BotGrep: Finding Bots with Structured Graph Analysis

Shishir Nagaraja, Prateek Mittal, Chi-Yao Hong, Matthew Caesar, Nikita Borisov
University of Illinois at Urbana-Champaign
{sn275,mittal2,cyhong,caesar,nikita}@illinois.edu

April 9, 2010

Abstract

A key feature that distinguishes modern botnets from worms or other malicious hosts is their increasing use of structured overlay topologies. This lets them carry out sophisticated coordinated activities, but it can also be used as a point of detection. In this work, we devise techniques to localize botnet members based on the unique communication patterns arising from their overlay topologies used for command and control. Experimental results on synthetic topologies embedded within Internet traffic traces from an ISP's backbone network indicate that our techniques (i) can localize the majority of bots with low false positive rate, and (ii) are resilient to the partial visibility arising from partial deployment of monitoring systems, and measurement inaccuracies arising from partial visibility and dynamics of background traffic.

1 Introduction

Malware is an extremely serious threat to modern networks. In recent years, a new form of general-purpose malware known as *bots* has arisen. Bots are unique in that they collectively maintain communication structures across nodes to resiliently distribute commands from a *command and control* node. The ability to coordinate and upload new commands to bots gives the botnet owner vast power when performing criminal activities, including the ability to orchestrate surveillance attacks, perform DDoS extortion, sending spam for pay, and phishing. This problem has worsened to a point where modern botnets control hundreds of thousands of hosts and generate revenue of millions of dollars per year for their owners [23, 47].

The growing size and functionality of modern botnets has challenged the efficiency and robustness of traditional botnet architectures. Efficiently coordinate bots in the presence of these challenges requires efficient communication graphs. The problem of designing efficient communication graphs has been widely studied in the realm of *structured* overlay networks (e.g., Kademlia [59], Chord [78], Koorde [46]), and modern botnets are making increasing use of these structures. Such structures provide a number of benefits. Their lack of centralization means a botnet herder can join and control at any place, simplifying ability to evade discovery. The topologies themselves provide low delay any-to-any communication and low control overhead to maintain the structure. Further, overlay mechanisms are designed to remain robust in the face of churn. With these benefits, and with increasing use of botnet countermeasures that detect centralized command and control structures [9, 50, 29, 33, 80, 8, 58, 32, 31, 86], it is not surprising that several recently discovered botnets, such as Storm, Peacomm and Conficker, are using these structured overlays [79, 66, 67].

While coordination increases the power of botnets, it may also lay the groundwork for an effective botnet defense. The need to maintain structures and communication patterns that differ substantially from background traffic may simplify the ability to distinguish botnet hosts from uncompromised ones. ISPs, enterprise networks, and IDSs have significant visibility into these communication patterns due to the potentially large number of paths between bots that traverse their routers. However, doing this efficiently seems extremely hard, due to the rich interconnections and high variability already present in background traffic. Past work has considered a number of approaches to botnet mitigation (see section 7 for a detailed review). While signature- and flow-based approaches may monitor traffic between hosts and trigger an alarm when a certain pattern of bits is received, such approaches require advance knowledge of the communication protocols of the bots (slowing reaction time), may incur legal or privacy risks from monitoring packet data, and such schemes can be defeated by encrypting traffic or switching ports.

In this work, we focus on the question: *is it possible to isolate botnet communication structures, based solely on observing which pairs of hosts communicate?* In particular, given a *communication graph* containing both botnet and non-botnet hosts that includes edges representing command and control communication of the botnet as well as ordinary traffic (and potentially attack traffic from the botnet towards non-compromised nodes), is it possible to distinguish which nodes form the botnet communication graph? To address this problem, we propose *BotGrep*, an inference algorithm that takes in as input a set of observations (IP address pair with no port or packet level information) of the communication graph and outputs a list of hosts suspected as being part of the botnet. BotGrep works by searching for structures within the communication graph—since these botnet topologies are much more highly structured than background Internet traffic, we can partition by detecting sub-graphs that exhibit different topological patterns from each other or the rest of the graph. To leverage this observation, our approach first analyzes the graph structure to determine several metrics regarding the connectivity of hosts to different hosts. It then attempts to partition the graph into two (or more) pieces based on this metric, to separate bot from non-bot hosts. Although graph analysis has been applied to botnet and P2P detection [14, 40, 89, 39] our approach exploits the spatial relationships in communication traffic to a significantly larger extent than these works. Based on experimental results, we find that under typical workloads and topologies our techniques localize 93-99% of hosts with a false positive probability of less than 0.6%. While our techniques do not achieve perfect accuracy, we believe they can be used in conjunction with previously proposed techniques (traceback, anomaly detection, etc.) to speed up or improve confidence in botnet detections.

Roadmap: We start by giving a more detailed problem description in Section 2. In Section 3, we describe our overall approach and core algorithms, and describe privacy-preserving extensions that enable sharing of observations across ISP boundaries in Section 4. We then evaluate performance of our algorithms on synthetic botnet topologies embedded in real Internet traffic traces in Section 5. Finally, we give a brief discussion of remaining challenges in Section 6, and conclude in Section 8.

2 System Architecture

In this section we describe several challenges involved in solving this problem, as well as several unique opportunities botnets provide that we leverage in our design. We then describe our overall architecture and system design.

Challenges: In order to solve our problem, we face several key challenges. First, an ideal botnet detection infrastructure would be able to observe traffic and tag packets (or flows, or hosts) that are suspected as being part of a botnet. However, botnet traffic may be encrypted, use arbitrary ports, have arbitrary packet sizes and interarrival delays, and hence seems hard to detect just by looking at packets and their payloads. Second, background traffic on the Internet is highly variable and continuously changing, and likely dwarfs the small amount of control traffic exchanged between botnet hosts. Moreover, the botnet structure and communication patterns may not be known in advance. Third, while it is clear that multiple networks, or multiple routers within a single network, may cooperate to isolate traffic, it is less clear specifically how that isolation should be done in an algorithmic fashion.

Opportunities: At the same time, several properties shared by modern botnets may simplify this problem. First, while bots may encrypt their communications, it seems much more difficult to hide their communication graphs. For example, if a pair of hosts directly communicate, it seems difficult to hide that fact from routers. While techniques such as encryption or backdoor channels may hide the contents of that communication, they do not hide the fact that the pair of hosts are exchanging some information. At the same time, botnet structures are explicitly organized to achieve low delay or other performance properties. The overlay structures required to achieve such goals share similar properties: they should be fast-mixing to ensure any pair of nodes can communicate with low stretch and they are often regular (with individual nodes having similar outdegrees and similar relative positions within the overlay topology) to provide resilience. Moreover, there are a relatively small number of overlay structures that achieve these techniques, and that are widely studied, with public descriptions or implementations available to the botnet network designer. For example, commonly-used structures may be classified into a small number of categories: chordal rings, tori, PRR trees, and hypercubes [34]. In addition, botnet traffic traverses enterprise and ISP networks, making these networks are natural observation points for botnet communications. The need to perform efficient accounting, traffic engineering and load balancing, detection of malicious and disallowed activity, and other factors has already led network operators to deploy infrastructures to monitor traffic across multiple vantage points in their networks. Internally, ISPs run monitoring infrastructures to collect information about flow-level traffic volumes, and enterprises run intrusion detection systems to collect more fine-grained information about protocols and bit patterns occurring in packets. Some networks coordinate monitoring across administrative boundaries through use of common blacklisting services and coordinated spam detection [2].

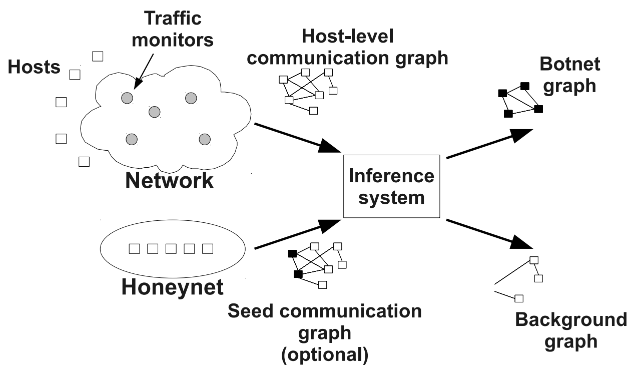


Figure 1: BotGrep architecture.

System architecture: We therefore propose an architecture and set of algorithms to isolate the set of nodes participating in a botnet network. Traffic monitoring itself has been studied in previous work (e.g., [50]), and hence our focus in this work is not on architectural issues but rather on building scalable botnet detection algorithms to operate on such an infrastructure. Our overall information-sharing architecture consists of two parts. First, *traffic monitors* are responsible for observing and sampling traffic information from the data-plane, and building a compact representation that is used for analysis and detection. These monitors may run at end-hosts, or on routers within the network using monitoring techniques such as Cisco IOS's NetFlow [41][1]. The monitors build an *observed communication graph* containing a list of (source, destination) pairs that it observed communicating, which is then used as input to our inference algorithms. Optionally, to supplement information from traffic monitors, our system may also receive as input information from Honeynets [76] or Darknets [6]. Information from these may be used as an initial "seed" for our inference

[1]By default, NetFlow *samples* traffic by only processing one out of every 500 packets. To evaluate the effect of sampling, we replayed packet-level traces collected by the authors of [47] from Storm botnet nodes, and simulated NetFlow to determine the fraction of botnet links that would be detected. We found that in the worst case (assuming each flow traversed a different router), after 50 minutes, 100% of botnet links were detected. Moreover, recent advances in counter architectures [88] may enable efficient tracking of the entire communication graph without need for sampling.

algorithms, as an input where the set of bots is known with a higher probability. Second, our *inference system* component periodically receives observed communication graphs from individual monitors, and merges them together to compute a *network-wide* communication graph. This graph contains the topology corresponding to all pairs of intercommunicating hosts observable across the set of monitors (where each pair of hosts that communicate is represented as an edge in the graph). It then runs an algorithm that analyzes the communication graph, and attempts to separate the graph into two (possibly overlapping) graphs: the botnet communication graph, and the non-botnet communication graph. Botnet hosts are then output as a set of *suspect* hosts. This list may then be sent to the set of clients that are *subscribers* to the service. The list may be used to install blacklists into routers, to configure intrusion detection, firewall systems, and traffic shapers; or as "hints" to human operators regarding which hosts should be investigated as botnet nodes. A key challenge of our work is designing an inference algorithm that can perform this partitioning efficiently, which we describe further in the next section.

