Mining Generic Patterns and Communities from Heterogeneous Network Traffic

Liyun Li

May 5, 2013
Mining Generic Patterns and Communities from Heterogeneous Network Traffic

DISSERTATION

Submitted in Partial Fulfillment

of the Requirements for the

Degree of

DOCTOR OF PHILOSOPHY (Computer Science)

at the

POLYTECHNIC INSTITUTE OF NEW YORK UNIVERSITY

by

Liyun Li

May 2013
Mining Generic Patterns and Communities from Heterogeneous Network Traffic

DISSERTATION

Submitted in Partial Fulfillment

of the Requirements for the

Degree of

DOCTOR OF PHILOSOPHY (Computer Science)

at the

POLYTECHNIC INSTITUTE OF NEW YORK UNIVERSITY

by

Liyun Li

May 2013

Approved :

__________________________
Department Head

Copy No. ____________________
Approved by the Guidance Committee:

**Major**: Computer Science

______________________________
Nasir Memon
Professor of
Computer Science and Engineering

______________________________
Lisa Hellerstein
Professor of
Computer Science and Engineering

______________________________
Yong Liu
Associate Professor of
Electrical and Computer Engineering

______________________________
Phyllis Frankl
Professor of
Computer Science and Engineering
Microfilm or other copies of this dissertation are obtainable from

UNIVERSITY MICROFILMS
300 N. Zeeb Road
Ann Arbor, Michigan 48106
Vita

Liyun Li was born in Chuzhou, Anhui, China in 1985. He received his Bachelor degree in Electronic Engineering from Tsinghua University in Beijing in 2006.

In September 2008, Liyun Li enrolled in the Computer and Information Science Department at Polytechnic University (which is now the Computer Science and Engineering Department at Polytechnic Institute of New York University) to further his study in a Ph.D program. He worked with Professor Nasir Memon as a research assistant on network security research projects. His research interests include data mining, machine learning and network security. During his Ph.D. study, he published several papers in major machine learning related conferences.
Dedicated to my beloved wife Lili Sun, who blessed and supported me.
Acknowledgements

The author wants to give his thanks to Nasir Memon, his supervisor.
The author also wants to give his thanks to all of the co-authors who spent endless hours and effort, to work with him on some research problems.

Liyun Li, Polytechnic Institute of New York University
May 5, 2013
ABSTRACT

Mining Generic Patterns and Communities from Heterogeneous Network Traffic

by

Liyun Li

Adviser: Professor Nasir Memon

Submitted in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy (Computer Science)

May 2013

In this thesis, we focus on the development and application of machine learning techniques with the goal of discovering generic patterns and communities from heterogeneous network traffic data. Most enterprises and institutions have ready access to network traffic data, but they currently extract little or no information from it. We study ways in which machine learning techniques can be applied to extract information from such network traffic data. The value of our extracted information falls into two main categories (1) distinctive patterns of network traffic which encode hidden information of the communication (2) inherent but non-evident community or clustering structures of network entities.

One of the growing problems for network administrators is the abuse of transferring certain contents. To prevent such network abuse, network administrators have become more and more interested in detecting network traffic patterns in a real-time fashion. We show that, by applying novel machine learning algorithms in classifying the characteristics of network traffic flows, one can accurately determine the content type being transferred between two network hosts in almost real time. Our approach is to use an ensemble of cost-efficient decision trees to do on-demand feature extraction and content type prediction.

In addition to the supervised learning of network traffic patterns, we utilize unsupervised state-of-the-art machine learning techniques in the unique setting of enterprise bipartite network traffic graph. We show that, using a statistical Bayesian approach, one can accurately decompose the bipartite network traffic into meaningful clusters. In
the scenario of peer-to-peer traffic, enterprises such as ISPs will find such clusters to be useful in policy enforcement, network monitoring and discovering new malicious p2p applications. Besides this p2p community detection, studying the user-to-website graph of network web access flows, we show that similar approaches can also infer and extract hidden topical community structures from the web-access data. With the help of crawled keywords from the websites, our method can not only detect the communities accurately, but also interpret them into meaningful topics. We also study the deployment of a machine learning technique driven system which can generate scores to detect infection sources and potential infectious external hosts from the viewpoint of an institution/enterprise where all the cross-boundary traffic can be observed.

With experimental deployment of all these proposed techniques, we demonstrate the significant benefits that machine learning techniques can bring to us in heterogeneous network data analysis, including network traffic pattern recognition, network community structure discovery and network behavior monitoring and modeling.
Contents

Vita iv

Acknowledgements vi

Abstract vii

1 Introduction 1

1.1 Machine Learning in Computer Networking 2

1.2 Problems We Solve 4

1.3 Contributions 6

2 Our Contributions 7

3 CoCoST: Computational Efficient Classifier to Classify Content Types from Network Flow Characteristics 10

3.1 Background and Related Work 10

3.2 Preliminaries About Cost Effective Classifiers and Decision Trees 13

3.3 CoCoST: Building the Universal Classifier from the Specialized 15

3.3.1 The Base Classifier: LASC Tree–Look Ahead Tree with Sup- pressed Cost 16

3.3.2 Building the Specialized: Boosting and Inverse Boosting 19

3.3.3 Combine the Specialized into a Meta-Classifier 20
3.4 Experimental Results ........................................... 21
   3.4.1 Data and features ....................................... 21
   3.4.2 Performance of different base classifiers ............. 24
   3.4.3 Performance of CoCoST Versus Random Forest ....... 26

4 Mining Generic Network Community Structures .................. 31
   4.1 Background and Related Work ............................ 32
       4.1.1 Peer-to-Peer Community Detection ................. 33
       4.1.2 P2P Botnet Detection: ........................... 34
       4.1.3 Web-Domain Topical Community Detection ........ 35
   4.2 Latent Dirichlet Allocation: An Statistical Bayesian Clustering Algorithm 36
       4.2.1 Cluster the hosts using the LDA model output .... 39
   4.3 Discovering Peer-to-Peer Application Communities ....... 41
       4.3.1 DataSet and Measurement Setup .................. 41
       4.3.2 Well known communities ......................... 44
       4.3.3 Tracking suspected malicious communities ....... 48
       4.3.4 Performance Evaluation .......................... 50
       4.3.5 Embedding Synthetic Communities ................ 51
       4.3.6 Synthetic Community Structures .................. 52
       4.3.7 Detection Performance ............................. 53
       4.3.8 Overlap between communities ...................... 54
       4.3.9 P2P vs. Client-Server Communities ................ 55
   4.4 Discovering Web Semantic Topical Communities ............ 56
       4.4.1 Evaluation Metric and Labeled Dataset from DHCP Log 57
       4.4.2 Performance of the Mined Topical Communities ... 57
       4.4.3 Expected and Unexpected Communities .............. 60
   4.5 Further Privacy Inference for Fully Anonymized Network Flow DataSet .......... 63
       4.5.1 Framework and Methodology to Make Privacy Inference 63
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.5.2</td>
<td>Tentative Proof of Concept</td>
<td>65</td>
</tr>
<tr>
<td>5</td>
<td>Detecting Infection Source and Building Predictive Blacklists with an Attack-Source Scoring System</td>
<td>66</td>
</tr>
<tr>
<td>5.1</td>
<td>Background and Related Work</td>
<td>68</td>
</tr>
<tr>
<td>5.2</td>
<td>The Attack-Source Scoring System for Tracing Maliciousness of External Hosts</td>
<td>69</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Attack-Source Score Charges Stems from IDS Alerts and Suspicious Behavior Changes</td>
<td>69</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Attack-Source Score Computation &amp; Propagations</td>
<td>71</td>
</tr>
<tr>
<td>5.3</td>
<td>Experimental Results</td>
<td>73</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Case 1: Finding Infection Source</td>
<td>73</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Case 2: Predictivity of the Hosts with Worst Attack-Source Scores</td>
<td>75</td>
</tr>
<tr>
<td>6</td>
<td>Conclusions</td>
<td>78</td>
</tr>
</tbody>
</table>
# List of Figures

3.1 Five Possible Components of the One Step Look-Ahead Tree with Suppressed Cost (LASC Tree) ........................................... 18

3.2 The Learning Curve: Number of Instances Versus Accuracy Using the C4.5 Classifier .............................................................. 23

3.3 Performance of base classifiers with Random Feature Selection. The lines are envelop of the performance points of the three classifiers. It shows that LASC Tree with Suppressed Cost is more cost-efficient than C4.5 and SVM ..................................................... 25

3.4 Individual Tree Performance of Inverse Boosted Trees .................. 28

3.5 Compare the Frequency-Cost distribution between CoCoST and Random Forest. Noting that the distribution for SVM is only one bar as all the instances have the same cost .............................................. 29

3.6 Structure of the CoCoST meta-classifier ....................................... 30

4.1 The probabilistic generative process that explains the dataset of (User, IP/Website) pairs. The numbers in parenthesis correspond to the steps in Section 4.2 and Figure 4.3 ....................................................... 38

4.2 The probabilistic generative process for Web-Topical-Community Detection, with the extracted keywords from websites as seed information to annotate the communities ................................................. 38
4.3 The probabilistic graphical model for LDA. Only the shaded nodes are observed. .................................................. 39

4.4 Perplexity curve for using different number of communities. .................. 42

4.5 CDF of the number of hosts in a cluster ........................................ 42

4.6 (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 which application it is (using [1]) ................... 44

4.7 Entropies of 5 flow features for each of $K = 165$ communities, sorted by entropy (□) along with corresponding entropies of the flows of random sets of hosts (△) equal in number to the size of the community being compared with. The dotted line is the median entropy of discovered communities. ........................................ 45

4.8 (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 an outage on the server running our script. ........................................ 49

4.9 Precision and recall performance on discovering Erdos-Renyi or 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 $\lambda$ ........................................ 51
4.10 Detection performance on overlapping communities for different overlap ratios. ......................................................... 55

4.11 Detection performance for different number of servers. As the number of servers increases, the community becomes more P2P-like. ............. 56

4.12 Detection precisions and accuracies on labeled demographical communities, compared with K-Means clustering algorithm. .................. 59

4.13 Performance of LDA communities with Changing Level of Website Access Flow Records Captured ................................................. 60

4.14 Examples of 4 content topical communities discovered: each column represents a topical community. The colored text is the name we give to the community, followed by the most frequent keywords from the websites associated with these users’ accesses. ...................... 62

4.15 Degree distribution for the case-studies: (1) Apple-Steve-Jobs community along with featuring keywords (2) The Iphone-Android Sales Community with featuring keywords. Both keyword lists are sorted by their mined frequencies. ......................................................... 63

5.1 Attack-source score change when infection happens. The X-axis is the time and Y-axis is the attack-source score. ................................. 74

5.2 Cumulative ratio for being blacklisted for the most worst attack-source-score ips generated by our approach. The red line is the cumulative ratio for the top-10 worst IPs being finally blacklisted by third-party blacklists, while the green line represents the ratio for the top-100 and the blue one for top-200. ................................................................. 76
List of Tables

3.1 Computational Prices(milliseconds) for Each Feature with Different Buffer Size ................................................. 22
3.2 Performance of different base classifiers on different sizes of features sets chosen by Sequential Forward Feature Selection. $\alpha = 1$ for LASC Tree. The measures for performances are accuracy(%) and cost(ms) .................................................. 26
3.3 Choosing different values of $\alpha$ will cause different cost sensitivity suppression .................................................. 27
3.4 Comparison among CoCoST, SVM, and the best Random Forest($\alpha=1$, consisting of 20 trees) .................................................. 29
4.1 Summary of notation .......................................................... 37
4.2 Labeled Ground Truth: Demographical Topical Communities ................................................................. 58
4.3 Detected communities and their statistics .......................................................... 61
5.1 Summary of Symbols for Behavior Features .......................................................... 70
5.2 Worst attack-source-score IPs generated by our system with IP, Domain and Third-Party Verifications .................................................. 76
Chapter 1

Introduction

Machine learning, as a branch of Artificial Intelligence, has been widely used in both on-line real-time data analysis and off-line batch data processing scenarios. In almost any area where data is readily available for analysis and people expect to obtain certain benefits from analyzing the existing data, such as medical diagnosis [56], computer networking [75], network intrusion detection [12] [89] [60] [90], spam filters [16], and recommender systems [81] [13], the technique of machine learning could be utilized to provide insights or inspirations which will ultimately lead to better human understandings of the data, thereby creating benefits including automatic prediction of unseen instances as well as the discovery of new knowledge.

The core of machine learning deals with two aspects: data representation and concept generalization. The representation of data instances/examples, as well as the functions evaluated on these instances are part of all machine learning systems. In constructing the machine learning system, the data representation is usually achieved by extracting a set of values, that is, feature values, from the set of data objects, as the examples to train the machine learning system. This procedure is usually called the feature extraction. Concept generalization is the process that transforms the data representation to certain properties of the system. Such process is also called model building. After that, the system will perform well on unseen data with accurate predictions if the
formed concept correctly incorporates the inherent characteristics of data instances. The conditions under which this can be guaranteed are a key object of study in the subfield of computational learning theory [84].

1.1 Machine Learning in Computer Networking

With the rapid growth and development of internet based technologies such as social networks, search engines, and cloud computing, machine learning has become more and more common in analyzing data and mining new knowledge in the computer networking areas. Especially with the boost of Web 2.0 and the emergence of social media, the role of web users has shifted from passive navigators to active content creators, moderators and consumers. Given the rich source of data and tools provided by Web 2.0, users have the ability to actively create and share content while collaborating and communicating online. Consequently, there are now many rich sources of heterogeneous data created by millions of users participating in numerous online social interactions, not only accessing but creating, sharing, and annotating content they are interested in.

