

# Gangs of the Internet: Towards Automatic Discovery of Peer-to-Peer Communities

Liyun Li\*  
LinkedIn  
liyli@linkedin.com

Suhas Mathur  
AT&T Security Research Center  
suhas@att.com

Baris Coskun  
AT&T Security Research Center  
baris@att.com

**Abstract**—Internet Service Providers and network administrators currently lack effective means for discovering and tracking peer-to-peer (P2P) applications on their networks. This ability would be very useful in various ways such as enforcing security policies on the use of P2P applications (e.g. banning file-sharing networks such as Bit Torrent), mitigating malicious P2P networks (i.e. botnets), or allocating network resources appropriately to improve network performance. To provide this ability, in this work we propose a method to discover P2P networks (both benign and malicious) from network flow records captured at the boundary of a tier-1 Internet backbone provider. The basic idea is that flows belonging to P2P applications can be modeled as observations from a mixed membership statistical model, with P2P applications acting as latent variables. Hence the communication patterns of hosts (who-talks-to-whom), as measured at the edge of a large network, can be decomposed into constituent application-layer P2P communities without any human effort in selecting specific features. This allows for automatic identification and isolation of P2P communities of interest, including those that take deliberate measures to remain hidden, as well as new or evolving ones such as P2P Botnets. In large scale experiments on flow records from a portion of IPv4 space of size  $/8$ , we demonstrate that the proposed method is able to detect a number of well known P2P networks, as well as a few evolving malicious P2P botnets.

## I. INTRODUCTION

P2P networks in the Internet represent a substantial fraction of total Internet traffic volume [1] (21-33%), [2] (30-70%), [3] (15-20%). Yet, relatively little can be observed about P2P networks - how they form, how they evolve, and what techniques some of them use to remain hidden. The reason is that, as opposed to client-server systems, tracking P2P systems is extremely hard. More specifically, in a typical client-server system, many client nodes communicate with few server nodes which are located using Domain Name System (DNS). Therefore, one can easily track a client-server system at the network level by monitoring few servers and their associated domain names. Whereas in P2P systems each node can potentially communicate with any other node, leading to a messier communication pattern and making impossible to gain insight about the entire system by monitoring only a few of its components.

The resource consumption that P2P systems present to Internet service providers and the lack of measurability they come with

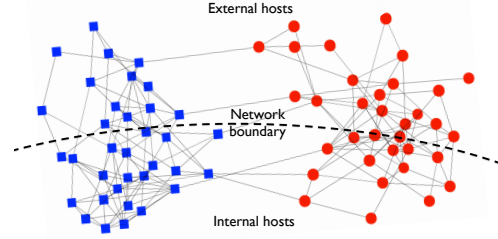


Fig. 1. Two weakly overlapping P2P communities: A (■) on the left and B (●) on the right. Only edges crossing the backbone boundary (dashed line) correspond to flows observable by us.

are at apparent loggerheads. Clearly, the growing importance of P2P systems [2] and their resource footprint warrant better methods for their measurement and tracking.

Apart from legitimate uses, P2P architectures have reportedly been used by Botnets [4], [5], [6], [7], [8] due to the resilience they provide against tracking and takedown attempts. Botnets are networks of compromised hosts and are responsible for a significant portion of malicious activities on the Internet today (i.e. spam, phishing, distributed denial of service (DDoS) attacks). These botnets use a P2P architecture to distribute commands, patches and other updates from the botmaster to compromised hosts (bots). Botmasters try their best to conceal such malicious P2P activities and therefore it is important to have better tools to discover P2P communities in the Internet in order to mitigate such P2P botnets. In fact there are few botnet detection techniques specifically focused on detecting P2P botnets [9][10]. However, despite being successful in separating out P2P botnet traffic embedded into background network traffic, neither technique is capable of inferring multiple simultaneously operating P2P communities.

Finally, organizations such as enterprises and schools often find it impossible to prevent the use of certain P2P applications on the hosts connected to their internal networks. This is due in part to the fact that a number of P2P applications take measures to evade detection by randomizing their traffic, so that simple signatures cannot be developed to accurately detect their traffic. An example is Skype, which randomizes the port number on which it operates [14]. If there were an automated mechanism to categorically ascertain that a given network flow crossing an enterprise network boundary belongs to a known P2P application that must be blocked or rate-limited as per the policy enforced by an organization's network, then it would go a long way in simplifying policy enforcement.

\*This work was done when Liyun Li was with Polytechnic Institute of NYU and AT&T Security Research Center.

TABLE I

SUMMARY OF RELATED WORK. <sup>1</sup>WHETHER MULTIPLE P2P COMMUNITIES CAN BE DISCOVERED SIMULTANEOUSLY (WE DO NOT CONSIDER EFFORTS MADE ON THE ENTERPRISE NETWORK SCALE TO BE ATTEMPTING DETECTION OF MULTIPLE/OVERLAPPING COMMUNITIES). <sup>2</sup>WHETHER MALICIOUS P2P COMMUNITIES WERE DISCOVERED IN THE WILD. <sup>3</sup>WHETHER OVERLAPPING P2P COMMUNITIES CAN BE DISCOVERED.

Prior work	Main idea	Multiple? <sup>1</sup>	Wild? <sup>2</sup>	Overlapping? <sup>3</sup>	Scale
BotGrep [9]	P2P traffic mixes faster in the communication graph than centralized traffic	✗	✗	✗	Backbone
BotMiner [11]	Clustering using hand crafted features. Not specific to P2P Botnet detection	N/A	✗	N/A	Enterprise
Friends of an Enemy [12]	P2P Botnet detection in an enterprise network by partitioning of a mutual-contacts graph	N/A	✗	N/A	Enterprise
Statistical fingerprints [10]	Clustering based on statistical flow features	N/A	✗	N/A	Enterprise
BLINC [13]	Feature based low classification followed by detection of approximate, non-overlapping cliques on communication graph. Requires access to payload data to separate out a class of P2P traffic.	✗	✗	✗	Backbone
Our work	Treat P2P communication graph as statistical mixture model. Infer best fit community structure.	✓	✓	✓	Backbone

In this paper, we describe a mechanism to detect and separate out individual P2P application communities of hosts in the Internet using only flow-level observations made at the peering points of a tier-1 backbone network. Our approach is based on approximate posterior inference on a probabilistic generative model that models P2P activity observed at the edge routers. While the proposed inference method is similar to existing Bayesian-inference based community detection methods, to the best of our knowledge, our work is the first to automatically decompose network flow records into constituent P2P communities, and to automatically detect malicious P2P communities *in the wild*. Our contributions in this paper are as follows:

- 1) We formulate the P2P community detection problem as a probabilistic model for partitioning Internet hosts into (potentially overlapping) P2P application communities using only “*who-talks-to-whom*” information.
- 2) We demonstrate that our method discovers real malicious P2P communities in the wild.
- 3) We validate our approach by identifying a number of well-known P2P communities (BitTorrent, Gnutella, etc.) in real-world data captured at the peering boundary of a tier-1 ISP.
- 4) We quantify the ability of our approach to infer P2P application communities using a combination of real-world network-flow data and synthetically embedded P2P communities.