3 Inference Algorithm

Our inference algorithm starts with a *communication graph* $G = (V, E)$ with V representing the set of hosts observed in traffic traces and undirected edges $e \in E$ inserted between communicating hosts. Embedded within G is a *P2P graph* $G_p \subset G$, and the remaining graph $G - G_p = G_n$ containing non-P2P communications. The goal of our algorithms is to reliably *partition* the input graph G into G_p, G_n in the presence of dynamic background traffic and in the presence of incomplete visibility.

3.1 Approach overview

The main idea behind our approach is that, since P2P topologies are much more highly structured than background Internet traffic, we can partition by detecting subgraphs that exhibit different topological patterns from each other or the rest of the graph. We do this by performing *random walks*, and comparing the relative mixing rates of the P2P sub-graph structure and the rest of the communication graph. Figure 2 shows a visualization of a P2P subgraph embedded within a communication graph collected from Abilene Internet2 ISP, and rendered using the force based algorithm of Fruchterman and Reingold [26] which minimizes the number of crossing edges. This figure highlights that we cannot partition the graph based on standard small-cut detection techniques [56] since a small-cut doesn't exist between the P2P subgraph and the rest of the graph.

The sub-graph corresponding to structured P2P traffic is expected to have a faster mixing rate than the sub-graph corresponding to the rest of the network traffic. The challenge of the problem is to partition the graph into these two sub-graphs when they are not separated by a small cut, and to do so efficiently for very large graphs.

Our approach consists of three key steps. Since the input graph could contain millions of nodes, we first apply a prefiltering step to extract a set of candidate peer-to-peer nodes. This set of nodes contains all peer-to-peer nodes, as well as false positives. Next, we use a clustering technique based on the SybilInfer algorithm [20] to cluster only the peer-to-peer nodes, and remove the false positives. The final step involves validating the result of our algorithms based on fast mixing characteristics of peer-to-peer networks.

3.2 Prefiltering Step

The key idea in the prefiltering step is that for a short random walk, the state probability mass associated with nodes in the fast mixing subgraph is likely to be closer to the stationary distribution that nodes in the

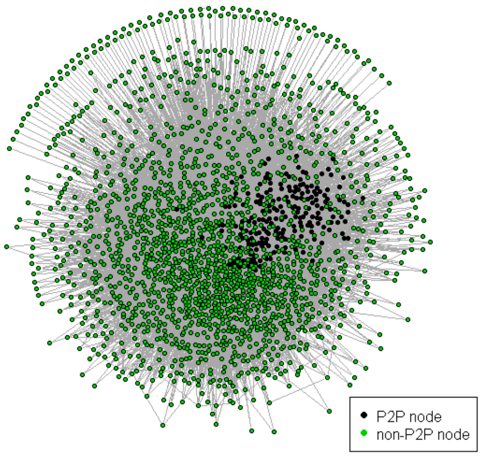


Figure 2: Abilene network with embedded P2P subgraph

slow mixing subgraph. Let P be the transition matrix of the random walks. P is defined as

$$P_{ij} = \begin{cases} \frac{1}{d_i} & \text{if } i \longrightarrow j \text{ is an edge in G} \\ 0 & \text{otherwise} \end{cases}, \tag{1}$$

where d_i denotes the degree of vertex i in G.

The probability associated with each vertex after the short random walk of length t, denoted by q^t, can be be used as a metric to compare vertices and guide the extraction of the P2P sub-graph. The initial probability distribution q^0 is set to $q_i^0 = 1/|V|$, which means that the walk starts at all nodes with the equal probability. We can recursively compute q^t as follows:

$$q^t = q^{t-1} \cdot P \tag{2}$$

Now, since nodes in the fast mixing subgraph are likely to have q^t values closer to the stationary distribution than nodes in the slow mixing subgraph, and because stationary distribution is proportional to node degrees, we can cluster nodes with homogeneous $\frac{q_i^t}{d_i}$ values. However, before doing so, we apply a transformation to dampen the negative effects of high-degree nodes on structured graph detection. High-degree nodes or hubs are responsible for speeding up the mixing rate of the non-structured subgraph G_n and can reduce the relative mixing rate of G_p vis-á-vis G_n. The transformation filter is as follows:

$$s_i = (\frac{q_i^t}{d_i})^{\frac{1}{r}} \tag{3}$$

where r is the dampening constant. We can now cluster vertices in the graph by using the *k-means* algorithm [57] on the set of values s. The *k-means* clustering algorithm divides the points in s into k ($k << |V|$) clusters such that the sum of squares J from points to the assigned cluster centers is minimized.

$$J = \sum_{j=1}^{k} \sum_{i=1}^{|} V| ||s_i - c_j||^2 \tag{4}$$

where c_j is the center of cluster j.

Each of the k clusters corresponds to a set of nodes in V_G, so we may partition our graph into subgraphs $G_1, G_2, \ldots, G_k$. We must now confirm or reject the hypothesis that each of these subgraphs is a structured P2P graph. The advantage in operating on individual clusters is that a) we can leverage parallelism when testing the hypothesis, b) we can leverage external sources of information (like honeynets) to focus our analysis on a particular cluster, the idea being that a structured P2P graph containing honeynet nodes is likely to be a P2P botnet.

Note that we can use the sparse nature of the matrix P to compute q^t using eqn 2 very efficiently in $O(|V|.d.t)$ time, where d is the average node degree. The time and space complexity of eqn 3 is $O(|V|)$, while eqn 4 can be computed in $\Omega(|V|)$ iterations in the worst case scenario. Thus the prefiltering step is a very efficient mechanism to obtain a set of candidate P2P nodes, capable of operating on multi-million node graphs.

3.3 Clustering P2P Nodes

The sub-graphs computed by the above step are likely to contain P2P nodes, but they are also likely to contain some non-P2P nodes due to the "leakage" of random walks out of the structured subgraph. We perform a second pass over the each sub-graph $G_l \in G_1, G_2, \ldots, G_k$ (or, only a particular subgraph based on honeynet information) to remove weakly connected nodes.

We cluster P2P nodes by using the SybilInfer framework introduced by Danezis and Mittal [20]. SybilInfer is a technique to detect Sybil identities in a social network graph. However, the detection algorithm used in SybilInfer relies on the existence of a small cut between the honest social network and the Sybil subgraph, and is thus not directly applicable to our setting. Next, we present a modification to the SybilInfer algorithm that is able to detect P2P nodes.

1. Generation of Traces : The first step of the clustering is the the generation of a set of random walks on the input graph. The walks are generated by performing a number n of random walks, starting at each node in the graph. A special probability transition matrix is used, defined as follows:

$$P'_{ij} = \begin{cases} \min(\frac{1}{d_i}, \frac{1}{d_j}) & \text{if } i \rightarrow j \text{ is an edge in G} \\ 0 & \text{otherwise} \end{cases}, \tag{5}$$

This choice of transition probabilities ensures that the stationary distribution of the random walk is uniform over all vertices. The length of the random walk is $O(\log|V|)$, while the number of random walks per node (denoted by n), is a tunable parameter of the system. Only the start vertex and end vertex of each random walk are used by the algorithm, and this set of vertex pairs is called the *traces*, denoted by T.

2. A probabilistic model for P2P nodes: At the heart of our detection algorithm lies a model that assigns a probability to each subset of nodes of being P2P nodes. Consider any cut $X \in V$ of nodes in the graph. We wish to compute the probability that the set of nodes X are all P2P nodes, given our set of traces T, i.e. $P(X = P2P|T)$. Through the application of Bayes theorem, we have an expression of this probability:

$$P(X = P2P|T) = \frac{P(T|X = P2P) \cdot P(X = P2P)}{Z} \tag{6}$$

where Z is a normalization constant. The prior probability P(X=P2P) can be used to encode any further knowledge about P2P nodes (using honeynets), or can simply be set to uniform over all possible cuts. Our key theoretical task here is the computation of the probability $P(T|X = P2P)$, since given this probability, we can compute $P(X = P2P|T)$ using the Bayes theorem.

Our intuition in proposing a model for $P(T|X = P2P)$ is that for short random walks, the state probability mass for peer-to-peer nodes quickly approaches the stationary distribution. Recall that the stationary distribution of our special random walks is uniform, and thus, the state probability mass for peer-to-peer nodes should be homogeneous. We can classify the random walks in the trace T into two categories: random walks that end in the set X, and random walks that end in the set $\overline{X}$ (complementary set of nodes).

Using our intuition that for short random walks, the state probability mass associated with peer-to-peer nodes is homogeneous, we assign a uniform probability to all walks ending in the set X. On the other hand, we make no assumptions about random walks ending in the set $\overline{X}$. Thus,

$$P(T|X = P2P) = \qquad \Pi_{w \in T} Prob(w|X = P2P) \tag{7}$$

where w denotes a random walk in the trace. Now if the walk w ends in vertex a in X, then we have that

$$P(w|X = P2P) = \sum_{v \in X} \frac{N_v}{n \cdot |V|} \cdot \frac{1}{|X|} \tag{8}$$

where N_v denotes the number of random walks ending in vertex v. Observe that this probability is the same for all vertices in X. On the other hand, if the walk w ends in vertex a in $\overline{X}$, then we have that

$$P(w|X = P2P) = \frac{N_a}{n \cdot |V|} \tag{9}$$

3. Metropolis-Hastings Sampling: Using the probabilistic model for P2P nodes, we have been able to compute the the probability $P(X = P2P|T)$ up to a multiplicative constant Z. However, computing Z is difficult since it involves enumeration over all subsets X of the graph. Thus, instead of directly calculating this probability for any configuration of nodes X, we will sample configurations X_i following this distribution. We use the celebrated Metropolis-Hastings algorithm [36] to compute a set of samples $X_i \tilde{P}(X|T)$. Given a set of samples S, we can compute marginal probabilities of nodes being P2P nodes as follows

$$P[i \text{ is } P2P] = \frac{\sum_{j \in S} I(i \in X_j)}{|S|} \tag{10}$$

where $I(i \in X_j)$ is an indicator random variable taking value 1 if node i is in the P2P sample X_j, and value 0 otherwise. Finally, we can use a threshold on the marginal probabilities (set to 0.5) to partition the set of nodes into fast mixing and slow mixing components.