In this thesis, we take a unique enterprise/institute point of view, where we observe the users inside an enterprise/institute communicate via internet with the external world. From this enterprise/institute point of view, such vast amount of networking data is very heterogeneous in nature and encourages the application of various machine learning techniques for pattern recognition, knowledge discovery and mining, and data analysis and visualization. Among the objectives of processing the vast amount heterogeneous networking data, there are two aspects that this thesis focuses on: mining generic patterns and communities from the networking data.

With more and more active users participating in networking activities, mining patterns from the networking data has been an attractive problem since understanding the content which the user is involved in will provide insights into user behavior. From an
enterprise/institute point of view, monitoring the content type of network activities is also important in the sense of preventing potential abuse of contents. For example, if an institute could accurately classify the content type of network flows, potential network abuse behaviors such as downloading illegal/un-copyrighted contents (both audio and video) could be detected and thus prevented [61]. Given the large amount and real-time nature of the network activities, such network abuse detection is only possible with online machine learning classifiers which can effectively predict the content type of network flows in a computationally efficient fashion.

Another important area where machine learning techniques are being applied in computer networking is understanding the community (or namely clustering) structures of network users. Understanding the implicit and hidden structures of self-formed user-communities could be important in various aspects. Take the application level user-community structure detection for example: if an institute such as ISP, Enterprise or University could accurately detect Peer-to-Peer communities, Internet-Service-Provider(ISPs) can better monitor their network and provide better service by detecting the malicious communities and blocking them; Companies could execute better policy enforcement such as limiting certain P2P applications to enhance efficiency of their employees; In addition, users who share, communicate, and interact with each other online come from diverse demographical, geographical, and even topical interest groups which may be implicitly reflected in content that they access and other online behavior patterns they demonstrate. If machine-learning driven community detection could be topical, the discovery of either explicit or implicit user topical communities will be beneficial since many web or enterprise based applications could specifically tailor and leverage such community information [93]. Applications such as recommendation [80], personalization [27], content distribution [97], and marketing [93], can become more efficient, targeted and focused. Understanding the topical interests within an organization could also be useful in many other ways. For example as an enterprise, if one can
understand which types of professional topics and industry trends are ongoing, he/she can then predict interest trends, grasp the most popular industry innovations and take opportunities to make the business more successful. For non-profit driven institutes such as colleges and universities, a better understanding of the topical communities will facilitate the school to be able to provide better educational resources as well as more specific talent discovery and matching.

Besides mining generic patterns and communities for the objectives we mentioned above, security has always been an important area where various techniques come into play. Therefore machine learning based network security monitoring system has been another important application field. Since security breaches and intrusions can be initiated from any of the entire external network entities and the attacks have been becoming more and more sophisticated and dynamic [47], singular detection of infected hosts within specific domains such as within an enterprise/institution does not necessarily imply being free of risk. For example, when one of the internal hosts gets infected and behaves as a botnet zombie controlled by the botmaster, the actions of only quarantining and fixing this infected machine may not be sufficient, since there might already be many other hosts infected and controlled by the botmaster. And as long as the infection source is still at large and accessible to internal hosts, further risks of more infection still persist. Therefore, building and deploying machine learning based network systems to accurately predict potential external risks (especially potential attack sources) has been an important problem in network security. What is desirable for such systems is an accurate and constantly updated concept of the view of the external world where most infectious sources known.

1.2 Problems We Solve

In this thesis, we study how to develop accurate, effective and computationally efficient machine learning systems and report our experiences in deploying these systems
in scenarios of network data pattern classification, network community detection and network attack-source inference. More specifically, we address the following problems in this work:

- First of all, one central problem which limits the practice of machine learning classifiers is the computational cost of acquiring the feature values which are necessary for making the prediction. We aim to solve the problem of designing and implementing prediction cost (or test) effective classifiers which can serve as online realtime prediction scenarios.

- Another challenge is how to make the classifiers accurate when predicting the generic patterns of network flow content type. As individual cost-effective classifiers might not be as accurate as other classifiers such as Support Vector Machines (SVMs) which use all the features to make predictions. How can we combine multiple diverse individual classifiers to make accurate predictions while still being accurate?

- We also looked into community detection (clustering) problems in extracting generic network community structures from the viewpoint of an enterprise/institute. More specifically, we studied two problems: (1) the P2P application community detection problem and (2) the Web semantic topical-community detection problem.

- Finally, we looked into machine learning applications to solve network security problems. We studied how to build a network behavior based machine learning driven system which computes a potential attack source score for all the externals hosts outside an institute’s boundary.
1.3 Contributions

Our main contributions in this thesis are new machine learning techniques for accurately and efficiently mining the generic patterns and communities from heterogeneous network traffic data. Specifically, we have the following contributions:

- We propose a generic framework of building cost-effective decision trees and combining individual trees to make an accurate classifier.

- We studied the state-of-art Latent-Dirichlet-Allocation unsupervised clustering algorithm and adapted it to the unique settings of discovering network communities with limited cross-boundary network flow observations.

- Experimentally, such LDA-driven clustering algorithms are shown to be able to accurately detect and discover generic topical communities in various use-cases such as P2P application community detection and Web-Semantic community detection.

- Finally, we presented an attack-source scoring system which could effectively identify the potential attackers to our network and construct a predictive blacklist.

The remainder of this thesis is organized as follows. In the next chapter, we provide technical background and discuss related work. In Chapter 2, we discuss our contributions in this thesis in more detail. In Chapter 3, we describe our cost-efficient meta-classifier to detect the content types of the network flows. Chapter 4 considers the application of an unsupervised statistical Bayesian approach to discover generic P2P application communities and web semantic topical communities, and Chapter 5 discusses a machine-learning driven system which mines the generic network communication patterns to infer attack sources and build predictive blacklists. Finally, Chapter 6 provides concluding remarks and some suggestions for future work.
Chapter 2

Our Contributions

We now discuss the contributions of this thesis in more details.

(1) Our first contributions in this thesis are new techniques for designing computation cost-effective decision trees and building tree-based accurate meta-classifiers. In particular:

(a) Firstly, by taking into consideration the frequency of each node (i.e., nodes with fewer data instances passing through them are allowed to contain more expensive features), and looking forward in building a decision tree, we propose a new decision tree algorithm, called one step look-ahead tree with ‘Suppressed Cost’ (abbreviated as 'LASC Tree’ in the rest of paper), as our base weak classifier. Experimental results show that, by taking into account the frequency information of each node and expanding the search space by looking ahead, an increase in efficiency can be expected with high probability.

(b) Secondly, to train different tree classifiers specialized at different target classes, we extend the technique of inverse boosting by taking the test cost into account, which actually provides inverse boosting with a new meaning for practice. Inverse boosting provides specialized trees which tend to use cheap
features to classify the easy instances.

(c) Finally, by combining inverse-boosted trees and standard Adaboost trees using stacked generalization, where the decisions of each tree on some validation data become features for the new meta-classifier, we obtain a meta-classifier with high accuracy and low cost. This technique of building meta-classifiers over various weak classifiers that are specialized at different targets, provides a general approach to construct better meta-classifiers.

(2) We study the problem of applying the state-of-art Latent Dirichlet Allocation (LDA) clustering algorithm in mining generic network community structures. In particular, our contributions are as follows:

(a) We tackle the problem of mining generic network communities from the unique viewpoint within an organization, and formulated the topical community detection problem as a probabilistic model for clustering users in a graph. With two different use case settings, P2P community detection and Web semantic topical community detection, we validate our approach which can accurately mine and discover the authenticate communities in both cases.

(b) In P2P community detection, we show that our approach is able to identify the presence of a number of well-known P2P communities in large-scale network-flow data, measured at the peering boundary of a tier-1 ISP. For Web semantic topical community detection, we report experimental results on a real-world network flow trace from an institute. Our proposed method can effectively detect topically meaningful communities, including both demographical communities and non-evident communities sharing similar interests in certain semantic topics, which cannot readily be detected by other known methods.

(c) We rigorously quantify the ability of our approach to infer communities,
using a combination of real-world network-flow data and synthetically embedded communities.

(d) We demonstrate the discovery of entirely newly-formed communities that we suspect are malicious. Our suspicion is corroborated by the gradual appearance of the hosts that we suspect to be malicious, in public blacklists over the course of 50 days.

(3) We propose an unique system which monitors the "attack-source" score of all the external hosts by leveraging both symptoms of infected hosts and communication patterns. In particular, we make the following contributions:

(a) We propose an attack source score system from a ubiquitously enterprise viewpoint, where we leverage the fully-observed network behaviors of our internal hosts to derive attack source scores for the entire external world.

(b) By quantifying the suspiciousness and maliciousness of internal host behaviors into a notion of charge score, we compute and propagate such charge scores as attack-source charges for all the external entities communicating with the internal hosts.

(c) With two use cases, we experimentally show that the proposed attack source scores could be used in both discovering infection sources and predicting future malicious external hosts even before they get blacklisted by third-parties.
Chapter 3

CoCoST: Computational Efficient Classifier to Classify Content Types from Network Flow Characteristics

Decision tree classifiers (DTC) are unstable and diverse classifiers that are easy to implement [73] [84]. The support of a decision tree, which we call ‘frequency’, is defined as the expected fraction of instances coming to a node. For convenience and without loss of generality, in this paper, we only discuss binary split decision trees because it is proved that any decision tree with multiple split can be uniquely transformed to an equivalent binary split decision tree [84]. In this section, we first describe feature computational cost and the expected computational cost for a decision tree. Then we give a brief description to the cost-efficient variants of the C4.5 trees by Quinlan [78].

3.1 Background and Related Work

There are three categories of major problems in machine learning: supervised, unsupervised, and semi-supervised. A supervised learning problem, or classification problem refers to the learning of concepts from a set of labeled examples. The label could be binary (0/1) or a set of distinct multipe labels. Such labels are usually referred to
as “Ground Truth” and in many scenarios, the labels are obtained manually. There are usually two phases, training and testing, in constructing a supervised classifier. Training is the process of building the concept (or model) of the classifier and testing, or prediction, refers to assigning a label (labels) to unseen examples based on the concept learned. The goal of supervised learning is to construct a concept which makes accurate predictions on unseen examples. Support Vector Machines (SVMs) and Decision Trees (DTs) could both be supervised learning methods.

In unsupervised learning problems, while there are still examples, the examples are not labeled. And the objective of unsupervised learning, or clustering, is to construct models/concepts which correctly put the examples, either seen or unseen, into the same category they belong to. Popular unsupervised machine learning algorithms include K-Means and Hidden-Markov-Model, etc.

Besides supervised and unsupervised learning problems, semi-supervised learning has become more and more attractive to the research community. In the setting of semi-supervised learning, there is a small set of labeled examples while lots of unlabeled examples are available. The unique characteristic of semi-supervised learning is that, by combining a small labeled dataset and a large set of unlabeled examples, the constructed model can make accurate predictions for more difficult problems.

One of the biggest practical concerns for applying machine learning techniques in practice is the computational cost. The computational cost of running a classifier online has two major contributors: i) information extraction from the unseen test instance, e.g. lab tests, parsing, remote database lookups, etc. ii) running the decision function or algorithm using the test and training information. Pre-computing models or summaries of the training information helps reduce the cost of the latter contributor. Since the unseen instances do not yield to pre-computation as much, feature extraction remains the bottleneck of computational cost of on-line classification. In order to fit the feature extraction costs to a computational budget, it is common to employ feature selection
which chooses a suitable subset of the features to be used with a traditional classification method. The number of features or the total budget of the feature extraction is an adjustable parameter in feature selection which we can use to tune the trade-off between the accuracy and cost of classification. As the number of selected features gets smaller, the computational cost of feature extraction gets lower, and depending on the information content of the dismissed features, classification accuracy gets lower as well. Traditional classification algorithms commonly require the extraction of a fixed set of features for all the training and testing instances, and in order to operate under computational budget limitations, they need to employ feature selection to reduce the size of this feature set.

Tree classifiers are naturally cost effective since only the feature values along the prediction path from root to leaf node are required. These test costs are incurred by feature extraction and measurement, and are usually measured in computing time, storage or power use. Earlier work [66] [30] [29] focused on minimizing the error weighted by misclassification cost, where the cost refers to the penalty incurred by misclassification. There is also research work [17] [77] [56] aiming to construct tree classifiers with low worst or average testing cost. Most of them are based on single ID-3 like tree classifiers. The heuristic is usually in the form of the entropy gain over the feature cost.

Individual tree classifiers are not very accurate and are susceptible to problems such as overfitting [85]. Various techniques are developed to improve the accuracy of tree classifiers [78]. AdaBoost [96], is a meta-algorithm that is constructed from various individual trees. The idea behind the adaptive boosting technique is to combine diversified weak hypotheses into a more accurate hypothesis. Breiman originated the meta-classifier Random Forest [64], which performs better than Adaboost on noisy data by utilizing bagging data and random feature selection. Both of these meta-classifiers are trying to combine different weak classifiers into a more robust and accurate hypothesis. There are also research papers that try to combine tree classifiers with genetic algo-
3.2 Preliminaries About Cost Effective Classifiers and Decision Trees

Computational cost associated with features can be measured in computing power, memory occupation and cpu time. It is common that a more computationally expensive feature usually provides more discrimination power in classifying ambiguous cases than a cheap feature [56]. The question is whether the high computational price paid for this expensive feature justifies the gain in discrimination power [56]. In reality, asymmetry usually exists in the instances. Some instances are easy to classify while others might be hard to differentiate. This phenomenon illuminates that we can choose cheap features to classify the easy instances, and put more computing power into those ambiguous instances where expensive features provide more discrimination information. With this approach, tree classifiers have an implicit advantage as decision trees are naturally hierarchical. The decisions given by a tree classifier are incremental while other state-of-the-art classifiers, such as Support Vector Machine (SVM), gives a decision only after calculating all the feature values of each instance. In this sense, tree classifiers are internally more efficient while still maintaining reasonable accuracy.