In Section II we discuss related work, in Section III we describe the real-world dataset that is the starting point of our analysis, and our statistical model for inferring P2P communities. Then, in Section IV, we validate our approach on our dataset by attempting to discover well known P2P networks. We also describe our experience in discovering and tracking three communities of hosts that appear to be P2P botnets. In Section V we formally evaluate the performance of our approach against perfectly known ground-truth by synthetically creating P2P communities with various types of random topologies, embedding them into a real network flow dataset and attempting to discover them. Finally, we close with a discussion of limitations of our approach in Section VI and conclude the paper in Section VII.

## II. RELATED WORK

Research efforts that are most closely related to our work are summarized in Table I. These have to do with detecting

communities of P2P hosts and P2P Botnets. As summarized in Table I, we are the first to be able to decompose network traffic into multiple constituent P2P communities, at the scale of an Internet backbone, and the first to report on the detection of malicious communities in the wild. There have also been other efforts to detect botnets in enterprise networks [15], [16], [17], [18]. All these methods focus on detecting centralized botnets, other than [17] which can be used to detect P2P botnets as it focuses on initial infections and is agnostic to the C&C architecture. [19] aims at P2P botnet detection, using dynamic analysis of malware binaries and active network probing. Notice that all these methods focus on detecting individual bots in enterprise networks, while our goal is to get a holistic picture about all P2P communities at the backbone level.

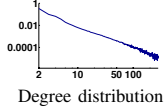
The approach we take to inferring P2P communities is one of several *community detection* methods described in literature. Detecting communities in complex graphs has been studied in various other contexts such as social networks, web graphs and biological protein-interaction networks (see [20] for a survey). Bayesian methods are especially attractive for detection of overlapping communities because they infer the probability that an entity belongs to a given community, and also provide a way to assign a confidence score to whether an inferred collection of hosts is really a meaningful community.

The ability to assign a confidence score of inference-based methods is especially useful in our context when monitoring the development of new communities, or malicious communities. That is, if we are very confident that a certain new inferred community is really a valid clustering of hosts, and some fraction of it appears to be malicious (say, via a blacklist), then we can be very confident about the implied maliciousness of other hosts in the community.

## III. DETECTION ALGORITHM

In this section, we describe our approach to infer potentially overlapping clusters from datasets of <source-IP, destination-IP> pairs (i.e. who-talks-to-whom). The reason why we only use who-talks-to-whom information is two-fold. First we do not focus on any specific P2P application, therefore we avoid employing any application specific signatures, such as specific port numbers, network topology, payload etc. Second, although it has been shown that clustering analysis based on certain network flow features helps identify members of some P2P

TABLE II  
DESCRIPTION OF 1 HOUR OF FLOW DATA FROM A /8 NETWORK ( $2^{24}$   
POTENTIAL INTERNAL IPS) AFTER FILTERING OUT NON-P2P FLOWS AND  
DEGREE-FILTERING.

# of Internal hosts	148,902	
# of External hosts	1,946,879	
# of Flows	47,908,665	
# of edges	8,302,666	

communities [10], in principle members of a P2P community (especially a botnet) can slightly perturb their flows to randomize flow features [21], thereby evading clustering analyses. Unlike flow features, however, who-talks-to-whom information is the very essence of a P2P community since peers *must* communicate with each other to form a functioning P2P network.

Before we explain the further details of our method, we first present our measurement setup and our dataset.

#### A. Dataset

**Measurement Setup.** We make measurements at the peering routers of a large tier-1 US backbone provider. The boundary of this network (depicted as a dashed line in Figure 1) consisting of routers that peer to other ISPs, divides internet-connected hosts into *internal* and *external* hosts. Our setup allows us to record any network flow that crosses a peering router, irrespective of the direction of the flow. These network flow records comprise source and destination IP addresses, port numbers, protocol, duration, the number of packets and the number of bytes. However, as discussed before, we do not use any of these quantities other than IP addresses (i.e. who-talks-to-whom) in inferring P2P communities. We only make use of these flow features in validating some of the inferred communities (Section V).

Notice that, we are not able to capture network flows that have both end-points either completely inside the network, or completely outside the network. In other words, the observed communication graph is bi-partite. Also notice that, we use the term *P2P community* to refer to a collection of hosts that participate in a common P2P application (e.g. Skype) within a given time interval (e.g. 1 hour). With this definition, each community belongs to a specific application, but all users participating in that application may not necessarily be grouped into a single community (e.g. there may be 3 disjoint Skype communities). Finally, each host can participate in more than one P2P application (e.g. a host using Skype and BitTorrent within one hour).

**Filtering out Non-P2P flows:** Our method requires a set of P2P flows (Source IP, Dest IP pairs) as input. To remove non-P2P flows from our dataset, we employ a local database of DNS records. This database keeps track of the IP address that each domain name resolved to, for all DNS requests and responses that cross the backbone boundary within the most recent 1-day period. We label a flow as P2P only when both its source IP and destination IP are *not* associated with any domain name in this database. The intuition behind this is that, in the client-server model, servers are usually addressed by their domain names, whereas in P2P networks, peers find

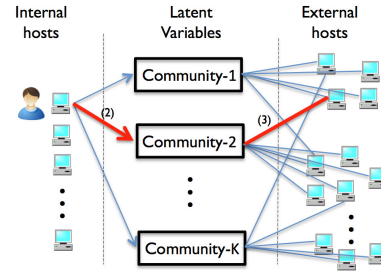


Fig. 2. The probabilistic generative process used to explain the dataset of (source-IP, destination-IP) pairs. The numbers in parenthesis correspond to the numbering in Figure 3 and the steps described in Section III.

each other using their IP addresses directly. A similar DNS-based filtering approach is used in [10]. Notice that, being assigned a domain name is not sufficient for an IP address to be on this database. The database only contains domain name/IP address pairs which have been actually queried on the DNS infrastructure. Therefore, client hosts which are assigned domain names by their ISPs or institutions (e.g. dynamic DNS) are not included in this dataset with high probability since their domain names are rarely queried. Therefore such hosts still appear in our P2P analysis if they are engaged in any P2P communication.