3.4 Validation

We note that a general graph may be composed of multiple subgraphs having different mixing characteristics. However, our modified SybilInfer based clustering approach only partitions the graph into two subgraph. This means we may have to use multiple iterations of the modified SybilInfer based clustering algorithm to get to the desired fastest mixing subgraph. This raises an important question - what is the termination condition for the iteration. In other words, we need a validation test to establish that we have obtained the fast mixing P2P subgraph that we were trying to detect. Next, we propose several such validation tests.

- Graph Conductance test: It has been shown [71] that the presence of a small cut in a graph results in slow mixing and also that fast mixing implies absence of small cuts. To formalize the notion of a small cut, we use the measure of *graph conductance (Φ_x)* [48] between cuts ($X, \overline{X}$), defined as

$$\Phi_X = \frac{\Sigma_{x \in X} \Sigma_{y \notin X} \pi(x) P_{xy}}{\pi(X)}$$

  Since peer-to-peer networks are fast mixing, their graph conductance should be high (they do not have a small cut). Thus we can prevent further partitioning of a fast mixing subgraph by testing that the graph conductance between the cuts is high.

- $q^{(t)}$ *entropy comparison test:* Random walks on structured homogeneous P2P graphs are characterized by high entropy state probability distributions. This means that on a graph with n nodes, a random walk of length $t \approxeq log|n|$ results in $q_i^{(t)} = 1/n$. In this sense they are theoretically optimal. We compute the relative entropy of the state probability distribution in graph $G(V,E)$ versus its theoretical optimal equivalent graph G^T. For this we use the Kullback-Leibler (KL) divergence measure [53] to calculate the relative entropy between q_G and q_{G^T}: $F_G = \sum_x q_{G^T}(x) \log \frac{q_{G^T}(x)}{q_G(x)}$ When F_G is close to zero then the mixing rates of G and G^T are comparable. This step can be computed in $O(|V|)$ time and $O(|V|)$ space.
- *Degree-homogeneity test:* The entropy comparison test above does not rule out fast-mixing heterogeneous graphs such as a star topology. However since structured P2P graphs have relatively homogeneous degree distributions (by definition), we need an additional test to measure the dispersion of degree values. In our study, we measured the coefficient of variation of the degree distribution of G, defined as the ratio of standard deviation and mean: $c_G = \sigma/\mu$. Since the average degree is greater than zero in our graphs, it is safe to use this metric. $c_G = 0$ for a fully homogeneous degree distribution. This metric can also be computed within $O(|V|)$ time and space.

4 Privacy Preserving Graph Algorithms

In general, ISPs treat the monitoring data they collect from their own networks as confidential, since it can reveal proprietary information about the network configuration, performance, and business relationships. Thus, they may be reluctant to share the pieces of the communication graph they collect with other ISPs, presenting a barrier to deploying our algorithms. In this section, we present privacy-preserving algorithms for performing the computations necessary for our botnet detection. Fundamentally, these algorithms support the task of performing a random walk across a distributed graph.

4.1 Establishing a Common Identifier Space

Our algorithms are expressed in terms of a graph $G = (V,E)$, where the vertices are Internet hosts and edges are connections between them. This graph is assembled from m subgraphs belonging to m ASes, $G_i = (V_i, E_i)$ such that $G = \bigcup_{i=1}^{m} G_i$. To simplify computations, we would like to generate an index mapping $I : \mathbb{Z}_{|V|} \to V$. In order to do this, we use private set intersection protocols to compute $V_i \cap V_j$, for all $1 \leq i < j \leq m$. A private set intersection protocol allows two parties to compute the intersection of two sets such that each party learns only the list of elements that is part of the intersection. A number of protocols for set intersection have been proposed [11, 24, 37, 42, 52]; recently, de Cristofaro and Tsudik developed a protocol that has a *linear* cost in the number of elements in the two sets; i.e., the runtime is $O(|V_i| + |V_j|)$ [17]. This makes set intersection practical even for quite large sets.

After the intersection has been computed, the protocol assignment proceeds as follows: the first AS assigns a random index in the range $0..|V_1| - 1$ to each node in V_1. It also sends to each AS j the indices of nodes in the intersection $V_1 \cap V_j$. The next AS will assign random IDs in the range $|V_1|..|V_1 \cup V_2| - 1$ to the nodes in $V_2 \setminus V_1$. It will then send to each AS $j > 2$ the indices of common nodes, i.e., those in the intersection $(V_2 \setminus V_1) \cap V_j$. This process is repeated for each AS; at the end, all nodes in V have a unique index assigned to them, and each AS knows the index for each node in V_i.

Next, the ASes need to eliminate duplicate edges. A simple protocol is, for each pair of ASes, to compute $E_i \cap E_j$. In this computation, each node needs to include only those edges that cover nodes seen by both ASes, i.e., $e = (v_1, v_2)$ such that $v_1, v_2 \in V_i \cap V_j$. Whenever a duplicate edge is detected, one of the ASes drops the edge from its set of edges E_i.

Finally, to perform random walk, each AS needs to learn the degree of each node. Since we eliminated duplicated edges, $d(v) = \sum_{i=1}^{m} d_i(v)$, where $d_i(v)$ is the degree of node v in G_i. The sum can be computed by a standard privacy-preserving protocol, which is an extension of Chaum's dining cryptographer's protocol [12]. Each AS i creates m random shares $s_j^{(i)} \in \mathbb{Z}_l$ such that $\sum_{j=1}^{m} s_j^{(i)} \equiv d_i(v) \bmod l$ (where l is chosen such that $l > \max_v d(v)$). Each share $s_j^{(i)}$ is sent to AS j. After all shares have been distributed, each AS computes $s_i = \sum_{j=1}^{m} s_i^{(j)} \bmod l$ and broadcasts it to all the other ASes. Then $d(v) = \sum_{i=1}^{m} s_i \bmod l$. This protocol is information-theoretically secure: any set of malicious ASes S only learns the value $d(v) - \sum_{j \in S} d_i(v)$. The protocol can be executed in parallel for all nodes v to learn all node degrees.

4.2 Random Walk

We perform a random walk by using matrix operations. In particular, given a transition matrix T and an initial state vector $\vec{v}$, we can compute $T\vec{v}$, the state vector after a single random walk step. Our basic approach is to create matrices T_i such that $\sum_{i=1}^{m} T_i = T$. We can then compute $T_i\vec{v}$ in a distributed fashion and compute the final sum at the end.

To construct T_i, an AS will set the value $(T_i)_{j,k}$ to be $1/deg(v_j)$ for each edge $(j,k) \in E_i$ (after duplicate edges have been removed). Note that this transition matrix is sparse; it can be represented by N linked lists of non-zero values $(T_i)_{j,k}$. Thus, the storage cost is $O(|E_i|) \ll O(|V_i|^2)$.

To protect privacy, we use Paillier encryption [65] to perform computation on an encrypted vector $E(\vec{v})$. Paillier encryption supports a homomorphism that allows one to compute $E(x) \oplus E(y) = E(x+y)$; it also allows the multiplication by a constant: $c \otimes E(x) = E(cx)$. This, given an encrypted vector $E(\vec{v})$ and a known matrix T_i, it is possible to compute $E(T_i\vec{v})$.

Damgård and Jurik [19] showed an efficient distributed key generation mechanism for Paillier that allows the creation of a public key K such that no individual AS knows the private key, but together, they can decrypt the value. In the full protocol, one AS creates an encrypted vector $E(\vec{v})$ that represents the initial state of the random walk. This vector is sent to each AS, who then computes $E(T_i\vec{v})$. Finally, the ASes sum up the individual results to obtain $E(\sum_{i=1}^{m} T_i\vec{v}) = E(T\vec{v})$. This process can be iterated to obtain $E(T^k\vec{v})$. Finally, the ASes jointly decrypt the result to obtain $T^k\vec{v}$.

Note that Paillier operates over members $\mathbb{Z}_n$, where n is the product of two large primes. However, the vector $\vec{v}$ and the transition matrices T_i contain fractional values. To address this, we used fixed-point representation, storing $\lfloor x \times 2^c \rfloor$ (equivalently, $(x - \varepsilon) \times 2^c$, where $\varepsilon < 2^{-c}$). Each multiplication results in changing the position of the fixed point, since:

$$((x-\varepsilon_1) \times 2^c)((y-\varepsilon_2) \times 2^c) = (xy - \varepsilon_3) \times 2^{2c}$$

where $\varepsilon_3 < 2^{-c+1}$. Therefore, we must ensure that $2^{kc} < n$, where k is the number of random walk steps. The maximal length random walk we use is $2\log_{\bar{d}}|V|$, where $\bar{d}$ is the average node degree, so $k < 40$, which gives us plenty of fixed-point precision to work with for a typical choice of n (1024 or 2048 bits).

4.3 Performance

Although the base privacy-preserving protocols we propose are efficient, due to the large data sizes, the operations still take a significant amount of processing time. We estimate the actual processing costs and bandwidth overhead, using some approximate parameters. In particular, we consider a topology of 30 million hosts, with an average degree of 20 per node.[3]

[2]The CPU time is estimated based on experiments on different hardware; however, these numbers are intended to provide an order-of-magnitude estimate of the costs.