The expected testing cost for a decision tree can be defined as:

\[
Cost(T) = \frac{\sum_{i=1}^{N} PathCost(i)}{N}
\]  

(3.1)

where \(PathCost(i)\) is the total computational cost of instance \(i\) incurred along its path to the leaf and \(N\) is the total number of instances. If all the feature costs are independent and there is no overlapping of the feature costs, \(PathCost(i)\) can be written as:

\[
PathCost(i) = \sum_{j \in \{x_{k_1}...k_i\}} c(j)
\]  

(3.2)
where \( c(j) \) is computational cost associated with feature \( j \), and the sum is taken over all the \( k_i \) features \( x_{k_1}, ..., x_{k_i} \) along the path.

ID3 and its successor C4.5 tree families \([71]\) \([78]\) take an information theoretic approach to choose the feature for splitting. They use Shannon’s Entropy, defined as:

\[
H = - \sum_i p_i \log_2 p_i
\]

to design the heuristic. In ID3, the feature that yields the maximum information-gain between the parent and its children, defined as:

\[
I_k(C_k; X_k) = H(C_k) - H(C_k | X_k),
\]

is chosen as the feature in that node. In C4.5, the GainRatio, which is the entropy gain normalized by the entropy of the feature, is used as the split criterion. In both ID3 and C4.5, the trees are built using a top-down approach.

There are variants of C4.5 trees which take attribute costs into consideration. Examples are EG2 \([77]\), CS-ID3 \([71]\), and IDX \([71]\). Most of these cost efficient tree classifiers apply a greedy algorithm and use heuristics to find the optimal feature to use at each node. The feature that gives the maximum heuristic value is chosen as the feature of the current node. The EG2 uses heuristic:

\[
\frac{2 \triangle I_i - 1}{(c(i) + 1)^\omega},
\]

where \( c(i) \) is the cost of the feature, \( \omega \) is a constant parameter and \( \triangle I_i \) is the entropy gain, the same as the entropy gain in C4.5. The heuristic used in building CS-ID3 tree is: \( \frac{\triangle I_i^2}{c(i)} \). And the IDX tree uses a similar heuristic: \( \frac{\triangle I_i}{c(i)} \).

One common limitation among those trees is that they do not take the expected number of instances at each node into consideration. But in reality, it is highly possible that we have different cost preferences at nodes with different sizes. At the root node (i.e. the biggest node), where each unseen instance needs to be tested against the root feature, we want the root feature to be very cheap and efficient. At a deep level node, where few instances enter, we can tolerate using features with expensive cost for better discrimination power. Therefore, we need a new heuristic, which takes into account both the original feature cost and the frequency information (how many of the
examples are using this feature). Then using this heuristic, we can build better cost efficient tree classifiers. In the next section, we will present a new heuristic using our original “Suppressed Cost”, where the sensitivity of the cost is suppressed as the tree grows to the bottom leaves.

As individual trees may not be accurate, meta-classifiers are built to improve the generalization accuracy. Among those meta-classifiers, bagging and boosting are the most popular and effective ones. They are meta-classifiers constructed from a pool of individual classifiers [28]. Brieman’s Random Forest [64] applies the bagging method, where each individual tree is constructed from N examples sampled with replacement from the original training data. Because of randomization, it is possible that the same instance might be sampled more than once in building the decision tree. A decision forest is a combination of these decision trees, and the final decision for one unseen instance is the class that wins the most votes from the unit vote of the individual trees. Random Forest has many advantages, such as robustness and insensitivity to overfitting [64]. And the accuracy on the out-of-bag data gives an unbiased estimation of the prediction accuracy.

Another meta-classifier is boosting forest [96], which Breiman called ’the best off-the-shelf classifier in the world’. Initially in boosting, every instance has an equal weight. After a classifier is built in one iteration, the weight for each instance is updated and in the next iteration the classifier will be constructed from the new distribution of data. Misclassified instances get more weight in the next iteration and the correctly classified instances are given less weight. Boosting is very successful in binary classification, and can also be applied to multiple classes.

3.3 CoCoST: Building the Universal Classifier from the Specialized

CoCost is a novel cost-sensitive classifier that combines cost-aware decision trees, each of which specializes in cost-efficiently classifying a subset of the input space, with
a cost-aware meta-classifier to achieve high accuracy and low expected classification cost on the overall input space.

There are three key concepts in building the CoCoST classifier: Suppressed Cost, Inverse Boosting and Stacked-Generalization for the meta-learning. We describe each concept in detail in this section.

### 3.3.1 The Base Classifier: LASC Tree–Look Ahead Tree with Suppressed Cost

LASC trees have a greedy tree construction algorithm like most traditional decision trees. There are two discriminating properties in LASC trees as the name implies: i) Suppressed Cost Heuristic and ii) Look Ahead Entropy Gain. Suppressed Cost brings into consideration the estimate of how often a decision tree node will be visited during classification of a large number of test instances. It is gathered from the number of training instances that are being considered during the recursive greedy tree construction step. Look Ahead Entropy Gain enables us to find possible combinations of features that provides the largest discriminating power.

Suppressed Cost Heuristic is defined as:

\[
H = \frac{\triangle I}{freq^\alpha C + (1 - freq^\alpha)}
\]

(3.4)

Here, \(\triangle I\) is the entropy gain of the possible component that is being considered, and \(C\) is the normalized cost of extracting it. \(freq\) is the fraction of training instances that have followed the path from root of the decision tree to the current node. The exponent \(\alpha\) is the parameter for controlling the sensitivity of the heuristic to feature extraction cost. We will first discuss how Suppressed Cost Heuristic works, and then describe how Look Ahead Entropy is used to improve the trees.

**Suppressed Cost**

The most important feature for our tree construction heuristic is that its sensitivity to
feature extraction cost is suppressed when we are adding nodes to the tree that are far from the root node. It is built on the observation that the expected number of instances that will be processed by an internal node decreases as we move down the tree, hence more expensive features are allowable.

In Equation [3.4] for any fixed $\alpha$, when we are near the root level and $freq$ is almost 1, $freq^\alpha C$ will be dominating while $1 - freq^\alpha$ will be near zero. Therefore, the most efficient combination of features will be chosen at the root levels. As the tree grows, the instances remaining at the newly-grown nodes are fewer. The $freq^\alpha$ will become smaller, resulting in less influence from the feature extraction cost on the choice of features. Meanwhile, the item $1 - freq^\alpha$ will get larger. Noting that this item is a constant with respect to the cost, the larger this item is, the less sensitive the heuristic will be. At the bottom nodes towards the leaves where $freq$ is almost zero, the Suppressed Cost Heuristic will just select the features that provide the biggest entropy gain, i.e. the features and/or leaves that provide the largest discriminating power, which is commonly used in construction of ordinary decision trees.

The parameter $\alpha$ is to control the speed at which the effect of cost is "suppressed". The smaller this parameter is, the slower the sensitivity to cost is suppressed. Take two extreme cases for example. If $\alpha$ is very small, say 0.01, even a very small frequency value will still yield a big value of the item $freq^\alpha$, which makes the item $freq^\alpha C$ more important than the item $1 - freq^\alpha$ and results in a very sensitive choice of features due to the cost $C$. If $\alpha$ is large, say 10, even if the frequency is near 1, the quantity $freq^\alpha$ will be very small and cause the effect of the cost $C$ to be much less significant. It has been observed that, for a large $\alpha$, after the root node, the heuristic $H$ will choose the features that give the maximum information gain because the denominator will be almost constant.

$\Delta I$: The Look Ahead Entropy Gain for all possible components in building LASC

In C4.5, the calculation of entropy gain in the heuristic only involves one node. How-
Figure 3.1: Five Possible Components of the One Step Look-Ahead Tree with Suppressed Cost (LASC Tree)

ever in LASC, as we enlarged our search space significantly by exhaustively looking ahead one step, the possible component chosen by the heuristic may consist of two levels of nodes, as shown in Figure 3.1. If the current frequency is so small that there is not much statistical support of the entropy gain, a leaf node could also be chosen, which corresponds to the case e) in Figure 3.1.

The tree is grown recursively adding the “fittest” component which has the largest heuristic value (H value) among the five types of possible components. To prevent overfitting for individual trees, pre-pruning is performed when necessary so that the tree stops growing if there are too few instances.

Experiments show that the look ahead enables us to find good combinations of two features that might not be found by a node-by-node search. The price we pay is that the complexity of searching the component is now $O(n^3)$ with respect to the number of features. The advantage of looking ahead is discussed in [33]. It is worth mentioning
that unless we take feature extraction cost into consideration, even though it is highly possible that looking ahead will yield better trees (better accuracy and smaller size), it may also result in trees with pathology (bigger tree and lower accuracy).

### 3.3.2 Building the Specialized: Boosting and Inverse Boosting

The idea of boosting is to iteratively focus on the hard instances, which cannot be correctly labeled by the classifiers in hand, so that a lower error rate will be achieved if all the classifiers of the iterations are combined. When decision trees are used as base-classifiers in boosting, these highly specialized trees tend to have larger size, and they incur larger test costs as well, when they make use of expensive features.

Inverse boosting was originally invented to create ensembles of classifiers with high diversity [34, 62]. However, when combined with cost efficient classifiers, inverse boosting can be utilized to obtain more specialized classifiers. We propose the technique of inverse-boosting to train classifiers that are more specialized on the easy part of the data. Since this relieves the tree construction from the obligation to correctly classify harder instances, inverse-boosted trees are smaller, and they make errors by bending the stick to the other side. This is especially useful in practice when there are asymmetries in the difficulty of classifying instances. Here by asymmetry we mean that some instances are more difficult to classify than other instances. Note that difficulty of classification might be typical among instances of specific classes, or it could be for instances that are near a decision boundary; The algorithm does not treat the two cases separately.

The idea of inverse-boosting is to iteratively generate computationally cheaper classifiers specialized on the easy instances, i.e. the instances that have already been classified correctly by the classifier at the previous iteration.

In standard boosting, given the weights $D_t(i), i = 1 \ldots M$ and the error rate $\epsilon_t$ of the
current classifier, the new weight is updated as:

\[
D_{t+1}(i) = \frac{D_t(i) \exp(-\beta_i I(y_t(i) = h_t(\vec{X}_i)))}{Z_t}
\]  

(3.5)

where the indicator \( I \) selects those correctly labeled instances from the previous iteration,

\[
I(y_t(i) = h_t(\vec{X}_i)) = \begin{cases} 
1, & \text{if } y_t(i) = h_t(\vec{X}_i) \\
-1, & \text{otherwise}
\end{cases}
\]

(3.6)

and \(-\beta\) reduces their weight at each iteration so that the classifier focuses on incorrectly labeled instances.

\[
\beta_t = \frac{1}{2} \frac{1 - \epsilon_t}{\epsilon_t}.
\]

(3.7)

Here \( Z_t \) is the normalizing factor. In inverse-boosting, we want to focus on the 'easy' part of the data. Therefore, the new weight for each instance is obtained by simply changing the sign in front of \( \beta \):

\[
D_{t+1}(i) = \frac{D_t(i) \exp(\beta_i I(y_t(i) = h_t(\vec{X}_i)))}{Z_t}
\]

(3.8)

Note that Equation 3.8 is not the only way to reassign the weights in inverse-boosting, and it does not have the statistical meanings of the Equation 3.5 in standard boosting. An empirical result is presented in the next section that shows how accuracy of inverse-boosting converges with different choices of re-weighting functions.

3.3.3 Combine the Specialized into a Meta-Classifier

CoCost uses a cost-sensitive meta-classifier to combine the classification power of different trees which are specialized at different types of instances. As mentioned in the previous section, we can create such trees from boosting and inverse-boosting LASC trees. These inverse-boosted trees tend to be inexpensive and accurate on the easy part of the data, and the standard boosted trees will be more expensive as they are trained to
focus on the hard instances. CoCost judiciously combines these specialized LASC trees to construct a more accurate meta-classifier that works well on all types of instances.

The way we generate the meta-classifier is to use the 'Stacked Generalization' method. The decision of each boosted and inverse-boosted tree on the validation data becomes a new feature in the meta-classifier, and the computational cost of each new feature is the expected computational cost of each tree. Eventually in the meta-classifier, each feature variable is actually one boosted tree. Note that the number of "features" (or trees) that the meta-classifier processes might be arbitrarily large, since it is simply the number of base-classifiers that we build.

CoCost’s way of building the meta-classifier is also to use the cost sensitive LASC tree (with $\alpha=1$). The LASC tree meta-classifier allows us to choose a unique set of specialized trees in order to identify each test instance. In the next section, we present experimental results that demonstrate the advantages of using CoCost’s meta-classifier.

### 3.4 Experimental Results

We test our algorithm in a network flow detection scenario. The objective is to classify the underlying network traffic data type from the statistical features of the network package payload. The advantage of avoiding looking into the headers is that it is easy to forge the file headers but it is almost impossible to change the underlying statistical characteristics of the file content. The only way to obscure the statistical features of the file is to encrypt.