**Degree Filtering:** After remove non-P2P flows, we treat the remaining flows as a graph in which vertices are IPs and edges indicate communication between them. We observe that, the degree of nodes in this graph is highly diverse: 1) there are high degree nodes which almost talk to every other node; 2) there are pair of nodes which only talk to each other only and no other node. For our algorithm, flows of these two kinds of nodes provide very little information in inference. In the former case, high degree nodes are associated with many communities and therefore do not provide distinguishing information between different communities. In the latter case, on the other hand, isolated pairs of nodes do not provide information to link other nodes together in a community since each one has connection only to one another. We therefore filter such flows out., The remaining flows are fed as input into the community detection algorithm. Table II shows a summary of this dataset.

#### B. P2P community discovery via Latent Dirichlet Allocation

There are several challenges involved in partitioning hosts into (potentially overlapping) P2P communities.

**Limited visibility.** First, it is not possible to observe network flows between hosts on the same side of the network boundary.

**Overlap.** Second, a given host can participate in more than one P2P application, therefore it is important to allow for overlapping communities.

**IP churn.** Finally, we can only identify hosts via their IP addresses, which can change with time, and therefore, we must work with a dataset that is short enough in duration to avoid dynamically changing IP addresses from becoming a major problem.

To address these challenges, we employ an unsupervised learning approach based on a probabilistic generative model

called Latent Dirichlet Allocation (LDA) [22]. LDA is a probabilistic clustering model that has been successfully used in the natural language processing community for the task of modeling topics in text corpora. The goal in topic modeling is to discover a latent structure in a collection of text documents in terms of hidden 'topics', i.e. collections of terms that tend to co-occur. Our problem of decomposing network flows into constituent P2P communities is of a similar flavor: Just as words that co-occur often in documents probably belong to the same topic, hosts that share many of the hosts communicating with them, are likely to belong to a common P2P community. The membership of hosts in the P2P communities are the latent variable we wish to infer.

If we were to treat each host as a node in a graph and place an edge between two entities if they are found to communicate, then densely connected regions (nodes and edges) of the graph are likely to belong to the same P2P application. This is because there are likely to be many more links between members of a single P2P application community in the communication graph, than *between* the two different application communities.

Imagine that each network flow between an internal host and an external host was produced by an imaginary two-step process (see Figure 2): each internal host first picks, at random, a P2P application community to participate in, out of  $K$  possible communities, and then, given the chosen community, an external host that belongs to that community is picked, also at random. This two step process is repeated for each flow in the observed dataset, with the random draws made independent of other draws. Each of the two random draws are made according to specific (as yet unknown) distributions – the first, a distribution over communities, and the second, over external hosts, given a community. These distributions can be treated as tunable parameters of a model. Using Bayes' rule to answer the question: "What setting of the parameters best explains the observed communication patterns?" gives us the distributions, from which we can infer a grouping of hosts into communities. Since each community is simply a probability distribution over external hosts, the same host can appear in more than one community. The same is true for internal hosts. In the remainder of this section we describe the LDA model in detail, as applied to our problem.

**The LDA model in detail.** Consider a data set  $\mathcal{D}$ , consisting of {source IP, Destination IP} pairs in a given time interval. If we think of  $\mathcal{D}$  as a graph  $\mathcal{G}$  where vertices represent hosts, and undirected edges represent flows between pairs of hosts, then  $\mathcal{G}$  must be a bipartite graph because of our measurement setup (Section III-A). Each IP in  $\mathcal{D}$  can be a member of one or more P2P communities. We assume there are  $K$  P2P communities in all. We first define two families of multinomial distributions (see Table III for summary of notations):

$\theta$ : For each internal IP  $i$ , consider a  $K$ -dimensional multinomial probability distribution  $\theta_i$ , where  $i = 1, \dots, N_{int}$ , and  $N_{int}$  is the number of internal IP addresses in  $\mathcal{D}$ . The  $n^{th}$  element of  $\theta_i$  represents the extent to which internal IP  $i$  belongs to the  $n^{th}$  P2P

TABLE III  
SUMMARY OF NOTATION

Symbol	Meaning
$\mathcal{D}$	Dataset of {Source IP, Dest IP} pairs
$\mathcal{D}_i$	Set of all flows involving internal host $i$
$N_{int}$	Number of internal IPs
$N_{ext}$	Number of external IPs
$K$	Approximate number of P2P communities
$\theta_i$	Multinomial of dimension $K$
$\beta_j$	Multinomial of dimension $N_{ext}$
$C_{i,n}$	Community of the $n^{th}$ flow of internal host $i$
$H_{i,n}$	External IP involved in $i^{th}$ internal host's $n^{th}$ flow
$\alpha$	Dirichlet hyperparameter for producing prior $\theta_i$ s
$\eta$	Dirichlet hyperparameter for producing prior $\beta_j$ s
$\gamma$	Fraction of hosts that are internal

community.

$\beta$ : For each community  $C_j$ ,  $j = 1, \dots, K$ , consider a multinomial probability distribution  $\beta_j$ ,  $j = 1 \dots K$  over all the external IPs in  $\mathcal{D}$ . The dimensionality of each  $\beta_j$  is  $N_{ext}$ , the number of external IP addresses. The  $m^{th}$  element of  $\beta_j$  represents the extent to which the  $m^{th}$  external IP participates in the  $j^{th}$  P2P community.