[3]The average degree in the CAIDA data set is smaller, but we expect that greater visibility will increase node degrees.

Table 1: Privacy Preserving Operations

Step	**CPU time (s)**[2]
1. Determine common identifiers	1 050 000
2. Eliminate duplicate edges	21 000 000
3. Compute node degrees	*(no crypto)*
4. Random walk (20 steps)	8 000 000

The running time of the intersections to compute a common representation is linear in $|V_i| + |V_j|$. We expect that $|V_i| < |V|$, but in the worst case, each ISP sees all of the nodes. De Cristofaro and Tsudik find that their prototype is able to compute an intersection of two sets of size 1000 in approximately 1 second. Projecting linearly, we expect to spend about 30 000s on an intersection between two ISPs. Since each pair of ISPs must perform this intersection, there will be a total of $\frac{m(m-1)}{2}$ intersections, but note that each ISP need only perform $m-1$ of these. We expect m to be around 35, based on our analysis of visibility of bot paths by tier-1 ISPs (§5.1).

The next series of set intersections involve edge sets. We expect that in most cases, the number of intersected edges will be reduced because only those edges that are in the intersection of both subgraphs are included. However, in the worst case, each ISP will see each edge, resulting in a computation time of 600 000s for this step. Even if this conservative estimate were correct, the computations involved are trivially parallelizable, thus a multi-core machine would be able to greatly reduce the actual time taken for the intersections.

A step of the random walk requires $O(|E|)$ homomorphic multiplications and additions of encrypted values. Our measurements with `libpaillier`[4] show that the multiplications are two orders of magnitude slower than additions. We were able to perform approx. 1500 multiplications per second using a 2048-bit modulus. This means that a single step would take 400 000s of computation; once again, the computation is trivial to parallelize across multiple cores or multiple machines. We expect the cost of the random walk in the pre-filtering step to dominate, since later walks will be performed on the reduced subgraph identified in the pre-filtering step.

These numbers show that a privacy-preserving version of these algorithms is feasible, given a moderate investment in computational resources by ISPs. In future work, we plan to investigate more efficient privacy-preserving schemes for collaboratively detecting botnets.

5 Results

To evaluate performance of our design, we evaluate it in the context of real Internet traffic traces. Ideally, to evaluate our design, we would like to have a list of all bots in the Internet, along with which logs of packets flowing between them, in addition to packet traces between non-botnet hosts. Unfortunately, acquiring data this extensive is very hard, due to the (understandable) reluctance of ISPs from sharing their internal traffic, and the difficulty in gaining ground truth on which hosts are part of a botnet.

To address this, we replay our approach against synthetic traces. In particular, we construct a topology containing a botnet communication graph, and embed it within a communication graph corresponding to background traffic. To improve realism, we build the background traffic communication graph by using real traffic communication graphs collected from Netflow logs from the IP backbone of the Abilene Internet2 ISP. Since Abilene's NetFlow traces are aggregated into /24-sized subnets for anonymity, we perform the same aggregation for the botnet graph, and collect experimental results over the resulting subnet-level communication graph (we expect if our design were deployed in practice with access to per-host information,

[4] `http://acsc.cs.utexas.edu/libpaillier/`

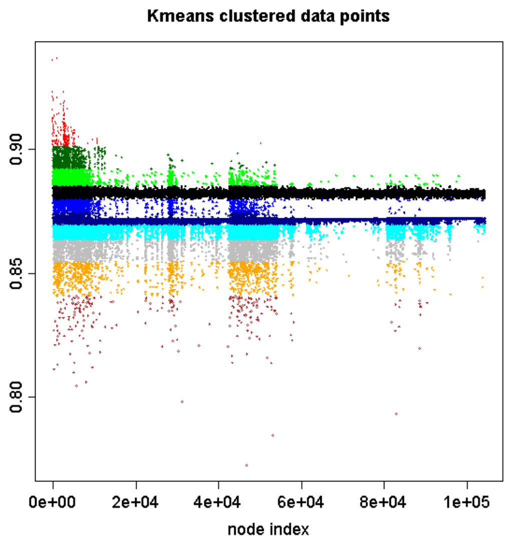


Figure 3: The filtered limit distribution (s_i) after clustering

its performance would improve due to increased visibility). To investigate sensitivity of our results to this methodology and data set, we also use packet-level traces collected by CAIDA on OC192 Internet backbone links [3]. To construct the botnet graph, we select a random subset of nodes in the background communication graph to be botnet nodes, and synthetically add links between them corresponding to a particular structured overlay topology. We then pass the combined graph as input to our algorithm. By keeping track of which nodes are bots (this information is not passed to our algorithm), we can acquire "ground truth" to measure performance. To investigate sensitivity of our techniques to the particular overlay structure, we consider several alternative structured overlays, including (a) Chord, (b) de Bruijn, (c) Kademlia, and (d) the "robust ring" topology described in [44]. The remainder of this section contains results from running our algorithms over the joined botnet and Internet communication graphs, and measuring the ability to separate out the two from each other.

Before we proceed to the results, we first illustrate our inference algorithm with an example run.

5.1 Algorithm Example

Let us consider a specific application of our algorithm on a synthetically-generated de Bruijn [46] peer-to-peer graph embedded within a communication graph sampled from the Internet (using NetFlow traces from the Abilene Internet2 ISP). The Abilene communication graph G_D contains $|V_D| = 104426$ nodes and $|E_D| = 547053$ directed edges. We then generated a de Bruijn graph G_p of 10000 nodes, with $m = 10$ outgoing links and $n = 4$ dimensions (10% of $|V|$). G_p is then embedded in G_D by mapping a node in G_p into a node in G_D: for every node $i \in V_B$ we select a node $j \in V_D$ uniformly at random between 1 and $|V_D|$ without replacement, and add the corresponding edges in E_B to E_D. The resulting graph is $G(V,E)$ with $N = |V| = 104426$ nodes and $|E| = 647053$ edges. The goal of our detection technique is to extract G_p from G_D as accurately as possible.

First, we apply the pre-filtering step: we carry out a short random walk starting from every node with probability $1/N$ to obtain $q^{(t)}$, on which the transformation filter of eqn 3 is applied to obtain s. We used a dampening constant of $r = 100$ to undermine the influence of hub nodes on the random walk process. The data points in s corresponding to each of the partitions returned by k-means clustering is shown in Figure 3.

In the example we consider here, applying the k-means algorithm gives us ten potential P2P candidates.

In a completely unsupervised setting, we would need to run the modified SybilInfer algorithm on each of the candidate sets. However we expect that the analysis can simply be focused on the candidate set containing the set of honey-net nodes. Thus, let us consider the graph nodes corresponding to the fourth cluster (colored in yellow). The cluster size is 17576 nodes.

Next, we recursively apply the modified SybilInfer partitioning algorithm to this cluster. After three iterations of the SybilInfer partitioning algorithm, we obtain a subgraph of size 10143 nodes, containing 9905 P2P nodes, and 238 other nodes. At this stage, our set of validation conditions indicates that the subgraph is indeed fast mixing, and we stop the recursion. The final subgraph has a graph conductance of about 0.9 (In all other scenarios, the graph conductance is less than 0.5, so we can safely set our threshold of the graph conductance test to be 0.5); the KL-divergence of the final graph is less than 0.1 (the KL-divergence is greater than 0.45 in all other scenarios); the entropy of the final graph is greater than 0.97 (it is lesser than 0.63 in all other scenarios); and the coefficient of variation is less than 1 in the final graph (it is greater than 4.6 in all other scenarios).

To evaluate performance, we are concerned with the *false positive rate* (the fraction of non-bot nodes that are detected as bots) and the *false negative rate* (the fraction of bot nodes that are not detected). These results are shown in Tables 2 and 3. All results are averaged over five random seeds. Overall, we found that BotGrep was able to detect 93-99% of bots over a variety of topologies and workloads. In particular, we observed several key results:

Effect of botnet topology: To study applicability of our approach to different botnet topologies, we consider Kademlia [59], Chord [78], and de Bruijn graphs. In addition, we also consider the LEET-Chord topology [44], a recently proposed overlay topology that aims to be difficult to detect (cannot be reliably detected with traffic dispersion graph techniques). Overall, we find performance to be fairly stable across multiple kinds of botnet topologies, with detection rates higher than 95%. In addition, BotGrep is able to achieve a false positive rate of less than 0.42% on the harder-to-detect LEET-Chord topology. While our approach is not perfectly accurate, we envision it may be of use when coupled with other detection strategies (e.g., previous work on botnet detection [43, 40], or if used to signal "hints" to network operators regarding which hosts may be infected.

Although LEET-Chord topology is harder to detect by our approach, this comes at a tradeoff with less resilience to failure. To study the robustness of LEET-Chord topology, Figure 4 shows the performances of Chord and LEET-Chord by randomly removing varying percentages of nodes. We observed that LEET-Chord is much less resilient to node failures as compared with Chord.

Effect of botnet graph size: Next, we vary the size of the embedded botnet. We do this to investigate performance as a function of botnet size, for example, to evaluate whether BotGrep can efficiently detect small botnets (e.g., bots in early stages of deployment, which may have greater chance of containment) and large-scale botnets (which may pose significant threats due to their size and large topological coverage). We perform this experiment by keeping the size of the background traffic graph constant, and generating synthetic botnet topologies of varying sizes (between 100 and 100,000 bots). Overall, we found that as the size of the bot graph increases, performance degrades, but only by a small amount. For example, in Table 2, with the fully visible de Bruijn topology, for 100 nodes the false positive rate is zero, while for 10,000 nodes the rate becomes 0.12%.