#### 3.4.1 Data and features

There are eight data types in this application–TXT, BMP, WAV, JPG, MP3, MPG, ZIP and ENC. The feature set we used comes from a Network Abuse Detection system called Nabs. The features are generally the statistical characteristics of the network flows, such as mean, variance, and entropy. A detailed feature description can be found
Table 3.1: Computational Prices (milliseconds) for Each Feature with Different Buffer Size

<table>
<thead>
<tr>
<th>Feature</th>
<th>16K</th>
<th>8K</th>
<th>4K</th>
<th>2K</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entropy</td>
<td>2.381</td>
<td>1.521</td>
<td>0.734</td>
<td>0.802</td>
</tr>
<tr>
<td>Mean</td>
<td>0.548</td>
<td>0.271</td>
<td>0.217</td>
<td>0.094</td>
</tr>
<tr>
<td>Variance</td>
<td>3.589</td>
<td>1.824</td>
<td>0.904</td>
<td>0.661</td>
</tr>
<tr>
<td>Autocorrelation</td>
<td>42.27</td>
<td>18.37</td>
<td>10.46</td>
<td>5.358</td>
</tr>
<tr>
<td>Mean, Var, Power and Skewness in the first $\frac{1}{4}$ frequency band*</td>
<td>1.614</td>
<td>0.939</td>
<td>1.002</td>
<td>1.543</td>
</tr>
<tr>
<td>Mean, Var, Power and Skewness in the second $\frac{1}{4}$ frequency band*</td>
<td>1.251</td>
<td>0.778</td>
<td>0.912</td>
<td>0.979</td>
</tr>
<tr>
<td>Mean, Var, Power and Skewness in the third $\frac{1}{4}$ frequency band*</td>
<td>1.097</td>
<td>1.007</td>
<td>0.907</td>
<td>0.967</td>
</tr>
<tr>
<td>Mean, Var, Power and Skewness in the fourth $\frac{1}{4}$ frequency band*</td>
<td>1.591</td>
<td>1.204</td>
<td>1.193</td>
<td>1.185</td>
</tr>
<tr>
<td>Skewness</td>
<td>13.94</td>
<td>6.668</td>
<td>3.558</td>
<td>1.536</td>
</tr>
<tr>
<td>Kurtosis</td>
<td>13.203</td>
<td>6.491</td>
<td>3.746</td>
<td>1.671</td>
</tr>
</tbody>
</table>

In Nabs, the features are extracted from a chunk of 16K data randomly sampled from the network flow. We expanded the 22 features by randomly sampling chunks of 2K, 4K, 8K, and 16K from the network flow and recorded the average computational cost for the features in terms of computing time. The computational costs for these features are shown in Appendix A. In particular, there are overlapping costs in different features. All the frequency domain features (labeled by an * in Table 3.1) require the computation of an FFT from the sampled data. This implies that, if one of the frequency domain features is calculated and the FFT result is calculated and stored, then the cost for other frequency domain features will be lowered greatly. This important character of the feature costs give tree and forest classifiers an extra advantage. Trees give decisions incrementally and the overlapping of the feature costs decreases the average cost of one instance.
The raw data we used consists of files we arbitrarily downloaded from the internet. There are 1200 files, each type with 150 files. We choose an uniform distribution of the data types because we do not differentiate between the importance of various data types and assume that they are equally important. The only constraint on these files is that the file should be bigger than 50K, which makes the random sampling of different data chunks meaningful. We put 2/3 of these files as training files where training instances are extracted, while the remaining 1/3 are used as testing files. We extracted 2400 data instances from 800 files, each type with 100 different files. The remaining 400 test files are used to provide 1200 testing instances. The reason for choosing 2400 instances for training is that the learning curve (Figure 3.2) shows that the accuracy (on the same testing data) gets saturated when it is more than around 1500 training instances. And we use more than 1500 instances because we can reserve a proportion of the training data as validation data in building a meta-classifier.
3.4.2 Performance of different base classifiers

Performance Comparison on Random Feature Selection

First, we compare the performance of our base classifiers, the LASC Tree with Suppressed Cost with the C4.5 trees and the state-of-the-art SVM classifiers. We use the public SVM library [48] and choose the RBF kernel. To get a general view of the possible performance of the tree classifiers, we randomly pick up a subset of all the features, and build each classifier from these features. The parameters for SVM are chosen using a 10-fold cross validation on the training set. The performance of a classifier is the expected computational cost of one instance, and the weighted accuracies on all the target types. We do not take model building time into consideration because the training is performed offline.

As shown in Figure 3.3, 5 to 88 different features are randomly chosen from all the 88 features, and to generate more diverse tree classifiers, we injected randomness into these trees as follows: at each node of any tree, only a random half of all the selected features are available for building the tree. At each feature set, the plus points are the performances of C4.5 trees while the star points are the performances of LASC Trees with suppressed cost. Note that $\alpha$ is set to a median sensitivity of 1, and the effect of this parameter is discussed later. It is obvious that through building trees computational cost efficiently, we can get much lower cost at the same accuracies. The only drawback is that LASC Tree cannot get the highest accuracy 88.75% achieved by SVM using all the features at a very expensive cost around 200ms.

Performance Comparison on Deterministic Feature Selection

The performance of random feature selection provides a picture that through cost efficiently building classifiers we can save significant computational cost. However, it is notable that the number of all the possible feature combinations out of 88, $\sum_{i=1}^{88} C_{88}^i$, is
Figure 3.3: Performance of base classifiers with Random Feature Selection. The lines are envelop of the performance points of the three classifiers. It shows that LASC Tree with Suppressed Cost is more cost-efficient than C4.5 and SVM.

an astronomical number, which we cannot cover all the possibilities using random feature selections. Therefore, we need to deterministically choose a set of features from the 88 features.

With SFS (Sequential Forward Feature Selection) [88], we can choose the features with the most discrimination information. Taking the cost of each feature into consideration, the feature is re-ordered by its discrimination score over its cost. Table 3.2 provides the performance of the LASC Tree and other classifiers at different subsets of features.

It can be seen from the table that among all the classifiers, the LASC Tree usually gives the best performance but the difference in accuracy diminishes as bigger feature sets are used. CS-ID3 and EG2 trees perform favorably over IDX trees on larger feature sets, and CS-ID3 and IDX generate identical trees when using 6 and 7 selected features because the two similar heuristic function $\frac{\Delta I_i}{C_i}$ and $\frac{\Delta I^2_i}{C_i}$ choose the same features with certain feature sets.
Table 3.2: Performance of different base classifiers on different sizes of features sets chosen by Sequential Forward Feature Selection. $\alpha = 1$ for LASC Tree. The measures for performances are accuracy(%) and cost(ms)

<table>
<thead>
<tr>
<th>Buffer Size(KB)</th>
<th>2KB</th>
<th>2KB,4KB</th>
<th>2KB,4KB and 8KB</th>
<th>2KB,4KB, 8KB and 16KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. Of Features</td>
<td>6</td>
<td>7</td>
<td>14</td>
<td>21</td>
</tr>
<tr>
<td>LASC</td>
<td>80.45%</td>
<td>81.44%</td>
<td>82.51%</td>
<td>82.70%</td>
</tr>
<tr>
<td></td>
<td>3.167</td>
<td>3.279</td>
<td>3.878</td>
<td>8.852</td>
</tr>
<tr>
<td>SVM</td>
<td>78.26%</td>
<td>79.34%</td>
<td>79.59%</td>
<td>81.44%</td>
</tr>
<tr>
<td></td>
<td>5.470</td>
<td>6.449</td>
<td>12.60</td>
<td>20.35</td>
</tr>
<tr>
<td>IDX</td>
<td>80.37%</td>
<td>80.61%</td>
<td>80.53%</td>
<td>82.11%</td>
</tr>
<tr>
<td>CS-ID3</td>
<td>80.37%</td>
<td>80.61%</td>
<td>81.95%</td>
<td>82.03%</td>
</tr>
<tr>
<td></td>
<td>3.367</td>
<td>3.199</td>
<td>4.208</td>
<td>8.753</td>
</tr>
<tr>
<td>EG2</td>
<td>78.35%</td>
<td>80.54%</td>
<td>81.43%</td>
<td>82.12%</td>
</tr>
<tr>
<td></td>
<td>3.098</td>
<td>4.929</td>
<td>5.133</td>
<td>10.23</td>
</tr>
<tr>
<td>DL8</td>
<td>80.50%</td>
<td>81.82%</td>
<td>83.58%</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>19.16</td>
<td>6.55</td>
<td>9.170</td>
<td>—</td>
</tr>
<tr>
<td>C4.5</td>
<td>79.50%</td>
<td>80.14%</td>
<td>83.58%</td>
<td>82.64%</td>
</tr>
<tr>
<td></td>
<td>19.34</td>
<td>5.197</td>
<td>9.170</td>
<td>14.38</td>
</tr>
</tbody>
</table>

3.4.3 Performance of CoCoST Versus Random Forest

In this section we compare the performance of CoCoST with Random Forest. We used the LASC Tree as the base classifier for both Random Forest and CoCoST. To make Random Forest more efficient, short-cutting is applied in the unit vote. For example, if among 5 trees, the first 3 of them agree on the same decision, there is no need to calculate the cost for the other two and the cost is saved.

As the number of trees increases, the accuracy of Random Forest quickly converges and the cost incurred is larger. Efficiency of Random Forest decreases quickly as the number of the tree members in the forest increases. With many trees in the forest, the accuracy saturates to the limit but the cost increases approximately linearly. Also with different $\alpha$ values, the accuracy of the forest increases as $\alpha$ increases. We tried Random Forests that consist of different values of the sensitivity parameter $\alpha$. The best result is recorded as the benchmark for evaluating CoCoST.
Table 3.3: Choosing different values of $\alpha$ will cause different cost sensitivity suppression

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>Accuracy(%)</th>
<th>Cost(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>77.97%</td>
<td>3.115</td>
</tr>
<tr>
<td>0.2</td>
<td>78.24%</td>
<td>4.241</td>
</tr>
<tr>
<td>0.4</td>
<td>79.13%</td>
<td>4.940</td>
</tr>
<tr>
<td>0.8</td>
<td>81.29%</td>
<td>6.558</td>
</tr>
<tr>
<td>0.95</td>
<td>81.78%</td>
<td>6.384</td>
</tr>
<tr>
<td>1.0</td>
<td>82.51%</td>
<td>7.625</td>
</tr>
<tr>
<td>1.5</td>
<td>83.34%</td>
<td>11.58</td>
</tr>
<tr>
<td>2.0</td>
<td>83.93%</td>
<td>18.55</td>
</tr>
<tr>
<td>4.0</td>
<td>84.73%</td>
<td>33.10</td>
</tr>
</tbody>
</table>

Changing the Sensitivity to Cost by Tuning $\alpha$

The parameter $\alpha$ in our base classifier controls the trade-off between accuracy and cost. It determines how fast the sensitivity to cost is "suppressed" as the tree grows. Table 3.3 shows how the accuracy and cost will change as the parameter $\alpha$ changes. The classifiers are built on the set of 14 features selected by SFS. As $\alpha$ increases, the tree tends to ignore the cost and just picks up the features that provide more significant entropy gain. Therefore, with big $\alpha$ values, such as 2 or 4, the accuracy increases along with the cost. On the other hand, when $\alpha$ is small, say 0.1, we obtain a tree that is very cheap but has a lower accuracy. A good point for this cost accuracy trade-off is to set $\alpha$ to 1 when we want to use a single tree as the classifier.

Performance of CoCoST

For the CoCoST meta-classifier, we boosted and inverse-boosted 10 trees for each $\alpha$ appearing in Table 3.3. Figure 3.4 shows the performance of inverse boosted trees with different re-assigning weights. The individual boosted tree’s total accuracy actually decreases as we put more focus on the misclassified examples.

Depending on how we update the inverse-boosting weights, the inverse-boosted tree points are plotted in different tracks in the picture, each track representing a path of inverse-boosted trees using a different weighting function. Track 1 is the inverse-boosting we suggested in Section 3, while for Track 2 and 3 we tried the square and
the square root of the reweighting function in Track 1. It can be seen that the only difference of these tracks is the speed they converge. The arrows in the graph shows that difference of accuracies at same costs will finally decrease to a very small value.

Because we only want trees that are specialized in a local part of the data, we discarded the first 9 trees and only kept the last tree in building the meta-classifier. Finally we have 16 candidate trees, boosted and inverse-boosted by the 8 different values of $\alpha$. Their decisions on the validation data become the new training data on the meta-classifier, and the expected cost of each tree is now treated as the new price of the features. We again use our base learner to build the meta-classifier and set $\alpha$ to 1 in this meta learning.

The tree structure of the meta-classifier is plotted in Figure 3.6. The accuracy of the classifier is 89.51% with an expected cost for each instance of 14.63, which is more than 10 times cheaper than the SVM classifier with an accuracy of 88.75%. It is also much better than the most accurate Random Forest we have ever obtained. The comparison is shown in Table 3.4.

The final CoCoST meta-classifier expressed our idea that each tree is specialized at its own local part of the data, and combining them we can get very good results. The
Figure 3.5: Compare the Frequency-Cost distribution between CoCoST and Random Forest. Noting that the distribution for SVM is only one bar as all the instances have the same cost.

Table 3.4: Comparison among CoCoST, SVM, and the best Random Forest ($\alpha=1$, consisting of 20 trees)

<table>
<thead>
<tr>
<th></th>
<th>CoCoST</th>
<th>SVM</th>
<th>Random Forest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>89.51%</td>
<td>88.75%</td>
<td>86.53%</td>
</tr>
<tr>
<td>Cost</td>
<td>14.63</td>
<td>189.2</td>
<td>40.11</td>
</tr>
</tbody>
</table>

Raw types (TXT, BMP and WAV) can be determined by the cheap inverse-boosted trees, while the compressed types (ZIP and ENC) are usually decided by two of the specialized boosted trees. Also the CoCoST has nice properties of generating classification rules that are interpretable. For example, the left most TXT decision presents the rule that if two inverse-boosted trees, which are already cheap and accurate at the raw types, agree that it is a TXT instance, then it is precise enough to just give decision of TXT and there is no need to classify the instance using other trees. As we build the meta-classifier also cost-efficiently, it is shown in the graph that larger $\alpha$ values only appear at the bottom. For example, the largest $\alpha$ values in the structure are 1.5 and 2.0. And they only appear after a series of trees with smaller $\alpha$ values have been tested. These trees are selected to classify the most confusing part of the data.
**Cost Distribution: CoCoST versus Random Forest**

Noting that the averaged cost does not tell the whole story, the frequency distributions for each testing instances in CoCoST and the best Random Forest are plotted in Figure 3.5. It shows that CoCoST has a distribution of two peaks, where the easy and difficult instances are clustered. However, for Random Forest, the distribution is more like uniform because there is no mechanism allocating the cost. This graph explains why CoCoST performs much more efficiently than ordinary classifiers. Because we do not know where the incoming instance may lie in the spectrum, Random Forest is more likely to incur more cost in classifying this instance. However, if we use CoCoST, the possible testing cost for the instance can only be around the two peaks of the cost distribution.
Chapter 4

Mining Generic Network Community Structures

The two community detection problems we are solving, P2P community detection and web semantic topical community detection, share the same characteristic that the traffic graph is bipartite in nature since only traffic across the boundary is observed. Given the bipartite nature of the fully observed network IP-IP and the User-Web access graph, as well as limited seed information, we propose to apply LDA as a topic modeling tool to solve the community detection problem. In the context of our problem, the LDA model formalizes the intuition that peer-to-peer hosts or web users visit each other for a ”topical purpose”. The idea is that, if we were to treat users and websites as nodes 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 topical interest. The rationale for this is that there are likely to be many more links between members of the same topical community in the peer-to-peer graph and web-visiting graph, than between two different communities.