We first collect all the flows in  $\mathcal{D}$  involving internal IP  $i$  into a dataset  $\mathcal{D}_i$ . Let  $|\mathcal{D}_i|$  denote the number of flows in  $\mathcal{D}_i$ . We ignore the order in which these flows occur. The behavior of each internal host  $i$  in  $\mathcal{D}$  can be described via a probabilistic generative process as follows (see Figure 2 for an illustration of the generative process and Figure 3 for the probabilistic graphical model that represents the generative process described in the steps below. The numbers within parentheses in Figures 2 and 3 correspond to the numbering of the 3 steps below):

- 1) A Dirichlet distribution (a distribution over distributions) is sampled to randomly pick a multinomial for each of the  $\theta_i$  and  $\beta_j$  distributions. In Figure 3,  $\alpha$  and  $\eta$  are parameters of the two separate Dirichlet distributions. We use  $\alpha = 0.3$  and  $\eta = 0.01$  based on suggestions in [23], [24].
- 2) For each flow in  $\mathcal{D}_i$ , first randomly pick one of the  $K$  communities, by sampling the multinomial  $\theta_i$ . Let the chosen community be denoted by  $C_{i,n}$ .
- 3) Then given  $C_{i,n} = k$  was picked, ( $k \in \{1, \dots, K\}$ ), pick an external IP by sampling from  $\beta_k$ . Let the external IP picked be denoted by  $H_{i,n}$ .

The last two steps above are repeated for each of the  $|\mathcal{D}_i|$  flows in  $\mathcal{D}_i$ , and the set of three steps above is repeated for each internal host in  $\mathcal{D}$ . Given prior distributions  $\theta_i$ ,  $i = 1, \dots, N_{int}$  and  $\beta_j$ ,  $j = 1, \dots, K$ , and this probabilistic generative process (i.e., how observed data  $\mathcal{D}$  is linked to a given set of prior distributions  $\theta_i$  and  $\beta_j$ ), our objective is to infer posteriors  $\theta_i, \beta_j | \mathcal{D}$ . Exact computation of the posteriors on  $\theta_i$  and  $\beta_j$  is intractable. Standard techniques are available to perform approximate posterior inference on probabilistic graphical models (e.g. Belief propagation, Markov Chain Monte Carlo algorithms such as Gibbs sampling, Variational Methods, etc.). We select collapsed Gibbs sampling [24] as our approximate inference algorithm as it can be paralleled over a compute cluster [23]. The Gibbs sampling algorithm provides us with



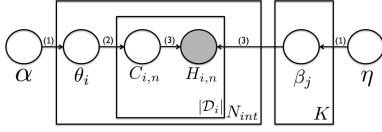


Fig. 3. Probabilistic graphical model for LDA (shaded nodes are observed)

a matrix of size  $N_{ext} \times K$  in which the  $(i, j)^{th}$  element is the estimate of the number of flows that external host  $i$  has been involved in while participating in community  $j$ . The columns of this matrices can be normalized to sum up to 1 to yield the posterior distributions  $\beta_i, i = 1, \dots, K$ . Since internal hosts are connected to external hosts by flows, and we have assigned external hosts to communities, we can now infer a second matrix of size  $N_{int} \times K$  containing the estimated number of flows for each internal host in each community. The rows of this matrix can be normalized to sum up to 1, to yield the posterior distributions  $\theta_i, i = 1, \dots, N_{int}$ .

#### Cluster the hosts using the LDA model output.

In each of the two matrices above, the  $(i, j)^{th}$  element is an estimate of the number of flows of host  $i$ 's flows in community  $j$ . We assign host  $i$  to community  $j$  if the  $(i, j)^{th}$  element of this matrix is greater than a certain threshold. In practice, we found that threshold =  $\frac{\text{\#of flows in dataset}}{\text{\#of hosts in dataset}}$ , i.e., average number of flows per host, provides satisfactory performance in terms of both precision and recall (in Section V we provide precise definitions and evaluate the precision and recall of detecting a P2P community). If no element in the  $i^{th}$  row of the matrix is greater than this threshold, and there is a single non-zero element in the row at the  $l^{th}$  position, then we assign host  $i$  to community  $l$ . Otherwise we do not assign host  $i$  to any community. However this is a rare case because it corresponds to a host with few very flows going to more than one community.

#### IV. DETECTING REAL P2P COMMUNITIES IN THE WILD

In this section we validate our algorithm on real-world network-flow data (see Section III-A) by checking whether the communities inferred by us correspond to meaningful P2P communities and whether they are malicious. Notice that the real-world data captured in the wild is inherently free of any labels. And in the absence of any labeled data, validating our method is a tricky problem since it is impossible to compute traditional metrics, such as false positive rate, detection rate etc. Nevertheless, to demonstrate that our algorithm infers valid P2P communities, we employ three basic strategies: **i)** We track the members of inferred communities using a public blacklist over a period of time. The intuition is that if a significant fraction of the members of a community appears on a blacklist, then that community is highly likely to be a real malicious P2P community. **ii)** We check whether a specific port number is mostly used when the members of an inferred community communicate with each other. This would verify that our method is able to successfully identify some well-known P2P communities employing a standard port number (e.g. BitTorrent). Notice that, while such well-known communities can easily be identified using a simple port

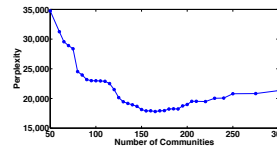


Fig. 4. Perplexity curve for using different number of communities.

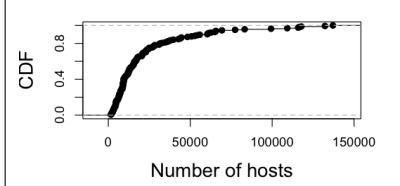


Fig. 5. CDF of the number of hosts in a cluster.

analysis, being able to identify these communities indicates that other inferred communities by our method, which don't employ a standard port number, are also likely to be real P2P communities. **iii)** Finally we check how several flow-level features vary among the members of an inferred community. The idea is that, if the inferred community is indeed a real community, then the members of that community should exhibit some similarity on some flow-level features. Before presenting our results, we first address how we set an important algorithm parameter below:

**Choosing the number of communities (K):** Recall that one important parameter of our inference algorithm is the number of communities ( $K$ ). If  $K$  is chosen too small, only strong communities will show up and weak communities are ignored by the detection algorithm. On the other hand, specifying too many communities will split true communities into pieces randomly. To estimate  $K$  using our dataset itself, we employ the notion of *perplexity* [22]. Roughly speaking, perplexity is the negative log likelihood of new, unseen data, given a model developed using seen data (Formally, perplexity =  $\exp\left(\frac{-\log(p(\text{TestData}|\text{Model}(K)))}{\text{\#of external hosts}}\right)$ ). We split up the dataset by flows, into a training portion and a test portion. Given a model trained on the training set, perplexity measures how surprising it is to observe the test data. If we split the dataset randomly several times, then a model (parametrized by  $K$ ) that consistently gives lower perplexity on many different splits than other models is a better fit to the data.