Effect of background graph size: One concern is that BotGrep may perform less accurately with larger background graphs, as it may become easier for the botnet structure to "hide" in the increasing number of links in the graph. To evaluate sensitivity of performance to scale, we vary the size of the background communication graph, by evaluating over both the Abilene and CAIDA dataset (104,426 and 3,839,936 nodes, respectively). To get a rough sense of performance on much larger background graphs, we also build a "scaled up" version of the CAIDA graph containing 30 million hosts while retaining the statistical properties of the CAIDA graph. To scale up the CAIDA graph G_c by a factor of k, we make k copies of G_c,

Topology	$\lvert V_B \rvert$	% FP	% FN	% Detected
de Bruijn	100	0.00	2.00	98.00
	1000	0.01	2.40	97.60
	10000	0.12	2.35	97.65
Kademlia	100	0.00	3.20	97.80
	1000	0.01	2.48	98.52
	10000	0.10	2.12	97.88
Chord	100	0.00	3.00	97.00
	1000	0.01	2.32	97.68
	10000	0.08	1.94	98.06
LEET-Chord	100	0.00	3.00	97.00
	1000	0.03	1.60	98.40
	10000	0.42	1.00	99.00

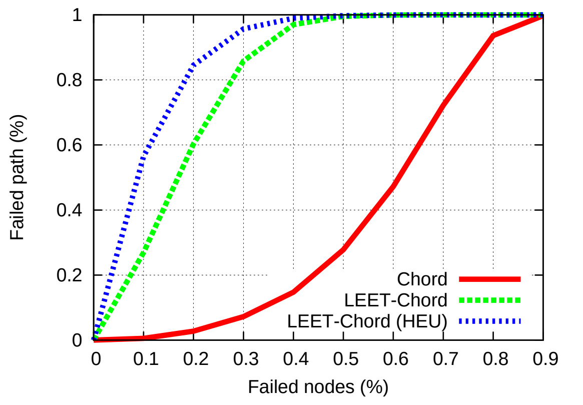
Table 2: Abilene – detection and error rates of inference

Topology	$\lvert V_B \rvert$	% FP	% FN	% Detected
de Bruijn	1000	0.00	1.80	98.20
	10000	0.01	0.93	99.07
	100000	0.09	0.67	99.33
Kademlia	1000	0.00	2.10	97.90
	10000	0.01	0.80	99.20
	100000	0.19	0.17	99.83
Chord	1000	0.00	2.20	97.80
	10000	0.01	0.48	99.52
	100000	0.06	0.46	99.54
LEET-Chord	1000	0.00	0.40	99.60
	10000	0.02	0.48	99.52

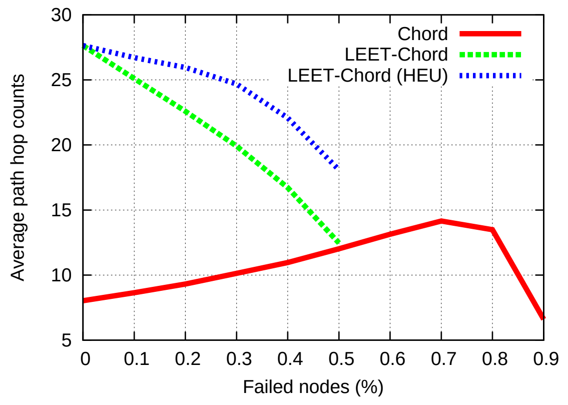
Table 3: CAIDA – detection and error rates of inference

Topology	$\lvert V_B \rvert$	% FP	% FN	% Detected
de Bruijn	100000	0.01	0.8	99.20
Kademlia	100000	0.01	0.4	99.60
Chord	100000	0.01	0.4	99.60

Table 4: CAIDA 30M – detection and error rates of inference



(a) Percentage of failed path for varying percentages of node failures across Chord-based topologies



(b) Average path hop-counts of survival paths for varying percentages of node failures across Chord-based topologies

Figure 4: Robustness of Chord and Leet-Chord with 65,536 nodes. LEET-Chord (HEU): routing preceeds as usual LEET-Chord. But when the destination is outside the cluster of node and long range links are all failed, it greedily forwards the packet to next clockwise cluster using intra-cluster links.

namely $G_1 \dots G_k$ with vertex sets $V_1 \dots V_k$ and edge sets $E_1 \dots E_k$. Note that for each edge (p,q) in E_r, we have a corresponding edge in each copy $G_1 \dots G_k$, we refer to these as $(p_1,q_1)\dots(p_k,q_k)$. We then compute the graph disjoint union over them as $G_S(V_S,E_S)$ where $V_S = (V_1 \cup V_2 \cdots \cup V_k$ and $E_S = E_1 \cup E_2 \cdots \cup E_k)$. Next, we randomly select a fraction of links from E_S to obtain a set of edges E_r that we shall rewire. As a heuristic, we set the number of links selected for rewiring to $|E_r| = k\sqrt{(N)}\log(N)$ where N is the number of nodes in the CAIDA graph G_c. For each edge (p,q) in E_r we wish to rewire, we choose two random numbers a and b $(1 \le a,b \le k)$ and rewire edges (p_a,q_a) and (p_b,q_b) to (p_a,q_b) and (p_b,q_a) such that $d_{p_a} = d_{p_b}$ and $d_{q_a} = d_{q_b}$. This edge rewiring ensures that (a) the degree of all four nodes p_a,q_a,p_b and q_b remains unchanged, (b) the joint degree distribution $P(d_1,d_2)$ – the probability that an edge connects d_1 and d_2 degree nodes remains unchanged, and (c) $P(d_1,d_2,\dots d_l)$ remains unchanged as well, where l is the number of unique degree values that nodes in G_c can take.

Overall, we found that BotGrep scales well with network size, with performance remaining stable as network size increases. For example, in the CAIDA dataset with a background graph of size 3.8 million hosts, the false positive rate for the de Bruijn topology of size 100000 is 0.09% (shown in Table 3), while for the scaled up 30 million node CAIDA topology, this rate is 0.01 (shown in Table 4). Observe that the false positive rate has decreased by a factor of 9, which is approximately equal to the scale up factor between the two topologies, indicating the the actual number of false positives remains the same.

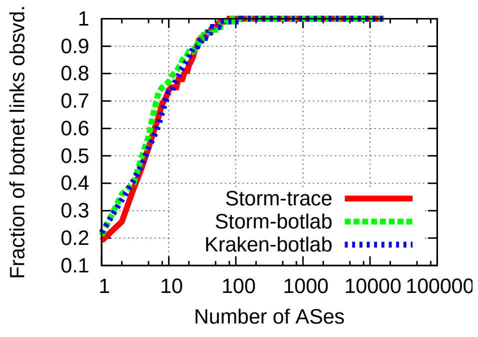


Figure 5: Number of visible botnet links, as a function of number of most-affected ASes contributing views.

Topology	$\lvert V_B \rvert$	% FP	% FN	% Detected
de Bruijn	100	0.00	3.00	97.00
	1000	0.02	2.80	97.20
	10000	0.17	3.31	96.69
Kademlia	100	0.00	3.75	96.25
	1000	0.01	2.90	97.10
	10000	0.19	2.07	97.93
Chord	100	0.00	9.00	91.00
	1000	0.02	3.50	96.50
	10000	0.13	2.54	97.46
LEET-Chord	100	0.00	6.00	94.00
	1000	0.06	2.70	97.30
	10000	0.58	1.80	98.20

Table 5: Abilene – results if only Tier-1 ISPs contribute views

Topology	$\lvert V_B \rvert$	% FP	% FN	% Detected
de Bruijn	1000	0.00	2.70	97.30
	10000	0.00	4.22	95.78
	100000	0.12	1.74	98.26
Kademlia	1000	0.00	0.50	99.50
	10000	0.01	0.30	99.70
	100000	0.09	0.53	99.47
Chord	1000	0.00	3.40	96.60
	10000	0.01	0.65	99.35
	100000	0.06	5.36	94.64
LEET-Chord	1000	0.01	0.20	99.80
	10000	0.02	1.09	98.91

Table 6: CAIDA – results if only Tier-1 ISPs contribute views

Effect of reduced visibility: In the experiments we have performed so far, the embedded structured graph G_p is present in its entirety. However, just as G_D is obtained by sampling Internet or enterprise traffic, only a subset of botnet control traffic will actually be available to us. It is therefore important to evaluate how well our algorithms work with graphs where only a fraction of the structured subgraph edges are known. To study this, we evaluate performance of our scheme when deployed at only a subset of ISPs in the Internet. To do this, we collected roughly 4,000 Storm botnet IP addresses from Botlab [1] (*botlab-storm*), and measured what fraction of inter-bot paths were visible from tier-1 ISPs. From an analysis of the Internet AS-level topology, we find that 60% of inter-bot paths traverse tier-1 ISPs. We found that if the most-affected ASes cooperate—the ASes with the largest number of bots—this number increased to 89%). Figure 5 shows this result in more detail. Here, we vary the number of ASes cooperating to contribute views (assuming the most-affected ASes contribute views first), plotting the number of visible inter-bot links. We repeat the experiment also for the Kraken botnet trace from [1] (*kraken-botlab*), as well as a packet-level trace from the Storm botnet (*storm-trace*). We find that if only the 5 most-affected ASes contribute views, 57% of Storm links and 65% of Kraken links were visible.

We therefore removed 40% of links from our botnet graphs (Table 5 and Table 6). While the false-negative rate increases, our approach still detects over 90% of botnet hosts with high reliability (the false positive rate for the hard to detect LEET-Chord topology still remains less than 0.58%). Disabling or removing such a large fraction of nodes will lead to certain loss of operational capability.

Topology	$\lvert V_B \rvert$	% FP	% FN	% Detected
de Bruijn	100000	0.04	0.8	99.20
Kademlia	100000	0.05	0.4	99.60
Chord	100000	0.04	0.4	99.60

Table 7: Leveraging Honeynets : CAIDA – detection and error rates of inference.

Leveraging Honeynets: We shall now present an extension to our inference algorithm that leverages the knowledge of a few known bot nodes. This extension considers random walks starting only from the honeynet nodes to obtain a set of candidate P2P nodes in the prefiltering stage. Using this extension, we find that there is a significant gain in terms of reducing the false positives, as well as speeding up the efficiency of the protocol. As Table 7 shows, the false positive rate for the Kademlia topology has been reduced by a factor of 4 as compared to corresponding value in Table 3. Furthermore, only a single iteration of the modified SybilInfer algorithm was required to obtain the final subgraphs, providing a significant gain in efficiency.