Imagine that each peer-to-peer flow or web-access record between an internal host and an external host(website) was produced by an imaginary two-step process (see Fig 4.2): each internal host first picks, at random, one peer-to-peer/topical community to participate in one of $K$ possible communities, and then given the chosen community, an external-host(website) that belongs to that community is picked, also at random.
This two step process is repeated for each web-access network record, 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 websites. 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 our observed web-access activities?”, gives us the distributions, from which we can infer a grouping of hosts into communities. In the scenario of P2P community detection, the above described LDA process directly generates clusters including both internal and external hosts. From seed information such as port distribution and payload signature, we can infer which p2p application the discovered community is mostly likely to be. In web semantic topical community detection, where the external host(ip) is treated as external websites, external websites are also clustered into different topical communities together with the users. Then, with content keywords extracted from these websites, we are able to label and explain the topical communities in our results.

4.1 Background and Related Work

Detecting communities in complex networks has been studied in various contexts including social networks, web graphs, biological networks, etc. [14, 20, 24, 35, 40, 46, 65, 76, 83, 98–101]. Roughly speaking, most of the recent community detection algorithms fall into two main categories: (i) Optimization-based methods try to partition a given graph by optimizing an objective function (e.g. modularity [35]) which measures the quality of the resulting community structure. (ii) Statistical inference-based methods employ probabilistic generative models and infer the community structure from the observed data [14, 20, 46, 76, 83, 98–100]. Bayesian inference-based methods generally allow 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 community is really a community. Due to these favorable properties, in this thesis we employ a Bayesian-inference-based method. In fact, the community detection scheme in this thesis has many similarities with the existing inference-based methods [46, 83, 98–100]. One subtle difference is that, in our context, the network flow records form a bipartite graph. More specifically, sensors are deployed only on edge routers and therefore only flows crossing the ISP boundary are observed. That is, none of the flows between two internal hosts or two external hosts is observable. 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.

4.1.1 Peer-to-Peer Community Detection

To detect P2P traffic, several methods based on transport layer statistics and communication patterns have been proposed in [32, 57, 75]. In [50], the authors incorporate various graph theoretic features (e.g. degree distribution, clustering coefficient, etc.) from traffic dispersion graphs. In [58], the authors propose a flow-based traffic classification method. Authors further present a simple algorithm to identify user communities within a given class of traffic where user communities are assumed to form approximate and non-overlapping cliques on the communication graph. This method implicitly requires access to payload data to separate out a class of P2P traffic (e.g. gaming traffic in [58]). In contrast to prior work on P2P traffic detection, our work in this thesis does not require hand crafted traffic features or heuristics. Our proposed scheme relies only on the observation that members of a P2P community communicate with each other, which is essential to any P2P network and cannot be easily avoided. This does not
preclude the use of signature-based methods if a signature is available, but allows us to discover P2P communities that deliberately randomize their signatures in order to remain hidden, as well as new P2P communities for which signatures are not available, such as P2P Botnets.

4.1.2 P2P Botnet Detection:

Clustering based P2P botnet detection methods are explored in [38] and [95] based on the observation that P2P bots exhibit similar malicious activities and similar flow features. A major shortcoming of these methods is that they can only detect P2P bots which exhibit some malicious activity, such as sending spam, performing a port-scan, etc., during the observation period. In [74], the basic idea is that P2P botnets form a faster mixing component than a client-server model on a communication graph, but their validation is limited to using artificial P2P networks embedded into real background traffic. This makes the problem significantly easier because there is only one P2P community to detect. In contrast, our work naturally deals with the presence of multiple different P2P communities that are likely to be part of any realistic large scale network trace. In [25] the authors propose a method to identify other members of a P2P botnet once one member is identified, based on the observation that members of a P2P botnet are likely to share mutual contacts. In [102], the authors first identify P2P botnet traffic based on the heuristic that P2P botnet malware is persistent and stays active as long as the host system is online. The authors cluster P2P botnet traffic using statistical fingerprints derived from flow features in order to separate different P2P botnets. As mentioned above, P2P botnets can potentially randomize flow features and therefore invalidate the statistical fingerprints employed in the above scheme.
4.1.3 Web-Domain Topical Community Detection

Detecting community structures from complex networks has been studied in different contexts such as Social Network [91] [69] [11] [79] [52] [9], Web Graph [70], and Biology [36]. Most of these approaches could be categorized as either modularity-driven [36], where the goal is to optimize a certain objective function given the graph structure, or statistical inference based where a Bayesian Generative Model is applied [54] [44]. Working under the definition of a community which should have better internal connectivities than external connectivities, the objective function to optimize is often related to the cut of a graph, and the problem under this scenario is proven to be NP-hard [63]. Therefore, different heuristics [51] are employed to solve the problem. Meanwhile, there are also approaches using Bayesian Generative Models to tackle the community detection problem [54] [44]. Then the question becomes what posterior settings of members belonging to each community best explains the membership data-observed. For detailed comparison of the community detection algorithms, we point to the survey and comparison in [51].

Studying the community detection problem from an enterprise angle for either talent discover or industry revenue/trend prediction has been proposed by Lin in [93] [27] [97], where the behavior of users within an enterprise is studied and modeled. Users’ behaviors on different social websites as well as content information is utilized in the work [93] to bring predictive value for the enterprise. Community detection has also attracted interest from the social network area. In [42], the authors proposed an LDA approach to model user interaction behavior communities. In [10] [59], researchers demonstrated information leakage could happen for certain user groups. More specifically, by adopting a similar LDA-based approach, they are able to infer demographic information based on user 'likes' on social network sites such as Facebook. In [94] [82], both content and user interaction data are utilized to provide semantically improved clusterings.
4.2 Latent Dirichlet Allocation: An Statistical Bayesian Clustering Algorithm

Now we describe the LDA model as the approach to both the P2P community and web topical community detection problems. The LDA model [23] has been applied to community discovery in social networks [83,98]. We describe it here in our problem’s setting. Consider a data set $D$, consisting of \{source IP, Destination IP/Web\} pairs in a given time interval. If we think of $D$ as a graph $G$ where vertices represent hosts, and undirected edges represent flows between pairs of hosts, then $G$ must be a bipartite graph because of our measurement setup (Section 4.3.1). Each IP/Website in $D$ can be a member of one or more communities. We assume there are $K$ communities in all.

We first define two families of multinomial distributions (see Table 5.1 for a summary of notation):

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

$\beta$: For each community $C_j$, $j = 1, \ldots, K$, consider a multinomial probability distribution $\beta_j$, $j = 1 \ldots K$ over all the external IP/Website(s) in $D$. The dimensionality of each $\beta_j$ is $N_{ext}$, the number of external IP/Websites. The $m^{th}$ element of $\beta_j$ represents the extent to which the $m^{th}$ external IP/Website participates in the $j^{th}$ community.

We first collect all the flows in $D$ involving internal IP $i$ into a dataset $D_i$. Let $|D_i|$ denote the number of flows in $D_i$. We ignore the order in which these flows occur. The behavior of each internal host $i$ in $D$ can be described via a probabilistic generative process as follows (see Figure 4.2 for an illustration of the generative process for
P2P Communities and Web semantic topical ones, and Figure 4.3 for the probabilistic graphical model that represents the generative process described in the steps below. The numbers within parantheses in Figures 4.2 and Figure 4.3 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 4.3, \( \alpha \) and \( \eta \) are parameters of the two separate Dirichlet distributions. We use \( \alpha = 0.3 \) and \( \eta = 0.01 \) based on suggestions in [37, 67].

2. For each flow in \( D_i \), 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, \( (k \in \{1, \ldots, K\}) \), pick an external IP/Website by sampling from \( \beta_k \). Let the external IP/Website picked be denoted by \( H_{i,n} \).

The last two steps above are repeated for each of the \( |D_i| \) flows in \( D_i \), and the set of three steps above is repeated for each internal host in \( D \). Given prior distributions \( \theta_i, i = 1, \ldots, N_{int} \) and \( \beta_j, j = 1, \ldots, K \), and this probabilistic generative process (i.e., how observed data \( D \) is linked to a given set of prior distributions \( \theta_i \) and \( \beta_j \)), our objective is to infer posteriors \( \theta_i, \beta_j | D \). Exact computation of the posteriors on \( \theta_i \) and \( \beta_j \) is intractable. Standard techniques are available to perform approximate posterior

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathcal{D} )</td>
<td>Dataset of {Source IP, Dest IP/Websites} pairs</td>
</tr>
<tr>
<td>( D_i )</td>
<td>Set of all flows involving internal host ( i )</td>
</tr>
<tr>
<td>( N_{int} )</td>
<td>Number of internal IPs</td>
</tr>
<tr>
<td>( N_{ext} )</td>
<td>Number of external IP/Websites</td>
</tr>
<tr>
<td>( K )</td>
<td>Approximate number of P2P communities</td>
</tr>
<tr>
<td>( \theta_i )</td>
<td>Multinomial of dimension ( K )</td>
</tr>
<tr>
<td>( \beta_j )</td>
<td>Multinomial of dimension ( N_{ext} )</td>
</tr>
<tr>
<td>( C_{i,n} )</td>
<td>Community of the ( n^{th} ) flow of internal host ( i )</td>
</tr>
<tr>
<td>( H_{i,n} )</td>
<td>External IP/Website involved in ( i^{th} ) internal host’s ( n^{th} ) flow</td>
</tr>
<tr>
<td>( \alpha )</td>
<td>Dirichlet hyperparameter for producing prior ( \theta_i )</td>
</tr>
<tr>
<td>( \eta )</td>
<td>Dirichlet hyperparameter for producing prior ( \beta_j )</td>
</tr>
<tr>
<td>( \gamma )</td>
<td>Fraction of hosts that are internal</td>
</tr>
</tbody>
</table>

Table 4.1: Summary of notation
Figure 4.1: The probabilistic generative process that explains the dataset of (User, IP/Website) pairs. The numbers in parenthesis correspond to the steps in Section 4.2 and Figure 4.3.

Figure 4.2: The probabilistic generative process for Web-Topical-Community Detection, with the extracted keywords from websites as seed information to annotate the communities.
Figure 4.3: The probabilistic graphical model for LDA. Only the shaded nodes are observed.

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 \([37]\) as our approximate inference algorithm as it can be parallelized over a compute cluster \([67]\). The Gibbs sampling algorithm provides us with 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 IP/Website \(i\) 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 \(\beta_i, i = 1, \ldots, K\). Since internal hosts are connected to external IP/Website by flows, and we have assigned external IP/Website 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 \(\theta_i, i = 1, \ldots, N_{int}\).

### 4.2.1 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 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 4.3.4 we provide precise definitions and evaluate the precision and recall). 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 very few flows going to more than one community.

Given the discovered communities of our algorithm, the remaining problem is how to label and interpret these communities that we have generated. For the P2P community detection problem, we could look into other seed information which we haven’t used in discovering the communities. For example, if one of our detected communities is the BitTorrent Community, it is highly likely that port number 6881 will appear more frequently than other random ports in the port occurrence distribution. Thus we can use information such as port number distribution and payload signatures to label the Peer-to-Peer communities we have detected. For the web community detection problem, we use the mined frequencies of keywords associated with the cluster of websites that users have visited, to interpret our discovered communities. These keywords are filtered to remove stop words without semantic significance. By labeling each user community with a collection of keywords, the keywords and their associated frequencies constitute a description of the ”topic” in this community. By annotating the discovered communities with these mined text keywords, we have decomposed the user-to-website access record into interesting descriptive clusters. Each community will have a ”description”, which is essentially a distribution over the keywords. These keywords somehow represent the popular topic(s) within the discovered community, and enable us to explain and interpret the topics for the community. As a simple example, if we see a community whose keywords are mostly related to a country, then we have probably discovered the demographical community from that country.
4.3 Discovering Peer-to-Peer Application Communities

In this section we validate our algorithm on real-world network-flow data by checking whether the communities inferred by us correspond to real P2P applications. In the absence of any labeled data, this is a tricky problem. Fortunately, some P2P applications have very simple signatures, in particular, well known port number(s) (e.g. BitTorrent). Being able to detect a well-known P2P community like BitTorrent indicates that other clusterings inferred by us (for which we do not know the corresponding P2P application) are also likely to be meaningful. In the following sub-sections, we first describe our dataset and measurement setup, then we perform a more rigorous evaluation by artificially embedding P2P communities of various topologies and connectedness into our dataset, and checking if we are able to discover them.