To choose an appropriate  $K$ , we compute perplexity using the 10-fold cross validation on 10 minutes of our dataset under different  $K$  values. The perplexity curve is shown in Figure 4, which shows that the perplexity value drops quickly as the number of communities is increased at the beginning. The minimum is attained around  $K \sim 165$ , and then it starts to increase which ostensibly indicates overfitting. Therefore, we choose the number of communities to be  $K = 165$ . Note that  $K = 165$  is not the exact true number of communities, but rather an approximation. Running our algorithm on our dataset with  $K = 165$  produces communities with sizes shown in Figure 5.

#### A. Validating malicious communities using blacklist

To confirm whether inferred communities are indeed real malicious P2P communities, we checked the IP addresses of the members of each community against the Spamhaus blacklist [25]. We found that three communities—with 1867, 2908, and 2273 hosts respectively—had significantly high fractions of host members blacklisted (see Figure 6(a)). Each of these three communities had roughly 10% of their host members

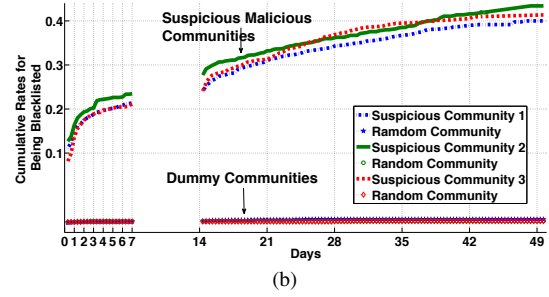
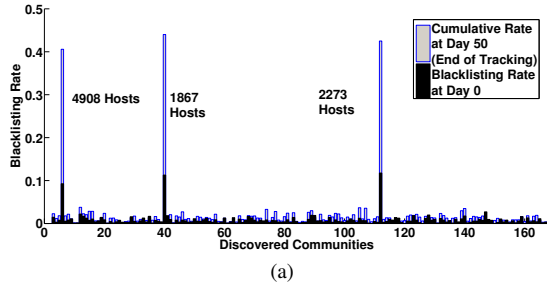


Fig. 6. (a) Fraction of hosts in each discovered community, that are present in Spamhaus’ blacklist on the day that communities are initially discovered and on day 50. (b) Growth in the fraction of hosts that are ever blacklisted during a 50 day period, for the 3 suspicious communities in (a), compared with 3 random communities of the same sizes as the suspicious communities. The blacklist is checked every 8 hours. There was a 7-day gap in data collection due to a outage on the server running our script.

blacklisted at the time we inferred communities (Figure 6(a)). 10% is a statistically significant fraction, considering that a random set of IP addresses of the same size from our dataset is likely to have a very small fraction of IP addresses blacklisted. Therefore, we conjectured that these communities were P2P botnets. Notice in Figure 6(a) that, a small fraction members of almost all  $K = 165$  communities are also blacklisted. This is explained by the fact that our model allows for some overlap between communities—therefore blacklisted hosts that appear in the 3 suspicious communities may also be part of other non-malicious P2P communities. A second reason for this is that as any machine learning algorithm, LDA, followed by our thresholding process, does not perfectly classify every host into its member communities because of lack of sufficient evidence (data).

Next, to improve our confidence that these three communities are malicious, we tracked their members over time, by repeatedly checking against the SpamHaus for a period of 50 days. Our goal was to confirm whether more of their member hosts show up in Spamhaus’ blacklist. As a baseline for comparison, we created 3 random communities of the same sizes as the suspected malicious communities, and also tracked their IPs across time, recording their cumulative blacklisted fractions. The random communities were created by uniform random sampling of IP addresses in our original dataset. In order to make sure our results were statistically significant, each of the three dummy communities was actually constructed as a set of 10 random communities of the same size, and the results from monitoring the Spamhaus blacklist were the average over these 10 communities for each of the 3 dummy communities. As shown in Figure 6(b), the cumulative ratio of blacklisted hosts for these 3 communities show a significant increase from 10% to  $> 40\%$ , while the ratios for the 3 dummy communities exhibits almost no increase. The cumulative blacklisted fractions for each of the  $K = 165$  communities after 50 days of tracking are also shown along with the results from day 0 in Figure 6(a). The gradual blacklisting of hosts in our suspected malicious communities leads us to believe that these 3 communities are with high likelihood, parts of P2P botnets used for malicious campaigns. The observation that the initial blacklisted fraction is not very high (10%) but steadily grows to a high value can be explained by the fact that a botmaster does not need to use all the bots in one

campaign simultaneously, but instead may use them in stages. This is a significant result, with the implication that bots can be detected simply from their communications patterns with peer bots, ostensibly *before* they display malicious behavior and certainly before they appear in present day blacklists.

### B. Well-known communities with standard port numbers

In order to identify known communities that use standard ports, we compute for each port (1 to 65535), the fraction of hosts which use that port at least once in each inferred community. We refer to this fraction as the *popularity of the port*. In computing popularities, hosts at both ends of a flow are considered associated with both ports (source and destination) in a flow. Only flows for which the hosts at both ends belong to exactly one community are used. Port popularities represent how frequently a specific port occurs within a community. For many P2P applications (e.g. BitTorrent), there are default ports (6881-6889) used by client-software which most peers have communication flows on. If the communities discovered by us are meaningful, then we would expect such popular ports to appear in at least some of the communities we discover (not all P2P applications used fixed ports numbers or ranges). Again we stress that port information is not sufficient to address the problem of discovering P2P communities in general. However, after detecting the communities, observing a large popularity for a certain ports within a community serves as a strong validator for the P2P application associated with the community.