Topology	BotGrep	Fast Greedy Modularity	Girvan-Newman Betweenness	Modularity Eigenvector
de Bruijn	0.78/2.55	14.43/7.65	19.73/15.31	0.92/43.88
Chord	0.77/7.15	7.58/10.13	6.05/19.50	4.24/20.19
Kademlia	0.92/7.00	14.66/33.80	18.06/4.75	5.70/48.70

Table 8: 2k Abilene Results (% FP /% FN)

Effect of inference algorithm: For comparison purposes, we also consider several graph partitioning algorithms that have been proposed in the literature. While these techniques were not intended to scale up to the large data sets we consider here, we can compare against them on smaller data sets to get a sense of how BotGrep compares against these approaches. In particular, several algorithms for *community detection* (detecting groups of nodes in a network with dense internal connections) have been proposed. Work in this space mainly focuses on *hierarchical clustering methods*. Work in this space can be classified as following two categories, and for our evaluation we implement two representative algorithms from each category:

- Edge importance based community structure detection iteratively removes the edges with the highest *importance*, which can be defined in different ways. Girvan and Newman [28] defined edge importance by its shortest path betweenness. The idea is that the edge with higher betweenness is typically responsible for connecting nodes from different communities. In [21], *information centrality* has been proposed to measure the edge importance. The information centrality of an edge is defined as the relative *network efficiency* [54] drop caused by the removal of that. The time complexity of algorithm in [28] and [21] are $O(|V|^3)$ and $O(|E|^3 \times V)$, respectively.

- The spectral-based approach detects communities by optimizing the modularity (a benefit function measures community structure [62] over possible network divisions. In [63], the communities are detected by calculating the eigenvector of the modularity matrix. It takes $O(|E|+|V|^2)$ time to separating each community. Moreover, Clauset *et al.* [13] proposed a hierarchical agglomeration algorithm for community detecting. The proposed greedy algorithm adopts more sophisticated data structures to reduce the computation time of modularity calculation. The time complexity is $O(|E|+|V|\log 2|V|)$ in average.

As the time complexity of above algorithms is not acceptable for computing large-scale networks, here we consider a small-scale scenario for performance evaluation. We extract subgraphs from full Abilene data by performing a Breadth-First-Search (BFS) starting at a randomly selected node, in which the overall visited nodes are limited by a size of 2000. Results from our comparison are shown in Table 8. The information centrality algorithm took more than one month to run for just one iteration on this 2000-node graph, and was hence excluded from further analysis (we tested information centrality on smaller 50-node graphs, and found performance comparable to the Girvan and Newman Betweenness algorithm). Overall, we found that our approach outperformed these approaches. For example, on the Chord topology, BotGrep's false positive rate was 0.77%, while false positive rates for the other approaches ranged from 4.24-7.58%. The performance of BotGrep is less on this scaled down 2000-node topology as compared to the earlier

Abilene and CAIDA datasets, because our method of generating the scaled-down 2000 node graph selected the densely connected core of the graph, which is fast-mixing, while on more realistic graphs, it is easier for BotGrep to distinguish the fast-mixing botnet topology from the rest of the non-fast-mixing background graph.

Moreover, we found that run-time was a significant limiting factor in using these alternate approaches. For example, the Girvan-Newman Betweenness Algorithm took 2.5 hours to run on a graph containing 2000 nodes (in all cases, BotGrep runs in under 10.4 seconds on a Core2 Duo 2.83GHz machine with 4GB RAM on a using a single core). While these traditional techniques were not intended to scale to the large data sets we consider here, they may be appropriate for localizing smaller botnets in contained environments (e.g., within a single Honeynet, or the part of a botnet contained within an enterprise network). Since these techniques leverage different features of the inputs, they are synergistic with our approach, and may be used in conjunction with our technique to improve performance.

6 Discussion

As we have demonstrated, analysis of core Internet traffic can be effective at identifying nodes and communication links of structured overlay networks. However, many challenges remain to turn our approach into a full-scale detection mechanism.

Finding malicious P2P traffic: It is easy to see that other forms of P2P activity, such as file sharing networks, will also be identified by our techniques. While there is some benefit to being able to identify such traffic as well, it requires a dramatically different response than botnets and so it is important to distinguish the two. We believe that fundamentally, our mechanisms need to be integrated with detection mechanisms at the edge that identify suspicious behavior. Also, multiple intrusion detection approaches can reinforce each other and provide more accurate results [85, 75, 32]; e.g., misbehaving hosts that follow a similar misuse pattern and at the same time are detected to be part of the same botnet communication graph may be precisely labeled as a botnet, even if each individual misbehavior detection is not sufficient to provide a high-confidence categorization.

Scale and cooperation: Our experiments show our design can scale to large traffic volumes, and in the presence of partial observations. However, several practical issues remain. First, large ISPs tend to use sampled data analysis to monitor their networks. This can miss low-volume control communications used by botnet networks. New counter architectures or programmable monitoring techniques should be used to collect sufficient statistics to run our algorithms [82]. Also, for best results multiple vantage points should data to obtain a better overall perspective. Such data may reveal sensitive business information or violate privacy of the ISPs clients.

Tradeoffs between structure and detection: The communication structure of botnet graphs plays an important role in their delay penalty, and how resilient they are to network failures. At the same time, our results indicate that the structure of the communication graph has some effect on the ability to detect the botnet host from a collection of vantage points. As part of future work, we plan to study the tradeoff between resilience and the ability to avoid detection, and whether there exist fundamentally hard-to-detect botnet structures that are also resilient.

Containing botnets: The ability to quickly localize structured network topologies may assist existing systems that monitor network traffic to quickly localize and contain bot-infected hosts. When botnets are detected in edge networks, the relevant machines are taken offline. However, this may not be always easy with in-core detection; an interesting question is whether in-core filtering or distributed blacklisting can be an effective response strategy when edge cooperation is not possible. Another question we plan to address is whether there exist responses that do not completely disconnect a node but mitigate its potential malicious activities, to be effected when a node is identified as a botnet member, but with a low confidence.

7 Related Work

The increasing criticality of the botnet threat has led to vast amounts of work that attempt to localize them. We can classify this work into host based approaches (Section 7.1) and network based approaches (Section 7.2). We then describe work on botnet detection using graph analysis, the most closely related work to our approach (Section 7.3).

7.1 Host-based approaches

An initial defense against botnets is to prevent systems from being infected in the first place. Anti-virus software, firewalls, filesystem intrusion detection systems, and vulnerability patches help, but completely preventing infection is very difficult task. Malware authors use encryption [87] and polymorphism [81] among other obfuscation techniques [81] to thwart static analysis based approaches used by anti-virus software. In response, dynamic analysis (see Vasudevan et al. [83] and references therein) overcomes obfuscations that prevent static analysis. Malware authors have countered this by employing trigger based behavior such as bot command inputs and logic bombs which exploit analyzer limitations of only observing a single execution path. These limitations are overcome by analyzing multiple execution paths [60, 10], but bots may in turn counter this using schemes relying on the principles of secure triggers [27, 73]. In order to remain invisible to detection, bots can also use a variety of VM (Virtual Machine) based techniques for extra stealth, such as installing virtual machines underneath the existing operating system [51] to prevent access from software running on the target system and being able to identify a virtual analysis environment including VMs and Honeypots [22]. Graph analysis techniques have also been used in host-based approaches. BLINC [49] is a traffic-classification method that uses "Graphlets" to model flow characteristics of a host and touches on the benefit of analyzing the "IP social-network" of a machine. Graph analysis has also been applied to automated malware classification based on function call graphs [38].

7.2 Network based approaches

One approach to detect and neutralize botnets is by analyzing incoming and outgoing host traffic. Several pieces of work isolate bot-infected hosts by detecting the malicious traffic they send, which may be divided into schemes that analyze *attack traffic*, and schemes that analyze *control traffic*.

Attack traffic: For example, network operators may look for sources of denial of service attacks, port scanning, spam, and other unwanted traffic as a likely bot. These works focus on the symptoms caused by the botnets instead of the networks themselves. Several works seek to exploit DNS usage patterns. Dagon et al. [18] studied the propagation rates of malware released at different times by redirecting DNS traffic for bot domain names. Their use of DNS sinkholes is useful in measuring new deployments of a known botnet. However, this approach requires a priori knowledge of botnet domain names and negotiations with DNS operators and hence does not target scaling to networks where a botnet can simply change domain names, have a large pool of C&C IP addresses and change the domain name generation algorithm by remotely patching the bot. Subsequently, Ramachandran et al. [70] use a graph based approach to isolate spam botnets by analyzing the pattern of requests to DNS blacklists maintained by ISPs. They observed that legitimate email servers request blacklist lookups and are looked up by other email servers according to the timing pattern of email arrival, while bot-infected machines are a lot less likely to be looked up by legitimate email servers. However, DNS blacklists and phishing blacklists [74], while initially effective have are becoming increasingly ineffective [69] owing to the agility of the attackers. Much more recently, Villamar et al. [84] applied Bayesian methods to isolate centralized botnets that use fast-flux to counter DNS blacklists, based on the similarity of their DNS traffic with a given corpus of known DNS botnet traces. Further, in order to study bots, Honeypot techniques have been widely used by researchers. Cooke et al. [16] conducted several studies of botnet propagation and dynamics using Honeypots; Barford and Yegneswaran [7] collected bot samples and carried out a detailed study on the source code of several families; finally, Freiling et al. [25] and Rajab et al. [68] carried out measurement studies using Honeypots. Collins et al. [15] present a novel botnet detection approach based on the tendency of unclean networks to contain compromised hosts for extended periods of time and hence acting as a *natural* Honeypot for various botnets. However Honeypot-based approaches are limited by their ability to attract botnets that depend on human action for an infection to take place, an increasingly popular aspect of the attack vector [61].