We then study whether any of the P2P communities we infer are malicious in nature by checking if any of them contain an abnormally large number of hosts that are blacklisted in public blacklists. We indeed find this to be the case, suggesting that we have discovered (at least parts of) some P2P botnets. We find that over the course of 50 days, the fraction of hosts in these communities that are blacklisted grows significantly with time, indicating that we have been able to detect malicious communities earlier than they show up on public blacklists. To the best of our knowledge, we are the first to report on the use of an automatic method to detect malicious P2P activity in the wild.

4.3.1 DataSet and Measurement Setup

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.
To estimate $K$ using our dataset itself, we employ the notion of perplexity \[23\]. 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. In our setting, perplexity is the average negative log-likelihood of
the probability of observing a test-dataset per external host, with an \( \exp \) function applied at the end so as to transform back from the log domain to the probability domain. A detailed mathematical definition of perplexity can be found in [23]. Therefore, we have

\[
\text{perplexity}(\text{Model}(K) | \text{TestData}): = \exp \left(\frac{-\log (p(\text{TestData} | \text{Model}(K))))}{\# \text{of external hosts}}\right) \tag{4.1}
\]

To choose an appropriate \( K \), we compute perplexity using 10-fold cross validation on 10 minutes of our dataset under different \( K \) values. The perplexity curve is shown in Figure 4.4, 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 4.5.
4.3.2 Well known communities

Figure 4.6: (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 which application it is (using [1]).

We now return to the features of each flow (Section 4.3.1 – port number, protocol, duration, and number of bytes and packets – that we did not use in our inference algorithm). Recall that in general it is not possible to identify communities by simply utilizing these features, since communities do not necessarily exhibit (or deliberately hide) recognizable signatures in terms of these features. However, these features can be very useful in evaluating how well our system detects communities because members of a community run the same P2P application and therefore their flows can be expected to have coarsely similar features. With the goal of validating whether the communities inferred by LDA are meaningful, we use these features to answer the following questions: (1) Can we identify some of the communities we infer? (2) For each community we infer, do the hosts have similar flow features?
Using Port Numbers to Label the Communities

In order to identify known communities that use well-known ports, we compute for each port (1 to 65535), the fraction of hosts in a community which use that port at least once. We refer to this fraction as the *popularity of the port*. In computing popularities, hosts at both ends of a flow are considered to be 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 port within a community serves as a strong validator for the P2P application associated with the community.

![Figure 4.7: Entropies of 5 flow features for each of $K = 165$ communities, sorted by entropy (□) along with corresponding entropies of the flows of random sets of hosts (△) equal in number to the size of the community being compared with. The dotted line is the median entropy of discovered communities.](image)

We check whether there are 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 4.3.2. These communities all exhibit at least one featuring port which is well-known and associated with a 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 [1] for a detailed mapping of port numbers to well-known services. We found that there are 123 communities with an 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 4.3.2(b)), which is not very well-known, but in [2], the authors report that port 8247 is associated with a plug-in which users download to watch 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. 4.3.2(d)). There are also several P2P VoIP communities (which typically use port the Session Initiation protocol (TCP port 5060) to setup a call).

Note that many of the peaks in Figure 4.3.2 are close to 1, indicating that almost every host in these communities has had at least one flow on that specific port. Not all peaks are exactly at 1 because of two reasons: (i) The protocol might not require the use of a specific port by every participating host. For example, in the BitTorrent community (Fig. 4.3.2(c)), even though the TCP port 6881 is the default for many applications, users could potentially change this default port, where as in the VoIP community (Fig 4.3.2(f)), it appears that every host uses SIP (port 5060) at least once, and so the peak is almost at 1. (ii) Our inferred communities are not necessarily 100% pure, i.e. not every single host in a community may in fact be a member of that community. In Section 4.3.4 we quantify the purity of communities detected by our algorithm.
Evaluating the quality of inferred communities

With featuring ports for some communities, we are able to infer the underlying P2P applications, but besides these, there are also many communities which either do not have a featuring port, or feature dynamic ports. Given that members of a community run the same P2P application, the flows within 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, the entropy is computed after first quantizing the distribution of that feature into bins, so that the computed 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 compares the community entropies for the 5 traffic features in Figure 4.7. 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.
4.3.3 Tracking suspected malicious communities

One of the goals of detecting P2P communities automatically is to discover new or emerging malicious P2P communities, so that they can be tracked and subsequently mitigated. We now describe our approach to discovering and tracking communities that we suspect to be P2P botnets or portions thereof.

Using the inferred communities in Section 4.3.2, we checked the IP addresses in every community against the Spamhaus blacklist [3] and found three communities of 1867, 2908, and 2273 hosts respectively, that had significantly high fractions of host members blacklisted (see Figure 4.8(a)). Each of these three communities had roughly 10% of their host members blacklisted at the time we inferred communities (Figure 4.8(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. Note that the occurrence of a small fraction of blacklisted hosts in almost all of the $K = 165$ communities in Figure 4.8(a) 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 is that, like 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).
Figure 4.8: (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 an outage on the server running our script.

Next, we tracked these three suspicious communities over time, by repeatedly check-
ing for a period of 50 days, 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 4.8(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 4.8(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.

4.3.4 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, which is impossible to obtain. Therefore, we employ the following approach: We embed synthetically generated communities with various P2P topologies and connectedness into our dataset. The resulting dataset is then fed to our inference algorithm. With this strategy, we are able to not only accurately measure the system’s performance, but also also determine the conditions under which it fails.

4.3.5 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. By embedding synthetic commu-
nities with 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. We embed the edges corresponding to a synthetic community only if they cross the network boundary. This follows from the fact that our observation of the network traffic (see Section 4.3.1) is restricted to the flows that cross the backbone boundary. Therefore, all embedded communities are also bipartite graphs.

**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 IP in common 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,

$$\text{precision}_C(\hat{C}) = \frac{|C \cap \hat{C}|}{|\hat{C}|}; \quad \text{recall}_C(\hat{C}) = \frac{|C \cap \hat{C}|}{|C|}$$

(4.2)

### 4.3.6 Synthetic Community Structures

We measure the precision and recall of detecting synthetic P2P communities generated as per two different random graph models: (i) the Erdos-Renyi (ER) model [31], and (ii) the Barabasi-Albert (BA) model [15]. 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, in the Conficker-C botnet [4], 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. [15]. More specifically, the 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.

4.3.7 Detection Peformance

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 4.9 for various connectivity and activity parameters (i.e. $p$ and $\lambda$ for Erdoes-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 raised above a critical value. We find that the precision and recall curves for ER communities both exhibit a sharp increase as \( p \) increases. This is consistent with our expectations, given the above mentioned behavior of ER graphs with respect to \( p \). 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.

### 4.3.8 Overlap between communities

In general, P2P communities may overlap with each other since some nodes may participate in more than one community during the network flow measurement period. 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 4.10 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 4.10 since the system performs almost equally for A and B when \( q < 75\% \). On the other hand,
when the 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 \( \geq 75\% \).

![Figure 4.10: Detection performance on overlapping communities for different overlap ratios.](image)

### 4.3.9 P2P vs. Client-Server Communities

Despite filtering out most centralized communities in the preprocessing step (Section 4.3.1), the reverse DNS-based filtering may still leave behind some centralized communities which address their servers directly using their IP addresses. Some examples of such services that we came across are web analytics services and page redirec-
and 1,000 internal nodes, but nodes are of two types of nodes: servers and clients. Clients can only talk to servers, while servers can talk to both clients and other servers. To generate a community, we randomly choose one server for each client and generate 30 flows between them, and for each server, we create 30 flows randomly distributed among the other servers. The number of servers controls how centralized the community is, evolving from a purely centralized one (a star), and eventually becoming a pure P2P Erdos-Renyi network as the number of servers is increased. The number of flows generated by a community is constant regardless of how many servers it has, thereby ensuring a fair comparison between different structures. Detection results for these synthetic communities – shown in Figure 4.11 – show that detection is better as the community becomes more P2P-like. This is not surprising, since our inference method favors communities in which any two members share a large number of common hosts that they communicate with. On the other hand, since the members of a more centralized community can only talk to a few common IP addresses (i.e. servers), they are more likely to be omitted by the system.

![Figure 4.11: Detection performance for different number of servers. As the number of servers increases, the community becomes more P2P-like.](image)

### 4.4 Discovering Web Semantic Topical Communities

We now present experimental results showing that our approach can accurately identify topical communities. In addition, we examine and present the characteristics
of several expected communities as well as some unexpected ones.

### 4.4.1 Evaluation Metric and Labeled Dataset from DHCP Log

Given our unique problem setting and viewpoint, obtaining a labeled dataset is difficult for several reasons. First, getting the complete topical interests from users is difficult due to privacy concerns. In addition, each user has his/her own unique topical interests and a reasonably large dataset has to be obtained to cover a large portion of users in order to perform accurate evaluation. Rather than manually obtaining a small labeled dataset by individual user survey, we explored the dhcp log associated with our network access traffic. The dhcp log includes information on which IP addresses have been assigned to each device. A device, which could either be a PC, laptop, tablet or even mobile phone, is identified by either its MAC address or a host name. In our case, we found that many devices actually have meaningful host names associated with them. In many cases, the host name included information of the users’ actual name or information of the office location of the computer. By connecting this information with our roster database, we were able to accurately identify 262 users’ identities. With this information, we constructed a labeled dataset which describes the demographical and ethnic data of these users, as shown in Table 4.2. This dataset then was used as a benchmark for evaluating our approach to measuring performance of identification of ethnic communities. It should be noted that the experiments were done with procedures approved by the university IRB and proper procedures to protect the privacy of individuals were put in place.

### 4.4.2 Performance of the Mined Topical Communities

To evaluate the accuracy of our approach, we ran our clustering algorithm and compared with the ground truth labeled community dataset for the period of time where the corresponding dhcp log was recorded. To pick an appropriate number of communities,
we adapted the notion of perplexity measure as suggested in [22]. Note that the precise number of communities is difficult to compute and remains an open problem. However, in our scenario, a good approximation which enables us to capture most active communities will suffice. In our problem, the number K was tuned to be 40.

Two metrics, precision and recall, were used to measure performance. Let $C$ denote a labeled demographical community and $\hat{C}$ be its detected version. In order to pick up the "detected" version of $C$, we went through every discovered community and chose the one with most "similarity" with respect to $C$. More specifically, for each community, we compute its Jaccard Similarity with the labeled demographical community, and select the one with highest Jaccard Similarity as the "detected" corresponding one. Precision and recall were then computed from $C$ and $\hat{C}$. Precision was defined as the fraction of the members in $\hat{C}$ that actually come from the underlying ground-truth community $C$. Recall was defined as the fraction of members in $C$ that appear in the detected community $\hat{C}$. Formally: \[ \text{precision}_C(\hat{C}) = \frac{|C \cap \hat{C}|}{|\hat{C}|}; \quad \text{recall}_C(\hat{C}) = \frac{|C \cap \hat{C}|}{|C|} \]

<table>
<thead>
<tr>
<th>Community Information</th>
<th>Community Size</th>
<th>Total Number of Web Access Flows</th>
</tr>
</thead>
<tbody>
<tr>
<td>India(IN)</td>
<td>48</td>
<td>6594</td>
</tr>
<tr>
<td>China(CN)</td>
<td>132</td>
<td>10270</td>
</tr>
<tr>
<td>Germany(DE)</td>
<td>132</td>
<td>2787</td>
</tr>
<tr>
<td>Russia(RU)</td>
<td>29</td>
<td>1069</td>
</tr>
<tr>
<td>Italy(IT)</td>
<td>13</td>
<td>385</td>
</tr>
<tr>
<td>Poland(PL)</td>
<td>12</td>
<td>1161</td>
</tr>
<tr>
<td>Turkey(TR)</td>
<td>9</td>
<td>3601</td>
</tr>
<tr>
<td>Japan(JP)</td>
<td>7</td>
<td>104</td>
</tr>
<tr>
<td>Total</td>
<td>262</td>
<td>23971</td>
</tr>
</tbody>
</table>

Table 4.2: Labeled Ground Truth: Demographical Topical Communities

To compare we also implemented the K-Means clustering algorithm [51], where
the distance between two internal users is measured by the Jaccard Similarity between the sets of the external websites these two users have visited. Experimental results on precisions and recall are shown in the bar graph Fig. 4.12, where the height of the bars indicates accuracies and recalls. The results are ordered by the sizes of the ground-truth communities.

From the graph, it can be seen that the LDA-approach performs consistently and significantly better than the K-Means algorithm. More specifically, the algorithm tends to have better performance given more active communities. For example, the Russian demographical topic community has a larger size in terms of number of users. However, the LDA algorithm shows better detection for the Turkish community where the number of captured web-access flow records is larger. Also, even though the algorithm is able to detect most communities, there are certain small communities (such as the Japanese and Italian community), which are not very well detected. This is not surprising given that LDA is a Bayesian algorithm where its performance gets better with more data (essentially more stable statistics). Given the smaller size of these communities and lower activity, LDA will not be able to detect these ones given the inherent “resolution” of LDA’s application in our problem.
To have a sense of how active a community has to be in order to be detected, we conducted experiments on a relatively large community (the Chinese community). By intentionally deleting some web-access flows randomly, we repeated our experiments to see how the portion of remaining flows will affect detection performance. The graph in Fig 4.13 shows how the detection performance changes with respect to the remaining number of web-access records. While precision is slowly decreasing, recall is more affected by the activities of a community as it decreases more quickly as the activity level of the community decreases.