We check whether there is one or more ‘featuring ports’ (a port with a sharp peak) for these communities separately for TCP and UDP. The port popularities for 8 of our inferred communities are shown in Figure 7. These communities all exhibit at least one featuring port which is well-known and associated with specific P2P application or protocol. For example, it is highly likely that the community featuring TCP port 6346 is Gnutella while the one featuring TCP port 6881 is BitTorrent. See [26] for a detailed mapping of port numbers to well-known services. We found that there are 123 communities with identifiable featuring port for a known P2P application. To name a few, we detected 2 Gnutella-based communities with a total number of 9138 hosts. Also it is interesting that we found a community featuring port 8247 (Figure 7(b)), which is not very well-known, but in [27], the authors report that

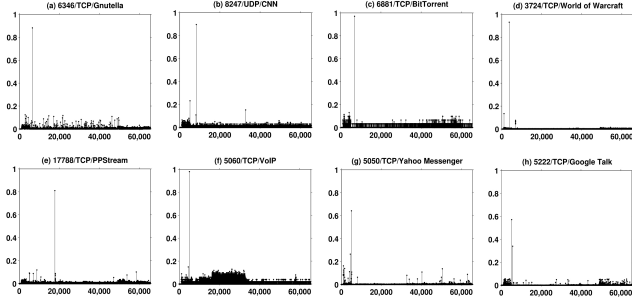


Fig. 7. (a)-(h) Port popularities: Fraction of hosts in each of 8 communities discovered by us, that use a given port at least once. The title of each plot indicates the most popular port, the protocol (UDP/TCP), and our estimate for what application it is (using [26])

port 8247 is associated with a plug-in which users download to watch the CNN news videos smoothly using P2P links to other CNN viewers. Hence we believe this to be a P2P video-sharing community for watching CNN. One of the biggest P2P applications we detected with more than 15,000 hosts, appears to be associated with the World of Warcraft gaming service (Fig. 7(d)). There are also several P2P VoIP communities (which typically use port the Session Initiation protocol (TCP port 5060) to setup a call).

### C. Variance of flow features within communities

Given that members of a community run the same P2P application, some flow level behavior among the members of a community should be similar to each another on average, but different from flows randomly chosen from our dataset. For example, the flows between BitTorrent peers are likely to use similar ports in the range 6881-6889, and also similar (likely longer) duration and payloads. The distributions of these flow features, will be different from other non-file-sharing communities as well as from randomly picked flows. Therefore, to evaluate the quality of our inferred communities, we measure the entropy of the flow features (*TCP port distribution*, *UDP port distribution*, *packets-per-second(PPS)*, *bytes-per-second(BPS)*, and *bytes-per-packet(BPP)*) for flows of each inferred community to check whether hosts in a community have similar flows with similar features.

For each feature, entropy is computed after first quantizing the distribution of that feature, so that entropy is that of a discrete random variable. Since ports are already discrete (0-65535), it is straightforward to use 65535 bins. A uniform quantization of 20 bins is applied for the other features (PPS, BPS, BPP). For a fair comparison, the community entropy for each flow feature is compared with a random sample of flows, equal in number to the number of flows in the community. We compare the community entropies for the 5 traffic features in Figure 8. In each plot, communities are arranged in increasing order of their feature entropies. Most inferred communities exhibit significantly lower entropy compared to a random set (of the same size) of flows. For example, around 160 communities have significantly lower BPS and BPP entropies than the baseline benchmark, which suggests that these communities are well clustered and represent meaningful P2P communities.

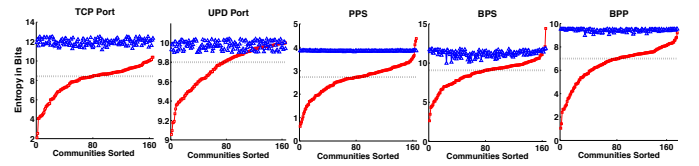


Fig. 8. Entropies of 5 flow features for each of  $K = 165$  communities, sorted by entropy ( $\square$ ) along with corresponding entropies of the flows of random sets of hosts ( $\triangle$ ) equal in number to the size of the community being compared with. The dotted line is the median entropy of discovered communities.

## V. PERFORMANCE EVALUATION

Our goal in this section is to quantify how accurately our system can discover a P2P community, in terms of the properties of the community. In particular, we would like to explore what types of communities can and cannot be detected by the system. In order to measure this in full generality, we require labeled data specifying which P2P communities each host is part of. For this purpose we synthetically generate communities with various P2P topologies and connectedness into our dataset to serve as a labelled data.

**Embedding Synthetic Communities:** When embedding communities, all hosts in the embedded communities are chosen from IPs that already exist in the dataset. In other words, we do not add any new nodes but only new edges to the underlying communication graph. This is important because simply embedding an isolated community consisting of new nodes is not appropriate for measuring the performance of our system, as it would be trivial to detect such isolated communities. Also, we make sure that the embedded graph is a bipartite graph, as would be observed via our measurement setup as described in Section III-A. That is, we embed the edges corresponding to a synthetic community only if they cross the network boundary. By embedding synthetic communities IP addresses already in the dataset, we effectively make each member of a synthetic community belong to at least one more community other than the synthetic one. Hence, the results we present in this section essentially serve as lower bounds on the real-world performance of our system.

**Metrics:** Let  $C$  denote an embedded community and let  $\hat{C}$  denote its detected version. We take  $\hat{C}$  to be the inferred community with the most common IP addresses with the embedded community  $C$ . To evaluate accuracy we employ two metrics: *precision* and *recall*. *Precision* is defined as the fraction of the members of  $\hat{C}$  that actually come from the underlying ground-truth community  $C$ . *Recall* is defined as the fraction of members in  $C$  that appear in the detected community  $\hat{C}$ . Formally,  $precision_C(\hat{C}) = \frac{|C \cap \hat{C}|}{|\hat{C}|}$ ;  $recall_C(\hat{C}) = \frac{|C \cap \hat{C}|}{|C|}$ .

**Synthetic Community Structures:** We measure precision and recall of detecting synthetic P2P communities generated as per two different random graph models: (i) the Erdos-Renyi (ER) model [28], and (ii) the Barabasi-Albert (BA) model [29]. In Erdos-Renyi random graphs, each vertex has an equal probability  $p$  of having an edge to any other vertex. The degree distribution of an Erdos-Renyi graph is Binomial. In an ideal case where every host is equally accessible, P2P applications exhibit an Erdos-Renyi structure. For example,