Control traffic: Another direction of work, is to localize botnets solely based on the control traffic they use to maintain their infrastructures. This line of work can be classified as *traffic-signature* based detection and *statistical traffic analysis* based detection. Techniques in the former category require traffic signatures to be developed for every botnet instance. This approach has been widely used in the detection of IRC-based botnets. Blinkley and Singh[9] combine IRC statistics and TCP work weight to generate signatures; Karasaridis et al. [50] present an algorithm to detect IRC C&C traffic signatures using Netflow records; Rishi [29] uses n-gram analysis to identify botnet nickname patterns. The limitations of these approaches are analogous to the scalability issues faced by host-based detection techniques. In addition, such signatures may not exist for P2P botnets. In the latter category, several works [33, 80, 8, 58] suggest that botnets can be detected by analyzing their flow characteristics. In all these approaches, the authors use a variety of heuristics to characterize the network behavior of various applications and then apply clustering algorithms to isolate botnet traffic. These schemes assume that the statistical properties of bot traffic will be different from *normal* traffic because of synchronized or correlated behavior between bots. While this behavior is currently somewhat characteristic of botnets, it can be easily modified by botnet authors. As such it does not derive from the fundamental property of botnets.

Other works use a hybrid approach such as Bothunter [32] which automates traffic-signature generation by searching for a series of flows that match the infection life-cycle of a bot; BotMiner [31] combines packet statistics of C&C traffic with those of attack traffic and then applies clustering techniques to heuristically isolate botnet flows. TAMD [86] is another method that exploits the spatial and temporal characteristics of botnet traffic that emerges from multiple systems within a vantage point. They aggregate flows based on similarity of flow sizes and host configuration (such as OS platforms) and compare them with a historical baseline to detect infected hosts.

Finally, there are also schemes that combine network- and host-based approaches. The work of Stinson et al. [77] attempts to discriminate between locally-initiated versus remotely-initiated actions by tracking data arriving over the network being used as system call arguments using taint tracking methods. Following a similar approach, Gummadi et al. [35] whitelist application traffic by identifying and attesting human-generated traffic from a host which allows an application server to selectively respond to service requests. Finally, John et al. [45] present a technique to defend against spam botnets by automating the generation of spam feeds by directing an incoming spam feed into a Honeynet, then downloading bots spreading through those messages and then using the outbound spam generated to create a better feed. While all the above are interesting approaches they again deal with the side-effects of botnets instead of tackling the problem in its entirety in a scalable manner.

7.3 Graph-based approaches

Several works [14, 40, 39, 89, 43] have previously applied graph analysis to detect botnets. The technique of Collins and Reiter [14] detects anomalies induced in a graph of protocol specific flows by a botnet control traffic. They suggest that a botnet can be detected based on the observation that an attacker will increase the number of connected graph components due to a sudden growth of edges between unlikely neighboring nodes. While it depends on being able to accurately model valid network growth, this is a powerful approach because it avoids depending on protocol semantics or packet statistics. However this work only makes minimal use of spatial relationship information. Additionally, the need for historical record keeping makes it challenging in scenarios where the victim network is already infected when it seeks help and hasn't stored past traffic data, while our scheme can be used to detect pre-existing botnets as well. Illiofotou et al. [40, 39] also exploit dynamicity of traffic graphs to classify network flows in order to detect P2P networks. It uses static (spatial) and dynamic (temporal) metrics centered on node and edge level metrics in addition to the largest-connected-component-size as a graph level metric. Our scheme however starts from first principles (searching for expanders) and uses the full extent of spatial relationships to discover P2P graphs including the joint degree distribution and the joint-joint degree distribution and so on.

Of the many botnet detection and mitigation techniques mentioned above, most are rather ad-hoc and only apply to specific scenarios of centralized botnets such as IRC/HTTP/FTP botnets, although studies [30] indicate that the centralized model is giving way to the P2P model. Of the techniques that do address P2P botnets, detection is again dependent on specifics regarding control traffic ports, network behavior of certain types of botnets, reverse engineering botnet protocols and so on, which limits the applicability of these techniques. Generic schemes such as BotMiner [31] and TAMD [86] using behavior based clustering are better off but need access to extensive flow information which can have legal and privacy implications. It is also important to think about possible defenses that botmasters can apply, the cost of these defenses and and how they might affect the efficiency of detection. Shear and Nicol [72, 64] describe schemes to mask the statistical characteristics of real traffic by embedding it in synthetic, encrypted, cover traffic. The adoption of such schemes will only require minimal alterations to existing botnet architectures but can effectively defend against detection schemes that depend on packet level statistics including BotMiner and TAMD.

8 Conclusion

The ability to localize structured communication graphs within network traffic could be a significant step forward in identifying bots or traffic that violates network policy. As a first step in this direction, we proposed BotGrep, an inference algorithm that identifies botnet hosts and links within network traffic traces. BotGrep works by searching for structured topologies, and separating them from the background communication graph. We give an architecture for a BotGrep network deployment as well as a privacy-preserving extension to simplify deployment across networks. While our techniques do not achieve perfect accuracy, they achieve a low enough false positive rate to be of substantial use, especially when combined with complementary techniques. There are several avenues of future work. First, performance of our approach may be improved by leveraging temporal information (observing how parts of the the communication graph change over time) to assist in separating out the botnet graph. In addition, it may be desirable to distinguish other peer-to-peer structure from other Internet background traffic, perhaps by observing more fine-grained properties of communication patterns. Finally, we do not attempt to address the challenging problem of botnet *response*. Future work may leverage our inferred botnet topologies by dropping crucial links to partition the botnet, based on the structure of the botnet graph.

References

[1] Botlab: A real-time botnet monitoring platform. `botlab.cs.washington.edu`.

[2] Spamhaus. `www.spamhaus.org`.

[3] The Cooperative Association for Internet Data Analysis. `http://www.caida.org/`.

[4] *First Workshop on Hot Topics in Understanding Botnets (HotBots)*, 2007.

[5] *Proceedings of the Network and Distributed System Security Symposium, NDSS 2008, San Diego, California, USA, 10th February - 13th February 2008*. The Internet Society, 2008.

[6] M. Bailey, E. Cooke, F. Jahanian, N. Provos, K. Rosaen, and D. Watson. Data Reduction for the Scalable Automated Analysis of Distributed Darknet Traffic. In *Proceedings of IMC*, 2005.

[7] P. Barford and V. Yegneswaran. *An Inside Look at Botnets*, volume 27 of *Advanced in Information Security*, chapter 8, pages 171–192. Springer, 2006.

[8] A. Barsamian. Network characterization for botnet detection using statistical-behavioral methods. Masters thesis, Thayer School of Engineering, Dartmouth College, USA, June 2009.

[9] J. R. Binkley and S. Singh. An algorithm for anomaly-based botnet detection. In *SRUTI'06: Proceedings of the 2nd conference on Steps to Reducing Unwanted Traffic on the Internet*, pages 7–7, Berkeley, CA, USA, 2006. USENIX Association.

[10] D. Brumley, C. Hartwig, Z. Liang, J. Newsome, D. X. Song, and H. Yin. Automatically identifying trigger-based behavior in malware. In Lee et al. [55], pages 65–88.

[11] J. Camenisch and G. Zaverucha. Private intersection of certified sets. In *Financial Cryptography and Data Security*, page 127. Springer, 2009.

[12] D. Chaum. The dining cryptographers problem: unconditional sender and recipient untraceability. *J. Cryptol.*, 1(1):65–75, 1988.

[13] A. Clauset, M. E. J. Newman, and C. Moore. Finding community structure in very large networks. *Physical Review E*, 70(6), 2004.

[14] M. P. Collins and M. K. Reiter. Hit-list worm detection and bot identification in large networks using protocol graphs. In C. Krügel, R. Lippmann, and A. Clark, editors, *RAID*, volume 4637 of *Lecture Notes in Computer Science*, pages 276–295. Springer, 2007.

[15] M. P. Collins, T. J. Shimeall, S. Faber, J. Janies, R. Weaver, M. De Shon, and J. Kadane. Using uncleanliness to predict future botnet addresses. In *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 93–104, New York, NY, USA, 2007. ACM.

[16] E. Cooke and F. Jahanian. The zombie roundup: Understanding, detecting, and disrupting botnets. In *Steps to Reducing Unwanted Traffic on the Internet Workshop*, 2005.

[17] E. D. Cristofaro and G. Tsudik. Practical private set intersection protocols. Cryptology ePrint Archive, Report 2009/491, 2009. `http://eprint.iacr.org/`.

[18] D. Dagon, C. Zou, and W. Lee. Modeling botnet propagation using time zones. In *Network and Distributed Systems Security Symposium*, 2006.

[19] I. Damgard and M. Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *Public Key Cryptography*. Springer, 2001.

[20] G. Danezis and P. Mittal. Sybilinfer: Detecting sybil nodes using social networks. In *NDSS*, 2009.

[21] S. Fortunato, V. Latora, and M. Marchiori. Method to find community structures based on information centrality. *Physical Review E*, 70(5), 2004.

[22] J. Franklin, M. Luk, J. M. McCune, A. Seshadri, A. Perrig, and L. van Doorn. Towards sound detection of virtual machines. In Lee et al. [55], pages 89–116.

[23] J. Franklin, V. Paxson, A. Perrig, and S. Savage. An inquiry into the nature and causes of the wealth of internet miscreants. In *ACM conference on Computer and communications security*, pages 375–388, New York, NY, USA, 2007. ACM.

[24] M. Freedman, K. Nissim, B. Pinkas, et al. Efficient private matching and set intersection. In *EUROCRYPT*, pages 1–19. Springer, 2004.

[25] F. C. Freiling, T. Hoz, and G. Wichereski. Botnet tracking: Exploring a root-cause methodology to prevent distributed denial-of-service attacks. In *European Symposium on Research in Computer Security*, 2005.