### 4.4.3 Expected and Unexpected Communities

Demographical communities, while being measurable against ground truth, are perhaps less interesting and valuable to an enterprise than other types of subject matter based communities. In addition to comparing with the ground truth communities to evaluate the exact performance of our algorithm, it is also interesting to study the "expected" and "unexpected" communities that were discovered, especially the topic driven ones. Most of these communities are either identifiable by demographically featuring keywords as shown in the domain names, or driven by topical content. We address these as the "expected" communities as we expect our algorithm to be able to discover existing and on-going topics within our institute. Among the 40 communities discovered, around 50% of them (22 communities)
were highly associated with demographical keywords such as a specific country. The remaining communities were more driven by specific keywords such as "linux" and "hack". Note that the results were conducted based on datasets collected from an engineering-focused college. More characteristics of the detected communities are shown in Table 4.3. Given that our NetFlow data comes from a university, where a significant number of students are international students, the demographical communities are mainly driven by nationalities and languages of the visited external websites. More interestingly, for the content-based communities, there are many topical communities which we can interpret from the associated keywords. We present as examples, four communities of "Consumer-Electronics", "Hacker-Security", "Deals-Online-Sales", and "Academia-Education-Admission", along with their most frequent keywords in Fig 4.14. From the keywords associated with these communities, we can infer the topics driving the common interests of the users within these communities.

<table>
<thead>
<tr>
<th>Categories</th>
<th>Total Number of Communities</th>
<th>Total Number of Users</th>
<th>Example Communities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Demographical</td>
<td>22</td>
<td>1227</td>
<td>Chinese</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Indian</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Korean</td>
</tr>
<tr>
<td>Content Based</td>
<td>15</td>
<td>538</td>
<td>Consumer-Electronics</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Hacker-Security</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Deal-Sales-Shopping</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Academia-Education-Admission</td>
</tr>
</tbody>
</table>

In addition to these foreseeable demographical or common topical communities, we also present case studies of two correlated special communities. Both of them have unique keywords distributions making them very self-explanatory. These communities we address as "unexpected".
Two Special Communities: The "Apple-Steve-Jobs-Death" Topic and the "Sales-Deals-Buy-Iphone" Topic.

By coincidence, we built the labeled datasets at about the same time that there were many news reports about the death of Steve Jobs. Interestingly, two related topics, one featuring keywords such as "apple, steve-jobs, death", and the other featuring keywords such as "sales, deals, iphone", both get discovered in our system. The degree distributions, along with the featuring keywords are shown in Figure 5.15. The associated keywords somehow tell the story. There is a popular topic about the death of Steve Jobs while there is another overlapping community discussing buying iphone and Android phones. Interestingly, the degree distribution for the first one is very concentrated, meaning that the death of Steve Jobs is the only dominant topic in this community. However for the "Buy-Iphone" topic, there are two peaks. Looking into the graph
structure of the community, it actually has "iphone" and "android" as two clustering centers, thereby making two peaks in the degree distribution.

Figure 4.15: Degree distribution for the case-studies: (1) Apple-Steve-Jobs community along with featuring keywords (2) The Iphone-Android Sales Community with featuring keywords. Both keyword lists are sorted by their mined frequencies.

4.5 Further Privacy Inference for Fully Anonymized Network Flow DataSet

We have shown in that from network traffic traces, as long as we can connect the external hosts with semantic information such as the text in certain domains, we can decompose and interpret the hidden communities in a network. In practice, it is even more interesting if we can mine privacy information from network flow trace where both the source and destination are anonymized.

4.5.1 Framework and Methodology to Make Privacy Inference

We propose a framework and methodology to make privacy inferences on network flow traces where both the source and destinations are anonymized, which we refer to as "fully anonymized" data. Given that we can always mine the community structures using only the who-talk-to-who traffic as long as the anonymization is consistent,
our idea is to leverage other traffic statistical information to further infer the private information for the fully anonymized data.

Consider a fully anonymized network flow dataset \( D \), and by applying our proposed LDA approach, we have constructed a number of communities, namely, \( C_1, C_2, ..., C_N \) from this dataset. Given that the external IP address are anonymized, it is impossible to interpret the communities with extracted keywords from the websites. However, if the other traffic features, such as flow session length(len), bytes per second(bps), packets per second(pps), and bytes per packet(bpp) are not anonymized, we can still use these pieces of information to make certain inferences about the fully anonymized.

Suppose we have some prior knowledge that certain hosts in the anonymized network may tend to visit a set of websites. Then we can obtain the knowledge of characteristics of the network traffic session associated with visiting each of these websites by actually visiting these websites and log the sessions. The traffic statistics for visiting these seed websites, (denoted as \( S \)), could be viewed as a bunch of Gaussian Distribution of each of the traffic features(len, bps, pps, bpp). Within each generating community \( C_i \), if the traffic, we can treat the traffic features of flows as generated by sampling from the distributions of the traffic features from visiting each membership website. Formally, let’s denote the aggregated feature vector for anonymized community \( C_i \) as \( f_{vC_i} = \text{len}, \text{bps}, \text{pps}, \text{bpp} \). We assume this distribution is generated from traffic from visiting a set of websites specific to this community.

\[
f_{vC_i} = \sum_{w \in W_{C_i}} P(w)f_{v}(w), \tag{4.3}
\]

where \( w \) is the destinations specific for community \( C_i \) and \( P(w) \) is the probability that website \( w \) is visited by a member in community \( C_i \).

In this way, we can decompose each of the aggregated traffic feature within each community to a number of Gaussian traffics. By matching these decomposed Gaus-
sian features with the obtained seed traffic feature, we can map the fully anonymized communities to the prior knowledge we have constructed.

4.5.2 Tentative Proof of Concept

As a proof of concept, we conduct experiments with certain on deal-tracking websites such as dealsea.com, dealnews.com, dealslist.com, slickdeals.com, dealsmoon.com, and constructed the feature vectors of these websites by setting up 10 machines to randomly connecting to these websites and randomly crawl the pages. Then we use the E-M method to decompose the traffic features in each of the communities. As we expected, the "Deal-Sale-Shopping" community has the most similar decomposed components with the traffic features of these seed websites. Therefore, if we do not have any initial knowledge that this community is "Deal-Sale-Shopping" community since both the source and destinations are anonymized, we will be able to infer the "topic" of this community given a proper prior set of seed websites that members in a "Deal-Sale-Shopping" community may potentially visit.

In general, the prior knowledge set of seed websites is very important. However, since it is impossible to have a complete set of the websites in each community, solving the privacy inference problem is inherently a hard problem. We further discuss the possible approaches in Chapter 6.
Chapter 5

Detecting Infection Source and Building Predictive Blacklists with an Attack-Source Scoring System

Our attack-source detection system works on network-flow data and is deployed at the boundary of an institution, thereby naturally separating all the hosts into two categories: the ones inside our network perimeter are referred as “internal” hosts, while the ones outside our boundary are “external” hosts.

From an institute’s point of view, we fully observe the network activities of our internal hosts interacting with the external world, while for any individual external host we can only observe their traffic communicating with inside hosts. Therefore, we have a good picture of our internal hosts’ behaviors interacting with the external world. The idea of our approach is that we believe that benign behaviors are natural and inherent while there are reasons for malicious/suspicious activities. Especially when one internal host begins to exhibit malicious/suspicious activities, the external hosts that this internal host has talked with should somehow be held responsible and subject to an “attack-source score” charge.

In other words, the behavior of the internal hosts is associated with their interac-
tions with the external world. Hence an internal host is affected by the external host(s) he/she talked with and the external host(s) should be somehow responsible for our internal host’s behavior. More specifically, we compute an attack-source score based on such abnormal, suspicious or even malicious “behavioral” events triggered by network activities exhibited by our internal hosts. We quantify such behavior changes into additional attack-source scores and propagate such scores through historical interactions between internal and external hosts. In addition, our system has the following properties:

1. Attack-source scores of external hosts are from $[-1, 1]$, where negative is the malicious side and positive the benign side. Strangers start from a neutral score slightly towards the negative side (-0.1).

2. To bootstrap, IPs from a whitelist \([5]\) start from an initial attack-source score of 0.9. IPs from a blacklist \([6]\), if shown in our traffic, start from a score of -0.9.

3. Attack-source scores gradually “decay” to neutral without evidence of observed traffic.

4. We memorize all the IPs which have hit a certain threshold of significant attack-source score. Negative attack-source scores get forgotten much more slowly than positive ones. If attack-source score downgrade repeats, it gets a more compounded effect from its previous attack-source score;

5. Every time internal exhibited suspiciousness/maliciousness gets detected, the symptoms are quantified as attack-source score charges. And this charge score is subtracted from responsible external hosts to make their attack-source score closer to the negative side(i.e. more likely to be responsible as attack-source).
5.1 Background and Related Work

Even though the generation of the attack-source score in this thesis work is very different from traditional reputational scores, our attack-source scores are still related and to some extent similar to reputation scores in the sense of measuring the “maliciousness”. In the area of network security research, reputation systems [49] [55] have been widely used to model the spread of attacks in a network, in particular because of their power to capture big-picture attacker activities and to provide a uniform view of the threat posed by network nodes [18] [103] [87] [45]. Many existing reputation systems are domain name based and built on various characteristics of DNS (prefixes and suffixes [21], WHOIS registration data [18], DNS server [103]) to determine whether a new name is "in the neighborhood" of a known-malicious name. The underlying logic is inherently proximity based: both malicious and benign hosts stay close to its own peers while keeping a distance away from peers in a different category. Notos [18] is a dynamic reputation system based on domain name identity. It models malicious and benign domain names, and predicts a reputation score for any given domain name. The intuition is that benign and malicious DNS names, when mapped to different address spaces, have some significant differences. Exposure [21] also works entirely on passive DNS query data and focuses on the bursty access nature of malicious domain name services in a temporal perspective. HPB [103] is an IP based blacklisting system. The system generates rankings of offensive IPs for each of the contributing institutes who share security logs. Many other techniques in botnet-detection could also been seen as assigning reputation scores to potential bot members [39] [26]. [55] provides a survey on the attacks and defenses of network reputation systems. Such scoring based reputation systems to measure risk and trust, are also used widely in area such as e-Commerce [41] [53], file-sharing systems [86] [68], and ad-hoc wireless networks [49]. [49] provides a survey of the various applications of reputation systems.
Other works on detection infection/attack-sources have mainly focused on other perspectives. For example, [19] studied the problem of tracing attack-sources on the ISP level by leveraging router logs. In [72], the problem of preventing internal hosts from originating DDoS attacks has been studied.

5.2 The Attack-Source Scoring System for Tracing Maliciousness of External Hosts

To accurately capture any suspicious behavior changes within our network perimeter for any internal host, we utilize the behavior features in Table 5.1 to monitor individual host’s behavior. We compute the distributions of four categories of these features: workday-daytime, workday-offtime, weekend-daytime and weekend-offtime. These features are mostly based on network activities and could be directly computed from our network-flow for each internal host. All these features are computed within a time window $W$. To have enough resolution, the window length we are using is 10 minutes. For example, we compute the unique number of IPs contacted by any internal host within the window. We also consider the ratios of successful TCP connection, as well as ratios of blacklisted IPs/Domains contacted by each internal host. With all these behavior feature values’ distributions at different times, we profile each user with his/her own distributions. Any severe deviation from its behavior profile will be considered suspicious and under scrutiny. To abbreviate, we will refer to the vector of all these behavior feature values as $\vec{f}$ in the rest of the paper.

5.2.1 Attack-Source Score Charges Stems from IDS Alerts and Suspicious Behavior Changes

Given each user’s behavior profile, any large deviation from a user’s behavior profile could potentially be malicious. We explicitly quantify such large deviation, along
Table 5.1: Summary of Symbols for Behavior Features

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_{IP}$</td>
<td>Number of unique IP contacted</td>
</tr>
<tr>
<td>$en_{Port}$</td>
<td>Entropy of distribution of opened ports</td>
</tr>
<tr>
<td>$w_{TCP}$</td>
<td>Ratio of successful TCP connections / all TCP connections</td>
</tr>
<tr>
<td>$n_{Byt_{s/r}}$</td>
<td>Number of bytes sent(received)</td>
</tr>
<tr>
<td>$n_{Pkt_{s/r}}$</td>
<td>Number of packets sent(received)</td>
</tr>
<tr>
<td>$n_{AS}$</td>
<td>Number of AS systems contacted</td>
</tr>
<tr>
<td>$n_{Peer}$</td>
<td>Number of unique peer(IP-Port) pair contacted</td>
</tr>
<tr>
<td>$w_{BlkIP}$</td>
<td>Ratio of blacklisted IPs contacted/contacted IPs</td>
</tr>
<tr>
<td>$w_{BlkDN}$</td>
<td>Ratio of blacklisted Domain Names contacted/contacted Domain Names</td>
</tr>
</tbody>
</table>

with any raised network IDS alerts \cite{7}, into a score, which we call the ““attack-source-score” charge. Formally,

$$C(T_k, intIP_i) = \alpha Dist_{L2}(\vec{f}_{intIP_i}(T_k), \vec{f}_{T_k}) + (1 - \alpha)N_{T_k},$$

(5.1)

where the charge $C$ at the time window $T_k$ has two contribution sources. One is from the deviation of the current behavior feature values from the historical behavior feature values of the internal IP $intIP_i$, which is weighted by $\alpha = 0.5$. The other contributor is the normalized number of IDS alerts triggered by this $intIP_i$ in this time window. Both of the $Dist$ and $N_{T_k}$ are normalized into (0,1). The function $Dist_{L2}()$ computes the deviation of the current behavior compared with the internal hosts’ profile, given whether the window is during workday and/or daytime. To alleviate the effect of noise, the function $Dist_{L2}$ only produces non-zero charges when the current time period feature vector $\vec{f}_{T_k}$ is larger than the profile feature vector $\vec{f}$ by 3 times the standard deviation. Note that $\vec{f}$ must fall into one of the four predefined categories.

At the end of each time window, we compute the behavior attack-source score charge for each internal host. These charges are the source of raw negative charges placed on the attack-source scores of the external hosts. For each internal host, the charges are allocated to the external hosts it has interacted with within a window $W_{accuse}$, as subtractions to the current attack-source score. From the viewpoint of external hosts, each external host receives an attack-source score charge from every internal
host it has been communicating with. This is how we transfer the suspicious behavior changes of our internal hosts into attack-source score charges and then convert them to negative attack-source scores sent to the responsible external hosts.