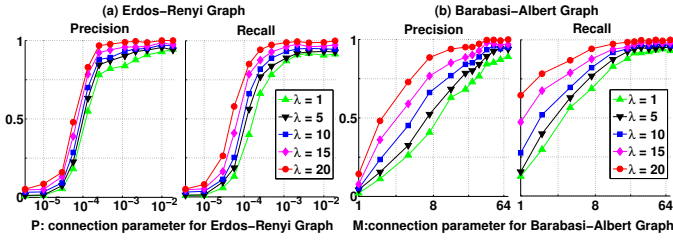


Fig. 9. Precision and recall performance on discovering Erdos-Renyi and Barabasi-Albert structured P2P communities. The detection precision and recall sharply increase for the Erdos-Renyi communities around the critical value  $p = \frac{\ln n}{n}$ , while for the Barabasi-Albert communities, the detection ratio gradually improves with better connectivity  $M$  and increased activity  $\lambda$

in the Conficker-C botnet[30], when a new peer wants to join the botnet, it randomly generates a list of IPs from IPv4 space and tries to establish a connection with the selected IP address. Assuming each peer has equal accessibility, the chances that two peers get connected is the same for any pair of peers. However in practice, not every peer has the same level of accessibility. Some peers are behind a NAT or firewall, which prohibit incoming connections. Hence peers with public static IP addresses will be more reachable and preferred. This selectivity can be captured by the Barabasi-Albert random graph, which is based on a ‘preferential attachment’ mechanism. In fact the Barabasi-Albert model is shown to capture the dynamics of many real-world networks, such as the World Wide Web, Protein Networks, Co-Authorship, etc.[29]. More specifically, Barabasi-Albert model defines a random graph generation process as follows: (i) New peers join the P2P community one at a time by establishing  $M$  links to the existing peers in the network. (ii) The probability of a peer being selected by a newcomer is proportional to its degree. Better connected peers are preferred more and essentially become hubs of the network.

**Detection Performance:** To embed a synthetic community, we first create an ER or BA random graph, with 10,000 external hosts and 1000 internal hosts, and remove any edges that do not cross our network boundary. We then add artificial flows to embed the synthetic community into our dataset. In the BA model, The number of artificial flows per edge in the random graph is determined by a Poisson distributed parameter  $\lambda$  which captured how active a pair of hosts is. Then we run our algorithm and check precision and recall of the detected community. We independently repeat each experiment 5 times and the averaged results are shown in Figure 9 for various connectivity and activity parameters (i.e.  $p$  and  $\lambda$  for Erdos-Renyi model,  $M$  and  $\lambda$  for Barabasi-Albert model). As expected, the detection performance monotonically improves with increased connectivity (increasing  $p$  or  $M$ ), as well as increased activity ( $\lambda$ ). That is, when a community is well-connected and exhibits reasonable activity, our system can accurately identify it. For example, at an average activity rate of  $\lambda = 15$  and  $p = 0.01$ , the precision and recall are both better than 96%.

**Effect of connectedness ( $p$  and  $M$ ):** An Erdos-Renyi graph is in general disconnected for small values of  $p$  and is known to form a single giant component with high probability when  $p$  is

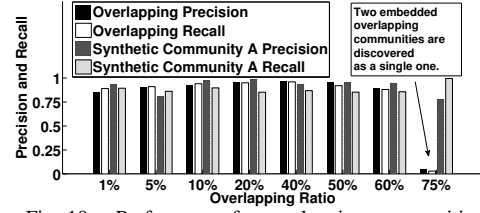


Fig. 10. Performance for overlapping communities.

raised above a critical value. We observe this effect in Figure 9 as a sharp increase in precision and recall values around the critical value of  $p$ . In contrast, Barabasi-Albert graphs have no such critical value for connectivity (BA graphs are always a single connected component) and the performance smoothly improves as  $M$  increases.

**Overlap between communities:** In general, P2P communities may overlap with each other since some nodes may participate in more than one community. To test how accurately our system can detect and distinguish between overlapping communities, we embed two overlapping synthetic Barabasi-Albert communities, A and B, simultaneously into real-world dataset. Each community has 11,000 nodes (1,000 internal and 10,000 external). The amount of overlap is controlled by the overlap ratio  $q$ , which is defined as the fraction of hosts shared by both communities (i.e.  $q = \frac{\text{\#sharedHosts}}{11000}$ ). After creating flows with  $\lambda = 15$  and embedding into our real-world data, we feed the mixed dataset to our algorithm. We are interested in how accurately our model detects embedded communities, as well as how accurately it identifies the overlapping portion (i.e. hosts which are members of both communities). The precision and recall results for varying  $q$  are shown in Figure 10 for both the overlapping part and for community A. We observe that, when the overlap ratio is smaller than 75%, the system is able to identify the embedded communities and the overlapping part fairly accurately. Note that, we present the results only for community A in Figure 10, since the system performs almost equally for A and B when  $q < 75\%$ . On the other hand, when overlapping ratio is  $q \geq 75\%$ , the system detects both communities as a single community (i.e. community A). Since community B is no longer detected, the precision and recall for just the overlapping part drops almost to zero for  $q \geq 75\%$ . This provides us with a rough figure of 75% as the overlap threshold above which communities lose their individuality and instead get discovered as a single fused community. However, since overlap refers to participation of hosts in multiple P2P application communities within a time interval small enough to avoid the effects of dynamically changing IP addresses (1 hour in all our experiments), we believe it would be very unlikely for communities to have an overlap of  $\geq 75\%$ .

## VI. LIMITATIONS AND FUTURE WORK

The most important limitations of our work are: (i) We must deal with limited amounts of data at a time because IP addresses assigned to hosts can change over time, (ii) We can only measure flows that cross a certain network boundary, and (iii) We require centralized communities to be filtered prior to



inference and this is not easy to achieve perfectly. Naturally, all these limitations affect the final quality of inference.

It would be useful to extend our statistical model to include flow features such as bits per second, packets per second, etc. It would also be useful to incorporate the timestamps of flows – if a host has flows to two different hosts within a very short time interval, it is likely that the flows, and hence the two hosts, belong to the same P2P application. Finally, we estimated the number of communities,  $K$ , by empirically evaluating how well the observed data fits the model for various values of  $K$ . This may not be the best way to address this issue. In particular, it is possible to estimate  $K$  more directly by using the Hierarchical Dirichlet Process [31] or non-parametric models like the Indian Buffet Process [32]. Finally, in the future, we are interested in tracking the evolution of suspicious communities (e.g. suspected botnets) over time by running inference repeatedly over successive time-windows.