[26] T. M. J. Fruchterman, Edward, and E. M. Reingold. Graph drawing by force-directed placement. *Softw. Pract. Exper.*, 1991.

[27] A. Futoransky, E. Kargieman, C. Sarraute, and A. Waissbein. Foundations and applications for secure triggers. *ACM Trans. Inf. Syst. Secur.*, 9(1):94–112, 2006.

[28] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences of the United States of America*, 99(12), 2002.

[29] J. Goebel and T. Holz. Rishi: Identify bot contaminated hosts by IRC nickname evaluation. In *Hot Topics in Understanding Botnets* [4].

[30] J. B. Grizzard, V. Sharma, C. Nunnery, B. B. Kang, and D. Dagon. Peer-to-peer botnets: Overview and case study. In *Hot Topics in Understanding Botnets* [4].

[31] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of the 17th USENIX Security Symposium (Security'08)*, 2008.

[32] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. Bothunter: Detecting malware infection through IDS-driven dialog correlation. In *USENIX Security Symposium*, 2007.

[33] 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 (NDSS'08)* [5].

[34] K. P. Gummadi, R. Gummadi, S. D. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of DHT routing geometry on resilience and proximity. In *Proceedings of ACM SIGCOMM 2003*, Aug. 2003.

[35] R. Gummadi, H. Balakrishnan, P. Maniatis, and S. Ratnasamy. Not-a-Bot (NAB): Improving Service Availability in the Face of Botnet Attacks. In *NSDI 2009*, Boston, MA, April 2009.

[36] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, April 1970.

[37] C. Hazay and Y. Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *Theory of Cryptography Conference*. Springer, 2008.

[38] X. Hu, T. cker Chiueh, and K. G. Shin. Large-scale malware indexing using function-call graphs. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, 2009.

[39] M. Iliofotou, M. Faloutsos, and M. Mitzenmacher. Exploiting dynamicity in graph-based traffic analysis: Techniques and applications. In *ACM CoNext*, 2009.

[40] M. Iliofotou, P. Pappu, M. Faloutsos, M. Mitzenmacher, G. Varghese, and H. Kim. Graption: Automated detection of P2P applications using traffic dispersion graphs (TDGs). In *UC Riverside Technical Report, CS-2008-06080*, 2008.

[41] C. S. Inc. Cisco IOS Netflow. `http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html`.

[42] S. Jarecki and X. Liu. Efficient Oblivious Pseudorandom Function with Applications to Adaptive OT and Secure Computation of Set Intersection. In *Theory of Cryptography Conference*, pages 577–594. Springer, 2009.

[43] M. Jelasity and V. Bilicki. Towards automated detection of peer-to-peer botnets: On the limits of local approaches. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2009.

[44] M. Jelasity and V. Billicki. Towards automated detection of peer-to-peer botnets: On the limits of local approaches. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2009.

[45] J. P. John, A. Moshchuk, S. D. Gribble, and A. Krishnamurthy. Studying spamming botnets using botlab. In *NSDI'09: Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, pages 291–306, Berkeley, CA, USA, 2009. USENIX Association.

[46] M. Kaashoek and D. Karger. Koorde: A simple degree-optimal distributed hash table. In *Proceedings of the 1st International Peer To Peer Systems Workshop*, 2003.

[47] C. Kanich, C. Kreibich, K. Levchenko, B. Enright, G. Voelker, V. Paxson, and S. Savage. Spamalytics: An empirical analysis of spam marketing conversion. In *CCS*, Oct. 2008.

[48] R. Kannan, S. Vempala, and A. Vetta. On clusterings: Good, bad and spectral. *J. ACM*, 51(3):497–515, 2004.

[49] T. Karagiannis, K. Papagiannaki, and M. Faloutsos. BLINC: multilevel traffic classification in the dark. In *ACM SIGCOMM*, 2005.

[50] A. Karasaridis, B. Rexroad, and D. Hoeflin. Wide-scale botnet detection and characterization. In *Hot Topics in Understanding Botnets* [4].

[51] S. T. King, P. M. Chen, Y.-M. Wang, C. Verbowski, H. J. Wang, and J. R. Lorch. Subvirt: Implementing malware with virtual machines. *Security and Privacy, IEEE Symposium on*, 0:314–327, 2006.

[52] L. Kissner and D. Song. Privacy-preserving set operations. In *CRYPTO*, volume 3621 of *LNCS*, pages 241–257. Springer, 2005.

[53] S. Kullback and R. A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22:49–86, 1951.

[54] V. Latora and M. Marchiori. Economic small-world behavior in weighted networks. *The European Physical Journal B - Condensed Matter*, 32(2), 2002.

[55] W. Lee, C. Wang, and D. Dagon, editors. *Botnet Detection: Countering the Largest Security Threat*, volume 36 of *Advances in Information Security*. Springer, 2008.

[56] T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, 1999.

[57] S. Lloyd. Least squares quantization in PCM. *Information Theory, IEEE Transactions on*, 28(2):129–137, 1982.

[58] W. Lu, M. Tavallaee, and A. A. Ghorbani. Automatic discovery of botnet communities on large-scale communication networks. In *ASIACCS '09: Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, pages 1–10, New York, NY, USA, 2009. ACM.

[59] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Proceedings of the 1st International Peer To Peer Systems Workshop*, 2002.

[60] A. Moser, C. Kruegel, and E. Kirda. Exploring multiple execution paths for malware analysis. In *SP '07: Proceedings of the 2007 IEEE Symposium on Security and Privacy*, pages 231–245, Washington, DC, USA, 2007. IEEE Computer Society.

[61] S. Nagaraja and R. Anderson. The snooping dragon: social-malware surveillance of the tibetan movement. Technical Report UCAM-CL-TR-746, University of Cambridge, 2009.

[62] M. E. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys Rev E Stat Nonlin Soft Matter Phys*, 69(2 Pt 2), 2004.

[63] M. E. J. Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical Review E*, 74(3), 2006.

[64] D. M. Nicol and N. Schear. Models of privacy preserving traffic tunneling. *Simulation*, 85(9):589–607, 2009.

[65] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Eurocrypt*, pages 223–238. Springer, 1999.

[66] P. Porras, H. Saidi, and V. Yegneswaran. A multi-perspective analysis of the Storm (Peacomm) worm. In *SRI Technical Report 10-01*, 2007.

[67] P. Porras, H. Saidi, and V. Yegneswaran. A foray into conficker's logic and rendezvous points. In *2nd Usenix Workshop on Large-Scale Exploits and Emergent Threats (LEET '09)*, 2009.

[68] M. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. A multifaceted approach to understanding the botnet phenomenon. In *Internet Measurement Conference*, 2006.

[69] A. Ramachandran, D. Dagon, and N. Feamster. Can dns-based blacklists keep up with bots? In *CEAS*, 2006.

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

[71] D. Randall. Rapidly mixing markov chains with applications in computer science and physics. *Computing in Science and Engineering*, 8(2):30–41, 2006.

[72] N. Schear and D. M. Nicol. Performance analysis of real traffic carried with encrypted cover flows. In *PADS '08: Proceedings of the 22nd Workshop on Principles of Advanced and Distributed Simulation*, pages 80–87, Washington, DC, USA, 2008. IEEE Computer Society.

[73] M. I. Sharif, A. Lanzi, J. T. Giffin, and W. Lee. Impeding malware analysis using conditional code obfuscation. In *NDSS* [5].

[74] S. Sheng, B. Wardman, G. Warner, L. F. Cranor, J. Hong, and C. Zhang. An empirical analysis of phishing blacklists. In *CEAS*, 2009.

[75] E. Spafford and D. Zamboni. Intrusion detection using autonomous agents. *Computer Networks*, 34(4):547–570, 2000.

[76] L. Spitzner. The Honeynet Project: trapping the hackers. *Security & Privacy Magazine, IEEE*, 1(2):15–23, 2003.

[77] E. Stinson and J. C. Mitchell. Characterizing bots' remote control behavior. In Lee et al. [55], pages 45–64.

[78] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of ACM SIGCOMM*, Aug. 2001.

[79] S. Stover, D. Dittrich, J. Hernandez, and S. Dietrich. Analysis of the Storm and Nugache trojans: P2P is here. *;login*, 32(6), Dec. 2007.

[80] W. T. Strayer, D. E. Lapsley, R. Walsh, and C. Livadas. Botnet detection based on network behavior. In Lee et al. [55], pages 1–24.

[81] P. Szor. *The Art of Computer Virus Research and Defense*. Addison-Wesley Professional, 2005.

[82] G. Varghese and C. Estan. The measurement manifesto. In *HotNets-II*, 2003.

[83] A. Vasudevan and R. Yerraballi. Cobra: Fine-grained malware analysis using stealth localized-executions. In *In Proceedings of 2006 IEEE Symposium on Security and Privacy (Oakland.06*, 2006.

[84] R. Villamarín-Salomón and J. C. Brustoloni. Bayesian bot detection based on dns traffic similarity. In *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing*, pages 2035–2041, New York, NY, USA, 2009. ACM.

[85] G. White, E. Fisch, and U. Pooch. Cooperating security managers: a peer-based intrusion detection system. *IEEE network*, 10(1):20–23, 1996.

[86] T.-F. Yen and M. K. Reiter. Traffic aggregation for malware detection. In *DIMVA '08: Proceedings of the 5th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 207–227, Berlin, Heidelberg, 2008. Springer-Verlag.

[87] A. Young and M. Yung. Cryptovirology: Extortion-based security threats and countermeasures. In *SP '96: Proceedings of the 1996 IEEE Symposium on Security and Privacy*, page 129, Washington, DC, USA, 1996. IEEE Computer Society.

[88] Q. Zhao, J. Xu, and Z. Liu. Design of a novel statistics counter architecture with optimal space and time efficiency. In *ACM SIGMETRICS*, June 2006.

[89] Y. Zhao, Y. Xie, F. Yu, Q. Ke, Y. Yu, Y. Chen, and E. Gillum. Botgraph: Large scale spamming botnet detection. In *NSDI*, 2009.