Available: http://dl.acm.org/citation.cfm?id=1416222.1416316

[79] E. Weingartner, H. Vom Lehn, and K. Wehrle, "A performance comparison of recent network simulators," in *Communications, 2009. ICC'09. IEEE International Conference on*. IEEE, 2009, pp. 1–5.

[80] T. R. Henderson, S. Roy, S. Floyd, and G. F. Riley, "ns-3 project goals," in *Proceeding from the 2006 workshop on ns-2: the IP network simulator*. ACM, 2006, p. 13.

[81] J. Liu, "Parallel real-time immersive modeling environment (prime) scalable simulation framework (ssf)," *Colorado School of Mines, Tech. Rep*, 2006.

[82] P. García, C. Pairot, R. Mondéjar, J. Pujol, H. Tejedor, and R. Rallo, "Planetsim: A new overlay network simulation framework," in *Software engineering and middleware*. Springer, 2005, pp. 123–136.

[83] J. Pujol-Ahulló and P. García-López, "Planetsim: An extensible simulation tool for peer-to-peer networks and services," in *Peer-to-Peer Computing, 2009. P2P'09. IEEE Ninth International Conference on*. IEEE, 2009, pp. 85–86.

[84] A. Varga *et al.*, "The omnet++ discrete event simulation system," in *Proceedings of the European Simulation Multiconference (ESM2001)*, vol. 9. sn, 2001, p. 185.

[85] G. Ewing, K. Pawlikowski, and D. McNickle, "Akaroa-2: Exploiting network computing by distributing stochastic simulation," 1999.

[86] I. Baumgart, B. Heep, and S. Krause, "OverSim: A flexible overlay network simulation framework," in *Proceedings of 10th IEEE Global Internet Symposium (GI '07) in conjunction with IEEE INFOCOM 2007, Anchorage, AK, USA*, May 2007, pp. 79–84.

[87] M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 8, no. 1, pp. 3–30, 1998.

[88] I. Vari, "INET Framework for OMNeT++," 2008. [Online]. Available: http://inet.omnetpp.org/

[89] G. Van Rossum *et al.*, "Python programming language," 1994.

[90] D. McRobb, K. Claffy, and T. Monk, "Skitter: CAIDA's macroscopic internet topology discovery and tracking tool, 1999."

[91] J. Winick and S. Jamin, "Inet-3.0: Internet topology generator," Technical Report CSE-TR-456-02, University of Michigan, Tech. Rep., 2002.

[92] J. Winick. (2002, June) Inet topology generator. Website. [Online]. Available: http://topology.eecs.umich.edu/inet/

[93] A. Medina, A. Lakhina, I. Matta, and J. Byers, "Brite: An approach to universal topology generation," in *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2001. Proceedings. Ninth International Symposium on.* IEEE, 2001, pp. 346–353.

[94] A. Medina. (2002) BRITE: Boston university representative internet topology generator. Website. [Online]. Available: http://www.cs.bu.edu/brite/

[95] S. Wei and J. Mirkovic, "A realistic simulation of internet-scale events," in *Proceedings of the 1st international conference on Performance evaluation methodolgies and tools*, ser. valuetools '06. New York, NY, USA: ACM, 2006. [Online]. Available: http://doi.acm.org/10.1145/1190095.1190131

[96] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "iplane: An information plane for distributed services," in *Proceedings of the 7th symposium on Operating systems design and implementation*. USENIX Association, 2006, pp. 367–380.

[97] N. Hu, L. E. Li, Z. M. Mao, P. Steenkiste, and J. Wang, "Locating internet bottlenecks: algorithms, measurements, and implications," in *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, ser. SIGCOMM '04. New

York, NY, USA: ACM, 2004, pp. 41–54. [Online]. Available: http://doi.acm.org/10.1145/1015467.1015474

[98] R. Ramey. (2009) Boost serialization. Website. [Online]. Available: www.boost.org/libs/serialization

[99] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design patterns: Elements of reusable software components," 1995.