5.2.2 Attack-Source Score Computation & Propagations

With the periodically generated “attack-source score charge” from the internal hosts, every external host’s attack-source score gets affected as we assume that it has to be, to a certain extent, responsible for the abnormal/suspicious behavior changes of the internal hosts it has talked with. For example, consider the scenario where an internal host clicked on a malicious link and got infected as a p2p bot. Then the infected host started to initiate many connections to discover more of its botnet peers \[26\]. In this scenario, all the previous websites that this internal machine has visited within some reasonable time window could be the infection source. Therefore we distribute the charges equally to all these external hosts. One may ask what happens if the infected host visited legitimate external entities such as \textit{google.com} after infection. This issue is solved by the way we alleviate the charge scores. More specifically, the charge scores are alleviated by dividing by a denominator which is the unique number of internal hosts visiting the external host. Since the benign ones tend to have more people talking to then and do not get infected and exhibited malicious behavior afterwards, external IPs for benign ones such as \textit{google.com} will have a larger size of set \(Q_j^{T_k}\) than the malicious link, where \(Q_j^{T_k}\) denotes the set of internal hosts communicating with external host \(j\) in time period \(T_k\). More specifically, compare two external websites \textit{google.com} and \textit{somemaliciousness.com}. Suppose one internal host gets infected and an charge \(C_{\text{accuse}}\) has been charged to both of these websites. However, it is highly likely that number of internal hosts which have visited \textit{google.com} is much bigger than the number of hosts which visited \textit{somemaliciousness.com}. In this way, the attack-source score charge placed to \textit{google.com} is alleviated by a larger denominator and thus \textit{google.com} receives a much
smaller charge while the number of internal hosts visiting somemaliciousness.com will be much smaller and the charge will be then more severe. Formally, the raw attack-source score of external host $j$ is computed as:

$$R^T_{jk} = -\beta \frac{1}{|Q^T_{jk}|} \sum_{intIP_k \in Q^T_{jk}} C(intIP_k, T_k) + \exp^{-(1-\beta)}R^T_{jk-1}, \quad (5.2)$$

the weight $\beta$ controls the weight between the previous attack-source-score and the charge at the current time period. Note that without any explicit charges from the internal hosts, the value of $R_j$ will decays towards 0(neutral) with the only contributor as its previous score. This is actually the most common case as malicious behavior changes and triggered IDS alerts are not very frequent.

However, given that some external IP’s attack-source scores get bad enough to an extent, we record this IP in the database, and explicitly change its decaying factor $\beta$ to be larger, which will induce two effects. Firstly, with a large $\beta$ value in Equation 5.2, the new attack-source score charge will be larger than before, given the same level of malicious behavior change. Secondly, the exponential decaying factor of its previous attack-source-score will decay much slower, with a smaller abstract value of $1 - \beta$. In this way, if anything malicious has been recorded towards an external IP, new charges will compound into more charges while its previous bad attack-source-score maintains.

Given these raw attack-source scores computed for each external host, we can propagate these score values from the potentially responsible external nodes to other external hosts. However, we have to be careful how we do this propagation as our observed cross-boundary traffic only constitutes a small part of the external hosts’ activities. Thus simple propagating with respect to “network proximity” of external hosts such as IP-proximity might be biased because of our limited viewpoint of the traffic. Instead, we compute the distance measure of two external hosts based on the similarity of their traffic patterns in communicating with our internals. Consider the graph $G = (V, E)$,
where $V$ represents all the external IPs and the weight $w$ on edge $E(i,j)$ is computed in the following way: let $I_i$ and $J_i$ denote the set of internal hosts that external IP $i$ and $j$ have contacted during this time window. The weight $w$ is the Jaccard similarity of these two sets, i.e., $w = \frac{|I_i \cap I_j|}{|I_i \cup I_j|}$. For each time period, charges on the attack-source scores are injected to the responsible external hosts and we propagate using the "dye-pump" algorithm as described in [26].

5.3 Experimental Results

In this section, we experimentally present two use cases of our proposed approach: discovering the infection source and generating predictive blacklisting for potentially malicious hosts.

5.3.1 Case 1: Finding Infection Source

To accurately find the infection source, we set up virtual machines in a subnet and mimicked routine user network activities by making the hosts randomly visit external-websites. In addition, we also made these hosts actually visit websites hosting malware. More specifically, the virtual machines downloaded and installed Config-C from www.offensivecomputing.com. After infection, we restrict any out-going connections from these infected machines to subnets within our network but allow their communication with the malware peers outside our network. In fact, a few minutes after the virtual machines get infected, the infected hosts attempted to establish connections with around 40K hosts outside our network perimeter. This syndrome is captured as malicious behavior change by our system’s internal behavior features and triggered the deterioration of the attack-source-scores of the external IPs that the virtual machines visited.

We monitored the attack-source-scores of the IPs for the external websites that these hosts visited. Fig5.1 shows the attack-source-score of the infection source. To compare,
Figure 5.1: Attack-source score change when infection happens. The X-axis is the time and Y-axis is the attack-source score.

we also traced the average attack-source-score of random hosts from either the entire external IP set (the blue line in Fig 5.1) and the external IPs visited by the infected internal hosts (the black line in Fig 5.1).

As shown in Fig 5.1, the infection source (www.offensivecomputing.com) decreases much more significantly than the scores of compared average random hosts. The point where the attack-source-score of the infection source decreases corresponds with the event where the malware was downloaded and installed. On the other side, the average attack-source-score from (100) random hosts that the infected internal IP talks to also gets worse. This is reasonable because when an internal host gets infected, all of the its visited external hosts are suspicious for the infection to a certain extent. However, the attack-source scores of these hosts get better and approach the attack-source-score of random neutral non-malicious hosts (blue line) as time goes and no further malicious behavior gets exhibited. Therefore from this case study, we demonstrated via experiments that our proposed external-hosts based attack-source scoring system could
discover the correct infection source.

5.3.2 Case 2: Predictivity of the Hosts with Worst Attack-Source Scores

Another use case for our external attack-source-score system is to infer the unknown risks of the external world. Given the external hosts with worst generated attack-source-score (i.e., the most negative attack-source-score), we aim to verify whether they are indeed responsible for some of our internal malicious behaviors as well as any unobserved malicious behaviors outside our network. Given access to limited machines, one way to verify the attack-source-score is to check whether these IPs are also listed by third-party blacklists such as Spamhause [6] or Barracuda [8]. Table 5.2 shows the top IPs with worst attack-source-score as we ran our system on one week of network data and then traced them for two months. As shown in the table, the top worst attack-source-score IPs all have third-party reported malicious activities. The verification mainly comes from two categories. First we check these IPs against all the public-blacklists. In addition, we google these IPs on a daily basis and check the result. Some IPs are blacklisted by public blacklists, while others are complained and discussed associated with malicious activities from the Google-returned pages.

To quantitatively measure how many of the worst attack-source-score IPs are eventually blacklisted by entities besides us, we checked the top 10, 100 and 200 IPs with public blacklisting resources described in the above section. As shown in Figure 5.2, all the top-10 IPs are finally justified to be malicious with corresponding records from third-parties, while for the top-100 the ratio is around 95% and the top-200 is around 49%. Note that it is not necessary that all the top-200 external IPs are necessarily malicious. However, the accuracy of our assigned attack-source score lies in the fact that, if we make a ranking of the worst attack-source-score IPs between top-10 to 100, the precision that these external hosts have been infectious and exhibited offensive behaviors is more than 95% accurate. More importantly, our attack-source-score is based on a
Table 5.2: Worst attack-source-score IPs generated by our system with IP, Domain and Third-Party Verifications

<table>
<thead>
<tr>
<th>IP</th>
<th>Domain Name</th>
<th>Blacklist</th>
</tr>
</thead>
<tbody>
<tr>
<td>207.66.0.10</td>
<td>offensivecomputing.com</td>
<td>Infection Source</td>
</tr>
<tr>
<td>99.194.104.94</td>
<td>dyn.centurytel.net</td>
<td>Reported by: ThreatExpert.com</td>
</tr>
<tr>
<td>212.235.111.224</td>
<td>netvision.net.il</td>
<td>Subnet Dictionary Attack</td>
</tr>
<tr>
<td>89.178.231.5</td>
<td>corbina.ru</td>
<td>CASA, NOMOREFUNN</td>
</tr>
<tr>
<td>93.80.68.54</td>
<td>corbina.ru</td>
<td>BARRACUDA RATS-Dyna Spamhaus</td>
</tr>
<tr>
<td>94.75.193.168</td>
<td>bignaturalsonly.com</td>
<td>BARRACUDA RATS-NoPtr</td>
</tr>
<tr>
<td>95.79.194.228</td>
<td>UNKNOWN</td>
<td>BARRACUDA RATS-Dyna Spamhaus, Tiopan UCEPROTECT</td>
</tr>
<tr>
<td>94.179.142.100</td>
<td>pool.ukrtel.net</td>
<td>Barracuda RATS-Dyna Sorbs, Spamhaus, Tiopan</td>
</tr>
<tr>
<td>85.141.164.20</td>
<td>mtu-net.ru</td>
<td>Spamhaus</td>
</tr>
<tr>
<td>91.124.220.170</td>
<td>ukrtel.net</td>
<td>Reported by: BotsScount.com</td>
</tr>
<tr>
<td>83.22.174.108</td>
<td>adsl.tpnet.pl</td>
<td>Barracuda RATS-Dyna NOMOREFUNNSORBS CASA, Spamhaus</td>
</tr>
</tbody>
</table>

Figure 5.2: Cumulative ratio for being blacklisted for the most worst attack-source-score ips generated by our approach. The red line is the cumulative ratio for the top-10 worst IPs being finally blacklisted by third-party blacklists, while the green line represents the ratio for the top-100 and the blue one for top-200.
localized view and we assign a significant attack-source-score to these IPs even before most of them get recorded and blacklisted by other third-party services. Thereby, our generated attack-source-score is not only accurate but predictive in the sense that it will blacklist malicious hosts before one can query them on public blacklists.
Chapter 6

Conclusions

In this thesis, we studied the problems of mining generic traffic patterns and topical communities from heterogeneous network data.

First, we studied techniques for building cost-effective decision trees to predict unknown content pattern from network traffic features. We proposed a new cost effective decision tree as well as techniques to build an accurate meta-classifier from these trees. We experimentally compared the accuracy and cost of our proposed approach to existing techniques. Our results show that the cost-effective decision tree-based meta-classifier outperforms the popular SVM approaches by a wide margin in prediction cost, while maintaining good prediction accuracy comparable to that of SVMs.

Second, we studied how to discover hidden implicitly formed network Peer-to-Peer and Web Semantic Topical communities from a unique institute/enterprise viewpoint. We studied the properties of a state-of-art clustering algorithm, Latent Dirichlet Allocation (LDA) and proposed to apply it in the settings of bipartite networking graph clustering as a general framework for discovering community structures in bipartite network traffic observation settings. Our experimental results on two scenarios, detecting P2P application communities and Web semantic topical communities, both show that the proposed Bayesian statistical approach could accurately discover the community structures. In addition, in the P2P application case, we are also able to discover
unknown new malicious P2P communities as we trace the members in certain suspicious communities against public spam blacklists.

Finally, we studies a network-behavior feature drive attack-source generating system to assign potential attack scores to external hosts. We demonstrated the use of our system in two use cases: attack source detection and predictive blacklisting.

**Open Questions and Possible Future Work** There are a number of interesting open problems related to this work, and we describe these open problems as potential directions of future work. Firstly, the main limitation of the CoCoST classifier is the asymmetry requirement in the data. If the instances are associated with features that are of the same computational cost, it is possible that we cannot construct classifiers of different prices and specialities. A further investigation of the measurement of the asymmetry between classification difficulty and cost will be interesting, since it will explore more properties on such dependencies. Also noting that each individual tree is only for binary decisions(such as when the decision is TXT), there is still space to cut the cost. We leave this as future work.

For the community detection general framework of applying LDA, given the Bayesian nature of our approach, our performance is expected to get better with more observed data. However, there is a limit to the length of the time-window on which we conduct our solution. As we used internal IP address to identify an internal user, the IP-Churn [92] problem places a limit on the time window we can employ while still making statistical sense. Once the user’s IP changes, the collected flow data is no longer consistent in the sense of indicating internal users. This could be partially solved given access to the DHCP log which records the allocation of IP addresses to physical devices. However, it is not common to assume that the DHCP log is always available. In fact, the DHCP log is more privacy sensitive than Netflow data. To overcome this limitation, possible approaches may leverage Hierarchical Dirichlet Process (HDP) [43], where the IP-Churn is also captured by the generative model. This could be a poten-
tially interesting directions for future work. For the web semantic topical community detection, our approach could also be leveraged as a tool to get privacy information from anonymous data sets. It is very interesting to see how much private information such statistical clustering based algorithms can mine from anonymized datasets.

Regarding the attack source detection, given the computation of attack-source-score, the infect source detection is limited to the fixed observation window within which all contacted external hosts are accused. While our window is long enough to capture most infection sources, exhibited maliciousness after extreme silence will make it hard to capture a malicious external host and will cause false-negatives. To resolve this issue, a dynamic and customized accuse window could be applied to each internal host depending on its traffic patterns. We leave this also as future work.
Bibliography


[69] Y. Sun M. Gupta, J. Gao and J. Han. Integrating community matching and outlier detection for mining evolutionary community outliers. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining* *KDD*, 2012.


[91] Y. Tian W. Lee J. McPherson X. Liu, Q. He and Jiawei Han. Event-based social networks: Linking the online and offline social worlds. In ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), 2012.