## VII. CONCLUSIONS

We formulated an unsupervised probabilistic inference model to address the problem of discovering P2P application communities. The proposed method can discover P2P communities independent of their specific protocol or topology, using only communication patterns, allowing: (i) policy enforcement related to P2P traffic in private networks such as enterprises and schools, (ii) discovery of new/evolving malicious P2P botnets, and (iii) network service providers a means to monitor P2P traffic as separate communities at the application level. We evaluated and validated our method in a number of ways. Our results indicate that the inferred groupings of hosts into communities is meaningful as we are able to detect well-known communities that operate on known ports, and also because the entropy of flow features for a community is typically much lower than what it would be for randomly sampled flows. We are the first to demonstrate the discovery of communities that are very likely to be P2P botnets. We found that the IP addresses of hosts in 3 suspicious communities discovered by us gradually appear in public blacklists over the course of 50 days, indicating that discovering P2P communities by our approach can lead to quicker discovery of bots than current reactive methods.

## REFERENCES

- [1] C. G. Bartlett, J. Heidemann and J. Pepon, “Estimating P2P traffic volume at USC,” in *Technical Report, USC/Information Sciences Institute*, 2007.
- [2] A. Madhukar and C. Williamson, “A longitudinal study of P2P traffic classification,” in *Proc. of Int. Symposium on Modeling, Analysis & Simulation of Computer and Telecommunications Systems*, 2006.
- [3] T. Karagiannis, A. Broido, M. Faloutsos, and K. Claffy, “Transport Layer Identification of P2P Traffic,” in *Internet Measurement Conference (IMC)*, 2004, pp. 121–134.
- [4] D. Dittrich and S. Dietrich, “P2P as botnet command and control: A deeper insight,” in *MALWARE 2008. 3rd International Conference on Malicious and Unwanted Software*, 2008.
- [5] —, “New directions in peer-to-peer malware,” in *IEEE Sarnoff Symposium*, April 2008.
- [6] J. B. Grizzard, V. Sharma, C. Nunnery, B. B. Kang, and D. Dagon, “Peer-to-peer botnets: Overview and case study,” in *First Workshop on Hot Topics in Understanding Botnets*, 2007.
- [7] 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 *LEET’08: Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, 2008.
- [8] <http://www.symantec.com/connect/blogs/zeusbotspyeye-p2p-updated-fortifying-botnet>.
- [9] S. Nagaraja, P. Mittal, C. Y. Hong, M. Caesar, and N. Borisov, “Botgrip: Finding P2P bots with structured graph analysis,” in *Proceedings of the 19th USENIX conference on Security*, 2010.
- [10] J. Zhang, R. Perdisci, W. Lee, U. Sarfraz, and X. Luo, “Detecting stealthy P2P botnets using statistical traffic fingerprints,” in *International Conference on Dependable Systems and Networks, Dependable Computing and Communications Symposium*, 2011.
- [11] G. Gu, R. Perdisci, J. Zhang, and W. Lee, “BotMiner: Clustering analysis of network traffic for protocol- and structure-independent botnet detection,” in *17th USENIX Security Symposium*, 2008.
- [12] B. Coskun, S. Dietrich, and N. Memon, “Friends of an enemy: Identifying local members of peer-to-peer botnets using mutual contacts,” in *26th Annual Computer Security Applications Conference*, 2010.
- [13] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, “Blink: multilevel traffic classification in the dark,” in *Proceedings of the ACM SIGCOMM 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Philadelphia, Pennsylvania, USA, August 22-26, 2005*, 2005.
- [14] S. A. Baset and H. Schulzrinne, “An analysis of the skype Peer-to-Peer internet telephony protocol,” in *IEEE International Conference on Computer Communications (INFOCOM’06)*, 2004.
- [15] 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)*, February 2008.
- [16] T.-F. Yen and M. K. Reiter, “Traffic aggregation for malware detection,” in *Proceedings of the 5th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 2008.
- [17] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, “BotHunter: Detecting malware infection through ids-driven dialog correlation,” in *Proceedings of the 16th USENIX Security Symposium*, August 2007.
- [18] G. Gu, V. Yegneswaran, P. Porras, J. Stoll, and W. Lee, “Active botnet probing to identify obscure command and control channels,” in *Proceedings of the 2009 Annual Computer Security Applications Conference*, ser. ACSAC ’09, 2009.
- [19] Z. Xu, L. Chen, G. Gu, and C. Kruegel, “Peerpress: utilizing enemies’ p2p strength against them,” in *Proceedings of the 2012 ACM conference on Computer and communications security*, ser. CCS ’12, 2012.
- [20] S. Fortunato, “Community detection in graphs,” *Physics Reports*, vol. 486, pp. 75–174, 2010.
- [21] E. Stinson and J. C. Mitchell, “Towards systematic evaluation of the evadability of botnet detection methods,” in *Proceedings of the 2nd conference on USENIX Workshop on offensive technologies*, 2008.
- [22] D. M. Blei, A. Y. Ng, M. I. Jordan, and J. Lafferty, “Latent dirichlet allocation,” *Journal of Machine Learning Research*, vol. 3, 2003.
- [23] Z. Liu, Y. Zhang, E. Y. Chang, and M. Sun, “PLDA+: Parallel Latent Dirichlet Allocation with data placement and pipeline processing,” in *ACM Transactions on Intelligent Systems and Technology, special issue on Large Scale Machine Learning*, 2011, <http://code.google.com/p/plda>.
- [24] T. L. Griffiths and M. Steyvers, “Finding scientific topics,” *PNAS*, vol. 101, no. suppl. 1, pp. 5228–5235, 2004.
- [25] <http://www.spamhaus.org/zen/>.
- [26] <http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.txt>.
- [27] <http://windowssecrets.com/top-story/watch-a-live-video-share-your-pc-with-cnn>.
- [28] P. Erdos and A. Renyi, “On the evolution of random graphs,” in *Publication of the Mathematical Inst. of the Hungarian academy of sciences*, 1960, pp. 17–61.
- [29] R. Z. Albert and A. B. Director, “Statistical mechanics of complex networks,” in *Reviews of Modern Physics* 74, 2001.
- [30] <http://mtc.sri.com/Conficker/addendumC/>.
- [31] Y. W. Teh, M. I. Jordan, M. J. Beal, and D. M. Blei, “Hierarchical dirichlet processes,” *Journal of the American Statistical Association*, vol. 101, 2004.
- [32] T. Griffiths and Z. Ghahramani, “Infinite latent feature models and the Indian buffet process,” Gatsby Unit Technical Report GCNU-TR-2005-001, Tech. Rep., 2005.